



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 router, 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 software for the router system as clients:

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

The chapter contains the following sections:

- [Cisco XML Transport and CORBA IDL, page 12-115](#)
- [CORBA NameServer, page 12-118](#)
- [CORBA Notification Structure, page 12-119](#)
- [CORBA XML Agent Initiative, page 12-120](#)
- [CORBA XML Limitations, page 12-120](#)
- [XML Agent Errors, page 12-121](#)
- [TTY-Based Transports, page 12-121](#)

Cisco XML Transport and CORBA IDL

CORBA/IIOP is the 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 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 an 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 12-1](#) describes the “login()” implementation in the CORBA XML agent that uses AAA options as part of the Authentication API.

Table 12-1 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 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
```



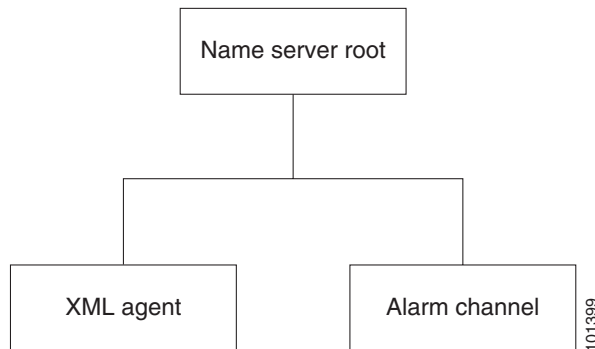
Note

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

LR Name Server Tree

The name hierarchy for the router is shown in [Figure 12-1](#). 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 12-1 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:

```

RP/0/RPO/CPU0:router# configure terminal
RP/0/RPO/CPU0:router(config)# xml agent corba ssl
RP/0/RPO/CPU0:router(config)# commit
RP/0/RPO/CPU0:router(config)# exit

```

CORBA XML Limitations

The system supports a minimum of 50 simultaneous sessions.

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 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 11, “Error Reporting in Cisco XML Responses.”](#)

[Table 12-2](#) describes the error codes specific to the XML agent for CORBA.

Table 12-2 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 |

TTY-Based Transports

These sections describe how to use the TTY-based transports:

- [Enabling the TTY XML Agent, page 12-121](#)
- [Enabling a Session from a Client, page 12-122](#)
- [Sending XML Requests and Receiving Responses, page 12-122](#)
- [Errors That Result in No XML Response Being Produced, page 12-122](#)
- [Ending a Session, page 12-122](#)

Enabling the TTY XML Agent

To enable the TTY agent on the router, which is ready to handle incoming XML sessions over Telnet and Secured Shell (SSH), enter the **xml agent tty** command, as shown in the following example:

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router(config)# xml agent tty
RP/0/RP0/CPU0:router(config)# commit
RP/0/RP0/CPU0:router(config)# exit
```

For more information about the **xml agent tty** command, see the *Cisco IOS XR System Management Configuration Guide*.

Enabling a Session from a Client

To enable a session from a remote client, invoke SSH or Telnet to establish a connection with the management port on the router. When prompted by the transport protocol, enter a valid username and password. After you have successfully logged on, enter **xml** at the router prompt to be in XML mode.

A maximum of 50 XML sessions can be started over TTY and SSH.



Note

You should use, if configured, either the management port or any of the external interfaces rather than a connection to the console or auxiliary port. The management port can have a significantly higher bandwidth and offer better performance.

Sending XML Requests and Receiving Responses

To send an XML request, write the request to the Telnet/SSH session. The session can be used interactively, for example, typing or pasting the XML at the XML> prompt from a window.



Note

The XML request must be followed by a new-line character; for example, press **Return**, before the request is processed.

Any responses, either synchronous or asynchronous, are also displayed in the session window. The end of a synchronous response is always represented with </Response> and asynchronous responses (for example), notifications, end with </Notification>.

The client application is single threaded in the context of one session and sends requests synchronously; for example, requests must not be sent until the response to the previous request is received.

Errors That Result in No XML Response Being Produced

If the XML infrastructure is unable to return an XML response, the TTY agent returns an error code and message in the following format:

```
ERROR: 0x%x %s\n
```

Ending a Session

If you are using a session interactively from a terminal window, you can close the window. If you want to manually exit the session, at the prompt:

1. Enter the **exit** command to end XML mode.
2. Enter the **exit** command to end the Telnet/SSH session.