Programmer’s Guide for Cisco Enhanced
Device Interface
Software Release 2.2

Americas 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: 408 527-0883

Text Part Number: OL-12570-01
CONTENTS

Preface ix
Audience ix
Conventions x
Obtaining Documentation, Obtaining Support, and Security Guidelines x

CHAPTER 1
Introduction 1-1
What Is a Programmatic Interface? 1-1
What Is Cisco E-DI PI? 1-1
Example Use Case 1-3
Using Cisco E-DI With CatOS Devices 1-4
Using a Filter With CatOS Devices 1-4
Programmers' UI Tools 1-4
Operational Data Model 1-4
Command Analyzer 1-4
Command Modeler 1-5
XML PI Java SDK 1-5

CHAPTER 2
XML Programmatic Interface 2-1
Establishing a Session 2-1
RPC Messages 2-3
<rpc> element 2-3
<rpc-reply> element 2-3
<rpc-error> element 2-4
<ok> element 2-4
RPC Message Pipelining 2-4
Configuration Datastores 2-4
Supported Capabilities 2-5
Writable-Running Capability 2-5
Candidate Configuration Capability 2-5
Validate Capability 2-5
Distinct Startup Capability 2-5
Limitations 2-5
Preface

This section explains the objectives, intended audience, and organization of this guide and describes the conventions that convey instructions and other information.

Cisco Enhanced Device Interface (Cisco E-DI) provides an XML (eXtensible Markup Language) Programmatic Interface (PI) based on NETCONF standards, and the supported data model is published through XSD (XML Schema Definitions) files.

Cisco E-DI also allows users to send CLI commands programatically as text enclosed in XML tags. The response to the CLI commands will be the output from the device in text format enclosed in the appropriate XML tags.

While CLI scripts and macro commands can provide some programmatic support for managing large networks, the approach can still be cumbersome and unsuitable for comprehensive management of large networks. Management applications handling multi-vendor devices expect a standards based programmatic interface.

Use this document to learn how to program the XML PI to configure devices.

For the latest and updated documentation of Cisco Enhanced Device Interface, 2.2, please check Cisco.com:

The following documentation is a part of Cisco E-DI 2.2:
- Release Notes for Cisco Enhanced Device Interface 2.2 on Windows - OL-12568-01
- Release Notes for Cisco Enhanced Device Interface 2.2 on Linux - OL-12569-01
- Installation and Setup Guide for Cisco Enhanced Device Interface 2.2 on Windows - OL-12565-01
- Installation and Setup Guide for Cisco Enhanced Device Interface 2.2 on Linux - OL-12566-01
- User Guide for Cisco Enhanced Device Interface 2.2 - OL-12567-01
- Programmer’s Guide for Cisco Enhanced Device Interface 2.2 - OL-12570-01

Audience

The primary audience for this guide includes system administrators and programmers. This guide assumes the reader is familiar with the following products and topics:
- XML
- XSD files
Conventions

This document uses the following conventions:

Table 1  Document Conventions

<table>
<thead>
<tr>
<th>Item</th>
<th>Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands and keywords</td>
<td>boldface font</td>
</tr>
<tr>
<td>Displayed session and system information</td>
<td>screen font</td>
</tr>
<tr>
<td>Information that the user must enter</td>
<td>boldface screen font</td>
</tr>
<tr>
<td>Variables that the user must supply</td>
<td>italic screen font</td>
</tr>
<tr>
<td>Menu items and button names</td>
<td>boldface font</td>
</tr>
<tr>
<td>Selecting a menu item</td>
<td>Option &gt; Network Preferences</td>
</tr>
</tbody>
</table>

Caution

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

Note

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

Timesaver

Means the described action saves time. You can save time by performing the action described in the paragraph.

Tip

Means the information will help you solve a problem. The tips information might not be troubleshooting or even an action, but could be useful information, similar to a Timesaver.

Obtaining Documentation, Obtaining Support, and Security Guidelines

For information on obtaining documentation, obtaining support, providing documentation feedback, security guidelines, and also recommended aliases and general Cisco documents, see the monthly What’s New in Cisco Product Documentation, which also lists all new and revised Cisco technical documentation, at:

Introduction

This section gives a brief overview of the features available in the Cisco Enhanced Device Interface (Cisco E-DI) XML Programmatic Interface (PI).

This chapter includes the following details:

- What Is a Programmatic Interface?
- What Is Cisco E-DI PI?
- Using Cisco E-DI With CatOS Devices

What Is a Programmatic Interface?

In a network management context, a PI is an interface for management applications to control and monitor networking devices. It is crafted specifically to meet the needs of network management applications. This is in contrast to a command line interface (CLI), which is intended and crafted for human operators.

What Is Cisco E-DI PI?

Cisco E-DI provides an XML PI based on the NETCONF configuration protocol (draft-ietf-netconf-prot-07.txt). Cisco E-DI acts as the NETCONF agent on behalf of a managed device. NETCONF specifies the protocol operations. Currently the device data model used in an operation is not specified by the protocol. Cisco E-DI defines the device data model based on the device command structure. The data models are described in XML schema definition (XSD) files. These data models are generated at development time, and are specific to a device type and OS combination. The XSD files are available for applications using XML PI.

Some users have management applications using the XML-based data model and script-based tools using CLI. In order to accommodate these users who want to continue making use of the CLI rather than the XML command model, the PI operations allow the transport of CLI commands and responses. See Chapter 2, “XML Programmatic Interface” for details about how to use the XML data mode, and Chapter 3, “CLI Mode” for details about how to use the CLI commands in NETCONF operations.
What Is Cisco E-DI PI?

Note
Refer to the NETCONF Configuration Protocol draft-ietf-netconf-prot-07 document which is included on the Cisco E-DI Documentation CD-ROM. This document defines the NETCONF configuration protocol which provides mechanisms to install, manipulate, and delete the configuration of network devices. It uses an Extensible Markup Language (XML) based data encoding for the configuration data as well as the protocol messages. The NETCONF protocol operations are realized on top of a simple Remote Procedure Call (RPC) layer.

The Cisco E-DI XML PI uses Telnet and SSH version 1 or version 2 for transport.
Cisco E-DI PI is used for the following device level tasks:

- Configuring the device (also available through the CLI)
- Executing EXEC commands on the device (also available through the CLI)

The Cisco E-DI XML PI supports:

- Sessions Establishment
- Session Release
- Protocol operations/services.
- RPC messages based communication model for message exchange
- RPC message pipelining
- Various data stores during configuration e.g. running, startup
- Capabilities such as writable-running capability, candidate-configuration capability
- Sub-tree filtering
- The following protocol operations/services:
  - <get-config>
  - <edit-config>
  - <copy-config>
  - <delete-config
  - <lock>
  - <unlock>
  - <get>
  - <close-session>
  - <kill-session>
  - <commit>
  - <discard-changes>

  Versioning of the data model

Note
Management operations and data models will be backward compatible. See Appendix A, "Level of Support for NETCONF Protocol Operations and Features".
Example Use Case

This simple use case details how to add a user to a running configuration. It identifies the protocol operations that are available, and explains how to use them. The remaining chapters of this guide will go into details of individual protocol operations and how to use them with E-DI XML PI.

Applications can use the NETCONF primitives to build more complex management scenarios.

1. The application establishes a NETCONF session with Cisco E-DI for the device to be managed—Cisco E-DI provides various ways of establishing a NETCONF session. See Appendix B, “NETCONF Client GUI” for more details.
2. Get the running configuration using a filter on the username—Applications use the standard <get-config> operation. In the filter, to express the command for the username, application refers to the device specific XSD. Alternatively, application can use CLI commands.
3. Make sure that the user does not already exist—This is done in the application's code.
4. Add a username to the candidate configuration—Application uses the <edit-config> operation with the candidate as the target data store.
5. Validate the candidate configuration—Application uses the <validate> operation.
6. Get a lock on the running configuration—Application uses the <lock> operation.
7. Commit the configuration change—Application uses the <commit> operation.
8. Check the running configuration with a filter on the username—Applications use the standard operation. In the filter, to express the command for the username, application refers to the device specific XSD. Alternatively, application can use CLI commands.
9. Make sure that the username is now returned—This check is done by the application in its own code.
10. Release the lock on the running configuration—Application uses the <unlock> operation.
11. Close the session—Application uses the <close-session> operation.

The Cisco E-DI XML PI includes the following supplementary material to help you understand and use the XML PI. These files are available on the Cisco E-DI Documentation CD-ROM:

- A simple GUI client application—Appendix B, “NETCONF Client GUI” explains how to use this GUI tool.
- The application is shipped as jar files.
- A set of JAVA API classes—These classes help the applications to connect to Cisco E-DI and conduct NETCONF sessions.
- A simple java program which illustrates the usage of API classes.
- A set of javadoc documentation for the API classes.
Using Cisco E-DI With CatOS Devices

Cisco Catalyst OS (CatOS) devices do not have a startup configuration. For consistency with Cisco IOS devices, Cisco E-DI allows the notion of a startup configuration for CatOS devices, and shows the running configuration as the startup configuration. Since both configurations are essentially the same, the following variations in the behavior of NETCONF operations occur:

- Locking the running configuration or startup configuration locks both running and startup configurations. Unlocking unlocks both running and startup configurations.
- Copying a running configuration into a startup configuration has no effect.

Using a Filter With CatOS Devices

In CLI mode, <get-config> and <get> permits filter specification using the <text-filter-spec> element:

- On Cisco IOS devices, this can take any values which are allowed on the device following the show running-config command.
- On CatOS devices, this can take any values allowed on the device following the show config command.

Note

For details of the Command Line Interface (CLI) mode provided by Cisco E-DI XML PI, see Chapter 3, “CLI Mode”.

Programmers’ UI Tools

The following UI tools are provided as part of the Cisco Enhanced Device Interface installation:

- Operational Data Model
- Command Analyzer
- Command Modeler

Operational Data Model

Provides applications with an XML interface that let you retrieve the operational data from Network Elements. This data is available through CLI show commands. It also allows you to create a Model spec file, an XML file, and an XSD.

For details see the section Using the Operational Data Modeler (ODM) in this guide.

Command Analyzer

Enables you to compare two IOS images. Also allows you to check commands for completeness.

For details see the section Using Command Analyzer in this guide.
Chapter 1  Introduction

**Command Modeler**

Provides an infrastructure to generate and validate device-independent CLI models. You can use these device-independent CLI models to generate device-specific Java code.

You can use this generated Java code in Domain Manager applications instead of the hard coded, template-based CLI generation mechanism. Command Modeler is built over the E-DI Device CLI Knowledge Base.

For details see the section Using Command Modeler in this guide.

**XML PI Java SDK**

Cisco E-DI provides Java APIs over XML PI. E-DI provides Java client API libraries that accept Java objects as inputs and return Java objects, so that you can use these APIs to develop your own applications. You can also use these APIs in existing applications.

For details see the section Java SDK for XML PI in this guide.
CHAPTER 2

XML Programmatic Interface

This chapter describes the XML programmatic interface (PI) provided by Cisco E-DI:

- Establishing a Session
- RPC Messages
- Configuration Datastores
- Supported Capabilities
- Limitations
- Subtree Filtering
- Protocol Operations/Services
- Session Release
- Data Model
- Versioning and Backwards Compatibility

Establishing a Session

Cisco E-DI acts as a NETCONF agent on behalf of the device it is managing. To establish a NETCONF session through Cisco E-DI, the manager, that is the client application, needs to identify which device it wants to manage through the NETCONF session. Cisco E-DI supports sessions over Telnet and SSH protocols.


The steps required to establish a session with Cisco E-DI are as follows:

1. Connect to Cisco E-DI—This is specified in the NETCONF/SSH specification.
2. Send and receive hello messages—This behavior is part of the NETCONF protocol.
3. Send a Cisco E-DI specific message, associate-devices—This is a Cisco E-DI specific requirement, as Cisco E-DI needs to be told which device to attach to the session.

The specified device must be managed by Cisco E-DI prior to establishing the XML session. Refer to Cisco Enhanced Device Interface User Guide, 2.2, Chapter 3 for details of how to add a device to Cisco E-DI using the CLI.

The associate-devices message requires the device IP address or name, and the device credentials.
A client can use the NETCONF sub-system to connect to Cisco E-DI as specified in the NETCONF/SSH specification.

Alternatively, a client can use the Cisco E-DI API classes to connect to Cisco E-DI through Telnet or SSH.

An example Java program, XMLClient.java, which shows how to use the API classes is included on the Cisco E-DI Documentation CD-ROM.

Establish a NETCONF session with Cisco E-DI as follows:

**Step 1** Establish a NETCONF session:

```bash
[user@client] ssh user@edi-server.example.org -s netconf
```

**Step 2** Send the `hello` message:

```xml
<?xml version="1.0"?>
<hello xmlns='urn:ietf:params:xml:ns:netconf:base:1.0'>
  <capabilities>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
  </capabilities>
</hello>
```

**Step 3** Receive the `hello` message, and extract the `session-id` as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<hello>
  <capabilities>
    <capability>
      urn:ietf:params:xml:ns:netconf:base:1.0
    </capability>
    
    <capability>
      urn:ietf:params:xml:ns:netconf:base:1.0#startup
    </capability>
    
    <session-id>4</session-id>
  </capabilities>
</hello>
```

**Step 4** Send `associate-devices` as follows:

```xml
<?xml version='1.0'?>
<rpc message-id='1' xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <associate-devices xmlns="http://www.cisco.com/edi_2.2/schema/netconf-extensions">
    <networks>
      <device>
        <ipaddress>192.168.1.2</ipaddress>
      </device>
    </networks>
    <credentials>
      <login>cisco</login>
      <password>cisco</password>
    </credentials>
  </associate-devices>
</rpc>
```

For more information about namespaces, see [Namespaces](#).
A NETCONF session is established and a client can conduct NETCONF operations over this session. To close the session, enter a close-session message.

### RPC Messages

The NETCONF protocol uses an RPC-based communication model. NETCONF peers use elements to provide application protocol-independent framing of NETCONF requests and responses. The application client is the Manager element and Cisco E-DI serves as the NETCONF Agent on behalf of the device.

**Note**

For more details, refer to the NETCONF Configuration Protocol draft-ietf-netconf-prot-07 document which is included on the Cisco E-DI Documentation CD-ROM.

#### <rpc> element

The <rpc> element is used to enclose a NETCONF request sent from the manager to the agent.

The <rpc> element has a mandatory attribute "message-id", which is an arbitrary string chosen by the sender of the RPC that will commonly encode a monotonically increasing integer. The receiver of the RPC does not decode or interpret this string, but simply saves it to use as a "message-id" attribute in any resulting <rpc-reply> message.

Example:

```xml
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <some-method>
    ...
  </some-method>
</rpc>
```

Cisco E-DI supports mandatory attribute <message-id> in the <rpc> element. Any other attributes specified in <rpc> will be returned in <rpc-reply> unmodified.

#### <rpc-reply> element

The <rpc-reply> message is sent in response to an <rpc> operation. Cisco E-DI supports mandatory attribute <message-id> in the <rpc> element. Any other attributes specified in <rpc> will be returned in <rpc-reply> unmodified.

Example:

```xml
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <some-content>
    ...
  </some-content>
</rpc-reply>
```
<rpc-error> element

The <rpc-error> element is sent in <rpc-reply> messages if an error occurs during the processing of an <rpc> request in Cisco E-DI or if the device returns an error.

Cisco E-DI will support following elements as specified in NETCONF
- <error-type>
- <error-tag>
- <error-severity>
- <error-app-tag>
- <error-message>
- <error-info>

Cisco E-DI will not support <error-path> element which identifies the error causing node using XPATH.

<ok> element

The <ok> element is sent in <rpc-reply> messages if no error occurred during the processing of an <rpc> request.

Example:
```xml
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

RPC Message Pipelining

Cisco E-DI will process <rpc> requests serially in the order they are received. The user is allowed to send multiple requests, and the responses will be sent in the order the requests are received.

Configuration Datastores

Cisco E-DI supports following types of configuration datastores.
- Running
- Startup
- Candidate configuration datastore—Used to store configuration data that can be manipulated without impacting the device's current configuration. It is stored locally on the Cisco E-DI server and can be manipulated using the NETCONF specified operations. There is one candidate configuration per session.

Any of these datastores can be used as a source or a destination in any protocol operation in which they are defined as valid by NETCONF. Additional constraints imposed by Cisco E-DI are described in Appendix A, "Level of Support for NETCONF Protocol Operations and Features."
Chapter 2  XML Programmatic Interface

Supported Capabilities

Note
For more details, refer to the NETCONF Configuration Protocol draft-ietf-netconf-prot-07 document which is included on the Cisco E-DI Documentation CD-ROM.

Supported Capabilities
Cisco E-DI supports the following capabilities:
- Writable-Running Capability
- Candidate Configuration Capability
- Validate Capability
- Distinct Startup Capability

Writable-Running Capability
The #writable-running capability indicates that the device supports writes directly to the <running> configuration datastore.

Candidate Configuration Capability
The candidate configuration capability, #candidate, indicates that the device supports a candidate configuration datastore which is used to hold configuration data that can be manipulated without impacting the device's current configuration.

Validate Capability
The validate capability consists of checking a candidate configuration for syntactical and semantic errors before applying the configuration to the device. Cisco E-DI supports only syntactic validation.

Distinct Startup Capability
Cisco E-DI supports startup configuration datastore if the device supports it.

Limitations
Following NETCONF defined capabilities are not supported:
- URL capability
- XPATH capability
Subtree Filtering

XML subtree filtering is a mechanism that allows an application to select particular XML subtrees to include in the <rpc-reply> for a <get> or <get-config> operation. A small set of filters for inclusion, simple content exact-match, and selection is defined in NETCONF.

Note
For more details, refer to the NETCONF Configuration Protocol draft-ietf-netconf-prot-07 document which is included on the Cisco E-DI Documentation CD-ROM.

Cisco E-DI will support the following aspects as defined in NETCONF:

- No filter—If no filter is specified in the <get> or <get-config>, the entire configuration will be returned.
- Empty filter—If empty filter is specified, the response will contain no content.
- Select entire subtree—The entire subtree under a specified node will be returned.
- Select multiple subtrees—Subtrees under specified nodes will be returned.

Cisco E-DI does not support XPATH based filters.

Protocol Operations/Services

These include:

- Configuration Operations
- Supported NETCONF Operations

Configuration Operations

This section includes examples where an application uses NETCONF defined protocol operations such as <get-config>, and <edit-config>.

Note
For more details, refer to the NETCONF Configuration Protocol draft-ietf-netconf-prot-07 document which is included on the Cisco E-DI Documentation CD-ROM.
In the following example, an application is trying to edit part of a configuration. The protocol operation is `<edit-config>`:

```xml
<rpc message-id="7" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <edit-config>
    <target>
      <running/>
    </target>
    <default-operation>merge</default-operation>
    <test-option>test-then-set</test-option>
    <error-option>ignore-error</error-option>
    <config>
      <!-- Add your configuration commands here -->
    </config>
  </edit-config>
</rpc>
```

Everything in this example is NETCONF defined, except for the actual device commands to send. This is the data model. The data model definition is done by the program which is implementing the protocol as whatever you put within `<config>` and `</config>` should be understood by the implementing protocol.

Cisco E-DI accepts two types of configuration data from the XML client:
1. XML configuration—Developed and published by Cisco E-DI.
2. Plain text configuration—The device’s CLI commands.

The application needs to tell Cisco E-DI which of these forms (XML or plain text) it intends to use in this protocol operation.

The non-NETCONF elements, `xml-config-data` and `cli-config-data-block`, are highlighted in the following examples.

The following example uses the XML form:

```xml
<rpc message-id="7" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <edit-config>
    <target>
      <running/>
    </target>
    <default-operation>merge</default-operation>
    <test-option>test-then-set</test-option>
    <error-option>ignore-error</error-option>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <!-- express your commands in XML form here -->
      </cpi:xml-config-data>
    </config>
  </edit-config>
</rpc>
```

The following example uses the plain text form:

```xml
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:cpi="http://www.cisco.com/cpi_10/schema">
  <edit-config>
    <target>
      <running/>
    </target>
    <default-operation>merge</default-operation>
    <test-option>test-then-set</test-option>
```
Chapter 2      XML Programmatic Interface

Protocol Operations/Services

The non-NETCONF elements used in the above examples, xml-config-data and cli-config-data-block, are defined in a Cisco Programmatic Interface (CPI) schema. See Appendix C, “CPI Schema” for more details.

If the application uses the XML format, then it needs to know the device data model to send the commands to in XML format. The associated elements are highlighted in the following example. For example:

In this example, the application is trying to add a User ciscotest10 to the device. The elements including UserName and Naming are the XML representation of the device commands. This XML structure is expressed as XSDs which are generated by Cisco E-DI and published. In this example, the generated XSDs are for the device type Cisco 1700 with Cisco OS release 12.2(15)T14 and IDU version 1.2.

Supported NETCONF Operations

This section includes the following information:

- <get-config>
- <get>
- <edit-config>
- <copy-config>
- <delete-config>
- <lock>
Chapter 2 XML Programmatic Interface

Protocol Operations/Services

- <unlock>
- <commit>
- <discard-changes>
- <validate>
- <transform>

Note

For the protocol defined XSD (netconf xsd), refer to the NETCONF Configuration Protocol draft-ietf-netconf-prot-07 document which is included on the Cisco E-DI Documentation CD-ROM.

<get-config>

<get-config> is used to retrieve all or part of the configuration of the device. A filter may be specified to retrieve only a part of the configuration.

Example:

```xml
```

Response to <get-config>:

```xml
```

```xml
</rpc-reply>
```
The <get> operation retrieves configuration and operational information. To retrieve the configurational information, the <get> operation is equivalent to a <get-config> with running as the source.

Example for configurational data:
   <nc:get>
   </nc:get>
</nc:rpc>

To retrieve the operational data specify the show command. In the example below, show vtp status is specified within these tags:
<oper-cmd-format-text> </oper-cmd-format-text>

Example for operational data:
   <get xmlns="http://www.cisco.com/edi_2.0/schema/netconf_extension">
      <oper-cmd-format-text>show vtp status</oper-cmd-format-text>
   </get>
</rpc>

The <edit-config> operation loads a specified configuration to a specified target configuration. The target configuration can be running, startup or candidate configuration datastore.

The following parameters are supported with specified exceptions:

- The <error-option> protocol allows the values:
  - stop-on-error
  - ignore-error
  - rollback-on-error.
  Cisco E-DI supports only ignore-error.

- The <default-operation> protocol allows the values:
  - merge
  - replace
  - none
Cisco E-DI supports only **merge**.

**<edit-config>** request with **merge** operation highlighted in the following example:

```xml
<?xml version="1.0"?>
  <edit-config>
    <target>
      <running/>
    </target>
    <default-operation>merge</default-operation>
    <test-option>test-then-set</test-option>
    <error-option>ignore-error</error-option>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <DeviceConfiguration xmlns="http://www.cisco.com/edi_2.2/Cat3550/12.1(14)EA1a/1.2" xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <username>
            <UserName>ciscotest1</UserName>
          </username>
        </DeviceConfiguration>
      </cpi:xml-config-data>
    </config>
  </edit-config>
</rpc>
```

**<edit-config>** request with **delete** operation highlighted in the following example:

```xml
<?xml version="1.0"?>
  <edit-config>
    <target>
      <running/>
    </target>
    <default-operation>merge</default-operation>
    <test-option>test-then-set</test-option>
    <error-option>ignore-error</error-option>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <DeviceConfiguration xmlns="http://www.cisco.com/edi_2.2/Cat3550/12.1(14)EA1a/1.2" xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <username>
            <UserName operation="delete">ciscotest1</UserName>
          </username>
        </DeviceConfiguration>
      </cpi:xml-config-data>
    </config>
  </edit-config>
</rpc>
```

**<copy-config>**

The **<copy-config>** operation creates or replaces an entire configuration datastore with the contents of another complete configuration datastore. If the target datastore exists, it is overwritten. Otherwise, a new one is created.

The source can be any running, startup or candidate configuration datastore.
Protocol Operations/Services

Chapter 2       XML Programmatic Interface

Example:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
    <source>
      <running/>
    </source>
    <target>
      <startup/>
    </target>
  </copy-config>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>

<delete-config>

The <delete-config> operation deletes a configuration datastore. The <running> configuration datastore
cannot be deleted. Example:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-config>
    <target>
      <startup/>
    </target>
  </delete-config>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>

<lock>

The lock operation allows the client to lock a local configuration datastore of a device in Cisco E-DI.
The datastore can be any running, startup or candidate configuration datastore. If a datastore is locked,
access to it is limited to read-only for any other clients. All locks will be cleared on a normal or abnormal
session termination.

Example: Successful locking of running configuration datastore.

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <target>
      <running/>
    </target>
  </lock>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/> <!-- lock succeeded -->
</rpc-reply>

If a datastore is locked by some other session, any lock operation will fail with the error-code
lock_denied.

Example: Unsuccessful locking of running config.

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">\n  <lock>
    <target>
      <running/>
    </target>
  </lock>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/> <!-- lock succeeded -->
</rpc-reply>
<unlock>

The unlock operation releases a configuration lock, previously obtained with the <lock> operation. A client is not permitted to unlock a configuration datastore that it did not lock. The administrator can use the <kill-session> operation to terminate a lock acquired by any user.

Example:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <target/>
  </unlock>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>

<commit>

When a candidate's configuration content is complete, the configuration data can be committed, publishing the dataset to the device, and requesting the device to conform to the behavior described in the new configuration. The <commit> operation is used to commit the candidate configuration as the device's new current configuration.

Example:

<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit/>
</rpc>

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
<discard-changes>

If the client decides that the candidate configuration should not be committed, the <discard-changes> operation can be used to revert the candidate configuration to the current committed configuration.

If the client decides that the candidate configuration should not be committed, the <discard-changes> operation can be used to discard the candidate configuration. Cisco E-DI then sets the candidate configuration to an empty store.

Example:

```xml
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <discard-changes/>
</rpc>
```

This operation discards any uncommitted changes.

<validate>

This protocol operation validates the contents of the specified configuration.

Cisco E-DI supports the validate operation on candidate, running and startup configurations. Currently, Cisco E-DI only does syntax checking.

Example:

```xml
  <validate>
    <source>
      <candidate/>
    </source>
  </validate>
</rpc>
```

<transform>

This protocol operation transforms the configuration or the operational data into XML format without connecting to the device. It takes the configurational or the operational data in the form of a string.

User needs to specify the device type and the OS version for which the transformation is required, exactly from the E-DI supported OS-Device types.

For transforming the configuration data, the user needs to give the IDU version while for transforming the operational data, the user needs to give the `show` command.

Example (configuration data):

```xml
  <transform-config xmlns="http://www.cisco.com/edi_2.0/schema/netconf_extension">
    <DeviceConfiguration xmlns="http://www.cisco.com/edi_2.0/Cat3550/12.1(14a)/1.2" xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <cli-data-block>
        <hostname cisco>
          interface fastethernet0/1
          description hello
        </hostname>
      </cli-data-block>
    </DeviceConfiguration>
  </transform-config>
</rpc>
```
Here, <DeviceConfiguration xmlns="http://www.cisco.com/edi_2.0/Cat3550/12.1(14)EA1a/1.2" has the device type information as Cat3550, OS version as 12.1(14)EA1a, and 1.2 as the IDU version.

Also, <cli-data-block> specifies the configuration data, in this case, it is a set of commands.

In case of operational data, you need to give the show command which will be specified as shown below:

<oper-cmd-format-text>show vtp status</oper-cmd-format-text>

This should appear after the DeviceConfiguration tag.

Cisco Extensions

The following new operation, or Cisco extension, is provided:

* <associate-devices>—Associates the client session with a specified device. All the subsequent operations in the session are applied on the associated device.

Session Release

This section includes the following information:

- <close-session>
- <kill-session>

<close-session>

The <close-session> operation terminates an active session gracefully. Any pending requests are allowed to complete. Upon termination, any active locks are released.

Example:

```xml
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <close-session/>
</rpc>
```

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>

<kill-session>

The <kill-session> is used by one session to kill another session. It can be used by the administrator to terminate any user sessions.

For example, a NETCONF session(A) can kill another NETCONF session(B) provided A has admin privileges or has FULL_CONTROL on SERVER. When a session is killed, any operations pending on it will be removed, and not executed by the Cisco E-DI server. Any locks held by the session will be released. However, if there is an operation currently being executed by Cisco E-DI, for example an edit-config, that operation cannot be stopped. Changes made to the candidate configuration will be removed. Changes to other config stores are always implicitly committed by Cisco E-DI.

Example:

```xml
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <kill-session/>
</rpc>
```

<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
Data Model

The data model will be exposed to the clients in the form of multiple XSD files. XSD representation of
the data model is generated for a specific device family and Cisco OS release. XSD is generated from
the CLI knowledge base which provides information on what CLIs are supported in a Cisco IOS release.
The Cisco IOS CLI reference manual lists supported CLI commands.
XSDs for each device family will be published on CCO in ZIP file format. Each set of XSDs will also
contain a README.txt file which describes the contents and lists known issues/limitations.
The Cisco E-DI data model closely resembles the CLI syntax as follows:
- XML tags associated with keywords are in most cases the keyword themselves.
- XML tags associated with a submode are suffixed with -Configuration.
- XML tags associated with parameters are named using a Camel naming convention. That is, the first
  letter of each word in the name is capitalized.

Validation

Cisco E-DI validates XML requests coming to clients against the NETCONF schema. Cisco E-DI
validation checks a candidate configuration for syntactical errors before applying the configuration to
the device.

Namespaces

Namespaces are used to specify to which device type and version a given XML/XSD belongs. The
namespaces are generally defined in the form http://www.cisco.com/edi/releases/<device_type>/<OS
versions>/<IDU number>. For example, http://www.cisco.com/edi_2.2/Cisco1700/12.2(15)T14/1.2.
In the Cisco E-DI 2.2 release, an enhanced data-model is introduced, identified by the namespace
fragment, edi_2.2. This is a minor change in the naming convention compared to previous releases. In
Cisco E-DI 2.0.1, this fragment is given as edi_20. In Cisco E-DI 2.2, the new format is edi_2.2 or
edi_2.0.
Applications have a choice whether to use the 2.2 model or the previous model (2.0). The infrastructure
to facilitate this is as follows:
1. By explicitly specifying the desired model as part of the namespace in the protocol operation.
   For example, in an edit-config request, to specify that the 2.0 model should be used, enter the
   namespace as follows:
   http://www.cisco.com/edi_2.0/Cisco1700/12.2(15)T14/1.2.
To specify the new 2.2 data model, enter the namespace as follows:

http://www.cisco.com/edi_2.2/Cisco1700/12.2(15)T14/1.2

2. When any namespace is not specified by the application, for example, consider this simple get operation:

```xml
<?xml version="1.0"?>
<rpc message-id="123" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <get/>
</rpc>
```

In this request there is no indication of which model to use, 2.0 or the new 2.2 model. Cisco E-DI then looks at the default model value (see item 3 below). The default value is 2.2, this is set during the Cisco E-DI installation.
3. An administrator can change the default data model set (2.2) to the previous model, 2.0, enter the following command in server config mode:

```
xml model-version 2.0
```

Cisco E-DI will then use the 2.0 data-model, where a specific namespace is not used in the context, for interpreting the xml configuration and to express the xml configuration.

**Note** The `xml model-version` is a global setting in Cisco E-DI.

4. Where a NETCONF session needs to specify a default model version, irrespective of any global setting (see item 3 above), a finer control can be defined during the hello message exchange by using a specific capability as follows:

```
<?xml version="1.0"?><hello xmlns='urn:ietf:params:xml:ns:netconf:base:1.0'>  <capabilities>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
    <capability>urn:ietf:params:xml:ns:cisco:edi:DEFAULT_EDI_MODEL=edi_2.0</capability>
  </capabilities>
</hello>
```

**Versioning and Backwards Compatibility**

When a user selects a device family, Cisco E-DI will try to find the exact match. If the exact match cannot be found, then it will find the closest but preceding version.

When a user selects multiple devices of different types and different Cisco IOS versions, Cisco E-DI automatically determines the least common denominator set of commands, and presents them to the user. This behavior is the same as Cisco IOS.

The data models are backward compatible.
CLI Mode

This chapter describes the Command Line Interface (CLI) mode provided by Cisco E-DI XML PI. Cisco E-DI allows the user to send CLI commands as text enclosed in XML tags. The response to the CLI commands will be the output from the network element in text format enclosed in the appropriate XML tags.

Cisco E-DI supports two ways of sending CLI commands:
- As content of the element `config-format-text-cmd`. This is `cli` format.
- As content of the element `config-format-text-block`. This is `cli-block` format.

This chapter includes the following details:
- Sending Edit Configuration Commands in cli Format
- Sending Edit Configuration Commands in cli-block Mode
- Sending EXEC Commands in cli-block Mode
- Getting a Configuration Using Filters in cli Format

Sending Edit Configuration Commands in cli Format

The following example details how to send edit configuration commands in cli format

Request:

```xml
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:cpi="http://www.cisco.com/cpi_10/schema">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <default-operation>merge</default-operation>
    <test-option>test-then-set</test-option>
    <error-option>ignore-error</error-option>
    <config>
      <cpi:cli-config-data>
        <cpi:cmd>no username testcliedit</cpi:cmd>
      </cpi:cli-config-data>
    </config>
  </edit-config>
</rpc>
```
Sending Edit Configuration Commands in cli-block Mode

The following example details sending edit configuration commands in cli-block format.

Request:

```xml
<rpc message-id="101" xmlns:cpi="http://www.cisco.com/cpi_10/schema"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/></target>
    <default-operation>merge</default-operation>
    <test-option>test-then-set</test-option>
    <error-option>ignore-error</error-option>
    <config>
      <cpi:cli-config-data-block>username testcliedit1       username
                   testcliedit2</cpi:cli-config-data-block>
    </config>
  </edit-config>
</rpc>
```

Response:

```xml
<rpc-reply message-id="101" xmlns:cpi="http://www.cisco.com/cpi_10/schema"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Sending EXEC Commands in cli-block Mode

The following example details sending EXEC commands in cli-block mode.

Note

The validate-syntax attribute is not supported.

Request:

```xml
<rpc message-id="101" xmlns:cpi="http://www.cisco.com/cpi_10/schema"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter>
      <cpi config-format-text-block>
        <text-filter-spec>| include interface</text-filter-spec>
      </cpi>
    </filter>
  </get>
</rpc>
```

Response:

```xml
<rpc-reply message-id="101" xmlns:cpi="http://www.cisco.com/cpi_10/schema"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```
Getting a Configuration Using Filters in cli Format

The following example details getting a configuration using filters in cli format

Request:

```xml
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:cpi="http://www.cisco.com/cpi_10/schema">
  <get>
    <filter>
      <cpi:config-format-text-cmd
```
<text-filter-spec> include interface</text-filter-spec>
</cpi:config-format-text-cmd>
<cpi:oper-data-format-text-block>
<show>interfaces</show>
<show>arp</show>
</cpi:oper-data-format-text-block>
</filter>
</get>
</rpc>

Response:
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:cpi="http://www.cisco.com/cpi_10/schema">
<data>
<cpi:cmd>interface Loopback0</cpi:cmd>
<cpi:cmd>interface Loopback5</cpi:cmd>
<cpi:cmd>interface Virtual-TokenRing1</cpi:cmd>
<cpi:cmd>interface Virtual-TokenRing1.1</cpi:cmd>
</cpi:cli-config-data>
</data>
</rpc-reply>
Using the Operational Data Modeler (ODM)

The Operational Data Modeler (ODM) enables you to create a unique Data Model for the operational data that is available through CLI show commands. Using this Model, you can further create an XML programmatic interface for this operational data.

Developers usually create and validate a device data model file manually. ODM will help developers save valuable time in model creation.

ODM provides an interface for creating a new Operational Data Modeler spec file from a CLI data file, for a particular show command. That is, the output of each show command is associated with an Operational Data Modeler spec file. This spec file represents the following two components of a given show command output:

- Spatial information to extract or parse data.
- Structural information to model the data.

The ODM spec file serves as the knowledge base for a given command output.

ODM also allows for the modification of existing ODM spec files.

ODM supports the following CLI complexities:

- Verbose textual data containing attribute value pairs. For example, `show version`
- Numerical (Statistical) data with clean attribute value pairs. For example, `show vtp counters`, `show snmp statistics`, etc.
- Complex nested data. For example, `show buffers`
- Single/Multiple Tabular data. For example, `show mac-address-table`, `show arp table`
- Hybrid Textual and tabular data. For example, `show file-system`
- Multiple instance data. For example, `show interface`

The topics in this chapter are:

- Getting Started With ODM
- Understanding the ODM Workflow
- Understanding the ODM UI
- Creating a Model
- Creating an XML File From the Model
- Creating an XSD for the Model
- Validating the XML File Using the XSD
- Creating and Using a JDP
• A Sample CLI
• A Sample Operational Data Modeler Spec File
• Tag Specifications

Getting Started With ODM

ODM is an Eclipse-based standalone UI tool. It is available as an executable that you can run independently.

ODM is packaged with Cisco E-DI, and will be available after you have installed Cisco E-DI.

To install Cisco E-DI, see:
• Installation Guide for Enhanced Device Interface, 2.2 on Windows
• Installation Guide for Enhanced Device Interface, 2.2 on Linux

ODM does not need the E-DI server to be running.

Launching ODM

Follow these steps to launch ODM:

Note

The E-DI server does not need to be running for you to be able to use ODM. It is a stand-alone application.

Step 1

Navigate to the directory  E-DI Install Location\Cisco Systems\Cisco EDI\edi\dist\ui_products\ODM
• On Windows: Double-click on launcher .exe.
• On Linux: Run ./launcher

ODM opens.

Alternatively, you can copy this file and extract it in any other location of your choice.

For details about the ODM UI, see Understanding the ODM UI.

Understanding the ODM Workflow

Step 1

Create a Model.

A Model is a hierarchy structure that will help you create your XML spec file. To do this, create a show command Container for your Model (see Creating a Model).

The Container appears as a top-level node in the Model Tree view (see Understanding the ODM UI).

Step 2

Open the required CLI data file for the show command.

This appears the Editor view (see Understanding the ODM UI).
Step 3 Within the top-level Container, proceed to create your hierarchical representation of the Model. Create the sub-Containers or Table tree nodes, and the Property or Table Header leaf nodes using the guidelines in the topic Specifying Elements in the Model.

Step 4 After you create your Model, validate your model by running it against the ODM engine using the Command > Run option (see Verifying the Model). The CLI from the CLI data file is displayed in the Console View in the XML format. This gives you a preview (or draft) of the XML file that you are about to generate. You cannot edit the output in the Console View. However, you can make changes to your Model Tree view and revalidate the Model.

Step 5 After you are satisfied with your draft Model, create the XML file using the Command > Generate > XML Spec menu option (see Creating an XML File From the Model). Use the Save option to save the XML spec file.

Step 6 You can then create an XSD for the Model using the XSD > Generate option (see Creating an XSD for the Model).

Step 7 Then you can validate your XML spec file against the corresponding XSD using the XSD > Validate > XML Output option (see Validating the XML File Using the XSD).

Step 8 To use the spec files through E-DI, you need to create a JDP from the spec files using Tools > Create JDP. To use this, see:

---

Understanding the ODM UI

To launch the ODM UI, see the topic Launching ODM.

The ODM User Interface has these main Views:

- Model Tree View
- Editor View
- Console View

Model Tree View

The Model Tree View displays a hierarchical representation of the elements of a Model. In this View, Containers and Tables are represented as tree nodes and the Properties and Legends are represented as the leaf nodes.

Editor View

The Editor enables you to open CLI data files. You can also open XML spec files and make modifications to them using the Editor. The Editor allows you to do cut, copy, paste etc.

Console View

The Console View is used to display the XML format of CLI Data. Once the Model is created, you can verify it by generating the XMLized CLI output which is displayed in the Console View.
Creating a Model

A Model is a hierarchical structure that will help you create your XML spec file. You will use the Model Tree view and the CLI Data file to create this.

To create a Model:

**Step 1** Select **Command > Add** from the ODM Main Menu to add a top-level Container to the Model Tree View. To know more about the Container, see the `<Container>` topic.

The Show Command Data dialog opens.

**Step 2** Enter the required details in the Show Command Data dialog.

The fields in the Show Command Data dialog are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Command</td>
<td>Enter a valid show command for which you want to create a Model. For example, <code>show atm map</code>.</td>
</tr>
</tbody>
</table>
| Alias Info    | You can use the Alias List field to specify all the commands that can be mapped to this Command or Model. For example, for IOS commands `show interfaces` and `show interfaces vlan`, the same spec file can be used, as their CLI data structures are similar. In such a case, you can specify both these commands in your Model using the Alias List field.  
1. Click the Add button and enter the aliased command in the Input dialog box that appears.  
2. Click **OK**.  
   The aliased command appears in the Alias field.  
   To remove an Alias, select the Alias from the Alias List and click **Remove**. |
| OS Type       | Select the OS Type.                                                          |
| XML Tag       | The field is auto-populated based on the show command or top-level Container that you specify in the Show Command field. For example, the XML tag for the show command `show atm map` would be `ShowAtm Map`. You can change this tag as required. |

**Step 3** Click **OK**.

The Container for the show command appears as a node in the Model Tree View.

**Step 4** Using **File > Open**, select the CLI data file for which you want to create a Model.

The selected file opens in the Editor area. The CLI data file that is displayed in the Editor is rendered as read-only by ODM. This is to prevent any accidental edits to the CLI data file when you are in the process of creating your Model.

**Step 5** You can add different elements or nodes in your Model or Model Tree View using the guidelines specified in the topic **Specifying Elements in the Model**.
As you specify each element, the element appears in the Model Tree View as a node. You can open more than one CLI data file in the Editor and use these files to add elements to your Model Tree.

Complete your Model by specifying elements as required.

**Caution**

While creating elements (container, property and table) ensure that the elements appear in the same order as they appear in the CLI Data file both from left to right and from top to bottom. This is mandatory for the Model to work properly.

You can save the Model with a .odm extension for later edits, if required. This will preserve your tree hierarchy as well as the XML information that is based on your Model.

To save your Model, select File > Save As.

See Figure 4-1: The Completed ODM Model Showing the Model Tree View, the ODM Model File in Editor, and CLI in XML Format in the Console

---

**Figure 4-1**  The Completed ODM Model Showing the Model Tree View, the ODM Model File in Editor, and CLI in XML Format in the Console

---

**Specifying Elements in the Model**

In order to create a Model, you should specify the elements for your Model. The elements that you specify, will translate to XML tags and properties in the XML spec file that you create. To know more about tags and attributes, see the topic Tag Specifications.
Adding a Container

You can add a Container to your Model. To know more about the Container, see the `<Container>` topic.

To add a Container:

**Note**
Ensure that your CLI data file is open in your Editor.

**Step 1**
In the CLI data file, drag your mouse and select the word or the set of words that you want to add as a Container in your Model.

**Step 2**
In the Model Tree View, select the node that you want to specify as the parent for your Container.

**Step 3**
Right-click on the selected node and select **Add > Container**.

The Container Data dialog appears. The Container name is pre-selected in the Name field. You can modify this if required.

**Step 4**
Enter the Alias for the Container. This is the tag name that will appear in your XML spec file. To know more about Alias, see the `alias` topic.

**Step 5**
Select the value for the Repeating field.
- Select True if Container model has more than one instance in the CLI data file.
- Select False if the Container model has just one instance in the CLI data file.

**Step 6**
Click OK.

The Container appears in your Model Tree View. See Figure 4-2A Top-level Container
Figure 4-2  A Top-level Container

Adding a Property

You can add a Property to your Model. To know more about the Property tag, see the &lt;Property&gt; topic.

To add a Property:

Note
Ensure that your CLI data file is open in your Editor.

---

Step 1  From the Model Tree View, select the parent Container.
Step 2  From the CLI data file, drag your mouse and select an attribute that you want to specify as a Property in your Model.
Step 3  Right-click and select Keyword from the Context Menu that appears. Keyword maps to the name of your Property in your Model.
Step 4  From the CLI data file, drag your mouse and select the string that you want to specify as the Value for the Property that you specified as the Keyword in Step 3.
Step 5  Right-click and select Value from the Context Menu that appears.

The property Data Dialog appears. You should specify the values for your Property here.
Chapter 4  Using the Operational Data Modeler (ODM)

Creating a Model

Step 6  Enter the values in the property Data Dialog. The fields in this dialog box are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Displays the attribute that you selected from the CLI data file as the Value. For details see the name topic.</td>
</tr>
<tr>
<td>Alias</td>
<td>Enter the Alias if required. For details, see alias.</td>
</tr>
<tr>
<td>Parent</td>
<td>ODM displays the parent Container name.</td>
</tr>
<tr>
<td>Distance</td>
<td>ODM calculates and displays the distance. For details, see distance.</td>
</tr>
<tr>
<td>Length</td>
<td>ODM calculates and displays the length. For details, see length.</td>
</tr>
<tr>
<td>End-Delimiter</td>
<td>Specify the end-delimiter. For details, see end-delimiter.</td>
</tr>
<tr>
<td>Legend</td>
<td>Select True for a legend if the Property describes the legend, otherwise select False. For more details, see the &lt;Legend/&gt; topic.</td>
</tr>
<tr>
<td>Type</td>
<td>ODM displays the appropriate type for the value. For example, String, Integer, IP Address, MAC Address, Port, Enum. etc. For details, see type. Here, if you select Enum, an additional field, Enum. Data appears. Enter the comma separated values for enumerated data in this field.</td>
</tr>
</tbody>
</table>

Step 7  Click OK.

The Property is added to the Parent Container in the Model Tree View. See Figure 4-3Adding a Property You can also add a Property from the Model Tree View by right-clicking on a parent container and selecting Add &gt; Property. The property Data Dialog appears. Step 6 However, you will need to add the name of the property. You will also need to specify the distance and length (although some default values are displayed here), and select the type of the property from the drop-down.
Figure 4-3  Adding a Property

Adding a Table

You can add a Table to your XML file. Also see Table-specific Attributes and Header-specific Attributes.

To add a Table:

Note Ensure that your CLI data file is open in your Editor.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the Container under which you want to add a table.</td>
</tr>
<tr>
<td>2</td>
<td>In the CLI data file, drag your mouse and select the word or the set of words that you want to add as a Table name in your Model.</td>
</tr>
<tr>
<td>3</td>
<td>Right-click and select Table &gt; New from the context-menu that appears.</td>
</tr>
</tbody>
</table>

The Table Data dialog opens.
The fields in the Table Data dialog are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the table.</td>
</tr>
<tr>
<td>Position Specification</td>
<td>Select True if the columns in your proposed table have either a null value or a multiple word value. Select False if the columns in your proposed table has a single word value.</td>
</tr>
</tbody>
</table>
Creating a Model

Step 4

Click OK.

The Table node appears in the Model Tree View.

Step 5

To add table column headers, in the CLI data, drag your mouse and select the text that you want as your header, up to the full width of the column. Right-click and select Table > Add Header from the context-menu.

You can add headers only if you had specified the Position Specification as True while specifying the Table properties in Step 3.

If your table has no header, then in the CLI data file, drag and select the longest entry for the value in a particular column right-click and Select Table > Add Header from the context-menu.

Right-click, and select Table > Table Header from the context-menu.

The table Header Data dialog appears.

The fields in the Header Data dialog are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header Known</td>
<td>Select True if your proposed table has a header. Select False if the proposed table has no header.</td>
</tr>
<tr>
<td>End Of Table</td>
<td>Optional. This is any line or text that should appear at the end of the table in the XML spec file. You can enter it here. Alternatively, you can select a table from the Model Tree View and then from the CLI data file, drag and select the Endof Table line. Right-click and select Table &gt; Endofthe Table in the context-menu that appears.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name attribute is used to give a name to your tabular data. If you had selected header text from the CLI data file, this appears here. If not, enter a header name. For more details, see the start topic.</td>
</tr>
<tr>
<td>Alias</td>
<td>The alias attribute is used for giving a name to the element which will be displayed as a tag in the XMLised output. If the alias attribute is not specified, the name attribute is used instead. For more details, see the start topic. Enter an alias for your table.</td>
</tr>
<tr>
<td>Start</td>
<td>ODM calculates and displays the Start position of a column, based on the your drag and select operation on the header or column, in the CLI data file. For more details, see the start topic.</td>
</tr>
<tr>
<td>End</td>
<td>ODM calculates and displays the End position of a column, based on the your drag and select operation on the header or column, in the CLI data file. For more details, see the end topic.</td>
</tr>
<tr>
<td>Nullable</td>
<td>Select True only if you know that the column can have a null value. Otherwise, select False. For more details, see the nullable topic.</td>
</tr>
</tbody>
</table>
Chapter 4  Using the Operational Data Modeler (ODM)

Creating a Model

Click OK.
See Figure 4-4 Adding a Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrappable</td>
<td>Select True only if you know the value in the column will wrap to the next line. Otherwise, select False. For more details, see the wrappable topic.</td>
</tr>
<tr>
<td>Legend</td>
<td>Select True for a legend column, if the table has a legend, otherwise, select False. For more details, see the &lt;Legend/&gt; topic.</td>
</tr>
<tr>
<td>Type</td>
<td>The type of the header is pre-selected. For example, String, Integer, IP Address, Enum, etc. For more details, see the type topic. Here, if you select Enum, an additional field, Enum Data appears. Enter the comma separated values for enumerated data in this field.</td>
</tr>
</tbody>
</table>

Adding a Legend

You can add a Legend to your Model. A Legend should be added to a top-level Container. See legend.

To add a Legend:

Note
Ensure that your CLI data file is open in your Editor.
Creating a Model

Step 1 In the CLI data file, drag your mouse and select the Legend that you want to add.

Step 2 Right-click and select Add Legend.

The Legend Data dialog appears. The Legend name is pre-selected in the Name field. You can modify this if required.

Step 3 Enter the Alias for the Legend. The Alias describes what the legend represents. For instance '+' may mean that the device is "Active".

Active will appear as a tag in your XML file.

Step 4 Click OK.

The Legend appears in the Model Tree View. See Figure 4-5Adding a Legend.

Alternatively, you can add a Legend to your Model Tree View, by right-clicking on the top-level Container and selecting Add > Legend.

Viewing and Editing the Model

You can view the Model either in the Model Tree view area or as an XML file in the Editor area.

The Model Tree View presents an hierarchical representation of the elements of the Model.

From within the Model Tree View, you can right-click and add a new node, edit the attributes of an existing node, or remove a node from the Model Tree.
You can also edit the attributes of existing node by right-clicking on it and selecting Edit. The Data Dialog box appears with your values. You can change these and click OK.

You can also remove a node from the Model Tree View by right-clicking on it and selecting Remove Node.

While creating elements (container, property and table) it is important to ensure that the elements appear in the same order as they appear in the CLI Data file both from left to right and from top to bottom. To change the order of the elements as required, use the Up and Down options from the Model Tree context menu. (To launch this context menu, right-click on a node.)

After you run your Model, the XML The Editor displays along with Syntax Highlighting (The content text in editing pane shall be color coded in different colors for users to easily view and differentiate the sections of spec file).

### Table 4-1  Color Codes Used in the XML Editor to Highlight Syntax

<table>
<thead>
<tr>
<th>XML Content Type</th>
<th>XML Content Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>Red</td>
<td>&lt;!-- Comment --&gt;</td>
</tr>
<tr>
<td>Tag</td>
<td>Blue</td>
<td>&lt;Target&gt;</td>
</tr>
<tr>
<td>Attribute's Value</td>
<td>Green</td>
<td>serverSrc</td>
</tr>
<tr>
<td>Tag Value</td>
<td>Black</td>
<td>&lt;VCI&gt;100&lt;/VCI&gt;</td>
</tr>
<tr>
<td>File Start</td>
<td>Grey</td>
<td>&lt;?xml version='1.0'&gt;</td>
</tr>
</tbody>
</table>

### Modifying the Model

You can modify an existing Model and validate it by providing the CLI data for that Model.

You can modify a newly-created Model or an existing Model from the Model Tree View or from the Editor View.

If you modify the Model using Model Tree viewer, the Editor is reflected with your changes immediately.

If you modify a Model using the Editor, the Model Tree View is refreshed with you latest hierarchy only after you do a View > Refresh operation on it from the ODM Main Menu. After you modify the Model as required you can validate it. See Verifying the Model.

To make your changes using the Model Tree View:

Select the node that you want to modify and right-click on it, select Edit from the context-menu that appears.

If you are in the process of finalizing your Model and want to save it for later use, you can save it from the Editor with a .odm extension. This is an interim work-in-progress spec file that you can treat as a draft, modify, and verify before creating the final output of this .odm file, which will be a .xml file.

To remove a node from the Model Tree View, select the node, right-click and select Remove Node from the context-menu that appears.

### Verifying the Model

After you create a Model, you can verify it by running the Model over the ODM engine. If there is an error you can edit the Model and revalidate it again.
Creating an XML File From the Model

To verify your Model:

**Step 1**  From the ODM Main Menu, select **Command > Run**. The Run dialog appears.
**Step 2**  Enter the location of the CLI data file, or use **Browse** to navigate to the file location.
**Step 3**  Click **OK**. ODM translates the CLI data into XML and displays it in the Console View. You can validate the XML output by checking for all the modeled elements and their required values. The Console View is read-only. To make your corrections, edit the Model Tree hierarchy or use the Editor. See **Modifying the Model**.

You can also verify the Model across different device types and OS Versions by loading the respective CLI data file each time you do the validation.

Creating an XML File From the Model

After you create and verify your Model (see **Verifying the Model**), you can create the XML spec file from the Model.

To create an XML file from a Model:

**Step 1**  From the ODM Main Menu, select **Command > Generate > XML Spec** (see **Creating an XML File From the Model**). The XML file is created and appears in the Editor.
**Step 2**  Save your XML file using the **File > Save** option from the ODM Main Menu. You should use the .xml extension to save your file.

Creating an XSD for the Model

After you have created a Model, you can create an XSD file for the selected Model.

This XSD file can be used to validate the XMLised output generated using the Model and the CLI data file.

To do this:

**Step 1**  From the Model Tree View, select the Model for which you want to create an XSD file. You can also open a previously created Model using the **File > Open** option. The Model file is displayed in the Editor.
**Step 2**  From the ODM Main Menu, select the **XSD > Generate** option. The generated XSD file will be displayed as an untitled file in the Editor. Save this file to a location of your choice using **File > Save** from the ODM Main Menu. You should save this file with a .xsd extension.
Validating the XML File Using the XSD

You can check whether the XML output that you created from the Model complies with the XSD.

To validate your XML output file:

---

**Step 1**
From the ODM Main Menu, select **XSD > Validate > XML Output**.

The XML File Validation dialog box opens.

**Step 2**
Enter the location of the XSD file against which you want to validate your XML file.

You can use **Browse** to locate the file.

**Step 3**
Enter the location of the XML file that you want to validate.

You can use **Browse** to locate the file.

**Step 4**
Click **OK**.

If the XML file complies with the XSD, a message appears in the Console view to confirm the same.

If the XML file does not comply with the XSD, the errors will be displayed in the Console view.

You can correct the XML file based on these, and revalidate the XML file.

Creating and Using a JDP

E-DI provides the support for a set of show commands for particular set of device types and versions.

If you want to support a particular set of show commands, then you can create spec files for those commands using the ODM UI. To use these spec files in E-DI, you should create a JDP.

**Prerequisites for Creating a JDP**
Move your spec files from the **SpecLocation** to a directory structure such as this:

- **SpecLocation/cisco/catos**, for CatOS commands.
- **SpecLocation/cisco/ios**, for IOS commands.

where **SpecLocation** is a directory that contains the spec files that you have created.

In addition, you should place two files in the **cisco** directory. These are:

- **ODMSpecInfo.xml**
- **ODMVendorSpec.xml**

**ODMSpecInfo.xml** has the following contents:

```xml
<?xml version='1.0'?>
<ODMSpecInfo>
  <Copyright>Copyright (C) 2006 Cisco Systems Inc. All rights reserved</Copyright>
  <Name>Cisco_ODM</Name>
  <PackageVersion>1.0</PackageVersion>
  <DeviceVendor>Cisco</DeviceVendor>
  <Description>ODM spec files for Cisco devices</Description>
  <EDIVersion>2.2</EDIVersion>
  <ODMEngineVersion>1.0</ODMEngineVersion>
</ODMSpecInfo>
```
ODMVendorSpec.xml has the following contents:
```
<?xml version="1.0" encoding="UTF-8"?>
<ODMSpecInfo>
  <Copyright>Copyright (C) 2006 Cisco Systems Inc. All rights reserved</Copyright>
  <Name>Cisco_ODM</Name>
  <PackageVersion>1.0</PackageVersion>
  <DeviceVendor>Cisco</DeviceVendor>
  <Description>ODM spec files for Cisco devices</Description>
  <EDIVersion>2.2</EDIVersion>
  <ODMEngineVersion>1.0</ODMEngineVersion>
  <Encrypted>false</Encrypted>
</ODMSpecInfo>
```

Ensure that the tag `<Encrypted/>` has value as `false`.

Alternatively, you can pick up these files from the following locations:
- **E-DI Install location**
  - `edi/resources/odm/specs/cisco`

To create a JDP:

**Step 1** Ensure that all the prerequisites are met. See Prerequisites for Creating a JDP

**Step 2** From the Main Menu, select **Tools > Create JDP**

The JDP Data dialog appears.

Enter the values in the JDP Data dialog:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODM Spec Directory</td>
<td>The location which contains all your spec files for the show commands that you want to add. See Prerequisites for Creating a JDP.</td>
</tr>
<tr>
<td>JDP Output Directory</td>
<td>The location where JDP will be created.</td>
</tr>
<tr>
<td>Schema File</td>
<td>The ODM schema file which will be used to validate the spec file. Select this file from this location: <strong>EDI Install Location</strong>/edi/resources/odm/schema/odmSchema.xsd</td>
</tr>
<tr>
<td>JDP Filename</td>
<td>The name of the JDP file.</td>
</tr>
</tbody>
</table>

**Step 3** Click **OK** to create the JDP.

**Step 4** After JDP is created, copy it to this location: **EDI Install Location**/edi/diagpackages/odm

**Step 5** To provide the device support information of the spec files, you should append the device support information in the following file:

**EDI Install Location**/resources/odm/mappings/cisco-devicefamily-odm.xml
Step 6  The following information should be appended to the end of the file in the following format, for each spec file in the JDP:

```xml
<ODMSpec resourcePath="sdm/cisco/ios/show-arp.odm">
  <SupportedDevices>
    <DeviceFamily name="Cat2950">
      <OSVersion>12.1(13)EA1c</OSVersion>
    </DeviceFamily>
    <DeviceFamily name="Cat3550">
      <OSVersion>12.1(14)EA1a</OSVersion>
      <OSVersion>12.1(22)EA2</OSVersion>
    </DeviceFamily>
  </SupportedDevices>
</ODMSpec>
```

Step 7  Restart the EDI service.

You can open a JDP file using WinZip.

### A Sample CLI

```
Version : running VTP2 (VTP3 capable)
Domain Name : name
Notifications: disabled                       Updater ID: 172.25.86.116

Feature      Mode       Revision
------------- --------  -----------
VLAN         Transparent 0

Pruning      : disabled
VLANs prune eligible: 2-1000
```
A Sample Operational Data Modeler Spec File

The following is a sample of an ODM spec file. This sample contains all the tags supported by ODM and the attributes.

```
<ODMSpec>
  <Command>
    <Name>show vtp domain</Name>
  </Command>
  <OS>catos</OS>
  <DataModel>
    <Container name="ShowVtpDomain">
      <Property name="Version" distance="2" length="2" type="String"/>
      <Property name="capable" alias="VersionCapability" distance="-1"/>
      <Property name="Notifications" type="enum">
        <Option value="enabled" alias="Enabled"/>
        <Option value="disabled" alias="Disabled"/>
      </Property>
      <Table name="VlanInfoTable">
        <Header name="Feature" type="String" start="0" end="14"/>
        <Header name="Mode" type="String" start="15" end="29"/>
        <Header name="Revision" type="Integer" start="30" end="-1"/>
      </Table>
      <Container name="Pruning" alias="PruningInfo">
        <Property name="Pruning" type="enum">
          <Option value="enabled" alias="Enabled"/>
          <Option value="disabled" alias="Disabled"/>
        </Property>
        <Property name="prune eligible" type="String"/>
      </Container>
    </Container>
  </DataModel>
</ODMSpec>
```

Tag Specifications

The following sections describe the ODM tags.

- `<Command>`
- `<Name>`
- `<Command>`
- `<Alias>`
- `<OS>`
- `<Container>`
- `<Property>`
- `<Table>`
- `<Header>`
- `<Option>`
- `<Legend>`
Chapter 4  Using the Operational Data Modeler (ODM)

Tag Specifications

**<Command>/<Name>**

The `<Command>/<Name>` tag is used to specify the name of the command.

**<Command>/<Alias>**

A single spec file can be used for multiple commands if their CLI data structure is the same.

For example, for IOS commands `show interfaces` and `show interfaces vlan` the same spec file can be used as their CLI data structure is similar.

These tags are used to describe all the commands for which this spec file can be used.

For the above example, tag would be:

```
<Command>
    <Name>show interfaces</Name>
    <AliasSet>
        <Alias>show interfaces value</Alias>
        <Alias>show interfaces Vlan value</Alias>
    </AliasSet>
</Command>
```

Therefore, using the `<Alias>` tag we can define any number of commands for which this spec file will be applicable.

**<OS>**

The `<OS>` tag specifies whether it is an IOS command or a CatOS command.

**<Container>**

The `<Container>` tag stands for Container. A Container is used for grouping related data. It contains properties, or table data in it.

There should always be at least one top-level Container. See A Sample Operational Data Modeler Spec File. In that sample spec, two Containers have been defined:

- A top level Container named `showVtpDomain`
- A lower-level Container named `Pruning` with the alias as `PruningInfo`. This contains information related to `Pruning`.

**<Property>**

The `<Property>` tag stands for the property or parameter for which a value is required.

See A Sample Operational Data Modeler Spec File. In that sample multiple parameters from CLI have been modeled as `Version`, `Domain name`, `Notifications`, etc.
Tag Specifications

<Table>

The <Table> tag is used to model tabular data present in the CLI data file. This tag is used to give a name to your tabular data. It contains a list of column headers, that are specified using the <Header> tag.

<Header>

The <Header> tag stands for table header. All the column headers specified within the <Table> tag are matched against each column of a table, defined in the CLI file. If all the column headers are present, only then the entries are further processed. See A Sample Operational Data Modeler Spec File. As shown in the sample, headers (named Feature, Mode, Revision) have been defined which map to corresponding table columns in CLI.

<EndOfTheTable>

The value of the <EndOfTheTable> tag specifies the line that is going to appear at the end of the table. It is an optional tag and can be specified if a static line needs to be present at the end of the table.

<Option>

The <Option> tag is used to specify enumerated values. One of the options defined, should be present in the CLI. See A Sample Operational Data Modeler Spec File. As shown in the sample, the Notifications parameter can have either enabled or disabled as the values. These possible values have been defined using the <Option> tag.

<Legend/>

The <Legend/> tag is used for defining legends and values for those legends.

Attribute Specifications

The ODM spec file has these types of attributes:

- General Attributes
- Container-specific Attributes
- Property-specific Attributes
- Table-specific Attributes
- Header-specific Attributes
General Attributes

The ODM general attributes are:

- **name**
- **alias**
- **type**
- **legend**

**name**

Except in the case of `<Table>` tag, in all tags the name attribute specifies the word which needs to be matched against the CLI data. Further processing is done for that element only if `name` is present. In `<Table>` tag, the `name` attribute is used to give a name to your tabular data.

**alias**

The `alias` attribute is used to give a a meaningful name to your XMLised data element. While the `name` attribute is used for matching a string against CLI data, the `alias` attribute is used for giving a name to the element which will be displayed as a tag in the XMLised output. If the `alias` attribute is not specified, the `name` attribute is used instead.

**type**

The `type` attribute indicates the type of value. It can be `String`, `Integer`, `Port`, `IpAddress`, `Enum` etc. If the type specified is `Enum`, then the possible enumerated values are specified using the `<option>` tag.

**legend**

The `legend` attribute can be specified in the `Property` and `Header` tags. Legend should be specified as `True` if the value for a given column or parameter is going to be legend. It makes use of a set of `legends` defined in the top-level Container to retrieve and display the appropriate value in the XMLised output.

Container-specific Attributes

The ODM Container-specific attribute is `dynamic`.

**dynamic**

The `dynamic` attribute in the Container tag indicates that the Container model has more than one instance in the CLI data file. All the tags modeled inside the Container will be repeated. For example, for the IOS command `show interfaces`, the CLI structure will be repeated for each Interface instance. In this case we can mark the entire model as `dynamic`.
Tag Specifications

Chapter 4 Using the Operational Data Modeler (ODM)

Property-specific Attributes

The ODM Property-specific attributes are:

- start
- start
- end-delimiter

distance

The distance attribute indicates the distance at which (or how far) the value of this Property is located. The default is 1. This attribute is used by the ODM Engine during runtime, to retrieve the value for the required Property.

length

The length attribute in the Property tag indicates the maximum number of words that can combine to form the value. The default is 1.

If the value is -1, it means that all the words till end of the line are combined to form a value.

This attribute is used by the ODM Engine during runtime.

end-delimiter

The value for the end-delimiter attribute in the Property tag specifies the ending delimiter of the required value. The end-delimiter takes precedence over length.

Table-specific Attributes

The ODM Table-specific attributes are:

- position-spec
- headerknown

position-spec

The position-spec should be specified as True if the start and end positions need to be specified for headers. You can specify the value as False if the following two conditions are true for the CLI table:

- Each column in CLI table will have a single word value.
- None of the columns can have null values.

By default, the position-spec is true. This means that ODM should calculate the start and end values for each Header.

headerknown

If the Table in your CLI data file does not have header names, then you should specify headerknown as False.
Header-specific Attributes

The ODM Header-specific attributes are:

- **start**
- **end**
- **nullable**
- **wrappable**

**start**

The start and end attributes in the header tag gives spatial information about a table column defined in the CLI data file. The start attributes indicate the starting point for the column data in a table.

**end**

The end attribute indicates end of the column data in a table. If end=-1, this means that ODM would need to extract the value till the end of the line.

**nullable**

The nullable attribute determines whether a column in your table can have a null value. If you specify the value for this attribute as True then the column can have a null value. If a column in your CLI data file can never have a null value, then specify this value as False.

**wrappable**

The wrappable attribute indicates whether the value in the column can wrap to the next line. If you specify this attribute as False, then the data in your XML-ised output will not be allowed to wrap to the next line.
Tag Specifications
CHAPTER 5

Using Command Modeler

Command Modeler provides an infrastructure for generating and validating device-independent CLI models. Developers can use the generated device independent CLI models to generate device-specific Java code.

The generated Java code can be used in Domain Manager applications as a replacement to the hard coded template-based CLI generation mechanism that is prevalent.

Command Modeler is a tool that is built over the EDI Device CLI Knowledge Base. The code generated by Command Modeler also takes care of device-specific capability and range variations. For example, the number of VLANs that can be created, the maximum number of MAC addresses that can be learned on a switch port when security is enabled, etc.

The device-independent virtual model is also called the Unified CLI View or Device Independent CLI Model.

The Unified CLI View is a unionized representation of all the CLIs that are available in a set of devices/IOS versions that the you select. Command Modeler support is only available for IOS in this release.

Command Modeler works on both Linux and Windows platforms.

To use Command Modeler, see these topics:

- Getting Started With Command Modeler
- Understanding the Command Modeler Workflow
- Understanding the Command Modeler UI
- Creating a Model and Generating Code
- Sample Model File

Getting Started With Command Modeler

Command Modeler is an Eclipse-based UI tool. It is packaged with Cisco E-DI, and will be available after you have installed Cisco E-DI. To install Cisco E-DI, see:

- Installation Guide for Enhanced Device Interface, 2.2 on Windows
- Installation Guide for Enhanced Device Interface, 2.2 on Linux
Getting Started With Command Modeler

To get started with Command Modeler, see these topics:

- Pre-requisites
- Launching Command Modeler
- Configuring Command Modeler

Pre-requisites

To work with Command Modeler you should have device packages available. If you have installed Cisco EDI, the device packages packaged with this release of Cisco EDI are available in this device package directory:

*EDI Install Location*/Cisco Systems/edi/dist/devpackages

To check for the device packages provided with this build, see the Cisco EDI FAQs in the User Guide for Cisco Enhanced Device Interface 2.2 or at:


Additional device packages or Incremental Device Updates (IDUs) /Service Packs will be made available through Cisco.com. IDUs allow Cisco EDI to be updated with support for new device packages. To download the files for the device packages:

2. Log in to Cisco.com using your user login and password.
3. Download the required device packages.
4. Copy these device packages into the *EDI Install Location*/Cisco Systems/edi/dist/devpackages directory.

See Cisco EDI Installation and Setup Guide 2.2, for details of the installation process.

Launching Command Modeler

Before you launch Command Modeler, from the command prompt, navigate to your Eclipse folder (where your eclipse.exe) resides. Run the command `eclipse -clean`. This ensures that the cache is cleaned.

Follow these steps to launch Command Modeler:

- **Step 1** Navigate to the folder *EDI Install Location*/Cisco EDI/edi/dist/cmdModeler/lib.
- **Step 2** Copy the Command Modeler jar file (com.cisco.EDI.cmdModeler_<buildnumber>.jar) from the lib folder.
- **Step 3** Paste the jar file into the Eclipse plugins folder (Eclipse Directory/plugins).
- **Note** Ensure that Eclipse is closed while the jar is copied into the plugins folder.
- **Step 4** Start Eclipse.
- **Step 5** From the Main Menu Bar, select Windows > Open Perspective > Other.
  The Open Perspective dialog box opens.
  Select Command Modeler and click OK.
Understanding the Command Modeler Workflow

The Command Modeler perspective opens.
For a description of the Command Modeler views, see Understanding the Command Modeler UI.

Your next task is to configure Command Modeler. See Configuring Command Modeler

Configuring Command Modeler

To be able to begin working with Command Modeler, you should configure Command Modeler for certain default settings such as the device packages directory and the output directory (where the generated code and the XML Model file will be stored).

To configure Command Modeler:

Step 1 From the Main Menu Bar, select Command Modeler > Setup Wizard.
The Command Modeler Setup Wizard opens.

Step 2 In the Device Packages Dir. field, enter the location of your device packages.
Or
Use Browse to select the directory in which your device packages reside. The default directory is EDI Install Location\Cisco Systems\Cisco EDI\dist\devpackages.

Step 3 In the Code Output Dir. field, enter the location where you want to save your device model output (generated code or XML file).
You can use Browse to select the directory in which your device model output (the Java code that you create) should reside. Alternatively, click Make New Folder, and create a new folder for your output.

Step 4 Click Finish.

Understanding the Command Modeler Workflow

The following is the high-level workflow of Command Modeler:

Step 1 Configure Command Modeler by specifying the device package directory and the output directory. See Configuring Command Modeler

Step 2 Specify a name for the model and select the devices and versions for which you need to create a Unified CLI View.
A Command Modeler Unified CLI View is created and all the sub-modes/views of the Unified CLI View are displayed in a tree format in the Commands Tree View. See Creating a Model and Generating Code.
You can view the details of each node using the Node Details View.

Step 3 Insert the commands into the Pre-Model by selecting a command and using the Add Menu option for the particular node, in the Tree View. See Creating a Model and Generating Code.
Step 4  Generate the Model (XML file) from the Pre-Model that you created.
This is a read-only file. However, you can select Save As and save it with an XML extension. You can
save this model under an existing Java project, and open it at any time to make any changes to the model.
See Creating a Model and Generating Code.
Step 5  Generate device-specific Java code using the Generate Code option from the Pre-Model view.
Command Modeler creates and saves the Java file to the output directory that you specified when you
configured Command Modeler.
To check for any information or errors during the Java code generation, see Troubleshooting Command
Modeler.

Understanding the Command Modeler UI

To launch the Command Modeler Perspective from the Eclipse Menu Bar, see Launching Command
Modeler.
The following are the component views of the Command Modeler Perspective:
- Commands Tree View
- Node Details View
- Pre-Model List View

Commands Tree View

The Tree View displays all the submode views from the Unified CLI view. You can drill down to see the
commands under each submode and nodes within each command.
One of the main submodes is the Configure submode.
The Interface command is not available under Configure submode.

Node Details View

When you select a node from the Tree View, the Node Details View shows the validity of the node under
the device-version pair.
The Node details are displayed for all the device-version pairs in the Unified CLI View that was created.
The following details are displayed for each node:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Example (for command <code>aggregate-address</code>) for Cat 3550 and Cat3750</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the node.</td>
<td>A.B.C.D</td>
</tr>
<tr>
<td>Type</td>
<td>The E-DI representation for the node. For example, key word node, number node, IP address node, etc.</td>
<td>IpAddressNode</td>
</tr>
</tbody>
</table>
Chapter 5      Using Command Modeler

Creating a Model and Generating Code

Creating a Model and Generating Code

Before you begin the task of creating a Model, ensure that you have configured Command Modeler. See Configuring Command Modeler.

You can create a device-independent virtual model. To do this:

Step 1  From the Main Menu Bar, select Command Modeler > Unified-View Creation Wizard.

The Unified-View Creation Wizard opens.

Step 2  From the Devices List, select the device-version pairs. You should select at least two device-version pairs to be able to create the Unified-View.

Step 3  Enter the Model name in the Model Name field. This can be any alphanumeric string of your choice.

Step 4  Click Finish.

After the selected device packages are loaded into the memory, the Commands Tree View on the left, is refreshed with the unified submodes. This may take a few seconds.
Step 5 Right-click on node from the Commands Tree View and select Add from the context-menu. The selected node is added to the Pre-Model List. For details about the Pre-Model List, see Understanding the Command Modeler UI. You can add more than one command to the Pre-Model List.

Step 6 Right-click in the Pre-Model List area and select Create Model. The Model, an XML representation of the selected commands is displayed in the Editor view. This XML file is not editable. However if you want to save this file and edit it later, select File > Save As and save the XML file with a .xml extension. You can review this file, or treat it as a draft, make changes to it or to the Pre-Model List until you have all the required commands, and then create the model again.

After all your changes are done you can use this XML file to generate code.

Step 7 Right-click in the Pre-Model List view and select Generate Code. If you had previously saved an XML file and have reopened it in the Editor to make changes, then you can right-click in the Editor and select Command Modeler > Recreate Model. It will rebuild the device package environment in which the model file was saved.

Add additional commands as needed. After completing the changes, you can right-click in the Editor and select Command Modeler > Generate Code. After the code is generated, the following message is displayed to show the success of the code generation:

File was saved in the Output Directory: directory path

Where directory path is the path to the output directory that you specified at the time of configuring E-DI.

Command Modeler creates Java interface and implementation files in the output directory. It creates one Java file for each of the device versions that you selected, and one interface file. The implementation files follow the naming convention of devicename_OSversion_modelname

For example, if you had selected Cat3550-12.1(14)EA1a, Cat3550-12.1(22)EA2 and Cat3750-12.1(19)EA1a, you will see the following Java files in your output folder:

- Cat3750_12_1_19_EA1a_ModelName.java
- Cat3550_12_1_22_EA2_ModelName.java
- Cat3550_12_1_14_EA1a_ModelName.java

and

- I_ModelName.java

where ModelName is the name of the model that you specified, and the prefix I indicates an interface file.

In the device-specific implementation files, set and check methods are implemented. toCli method returns a string which represents the commands for the model. negCli method returns a string which represents negation commands for the model.

The check method checks for the validity for a specific node.
Troubleshooting Command Modeler

You can view the information, errors, or warnings that may arise during XML file generation using the Eclipse Error Log.

To invoke the Eclipse Error Log:

**Step 1**
From the Main Menu Bar, select **Window > Show View > Other**.
The Show View dialog box appears.

**Step 2**
Select **PDE Runtime > Error Log**.
The Error Log appears.

The Error Log contains information, error and warning messages from all components. it is displayed under the columns Message, Plug-in, and Date:
- The Message column displays the message.
- The Plug-in column indicates the specific component that has generated the messages.
- The Date column displays the date when the message was generated.

For troubleshooting purposes, you can export the log using the Export Log option (icon) of Eclipse, and send it to customer support.

Launching the Command Modeler Help

To launch the help:

**Step 1**
Navigate to **E-DI Install Location\Cisco EDI\edi\dist\cmdModeler\lib**.

**Step 2**
Copy the jar file **com.cisco.edi.cmdModeler.doc.help.jar**.

**Step 3**
Paste the jar file into the Eclipse plugins folder (**Eclipse Directory\plugins**).

**Note**
Ensure that Eclipse is closed while the jar is copied into the plugins folder.

**Step 4**
Start Eclipse.

**Step 5**
From the Eclipse Menu Bar, select **Help > Help Contents**.
The help opens.

Sample Model File

The following is a sample Model file of Command Modeler:

In this sample, the format enclosed within the tags `<command>` and `</command>` has the Command ID and the View Name. If the View is not valid in a particular device it will not be added into the persisted file.
Sample XML Output File

The following is a sample XML output file of Command Modeler:

```
<model>
  <command>
    <id><![CDATA[router rip]]></id>
    <viewName>Cat2950-configure</viewName>
    <viewName>Cat3550-configure</viewName>
  </command>
  <command>
    <id><![CDATA[network 0.0.0.0]]></id>
    <viewName>Cat3550-configure-rip</viewName>
  </command>
  <command>
    <id><![CDATA[exit]]></id>
    <viewName>Cat3550-configure-rip</viewName>
  </command>
</model>
```

```
<model>
  <command>
    <id><![CDATA[router bgp 10]]></id>
    <viewName>Cat2950-configure#12.1(13)E1aC</viewName>
    <viewName>Cat3550-configure#12.1(14)E1aC</viewName>
  </command>
  <command>
    <id><![CDATA[network 0.0.0.0]]></id>
    <viewName>Cat3550-configure-bgp#12.1(14)E1aC</viewName>
  </command>
  <command>
    <id><![CDATA[exit]]></id>
    <viewName>Cat3550-configure-bgp#12.1(14)E1aC</viewName>
  </command>
</model>
```
Using Command Analyzer

Command Analyzer is a tool that allows you to:
- Check commands for completeness.
- Compare two IOS images.

Command Analyzer works on both Linux and Windows platforms.

To use Command Analyzer, see these topics:
- Getting Started With Command Analyzer
- Understanding the Command Analyzer UI
- Generating the Commands Report
- Generating the Command Comparison Report
- Troubleshooting Command Analyzer

Getting Started With Command Analyzer

Command Analyzer is a standalone UI tool. It is packaged with Cisco E-DI, and will be available after you have installed Cisco E-DI. To install Cisco E-DI, see:
- Installation Guide and Setup Guide for Enhanced Device Interface, 2.2 on Windows
- Installation Guide and Setup Guide for Enhanced Device Interface, 2.2 on Linux

To get started with Command Analyzer, see these topics:
- Pre-requisites
- Launching Command Analyzer
- Configuring Command Analyzer
- Understanding the Command Analyzer Terminology

Pre-requisites

To work with Command Analyzer you should have device packages available. If you have installed Cisco E-DI, the device packages packaged with this release of Cisco E-DI are available in this device package directory:

E-DI Install Location\edi\dist\devpackages
Getting Started With Command Analyzer

For the device packages provided with this build, see the E-DI 2.2 Release Notes.

Additional device packages or Incremental Device Updates (IDUs)/Service Packs will be made available through Cisco.com. IDUs allow Cisco E-DI to be updated with support for new device packages. To download the files for the device packages:

2. Log in to Cisco.com using your user login and password.
3. Download the required device packages.
4. Copy these device packages into the E-DI Install Location\edi\dist\devpackages directory.

See the Cisco E-DI Installation and Setup Guide 2.2, for details of the installation process.

Launching Command Analyzer

Follow these steps to launch Command Analyzer:

Note

The E-DI server does not need to be running for you to be able to use Command Analyzer. It is a stand-alone application.

Step 1

Navigate to the directory E-DI Install Location\Cisco Systems\Cisco EDI\edi\dist\ui_products\cmdAnalyzer

• On Windows: Double-click on launcher.exe.
• On Linux: Run ./launcher

Command Analyzer opens.

Your next task is to configure Command Analyzer. See Configuring Command Analyzer.

Configuring Command Analyzer

To be able to begin working with Command Analyzer, you should configure Command Analyzer for the location of the Device Packages.

To configure Command Analyzer:

Step 1

From the Main Menu Bar, select Command Analyzer > Wizards > Setup Wizard.

The Command Analyzer Setup Wizard opens.

Step 2

In the Device Packages Dir. field, enter the location of your device packages

Or

Use c Browse to select the directory in which your device packages reside. The default directory is E-DI Install Location\Cisco Systems\Cisco EDI\edi\dist\devpackages.

Step 3

Click Finish.
Understanding the Command Analyzer Terminology

In this chapter, you will frequently encounter the term device-version pair. A device-version pair in Command Analyzer terminology comprises a single device with any one of its image versions.

For example, Cat4000-12.1(19)EW1.

When two device-version pairs are referred to, it means either:

- Two devices each with a single IOS image version selected. For example, Cat2950-12.1(13)EA1c and Cat3550-12.1(14)EA1a constitute two valid device version pairs that can be compared.

or

- A single device with two IOS image versions selected. For example Cat3550-12.1(14)EA1a and Cat3550-12.1(22)EA2 constitute two valid device-version pairs that can be compared.

Understanding the Command Analyzer UI

To launch the Command Analyzer, see Launching Command Analyzer.

Command Analyzer has two main perspectives

- **Commands Report Perspective**
- **Compare Commands Perspective**

By default, when you launch Command Analyzer the Commands Report perspective opens. You can toggle or switch between these two perspectives in these ways:

- Go to File > Switch Perspective and select the other perspective.

or

- Click on the Open Perspective icon and select the other perspective.

Commands Report Perspective

The Commands Report Perspective enables you to select a device-version pair and generate a Tree view of the commands associated with that device-version pair.

You can drill down to specific commands, from a submode level and view the report of the commands in the Commands Report View. This Report displays whether the listed commands are complete commands and whether they have a valid negation command.

The Commands Report Perspective has the following views:

- **Commands Tree View**—Allows you to view the submodes for a selected device-version pair. Allows you to drill down several levels to the lowest level of the command.
- **Commands Report View**—Displays whether a command has callback, and whether the negation callback is applicable.
- **Error Log View**—Displays information and error messages. See Troubleshooting Command Analyzer.

To generate the Commands Report, see Generating the Commands Report.
Compare Commands Perspective

The Compare Commands Perspective enables you to select two device-version pairs and generate a diff of commands for the submodes associated with the device-version pairs.

The Compare Commands Perspective has the following views:

- **Diff Commands Tree**—Displays the submodes for the two device-version pairs that you selected.
- **Command Comparison Report**—Displays the Diff or differences between the commands present within the submodes for the two selected device-version pairs.
- **Error Log View**—Displays information and error messages. See Troubleshooting Command Analyzer.

To generate the Command Comparison Report, see Generating the Command Comparison Report.

Generating the Commands Report

To know more about the Commands Report see Commands Report Perspective.

The Compare Commands Report can be exported to a PDF.

To generate the Compare Commands Report:

**Step 1**
From the Main Menu Bar, select Wizards > Commands Report Wizard.

The Commands Report Generation Wizard opens.

**Step 2**
From the Devices List, highlight a single device-version pair to select it.

To know more about device-version pair, see Understanding the Command Analyzer Terminology.

**Step 3**
Click Finish.

**Step 4**
In the Commands Tree, drill down to the required submode or command node.

The command details applicable to the selected command appear in the Commands Report view, along with this information:

- Whether the command has callback.
- Whether the negation callback is applicable for that command.

**Step 5**
Export the report to PDF by selecting Tools > Export to PDF.

The Save As dialog box opens. Save the PDF to a location of your choice.

Generating the Command Comparison Report

To know more about the Command Comparison Report, see Compare Commands Perspective.

The Command Comparison Report can be exported to PDF.
To generate the Compare Commands Report:

**Step 1**  From the Main Menu Bar, select Wizards > Commands Comparison Wizard.

The Commands Comparison Wizard opens.

**Step 2**  From the Devices List, select two device-version pairs.

To know more about device-version pair, see Understanding the Command Analyzer Terminology.

**Step 3**  Click Finish.

**Step 4**  The Diff Commands Tree is populated. Expand the Root folder to see the submodes for the selected device-version pairs.

**Step 5**  From the Diff Command Tree, select a command submode.

The Command Comparison Report appears with the image information in two columns for the selected device-version pairs.

**Legends**

- Red indicates that the command was deleted from the first image.
- Green indicates that the command was newly added in the second image.
- Blue indicates that the command was changed.
  - The change could be a either a negation callback added or deleted denoted by the prefix [no] or
  - a callback was added or deleted, denoted by a suffix [cr].

**Step 6**  Export the report to PDF by selecting Tools > Export to PDF

The Save As dialog box opens. Save the PDF to a location of your choice.

---

**Troubleshooting Command Analyzer**

You can view the information, errors, or warnings that may arise when you generate the Command Report or compare two images using the Error Log view.

The Error Log is always open, irrespective of whether you are in the Commands Report perspective or in the Compare Commands perspective. It shows you information about successful completion of a task, and errors or warning messages.

For troubleshooting purposes, you can export the log using the Export Log icon in the Error Log toolbar and send it to customer support.
Java SDK for XML PI

E-DI provides Java client API libraries which take Java objects as inputs and return Java objects, so that customers can use these APIs to develop their own applications. They can also use these APIs in their existing applications.

E-DI provides these NetConfSession APIs:

- Connecting to E-DI
- Getting managed devices
- Getting the basic inventory from managed devices
- Getting the configuration of the managed devices
- Configuring the managed devices
- Error/Exception handling classes

All APIs allow per device operations as per the NETCONF protocol.
The APIs also provide object models for the get and set methods (that is, for NETCONF schemas and notifications).

Though connection, set and get methods take Java objects as inputs, they are internally converted to NETCONF complaint XML objects which communicate with devices.

In the current release, Java SDK provides only SSH1 connection APIs.
All classes meet draft-ietf-netconf-prot-07.txt specifications.

Note

After opening a session with E-DI using XML session, you should ensure that a Hello request is sent. This should be done before associate-device request is sent.

Accessing the Java APIs

E-DI provides the following sets of NETCONF APIs:

- E-DI Server Management APIs
- Device Management APIs

After you install E-DI, the APIs can be accessed from E-DI Install Location/edi/dist/client/lib.
E-DI Server Management APIs

E-DI provides the following Server Management APIs:

- `com.cisco.edi.ncclient.api.NetconfCreateCredentialSet`
- `com.cisco.edi.ncclient.api.NetconfManageDevice`
- `com.cisco.edi.ncclient.api.NetconfGetDevices`
- `com.cisco.edi.ncclient.api.NetconfTransform`

`com.cisco.edi.ncclient.api.NetconfCreateCredentialSet`

`com.cisco.edi.ncclient.api.NetconfCreateCredentialSet` is used to create a credential set, which can be used to manage devices in E-DI. At present, only Telnet transport type is supported for credential set creation.

The following is the constructor. If the `credentialSetName` is null (empty string) it throws an exception:

```java
public NetconfCreateCredentialSet (String credentialSetName, String login, String password, String enable, String readCom, String writeCom, String transportType) throws Exception
```

Here,
- `credentialSetName` is the credential set to be created.
- `login` is the login ID.
- `password` is the password.
- `enable` is the enable password.
- `readCom` is the SNMP read community
- `writeCom` is the SNMP write community
- `transportType` is the transport type (only Telnet is supported.)

```java
public String toNetconfXMLRequestString(String messageId)
```

returns a NetConf request for creating the credential set.

`com.cisco.edi.ncclient.api.NetconfManageDevice`

`com.cisco.edi.ncclient.api.NetconfManageDevice` is used to manage a device in E-DI.

The following is the constructor. If any of the fields are null (empty string) it throws an exception.

```java
public NetconfManageDevice (String ip, String credentialSetName) throws Exception
```

Here,
- `ip` represents the IP address of the device that needs to be managed.
- `credentialSetName` represents the credential set for managing the device.

```java
public String toNetconfXMLRequestString(String messageId)
```

returns a NetConf request for managing a device.
**com.cisco.edi.ncclient.api.NetconfDevices**

com.cisco.edi.ncclient.api.NetconfDevices is used to get the information about the online and offline devices.

The following is the default constructor:

```java
public NetconfGetDevices()
```

returns NetConf request for getting information about online and offline devices. If online = true, this function returns request for online devices. If online = false, this function returns a request for offline devices.

**com.cisco.edi.ncclient.api.NetconfTransform**

com.cisco.edi.ncclient.api.NetconfTransform is used to transform the configuration or the operational data into XML format without connecting to the device. It takes the configurational or the operational data in the form of a string.

The following is the constructor. If any of the fields are null (empty string) it throws an exception.

```java
public NetconfTransform(String devicetype, String osversion, String clidata) throws Exception
```

User needs to specify device type, OS version exactly from the E-DI supported OS-Device types, in the available constructors, along with the configurational or operational data.

To transform configurational data, user needs to set the IDU version as well. The following returns the XML request for the configurational data:

```java
public String getNetconfC2XRequest(String messageId)
```

To transform operational data, user needs to set the show command as well. The following returns the XML request for the operational data:

```java
public String getNetconfShowCmdRequest(String messageId, String Command)
```

For message ID information, see Note.

**Device Management APIs**

E-DI provides the following Device Management APIs:

- `com.cisco.edi.ncclient.api.NetconfHello`
- `com.cisco.edi.ncclient.api.NetconfGet`
- `com.cisco.edi.ncclient.api.NetconfGetConfig`
- `com.cisco.edi.ncclient.api.NetconfCopyConfig`
- `com.cisco.edi.ncclient.api.NetconfValidate`
- `com.cisco.edi.ncclient.api.NetconfLock`
- `com.cisco.edi.ncclient.api.NetconfCommit`
- `com.cisco.edi.ncclient.api.NetconfDiscardChanges`
Chapter 7  Java SDK for XML PI

Accessing the Java APIs

- com.cisco.edi.ncclient.api.NetconfUnlock
- com.cisco.edi.ncclient.api.NetconfCloseSession
- com.cisco.edi.ncclient.api.NetconfKillSession
- com.cisco.edi.ncclient.api.NetconfDeleteConfig
- com.cisco.edi.ncclient.clienttool.session.XMLSession
- com.cisco.edi.ncclient.api.util.Device

**com.cisco.edi.ncclient.api.NetconfHello**

```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId){
returns Hello request.
Hello request must be sent before associating device or server, after opening an XML session. To open an XML session see `com.cisco.edi.ncclient.clienttool.session.XMLSession`.
At times the Hello request may not give a response.
```

**Note**

If you specify a value for the RPC message ID, then the RPC reply will also have this value. The default message ID is 1.

**com.cisco.edi.ncclient.api.NetconfGet**

NetconfGet retrieves configuration information. To retrieve configurational information, the NetconfGet is equivalent to NetconfGetConfig with running configuration as the source.
```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId){
returns NetconfGet request.
For message ID information, see Note.
```

**com.cisco.edi.ncclient.api.NetconfGetShowCmd**

NetconfGetShowCmd is used to retrieve operational information. It takes `showcommand` as the input.
If the `showcommand` is null (empty string) it throws an exception.
```java
public String getNetconfShowCmdRequest(messageId)
public String getNetconfShowCmdRequest()
For message ID information, see Note.
```

**com.cisco.edi.ncclient.api.NetconfGetConfig**

NetconfGetConfig is used to retrieve all or part of the configuration of the device. A filter may be specified to retrieve only a part of the configuration. See `com.cisco.edi.ncclient.api.NetconfFilter`.
```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId){
returns GetConfig request.
```
For message ID information, see Note.

**com.cisco.edi.ncclient.api.NetconfFilter**

NetconfFilter is used to prepare a request to get a part of the configuration. NetconfGetConfig uses this filter to specify what configuration to get.

```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId){
    returns GetNetconf XML String.
}
```

User needs to set device type, OS version and IDU version exactly from the E-DI supported OS-Device types.

For message ID information, see Note.

**com.cisco.edi.ncclient.api.NetconfCopyConfig**

NetconfCopyConfig creates or replaces an entire configuration datastore with the contents of another complete configuration datastore. If the target datastore exists, it is overwritten. Otherwise, a new one is created.

The source can be any running, startup or candidate configuration datastore.

```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId){
    returns NetconfCopyConfig request.
}
```

For message ID information, see Note.

**com.cisco.edi.ncclient.api.NetconfValidate**

NetconfValidate validates the contents of the specified configuration.
Cisco E-DI supports the validate operation on candidate, running and startup configurations. Currently, Cisco E-DI only does syntax checking.

```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId){
    returns NetconfValidate request.
}
```

**com.cisco.edi.ncclient.api.NetconfLock**

NetconfLock allows the client to lock a local configuration datastore of a device in Cisco E-DI. The datastore can be any running, startup or candidate configuration datastore. If a datastore is locked, access to it is limited to read-only for any other clients. All locks will be cleared on a normal or abnormal session termination.

```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId){
    returns NetconfLock request.
}
```
com.cisco.edi.ncclient.api.NetconfCommit

When a candidate’s configuration content is complete, the configuration data can be committed, publishing the dataset to the device, and requesting the device to conform to the behavior described in the new configuration. NetconfCommit is used to commit the candidate configuration as the new current configuration of the device.

```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId)
```
returns NetconfCommit request.
For message ID information, see Note.

com.cisco.edi.ncclient.api.NetconfDiscardChanges

If the client decides that the candidate configuration should not be committed, NetconfDiscardChanges can be used to revert the candidate configuration to the current committed configuration. If the client decides that the candidate configuration should not be committed, NetconfDiscardChanges can be used to discard the candidate configuration. Cisco E-DI then sets the candidate configuration to an empty store.

```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId)
```
returns NetconfDiscardChanges request.
For message ID information, see Note.

com.cisco.edi.ncclient.api.NetconfUnLock

NetconfUnLock releases a configuration lock, previously obtained with NetconfLock. A client is not permitted to unlock a configuration datastore that it did not lock. The administrator can use NetconfKillSession to terminate a lock acquired by any user.

```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId)
```
returns NetconfUnLock request.
For message ID information, see Note.

com.cisco.edi.ncclient.api.NetconfCloseSession

NetconfCloseSession terminates an active session gracefully. Any pending requests are allowed to complete. Upon termination, any active locks are released.

```java
public static String getNetconfXMLRequestString()
public static String getNetconfXMLRequestString(String messageId)
```
returns NetconfCloseSession request.
For message ID information, see Note.
Chapter 7      Java SDK for XML PI

Accessing the Java APIs

`com.cisco.edi.ncclient.api.NetconfKillSession`

NetconfKillSession is used by one session to kill another session. It can be used by the administrator to terminate any user sessions.

For example, a NETCONF session(A) can kill another NETCONF session(B) provided A has admin privileges or has FULL_CONTROL on SERVER. When a session is killed, any operations pending on it will be removed, and not executed by the Cisco E-DI server. Any locks held by the session will be released. However, if there is an operation currently being executed by Cisco E-DI, for example an edit-config, that operation cannot be stopped. Changes made to the candidate configuration will be removed. Changes to other config stores are always implicitly committed by Cisco E-DI.

```java
public static String getNetconfXMLRequestString()
returns NetconfKillSession request.
For message ID information, see Note.
```

`com.cisco.edi.ncclient.api.NetconfDeleteConfig`

NetconfDeleteConfig deletes a configuration datastore. The <running> configuration datastore cannot be deleted. Only startup configuration can be deleted.

```java
public static String getNetconfXMLRequestString()
returns NetConfDeleteConfig request string.
```

`com.cisco.edi.ncclient.clienttool.session.XMLSession`

This class provides higher level access to E-DI server to create a netconf session, sends requests, receives responses, and also provides session management methods.

```java
public XmlSession()
public XmlSession() This constructor checks for log4j.properties. You need to set log4j.properties file using System.setProperty("properties file path").
public String openSession(StringBuffer helloFromEDI, int serverPort, String server, String login, String password, int transportType , String keyFile , String propertiesFile) throws ExceptionopenSession
method opens an XML session with E-DI.
serverPort is the EDI Telnet port. This should be 2323 for both Telnet and SSH.
server is E-DI server (IP or DNS name).
login is E-DI login.
password is E-DI password.
transportType 1 stands for Telnet, and 2 stands for SSH. If transportation type is SSH, any client machine can open only one SSH session as opensession does port forwarding. The response from the server listens at the 7777 port on the client side, since the client side port 7777 is open and engaged with the opened session.
keyFile Set this keystore file path: E-Di Install Location\edi\resources\server\.keystore
This should set for SSH connection, other it should be null.
propertiesFile
```
public void closeSession()
        closeSession method closes the current session.

public String getSessionId()
        getSessionId method returns the current session ID.

public boolean isConnected()
        isConnected returns the current session status.

public String sendRequest(String xmlRequest) throws Exception
        sendRequest method sends the XML request to the Server and returns response.

public String associateDevice(com.cisco.edi.ncclient.api.util.Device d) throws Exception
        associateDevice method associates the device with the current session.

public String sendHello(String hello) throws Exception
        sendHello method sends Hello request.

For message ID information, see Note.Command Translator

com.cisco.edi.ncclient.api.util.Device

public Device(IpAddress ipAddress,String login,String password) throws Exception
        IpAddress is device IP address.

login is the device username.

password is the device password.

public Device(String name,String login,String password) throws Exception
        name The only valid value is server. Any other value will result in an Exception being thrown.

login is the device username.

password is the device password

public String getNetconfXMLRequestString(String messageId)
        getNetconfXMLRequestString is the associate device request.

public String getNetconfXMLRequestString()
        getNetconfXMLRequestString is the associate device request.

public String getIPAddress() {
        getIPAddress method returns IP address of the device.

void setIPAddress(String ip) throws Exception
        setIPAddress method sets the IP address for the device.

public void setLogin(String login) throws Exception
        setLogin method sets the username for the device.

public void setPassword(String password) throws Exception
        sets the password for the device.

For message ID information, see Note.
## Level of Support for NETCONF Protocol Operations and Features

Table A-1 gives details of each operation and limitations, if any, in the current implementation.

**Note** For more details, refer to the NETCONF Configuration Protocol draft-ietf-netconf-prot-07 document which is included on the Cisco E-DI Documentation CD-ROM.

<table>
<thead>
<tr>
<th>#</th>
<th>Operation</th>
<th>Example (complete or segment of interest)</th>
<th>Supported</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td></td>
<td><code>&lt;source&gt;&lt;startup/&gt;&lt;/startup&gt;&lt;source&gt;</code></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td></td>
<td><code>&lt;source&gt;&lt;candidate/&gt;&lt;/candidate&gt;&lt;source&gt;</code></td>
<td>Yes</td>
<td>A candidate config store is private to a NETCONF session. A candidate config store is empty when a session is created. Configuration can be stored in this store either by an edit-config operation or by a copy-config operation.</td>
</tr>
<tr>
<td>1.4</td>
<td></td>
<td><code>&lt;source&gt;&lt;config/&gt;&lt;/config&gt;&lt;source&gt;</code></td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
## Table A-1 Protocol Operations and Level of Support in Cisco E-DI 2.2 (continued)

<table>
<thead>
<tr>
<th>#</th>
<th>Protocol Operation</th>
<th>Example (complete or segment of interest)</th>
<th>Supported</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
<td><code>&lt;source&gt;</code> <code>&lt;url&gt;</code></td>
<td></td>
<td>No.</td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;&gt;</code> <code>&lt;rpc message-id=&quot;123&quot; xmlns=&quot;urn:ietf:params:xml:ns:netconf:base:1.0&quot; xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;&gt;</code> <code>&lt;get-config&gt;</code> <code>&lt;source&gt;</code> <code>&lt;running&gt;</code> <code>&lt;filter type=&quot;subtree&quot;&gt;</code> <code>&lt;filter&gt;</code> <code>&lt;get-config&gt;</code> <code>&lt;/rpc&gt;</code></td>
<td>Yes subtree filters are supported.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td><code>&lt;filter type=&quot;xpath&quot;&gt;</code> <code>&lt;filter&gt;</code></td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;&gt;</code> <code>&lt;rpc xmlns=&quot;urn:ietf:params:xml:ns:netconf:base:1.0&quot; xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot; xsi:schemaLocation=&quot;urn:ietf:params:xml:ns:netconf:base:1.0 D:\edi\netconf_07.xsd&quot; message-id=&quot;123&quot;&gt;</code> <code>&lt;get&gt;</code> <code>&lt;/rpc&gt;</code></td>
<td>Partial In this implementation get is same as get-config of running</td>
<td>Operational data is not defined in this release.</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;&gt;</code> <code>&lt;rpc xmlns=&quot;urn:ietf:params:xml:ns:netconf:base:1.0&quot; xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot; xsi:schemaLocation=&quot;urn:ietf:params:xml:ns:netconf:base:1.0 D:\edi\netconf_07.xsd&quot; message-id=&quot;123&quot;&gt;</code> <code>&lt;get&gt;</code> <code>&lt;filter type=&quot;subtree&quot;&gt;</code> <code>&lt;filter&gt;</code> <code>&lt;/rpc&gt;</code></td>
<td>Yes subtree filtering is supported</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;&gt;</code> <code>&lt;rpc xmlns=&quot;urn:ietf:params:xml:ns:netconf:base:1.0&quot; xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot; xsi:schemaLocation=&quot;urn:ietf:params:xml:ns:netconf:base:1.0 D:\edi\netconf_07.xsd&quot; message-id=&quot;123&quot;&gt;</code> <code>&lt;get&gt;</code> <code>&lt;filter type=&quot;xpath&quot;&gt;</code> <code>&lt;filter&gt;</code> <code>&lt;/rpc&gt;</code></td>
<td>No xpath filtering is not supported</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table A-1       Protocol Operations and Level of Support in Cisco E-DI 2.2 (continued)

<table>
<thead>
<tr>
<th>#</th>
<th>Protocol Operation</th>
<th>Example (complete or segment of interest)</th>
<th>Supported</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0</td>
<td>edit-config</td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;rpc message-id=&quot;123&quot;&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;config&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;/config&gt;`</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;/edit-config&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;/rpc&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Refer to items 3.1 to 3.11 for details</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>(target)</td>
<td><code>&lt;url&gt;</code></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;/url&gt;`</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>(target)</td>
<td><code>&lt;running&gt;</code></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;/running&gt;`</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>(target)</td>
<td><code>&lt;startup&gt;</code></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;/startup&gt;`</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>&lt;default-operation&gt;merge&lt;/default-operation&gt;</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>&lt;default-operation&gt;nombre&lt;/default-operation&gt;</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>&lt;default-operation&gt;replacer&lt;/default-operation&gt;</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>&lt;test-option&gt;test-then-set&lt;/test-option&gt;</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>&lt;test-option&gt;set&lt;/test-option&gt;</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In Cisco E-DI, test is always done implicitly through transform functions. In case of errors Cisco E-DI does not report them to the user.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>&lt;error-option&gt;ignore-error&lt;/error-option&gt;</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.10</td>
<td>&lt;error-option&gt;rollback-on-error&lt;/error-option&gt;</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.11</td>
<td>&lt;error-option&gt;stop-on-error&lt;/error-option&gt;</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0</td>
<td>close-session</td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;&gt;</code></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;rpc message-id=&quot;123&quot;&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;/close-session&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;/rpc&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>When a session is closed, any locks held are released and the session is terminated. Any requests pending will be discarded.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table A-1  Protocol Operations and Level of Support in Cisco E-DI 2.2 (continued)

<table>
<thead>
<tr>
<th>#</th>
<th>Protocol Operation</th>
<th>Example (complete or segment of interest)</th>
<th>Supported</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>commit</td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;&gt;</code> <code>&lt;rpc message-id=&quot;123&quot;&gt; xmlns=&quot;urn:ietf:params:xml:ns:netconf:base:1.0&quot; xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-instance&quot;&gt; &lt;commit/&gt;&lt;/commit&gt; &lt;/rpc&gt;</code></td>
<td>Yes</td>
<td>When commit is issued, any configuration stored in the candidate store is written to the running configuration unless the device is locked by any client through Cisco E-DI or running store is locked by any other NETCONF session through Cisco E-DI.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;commit-confirmed&gt; &lt;/commit&gt;</code></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;commit-confirmed&gt; &lt;/commit&gt; &lt;/commit&gt;</code></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;copy-config&gt; &lt;/copy-config&gt;</code></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td></td>
<td><code>&lt;source&gt;candidate/&gt;&lt;/source&gt;</code></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td></td>
<td><code>&lt;source&gt;running/&gt;&lt;/source&gt;</code></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td></td>
<td><code>&lt;source&gt;running/&gt;&lt;/source&gt;</code></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>6.4</td>
<td></td>
<td><code>&lt;source&gt;startup/&gt;&lt;/source&gt;</code></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>6.5</td>
<td></td>
<td><code>&lt;source&gt;candidate/&gt;&lt;/source&gt;</code></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>6.6</td>
<td></td>
<td><code>&lt;source&gt;startup/&gt;&lt;/source&gt;</code></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>6.7</td>
<td></td>
<td><code>&lt;target&gt;url/&gt;&lt;/target&gt;</code></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>6.8</td>
<td></td>
<td><code>&lt;source&gt;url/&gt;&lt;/source&gt;</code></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>6.9</td>
<td></td>
<td><code>&lt;source&gt;config/&gt;&lt;/source&gt;</code></td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>6.10</td>
<td></td>
<td><code>&lt;target&gt; config/&gt;&lt;/target&gt;</code></td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
### Table A-1  Protocol Operations and Level of Support in Cisco E-DI 2.2 (continued)

<table>
<thead>
<tr>
<th>#</th>
<th>Protocol Operation</th>
<th>Example (complete or segment of interest)</th>
<th>Supported Notes</th>
</tr>
</thead>
</table>
| 7 | `kill-session` | ```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:ietf:params:xml:ns:netconf:base:1.0 D:\edi\netconf_07.xsd"
     message-id="123">
    <kill-session>
    <session-id/>
    </kill-session>
</rpc>
``` | A NETCONF session cannot kill itself.
A NETCONF session cannot kill a non-NETCONF session.
When a session is killed, any operations pending in that session are discarded. Any locks held by that session are released.
Only a NETCONF Session with admin privileges or having FULL_CONTROL on Cisco E-DI server can kill a NETCONF Session. |
| 8 | `lock` | ```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:ietf:params:xml:ns:netconf:base:1.0 D:\edi\netconf_07.xsd"
     message-id="123">
    <lock>
    <target/>
    </lock>
</rpc>
``` | Refer to Items 8.1 to 8.4 |
| 8.1 | `target<candidate"></candidate"></target>` | Yes | Candidate configuration is a private store for each session and is not a shared resource. Hence, there is no requirement to obtain an explicit lock on candidate configuration. |
| 8.2 | `target<running"></running"></target>` | Yes |
| 8.3 | `target<startup"></startup"></target>` | Yes |
| 8.4 | `target<url"></url"></target>` | No |
| 9 | `unlock` | ```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:ietf:params:xml:ns:netconf:base:1.0 D:\edi\netconf_07.xsd"
     message-id="123">
    <unlock>
    <target/>
    </unlock>
</rpc>
``` | Refer to Items 9.1 to 9.4 |
| 9.0 | | | |
Appendix A  Level of Support for NETCONF Protocol Operations and Features

Table A-1  Protocol Operations and Level of Support in Cisco E-DI 2.2 (continued)

<table>
<thead>
<tr>
<th>#</th>
<th>Protocol Operation</th>
<th>Example (complete or segment of interest)</th>
<th>Supported</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>&lt;target&gt;&lt;candidate&gt;&lt;/candidate&gt;&lt;/target&gt;</td>
<td>Yes</td>
<td>Candidate configuration is a private store for each session and is not a shared resource. Hence, there is no requirement to obtain an explicit lock or release lock on candidate configuration.</td>
<td></td>
</tr>
<tr>
<td>9.2</td>
<td>&lt;target&gt;&lt;running&gt;&lt;/running&gt;&lt;/target&gt;</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.3</td>
<td>&lt;target&gt;&lt;startup&gt;&lt;/startup&gt;&lt;/target&gt;</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.4</td>
<td>&lt;target&gt;&lt;url&gt;&lt;/url&gt;&lt;/target&gt;</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>validate</td>
<td>Only syntactic validation is conducted. No semantic validation is done. Refer to Items 10.1 to 10.5.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.0</td>
<td></td>
<td></td>
<td>Refer to Items 10.1 to 10.5</td>
<td></td>
</tr>
</tbody>
</table>
NETCONF Client GUI

The NETCONF client is a simple GUI client application that can be used to understand the implementation of the NETCONF protocol in Cisco E-DI.

This appendix includes the following information:

- Client Application Files
- Starting a NETCONF Session
- Making a NETCONF Request to the Cisco E-DI Server
- Ending the NETCONF Session

Client Application Files

The client is distributed as a set of three jar files, one startup script and a directory of example requests. Copy the client files from Cisco E-DI Documentation CD-ROM to D:\eft\client. The structure should look similar to the structure in figure that follows.
Starting a NETCONF Session

You will need the following information before you can create a NETCONF session:
1. The location of the Cisco E-DI server. For example, the localhost if it is running locally, or the IP address.
2. The username and password to login to Cisco E-DI
3. The IP address of the device to be managed through the NETCONF protocol.

Start the client GUI, enter the following command:

d:\soft\client\bin\ncclient .\examples
The client GUI opens. See the following figure.

Figure B-2 GUI Example

The GUI has not connected to Cisco E-DI server yet.

Refer to the Cisco Enhanced Device User’s Guide, 2.2 for information about operational modes such as session based device authentication, and IP-aliasing mode.

Table B-1 GUI Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDI Server</td>
<td>The IP address or the name of the Cisco E-DI server.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> If Cisco E-DI is running in IP-Aliasing mode and if the device to be managed is aliased, then that alias can be used in this field.</td>
</tr>
<tr>
<td>Login</td>
<td>The login id on the Cisco E-DI server.</td>
</tr>
<tr>
<td>Password</td>
<td>The password on the Cisco E-DI server.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> If a device alias is used (as above), then the login and password should be that of the device.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> If Cisco E-DI is running in session based device authentication mode, the credentials of the device are expected to be available on Cisco E-DI itself.</td>
</tr>
<tr>
<td>NC Session With Device</td>
<td>This is the IP address or the name of the device to be managed in this session.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> If Cisco E-DI is running in IP-Aliasing mode and the device is aliased, then you can use the device alias in the 'EDI Server' field and skip filling this field.</td>
</tr>
</tbody>
</table>
Starting a NETCONF Session

Enter the following information:

- Cisco E-DI server address (EDI Server field)
- Username (Login field)
- Password (Password field)
- Device IP address (NC Session with Device field)
- Device login—Required when session based device authentication is enabled.
- Device password—Required when session based device authentication is enabled.

Note
Cisco E-DI can have session based device authentication enabled or disabled. The device credentials (device login and password) are required when session based device authentication is enabled. Otherwise, these credentials are optional.

Note
When the session is established, the fields cannot be edited. To change the information, you will have to close the current client session, and open a new session.

Click Connect to E-DI.

Table B-1  GUI Fields (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Login</td>
<td>The login id to be used by Cisco E-DI during device authentication.</td>
</tr>
<tr>
<td></td>
<td>Note In IP-Aliasing mode this field is disabled.</td>
</tr>
<tr>
<td>Device Password</td>
<td>The password to be used by Cisco E-DI during device authentication.</td>
</tr>
<tr>
<td></td>
<td>Note In IP-Aliasing mode this field is disabled.</td>
</tr>
</tbody>
</table>
Making a NETCONF Request to the Cisco E-DI Server

When a session is created, the Session ID is added to the device IP field. The Connect to E-DI button now says **Send Hello**, and the button outline is colored cyan. See the figure that follows.

**Figure B-3**  Hello Message Exchange

![Hello Message Exchange](image)

**Making a NETCONF Request to the Cisco E-DI Server**

When a session with Cisco E-DI is created, you see a window similar to **Figure B-3**. When a client connects to Cisco E-DI, it sends a hello message from the NETCONF agent which contains the session id. In **Figure B-3**, the hello message is displayed in the response text area (lower left pane). The client extracts the session id, this is displayed in the Session id field.

The button is now **Send Hello**. As specified in the NETCONF protocol, both the agent (Cisco E-DI) and the manager (the GUI client) are required to send hello messages.

The default hello message from the client to Cisco E-DI is displayed in the request text area (upper left pane). You can edit this message, for example, to specify a different set of capabilities.

At this point, you can enter the device information (the device id, device login and password), and the tool will provide the **associate-devices** message which can be used after **send hello** is sent.

**Note**

The tool now has a connection with the Cisco E-DI server, but Cisco E-DI does not know which device you want to manage through the XML PI.

Enter the device information, and click **Send Hello**.
The tool prepares the default associate devices message, and displays it in the request text area. See Figure B-4. Associate devices is a Cisco E-DI specific operation. It is required to inform Cisco E-DI about the device the client wants to manage through the XML PI.

Click Associate With Device.

Figure B-4    Associate With Devices

When Associate devices is successful, Cisco E-DI now knows which device is being managed in this NETCONF session, and is ready to start NETCONF operations. See Figure B-5.

You can now send protocol messages such as get-config, and edit-config. In the following example, the user navigates to the device specific get-config protocol message.
Note

The NETCONF session is active when the first button reads Submit Request.

Know the Device Type

Each device type and OS combination has its own set of data model elements (the XSD). In operations where you have to express device commands in XML format, you will need to refer to the device specific namespace. The operations `edit-config` and `get-config` or `get` with `filter` will require this information.

An operation where the user is not expressing a specific command set in XML format does not need this namespace information. For example, `lock` is one operation which refers to a data store, and not configuration data.

If you do not know the device type, you can obtain the information from Cisco E-DI as follows:

Use an operation that does not require the namespace to be specified. For example, you can use a `get` operation without the `filter` element. This means that the `get` operation without a `filter` element looks the same no matter what device you are talking to. Select any `get` from any device specific example in the hierarchy in the right pane.

In Figure B-6, a `get` from the Cat6500 set of examples is selected, and the response shows that the device 172.25.86.106 is a Cisco2600.
Now that you know the device type, you can use the Cisco2600 specific example set. In this example, navigate through the hierarchy to locate the Cisco 2600 to see the running configuration of the device 172.25.86.106. See Figure B-7.

**Figure B-6** Finding the Device Type

**Figure B-7** Displaying the Running Configuration
Ending the NETCONF Session

When you want to close the session, you can use the close-session.xml file from the Sample requests in the right pane. You will see a window similar to Figure B-8.

Click Submit Request.

Note: You can also click the Close button to close the window, and end the session.

Figure B-8 Closing a Session
Ending the NETCONF Session
CPI Schema

This section gives an example of a Cisco Programmatic Interface (CPI) schema.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.cisco.com/cpi_10/schema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="unqualified">
<!-- The following elements define the cisco extensions for the content of the filter element in a <get-config> request. They allow the client to specify the format of the response and to select subsets of the entire configuration to be included. -->
- <xs:element name="config-format-text-block">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="text-filter-spec" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
- <xs:element name="config-format-text-cmd">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="text-filter-spec" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
- <xs:element name="config-format-xml">
  <xs:annotation>
    <xs:documentation>If this element appears in the filter of a get-config request, the results are to be returned in E-DI XML format. The content of this element is treated as a filter.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:anyType" />
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
- <xs:element name="oper-data-format-text-block">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="text-filter-spec" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
- <xs:element name="oper-data-format-text-cmd">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="text-filter-spec" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
- <xs:element name="oper-data-format-xml">
  <xs:annotation>
    <xs:documentation>If this element appears in the filter of a get-config request, the results are to be returned in E-DI XML format. The content of this element is treated as a filter.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:anyType" />
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
</xs:schema>
```
<xs:element name="show" type="xs:string" maxOccurs="unbounded" />
</xs:sequence>
</xs:complexType>
</xs:element>
- <xs:element name="oper-data-format-xml">
  - <xs:complexType>
    - <xs:sequence>
      <xs:any />
    </xs:sequence>
  </xs:complexType>
</xs:element>
- <!-- When config-format-text format is specified, the following describes the content of the data element in the response -->
  - <xs:element name="cli-config-data">
    - <xs:complexType>
      - <xs:sequence>
        <xs:element name="cmd" type="xs:string" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>Content is a command. May be multiple lines.</xs:documentation>
            <xs:documentation>Contains embedded newline characters that must be preserved as they represent the boundaries between the individual command lines.</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  - <xs:element name="cli-config-data-block" type="xs:string">
    <xs:documentation>The content of this element is the device configuration as it would be sent to a terminal session. It contains embedded newline characters that must be preserved as they represent the boundaries between the individual command lines.</xs:documentation>
  </xs:element>
  - <xs:element name="text-filter-spec">
    <xs:annotation>
      <xs:documentation>If this element is included in the config-format-text element, then the content is treated as if the string was appended to the "show running-config" command line.</xs:documentation>
    </xs:annotation>
  </xs:element>
  - <xs:element name="cli-oper-data-block">
    <xs:complexType>
      <xs:annotation>
        <xs:documentation>This element is included in the response to get operation. Content of this element is the operational data in text format.</xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element name="item" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="show" />
              <xs:element name="response" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  - <xs:element name="xml-oper-data">
    <xs:complexType>
      <xs:annotation>
        <xs:documentation>The content of this element is the operational data in XML format. May be multiple lines.</xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:any />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
This element is included in the response to get operation. Content of this element is the operational data in xml format.

This element is included in the response to get-config and get operations. Content of this element is the configuration data in xml format.
Cisco E-DI NETCONF Extension Schema

This section gives an example of a Cisco E-DI NETCONF extension schema.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema targetNamespace="http://www.cisco.com/edi_2.2/schema/netconf_extension"
  elementFormDefault="qualified" attributeFormDefault="unqualified" xml:lang="en"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">

  <!-- import standard XML definitions -->

  <!-- import base netconf definitions -->
  <xs:import namespace="urn:ietf:params:xml:ns:netconf:base:1.0"
    schemaLocation="c:\netconf\netconf_draft07.xsd" />

  <!-- simple types -->
  <xs:simpleType name="LoginType">
    <xs:annotation>
      <xs:documentation>User id</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

  <xs:simpleType name="PasswordType">
    <xs:annotation>
      <xs:documentation>Password</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

  <xs:simpleType name="IPAddressType">
    <xs:annotation>
      <xs:documentation>IP address</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

  <!-- complex types -->
  <xs:complexType name="CredentialsType">
    <xs:annotation>
      <xs:documentation>User credentials for target device(s)</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="login" type="LoginType" minOccurs="0"/>
      <xs:element name="password" type="PasswordType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="DeviceType">
    <xs:annotation>
      <xs:documentation>Device information</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="ipAddress" type="IPAddressType"/>
    </xs:sequence>
  </xs:complexType>

  <!-- enumerations -->
  <xs:simpleType name="DeviceStatusType">
    <xs:annotation>
      <xs:documentation>Device status</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="up"/>
      <xs:enumeration value="down"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- bindings -->
  <xs:element name="login" type="LoginType"/>
  <xs:element name="password" type="PasswordType"/>
  <xs:element name="ipAddress" type="IPAddressType"/>
  <xs:element name="device" type="DeviceType"/>
  <xs:element name="deviceStatus" type="DeviceStatusType"/>
</xs:schema>
```
An IP Address in standard notation:

- `<xs:restriction base="xs:string">
- `<xs:pattern value="((1[0-9]{1,2}|2[0-4]\d|25[0-5])\.(1[0-9]\d|2[0-4]\d|25[0-5])\.(3[0-2]\d|2[0-4]\d|25[0-5])\.(1[0-9]\d|2[0-4]\d|25[0-5]))">
- `<xs:annotation>
  `<xs:documentation>An IP Address should be of the form 172.168.11.23. This data type restricts the value of each field between 0 and 255 ie, [0-255].[0-255].[0-255].[0-255].</xs:documentation>
  </xs:annotation>
</xs:restriction>
</xs:simpleType>

- `<xs:simpleType name="NameType">
- `<xs:restriction base="xs:string">
- `<xs:minLength value="1" />
</xs:restriction>
</xs:simpleType>

- `<xs:complexType name="DeviceType">
- `<xs:annotation>
  `<xs:documentation>Specifies target device</xs:documentation>
</xs:annotation>
- `<xs:choice>
  `<xs:element name="ipaddress" type="IPAddressType" />
  `<xs:element name="name" type="NameType" />
</xs:choice>
</xs:complexType>

- `<xs:complexType name="NetworkType">
- `<xs:annotation>
  `<xs:documentation>Specifies network for the current context</xs:documentation>
</xs:annotation>
- `<xs:sequence>
  `<xs:element name="device" type="DeviceType" />
  `<xs:element name="credentials" type="CredentialsType" minOccurs="0" />
</xs:sequence>
</xs:complexType>

- `<associate-devices> operation

---

- `<xs:complexType name="associateDevicesType">
- `<xs:complexContent>
  `<xs:extension base="netconf:rpcOperationType">
    `<xs:sequence>
      `<xs:annotation>
        `<xs:documentation>indicates the device context required for the current netconf session</xs:documentation>
      </xs:annotation>
      `<xs:choice>
        `<xs:element name="network" type="NetworkType" />
      </xs:choice>
    </xs:sequence>
  </xs:extension>
</xs:complexContent>
</xs:complexType>

</xs:element>
</xs:schema>
Open Source License Acknowledgement

Cisco E-DI 2.2 uses third-party open source software subject to the following licenses:

- telnetd
- Java Service Wrapper
- Apache License Version 2.0
- Apache 1.1
- PostgreSQL License
- Javolution 4.1

**telnetd**

Java TelnetD library (embeddable telnet daemon)

Copyright (c) 2000-2005 Dieter Wimberger.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**Java Service Wrapper**

Copyright (c) 1999, 2005 Tanuki Software Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the Java Service Wrapper and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sub-license, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Apache License Version 2.0

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" means the rights and licenses as defined in this document, including the rights and licenses in source code, object code, object form, received and可供 distribution under the License.

"Licensed Work" means the work or contributions made available available under the License.

"Contribution" means any work or contribution made available available under the License.

"Contributor" means any person or entity that makes such a Contribution.

"Licensee" means any person or entity that receives a Contribution under the License.

"Patent License" means the right to practice or otherwise exercise any Patent covered by a License.

"Contributor License" means a license granted by a Contributor to Licensees.

"Contributor Notice" means a notice provided by a Contributor that is included with a Contribution.

"Source form" means the source code form of the Work and any contributed copyrighted works.

"Object form" means the object code form of the Work and any contributed compiled works.

"Derivative Work" means a work that is derived from the Source form of the Work and/or any contributed copyrighted works.

"Licensee Notice" means a notice provided by the Licensee to Licensees that is included with the Licensee.

"Derivative Work Notice" means a notice provided by a Licensee to Licensees that is included with a Derivative Work.

"Copyright Notice" means a notice provided by the copyright owner to the public that is included with the Work.

2. Grant of Copyright License.

Licensee hereby grants to Licensee a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works, distribute, and publicly display, publicly perform the Work and any contributed copyrighted works in Source or Object form, as well as a right to sublicense the same.


Contributor hereby grants to Licensee a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable patent license to make, have made, use, offer to sell, sell, import, and otherwise exercise any Patent covered by such Contribution.

4. Redistribution.

Licensee may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that Licensee meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
(b) You must retain, in the Source form of any Derivative Works that You distribute, any copyright, patent, trademark, and attribution notices from the Source form of the Work; and
(c) You must retain, in the Source form of any Derivative Works that You distribute, any copyright, patent, trademark, and attribution notices from the Source form of the Work; and
(d) You must retain, in the Source form of any Derivative Works that You distribute, any copyright, patent, trademark, and attribution notices from the Source form of the Work; and
(e) You must retain, in the Source form of any Derivative Works that You distribute, any copyright, patent, trademark, and attribution notices from the Source form of the Work; and
(f) You must retain, in the Source form of any Derivative Works that You distribute, any copyright, patent, trademark, and attribution notices from the Source form of the Work; and
(g) You must retain, in the Source form of any Derivative Works that You distribute, any copyright, patent, trademark, and attribution notices from the Source form of the Work; and
(h) You must retain, in the Source form of any Derivative Works that You distribute, any copyright, patent, trademark, and attribution notices from the Source form of the Work; and
(i) You must retain, in the Source form of any Derivative Works that You distribute, any copyright, patent, trademark, and attribution notices from the Source form of the Work; and
(j) You must retain, in the Source form of any Derivative Works that You distribute, any copyright, patent, trademark, and attribution notices from the Source form of the Work.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contributions submitted by You or any other person to the Licensor for inclusion in the Work are made available available under the License.

6. Trademarks.

Contributor Licensees may use trademarks, service marks, or other designations as of the date they were first used by Contributor.

7. Disclaimer of Warranty.

Licensee grants to Licensee and any other Licensee that obtains a copy of the Work or Derivative Works from Licensee a disclaimers of warranty as of the date of receipt of such Work or Derivative Works.

8. Limitation of Liability.

Licensee grants to Licensee and any other Licensee that obtains a copy of the Work or Derivative Works from Licensee a disclaimers of warranty as of the date of receipt of such Work or Derivative Works.


Licensee may terminate this License at any time by written notice to the Licensor, and Licensee shall have no further rights or obligations under this License.


This License is governed by the laws of the United States of America, without regard to its conflict of laws provisions.

11. MISCELLANEOUS.

The terms of this License are not waivable, severable, nor transferable.


Licensee accepts the terms and conditions of this License by reproduction or distribution of the Work or Derivative Works.


This License is governed by the laws of the United States of America, without regard to its conflict of laws provisions.

14. MISCELLANEOUS.

The terms of this License are not waivable, severable, nor transferable.

15. Acceptance.

Licensee accepts the terms and conditions of this License by reproduction or distribution of the Work or Derivative Works.
Apache 1.1

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

"This product includes software developed by the Apache Software Foundation (http://www.apache.org/)."

Alternate, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

The names 'Apache' and 'Apache Software Foundation' must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

Products derived from this software may not be called 'Apache', nor may 'Apache' appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

====================================================================
Appendix E  Open Source License Acknowledgement

PostgreSQL License

This software consists of voluntary contributions made by many individuals on behalf of
the Apache Software Foundation. For more information on the Apache Software Foundation,
please see <http://www.apache.org/>.

Portions of this software are based upon public domain software originally written at the
National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.

PostgreSQL License

License
PostgreSQL is released under the BSD license.
PostgreSQL Database Management System
(formerly known as Postgres, then as Postgres95)

Portions Copyright (c) 1996-2005, The PostgreSQL Global Development Group

Portions Copyright (c) 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is
hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copi es.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL,
INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND
ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED
HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE,
sUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Why not the GNU General Public License?
People often ask why PostgreSQL is not released under the GNU General Public License. The simple answer is because we like the BSD license and
do not want to change it. If you are keen to read more about this topic, then please take a look in the Archives at any of the many threads on this
subject, but please don’t start yet another debate on the subject!

Privacy Policy | Project hosted by hub.org | Designed by tinysofa
Copyright © 1996 – 2006 PostgreSQL Global Development Group

Javolution 4.1

Javolution - Java(TM) Solution for Real-Time and Embedded Systems
Copyright (c) 2006, Javolution (http://javolution.org)
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are
permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of
  conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
**Glossary**

**D**

**Data Model**
The implementation or concrete representation of an information model. A data model says what information is to be contained in a datastore, how the information will be used, and how the items in the datastore will be related to each other.

RFC3444 states: “Compared to IMs, DMs define managed objects at a lower level of abstraction. They include implementation- and protocol-specific details, e.g. rules that explain how to map managed objects onto lower-level protocol constructs”.

**Device-Independent CLI Model**
See Unified CLI View.

**device-version pair**
A single device with any one of its image versions. Terminology used in Cisco E-DI Command Analyzer and other user documentation.

**Two device-version pairs**
Cisco-EDI terminology. It means:
- Two devices each with a single IOS image version selected. For example, Cat2950-12.1(13)EA1c and Cat550-12.1(14)EA1a constitute two valid device-version pairs that can be compared.
- A single device with two IOS image versions selected. For example, Cat3550-12.1(14)EA1a and Cat3550-12.1(22)EA2 constitute two valid device-version pairs that can be compared.

**I**

**Information Model**
An abstract representation of the entities being managed. An information model definition includes object classes, properties/attributes, methods, and relationships. Often, an information model captures generic concepts, which are applicable across like managed entities.

**M**

**Managed Object**
An abstraction of a managed entity (e.g., a managed resource). The managed object definition includes a set of supported attributes, operations, and notifications.

**O**

**Object Model**
A collection of related managed objects forming a logical and consistent grouping.
Object Model Fragment
A uniquely identifiable portion of an object model. It is the equivalent of a single SNMP MIB module, which can be implemented by multiple NEs. It is labeled as a common object model fragment not according to its specification, but because multiple NEs implement it. That is, a subset of all object model fragments are those which are designated as being common because, and only because, they are implemented by more than one NE (i.e., an object model fragment does not become common because of any change in it's specification.)

Operational Data Modeler
This is a Cisco E-DI tool that enables you to create a unique Data Model for the operational data that is available through CLI `show` commands. Using this Model, you can further create an XML programmatic interface for this operational data. For details see the section, Using the Operational Data Modeler (ODM) in this guide.

Unified CLI View
The device-independent virtual model. Also called Device Independent CLI Model. These terms are used in Cisco E-DI documentation.

The Unified CLI View is a unionized representation of all the CLIs that are available in a set of devices/IOS versions that the you select.

XML Schema Definition
A way to describe and validate data in an XML environment. A schema is a model for describing the structure of information.
candidate configuration capability, supported capabilities 2-5
CatOS devices 1-4
CatOS devices, using a filter 1-4
Cisco E-DI PI, concept 1-1
cli-block format 3-1
cli-block mode
   edit configuration commands 3-2
   EXEIC commands 3-2
cli format 3-1
   getting a configuration using filters 3-3
   send edit configuration commands 3-1
CLI mode 3-1
close-session, session release 2-15
Command Modeler
   commands tree view 5-4
   concept, usage 5-1
   configuring 5-3
   creating a model 5-5
   generating code 5-5
   getting started with 5-1
   help, launching 5-2
   launching 5-2
   node details view 5-4
   per-requisites 5-2
   pre-model list view 5-5
   sample model file 5-7
   sample XML output file 5-3, 5-4
   troubleshooting 5-7
UI 5-4
   workflow 5-3
commit, configuration retrieval 2-13
   concept
      Cisco E-DI PI 1-1
      Operational Data Modeler 4-1
      programmatic interface 1-1
      configuration datastores 2-4
      configuration operations 2-6
      configuration retrieval
         commit 2-13
         copy-config 2-11
         delete-config 2-12
         discard-changes 2-14
         edit-config 2-10
         get 2-10
         get-config 2-9
         lock 2-12
         unlock 2-13
Configuring 5-3
   copy-config, configuration retrieval 2-11
   data model 2-16
   data model, definition 6-1
   delete-config, configuration retrieval 2-12
   device level tasks 1-2
   discard-changes, configuration retrieval 2-14
   distinct startup capability, supported capabilities 2-5
   workflow 5-3
edit-config, configuration retrieval 2-10
   edit configuration commands, cli-block mode 3-2
establishing a session  2-1
EXEC commands, cli-block mode  3-2

F
format
  cli  3-1
  cli-block  3-1

G
get, configuration retrieval  2-10
get-config, configuration retrieval  2-9
getting a configuration using filters, cli format  3-3

I
information model, definition  GL-1

K
kill-session, session release  2-15

L
Launching  5-7
limitations  2-5
lock, configuration retrieval  2-12

M
managed object, definition  GL-1

N
namespaces  2-16
NETCONF protocol operations, support  A-1

O
object model, definition  GL-1, GL-2
object model fragment, definition  GL-2
ODM tags, specifying  4-18
ok element, RPC messages  2-4
Operational Data Modeler
  attributes, general  4-21
  CLI, sample  4-17
  concept, usage  4-1
  console view  4-3
  container, adding  4-6
  container-specific attributes  4-21
    dynamic  4-21
    creating a model  4-4
    editor view  4-3
    general attributes
      alias  4-21
      legend  4-21
      name  4-21
      type  4-21
    getting started with  4-2
    header-specific attributes  4-23
      end  4-23
      nullable  4-23
      wrappable  4-23
    launching  4-2
    legend, adding  4-11
    model, modifying  4-13
    model, verifying  4-13
    property, adding  4-7
    property-specific attributes  4-22
      distance  4-22
      end-delimiter  4-22
      length  4-22
    sample, CLI  4-17
    sample spec file  4-18
    specifying elements  4-5
    table, adding  4-9
Index

programmatic interface, concept 1-1
protocol operations/services 2-4

R
rpc element, RPC messages 2-3
crp-error element, RPC messages 2-4
RPC messages
  ok element 2-4
  pipelining 2-4
  rpc element 2-3
  rpc-error element 2-4
  rpc-reply element 2-3

S
send edit configuration commands, cli format 3-1
session release
  close-session 2-15
  kill-session 2-15

ssh 1-2
subtree filtering 2-6

T
 Telnet 1-2
telnetd E-1
troubleshooting
  command modeler 5-7

U
unlock, configuration retrieval 2-13
using a filter with CatOS devices 1-4
Using Command Analyzer 6-1
using Command Modeler 5-1
using operational data modeler 4-1

V
validate capability, supported capabilities 2-5
validation 2-16
versions, SSH 1-2

W
workflow
  Command Modeler 5-3
  Operational Data Modeler 4-2
  writable-running capability, supported capabilities 2-5
Index

X

XSD, definition    GL-2
XSD files    1-1