Cisco WAE 7.1 User Guide

First Published: 2018-01-18

Last Modified: 2018-04-17

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

CONTENTS

CHAPTER 1

Overview

This section contains the following topics:

Cisco WAE Overview

The Cisco WAN Automation Engine (WAE) platform is an open, programmable framework that interconnects software modules, communicates with the network, and provides APIs to interface with external applications.

Cisco WAE provides the tools to create and maintain a model of the current network through the continual monitoring and analysis of the network and the traffic demands that are placed on it. This network model contains all relevant information about a network at a given time, including topology, configuration, and traffic information. You can use this information as a basis for analyzing the impact on the network due to changes in traffic demands, paths, node and link failures, network optimizations, or other changes.

The Cisco WAE platform has numerous use cases, including:

- Traffic engineering and network optimization—Compute TE LSP configurations to improve the network performance, or perform local or global optimization.
- Demand engineering—Examine the impact on network traffic flow of adding, removing, or modifying traffic demands on the network.
- Topology and predictive analysis—Observe the impact to network performance of changes in the network topology, which is driven either by design or by network failures.
- TE tunnel programming—Examine the impact of modifying tunnel parameters, such as the tunnel path and reserved bandwidth.
- Class of service (CoS)-aware bandwidth on demand—Examine existing network traffic and demands, and admit a set of service-class-specific demands between routers.

WAE Architecture

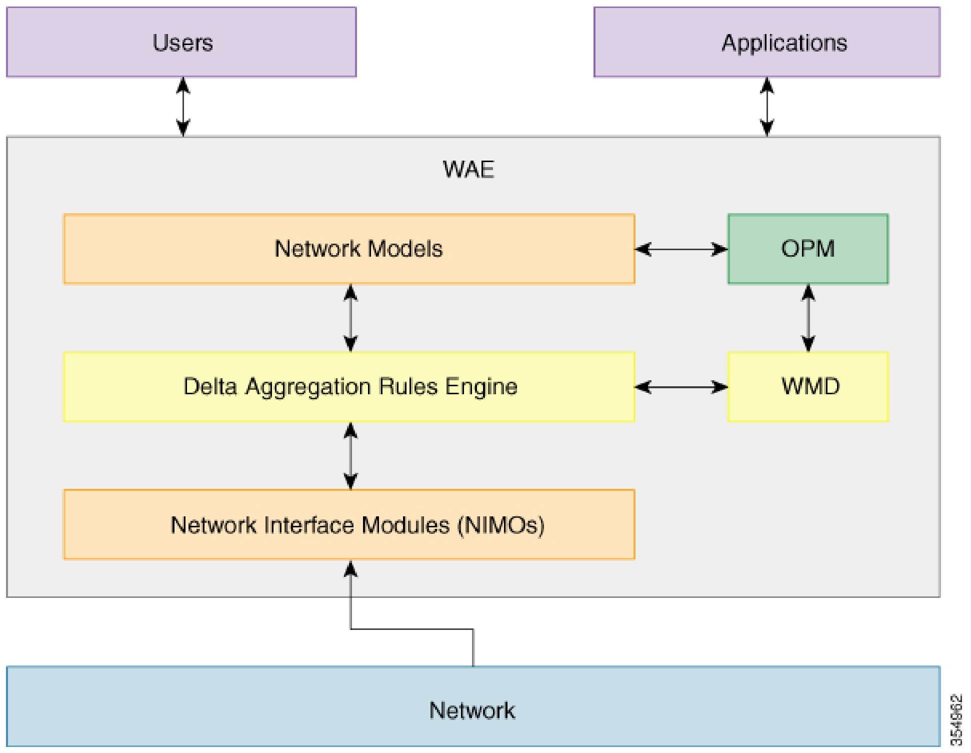
At its core, WAE defines an *abstract network model*, which can be built from an actual network by stitching together *network interface modules (NIMOs)*.

The WAE network model is defined in YANG and is *extensible* via standard YANG mechanisms. WAE itself is implemented on top of a YANG run-time system that automatically generates APIs (NETCONF, RESTConf, CLI) from the YANG models.

Optimization and prediction modules (OPMs) provide a powerful Python API to manipulate network models. The OPM API lets you operate on the network without having to worry about device-specific properties. Even if the underlying routers are replaced by routers from a different vendor, the API calls remain exactly the same.

The OPM APIs provide powerful "what-if" capabilities. For example, the OPM APIs let you answer the following questions:

- What is the impact if I bring this router down for maintenance?
- What happens if I increase the capacity of this circuit?
- Can my network handle a data center backup now?


Users
Applications
WAE
Network Models
OPM
Delta Aggregation Rules Engine
WMD
Network Interface Modules (NIMOs)
Network
354962

Optimization and Prediction Module

Optimization and prediction modules (OPMs) provides a powerful Python API to manipulate network models. The OPM API lets you operate on the network without having to worry about device-specific properties. Even if the underlying routers are replaced by routers from a different vendor, the API calls remain exactly the same.

The OPM APIs provides powerful ***what-if*** capabilities. For example, the OPM APIs let you answer the following questions:

- What is the impact if I bring this router down for maintenance?
- What happens if I increase the capacity of a particular circuit?
- Can my network handle a datacenter backup now?

Network Interface Modules

A *network interface module (NIMO)* is a WAE package that populates parts of the abstract network model, possibly querying the network to do so. Most NIMOs operate as follows:

1. They read a *source network model* (or simply, a *source model*).
2. They augment the source model with information obtained from the actual network.
3. They produce a *destination network model* (or simply, a *destination model*) with the resulting model.

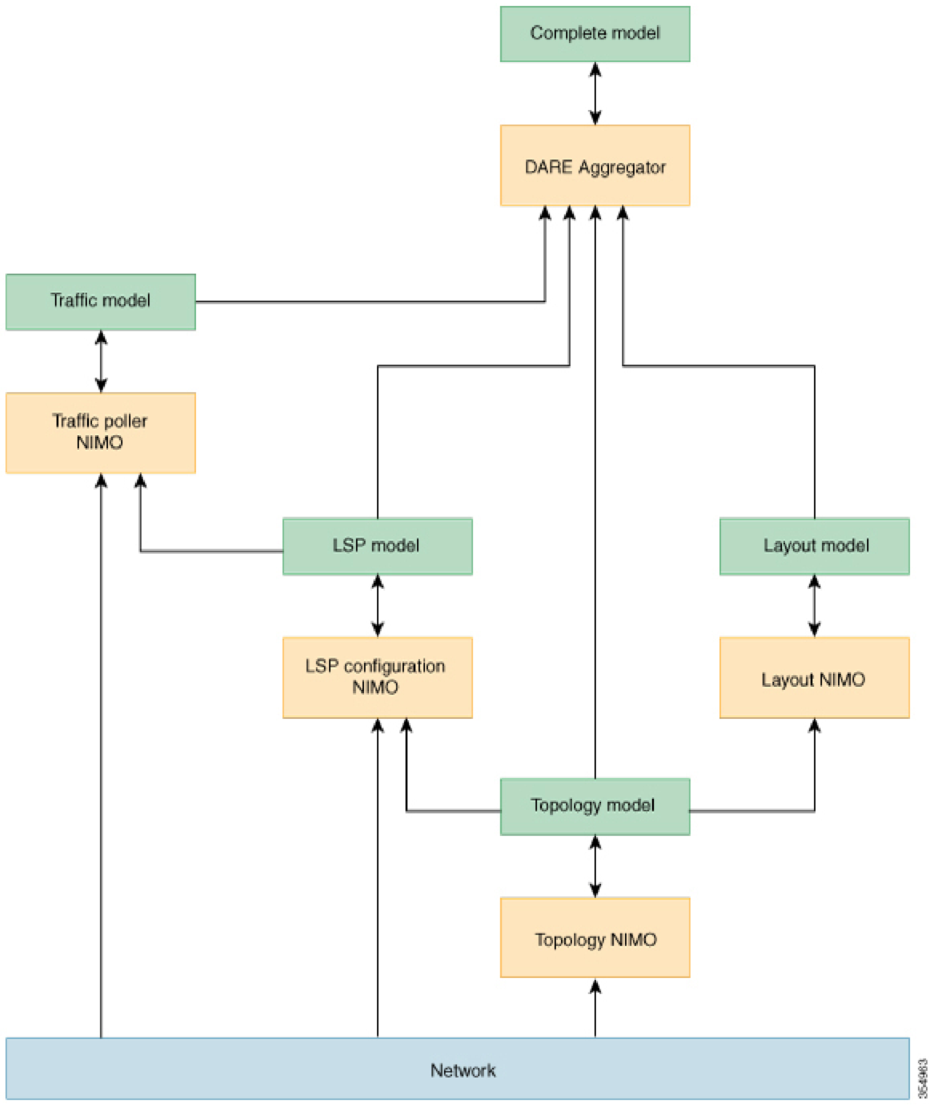
WAE includes several different NIMOs, such as:

- Topology NIMO—Populates a basic network model with topology information (nodes, interfaces, circuits) based on the discovered IGP database augmented by SNMP queries. The topology NIMO does not have a source model.
- LSP configuration NIMO—Augments a source model with LSP information, producing a destination model with the extra information.
- Traffic poller NIMO—Augments a source model with traffic statistics polled from the network, producing a new destination model with extra information.
- Layout NIMO—Adds layout properties to a source model to improve visualization. It produces a new destination model with the extra layout information. The NIMO records changes to the layout properties, so when the source model changes and the destination model is updated, the layout properties in the destination model are updated accordingly.

Network Models

A *model building chain* is an arrangement of NIMOs organized in such a way as to produce a network model with the desired information. Given the preceding NIMOs, for example, one chain could consist of the topology NIMO, followed by the LSP configuration NIMO, followed by the traffic poller NIMO. This chain contains three models, some with more information than others. Organizing model building chains lets you create different models for different use cases. You can branch the chain to have independent model building tracks.

The following diagram shows a chain tied together by the DARE aggregator:

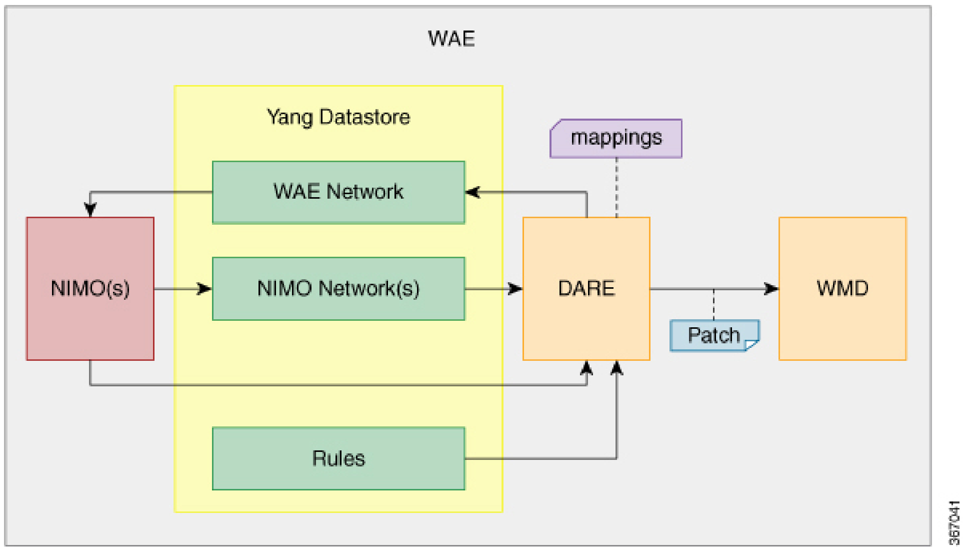


As shown in the preceding diagram, a model building chain can branch, allowing for independent, parallel model building tasks. The result is that each branch contains different model information. In the preceding example, one branch ends up with LSPs and traffic measurements; the other branch ends up with a model that can be better visualized.

The DARE aggregator is a WAE component that brings together various model building chain branches, selecting model information from each of them, and consolidating the information in a destination model. The preceding example is configured to look at all models in the chain. With the configuration shown, the aggregator immediately picks up changes to the topology model. Without a connection, the changes would have to propagate up the model building chain branches before being processed to the top-level model. The aggregator routes changes to its destination model to the correct downstream NIMO. In the preceding example, if you create an LSP at the top-level model, the aggregator forwards that change to the LSP model.

Delta Aggregation Rules Engine

Delta Aggregation Rules Engine (DARE) is a WAE component that brings together various model building chain branches, selects model information from each of them, and consolidates the information in a destination model (final network model). It collects changes to the NIMO models by subscribing to notifications and providing an API for NIMOs to publish changes directly to. These changes are aggregated based on configured rules. In addition, the changes are sent in parallel to the WAE Model Daemon (WMD) in the form of a patch. DARE stores its own state required for the aggregation in various maps on the file system.



Note Since DARE works and is based off of changes, it should be configured before changes are made to NIMO models.

For information on how to configure the aggregator to use DARE, see NIMO Collection Consolidation, on page 51.

WAE Modeling Daemon (WMD)

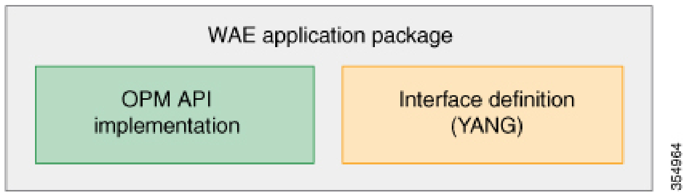
WMD receives changes from DARE, incorporating scheduled NIMO runs as well as reactive updates from the XTC Agent to Patch module. It also schedules insertions of measured traffic updates into the in-memory model from the traffic poller NIMO. All updates are consolidated into a near real-time Master Model of the network. WAE applications (described in the next section) are able to connect to WMD and gain access to a copy of this near real-time model in order to leverage WAE OPM API functionality.

For information on how to configure WMD, see Configure the WAE Modeling Daemon (WMD), on page 73.

WAE Applications

WAE provides a flexible and powerful application development infrastructure. A simple WAE application consists of:

- The application interface, defined in a YANG model. This interface usually includes RPCs and data models. The YANG models can, if necessary, extend the WAE network model, adding new data types.
- The application logic, implemented using the OPM API.



Because WAE automatically generates APIs from YANG definitions, a WAE application has its APIs automatically exposed. A WAE application is, in a sense, a seamless extension of WAE functionality.

Bandwidth on Demand Application

The Bandwidth on Demand (BWoD) application utilizes the near real-time model of the network offered by WMD to compute and maintain paths for SR policies with bandwidth constraints delegated to WAE from XTC. In order to compute the shortest path available for a SR policy with a bandwidth constraint and ensure that path will be free of congestion, a Path Computation Element (PCE) must be aware of traffic loading on the network. The WAE BWoD application extends the existing topology-aware PCE capabilities of XTC by allowing delegation of bandwidth-aware path computation of SR policies to be sub-delegated to WAE through a new XTC REST API. Users may fine-tune the behavior of the BWoD application, affecting the path it computes, through selection of application options including network utilization threshold (definition of congestion) and path optimization criteria preferences.

For information on how to configure the BWoD application, see Bandwidth on Demand Configuration Workflow, on page 107.

Bandwidth Optimization Application

The Bandwidth Optimization application is an approach to managing network traffic that focuses on deploying a small number of LSPs to achieve a specific outcome in the network. Examples of this type of tactical traffic engineering are deploying LSPs to shift traffic away from a congested link, establishing a low-latency LSP for priority voice or video traffic, or deploying LSPs to avoid certain nodes or links. WAE provides the Bandwidth Optimization application to react and manage traffic as the state of the network changes.

For information on how to configure the Bandwidth Optimization application, see Bandwidth Optimization Application Workflow, on page 112.

Cisco WAE Interfaces

Cisco WAE has three interfaces that you can use to configure your network model:

WAE UI

The WAE UI provides an easy-to-use interface that hides the complexity of creating a model building chain for a network. The WAE UI combines the configuration of multiple data collections under one network and can produce a single plan file that contains the consolidated data. However, there are certain operations that cannot be performed with the WAE UI. Any configurations done using the WAE Expert Mode or CLI may not appear in the WAE UI configuration screens. See Network Model Configuration—WAE UI, on page 9 and Important Notes, on page 10.

Expert Mode

The Expert Mode is a YANG model browser with additional device and service functionality that might not be available in the WAE UI. Users might prefer to use the Expert Mode over the WAE CLI because all options for each operation are visible in the Expert Mode. See Network Model Configuration—Expert Mode, on page 17.

WAE CLI

The WAE CLI is the interface in which the user responds to a visual prompt by typing a command; a system response is returned. It is the bare-bones interface for all WAE configurations. Operations available in the Expert Mode are also available in the CLI. See Network Model Configuration—WAE CLI, on page 25.

Network Model Creation Workflow

The following is a high-level workflow on how to configure individual network models. The detailed steps differ depending on what type of interface you use (Expert Mode, WAE UI, or WAE CLI).

If you plan to run multiple NIMOs and consolidate the information into one final network, do not run collections until after you have set up the aggregator NIMO. For more information, see NIMO Collection Consolidation, on page 51.

1. Configure device authgroups, SNMP groups, and network profile access.
2. (Optional) Configure agents. This step is required only for collecting XTC, LAG and port interface, or multilayer information.
3. Configure a network with a basic topology collection.
4. Run the collection.
5. Configure additional network collections.
6. (Optional) Schedule when to run collections.
7. Configure the archive file system location and interval at which plan files are periodically stored.
8. (Optional) View plan files in WAE applications.

CHAPTER 2

Network Model Configuration—WAE UI

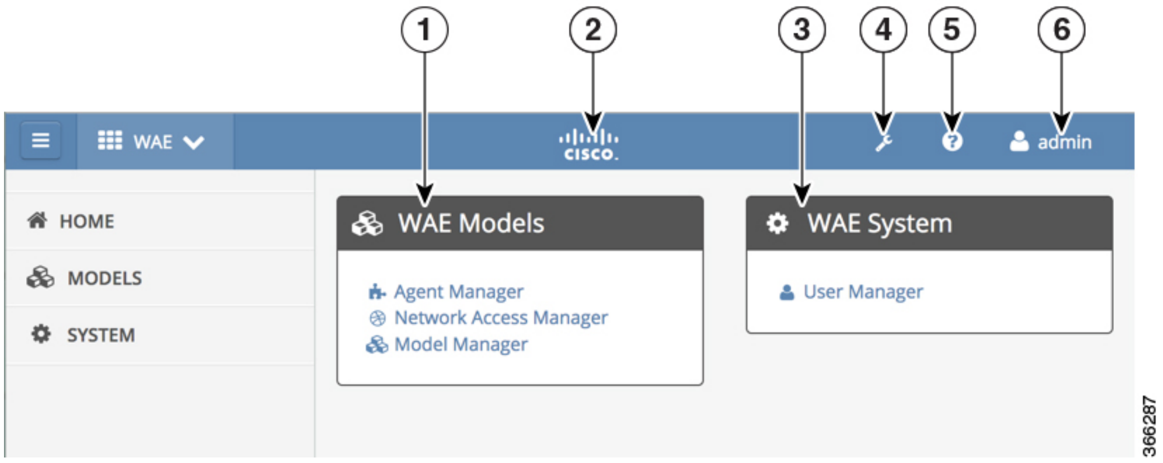
This section contains the following topics:

WAE UI Overview

The WAE UI provides an easy-to-use configuration tool for device and network access, network collection, user management, and XTC agents. The Model Manager hides the complexity of configuring network models with a wizard. The wizard takes you through the configuration of basic topology collection, advanced collection, and other NIMO capabilities against a single network.

For basic network model configuration we recommend starting with the Model Manager. There are certain operations (configuration of telemetry, parsing agent, some scheduler jobs, and LSP changes) that cannot be performed using the WAE UI. For these tasks, you can switch to the Expert Mode or the WAE CLI. Regardless of the interface you use, the last committed configuration is saved.

Figure 1: WAE UI



Callout No.	Name	Description
1	WAE Models	Contains network model configuration information: • Agent Manager—Configuration of XTC agents. • Network Access Manager—Configuration of device and network credentials. • Model Manager—Configuration of network models using NIMOs.
2	WAE Home (Cisco icon)	Takes you to the WAE Home page.
3	WAE System	Includes system and administrative tasks.
4	Expert Mode	Launches the Expert Mode in another window.
5	Help	Opens www.cisco.com/go/wae. You can find WAE documentation under the Support area.
6	User	Displays the current user logged into WAE. When clicked, allows the user to log out.

Important Notes

When using the WAE UI, note the following:

- If a network model or was created using the Expert Mode or CLI:
  - Do not add or remove NIMOs within that network model using the Model Manager.
  - Only edit existing NIMOs within that network model using the Model Manager.
- Model Manager splits the configuration of building network models into two phases. The first phase collects and consolidates different network topology sources (nodes, interfaces, LSP, VPN, BGP). This phase automatically produces a consolidated network model (aggregator NIMO) with the following naming convention: <*network_model_name*>_topology (for example, NetworkABC_topology). The second phase augments <*network_model_name*>_topology with additional data (traffic statistics, traffic demands, and visual layout). This phase automatically produces a final consolidated model (aggregator NIMO) with the following naming convention: <*network_model_name*> (for example, NetworkABC).
- If scheduling or archive is configured using the WAE UI, Model Manager automatically creates tasks in the Scheduler for each network model. The tasks have the following naming convention in the WAE CLI or Expert Mode:
  - _WAE_PLAN_ARCHIVE_<*network_model_name*>
  - _WAE_SCHEDULER_<*network_model_name*>
- Scheduler configuration done using the WAE Expert Mode or CLI will not appear in the WAE UI.

Configure a Network Model Using the WAE UI

This workflow describes the high-level steps to create a network model using the WAE UI.

If you are collecting information for a network using XTC, you must configure an XTC agent before configuring a network model.

Step	For more information, see...
1. Configure device credentials (network authgroups and SNMP groups).	Configure Network Access, on page 11
2. Create a network model name and configure plan archive settings.	Create a Network, on page 12
3. Configure basic topology collection.	Configure Basic Topology Collection, on page 12
4. Configure additional data collections.	Configure Additional NIMOs, on page 21
5. Run collection and view network details.	View Network Model Details, on page 14
6. (Optional) Schedule when to run collections.	Schedule Network Collections, on page 14
7. (Optional) Configure and view plan archives.	Configure the Archive and View Plan Files, on page 14

Configure Network Access

In this task, you are defining global device credentials by creating a network access profile.

Before you begin

Know the global network device credentials.

Step 1 From WAE Models, click **Network Access Manager**.

Step 2 Click **+ Add Network Access**.

Step 3 Enter the global device credentials:

- **Name**—Enter a name for the network access profile.
- **Login Type**—Choose which login protocol to use: SSH or Telnet. The SSH protocol is more secure. The Telnet protocol does not encrypt the username and password.
- **Username**
- **Password**
- **Enable Password**
- **SNMP Access Options**—Select either SNMPv2c or SNMPv3.

  If SNMPv2c, enter the SNMP RO community string that acts as a password. It is used to authenticate messages sent between the node and the seed router.

  If SNMPv3, enter the following default credentials:

  - **Security Level**—Select one of the following:

- **noAuthNoPriv**—Security level that does not provide authentication or encryption. This level is not supported for SNMPv3.
- **authNoPriv**—Security level that provides authentication but does not provide encryption.
- **authPriv**—Security level that provides both authentication and encryption.

- **Authentication Protocol and Password**—Select one of the following:
  - **md5**—HMAC-MD5-96 authentication protocol
  - **sha**—HMAC-SHA-96 authentication protocol
- **Encryption Protocol and Password**—The priv option offers a choice of DES or 128-bit AES encryption for SNMP security encryption. The priv option and the aes-128 token indicates that this privacy password is for generating a 128-bit AES key #.The AES priv password can have a minimum of eight characters. If the passphrases are specified in clear text, you can specify a maximum of 64 characters. If you use the localized key, you can specify a maximum of 130 characters.

Step 4 Click **Save**.

What to do next

Use the Model Manager to create a network model.

Create a Network

The initial step in configuring a complete network model is to create a new network with topology collection using the topo-igp-nimo or the topo-bgpls-xtc-nimo.

Step 1 From the Model Manager, click **Add New Network** and enter the following:

a) Network model name—This name cannot be changed after it is saved.
b) (Optional) Directory where you want WAE to store the plan files.
c) (Optional) How often to archive the plan files. We recommend starting with 15 minutes.

Step 2 Click **Save**.

What to do next

Configure global device credentials. See Configure Network Access, on page 11.

Configure Basic Topology Collection

In this task, you are configuring a topology collection that will be the source network for additional network collections. After the initial collection, the node IP address table is populated and you can add management IP addresses. For more information on basic topology collections, see Basic Topology Collection, on page 46.

Before you begin

- You must be in the **Model Manager** > **Network Discovery** tab for a particular network.
- If you are configuring topology collection on a network running XTC (topo-bgpls-xtc-nimo), an XTC agent must be configured and running before creating a network model using the Model Manager. See Configure an XTC Agent, on page 15.

Step 1 Click **Add Discovery**.

Step 2 Click one of the following NIMOs:

- **topo-igp-nimo**—To collect topology information using the IGP database. See IGP Topology Collection, on page 46.
- **topo-bgpls-xtc-nimo**—To collect topology information from a network running XTC. See BGP-LS Topology Collection Using XTC, on page 48.

Step 3 Click **Save**.

Step 4 Click **Run NIMO**. Note that the status is shown in the top-right corner of the window. Collection is complete when the status changes from "Running" to "Not Running."

Step 5 Click the table icon next to the NIMO you just configured. A table populated with node IP addresses appears.

Step 6 Add management IP addresses directly in the table and click **Save**.

Note You can click on the table anytime to see an updated node list.

What to do next

Configure additional collections.

Configure Additional Network Collections

This topic describes the general procedure to configure additional NIMOs. For NIMO descriptions, see NIMO Descriptions, on page 43.

Before you begin

- Confirm that a basic topology collection has been configured and collection is complete.
- You must be in the **Model Manager** > **Collection** tab for a particular network.

Step 1 From the Model Manager, click **Collection**.

Step 2 Click **Add NIMO**. All available NIMOs appear.

Step 3 Click a NIMO and enter the appropriate options. To view configuration options for a particular NIMO, see Network Interface Modules (NIMOs), on page 43.

Step 4 Click **Save**.

What to do next

You can configure more collections and add capabilities (continuous polling, demand mesh creation, and so on) using other NIMOs to create a complete network model.

View Network Model Details

To view node details, interfaces, and LSPs (if applicable) for a network, navigate to **Model Manager** > **View Model**. Consolidated NIMO collection data appears on this page.

Schedule Network Collections

This procedure describes how to schedule different network collections to run, and if applicable, at different times.

Before you begin

You must be in the **Model Manager** > **Schedule Collection** tab for a particular network.

Step 1 To schedule multiple network collections to run at different times, do the following:

a) Click **Add Task**.
b) Enter a task name and time interval to run the collection.
c) From the **Select Configured NIMOs** drop-down list, choose the collection to run.

Note Each collection is done in the order it is listed and configured.

Step 2 Click **Save**.

What to do next

View network model details and view plan files.

Configure the Archive and View Plan Files

After creating a network model and running collection, you have the option to retrieve and view plan files. Plan files capture all relevant information about a network at a given time, and can include topology, traffic, routing, and related information.

The Archive is a repository for plan files. See also Configure the Archive, on page 40, which describes how to configure the Archive using the WAE CLI.

Step 1 From the Model Manager, click the network model for which you want to view plan files.

Step 2 Click the **Archive** tab.

Step 3 The Archive path and interval fields might be populated if they were configured when the network was initially created. If not, or if you want to change them, enter new values.

Step 4 The Date/Time Range field displays the time range for which plan files are available. Click the link.

Step 5 Choose the time zone for the plan files you want to retrieve.

Step 6 Enter the beginning and end dates (including time) for the plan files and click **List Plan Files**. A list of plan files appears.

Step 7 Click the appropriate plan file, choose the plan file format to download, and click **Download**.

Configure XTC Agents Using the WAE UI

Agents perform information-gathering tasks and must be configured before certain network collection operations. This section describes how to configure XTC agents using the WAE UI. To configure the configuration parsing agent, see Configuring the Configuration Parsing Agent, on page 23.

Configure an XTC Agent

The XR Transport Controller (XTC) agent periodically collects information from XTC and keeps it as raw and normalized data. This data is consumed by different NIMOs to extract topology, LSPs, and so on. An agent must be configured for every XTC node in the network. XTC agents must be configured for any networks that use XTC before you can perform a network collection.

Before you begin

You must be in the **WAE UI** > **Agent Manager** window.

Step 1 Click **XTC**.

Step 2 Click **Add New Agent**.

Step 3 Enter the following information:

- XTC agent name.
- Host IP address of the XTC router.
- Port number to use for REST calls to the XTC host. The default is 8080.
- If you want to use HTTP basic authentication with defined credentials, choose **true** from the drop-down list.
- XTC credentials: username, password, and enable password.

Step 4 Click **Save**.

What to do next

Configure collections for networks that use XTC. For more information, see NIMO Descriptions, on page 43.

CHAPTER 3

Network Model Configuration—Expert Mode

This section contains the following topics:

Expert Mode Overview

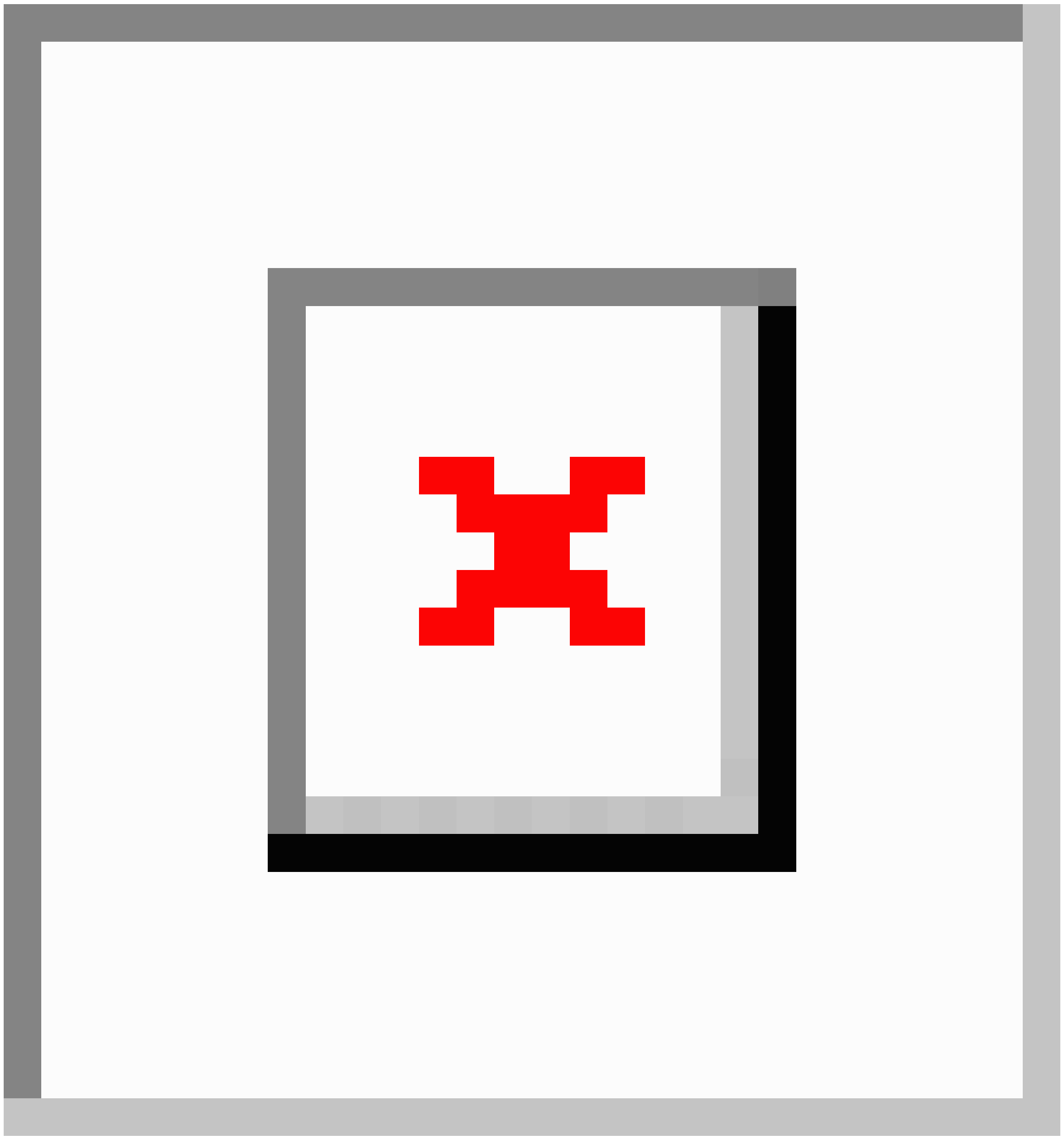
The Expert Mode is a YANG model browser with additional device and service functionality that might not be available in the WAE UI. You might also prefer to use the Expert Mode over the WAE CLI because all options for each operation are visible on the Expert Mode.

The Expert Mode is a mix of custom-built widgets and auto-rendering from the underlying device, service, and network models. The Expert Mode is immediately updated when new devices, NIMOs, or network models are added to the system.

In the top-right corner of the WAE UI (https://server-ip:8443), click the tool icon to access the Expert Mode:

The purpose of this section is to illustrate the Expert Mode and go over procedures to get you up and running. This section does not go into advanced configurations. It is assumed that once you understand the basic procedures, you can configure more complex operations.

Figure 2: Expert Mode Interface



Callout No.	Name	Description
1	wae:wae	Navigate to this path to configure global settings and agents.
2	wae:networks	Navigate to this path to configure network settings and NIMOs.
3	Root directory	Click this icon to access the wae:networks and wae:wae paths.
4	Breadcrumb	Indicates where in the directory structure you are.
5	Commit button	Click this button to commit any configuration changes.

Navigation and Commit

When an object type (for example, a network instance) is selected, a list of all related object instances is shown. When performing network configuration operations, click the **Commit** button to save the changes. For more information on Commit functionality, see Commit Flags, on page 147.

Configuring a Network Model Using the Expert Mode

This workflow describes the high-level configuration steps on how to create a network model using the Expert Mode.

Step	For more information, see...
1. Configure device authgroups and SNMP groups.	Configure Device Access Using the Expert Mode, on page 19
2. Configure a network access profile.	Configure Network Access, on page 20
3. Configure agents. **Note** This step is only required for collecting XTC or multi-layer information.	• Configuring XTC Agents Using the Expert Mode, on page 22 • Configuring the Configuration Parsing Agent, on page 23
4. Create a network and collect basic topology data. **Note** If you plan to consolidate network models and collect more than basic topology (for example, merge topo-bgpls-xtc-nimo and lsp-pcep-xtc-nimo information into one final network model), configure DARE and create networks where collections have not yet been executed. For more information, see NIMO Collection Consolidation, on page 51.	Create a Network Model, on page 21
5. Configure additional data collections.	Configure Additional NIMOs, on page 21
6. (Optional) Configure the Scheduler.	Scheduler Configuration, on page 115
7. Configure and view plan archives.	• From the WAE UI—Configure the Archive and View Plan Files, on page 14 • From the WAE CLI—Configure the Archive, on page 40

Configure Device Access Using the Expert Mode

Cisco WAE uses authgroups for login and SNMP access to the devices. The following procedure describes how to use the Expert Mode to configure authgroups and network access.

Step 1 From the Expert Mode, set up authgroups.

a) Navigate to **/ncs:devices** and click the **authgroups** tab.
b) Click **group**.
c) Click the plus (+) sign, enter an authgroup name, and click **Add**.
d) Click **default-map** and enter the default authentication parameters. For example, choose **remote-name** from the drop-down list, check the **default-map** check box, and enter the remote name string credentials.

Note If not visible, scroll down to view and enter the remote secondary password.

Step 2 Set up SNMP groups.

a) Navigate back to **/ncs:devices/authgroups** and click the **snmp-group** tab.
b) Click the plus (+) sign and enter an SNMP group name.
c) Check **default-map** and enter the default SNMP credentials. For example, choose **community-name** from the drop-down list, check the **default-map** check box, and enter the community string credentials.
d) If configuring SNMPv3, click the **usm** tab and enter in the applicable User-based Security Model (USM) values (remote user, security level, authentication, and privacy protocols). For more information on USM values, see the SNMPv3 options explained in the following WAE UI procedural topic: Configure Network Access, on page 11.
e) Click the **Commit** button.

What to do next

Create a network access profile. See Configure Network Access, on page 20.

Configure Network Access

Step 1 Navigate to **/wae:wae** and click the **nimos** tab.

Step 2 Click **network-access**.

Step 3 Click the plus (+) sign and enter a network access name.

Step 4 Select and enter the appropriate network access details.

The default authgroups and SNMP groups you configured earlier are available in the drop-down lists. For more information, see Configure Device Access Using the Expert Mode, on page 19.

Step 5 Click the **node-access** tab to enter the management IP address of the router.

a) Click the plus (+) sign, enter the IP address, and click **Add**.
b) Enter the associated management IP.

Repeat these steps as necessary for all management IPs.

Step 6 Click the **Commit** button.

What to do next

After completion of this task, you can create a network and run basic data collection.

Create a Network Model

When creating a network you must also configure basic topology collection using the topo-igp-nimo or the topo-bgpls-xtc-nimo. The following procedure describes the first configuration step using the Expert Mode.

Before you begin

- Confirm that device access and network access are configured. For more information, see Configure Device Access Using the Expert Mode, on page 19 and Configure Network Access, on page 20.
- If creating a network running XTC, confirm that XTC agents have been configured. For more information, see Configuring XTC Agents Using the Expert Mode, on page 22.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose one of the following:

- **topo-igp-nimo**—Collect topology information using the IGP database. For more information on options, see IGP Topology Collection, on page 46 and IGP Topology Advanced Options, on page 47.
- **topo-bgpls-xtc-nimo**—Collect topology information from a network running XTC. This NIMO requires a configured agent. For more information, see BGP-LS Topology Collection Using XTC, on page 48 and Configuring XTC Agents Using the Expert Mode, on page 22.

Step 6 Click the corresponding link. For example, if you chose topo-igp-nimo, click the **topo-igp-nimo** link and enter the applicable parameters.

Step 7 Click the **Commit** button. This network model can now be used as the source network for additional network collections.

What to do next

Configure additional network collections using this network model as the source network. For more information, see NIMO Descriptions, on page 43.

Configure Additional NIMOs

This topic only describes the general steps on configuring different types of advanced network data collection. NIMOs are used to collect different types of data. Some NIMOs require the configuration of agents. For more information, see NIMO Descriptions, on page 43.

Before you begin

You must have a network model with basic collection to be used as a source network. For more information, see Create a Network Model, on page 21.

Step 1 From the Expert Mode, navigate to **/wae:networks/network/*network_name***.

Step 2 Click the **nimo** tab.

Step 3 From the **Choice - nimo-type** drop-down list, select a NIMO to configure.

Step 4 Enter the appropriate parameters for the selected NIMO.

Step 5 Click the **Commit** button.

Step 6 Click **run-collection** > **Invoke run-collection**.

Configure Agents Using the Expert Mode

Agents perform information-gathering tasks and must be configured before certain network collection operations. This section describes how to configure these agents using the Expert Mode.

Configuring XTC Agents Using the Expert Mode

The XR Transport Controller (XTC) agent periodically collects information from XTC and keeps it as raw and normalized data. The agent is used to connect to the REST interface of XTC and retrieve the PCE topology. This data is consumed by different applications (Bandwidth on Demand) and NIMOs (topo-bgpls-xtc-nimo and lsp-pcep-xtc-nimo) to extract topology, LSPs, and so on. An agent must be configured for every XTC node in the network. XTC agents must be configured for any networks that use XTC before you can perform a network collection.

Step 1 From the Expert Mode, navigate to **wae:wae** and click the **agents** tab.

Step 2 Click **xtc**.

Step 3 Click the plus (+) icon to add agents.

Step 4 Enter the following information:

- XTC agent name.
- **xtc-host-ip**—Host IP address of the XTC router.
- **xtc-rest-port**—Port number to use for REST calls to the XTC host. The default is 8080.
- **use-auth**—To use HTTP basic authentication with defined credentials, choose **true** from the drop-down list.
- **auth-group**—XTC credentials that were defined in Configure Device Access Using the Expert Mode, on page 19.
- **batch-size**—Number of nodes to send in each message. Default is 1000.
- **keep-alive**—Interval in seconds to send keep-alive messages. Default is 10.
- **max-lsp-history**—Number of LSP entries to send. Default is 0.
- **enabled**—Enables the XTC agent. Default is true.

As long as the **enabled** option is set to true, the XTC agent starts right away after configuration or when WAE starts. In the same respect, the XTC agent stops when the configuration is removed, if WAE has stopped, or the enabled option is set to false.

Step 5 Click **Commit**.

Step 6 Repeat these steps for all XTC nodes.

Step 7 To view the raw data, navigate back to **/wae:wae/agents/xtc-agent:xtc/xtc/<*agent-name*>** and click the **pce** tab.

Step 8 Click the appropriate data containers (topology-nodes, tunnel-detail-infos, and xtc-topology-objects) to view the raw data.

To confirm data was collected successfully, navigate to **/wae:wae/agents/xtc-agent:xtc/xtc/<*agent-name*>** and click the **status** tab. You can view the time stamp of the last successful collection.

What to do next

Configure collections for networks that use XTC. For more information, see NIMO Descriptions, on page 43.

Configuring the Configuration Parsing Agent

The Configuration Parsing agent can collect (run-config-get) and parse data (run-config-parse) from Cisco, Juniper, and Huawei router configuration files. This agent must be configured before configuring the port-cfg-parse-nimo and optimal-nimo, which are used for multi-layer collection.

The agent can retrieve the configuration by determining the router type/vendor and parsing the configuration. After parsing this information, the tool matches corresponding interfaces in the IGP mesh to create the network topology. For information on what types of router configurations the agent can read, see Router Configuration Information, on page 24.

Step 1 From the Expert Mode, navigate to **wae:wae** and click the **agents** tab.

Step 2 Click **cfg-parse**.

Step 3 Click the plus (+) icon to add agents and enter a Configuration Parsing agent name. This can be any arbitrary name.

Step 4 If configurations are already being saved for your network, enter a directory where the configurations are stored; for example, `/home/user1/wae/etc/configs/gc_out`. Alternatively, enter the path where configurations are saved if you are using the agent to get configurations.

Step 5 If you want to retrieve configurations, click the **get** tab and choose the source network and network access. If you only want to parse existing configurations, skip to Step 8.

Step 6 Click **Commit**.

Step 7 Navigate back to the **cfg-parse** tab and click **run-config** > **Invoke run-config-parse**.

Step 8 Click the **parse** tab.

Step 9 Enter the following information:

- **igp-protocol**—Choose which interfaces are part of the topology: IS-IS and/or OSPF-enabled interfaces. The default is ISIS.
- **ospf-area**—The agent can read information for single or multiple areas. The -ospf-area option specifies the area ID or all. The default is area 0.
- **isis level**—The agent can read IS-IS Level 1, Level 2, or both Level 1 and Level 2 metrics. If both are selected, the agent combines both levels into a single network. Level 2 metrics take precedence.
- **asn**—ASN is ignored by default. However, for networks that span multiple BGP ASNs, use this option to read information from more than one IGP process ID or instance ID in an ASN.

Step 10 Click **include-object** to add collection types.

Step 11 Click the plus (+) icon and choose a collection type from the drop-down list. At minimum, you must add base, lag, and lmp.

Step 12 Click **Commit**.

Step 13 Navigate back to the **cfg-parse** tab and click **run-config-parse** > **Invoke run-config-parse**.

Step 14 To verify that the agent is running successfully, click the **model** tab > **nodes**. The list of nodes appears.

What to do next

Configure port-cfg-parse-nimo and optical-nimo for multi-layer collection. For more information, see NIMO Descriptions, on page 43.

Router Configuration Information

The following router configuration information can be read by the Configuration Parse agent:

• Router name • Router IP address (loopback) • Interface names (inside IGP topology) • Interface IP addresses • Interface capacities (if available)	• IGP type and metrics (IS-IS or OSPF) • RSVP reservable bandwidth (MPLS) • LAG ports (for Ethernet) and bundle ports (for different link types)

CHAPTER 4

Network Model Configuration—WAE CLI

This section contains the following topics:

WAE CLI Overview

WAE provides a network CLI that is automatically rendered using the data models described by the WAE YANG files. The CLI contains commands for manipulating the network configuration. The CLI is entirely data-model driven. The YANG model(s) define a hierarchy of configuration elements; the CLI follows this tree. The CLI provides various commands for configuring hardware and network connectivity of managed devices.

The CLI supports two modes: *operational mode*, for monitoring the state of WAE nodes; and *configure mode*, for changing the state of the network. The prompt indicates which mode the CLI is in. When moving from operational mode to configure mode using the **configure** command, the prompt is changed from **user@wae#** to **user@wae(config)#**. The prompts can be configured using the c-prompt1 and c-prompt2 settings in the wae.conf file.

For example:

```
admin@wae# configure
Entering configuration mode terminal
admin@wae(config)#
```

Operational Mode

Operational mode is the initial mode after successful login to the CLI. It is primarily used for viewing the system status, controlling the CLI environment, monitoring and troubleshooting network connectivity, and initiating the configure mode.

The following commands are the base commands available in operational mode. Additional commands are rendered from the loaded YANG files.

Invoke an action:

```
<path> <parameters>
```

Invokes the action found at *path* using the supplied *parameters*. This command is auto-generated from the YANG file. For example, given the following action specification in a YANG file:

```
tailf:action shutdown {
  tailf:actionpoint actions;
  input {
    tailf:constant-leaf flags {
      type uint64 {
        range "1 .. max";
      }
      tailf:constant-value 42;
    }
    leaf timeout {
      type xs:duration;
      default PT60S;
    }
    leaf message {
      type string;
    }
    container options {
      leaf rebootAfterShutdown {
        type boolean;
        default false;
    }
    leaf forceFsckAfterReboot {
      type boolean;
      default false;
    }
    leaf powerOffAfterShutdown {
      type boolean;
      default true;
    }
  }
 }
}
```

The action can be invoked in the following way:

```
user@wae> shutdown timeout 10s message reboot options { \
forceFsckAfterReboot true }
```

Built-in Operational Mode Commands

Command	Description
`commit (abort|confirm)`	Aborts or confirms a pending confirming commit. A pending confirming commit is aborted if the CLI session is terminated without doing **`commit confirm`**. The default is **`confirm`**. Example: `user@wae# commit abort`

`config (exclusive|terminal) [no-confirm]`	Enters configure mode. The default is **`terminal`**. • terminal—Edits a private copy of the running configuration; no lock is taken. • no-confirm—Enters configure mode, ignoring any confirm dialog. Example: `user@wae#` **`config terminal`** `Entering configuration mode terminal`

`file list` *<directory>*	Lists files in a directory. Example:
	`user@wae#` **`file list /config`**
	`rollback10001`
	`rollback10002`
	`rollback10003`
	`rollback10004`
	`rollback10005`

`file show` *<file>*	Displays contents of a file. Example:
	`user@wae#` **`file show /etc/skel/.bash_profile`**
	`# /etc/skel/.bash_profile`
	`# This file is sourced by bash for login shells. The following line`
	`# runs our .bashrc and is recommended by the bash info pages.`
	`[[ -f ~/.bashrc ]] && . ~/.bashrc`

`help` *<command>*	Displays help text for a command. Example:
	`user@wae#` **`help job`**
	`Help for command: job`
	`   Job operations`

`job stop` *<job id>*	Stops a specific background job. In the default CLI, the only command that creates background jobs is **`monitor start`**. Example:
	`user@wae#` **`monitor start /var/log/messages`**
	`[ok][...]`
	`admin@ncs# show jobs`
	`JOB COMMAND`
	`3 monitor start /var/log/messages`
	`admin@ncs# job stop 3`
	`admin@ncs# show jobs`
	`JOB COMMAND`

`logout session` *<session ID>*	Logs out a specific user session from WAE. If the user holds the **`configure exclusive`** lock, the lock is released. Example:
	`user@wae#` **`who`**
	`Session User Context From Proto Date Mode`
	`25 oper cli 192.0.2.254 ssh 12:10:40 operational`
	`*24 admin cli 192.0.2.254 ssh 12:05:50 operational`
	`user@wae#` **`logout session 25`**
	`user@wae#` **`who`**
	`Session User Context From Proto Date Mode`
	`*24 admin cli 192.0.2.254 ssh 12:05:50 operational`

`logout user` *<username>*	Logs out a specific user from WAE. If the user holds the **`configure exclusive`** lock, the lock is released. Example:
	`user@wae# who`
	`Session User Context From Proto Date Mode`
	`25 oper cli 192.0.2.254 ssh 12:10:40 operational`
	`*24 admin cli 192.0.2.254 ssh 12:05:50 operational`
	`user@wae# logout user oper`
	`user@wae# who`
	`Session User Context From Proto Date Mode`
	`*24 admin cli 192.0.2.254 ssh 12:05:50 operational`

`script reload`	Reloads scripts found in the scripts/command directory. New scripts are added. If a script file has been removed, the corresponding CLI command is purged.

`send (all` | *<user>***`)`** *<message>*	Displays a message on the screens of all users who are logged in to the device or on a specific screen.
	• all—Display the message to all currently logged in users.
	• *<user>*—Display the message to a specific user.
	Example:
	`user@wae# send oper "I will reboot system in 5 minutes."`
	The oper user sees the following message onscreen:
	`oper@wae# Message from user@wae at 13:16:41...`
	`I will reboot system in 5 minutes.`
	`EOF`

`show cli`	Displays CLI properties. Example:
	`user@wae# show cli`
	`autowizard false`
	`complete-on-space true`
	`display-level 99999999`
	`history 100`
	`idle-timeout 1800`
	`ignore-leading-space false`
	`output-file terminal`
	`paginate true`
	`prompt1 \h\M#`
	`prompt2 \h(\m)#`
	`screen-length 71`
	`screen-width 80`
	`service prompt config true`
	`show-defaults false`
	`terminal xterm-256color`
	`timestamp disable`

`show history` [<*limit*>]	Displays CLI command history. By default the last 100 commands are listed. The size of the history list is configured using the history CLI setting. If a history limit is specified, only the last commands up to that limit are shown. Example:
	
	`user@wae# `**`show history`**
	`06-19 14:34:02 -- ping router`
	`06-20 14:42:35 -- show running-config`
	`06-20 14:42:37 -- who`
	`06-20 14:42:40 -- show history`
	`user@wae# `**`show history 3`**
	`14:42:37 -- who`
	`14:42:40 -- show history`
	`14:42:46 -- show history 3`

`show jobs`	Displays jobs that are currently running in the background. Example:
	
	`user@wae# `**`show jobs`**
	`JOB COMMAND`
	`3 monitor start /var/log/messages`

`show log` <*file*>	Displays contents of a log file. Example: `user@wae# `**`show log messages`**
`show parser dump` <*command prefix*>	Shows all possible commands that start with the specified command prefix.

`show running-config` [<*path filter*> [**`sort-by`** <*idx*>]]	Displays the current configuration. By default the entire configuration is displayed. You can limit what is shown by supplying a path filter. The path filter can be either a path pointing to a specific instance, or if an instance ID is omitted, the part following the omitted instance is treated as a filter.
	
	The **`sort-by`** argument can be used when the path filter points to a list element with secondary indexes. The name of a secondary index is *idx*. When used, the table is sorted in the order defined by the secondary index. This lets you control the order in which to display instances.
	
	For example, to show the aaa settings for the admin user:
	
	`user@wae# `**`show running-config aaa authentication users user admin`**
	`aaa authentication users user admin`
	`  uid 1000`
	`  gid 1000`
	`  password $1$JA.1O3Tx$Zt1ycpnMlg1bVMqM/zSZ7/`
	`  ssh_keydir /var/ncs/homes/admin/.ssh`
	`  homedir /var/ncs/homes/admin`
	`!`
	
	To show all users who have group ID 1000, omit the user ID and instead specify gid 1000:
	
	`user@wae# `**`show running-config aaa authentication users user * gid 1000`**
	`...`

show *<path>* [**sort-by** *<idx>*]	Shows the configuration as a table provided that path leads to a list element and the data can be rendered as a table (that is, the table fits on the screen). You can also force table formatting of a list by using the | **tab** pipe command.
	
	The **sort-by** argument can be used when the path points to a list element with secondary indexes. The name of a secondary index is *idx*. When used, the table is sorted in the order defined by the secondary index. This lets you control the order in which to display instances. Example:
	
	`user@wae# show devices device-module`
	`NAME REVISION URI DEVICES`
	`--------------------------------------------------------------------------------`
	`junos - http://xml.juniper.net/xnm/1.1/xnm [ pe2 ]`
	`tailf-ned-cisco-ios - urn:ios [ ce1 ce0 ]`
	`tailf-ned-cisco-ios-stats - urn:ios-stats [ ce1 ce0 ]`
	`tailf-ned-cisco-ios-xr - http://tail-f.com/ned/cisco-ios-xr [ p1 p0 ]`

source *<file>*	Runs commands from a specified file as if they had been entered by the user. The autowizard is disabled when executing commands from the file.

timecmd *<command>*	Measures and displays the execution time of a command. Note that **timecmd** is only available if devtools has been set to true in the CLI session settings. Example:
	
	`user@wae# timecmd id`
	`user = admin(501), gid=20, groups=admin, gids=12,20,33,61,79,80,81,98,100`
	`Command executed in 0.00 sec`
	`user@wae#`

who	Displays currently logged on users. The current session—the session running the **show status** command—is marked with an asterisk. Example:
	
	`user@wae# who`
	`Session User Context From Proto Date Mode`
	`25 oper cli 192.0.2.254 ssh 12:10:40 operational`
	`*24 admin cli 192.0.2.254 ssh 12:05:50 operational`
	`admin@ncs#`

Configure Mode

Configure mode can be initiated by entering the configure command in operational mode. All changes to the network configuration are done to a copy of the active configuration. These changes do not take effect until a successful commit or commit confirm command is entered.

The following commands are the base commands available in configure mode. Additional commands are rendered from the loaded YANG files.

Configure a value:

```
<path> [<value>]
```

Set a parameter. If a new identifier is created and autowizard is enabled, the CLI prompts the user for all mandatory sub-elements of that identifier. This command is auto-generated from the YANG file.

If no *<value>* is provided, the CLI prompts the user for the value. No echo of the entered value occurs if *<path>* is an encrypted value of the type MD5DigestString, DESDigestString, DES3CBCEncryptedString, or AESCFB128EncryptedString as documented in the tailf-common.yang data-model.

Built-in Configure Mode Commands

Command	Description
annotate *<statement> <text>*	Associates an annotation with a given configuration. To remove an annotation, leave the text empty. This command is only available when the system has been configured with attributes enabled.
commit (**check** | **and-quit** | **confirmed** |**to-startup**) [**comment** *<text>*] [**label** *<text>*]	Commits the current configuration to running. • check—Validates the current configuration. • and-quit—Commits to running and quits configure mode. • comment *<text>*—Associates a comment with the commit. The comment is visible when examining rollback files. • label <text>—Associates a label with the commit. The label is visible when examining rollback files. **Note** A useful command is `commit dry-run`. This command validates and displays the configuration changes, but does not perform the actual commit. For more available `commit` commands, see Commit Flags, on page 147.
copy *<instance path> <new id>*	Makes a copy of an instance.

copy cfg [**merge** | **overwrite**] *<src path>* **to** *<dest path>*	Copies data from one configuration tree to another. Only data that makes sense at the destination is copied. No error message is generated for data that cannot be copied and the operation can fail completely without any error messages being generated. For example, to create a template from a part of a device config, first configure the device and then copy the config to the template configuration tree. Example:
	`user@wae(config)# devices template host_temp`
	`user@wae(config-template-host_temp)# exit`
	`user@wae(config)# copy cfg merge devices device ce0 config \`
	`  ios:ethernet to devices template host_temp config ios:ethernet`
	`user@wae(config)# show configuration diff`
	`+devices template host_temp`
	`+ config`
	`+ ios:ethernet cfm global`
	`+ !`
	`+!`

copy compare *<src path>* **to** *<dest path>*	Compares two arbitrary configuration trees. Items that appear only in the source tree are ignored.
delete *<path>*	Deletes a data element.
do *<command>*	Runs the command in operational mode.

`edit` *<path>*	Edits a sub-element. Missing elements in the path are created.
`exit` (**`level`** | **`configuration-mode`**)	• level—Exits from this level. If performed at the top level, exits configure mode. This is the default if no option is given. • configuration mode—Exits from configuration mode regardless of the edit level.
`help` *<command>*	Shows help text for the command.
`hide` *<hide-group>*	Rehides the elements and actions that belong to the hide groups. No password is required for hiding. This command is hidden and is not shown during command completion.
`insert` *<path>*	Inserts a new element. If the element already exists and has the indexedView option set in the data model, the old element is renamed as element+1 and the new element is inserted in its place.
`insert` *<path>*[**`first`** | **`last`** | **`before`** *<key>* | **`after`** *<key>*]	Injects a new element into an ordered list. The element can be added first, last (the default), before, or after another element.

`load` (**`merge`** | **`override`**) (**`terminal`** | *<file>*)	Loads the configuration from a file or terminal.
	• merge—Merges the content of the file or terminal with the current configuration.
	• override—Replaces the current configuration with the configuration from the file or terminal.
	For example, with the following current configuration:
	`devices device p1`
	`  config`
	`   cisco-ios-xr:interface GigabitEthernet 0/0/0/0`
	`    shutdown`
	`   exit`
	`   cisco-ios-xr:interface GigabitEthernet 0/0/0/1`
	`    shutdown`
	`  !`
	`!`
	The **shutdown** value for the entry `GigabitEthernet 0/0/0/0` is deleted. Because the configuration file is just a sequence of commands with comments in between, the configuration file looks like this:
	`devices device p1`
	`  config`
	`   cisco-ios-xr:interface GigabitEthernet 0/0/0/0`
	`    no shutdown`
	`   exit`
	`  !`
	`!`
	The file can then be used with the command **load merge** *FILENAME* to achieve the desired results.

`move` *<path>*[**`first`** | **`last`** | **`before`** *<key>* | **`after`** *<key>*]	Moves an existing element to a new position in an ordered list. The element can be moved first, last (the default), before, or after another element.
`rename` *<instance path> <new id>*	Renames an instance.

`revert`	Copies the running configuration to the current configuration and removes all uncommitted changes.
`rload`(**`merge`**|**`override`**) (**`terminal`** | <*file*>)	Loads the file relative to the current submode. For example, if a file has a device config, you can enter one device and issue the **rload merge**/**override** <*file*> command to load the config for that device, then enter another device and load the same config file using **rload**. See also the **load** command. • merge—Merges the content of the file or terminal with the current configuration. • override—Replaces the current configuration with the configuration from the file or terminal.
`rollback configuration` [<*number*>] [<*path*>]	Returns the configuration to a previously committed configuration. You can configure the number of old configurations to store in the wae.conf file. If the configurations to store exceed the threshold, the oldest configuration is removed before creating a new one. The configuration changes are stored in rollback files, where the most recent changes are stored in the file rollbackN with the highest number N. Only the deltas are stored in the rollback files. When rolling back the configuration to rollback N, all changes stored in rollback10001-rollbackN are applied. The optional path argument allows subtrees to be rolled back while the rest of the configuration tree remains unchanged. This command is available only if rollback has been enabled in wae.conf. Example: `user@wae(config)# ` **`rollback configuration 10005`**
`rollback selective` [<*number*>] [<*path*>]	Instead of undoing all changes from rollback10001 to rollbackN, you can undo only the changes stored in a specific rollback file. In some cases applying the rollback file might fail, or the configuration might require additional changes in order to be valid. The optional path argument allows subtrees to be rolled back while the rest of the configuration tree remains unchanged.
`show full-configuration` [<*pathfilter*> [**`sort-by`** <*idx*>]]	Shows the current configuration, taking local changes into account. The show command can be limited to a part of the configuration by providing a path filter. The **`sort-by`** argument can be given when the path filter points to a list element with secondary indexes. The name of a secondary index is *idx*. When used, the table is sorted in the order defined by the secondary index, which lets you control the order in which to display instances.
`show configuration` [<*pathfilter*>]	Shows current edits to the configuration.
`show configuration merge` [<*pathfilter*> [**`sort-by`** <*idx*>]]	Shows the current configuration, taking local changes into account. The show command can be limited to a part of the configuration by providing a path filter. The **`sort-by`** argument can be given when the path filter points to a list element with secondary indexes. The name of a secondary index is *idx*. When used, the table is sorted in the order defined by the secondary index, which lets you control the order in which to display instances.
`show configuration commit changes` [<*number*> [<*path*>]]	Displays edits associated with a commit, identified by the rollback number created for the commit. The changes are displayed as forward changes, as opposed to **show configuration rollback changes**, which displays the commands for undoing the changes. The optional path argument allows only edits related to a given subtree to be listed.
`show configuration commit list` [<*path*>]	Lists rollback files. The optional path argument allows only rollback files related to a given subtree to be listed.

`show configuration rollback listed` [<*number*>]	Displays the operations required to undo the changes performed in a commit associated with a rollback file. These are the changes that are applied if the configuration is rolled back to that rollback number.
`show configuration running` [<*pathfilter*>]	Displays the running configuration without taking uncommitted changes into account. An optional path filter can be provided to limit what is displayed.
`show configuration diff` [<*pathfilter*>]	Displays uncommitted changes to the running configuration in diff-style, with + and - in front of added and deleted configuration lines.
`show parser dump` <*command prefix*>	Shows all possible commands that start with the command prefix.
`tag add` <*statement*> <*tag*>	Adds a tag to a configuration statement. This command is available only when the system is configured with attributes enabled.
`tag del` <*statement*> <*tag*>	Removes a tag from a configuration statement. This command is available only when the system is configured with attributes enabled.
`tag clear` <*statement*>	Removes all tags from a configuration statement. This command is available only when the system is configured with attributes enabled.

`timecmd` <*command*>	Measures and displays the execution time of a command. This command is available only if devtools has been set to true in the CLI session settings. Example:
	`user@wae# timecmd id`
	`user = admin(501), gid=20, groups=admin, gids=12,20,33,61,79,80,81,98,100`
	`Command executed in 0.00 sec`
	`user@wae#`

`top` [<*command*>]	Exits to the top level of configuration, or executes a command at the top level of the configuration.
`unhide` <*hide-group*>	Unhides all elements and actions that belong to the hide-group. A password might be required. This command is hidden and is not shown during command completion.
`validate`	Validates the current configuration. This is the same operation as **commit check**.
`xpath` [**`ctx`** <*path*>] (**`eval`** | **`must`** | **`when`**) <*expression*>	Evaluates an XPath expression. A context-path can be used as the current context for the evaluation of the expression. If no context-path is given, the current sub-mode is used as the context-path. The pipe command trace can be used to display debug or trace information. This command is available only if devtools has been set to true in the CLI session settings. • eval—Evaluates an XPath expression. • must—Evaluates the expression as a YANG *must* expression. • when—Evaluate the expression as a YANG *when* expression.

Expert Mode and WAE CLI Comparison

Although this guide describes many configurations using the Expert Mode, it is important to note that you can use the Expert Mode and CLI interfaces interchangeably. The advantage of using the Expert Mode, other than the GUI, is that it displays all available fields for configuration. In the CLI, you must know the parameters or view the CLI command help to see all available options.

Configuration in the CLI follows the same path structure as navigating in the Expert Mode. The following table lists some equivalent CLI commands and Expert Mode configuration with sample data.

Configuration Type	Expert Mode	CLI Equivalent

Create a device authgroup with device credentials.	1. Navigate to **/ncs:devices** and click the **authgroups** tab.
	2. Click **group**.
	3. Click the plus (+) sign, enter **`groupABC`** as the authgroup name, and click **Add**.
	4. Click **default-map** and enter following authentication parameters: **remote-name**—**`rpc1`**, **remote-password**—**`XLydrf`**, **remote-secondary-password**—**`XLydrr`**.
		`# set devices authgroups group` **`groupABC`** `default-map`
		`remote-name` **`rpc1`** `remote-password`
		`XLydrf`
		`remote-secondary-password` **`XLydrr`**

Create a network model by discovering the network using XTC (topo-bgpls-xtc-nimo).
 	
Note This example assumes that the network access and an XTC agent have been configured and are running.
	1. Navigate to **/wae:networks**, click the plus (+) sign, and enter **`as54001_topo`**.
	2. Click **Add**.
	3. Click the **nimo** tab and choose **topo-bgpls-xtc-nimo** as the NIMO type.
	4. Enter the following:
		`# set networks network` **`as54001_topo`**
		`nimo topo-bgpls-xtc-nimo`
		`network-access` **`as54001`** `xtc-host` **`xtc11`**
		
		`backup-xtc-host` **`xtc12`** `igp-protocol`
		`isis`
		`extended-topology-discovery` **`true`** `asn`
		`54001`

Field	User Input
network-access	**`as54001`**
xtc-host	**`xt11`**
backup-xtc-host	**`xt12`**
asn	**`54001`**
igp-protocol	**`isis`**
extended-topology-discovery	**`true`**

Configuration Type	**Expert Mode**	**CLI Equivalent**

Consolidate network models.	**1.** Navigate to **/wae:networks**, click the plus (+) sign, and enter and **`as54001`**.
	2. Click **Add**.
	3. Click the **nimo** tab and choose **aggregator**.
	4. Click **aggregator** > plus (+) sign, and choose the source NIMOs: **as54001_topo**, **as54001_xtclsp**, **as54001_conflsp**, and **as54001_snmplsp**.
		`# set networks network as54001 nimo aggregator sources as54001_topo`
		`# set networks network as54001 nimo aggregator sources as54001_xtclsp`
		`# set networks network as54001 nimo aggregator sources as54001_conflsp`
		`# set networks network as54001 nimo aggregator sources as54001_snmplsp`

Configure a Network Model Using the WAE CLI

This workflow describes the high-level configuration steps on how to create a network model using the Expert Mode.

Step	**For more information, see...**
1. Configure device authgroups and SNMP groups.	Configure Device Access Using the CLI, on page 36
2. Configure a network access profile.	Configure a Network Access Profile, on page 38
3. Configure agents. **Note** This step is only required for collecting XTC or multi-layer information.	• Configuring XTC Agents Using the Expert Mode, on page 22 • Configuring the Configuration Parsing Agent, on page 23
4. Create a network and collect basic topology data.	Create a Network Model, on page 38
5. Configure additional data collections or NIMO capabilities.	Configure Additional NIMOs, on page 40
6. (Optional) Configure the Scheduler.	Scheduler Configuration, on page 115
7. Configure and view plan archives.	Configure the Archive, on page 40

Configure Device Access Using the CLI

WAE uses `authgroups` for login and SNMP access to devices. The following procedure describes how to configure device access using the CLI in configuration mode.

Step 1 Create device authgroup(s) with device credentials.

```
# set devices authgroups group <group_name>
# default-map remote-name <username>
# default-map remote-password <user_password>
# default-map remote-secondary-password <secondary_password>
# commit
```

Step 2 Create SNMP group(s) to be able to run SNMP tools.

```
# set devices authgroups snmp-group <group_name>
# default-map community-name <community_name>
# commit
```

For SNMPv3, you can set the following options:

```
# set devices authgroups snmp-group <group_name>
# default-map community-name <community_name>
# default-map usm remote-name <remote_user>
# default-map usm security-level <auth-priv or auth-no-priv or no-auth-no-priv>
# default-map usm auth <auth_protocol> remote-password <remote_password>
# default-map usm priv <priv_protocol> remote-password <remote_password>
# commit
```

Example

For example (using simple names and passwords for demonstration purposes):

```
user@wae(config)# set devices authgroups group ABCgroup default-map remote-name anyuser
remote-password password123 remote-secondary-password mypassword
user@wae(config)# commit

user@wae(config)# set devices authgroups snmp-group snmp_v2 default-map community-name
mycompany
user@wae(config)# commit

user@wae(config)# set devices authgroups snmp-group snmp_v3_01
user@wae(config)# default-map community-name mycompany
user@wae(config)# default-map usm remote-name User1
user@wae(config)# default-map usm security-level auth-priv
user@wae(config)# default-map usm auth md5 remote-password pass_a123
user@wae(config)# default-map usm priv aes remote-password pass_a123
user@wae(config)# commit

user@wae(config)# set devices authgroups snmp-group snmp_v3_02
user@wae(config)# default-map community-name mycompany
user@wae(config)# default-map usm remote-name User2
user@wae(config)# default-map usm security-level auth-no-priv
user@wae(config)# default-map usm auth sha remote-password pass_b456
user@wae(config)# commit

user@wae(config)# set devices authgroups snmp-group snmp_v3_03
user@wae(config)# default-map community-name mycompany
user@wae(config)# default-map usm remote-name User2
user@wae(config)# default-map usm security-level no-auth-no-priv
user@wae(config)# commit
```

Configure a Network Access Profile

Before you begin

Confirm that authentication and SNMP groups have been configured. For more information, see Configure Device Access Using the CLI, on page 36.

Step 1 Enter the following commands:

```
# set wae nimos network-access network-access <network-access-ID> auth-group <auth-group-ID>
# set wae nimos network-access network-access <network-access-ID> snmp-group <snmp-group-ID>
```

Step 2 Repeat the following command to enter each management IP address:

```
# set wae nimos network-access network-access <network-access-IP> node-access <node-access-ID-1>
auth-group <auth_group_ID> default-snmp-group <snmp-group-ID>
ip-manage <ip-address-1>
```

Step 3 Commit the configuration:

```
# commit
```

Example

For example:

```
# set wae nimos network-access network-access as64001 auth-group ABCgroup
# set wae nimos network-access network-access as64001 snmp-group snmp_v3_01
# set wae nimos network-access network-access as64001 node-access 1.1.1.1 ip-manage
10.18.20.121
# set wae nimos network-access network-access as64001 node-access 2.2.2.2 ip-manage
10.18.20.122
# commit

# set wae nimos network-access network-access netaccess_01 auth-group ABCgroup
# set wae nimos network-access network-access netaccess_01 snmp-group snmp_v2
node-access 122.168.200.2 ip-manage 192.18.20.2
```

Create a Network Model

When creating a network you must also configure basic topology collection using the topo-igp-nimo or the topo-bgpls-xtc-nimo. For more information, see IGP Topology Collection, on page 46 and BGP-LS Topology Collection Using XTC, on page 48.

Note You have the option to load an existing plan file to create a network model. See Load Plan Files, on page 39.

Before you begin

- Device access and network access must be configured.
- If creating a network running XTC, confirm that XTC agents have been configured. For more information, see Configuring XTC Agents Using the Expert Mode, on page 22.

Enter the following commands:

```
# networks network <topo-network-model-name> nimo <NIMO-name>
network-access <network-access> <parameter-1>
<parameter-1-option> <parameter-2> <parameter-2-option>
<parameter-x> <parameter-x-option>
# commit
```

Example

The following examples show two ways to configure the topo-bgpls-xtc-nimo.

Example 1:

```
# networks network NetworkABC_topo-bgpls-xtc-nimo nimo topo-bgpls-xtc-nimo network-access
TTE_lab_access
# networks network NetworkABC_topo-bgpls-xtc-nimo nimo topo-bgpls-xtc-nimo xtc-host TTE-xtc11
# networks network NetworkABC_topo-bgpls-xtc-nimo nimo topo-bgpls-xtc-nimo backup-xtc-host
 xtc12
# networks network NetworkABC_topo-bgpls-xtc-nimo nimo topo-bgpls-xtc-nimo asn 62001
# networks network NetworkABC_topo-bgpls-xtc-nimo nimo topo-bgpls-xtc-nimo igp-protocol
isis
# networks network NetworkABC_topo-bgpls-xtc-nimo nimo topo-bgpls-xtc-nimo
extended-topology-discovery true
```

Example 2:

```
# networks network NetworkABC_topo-bgpls-xtc-nimo nimo topo-bgpls-xtc-nimo
network-access TTE_lab xtc-host TTE-xtc11
backup-xtc-host TTE-xtc12 igp-protocol isis
extended-topology-discovery true asn 62001
```

What to do next

Configure additional network collections using this network model as the source network. For more information, see NIMO Descriptions, on page 43.

Load Plan Files

You can load plan files to create a network model. This is useful if, for example, you already have a plan file with collected topology and demand information. Instead of starting from scratch by creating a new network model with basic topology collection and then augmenting it with demands, you can load an existing plan file.

Use the following command to create a network model from a plan file:

```
# wae components load-plan run plan-file <plan-file-location> network-name
<network-model-name>
```

For example:

```
# wae components load-plan run plan-file /home/tommy/us_atlanta_wan1.txt network-name
NetworkABC_topo_demands
```

Configure Additional NIMOs

This topic describes the general steps to configure different types of advanced network data collection. NIMOs are used to collect different types of data. Some NIMOs require the configuration of agents. For more information, see NIMO Descriptions, on page 43.

Before you begin

You must have a network model with basic collection to be used as a source network.

Enter the following command:

```
# networks network <network-model-name> nimo <NIMO-name> source-network <source-network>

# networks network <network-model-name> nimo <NIMO-name> <parameter-x> <parameter-x-option>
```

Example

The following example shows an lsp-config-nimo configuration:

```
# networks network NetworkABC_lsp-config-nimo nimo lsp-config-nimo source-network
NetworkABC_topo-bgpls-xtc-nimo
# networks network NetworkABC_lsp-config-nimo nimo lsp-config-nimo in-sync true
# commit
```

Configure the Archive

You can also configure the Archive using the WAE UI. See Configure the Archive and View Plan Files, on page 14.

Step 1 Launch the WAE CLI and enter configuration mode.

```
# wae_cli -C
# config
(config)#
```

Step 2 Configure the archive directory.

```
(config)# networks network <network_model_name> plan-archive archive-dir <archive_directory>
(config)# commit
```

For example:

```
(config)# networks network Network_123 plan-archive archive-dir /archive/planfiles/Network_123
(config)# commit
```

Step 3 Run archive. This saves the current network model in a plan file (.pln format) into the archive directory you specified.

```
(config)# networks network <network_model_name> plan-archive run
```

For example:

```
(config)# networks network Network_123 plan-archive run
status true
message Successfully archived plan file 20170131_1919_UTC.pln for network Network_123
```

Step 4 Confirm that the plan file was saved by going to the archive directory. The archive directory is divided into the following subfolders: years, months, and days.

For example:

```
(config)# ls /Network_123/2017/01/31
20170131_0100_UTC.pln 20170131_0330_UTC.pln 20170131_1012_UTC.pln
20170131_1312_UTC.pln 20170131_1919_UTC.pln
```

What to do next

Schedule how often a plan file is saved to the Archive. See Configure the Archive and View Plan Files, on page 14.

Manage Plan Files in Archive

Confirm that the archive has been configured and an archive directory has been created. For more information, see Configure the Archive, on page 40.

Note You can configure the Archive and view plan files using the WAE UI. See Configure the Archive and View Plan Files, on page 14.

You can perform the following tasks in archive:

- To list all plan files:

  `(config)# networks network <network_model_name> plan-archive list`

- To save the network model to archive:

  `(config)# networks network <network_model_name> plan-archive run`

- To retrieve a plan file:

  `(config)# networks network <network_model_name> plan-archive get`

Note You can use an existing plan file to create a network model. See Load Plan Files, on page 39.

CHAPTER 5

Network Interface Modules (NIMOs)

The following sections describe how to configure different types of network collection and other NIMO capabilities. Although the following topics show how to use the Expert Mode for configuration, you can also use the WAE UI or WAE CLI. The topics describe the options that can be configured using any interface.

NIMO Descriptions

Each NIMO has capabilities (derived from NETCONF protocol capabilities) that determine what it collects or deploys. The following table lists a description of each NIMO.

To list the capabilities of each NIMO, click the **get-capabilities** button (in the Expert Mode) after a NIMO is configured.

Note If you wish to consolidate different data collections (NIMO collections) under a single network model, configure the aggregator before running any collections. For more information, see NIMO Collection Consolidation, on page 51.

Collection or Capability	NIMO	Description	Prerequisite/Notes
Network Collection NIMOs			
IGP Topology Collection, on page 46	topo-igp-nimo	Discovers IGP topology using login and SNMP.	This is a basic topology collection (topology NIMO). The resulting network model is used as the source network for other NIMOs.
BGP-LS Topology Collection Using XTC, on page 48	topo-bgpls-xtc-nimo	Discovers Layer 3 topology using BGP-LS via XTC. It uses raw XTC data as the source for the topology. Node and interface/port properties are discovered using SNMP.	• XTC agents must be configured before running this collection. See Configuring XTC Agents Using the Expert Mode, on page 22. • This is a basic topology collection for networks using XTC. The resulting network model is used as the source network for other NIMOs.
VPN Collection, on page 55	topo-vpn-nimo	Discovers Layer 2 and Layer 3 VPN topology.	A network model with basic topology collection must exist.
BGP Peer Collection, on page 60	topo-bgp-nimo	Discovers BGP peering using login and SNMP.	A network model with basic topology collection must exist.
LAG Ports and LMP Interface Collection, on page 59	port-cfg-parse-nimo	Discovers LAG ports and Link Management Protocol (LMP) interfaces from router configurations in the network.	• A network model with basic topology collection must exist. • A Configuration Parsing agent must be configured before running this collection. See Configuring the Configuration Parsing Agent, on page 23.
Configure Multi-Layer Collection, on page 80	optical-nimo	In conjunction with other NIMOs, the final network collection discovers Layer 1 (optical) and Layer 3 topology.	There are configurations that must take place before configuring the optical-nimo. See Multi-Layer Collection Workflow, on page 78 .
LSP Collection Using NSO NEDs, on page 55	lsp-config-nimo	Discovers LSPs using NEDs and the LSP binding SIDs via NETCONF.	A network model with basic topology collection must exist.
LSP Collection Using SNMP, on page 62	lsp-snmp-nimo	Discovers LSPs using SNMP.	A network model with basic topology collection must exist.
PCEP LSP Collection Using XTC, on page 58	lsp-pcep-xtc-nimo	Discovers PCEP LSPs using XTC.	The BGP-LS Topology Collection Using XTC, on page 48 must be completed before running this collection.

Collection or Capability	NIMO	Description	Prerequisite/Notes
Segment Routing LSP Traffic Collection, on page 62	sr-traffic-matrix-nimo	Discovers SR LSP traffic information.	• A network model with basic topology collection must exist. • Telemetry must be set up on the router.
NIMO Collection Consolidation, on page 51	—	Aggregates various NIMO information into a single consolidated network model.	Configured network models with information you want to merge into one final network model.
Autonomous System (AS) Model Consolidation, on page 53	as-merger	Resolves interfaces, circuits, etc., that are shared between AS models to create a single, consolidated network model.	• Confirm that collection has been completed on the individual AS network models that you want to merge. • Any AS network models that use the topo-bgpls-xtc NIMO must each have an Autonomous System Number (ASN) assigned to it.
Additional NIMOs			
Continuous Collection, on page 63	traffic-poll-nimo	Collects traffic statistics (interface measurements) using SNMP polling.	• A network model with basic topology collection. • If collecting LSP traffic, a network model with LSP collection must exist. See LSP Collection Using SNMP, on page 62. • If collecting VPN traffic, a network model with VPN collection must exist. See VPN Collection, on page 55.
Network Model Visualization, on page 66	layout-nimo	Adds layout properties to a source model to improve visualization.	• A consolidated network model (aggregator). • After the layout-nimo is configured, a plan file containing layout properties must be imported back into the layout-nimo model.
Demand Mesh Creation, on page 68	demandmesh-creator-nimo	Creates a demand mesh between a set of source and destination nodes.	A network model with basic topology collection must exist.

Collection or Capability	NIMO	Description	Prerequisite/Notes
Demand Deduction, on page 67	demand-deduction-nimo	Runs demand deduction using measured SNMP traffic and by applying demand mesh builds of end-to-end demands (traffic matrix) to the network model.	This NIMO requires that you use a consolidated network model (aggregator) as a source network that includes demands and interface measurements (traffic-poller).
Running External Scripts Against a Network Model, on page 69	external-executable-nimo	Runs customized scripts to append additional data to a source network model.	A source network model and a custom script.

Basic Topology Collection

The network model resulting from basic topology collections (topology NIMOs) is used as the source network for additional data collections. To consolidate topology and other data collections, you must first set up the aggregator before running any collection. For more information on the aggregator, see NIMO Collection Consolidation, on page 51.

IGP Topology Collection

The IGP topology (topo-igp-nimo) discovers network topology using the IGP database with the collection of node properties and interface and port discovery using SNMP. This is typically the first NIMO that is configured before other NIMOs, because it provides the basic data collection needed. This NIMO provides full topology discovery and, although not common, topology discovery without interfaces or port detail collection. The network model resulting from this topology discovery is used as the source network for additional collections. It provides the core node, circuit, and interface information used by other NIMOs.

Note

- It is assumed that you are in the middle of creating a network model when performing the tasks described in this topic. For more information, see Create a Network Model, on page 21.
- Although this topic shows how to use the Expert Mode for configuration, it can be referred to for configuring options using the WAE UI or WAE CLI.

Before you begin

Device and network access profiles must be configured. See Configure Network Access, on page 20.

Step 1 Choose **topo-igp-nimo** as the NIMO type.

Step 2 Choose the network access type.

Step 3 Enter the management IP address of the seed router.

Step 4 Choose the IGP protocol that is running on the network.

Step 5 From the collect-interfaces field, choose **true** to discover the full network topology.

Step 6 (Optional) To exclude individual nodes from collection, click the **node-blacklist** tab and enter the applicable IP addresses of the nodes.

Note For more information on advanced options, see IGP Topology Advanced Options, on page 47. From the WAE UI, you can also hover your mouse over each field to view tooltips.

Step 7 Click the **Commit** button.

Step 8 Click **run-collection** > **Invoke run-collection**.

Step 9 To verify that the collection ran successfully, navigate to back to the network (**/wae:networks/network/<*network-name*>**) and click the **model** tab.

Step 10 Click **nodes**. A list of nodes and details appears, indicating a successful collection.

What to do next

Use this network model as the source network to configure additional collections. See NIMO Descriptions, on page 43.

IGP Topology Advanced Options

This topic describes advanced options available when running IGP topology collection.

Option	Description
igp	
backup-router	Secondary seed router to use for automatic failover.
get-segments	Collect segment routing data from the IS-IS database. Only valid for IS-IS on Cisco IOS XR routers.
ospf-area	Collect a single OSPF area, or collect all areas. The area ID can be specified as an integer or as an IP address. If set to 'all', ABRs are identified from area 0 information and logged into for non-zero area information.
ospf-proc-id	OSPF process ID to use when there are multiple OSPF processes. The value is a positive integer.
database-file	File in which to write the raw IGP database.
offline	Run IGP discovery in offline mode. In offline mode, no login access to routers is performed; instead, the necessary configs must be provided in the database file. Offline mode is mainly used for testing.
login-record-mode	Record the discovery process. If set to 'record', messages to and from the live network are recorded in the login-record-dir as the tool runs. Used for debugging.
login-record-dir	Directory in which to save the login record. Used for debugging.
nodes	

Option	Description
remove-node-suffix	Remove node suffixes from node names if the node contains this suffix. For example, 'company.net' removes the domain name for the network.
nodes | interfaces	
net-recorder	If set to 'record', SNMP messages to and from the live network are recorded in the net-record-file as discovery runs. Used for debugging.
net-record-file	Directory in which to save the SNMP record. Used for debugging.
interfaces	
find-parallel-links	Find parallel links that aren't in the IGP database (when IS-IS TE extensions aren't enabled).
ip-guessing	Level of IP address guessing to perform for interfaces that are not present in the topology database. (Used when IS-IS TE extensions aren't enabled.) • off—Perform no guessing. • safe—Choose guesses that have no ambiguity. • full—Make best-guess decisions when there is ambiguity.
lag	Enable LAG discovery of port members.
lag-port-match	Determine how to match local and remote ports in port circuits. • exact—Match based on LACP. • none—Do not create port circuits. • guess—Create port circuits to match as many ports as possible. • complete—Match based on LACP first, and then try to match as many as possible.
cleanup-circuits	Remove circuits that don't have IP addresses associated to interfaces. Circuit removal is sometimes required with IS-IS databases to fix IS-IS advertising inconsistencies.
copy-descriptions	Copy physical interface descriptions to logical interfaces if there is only one logical interface and its description is blank.
get-physical-ports	Collect L3 physical ports for Cisco. Collect physical ports if there is an L1 connection underneath.
min-prefix-length	Minimum prefix length to allow when finding parallel links. All interfaces with equal or larger prefix lengths (but less than 32) are considered.
min-guess-prefix-length	Minimum IP guessing prefix length. All interfaces with equal or larger prefix lengths are considered.

BGP-LS Topology Collection Using XTC

BGP-LS XTC topology (topo-bgpls-xtc-nimo) discovers Layer 3 topology using BGP-LS via XTC. It uses raw XTC data as the source for topology. Node and interface/port properties are discovered using SNMP. For testing purposes, you can also use BGP-LS XTC topology discovery using XTC only (extended topology discovery disabled) when no SNMP access is available. The network model resulting from topology discovery is used as the source network for additional collections because it provides the core node/circuit/interface information used by other NIMOs.

BGP-LS XTC topology discovery ***using XTC only*** is used as a source for only some NIMOs because it does not collect the necessary information needed by most NIMOs.

Before you begin

- Device access and network access must be configured. For more information, see Configure Device Access Using the Expert Mode, on page 19 and Configure Network Access, on page 20.
- An XTC agent must be configured and running. For more information, see Configuring XTC Agents Using the Expert Mode, on page 22.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the NIMO name; for example, networkABC_bgpls_xtc.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **topo-bgpls-xtc-nimo**.

Step 6 Enter the following information:

- **network-access**—Choose the network access.
- **xtc-host**—Choose an XTC agent.
- **backup-xtc-host**—Choose a backup XTC agent. You can enter the same XTC agent if you do not have a backup.
- **asn**—Enter 0 to collect information from all autonomous systems in the network, or enter the autonomous system number (ASN) to collect information only from a particular ASN. For example, if the XTC agent has visibility to ASN 64010 and ASN 64020, enter 64020 to collect information only from ASN 64020. You must enter an ASN if you plan to use the as-merger NIMO to consolidate different AS models into one network model.
- **igp-protocol**—Choose the IGP protocol that is running on the network.
- **extended-topology-discovery**—Choose **true** to discover the full network topology (node and interfaces).

Note For more information on advanced options, see BGP-LS XTC Advanced Options, on page 50. From the WAE UI, you can also hover your mouse over each field to view tooltips.

Step 7 (Optional) To exclude individual nodes from collection, click the **node-blacklist** tab and enter the applicable IP addresses of the nodes. For example, you might not want to see the XTC nodes in the collection.

Step 8 Click the **Commit** button.

Step 9 Click **run-xtc-collection** > **Invoke run-collection**.

Step 10 To verify that the collection ran successfully, navigate to back to the network (**/wae:networks/network/<*network-name*>**) and click the **model** tab.

Step 11 Click **nodes**. A list of nodes and details appears, indicating a successful collection.

Example

For example, if using the WAE CLI (in config mode), enter:

```
# networks network <network-model-name> nimo topo-bgpls-xtc-nimo network-access
<network-access-ID>
# networks network <network-model-name> nimo topo-bgpls-xtc-nimo xtc-host <XTC-agent>
# networks network <network-model-name> nimo topo-bgpls-xtc-nimo backup-xtc-host
<XTC-agent-backup>
# networks network <network-model-name> nimo topo-bgpls-xtc-nimo asn <ASN-number>
# networks network <network-model-name> nimo topo-bgpls-xtc-nimo igp-protocol
<IGP-protocol-type>
# networks network <network-model-name> nimo topo-bgpls-xtc-nimo extended-topology-discovery
 <true-or-false>
```

What to do next

After performing this task, you can use this network model as the source network to configure additional collections. For more information, see NIMO Descriptions, on page 43.

BGP-LS XTC Advanced Options

This topic describes advanced options available when running BGP-LS topology collection using XTC.

Option	Description
nodes	
remove-node-suffix	Remove node suffixes from node names if the node contains this suffix. For example, 'company.net' removes the domain name for the network.
nodes | interfaces	
net-recorder	If set to 'record', SNMP messages to and from the live network are recorded in the net-record-file as discovery runs. Used for debugging.
net-record-file	Directory in which to save the SNMP record. Used for debugging.
interfaces	
find-parallel-links	Find parallel links that aren't in the IGP database (when IS-IS TE extensions aren't enabled).
ip-guessing	Level of IP address guessing to perform for interfaces that are not present in the topology database. (Used when IS-IS TE extensions aren't enabled.) • off—Perform no guessing. • safe—Choose guesses that have no ambiguity. • full—Make best-guess decisions when there is ambiguity.
lag	Enable LAG discovery of port members.

Option	Description
lag-port-match	Determine how to match local and remote ports in port circuits. • exact—Match based on LACP. • none—Do not create port circuits. • guess—Create port circuits to match as many ports as possible. • complete—Match based on LACP first, and then try to match as many as possible.
cleanup-circuits	Remove circuits that don't have IP addresses associated to interfaces. Circuit removal is sometimes required with IS-IS databases to fix IS-IS advertising inconsistencies.
copy-descriptions	Copy physical interface descriptions to logical interfaces if there is only one logical interface and its description is blank.
get-physical-ports	Collect L3 physical ports for Cisco. Collect physical ports if there is an L1 connection underneath.
min-prefix-length	Minimum prefix length to allow when finding parallel links. All interfaces with equal or larger prefix lengths (but less than 32) are considered.
min-guess-prefix-length	Minimum IP guessing prefix length. All interfaces with equal or larger prefix lengths are considered.

NIMO Collection Consolidation

The aggregator uses the Delta Aggregation Rules Engine (DARE) to combine user-specified NIMOs into a single consolidated network model. The aggregator reads the capabilities of source NIMOs. For more information on aggregator functions, see Network Models, on page 3.

Note For networks using XTC, you can get automated network updates to obtain real-time network models that can be used for automation applications. For more information, see Automation Applications, on page 107.

Before you begin

Configure NIMOs that you want to include in the final network model. It is important not to run a collection or execute these NIMOs until after the initial aggregator configuration.

Step 1 Create an empty network. This will be the final consolidated network model. From the Expert Mode, navigate to **/wae:networks**, click the plus (+) sign, and enter a final network model name.

Step 2 Navigate to **/wae:wae/components/dare:aggregators** and select the aggregator tab.

Step 3 Click the plus (+) sign.

Step 4 From the drop-down destination list, select the final network and click **Add**.

Step 5 Click the source link.

Step 6 Click the plus (+) sign to add source NIMOs. Repeat until all source NIMOs you want to consolidate collections for under the final network model are added.

Step 7 Click **OK**.

Step 8 (Optional) Rules are automatically generated when a source is added or removed from the source list, or when a refresh is invoked in the resulting aggregator (dependent on the capabilities reported by source NIMOs). To select a rule set other than the default, navigate back to **/wae:wae/components/dare:aggregators/aggregator/<*network_name*>/aggregator** and select an option under the **rule-set** drop-down list. To edit rule sets, navigate to **/wae:wae/components/dare:rule-sets/rule-set** and select either default or full.

Step 9 Click the **Commit** button.

Step 10 Run the source NIMOs. The final network model will update with the latest information from the source network models. See also Aggregator and Multi-Layer Collection CLI Configuration Example, on page 52.

Example

If using the WAE CLI (in config mode), enter:

```
# wae components aggregators aggregator <final-network-model>
# sources source <nimo_1>
# sources source <nimo_2>
# commit
```

After the aggregator is configured, then run the source NIMOs.

Aggregator and Multi-Layer Collection CLI Configuration Example

This example shows how to configure the aggregator to combine Layer 3 and Layer 1 network model information using the CLI.

The following shows that L1 (optical) and L3 (topo-igp-nimo) network models have been configured on the network. For more information on how to configure the optical NIMO and topo-igp-nimo, see IGP Topology Collection, on page 46 and Configure Multi-Layer Collection, on page 80.

```
# show running-config networks network nimo
```

Layer 1 network model:

```
networks network l1-network
    nimo optical-nimo source-network l3-network
    nimo optical-nimo network-access cisco:access
    nimo optical-nimo optical-agents cisco:network
        advanced use-configure-l3-l1-mapping true
        advanced l3-l1-mapping    bg1_mapping
    !
```

Layer 3 network model:

```
    nimo topo-igp-nimo network-access cisco:access
    nimo topo-igp-nimo seed-router 10.225.121.60
```

```
nimo topo-igp-nimo igp-protocol isis
nimo topo-igp-nimo collect-interfaces true
nimo topo-igp-nimo node-blacklist 10.11.255.12
!
nimo topo-igp-nimo advanced interfaces lag true
```

Note Collection has not yet been done on the configured L1 and L3 network models.

Configure the aggregator.

```
# config
# wae components aggregators aggregator l1-l3-final-model
# sources source l1-network
# sources source l3-network
# commit
```

After the aggregator is configured, run the L3 and L1 collections.

```
# networks network l3-network nimo topo-igp-nimo run-collection
```

A status message will appear once collection is complete. Once it completes, check to see that the nodes are populated.

```
# show running-config networks network l3-network model nodes node
```

You can also check to see that the final model is also populated with L3 information.

```
# show running-config networks network l1-l3-final-model model nodes node
```

Run the L1 network collection.

```
# networks network l1-network nimo optical-nimo build-optical-topology
```

You can check again to see that the final model is now populated with L1 information.

```
# show running-config networks network l1-l3-final-model model nodes node
```

You can also open WAE Design to view the final network model (**File** > **Open from** > **WAE Automation Server** and select the final network model.

Autonomous System (AS) Model Consolidation

The as-merger NIMO resolves interfaces, circuits, etc., that are shared between AS models to create a single, consolidated network model.

Before you begin

- Confirm that collection has been completed on the individual AS network models that you want to merge.

Note BGP collection must be part of all AS models.

- Any AS network models that use the topo-bgpls-xtc NIMO must each have an Autonomous System Number (ASN) assigned to it. For more information, see BGP-LS Topology Collection Using XTC, on page 48.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a name for the consolidated AS network model.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **as-merger-nimo**.

Step 6 Click **as-merger-nimo**.

Step 7 From **sources**, add the individual AS models you want to merge.

Step 8 From **subtrees**, enter **model** as the value.

Step 9 From **generate capabilities**, select one of the following:

- **false**—Select this option if the AS sources are DARE networks.
- **true**—Select this option if the AS sources are not DARE networks and are created by other NIMOs.

Step 10 Click the **Commit** button.

Step 11 Click **merge**.

Example

If using the WAE CLI (in config mode), enter:

```
# config
# networks network <as-merger-name> nimo as-merger sources [ <AS1-source> <AS2-sourse>
<ASn-source> ]
# networks network <as-merger-name> nimo as-merger subtrees [ model ]
# networks network <as-merger-name> nimo as-merger generate-capabilities <flag>
# commit
```

For example:

```
# config
# networks network AS100AS200 nimo as-merger sources [ AS6100 AS6200 ]
# networks network AS100AS200 nimo as-merger subtrees [ model ]
# networks network AS100AS200 nimo as-merger generate-capabilities <false>
# commit
```

VPN Collection

The VPN Collection (topo-vpn-nimo) discovers Layer 2 and Layer 3 VPN topology.

Before you begin

Network topology collection must be complete. For more information, see Create a Network Model, on page 21.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names; for example, networkABC_vpn.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **topo-vpn-nimo**.

Step 6 Click **topo-vpn-nimo** and enter the following:

- **source-network**—Choose the applicable network model that contains basic topology information.
- **network-access**—Choose the network access.

Step 7 Click the **vpn-types** tab.

Step 8 Click the plus (+) icon to add at least one VPN type:

- **VPWS**—Add this type when Virtual Private Wire Service is being used in the network.
- **L3VPN**—Add this type when Layer 3 VPN is being used in the network.

Step 9 Click the **Commit** button.

Step 10 Navigate back to the **topo-vpn-nimo** tab and click **run-collection** > **Invoke run-collection**.

LSP Collection Using NSO NEDs

The lsp-config-nimo discovers LSP configuration using NEDs. This collection lets you deploy changes to LSPs, named paths, and segment lists in the network. For a configuration example of creating segment lists and LSP paths, see Create Segment Routing LSPs, on page 57.

Before you begin

- You must have the Network Services Orchestrator (NSO) installed on your system and vendor network element drivers (NEDs). Contact your Cisco representative to acquire the appropriate NEDs.
- Confirm that a WAE LSA configuration with an NSO instance is configured. See Install LSA Packages, on page 126 for more information.
- A basic topology network model must exist.

Step 1 Copy the appropriate LSP NIMO configuration files from the WAE installation directory to your WAE run-time packages directory.

a) From the Unix shell, navigate to **`<wae_installation_directory>/packages/cisco-wae-lsp-config-nimo`** directory.

b) Copy the appropriate LSP NIMO package (for example, **cisco-wae-lsp-config-nimo-rfs**) directory to <*wae_run_time_directory*>/packages.

Step 2 Copy the appropriate NED packages to <*wae_run_time_directory*>/packages.

Step 3 From the Expert Mode, navigate to **/wae:networks**.

Step 4 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names; for example, networkABC_lsp_config.

Step 5 Click **Add**.

Step 6 Click the **nimo** tab.

Step 7 From the **Choice - nimo-type** drop-down list, choose **lsp-config-nimo**.

Step 8 Click **lsp-config-nimo** and enter the source topology network.

Note
- The lsp-config-nimo must follow a topology NIMO. For example, you cannot choose a layout network (layout-nimo) as the source network.
- For information on advanced options, see LSP Configuration Advanced Options, on page 56. From the WAE UI, you can also hover your mouse over each field to view tooltips.

Step 9 Click the **Commit** button.

Step 10 Click **run-collection** > **Invoke run-collection**.

LSP Configuration Advanced Options

This topic describes advanced options available when configuring the lsp-config-nimo.

Option	Description
Attributes	
action-timeout	Timeout (in minutes) to use for all lsp-config-nimo actions. The default value is 0. Large networks might require a larger value.
create-nodes	Whether the device-sync action creates nodes. The default value is 'false'.
perform-sync-from	By default, the run-collection and device-sync actions first perform a device sync-from, which takes several seconds per device. For large networks, it is more efficient to set this value to 'false' and perform a single device sync-from before performing a run-collection. The default value is 'false'.

Option	Description
preserve-device-config	Whether the run-collection commits changes to the device model and how 're-deploy reconcile' is performed. If set to 'true', collection will also preserve portions of the device model not represented in the service model since reconciliation is done with the 'keep-non-service-config' option. The default option is 'true'.
Actions	
copy-topology	Copy only the source network to this network. **Warning** This action removes all LSPs.
device-sync	Discover and reconcile LSPs in the topology network. The source network is not copied.
reconcile	Reconcile the LSPs in the topology network with the device model.

Create Segment Routing LSPs

The following procedure describes the commands to create segment routing LSPs.

Step 1 Launch the WAE CLI from the WAE run-time directory and enter configuration mode.

```
waerun# wae_cli -C
wae@wae# config
wae@wae$#
```

Step 2 Create a segment list with two hops.

```
wae@wae(config)# networks network <network_name> model nodes node <node_name>
 segment-lists segment-list <segment_list_name>
```

The first hop is a node hop.

```
wae@wae(config-segment-list-<segment_list_name>)# hops hop 1 node node-name <node_name>
wae@wae(config-hop-1)# exit
```

The second hop is an interface hop.

```
wae@wae(config-segment-list-<segment_list_name>)# hops hop 2 interface node-name <node_name>
interface-name <interface_name>
wae@wae(config-hop-2)# exit
wae@wae(config-segment-list-<segment_list_name>)# exit
```

Step 3 Create an LSP with two LSP paths.

```
wae@wae(config)# networks network <network_name> model nodes node <node_name>
 segment-lists segment-list <segment_list_name>
```

The first path is a PCE-delegated path.

```
wae@wae(config-node-<node_name>)# lsps lsp <lsp_name> destination <destination> lsp-paths lsp-path
1 type segment-routing pce-delegated true
wae@wae(config-lsp-path-1)# exit
```

The second path uses the previous segment list that was created earlier.

```
wae@wae(config-lsp-<lsp_name>)# lsp-paths lsp-path 2 type segment-routing segment-list <segment_list>
wae@wae(config)# commit
```

Example

For example:

```
wae@wae(config)# networks network <network_name> model nodes node <node_name> segment-lists
 segment-list <segment_list_name>
wae@wae(config-segment-list-<segment_list_name>)# hops hop 1 node node-name <node_name>
wae@wae(config-hop-1)# exit
wae@wae(config-segment-list-<segment_list_name>)# hops hop 2 interface node-name <node_name>
 interface-name <interface_name>
wae@wae(config-hop-2)# exit
wae@wae(config-segment-list-<segment_list_name>)# exit
wae@wae(config-node-<node_name>)# lsps lsp <lsp_name> destination <destination> lsp-paths
lsp-path 1 type segment-routing pce-delegated true
wae@wae(config-lsp-path-1)# exit
wae@wae(config-lsp-<lsp_name>)# lsp-paths lsp-path 2 type segment-routing segment-list
<segment_list>
wae@wae(config)# commit
```

PCEP LSP Collection Using XTC

PCEP LSP discovery using XTC (lsp-pcep-xtc-nimo) uses the data collected from the bgpls-xtc-nimo and appends PCEP LSP information, thus creating a new network model.

Before you begin

Confirm that BGP-LS topology collection using XTC (bgpls-xtc-nimo) has been completed for a network. You will need to use this model as the source network for collecting PCEP LSPs. For more information, see BGP-LS Topology Collection Using XTC, on page 48.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names; for example, **`networkABC_lsp_pcep_xtc`**.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **lsp-pcep-xtc-nimo**.

Step 6 Click **lsp-pcep-xtc-nimo** and enter the source network. This is the network model that contains topology information collected using the bgpls-xtc-nimo.

Step 7 Click the **xtc-hosts** tab.

Step 8 Click the plus (+) icon and enter the following:

- **name**—Enter an XTC hostname. This can be any arbitrary name.

- **xtc-host**—From the drop-down list, choose one of the XTC hosts that was previously configured. For more information, see Configuring XTC Agents Using the Expert Mode, on page 22.

Step 9 Click the **Commit** button.

Step 10 Click **run-xtc-collection** > **Invoke run-collection**.

Step 11 To verify that the collection ran successfully, navigate to back to the network (**/wae:networks/network/<*network-name*>**) and click the **model** tab.

Step 12 Click **nodes**. A list of nodes and details appears, indicating a successful collection.

Step 13 Choose one of the nodes that you know has an LSP and click the **lsps** tab.

Step 14 Click the **lsp** link. A table with a list of discovered LSPs appears.

LAG Ports and LMP Interface Collection

The port-cfg-parse NIMO discovers LAG ports and Link Management Protocol (LMP) interfaces from router configurations in the network. This NIMO is used for multi-layer collection.

Note You cannot use the WAE UI to configure this collection.

Before you begin

- A topology network model must exist. See Create a Network Model, on page 21.
- The Configuration Parsing agent must be configured and running. For more information, see Configuring the Configuration Parsing Agent, on page 23.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names; for example, networkABC_port-cfg-parse.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 Choose **port-cfg-parse-nimo** as the NIMO type.

Step 6 Click **port-cfg-parse-nimo** and enter the following information:

- **source-network**—Choose the applicable network model that contains topology information.
- **cfg-parse-agent**—Choose a configuration parsing agent.

Note For information on advanced options, see Port Config Parse Advanced Options, on page 60.

Step 7 Click the **Commit** button.

Step 8 Click **run-collection** > **Invoke run-collection**.

What to do next

After performing this task, you can use this network model as a source network to configure additional collections. For more information, see NIMO Descriptions, on page 43.

Port Config Parse Advanced Options

This topic describes advanced options available when creating the port-cfg-parse NIMO.

Option	Description
lag	Enable LAG discovery of port members.
lmp	Enable discovery of LMP interfaces.

BGP Peer Collection

The topo-bgp-nimo discovers BGP topology via SNMP and login. It uses a topology network (typically an IGP topology collection model) as its source network and adds BGP links to external ASN nodes.

Before you begin

A topology network model must exist. See Create a Network Model, on page 21.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names; for example, networkABC_topo_bgp.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **topo-bgp-nimo**.

Step 6 Click **topo-bgp-nimo** and enter the following information:

- **source-network**—Choose the applicable network model that contains basic topology information.
- **network-access**—Choose a network access profile that was previously configured.
- **min-prefix-length**—(Optional) Enter the min-prefix-length to control how restrictive IPv4 subnet matching is in discovering interfaces as BGP links.
- **min-IPv6-prefix-length**—(Optional) Enter the min-IPv6-prefix-length to control how restrictive IPv6 subnet matching is in discovering interfaces as BGP links.
- **login-multi-hop**—(Optional) Choose whether to disable login-multihop if you do not want to log in to routers that potentially contain multihop peers.

For more information on advanced options, see BGP Topology Advanced Options, on page 61.

Step 7 Click the **peer-protocol** tab and enter applicable IPv4 and IPv6 addresses.

Step 8 Click the **Commit** button.

Step 9 Click **run-collection** > **Invoke run-collection**.

BGP Topology Advanced Options

This topic describes advanced options available when running BGP topology collection.

Option	Description
force-login-platform	Override platform detect and use the specified platform. Valid values: cisco, juniper, alu, huawei.
fallback-login-platform	Fallback vendor in case platform detection fails. Valid values: cisco, juniper, alu, huawei.
try-send-enable	When logging in to a router, send an enable password if the platform type is not detected. This action has the same behavior as '-fallback-login-platform cisco'.
internal-asns	Specify internal ASNs. If used, the specified ASNs are set to internal; all others are set to external. The default is to use what is discovered.
asn-include	Specify ASNs of interest. If used, peer discovery is restricted to this list. The default is to peer with all discovered external ASNs.
find-internal-asn-links	Find links between two or more internal ASNs. Normally this action is not required because IGP discovers these links.
find-non-ip-exit-interface	Search for exit interfaces that are not represented as next-hop IP addresses, but rather as interfaces (which are rare). **Note** This action increases the amount of SNMP requests for BGP discovery, which affects performance.
find-internal-exit-interfaces	Collect exit interfaces to internal ASNs.
get-mac-address	Collect source MAC addresses of BGP peers connected to an Internet Exchange public peering switch. This action is required only for MAC accounting.
use-dns	Whether to use DNS to resolve BGP IP addresses.
force-check-all	Check all routers even if there is no indication of potential multi-hop peers. This action could be slow.
net-recorder	If set to 'record', SNMP messages to and from the live network are recorded in the net-record-file as discovery runs. Used for debugging.
net-record-file	Directory in which to save the SNMP record. Used for debugging.
login-record-mode	Record the discovery process. If set to 'record', messages to and from the live network are recorded in the login-record-dir as the tool runs. Used for debugging.
login-record-dir	Directory in which to save the login record. Used for debugging.

LSP Collection Using SNMP

The lsp-snmp-nimo discovers LSP information using SNMP.

Before you begin

A basic topology network model must exist. See Basic Topology Collection, on page 46.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names; for example, networkABC_lsp_config.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **lsp-snmp-nimo**.

Step 6 Click **lsp-snmp-nimo** and enter the following:

- **source-network**—Choose the applicable network model that contains basic topology information.
- **network-access**—Choose a network access profile that was previously configured.
- **get-frr-lsps**—Choose **true** if you want to discover Multiprotocol Label Switching (MPLS) Fast Reroute (FRR) LSP (backup and bypass) information.

Step 7 Click the **Commit** button.

Step 8 Click **run-collection** > **Invoke run-collection**.

Segment Routing LSP Traffic Collection

Segment Routing (SR) LSP Traffic Collection (sr-traffic-matrix-nimo) discovers SR LSP traffic. This NIMO enables the generation of demands between external interfaces of a network from collected telemetry data.

Before you begin

- A basic topology network model must exist. See IGP Topology Collection, on page 46 or BGP-LS Topology Collection Using XTC, on page 48.
- Telemetry must be configured on the router.

Note You cannot use the WAE UI to configure this collection.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names; for example, networkABC_sr_traffic_matrix.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **sr-traffic-matrix-nimo**.

Step 6 Click **sr-traffic-matrix-nimo** and enter the source network.

Step 7 Click the **Commit** button.

Step 8 Click **run-collection** > **Invoke run-collection**.

Example

If using the WAE CLI (in config mode), enter:

```
# networks network <network-model-name> nimo sr-traffic-matrix-nimo source-network
<source-network>

# commit
```

Continuous Collection

The traffic-poll-nimo collects traffic statistics (interface measurements) using SNMP polling.

Before you begin

This NIMO requires the following:

- Basic topology network model.
- If collecting VPN traffic, a VPN network model must exist. See VPN Collection, on page 55.
- If collecting LSP traffic, an LSP network model must exist. See LSP Collection Using SNMP, on page 62.

Limitations

- Node traffic information from external interfaces is not collected.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names; for example, networkABC_traffic_polling.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **traffic-poll-nimo**.

Step 6 Click **traffic-poll-nimo** and enter the following:

- **source-network**—Choose the applicable network model.
- **network-access**—Choose the network access.

Step 7 To run continuous traffic collection for interfaces, click the **iface-traffic-poller** tab and enter the following:

- **enabled**—Set to **true**.
- **period**—Enter the polling period, in seconds. We recommend starting with 60 seconds. See Tuning Traffic Polling, on page 65 to tune the polling period.
- **qos-enabled**—Set to **true** if you want to enable queues traffic collection.
- **vpn-enabled**—Set to **true** if you want to enable VPN traffic collection. If set to true, confirm that the source network model has VPNs enabled.

Step 8 To run continuous traffic collection for LSPs, click the **lsp-traffic-poller** tab and enter the following:

- **enabled**—Set to **true**.
- **period**—Enter the polling period, in seconds. We recommend starting with 60 seconds. See Tuning Traffic Polling, on page 65 to tune the polling period.

Step 9 Click the **Commit** button.

Step 10 Navigate back to the **traffic-poll-nimo** tab and click **run-snmp-traffic-poller** > **Invoke run-snmp-poller**. To stop continuous collection in the future, click **stop-snmp-traffic-poller**.

Traffic Polling Advanced Options

This topic describes advanced options available when configuring continuous collection (traffic-poll-nimo).

Option	Description
snmp-traffic-poller	
net-recorder	This option is typically used for debugging. Set to **record** to record SNMP messages to and from the live network in the net-record-file when discovery is running.
net-record-file	Enter the filename where recorded SNMP messages are saved.
verbosity	Set the poller logging level. The default is 40. • 40—INFO • 50—DEBUG • 60—TRACE
stats-computing-minimum-window-length	Enter the minimum window length for traffic calculation, in seconds. The default is 300 seconds.
stats-computing-maximum-window-length	Enter the maximum window length for traffic calculation, in seconds. The default is 450 seconds.
raw-counter-ttl	Enter how long to keep raw counters, in minutes. The default is 15 minutes.
snmp-traffic-population	

Option	Description
scheduler-interval	Enter the interval to perform traffic population, in seconds. The default is 300 seconds. It will send traffic statistics to the configuration database (CDB). If set to 0 (typically set when using the Bandwidth on Demand application), WMD will pull the traffic statistics from the RPC API. The traffic statistics will not be sent to CDB.
connect-timeout	Enter the maximum execution time for traffic population, in minutes.

Tuning Traffic Polling

To run traffic polling efficiently, do the following:

1. Start with the default options and run continuous collection for several hours. The default values are:

```
iface-traffic-poller/period = 60
lsp-traffic-poller/period = 60
advanced/snmp-traffic-poller/stats-computing-minimum-window-length = 300
advanced/snmp-traffic-poller/stats-computing-maximum-window-length = 450
advanced/snmp-traffic-poller/raw-counter-ttl = 15
advanced/snmp-traffic-population/scheduler-interval = 300
```

2. View the poller.log file. By default, the file is located in `<wae_run_time_directory>/logs/<network_name>-poller.log`.
3. Search for and filter the following text:
   - Interface Traffic Poller: Collection complete. Duration:
   - LSP Traffic Poller: Collection complete. Duration:
4. Note the duration on average and during a worst-case scenario.

For example, interface polling takes 30 seconds on average and 45 seconds in a worst-case scenario. LSP polling takes 90 seconds on average and a maximum of 120 seconds. To run interface polling efficiently, we recommend starting with the worst-case (higher) numbers. In this example, start with the interface poller every 45 seconds and the LSP poller every 120 seconds. Then, as time passes, you can tune these numbers as needed.

To calculate traffic, you need at least two counters (two polling runs). Counters are chosen among those stored in memory using a sliding window. The advanced `raw-counter-ttl` option dictates how long the counters are kept. The advanced `stats-computing-minimum-window-length` and `stats-computing-maximum-window-length` options specify the size of the sliding window. The basic calculation on how to configure these parameters is:

```
stats-computing-minimum-window-length = ( poller duration ) * 5
stats-computing-maximum-window-length = stats-computing-minimum-window-length * 1.5
raw-counter-ttl = stats-computing-minimum-window-length * 3 / 60
```

In this example, because LSP collection takes longer than interface collection, use the LSP worst-case duration (120 seconds) to determine the sliding window size. You get the following numbers:

```
stats-computing-minimum-window-length = 120*5 = 600
stats-computing-maximum-window-length = stats-computing-minimum-window-length * 1.5 = 600
* 1.5 = 900
```

```
raw-counter-ttl = stats-computing-minimum-window-length * 3 / 60 = 30
```

You can run traffic calculation as often as every 30 to 45 seconds, or as slowly as every 120 seconds, as configured with the advanced `scheduler-interval` option. Set it to 120 seconds to conserve CPU. It might take a while to calculate traffic in large networks, so make sure to set an adequate **`connect-timeout`** option. You can find the actual period in `logs/ncs-java-vm.log`. For example:

```
Traffic calculation took (ms) 3750
```

You can tune these parameters more aggressively or conservatively. We recommend starting with more conservative settings, and then tuning them as needed. If you see traffic being dropped from the output, you can increase the `stats-computing-maximum-window-length` and `raw-counter-ttl` options accordingly.

Network Model Visualization

The layout-nimo adds layout properties to a source network model to improve visualization when importing the plan file into WAE Design. The NIMO automatically records changes to the layout properties. When the source network model changes, the layout of the destination model is updated.

The layout in the destination network serves as a template that is applied to the source network. The resulting network is saved as the new destination network. If the source layout contains no layout information, the layout from the destination network is simply added to the source network. If the source network contains layout information, that layout is maintained unless there is a conflict with the layout in the destination network. If a conflict exists, the layout information in the destination network takes precedence over the information in the source network.

For example, assume that a new L1 node is added to the source network with a corresponding site assignment. This L1 node is then added to the destination network with its site assignment. Now assume that an existing L1 node has a different site assignment in the source and destination networks. In this case, the site assignment in the destination network is retained.

There are two steps:

1. Create a new network model using the layout-nimo.
2. Add a layout template to the new network model using WAE Design and then send a patch. For more information, see the Cisco WAE Network Visualization Guide.

Before you begin

- A basic topology network model must exist. See Basic Topology Collection, on page 46.

Note You cannot use the WAE UI to configure this collection.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names. This procedure uses **networkABC_layout** as an example.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **layout-nimo**.

Step 6 Click **layout-nimo** and enter a source network.

Note The source network cannot have a preexisting layout template assigned to it.

Step 7 Click the **Commit** button.

Step 8 Click **run-layout** > **Invoke run-layout**.

Step 9 Launch WAE Design and choose **File** > **Open From** > **WAE Automation Server**.

Step 10 Enter the appropriate details, choose the plan file for the network model you just created (networkABC_layout), and click **OK**.

Step 11 Edit the layout. See the "Using Layouts" chapter in the Cisco WAE Network Visualization Guide.

Step 12 Create and send the patch (**Tools** > **Patches** > **Create**). See the "Patch Files" chapter in the Cisco WAE Design User Guide.

Step 13 From the Expert Mode, navigate back to the layout-nimo network model (networkABC_layout).

Step 14 Click the **layouts** tab.

Step 15 Click **layout** to confirm that the table has been populated with layout data. The next time you open the plan file from WAE Design, the topology is displayed with the saved layout properties.

Demand Deduction

Traffic can be measured on interfaces, interface queues, and LSPs. You can use demand deduction to estimate demand traffic based on any of these measurements. For information on demand deduction, see the Cisco WAE Design User Guide. The demand-deduction-nimo runs demand deduction using measured SNMP traffic and by applying demand mesh builds of end-to-end demands (traffic matrix) to the network model.

Before you begin

A consolidated network model (aggregator NIMO) with demands (for example, networkABC_demandmesh) and interface measurements (for example, networkABC_traffic_poller) must exist.

Note You cannot use the WAE UI to configure this NIMO.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names; for example, networkABC_demand_deduction.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **demand-deduction-nimo**.

Step 6 Click **demand-deduction-nimo** and enter the source network. The source network is typically a consolidated network model (aggregator NIMO) that has demands and traffic poller information.

Step 7 Click the **input** tab.

- **nodes**—Use measured traffic on nodes. The default value is true.
- **interfaces**—Use measured traffic on interfaces. The default value is true.
- **lsps**—Use measured traffic on LSPs.
- **remove-zero-bw-demands**—Remove any demands with no (zero) traffic (or less than the `zero-bw-tolerance` option). The default is true.
- **zero-bw-demands-tolerance**—Enter a tolerance value (less than zero) that will be considered as zero traffic.
- **demand-upper-bound**—Enter an upper bound on the demand traffic levels. A warning is issued if the upper bound is reached. The default is 10,000 Mb/s.

Step 8 Click the **Commit** button.

Step 9 Navigate back to the **demand-deduction-nimo** tab and click **run** > **Invoke run**.

Step 10 To confirm that demand deduction succeeded, navigate to **/wae:networks/network/<*network_model*>/model/demands** and see if the traffic column was populated.

Demand Mesh Creation

Demand meshes are a time-efficient way of creating numerous demands for all or part of the network. The demandmesh-creator-nimo creates a demand mesh between a set of source and destination nodes.

Before you begin

A basic topology network model must exist. See Basic Topology Collection, on page 46.

Note You cannot use the WAE UI to configure this NIMO.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that contains the source network and NIMO names; for example, networkABC_demandmesh.

Step 3 Click **Add**.

Step 4 Click the **nimo** tab.

Step 5 From the **Choice - nimo-type** drop-down list, choose **demandmesh-creator-nimo**.

Step 6 Click **demandmesh-creator-nimo** and enter the source network.

Step 7 Click the **input** tab and enter:

- **source-nodes**—Enter nodes to restrict the demand mesh source to a specified set of nodes, external asynchronous systems, or external endpoints. You can view nodes from **/wae:networks/network/<*network_model*>/model/nodes**.
- **both-directories**—Include all demands from the destination to source and also from the source to destination. The default is true.
- **service-class**—Assign a service class type to demands that are created (for example, QoS). If empty, a default class is used.
- **topology**—Enter the topology. The default is that demands are applied to all topologies.

- **choice-destination: destination-nodes**—Choose this option if you want to create demands to destinations other than what has been selected as the source.
- **choice-destination: destination-equal-source**—If set to **true**, the destination node is set to source. The default is true.

Note To get a full demand mesh, do not edit any fields and set destination-equal-source to true. This picks up all the nodes in the network and creates all possible demands (including demands to the node itself).

Step 8 Click the **Commit** button.

Step 9 Navigate back to the **demandmesh-creator-nimo** tab and click **run** > **Invoke run**.

Step 10 To confirm that demands have been created, navigate to **/wae:networks/network/<*network_model*>/model/demands**. The table is populated with demand information.

Running External Scripts Against a Network Model

The external-executable-nimo lets you run a customized script against a selected network model. You might want to do this when you want specific data from your network that existing WAE NIMOs do not provide. In this case, you take an existing model created in WAE and append information from a custom script to create a final network model that contains the data you want.

Before you begin

You must have a source network model and a custom script.

Note You cannot use the WAE UI to configure this NIMO.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that is easily identifiable; for example, networkABC_my_script.

Step 3 Click the **nimo** tab.

Step 4 From the **Choice - nimo-type** drop-down list, choose **external-executable-nimo**.

Step 5 Click **external-executable-nimo** and select the source network.

Step 6 Click the **advanced** tab and enter the following:

- **input-file-version**—Enter the plan file version of the source network model, such as 6.3, 6.4, and so on. The default is 7.0.
- **input-file-format**—Specify the plan file format of the source network model. The default is .pln.
- **argv**—Enter arguments (in order) that are required for the script to run. Enter $$input for the source network model and $$output for the resulting network model (after the script runs). It is important to note that $$input, $$output, and other argv arguments must be listed in the order that is required by the script. For an example, see Running External Scripts Example, on page 70.

Step 7 From the external-executable-nimo tab, click **run...**.

Example

If using the WAE CLI (in config mode), enter:

```
networks network <network-model-name> nimo external-executable-nimo source-network
<source-network> advanced argv [ <arg_1> <arg_2> <arg_x> $$input $$output ]
admin@wae(config-network-<network-model-name>)# commit
Commit complete.
admin@wae(config-network-<network-model-name>)# exit
admin@wae(config)# exit

admin@wae# networks network <network-model-name> nimo external-executable-nimo run
```

Running External Scripts Example

This example describes how to use the external-executable-nimo with the WAE CLI. The sample python script (ext_exe_eg.py) appends a description to every interface in the network with "My IGP metric is <*value*>."

Contents of ext_exe_eg.py:

```
import sys
from com.cisco.wae.opm.network import Network

src  = sys.argv[1]
dest = sys.argv[2]

srcNet =  Network(src)

for node in srcNet.model.nodes:
    cnt = 1
    for iface in node.interfaces:
        iface.description = 'My IGP metric is ' + str(iface.igp_metric)
        cnt = cnt + 1

srcNet.write(dest)
```

In the WAE CLI, enter:

```
admin@wae(config)# networks network net_dest nimo external-executable-nimo source-network
net_src
advanced argv [ /usr/bin/python /home/user1/srcs/br1/mate/package/linux/run/ext_exe_eg.py
$$input $$output ]
admin@wae(config-network-net_dest)# commit
Commit complete.
admin@wae(config-network-net_dest)# exit
admin@wae(config)# exit

admin@wae# networks network net_dest nimo external-executable-nimo run
status true
message Changes successfully applied.
```

Confirm the script succeeded:

```
admin@wae# show running-config networks network net_dest model nodes node cr1.atl interfaces
 interface to_cr1.hst description
networks network net_dest
 model nodes node cr1.atl
  interfaces interface to_cr1.hst
   description "My IGP metric is 37"
  !
 !
!
```

CHAPTER 6

Configure the WAE Modeling Daemon (WMD)

WMD provides a real-time network model in memory. DARE receives network changes (from NIMOs and the XATP module) and sends a patch with these changes to WMD. For more information on how WMD and DARE works, see the Overview, on page 1 chapter.

To configure DARE, WMD, and the XATP module, see the following topics:

Configure the WAE Modeling Daemon (WMD)

WMD provides a near real-time representation (model) of the network in memory so that applications can get access to that model. It gets changes from DARE, the XTC Agent to Patch module, and also any scheduled feeds from NIMOs.

Before you begin

The following information should be on hand or configured:

- Final network model name
- Design RPC
- If using continuous polling, name of traffic-poll-nimo network

Step 1 From the Expert Mode, navigate to **/wae:wae/components/wmd:wmd** and click **config**.

Step 2 From the network-name drop-down list, select the final network model.

Step 3 From the enable drop-down list, select **true** to enable WMD.

Step 4 Click **rpc-connection** and enter Design RPC values.

Step 5 Click **app-subscriber-connections** and enter host and port information for all automation application connections.

Step 6 Click **measured-traffic-source** and enter continuous polling information.

Step 7 Click **dare** and enter the following values.

- **dare-destination**—Select the final network model.

- **connection-attempts**—Enter the number of times to try to reconnect until the connection is reestablished.
- **connection-retry-delay**—Enter the interval (in seconds) between connection attempts.

Step 8 (Optional) To enable demand mesh and deduction, click **demands** and enter the following values.

- **add-demands**—Select true to enable demands. When enabled, WMD is set up to run demand mesh and demand deduction for all applications using WMD. So, when the continuous poller updates WMD, WMD triggers demands.
- **demand-mesh-config**—Enter the applicable demand mesh options. For more information on fields, see Demand Mesh Creation, on page 68.
- **demand-deduction-config**—Enter the applicable demand mesh options. For more information on fields, see Demand Deduction, on page 67.

Example

WAE CLI (in config mode) example:

```
# wae components wmd config network-name <final_model_name> dare dare-destination
<final_model_name>
# wae components wmd config network-name <final_model_name> demands add-demands true
demand-mesh-config dest-equals-source true
```

Configure the XTC Agent to Patch Module

The XTC Agent to Patch (XATP) module connects to the XTC agent and sends any XTC changes or PCEP/LSPs as a patch to DARE. As part of the configuration you need to point to WMD to get the latest WAE model.

Before you begin

The following information should be on hand or configured:

- Final network model name
- XTC agent
- WMD

Step 1 From the Expert Mode, navigate to **/wae:wae/components/xatp:xatp** and click **config**.

Step 2 From the enable drop-down list, select **true** to enable XATP.

Step 3 From the xtc-agent drop-down list, select the source XTC agent.

Step 4 Click **WMD** and enter the following values.

- **host**—Enter the associated WMD instance.
- **port**—Enter the WMD port.
- **connection-attempts**—Enter the number of times XATP should try to reconnect to WMD if the connection is lost. If 0 is entered, XATP will try to reconnect until the connection is reestablished.
- **connection-retry-delay**—Enter the interval (in seconds) between connection attempts.

Step 5 Click **dare**. Select the final network model that patches will be sent to and the source NIMOs.

Example

WAE CLI (in config mode) example:

```
# wae components xatp config wmd connection-attempts 0
# wae components xatp config xtc-agent <xtc-agent-name>
# wae components xatp config dare aggregator-network <aggregator-network-name>
# wae components xatp config dare topo-bgpls-xtc-nimo-network
<topo-bgpls-xtc-nimo-network-name>
# wae components xatp config dare pcep-lsp-xtc-nimo-network <pcep-lsp-xtc-nimo-network-name>
# wae components xatp config enable true
```

CHAPTER 7

Multi-Layer Collection

Multi-layer (Layer 1 and Layer 3) network collection is an advanced collection configuration. This section describes how to configure collection from a multi-layer network.

Note You cannot use the WAE UI to configure this collection.

After this procedure, you should be able to collect and model the following information:

- Topologies from DWDM networks that support Generalized Multiprotocol Label Switching (GMPLS) with non-User Network Interface (UNI) circuits
- L1 circuit paths
- L1 topology with and without amplifiers
- Unprotected and restorable paths
- Actual L1 circuit path hops
- Feasibility metrics and limits
- Inactive L1 links
- L1 node and L1 link SRLGs
- Site information
- User properties
- Aging information and last seen date. To configure aging, see Configure Aging, on page 120.

This section contains the following topics:

Multi-Layer Collection Limitations

The following multi-layer (L1-L3) collection limitations exist:

- Collection is supported only on the following platforms:
  - Cisco Network Convergence System (NCS) 2000 platforms running version 10.61 for L1 devices.
  - Cisco Aggregation Services Routers (ASR) 9000, Cisco Carrier Routing System (CRS), and Cisco NCS 5500 platforms running IOS-XR for L3 devices.
- Multi-layer collection is limited to the collection of unprotected circuits.
- Collection of non-WSON circuits is not supported.
- L3-L1 mapping by LMP is supported only if the controller interface name is the same as the actual L3 interface name or of the form "dwdmx/x/x/x" where the "x/x/x/x" subscript matches that of the corresponding L3 interface.
- Lambda mapping is currently supported only for circuit paths but not for path hops.

Multi-Layer Collection Workflow

This workflow describes the high-level steps to configure multi-layer collection.

Step	For more information, see...
1. Review multi-layer collection limitations.	Multi-Layer Collection Limitations, on page 78
2. Configure and run the optical plug-in.	Configure the Optical Plug-In, on page 79
3. Obtain and configure L1 - L3 mapping information.	Configure L3-L1 Mapping Information, on page 78
4. Configure multi-layer collection.	Configure Multi-Layer Collection, on page 80

Configure L3-L1 Mapping Information

Note L3-L1 mapping by Link Management Protocol (LMP) is supported only if the controller interface name is same as the actual L3 interface name or in the form of “dwdmx/x/x/x” where the "x/x/x/x" subscript matches that of the corresponding L3 interface.

You can get L3-L1 information by running the configuration parsing agent. See Configuring the Configuration Parsing Agent, on page 23. The parse configuration agent should be specified in the optical nimo as follows: `networks/<multilayer_network_name>/nimo/optical-nimo/advanced/cfg-parse-agent`

If you are not running the configuration parsing agent, manually enter the mapping of L3 nodes and interfaces to L1 nodes and ports:

1. From the Expert Mode, navigate to **/wae:wae/nimos** and click the **l3-l1-mappings** tab.
2. Click the plus (+) sign, enter an arbitrary name for the L3-L1 mapping group, and click **Add**.
3. Click the **l3-l1-mapping** tab and click the plus (+) sign to enter each mapping. Repeat this step to enter all L3-L1 mappings.
4. Click the **Commit** button.

If using the WAE CLI (in config mode), enter:

```
wae@ncs(config)# wae nimos l3-l1-mappings l3-l1-mappings <mapping_name> l3-l1-mapping
<l3_node> <l3_interface> <l1_node> <l1_interface>
wae@ncs(config)# networks network <ml_network> optical-nimo optical-agents [<agent_name>]
advanced use-configured-l3-l1-mapping true l3-l1-mapping <mapping_name>
wae@ncs(config-networks-<ml_network_name>)# commit
```

Example using values:

```
# wae nimos l3-l1-mappings l3-l1-mappings bgl-phys
l3-l1-mapping Samurai-1 TenGigE0/0/1/0 SIte1_IP40 OCH_PORT:Unit-2/81
!
l3-l1-mapping Samurai-1 TenGigE0/0/1/1 SIte1_IP40 OCH_PORT:Unit-2/79
# commit
```

Configure the Optical Plug-In

The optical plug-in enables collection of optical information in the network. The configuration file for the optical plug-in needs to be updated with the correct credentials and seed Layer 1 node. The optical plug-in initiates the connection to the seed node to retrieve the optical network details.

Note HTTPS support can be activated in the **config/profile.properties** file by setting https and removing http on the following key: # HTTP instead HTTPS Profile spring.profiles.active=dwdm,https. The **config/spring-jetty.xml** file contains the configuration profile for https. The port on which the restconf requests can be sent to the plugin should be configured here.

Step 1 From the CLI, edit and save the `<wae-installation-directory>/packages/optical-ctc-plugin/config/optical-ctc-plugin.properties` file with the following information:

- **network.id**—Optical network name. It must be entered as *<optical_network_name>*:network; for example, cisco:network.
- **network.nodes.vendor**—Node vendor.
- **restconf.http.port** and **restconf.https.port**—The port number where the L1 network is running. By default WAE sends requests to http port 9000 and https port 8445 to collect and deploy information to the optical network.
- **network.discovery.start.node**—The Layer 1 IP address of the discovery seed node.
- **network.discovery.start.node.login**—The ID to log in to the seed node.
- **network.discovery.start.node.password**—The password to access the seed node. For added security, the credentials can be entered as device credentials (create authgroups) in WAE and then selected during optical agent configuration.

- **network.discovery.inactivity.period**—Time (in milliseconds) in which the discovery times out if there is no access to the network.

Note This information is also needed later when configuring the multi-layer collection (optical-nimo).

Step 2 Run the optical plug-in script:

```
# ./run.sh
```

Example

Optical plug-in configuration file example:

```
network.id=cisco:network
network.nodes.vendor=cisco

restconf.http.port=9000

network.discovery.start.node=10.89.204.17
network.discovery.start.node.login=CISCO15
network.discovery.start.node.password=pwcisco+11
network.discovery.inactivity.period=5000
```

Configure Multi-Layer Collection

In this task, you will do the following

- Configure an L3 and L1 network model. Do not run these collections until the aggregator configuration is complete.
- Select, configure, and run the optical agent.
- Configure the aggregator to consolidate L3 and L1 collections in a final network model.
- Run L3 and L1 collections.
- Confirm that L3 and L1 collections are consolidated in the final network model.

Before you begin

Confirm that you have completed all the preliminary tasks in Multi-Layer Collection Workflow, on page 78.

Step 1 Configure, but do not run collection, an L3 IGP topology network model with the following interface options set to **true**:

- lag
- get-physical-ports

Note For more information, see IGP Topology Collection, on page 46.

Step 2 Navigate to **/wae:wae/agents** and click **optical-agents**.

Step 3 Click Add (+) and enter an agent name.

Step 4 Select **Cisco** from the agent-type drop-down list.

Step 5 Click **cisco-optical-agent** and click the **optical-plugin-config** tab.

Note The optical agent name must match the "network.id" that was configured in the optical plug-in configuration file. See Configure the Optical Plug-In, on page 79.

Step 6 Enter the following values:

This information was configured in the optical plug-in configuration file.

- **optical-plugin-ip**—Enter the IP address of where the optical plug-in is installed.
- **optical-plugin-port**—Enter the http or https port where the optical plug-in is running.
- **optical-plugin-protocol**—Select http or https.
- **seed-node**—Enter the seed node IP address.
- **seed-node-access**—Select the appropriate access group.

Note Click the **advanced** tab for more options such as number and interval length of retries, data recording (if net-recorder is set to record, the file will be saved in the directory where net-record-file is set), and connection timeout settings.

Step 7 Click the **Commit** button.

Step 8 If you plan to utilize Lambda ID mapping (where you can set whether channel ID, central frequency, or wavelength will be mapped to the lambda ID), then you must load the Lambda ID configuration file. Enter the following command:

```
# ncs_load -lmj /wae/agents/optical-agents/optical-agent <agent-name>/lambda-mappings
```

Step 9 Navigate to the **cisco-optical-agent** tab, and click **run-optical-collection** > **Invoke run-optical-collection**.

Step 10 Create an L1 optical collection network model:

a) Navigate to **/wae:networks**.
b) Click the plus (+) sign and enter the L1 network model name used in. We recommend a unique name that contains the source network and NIMO names; for example, networkABC_L1.
c) Click **Add**.
d) Click the **nimo** tab.
e) From the **Choice - nimo-type** drop-down list, choose **optical-nimo**.
f) Click **optical-nimo** and enter the following information:

- **source-network**—Choose the applicable network model that contains topology information collected using one of the topology NIMOs.
- **network-access**—Choose a network access group that was previously configured.

Step 11 Click the **optical-agents** tab and add any agents that were created.

Step 12 Click the **advanced** tab to configure any of the following:

- **cfg-parse-agent**—Choose the configuration parse agent name if it was used for L1-L3 mapping.
- **lag**—Choose true if using the configuration parse agent.
- **lmp**—Choose true if using the configuration parse agent.
- **retain-amplifiers**—Choose true if you want to include amplifiers as part of the collection.
- **map-lambdas**—If set to true, a user table is created that displays the lambda mapping values (lambda ID, channel ID, central frequency, and wavelength). These values can then be pulled and displayed in WAE Design. Cisco recommends to set this value to true when collecting information from a network with L1 links supporting 96 channels.

- **map-lambda-id-to**—Set whether channel ID, central frequency, or wavelength will be mapped to the Lambda ID. When the plan file is imported, this is the field that WAE Design will display in the L1 Circuit Paths table.
- **use-configured-l3-l1-mapping**—Choose true if you manually configured the l3-l1 mapping (see Configure L3-L1 Mapping Information, on page 78). Choose false if you are running the configuration parsing agent.
- **l3-l1-mapping**—Choose the l3-l1 mapping group that you configured earlier. If you are running the configuration parsing agent, do not choose anything.

Step 13 Click the **Commit** button.

Step 14 Configure the aggregator to consolidate the L1 and L3 network models you just created. See example for aggregator rules to pick the data from proper source network. To view a CLI configuration example of the rest of this procedure, see Aggregator and Multi-Layer Collection CLI Configuration Example, on page 52.

a) Create an empty network. This will be the final consolidated network model. From the Expert Mode, navigate to **/wae:networks**, click the plus (+) sign, and enter a final network model name. For example, networkABC_L3L1. Also, if you are deploying, choose the aggregator NIMO and add sources.
b) Navigate to **/wae:wae/components/dare:aggregators/aggregator** tab.
c) Click the plus (+) sign and select the multilayer network (networkABC_L3L1) you just created from the destination drop-down list.
d) From the sources tab, click **source**, and add the L1 and L3 network models you want to combine the collections from.
e) Click **Commit**.

Step 15 Run the L3 collection.

a) Navigate to **/wae:networks/network/<*network-name*> nimo/topo-igp-nimo**.
b) From the topo-igp-nimo tab, click **run-collection**.

Step 16 Run the L1 collection.

a) Navigate to **/wae:networks/network/<*network-name*> nimo/optical-nimo**.
b) From the optical-nimo tab, select the L1 source network and click **build-optical-topology**.

Step 17 To verify that the merge was successful, you can open the network from WAE Design (**File** > **Open from** > **WAE Automation Server** and select the final network model).

Example

The following is an example of the aggregator rules (from WAE CLI):

```
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/area source-ownership source [not(../ml-collected)]
source l3-source operation collect_only
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/area source-ownership source [../ml-collected] source
 l1-source operation collect_deploy
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/igp-metric source-ownership source
[not(../ml-collected)] source l3-source operation collect_only
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/igp-metric source-ownership source [../ml-collected]
 source l1-source operation collect_deploy
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/ip-addresses/ip-address/ip-address source-ownership
source [not(../ml-collected)] source l3-source operation collect_only
networks network final-network nimo aggregator rules rule
```

```
model/nodes/node/interfaces/interface/ip-addresses/ip-address/ip-address source-ownership
source [../ml-collected] source l1-source operation collect_only
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/ip-addresses/ip-address/prefix-length source-ownership
 source [not(../ml-collected)] source l3-source operation collect_only
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/ip-addresses/ip-address/prefix-length source-ownership
 source [../ml-collected] source l1-source operation collect_deploy
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/isis-level source-ownership source
[not(../ml-collected)] source l3-source operation collect_only
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/isis-level source-ownership source [../ml-collected]
 source l1-source operation collect_deploy
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/name source-ownership source [not(../ml-collected)]
source l3-source operation collect_only
networks network final-network nimo aggregator rules rule
model/nodes/node/interfaces/interface/name source-ownership source [../ml-collected] source
 l1-source operation collect_deploy
networks network final-network nimo aggregator rules rule
model/nodes/node/ports/port/interface/interface-name source-ownership source
[not(../../l1-port/l1-port-name)] source l3-source operation collect_only
networks network final-network nimo aggregator rules rule
model/nodes/node/ports/port/interface/interface-name source-ownership source
[../../l1-port/l1-port-name] source l1-source operation collect_deploy
```

where `final-network` is the final network, `l1-source` is the optical-nimo network and `l3-source` is the topo-igp-nimo network. Note that the above rules for interfaces are based on the custom `ml-collected` field populated only by optical nimo. These rules are applicable only when you've only two source networks for aggregator (topo-igp-nimo network and optical-nimo network). For more information on rules, contact your Cisco WAE representative.

What to do next

You can open the plan file and view L1 and L3 topology in WAE Design.

CHAPTER 8

NetFlow Data Collection

This section contains the following topics:

NetFlow Data Collection

WAE can collect and aggregate exported NetFlow and related flow measurements. These measurements can be used to construct accurate demand traffic data for WAE Design. Flow collection provides an alternative to the estimation of demand traffic from interfaces, LSPs, and other statistics using Demand Deduction. NetFlow gathers information about the traffic flow and helps to build traffic and demand matrix. Importing flow measurements is particularly useful when there is full or nearly full flow coverage of a network's edge routers. Additionally, it is beneficial when accuracy of individual demands between external autonomous systems (ASes) is of interest.

Network data collected separately by NIMOs, including topology, BGP neighbors, and interface statistics, is combined with the flow measurements to scale flows and provide a complete demand mesh between both external autonomous systems and internal nodes.

WAE gathers the following types of data to build a network model with flows and their traffic measurements aggregated over time:

- Flow traffic using NetFlow, JFlow, CFlowd, IPFIX, and Netstream flows

- Interface traffic and BGP peers over SNMP

- BGP path attributes over peering sessions

NetFlow Collection Architectures

There are two types of flow collection architectures:

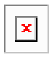
Note The collection architecture to deploy depends on the measured or estimated rate of NetFlow traffic export from the network in Mbps or fps.

- Centralized NetFlow (CNF)—Typically used for small to medium networks. This is a single-server architecture.
- Distributed NetFlow (DNF)—Typically used for larger networks. This architecture consists of a JMS broker, master, and agents.

CNF Collection

The following figure shows the workflow for collecting and computing flow data in CNF. The WAE Collector CLI tools, `flow_manage` and `flow_get`, integrate with an external configuration file and the NIMO collection process, respectively. Flow-based demands and demand traffic are passed to the WAE YANG run-time system.

Figure 3: Centralized Collection and Demand Creation



- **`flow_manage`**—This CLI tool configures network connectivity and manages the collection server, including starting, stopping and configuring the flow collection process. It uses input from the <NodeFlowConfigs> table from a configuration file to generate configuration information, which it then sends to the flow collection server.
- **Flow collection server**—This background process receives configuration information from `flow_manage`, which it uses to configure the collection server and receive flow data and BGP attributes. The collection server then aggregates this data and forwards the microflows file to the `flow_get` tool.
- **`flow_get`**—This CLI tool is configured inside the `nimo_flow_get.sh script` and is executed within the external-executable-nimo. It reads flow data (microflows file) from the collection server, produces NetFlow demands and demand traffic data, and inserts this data into the WAE YANG run-time database. In addition to producing demand and traffic data, `flow_get` also produces inter-AS (IAS) flow files.

Note In production networks, do not use -log-level=INFO | DEBUG | TRACE for `flow_get`.

DNF Collection

The following figures show the DNF architecture and the DNF workflow. In this architecture, each set of network devices exports flow data to a corresponding collection server. The DNF cluster performs flow computation so that each agent is responsible for the flow computation of its corresponding flow collection server that runs the flow collector. The master node aggregates this information and passes it back to `flow_collector_ias`.

Figure 4: DNF Architecture

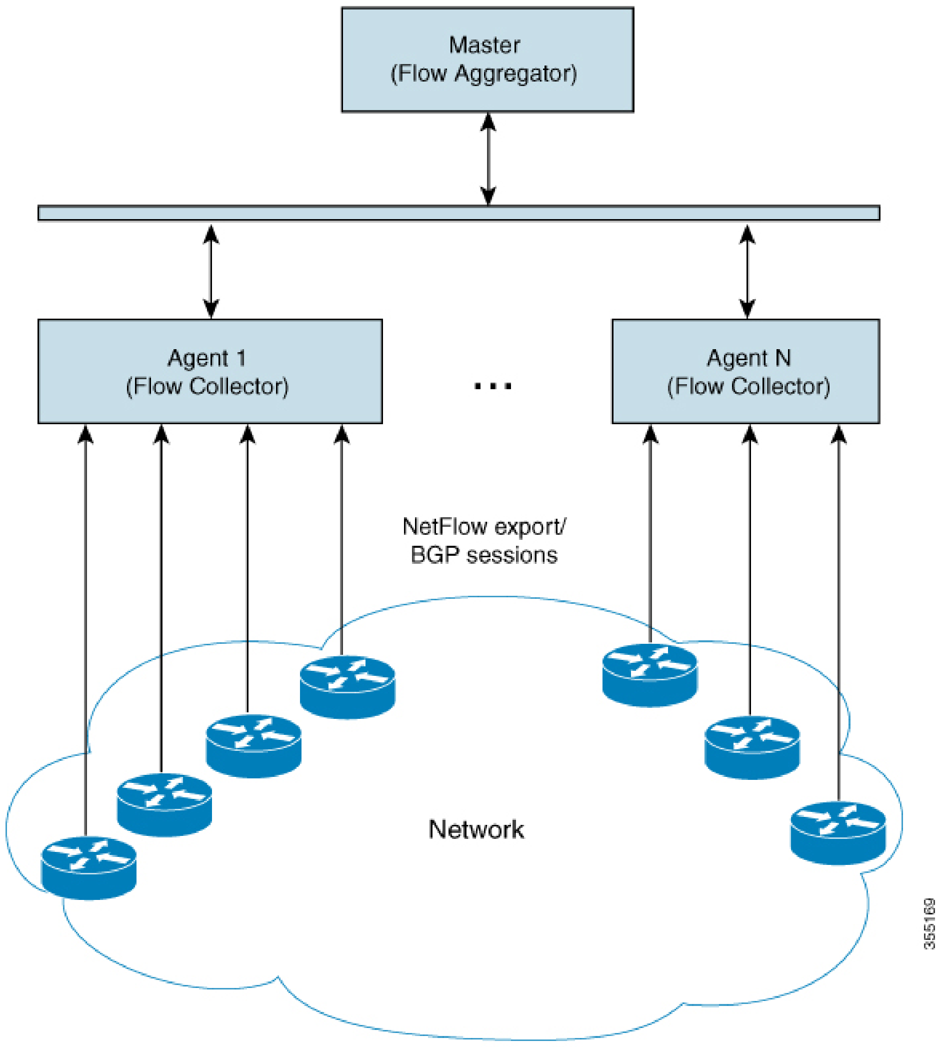


Figure 5: DNF Collection Workflow

- **`flow_cluster_manage`**—This CLI tool is used to configure and get status from the cluster. It takes a cluster configuration file and sends the configuration to the cluster. For more information, see Use the DNF Configuration File (Run flow_cluster_manage), on page 102.

  A REST API is also available to configure and request status from the cluster as an alternative to using `flow_cluster_manage`. For more information, see the API documentation from one of the following locations:

  - `<`*`wae-installation-directory`*`>docs/api/netflow/distributed-netflow-rest-api.html`
  - http://<*master-IP-address*>:9090/api-doc For example, to get the cluster configuration:

    For example, to get the cluster configuration:

```
curl -X GET http://localhost:9090/cluster-config > config-file-1
```

For example, to set the cluster configuration:

```
curl -X PUT http://localhost:9090/cluster-config @config-file-2
```

For example, to get the cluster status:

```
curl -X GET http://localhost:9090/cluster-status > config-file-1
```

- **`flow_cluster_master`**—The master service collects all flow data results from all the agents and aggregates the data, which is sent back to `flow_collector_ias`. For more information, see Master and Agents, on page 94.
- **`flow_cluster_agent`**—The agent service manages and tracks the status of the associated flow collector. Each agent receives and computes the flow data from its corresponding collection server.
- **`flow_cluster_broker`**—(not shown in diagram) The JMS broker service allows communication between all components within the architecture, including master and agents. For more information, see Java Message Server (JMS) Broker, on page 94.
- **`flow_collector_ias`**—This CLI tool, which is configured inside the `nimo_flow_collector_ias_and_dmd.sh` file and is executed within the external-executable-nimo, receives the flow data from the master and produces the IAS flows file. For more information, see Configure flow_collector_ias and flow_collector_dmd, on page 104.
- **`flow_collector_dmd`**—This CLI tool sends NetFlow demands and demand traffic to the WAE YANG run-time database. This is configured inside the `nimo_flow_collector_ias_and_dmd.sh` file and is executed within the external-executable-nimo.

Note In production networks, do not use -log-level=INFO | DEBUG | TRACE for `flow_collector_ias` or `flow_collector_dmd`.

Centralized NetFlow Configuration Workflow

To configure CNF and start collection:

Note Unless stated otherwise, do not change permissions on files that were deployed during WAE installation.

Step 1 Confirm that the CNF NetFlow Requirements , on page 89 are met.

Step 2 Prepare the Operating System for CNF, on page 89

Step 3 Create the CNF Configuration File, on page 90

Step 4 Use the CNF Configuration File (Run flow_manage), on page 91

Step 5 Configure CNF Collection, on page 91

a) Configure flow_get, on page 91

b) Configure the external-executable-nimo for CNF, on page 92

CNF NetFlow Requirements

For system requirements, see the *Cisco WAE System Requirements* document.

Licensing

Confirm with your Cisco WAE representative that you have the correct licenses for getting flow and flow demands when using the `flow_manage` and `flow_get` tools.

Prepare the Operating System for CNF

To prepare the OS for CNF, run the following `flow_manage` command from the WAE CLI:

```
sudo -E ./flow_manage -action prepare-os-for-netflow
```

The `prepare-os-for-netflow` option does the following:

- Uses the `setcap` command to allow non-root users limited access to privileged ports (0-1023). This is necessary when configuring the flow collector to use a port under 1024 to listen to BGP messages.
- Configures the OS instance to reserve up to 15,000 of file descriptors to account for the large number of temporary files that may be produced by `flow_get` in a CNF architecture.

Note After executing this command, you must reboot the server.

NetFlow Collection Configuration

The flow collection process supports IPv4 and IPv6 flows captured and exported by routers in the ingress direction. It also supports IPv4 and IPv6 iBGP peering.

Routers must be configured to export flows to and establish BGP peering with the flow collection server. Note the following recommendations:

- NetFlow v5, v9, and IPFIX datagram export to the UDP port number of the flow collection server, which has a default setting of 2100. Export of IPv6 flows requires NetFlow v9 or IPFIX.
- Configure the flow collection server on the routers as an iBGP route reflector client so that it can send BGP routes to edge or border routers. If this is not feasible, configure a router or route server that has a complete view of all relevant routing tables.
- Configure the source IPv4 address of flow export data grams to be the same as the source IPv4 address of iBGP messages if they are in the same network address space.
- Explicitly configure the BGP router ID.
- Configure static routing.

- If receiving BGP routes, the maximum length of the BGP **`AS_path`** attribute is limited to three hops. The reason is to prevent excessive server memory consumption, considering that the total length of BGP attributes, including **`AS_path`**, attached to a single IP prefix can be very large (up to 64 KB).

Create the CNF Configuration File

The <NodeFlowConfigs> table contains basic node configuration information used by the `flow_manage` tool when generating configuration information that it passes to the flow collection server. Thus, prior to executing `flow_manage`, you must construct this table as follows:

- Use a tab or comma delimited format.
- Include one row per node (router) from which you are collecting flow data.
- Enter contents described in the following table for each of these nodes. The BGP columns are required only if collecting BGP information.

Table 1: <NodeFlowConfigs> Table Columns

Column	Description
Name	Node name
SamplingRate	Sampling rate of the packets in exported flows from the node. For example, if the value is 1,024, then one packet out of 1,024 is selected in a deterministic or random manner.
FlowSourceIP	IPv4 source address of flow export packets.
BGPSourceIP	IPv4 or IPv6 source address of iBGP update messages. This column is needed if the `flow_manage -bgp` option is true.
BGPPassword	BGP peering password for MD5 authentication. Use this column if the `flow_manage -bgp` option is true and if BGPSourceIP has a value.

The following is a <NodeFlowConfigs> Table example:

Name	SamplingRate	FlowSourceIP	BGPSourceIP	BGPPassword
paris-er1-fr	1024	192.168.75.10	69.127.75.10	ag5Xh0tGbd7
chicago-cr2-us	1024	192.168.75.15	69.127.75.15	ag5Xh0tGbd7
chicago-cr2-us	1024	192.168.75.15	2001:db9:8:4::2	ag5Xh0tGbd7
tokyo-br1-jp	1024	192.168.75.25	69.127.75.25	ag5Xh0tGbd7
brazilia-er1-bra	1024	192.168.75.30	2001:db8:8:4::2	ag5Xh0tGbd7

Use the CNF Configuration File (Run flow_manage)

The `flow_manage` tool starts and stops the flow collection process (pmacct), as well as reloads the configuration information stored in the <NodeFlowConfigs> table when you change it. As such, you must run it before executing the CNF collection process:

```
flow_manage -server-ip 198.51.100.1 -action start -node-flow-configs-table flowconfigs.txt
```

We recommend that you configure your operating system to automatically start and stop `flow_manage` at system start or shutdown.

The following command reloads the <NodeFlowConfigs> table in the flowconfigs.txt file to a flow collection server with an IP address of 192.168.1.3.

```
flow_manage -server-ip 198.51.100.1 -action reload -node-flow-configs-table flowconfigs.txt
```

Sample Configuration File:

```
<NodeFlowConfigs>
Name,BGPSourceIP,FlowSourceIP,BGPPassword,SamplingRate
ar1.dus.lab.test.com,1.2.3.4,1.2.3.5,bgp-secret,666
ar1.ham.lab.test.com,1.2.3.41,1.2.3.52,bgp-secret-2,667
cr1.ams.lab.test.com,1.2.3.51,1.2.3.53,bgp-secret-3,8000
<IPPrefixFiltering>
NetworkAddress
198.51.100.1/24
198.51.100.1/23
198.51.100.1/22
198.51.100.1/21
```

For more information on `flow_manage` options, navigate to *`wae-installation-directory`*`/bin` and enter **`flow_manage -help`**.

Configure CNF Collection

Configure flow_get

This CLI tool is configured inside the `<WAE_installation_directory>/etc/netflow/ansible/bash/nimo_flow_get.sh` script and is executed within the external-executable-nimo. The tool combines the data from topology NIMO network models and the flow collection server.

Before editing, change the permissions on this file:

`chmod +x nimo_flow_get.sh`

Edit the `nimo_flow_get.sh` as follows:

- **CUSTOMER_ASN**—Enter the ASN.
- **SPLIT_AS_FLOWS_ON_INGRESS**—When multiple external ASNs are connected to an IXP switch, it determines whether to aggregate traffic from all ASNs or to distribute it proportionally to MAC accounting ingress traffic. The default value is aggregate. The other value is mac-distribute.
- **ADDRESS_FAMILY**—Enter list of protocol versions to include (comma-separated entries). The default is ipv4,ipv6.

`nimo_flow_get.sh` example:

```
#!/bin/bash
# modify as needed - BEGIN
CUSTOMER_ASN=4103291
SPLIT_AS_FLOWS_ON_INGRESS=aggregate
ADDRESS_FAMILY=ipv4,ipv6
# modify as needed - END
```

For more information on `flow_get` options, see https://www.cisco.com/c/en/us/td/docs/net_mgmt/wae/6-4/platform/configuration/guide/WAE_Platform_Configuration_Guide/wp_netflow.html#pgfId-1082437 or navigate to *`wae-installation-directory`*`/bin` and enter **`flow_get -help`**.

Configure the external-executable-nimo for CNF

The external-executable-nimo runs the `nimo_flow_get.sh` script against a selected network model. In this case, you take an existing model created in WAE and append information from `nimo_flow_get.sh` to create a final network model that contains the flow data you want.

Before you begin

- You must have a source network model. This is the final network model which includes topology collection and any other NIMO collections you want to include.
- Confirm that you have already completed the preliminary tasks in Centralized NetFlow Configuration Workflow, on page 88.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that is easily identifiable; for example, networkABC_CNF_flow_get.

Step 3 Click the **nimo** tab.

Step 4 From the **Choice - nimo-type** drop-down list, choose **external-executable-nimo**.

Step 5 Click **external-executable-nimo** and select the source network.

Step 6 Click the **advanced** tab and enter the following:

- **input-file-version**—Enter **7.1**.
- **input-file-format**—Select **.pln** as the plan file format of the source network model.
- **argv**—Enter ***`<directory_path>`*`/nimo_flow_get.sh $$input $$output`**.

Step 7 To verify configuration, click **run** from the external-executable-nimo tab.

Example

If using the WAE CLI (in config mode), enter:

```
networks network <network-model-name> nimo external-executable-nimo source-network
<source-network> advanced argv nimo_flow_get.sh $$input $$output ]
admin@wae(config-network-<network-model-name>)# commit
Commit complete.
admin@wae(config-network-<network-model-name>)# exit
```

```
admin@wae(config)# exit

admin@wae# networks network <network-model-name> nimo external-executable-nimo run
```

What to do next

Once the external-executable-nimo is configured, you can schedule it to run or access the data from WAE Design.

DNF NetFlow Configuration Workflow

To configure DNF and start collection:

Note Unless stated otherwise, do not change permissions on files that were deployed during WAE installation.

Step 1 Confirm that the Distributed NetFlow Requirements, on page 93 are met.

Step 2 Set Up the DNF Cluster, on page 95

a) Modify the DNF Configuration Files, on page 95

b) Deploy DNF Cluster, on page 98

Step 3 Configure DNF Cluster, on page 99

a) Create the DNF Cluster Configuration File, on page 99

b) Use the DNF Configuration File (Run flow_cluster_manage), on page 102

Step 4 Configure DNF Collection, on page 104

a) Configure flow_collector_ias and flow_collector_dmd, on page 104

b) Configure the external-executable-nimo for DNF, on page 105

Distributed NetFlow Requirements

For system requirements, see the *Cisco WAE System Requirements* document.

In addition, the following are required for all cluster elements (master, agents, JMS Broker):

- Ansible 2.1 or later.
- Java virtual machine (JVM) has the same installation path for all elements. The java executable should be in the path readable for all users.
- A sudo SSH user with the same name in each server dedicated for the cluster (broker, master, and all the agents) must exist. Make a note of this user name because it is used in the group_vars/all Ansible file (discussed later in this section).

  WAE Planning software must be installed on a server (installation server) with the appropriate license file.
- Agent system requirements meet the same requirements needed for WAE installation.

- The flow collection process supports IPv4 and IPv6 flows captured and exported by routers in the ingress direction. It also supports IPv4 and IPv6 iBGP peering. Routers must be configured to export flows to and establish BGP peering with the flow collection server. For more information, see NetFlow Collection Configuration, on page 89

Licensing

Confirm with your Cisco WAE representative that you have the correct licenses for getting flow and flow demands when using the `flow_cluster_master`, `flow_collector_ias`, and `flow_collector_dmd` tools.

Java Message Server (JMS) Broker

Each distributed flow collection setup must have a single JMS broker instance in order for the master, agents, and client within a cluster to exchange information. All information is interchanged through the broker and enables all the components to communicate with each other. DNF supports a dedicated JMS broker.

The broker must have the following features enabled in order for all JMS clients (master, agents, and `flow_collector_ias` instances) to work:

- Out of band file messaging
- Support of obfuscated passwords in configuration files

Master and Agents

Ansible files are used to install and run DNF configuration on the JMS broker, master, and agent servers.

Master

The master node provides the following services in the cluster:

- Monitors and tracks agent status.
- Monitors and tracks the status of the last completed IAS computation.
- Aggregates IAS flow data coming from all agents back to the client.
- Handles configuration and status requests from the cluster.

Agents

Only one agent per server is supported. Agents cannot be on the WAE installation or data collection server. Each agent receives and computes flow data from its corresponding collection server.

Note You have the option to deploy only one agent in the cluster. This is an alternative to CNF for networks that are expected to expand in size or grow in traffic.

Set Up the DNF Cluster

Modify the DNF Configuration Files

If you use default WAE installation options, there are only a few mandatory parameters that must be changed. These will be noted in the applicable configuration topics. The topics described in this section assume the following:

- The master server (installation server) is where the WAE planning software has been installed and default directories are used. In particular, the configuration files used for DNF on the installation server are located in `<wae_installation_directory>/etc/netflow/ansible`.
- A dedicated JMS broker will be used in DNF configuration.
- In configuration examples, the following values are used:
  - Master and JMS broker IP address—198.51.100.10
  - Agent 1 IP address—198.51.100.1
  - Agent 2 IP address—198.51.100.2
  - Agent 3 IP address—198.51.100.3

group_vars/all

The file is located in `<WAE_installation_directory>/etc/netflow/ansible/group_vars/all`. This file is the Ansible file that contains the variable definitions that are used in the playbook files.

Edit the following options:

Option	Description
LOCAL_WAE_INSTALLATION_DIR_NAME	The local path that contains the WAE installation file.
WAE_INSTALLATION_FILE_NAME	The filename of the WAE installation file.
TARGET_JDK_OR_JRE_HOME	The full path and filename of the Oracle JRE file. All machines in the cluster (broker, master, and all the agents) should have the JRE previously installed under this variable.
LOCAL_LICENSE_FILE_PATH	The full path to the license file.
SSH_USER_NAME	The SSH user name created or used when SSH was enabled on each machine. This sudo user is used by Ansible to deploy the cluster over SSH.

For example (comments removed):

```
LOCAL_WAE_INSTALLATION_DIR_NAME: "/wae/wae-installation"
WAE_INSTALLATION_FILE_NAME: "wae-linux-v16.4.8-1396-g6114ffa.rpm"
TARGET_JDK_OR_JRE_HOME: "/usr/lib/jvm/java-1.8.0-openjdk-1.8.0_45"
LOCAL_LICENSE_FILE_PATH: "/home/user1/.cariden/etc/MATE_Floating.lic"
TARGET_SSH_USER: ssh_user
```

hosts

The file is located in `<WAE_installation_directory>/etc/netflow/ansible/hosts`. This file is the Ansible inventory file and it includes a list of all the servers in the cluster.

Only edit the corresponding IP addresses for the broker, master, and all agents. Do not edit any of the other variables. If applicable, add more agents.

For example:

```
[dnf-broker]
198.51.100.10 ansible_ssh_user={{SSH_USER_NAME}}
[dnf-master]
198.51.100.10 ansible_ssh_user={{SSH_USER_NAME}}
[dnf-agent-1]
198.51.100.1 ansible_ssh_user={{SSH_USER_NAME}}
[dnf-agent-2]
198.51.100.2 ansible_ssh_user={{SSH_USER_NAME}}
[dnf-agent-3]
198.51.100.3 ansible_ssh_user={{SSH_USER_NAME}}
```

prepare-agents.yml

This file does not need to be edited and provides the following to all specified agents:

- Allows non-root users limited access to privileged ports (0-1023). This is necessary when configuring the flow collector to use a port under 1024 to listen to BGP messages.
- Configures the OS instance to reserve up to 15,000 of file descriptors to account for the large number of temporary files that may be produced.
- Reboots all the agents.

The file is located in `<WAE_installation_directory>/etc/netflow/ansible/prepare-agents.yml`.

startup.yml

The file is located in `<WAE_installation_directory>/etc/netflow/ansible/startup.yml`.

This file is used to automatically start the broker, master, and agents. If you have more than two agents, edit this file to add more.

For example:

```
- hosts: all
roles:
- check-ansible-version
- hosts: dnf-broker
roles:
- start-broker
- hosts: dnf-master
roles:
- start-master
- hosts: dnf-agent-1
roles:
- {role: start-agent, instance: instance-1}
- hosts: dnf-agent-2
roles:
- {role: start-agent, instance: instance-2}
- hosts: dnf-agent-3
roles:
- {role: start-agent, instance: instance-3}
```

service_conf

The file is located in `<WAE_installation_directory>/etc/netflow/ansible/bash/service.conf`.

This file provides the common configuration options that are used by the broker, master, and agents.

Edit the following options:

Option	Description
jms-broker-server-name-or-ip-address	IP address of the broker.
jms-broker-jms-port	JMS port number being used for the broker.
jms-broker-http-port	HTTP port number being used for the broker.
jms-broker-username	This is used internally and does not need to be changed.

jms-broker-password	We recommend generating and using an obfuscated password. For example:
	`# ./flow_cluster_manage -action print-obfuscation`
	`type in the clear text > password-0`
	`obfuscated text: ENC(h4rWRpG54WgVZRTE90Zb/JszY4dd4CGc)`

obfuscated text	From example above: **`ENC(h4rWRpG54WgVZRTE90Zb/JszY4dd4CGc)`**
jms-broker-use-tls	To encrypt all data communication in the DFC cluster, then enter true. If set to true, there will be some performance degradation.
append-to-log-file	If appending information to the local log file, enter true.
use-flume	If using a flume server, enter true.
flume-server	Enter the IP address of the server running the flume agent. If using the flume server that is automatically installed during WAE server installation, enter the installation server IP address.
log-level	Enter logging level type: • off • activity • fatal • error • warn • notice • info • debug • trace

For example :

```
# jms
jms-broker-server-name-or-ip-address=198.51.100.10
jms-broker-jms-port=61616
jms-broker-http-port=8161
jms-broker-username=user-0
jms-broker-password=ENC(ctrG7GGRJm983M0AsPGnabwh)
jms-broker-use-tls=false

# local logging
append-to-log-file=false

# distributed logging
use-flume=true
flume-server=198.51.100.10

# default for all commands, will be superseded if specified locally in each .sh
log-level=info
```

Deploy DNF Cluster

To deploy the DNF cluster:

Step 1 Install the broker, master and agents:

```
# ansible-playbook -i hosts install.yml
```

Note The `uninstall.yml` playbook file uninstalls the files and removes the TARGET_WAE_ROOT directory, which is defined in the all file.

Step 2 Prepare and reboot the agents for DNF:

```
# ansible-playbook -i hosts prepare-agents
```

Step 3 Start the master, broker, and agents.:

```
# ansible-playbook -i hosts startup.yml
```

Note The `shutdown.yml` playbook file shuts down the master, broker, and agents.

Step 4 Confirm that the master, broker, and agents are running:

```
# ansible-playbook -i hosts list.yml
```

Step 5 After the machines reboot, you can verify if all the agents are up by executing the following command:

```
# flow_cluster_manage -active request-cluster-status
```

A successful result should list running details of the master and all agents. At the end of the result, the CLUSTER SUMMARY should look similar to the following:

```
CLUSTER SUMMARY - BEGIN
cluster all OK: false
configured size: 0
agents up: 2
daemons up: 0
agents w/wrong IDs: []
agents w/low ulimit IDs: []
computation mode: ias-in-the-background
last result time: n/a
last no-result time: n/a
```

```
max diff time: 2 ms
max diff time OK: true
CLUSTER SUMMARY - END
```

Note In the preceding example, the `agents up` lists two running agents. The `cluster all OK` field is false because the cluster has not been configured yet. This status should change after configuring the cluster.

Configure DNF Cluster

Create the DNF Cluster Configuration File

To more easily create the cluster configuration file for `flow_manage_cluster`, you can use the CNF configuration file produced from `flow_manage` as a template for the cluster configuration file.

For example:

Step 1 Produce the template configuration file:

```
${CARIDEN_HOME}/flow_manage \
-action produce-config-file \
-node-flow-configs-table <input-path> \
-cluster-config-file <output-path> \
-interval 120 \
-bgp true \
-bgp-port 10179 \
-port 12100 \
-flow-size lab \
-server-ip ::
```

where *`<input-path>`* is the path of the node configuration .txt file used in CNF (see Configure and Run the Collector Server for more information on creating this file) and *`<output-path>`* is the path where you want the resulting seed cluster configuration file to reside. Verify that the output of the seed cluster configuration file is similar to the following:

```
{
    "agentConfigMapInfo": {
        "cluster_1::instance_1":
        {
            "flowManageConfiguration":
            {
                "maxBgpdPeers": 150,
                "bgpTcpPort": 179,
                "flowType": "Netflow",
                "useBgpPeering": true,
                "outfileProductionIntervalInSecs": 900,
                "networkDeploymentSize": "medium",
                "netflowUdpPort": 2100,
                "keepDaemonFilesOnStartStop": true,
                "purgeOutputFilesToKeep": 3,
                "daemonOutputFileMaskSuffix": "%Y.%m.%d.%H.%M.%s",
                "daemonOutputDirPath":
"<user.home>/.cariden/etc/net_flow/flow_matrix_interchange",
                "daemonOutputFileMaskPrefix": "out_matrix_",
                "daemonOutputSoftLinkName": "flow_matrix_file-latest",
                "extraAggregation": [],
                "routerConfigList":
                    [
                        {
                            "name": "ar1.dus.lab.cariden.com",
                            "bGPSourceIP": "1.2.3.4",
                            "flowSourceIP": "1.2.3.5",
                            "bGPPassword": "bgp-secret",
                            "samplingRate": "666"
                        },
                        {
                            "name": "cr1.ams.lab.cariden.com",
                            "bGPSourceIP": "1.2.3.51",
                            "flowSourceIP": "1.2.3.53",
                            "bGPPassword": "bgp-secret-3",
                            "samplingRate": "8000"
                        }
                    ],
                "appendedProperties":
                {
                    "key1": "value1",
                    "key2": "value2"
                }
            }
        },
}
```

Step 2 Edit the file to include each agent configuration. Copy, paste, and edit each section as it applies to each agent in the cluster. This example shows two agents:

```
{
    "agentConfigMapInfo": {
        "cluster_1::instance_1":
        {
            "flowManageConfiguration":
            {
                "maxBgpdPeers": 150,
                "bgpTcpPort": 179,
                "flowType": "Netflow",
                "useBgpPeering": true,
                "outfileProductionIntervalInSecs": 900,
                "networkDeploymentSize": "medium",
                "netflowUdpPort": 2100,
                "keepDaemonFilesOnStartStop": true,
                "purgeOutputFilesToKeep": 3,
                "daemonOutputFileMaskSuffix": "%Y.%m.%d.%H.%M.%s",
                "daemonOutputDirPath":
"<user.home>/.cariden/etc/net_flow/flow_matrix_interchange",
                "daemonOutputFileMaskPrefix": "out_matrix_",
                "daemonOutputSoftLinkName": "flow_matrix_file-latest",
                "extraAggregation": [],
                "routerConfigList":
                    [
                        {
                            "name": "ar1.dus.lab.anyname.com",
                            "bGPSourceIP": "1.2.3.4",
                            "flowSourceIP": "1.2.3.5",
                            "bGPPassword": "bgp-secret",
                            "samplingRate": "666"
                        },
                        {
                            "name": "cr1.ams.lab.anyname.com",
                            "bGPSourceIP": "1.2.3.51",
                            "flowSourceIP": "1.2.3.53",
                            "bGPPassword": "bgp-secret-3",
                            "samplingRate": "8000"
                        }
                    ],
                "appendedProperties":
                {
                    "key1": "value1",
                    "key2": "value2"
                }
            }
        },
```

The information for the second agent starts here:

```
        "cluster_1::instance_2":
        {
            "flowManageConfiguration":
            {
                "maxBgpdPeers": 150,
                "bgpTcpPort": 179,
                "flowType": "Netflow",
                "useBgpPeering": true,
                "outfileProductionIntervalInSecs": 900,
                "networkDeploymentSize": "medium",
                "netflowUdpPort": 2100,
                "keepDaemonFilesOnStartStop": true,
                "purgeOutputFilesToKeep": 3,
                "daemonOutputFileMaskSuffix": "%Y.%m.%d.%H.%M.%s",
                "daemonOutputDirPath":
"<user.home>/.cariden/etc/net_flow/flow_matrix_interchange",
                "daemonOutputFileMaskPrefix": "out_matrix_",
                "daemonOutputSoftLinkName": "flow_matrix_file-latest",
                "extraAggregation": [],
                "routerConfigList":
                    [
                        {
                            "name": "ar1.dus.lab.anyname.com",
                            "bGPSourceIP": "5.6.7.8",
                            "flowSourceIP": "5.6.7.9",
                            "bGPPassword": "bgp-secret-2",
                            "samplingRate": "666"
                        },
                        {
                            "name": "cr1.ams.lab.anyname.com",
                            "bGPSourceIP": "5.6.7.81",
                            "flowSourceIP": "5.6.7.83",
                            "bGPPassword": "bgp-secret-4",
                            "samplingRate": "8000"
                        }
                    ],
                "appendedProperties":
                {
                    "key1": "value1",
                    "key2": "value2"
                }
            }
        },
```

Use the DNF Configuration File (Run flow_cluster_manage)

The `flow_cluster_manage` tool diagnoses and controls the distributed NetFlow collection cluster. After creating the configuration file, use `flow_cluster_manage` to send the cluster configuration file to the cluster (**`flow_cluster_manage -send-cluster-configuration`**). All flow collection processes in all agents will reload the configuration information stored in that configuration file.

Note We recommend that you configure your system to automatically start and stop `flow_cluster_master`, `flow_cluster_agent`, and `flow_cluster_broker` at system start or shutdown.

You can also use the `flow_cluster_manage` tool to retrieve cluster status. For example:

```
# flow_cluster_manage -action request-cluster-status
```

Note The cluster will take approximately a minute to take the configuration.

Sample result of cluster status:

```
CLUSTER STATUS - BEGIN

    AGENT NODE - BEGIN
        cluster ID:             cluster_1
        instance ID:            instance_1
        process ID:             15292
        start time:             2017-07-10.09:19:43.000-0700
        up time:                00d 00h 00m 40s 824ms
        unique ID:
    bc.30.5b.df.8e.b5-15292-1729199940-1499703582925-1a23cb00-ed76-4861-94f5-461dcd5b2070
        last HB received:       2017-07-10.09:20:24.004-0700
        last HB age:            00d 00h 00m 04s 779ms
        skew time:              00d 00h 00m 00s 010ms computation sequence   0
        computational model     ias-in-the-background computing IAS:         false
        ip addresses:           [128.107.147.112, 172.17.0.1,
2001:420:30d:1320:24a8:5435:2ed5:29ae, 2001:420:30d:1320:be30:5bff:fedf:8eb5,
    2001:420:30d:1320:cd72:ec61:aac8:2e72,2001:420:30d:1320:dc55:a772:de80:a73f]
       mac address:          bc.30.5b.df.8e.b5 jvm memory utilization: 4116Mb/4116Mb/3643Mb
 max opened files:      15000
        processors:             8
        daemon period:          00d 00h 15m 00s 000ms
        daemon out dir:
   /media/1TB/user1/sandboxes/git/netflow-flexible/package/linux-release/lib/ext/pmacct/insta

    nces/flow_cluster_agent_cluster_1::instance_1
        daemon process ID: 15344
        daemon is: running
        bgp port: 179
        bgp port status: up
        netflow port: 2100
        netflow port status: up
    AGENT NODE - END

    AGENT NODE - BEGIN
        cluster ID: cluster_1
        instance ID: instance_2
        process ID: 15352
        start time: 2017-07-10.09:19:49.000-0700
        up time: 00d 00h 00m 30s 748ms
        unique ID:
    bc.30.5b.df.8e.b5-15352-1729199940-1499703589727-12989336-b314-4f85-9978-242882dd16da
        last HB received: 2017-07-10.09:20:20.746-0700
        last HB age: 00d 00h 00m 08s 037ms
        skew time: 00d 00h 00m 00s 014ms
        computation sequence 0
        computational model ias-in-the-background
        computing IAS: false
        ip addresses: [128.107.147.112, 172.17.0.1,
    2001:420:30d:1320:24a8:5435:2ed5:29ae, 2001:420:30d:1320:be30:5bff:fedf:8eb5,
    2001:420:30d:1320:cd72:ec61:aac8:2e72, 2001:420:30d:1320:dc55:a772:de80:a73f]
        mac address: bc.30.5b.df.8e.b5
        jvm memory utilization: 4116Mb/4116Mb/3643Mb
        max opened files: 15000
        processors: 8
        daemon period: 00d 00h 15m 00s 000ms
        daemon out dir:
    /media/1TB/user1/sandboxes/git/netflow-flexible/package/linux-release/lib/ext/pmacct/insta

    nces/flow_cluster_agent_cluster_1::instance_2
        daemon process ID: 15414
        daemon is: running
        bgp port: 10179
        bgp port status: up
        netflow port: 12100
        netflow port status: up
    AGENT NODE - END

    MASTER NODE - BEGIN
        cluster ID: cluster_1
        instance ID: instance_id_master_unique
        process ID: 15243
        start time: 2017-07-10.09:19:34.000-0700
        up time: 00d 00h 00m 50s 782ms
        unique ID:
    bc.30.5b.df.8e.b5-15243-415138788-1499703574719-cd420a81-f74c-49d4-a216-ffeb7cde31d5
        last HB received: 2017-07-10.09:20:25.563-0700
        last HB age: 00d 00h 00m 03s 220ms
        ip addresses: [128.107.147.112, 172.17.0.1,
    2001:420:30d:1320:24a8:5435:2ed5:29ae, 2001:420:30d:1320:be30:5bff:fedf:8eb5,
    2001:420:30d:1320:cd72:ec61:aac8:2e72, 2001:420:30d:1320:dc55:a772:de80:a73f]
        mac address: bc.30.5b.df.8e.b5
        jvm memory utilization: 2058Mb/2058Mb/1735Mb
        processors: 8
    MASTER NODE - END

    CLUSTER SUMMARY - BEGIN
        cluster all OK: true
        configured size: 2
        agents up: 2
        daemons up: 2
        agents w/wrong IDs: []
        agents w/low ulimit IDs: []
        computation mode: ias-in-the-background
        last result time: n/a
        last no-result time: n/a
        max diff time: 4 ms
        max diff time OK: true
    CLUSTER SUMMARY - END

CLUSTER STATUS - END
```

The CLUSTER SUMMARY entry at the end of the result gives you a quick summary of whether or not your cluster configuration is operational. You should confirm that `cluster all OK` is true and that the `configured size`, `agents up`, and `daemons up` match the number of agents you configured. There should be no value in `agents w/wrong IDs` and `agents w/low ulimit IDs`. The `max diff` time OK should also be set to true. If this is not the case, look into the agent and master details for troubleshooting information.

For more information on `flow_manage_cluster` options, navigate to *`wae-installation-directory`*`/bin` and enter **`flow_manage_cluster -help`**.

Configure DNF Collection

Configure flow_collector_ias and flow_collector_dmd

These CLI tools are configured inside the *`<WAE_installation_directory>`*`/etc/netflow/ansible/bash/nimo_flow_collector_ias_dmd.sh` script and is executed within the external-executable-nimo. The `flow_collector_ias` and `flow_collector_dmd` tools generate demands and demand traffic with NetFlow data received from the cluster. Edit the as follows:

Before editing, change the permissions on this file:

`chmod +x nimo_flow_collector_ias_dmd.sh`

- **CUSTOMER_ASN**—Enter ASN.
- **SPLIT_AS_FLOWS_ON_INGRESS**—When multiple external ASNs are connected to an IXP switch, it determines whether to aggregate traffic from all ASNs or to distribute it proportionally to MAC accounting ingress traffic. The default value is aggregate. The other value is mac-distribute.
- **ADDRESS_FAMILY**—Enter list of protocol versions to include (comma-separated entries). The default is ipv4,ipv6.
- **WAIT_ON_CLUSTER_TIMEOUT_SEC**—Enter the number of seconds to wait before for timing out when delegating the computation of the IAS flows into the distributed cluster. The default is 60 seconds.

`nimo_flow_collector_ias_dmd.sh` example:

```
#!/bin/bash

# this script should be called from NSO's 'external executable NIMO' configuration window
# in this way:
# /path-to/nimo_flow_collector_ias_and_dmd.sh $$input $$output

# modify as needed - BEGIN
CUSTOMER_ASN=142313
SPLIT_AS_FLOWS_ON_INGRESS=aggregate
ADDRESS_FAMILY=ipv4,ipv6
WAIT_ON_CLUSTER_TIMEOUT_SEC=60
# modify as needed - END
```

For more information on `flow_collector_ias` or `flow_collector_dmd`options, navigate to *`wae-installation-directory`*`/bin` and enter **`flow_collector_ias -help`** or **`flow_collector_dmd -help`**.

Configure the external-executable-nimo for DNF

The external-executable-nimo runs the `nimo_flow_collector_ias_dmd.sh` script against a selected network model. In this case, you take an existing model created in WAE and append information from `nimo_flow_collector_ias_dmd.sh` to create a final network model that contains the flow data you want.

Before you begin

- You must have a source network model. This is the final network model which includes topology collection and any other NIMO collections you want to include.
- Confirm that you have already completed the preliminary tasks in DNF NetFlow Configuration Workflow, on page 93.

Step 1 From the Expert Mode, navigate to **/wae:networks**.

Step 2 Click the plus (+) sign and enter a network model name. We recommend a unique name that is easily identifiable; for example, networkABC_DNF_flow_ias_dmd

Step 3 Click the **nimo** tab.

Step 4 From the **Choice - nimo-type** drop-down list, choose **external-executable-nimo**.

Step 5 Click **external-executable-nimo** and select the source network.

Step 6 Click the **advanced** tab and enter the following:

- **input-file-version**—Enter **7.1**.
- **input-file-format**—Select **.pln** as the plan file format of the source network model.
- **argv**—Enter ***<directory_path>*/nimo_flow_collector_ias_dmd.sh $$input $$output**.

Step 7 To verify configuration, click **run** from the external-executable-nimo tab.

Example

If using the WAE CLI (in config mode), enter:

```
networks network <network-model-name> nimo external-executable-nimo source-network
<source-network> advanced argv nimo_flow_collector_ias_dmd.sh $$input $$output ]
admin@wae(config-network-<network-model-name>)# commit
Commit complete.
admin@wae(config-network-<network-model-name>)# exit
admin@wae(config)# exit

admin@wae# networks network <network-model-name> nimo external-executable-nimo run
```

What to do next

Once the external-executable-nimo is configured, you can schedule it to run or access the data from WAE Design.

CHAPTER 9

Automation Applications

This section contains the following topics:

Automation Applications

Automation applications rely on having real-time network models to work with. Applications get a copy of the latest network model (master model) to either run optimization against or manipulate based on what the application's purpose or function is.

You can then view the network model using WAE Design (**File** > **Open From** > **WAE Modeling Daemon**).

Bandwidth on Demand Configuration Workflow

The Bandwidth on Demand application models and predicts the impact of a new service. This application is used when provisioning new services that requires persistent bandwidth and specific IGP or TE metric demands. The application finds a path for the SR policies being delegated in the network. For more information on the Bandwidth on Demand application, see Bandwidth on Demand Application, on page 6.

This workflow describes the high-level configuration steps to configure Bandwidth on Demand and other components.

Note Before enabling Bandwidth on Demand, confirm that the Bandwidth Optimization application is not running. You cannot run both applications at the same time.

Steps	For more information, see ...
1. Configure device authgroups and SNMP groups	Configure Device Access Using the Expert Mode, on page 19.

Steps	For more information, see ...
2. Configure a network access profile	Configure Network Access, on page 20
3. Configure the XTC agent	Configuring XTC Agents Using the Expert Mode, on page 22
4. Configure the aggregator	NIMO Collection Consolidation, on page 51
5. Configure the WAE Modeling Daemon (WMD)	Configure the WAE Modeling Daemon (WMD), on page 73
6. Configure the XTC Agent to Patch (XATP) module.	Configure the XTC Agent to Patch Module, on page 74
7. Run topology and additional NIMOs.	Network Interface Modules (NIMOs), on page 43
8. Configure the Bandwidth on Demand application and SR policies.	Configure Bandwidth on Demand, on page 108
9. Open the network model.	You can get a visual network model layout using WAE Design. From WAE Design, navigate to **File** > **Open From** > **WAE Modeling Daemon** and select the final network model. **Note** After initial configuration, you may run NIMOs anytime and the Bandwidth on Demand application will update the network model.

Configure Bandwidth on Demand

Before you begin

This procedure describes the configuration options for the Bandwidth on Demand application. For the complete configuration workflow, see Bandwidth on Demand Configuration Workflow, on page 107. For an example of the entire workflow, see Initial Bandwidth on Demand CLI Configuration Example, on page 109

Note The Bandwidth Optimization application cannot run concurrently with the Bandwidth on Demand application. You must disable (`enable = false`) one of the applications in order to use the other. For information on how to properly shut down the Bandwidth Optimization application, see Shut Down Bandwidth Optimization, on page 114.

Step 1 From the Expert Mode, navigate to **/wae:wae/components/bw-on-demand:bw-on-demand** and click the **config** tab.

Step 2 Enter the following values:

- **xtc-host**—Enter the XTC host IP address. This entry determines how to connect to XTC using a REST API. This is a persistent connection that, when enabled, is ready to take delegated PCEP requests.
- **xtc-port**—Enter the XTC port.

- **keepalive**—Enter the keepalive interval. The Bandwidth on Demand application and XTC exchange keepalive messages to maintain a persistent connection. If the connection fails, the Bandwidth on Demand application shuts down, clears current state, tries to reconnect, and re-delegate the SR policies.
- **priority**—In case there are multiple Bandwidth on Demand application instances, enter the priority level for this instance. XTC will then delegate instances depending on the priority level.
- **util-threshold**—Enter the congestion constraint (in percentage).When the Bandwidth on Demand application searches a path for the policies being delegated, it will avoid any paths that may exceed the congestion utilization threshold.
- **enable**—Select true to enable the Bandwidth on Demand application.

Note For **advanced** options, contact your Cisco WAE representative.

Step 3 Click **Commit** to save the configuration.

Step 4 Configure a new SR policy with bandwidth and IGP or TE metric type on a device. See the applicable Cisco IOS XR documentation for your specific device configuration (for example, "Configure SR-TE Policies"). Device configuration example:

```
segment-routing
    traffic-eng
        policy BWOD_2TO3_IGP
            bandwidth 10000
            color 100 end-point ipv4 198.51.100.3
            candidate-paths
                preference 10
                    dynamic mpls
                        pce
                            address ipv4 198.51.100.1
                            exit
                        metric
                            type igp
```

Step 5 Open the resulting network model using WAE Design (**WAE Design** > **File** > **Open From** > **WAE Modeling Daemon**).

Initial Bandwidth on Demand CLI Configuration Example

The following is an example of an initial Bandwidth on Demand CLI configuration within a Cisco Virtual Internet Routing Lab (VIRL) test environment. After initial configuration, you may run NIMOs anytime and the Bandwidth on Demand application will update the network model.

Configure device and network discovery.

```
# config
# devices authgroups group virl_test default-map
# devices authgroups group virl_test default-map remote-name cisco
# devices authgroups group virl_test default-map remote-password cisco
# devices authgroups group virl_test default-map remote-secondary-password cisco
# devices authgroups snmp-group virl_test default-map
# devices authgroups snmp-group virl_test default-map community-name cisco
# wae nimos network-access network-access virl_test default-auth-group virl_test
# wae nimos network-access network-access virl_test default-snmp-group virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.1 auth-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.1 snmp-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.1 ip-manage
192.0.2.131
# wae nimos network-access network-access virl_test node-access 198.51.100.2 auth-group
```

```
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.2 snmp-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.2 ip-manage
192.0.2.132
# wae nimos network-access network-access virl_test node-access 198.51.100.3 auth-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.3 snmp-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.3 ip-manage
192.0.2.133
# wae nimos network-access network-access virl_test node-access 198.51.100.4 auth-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.4 snmp-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.4 ip-manage
192.0.2.134
# wae nimos network-access network-access virl_test node-access 198.51.100.5 auth-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.5 snmp-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.5 ip-manage
192.0.2.135
# wae nimos network-access network-access virl_test node-access 198.51.100.6 auth-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.6 snmp-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.6 ip-manage
192.0.2.136
# wae nimos network-access network-access virl_test node-access 198.51.100.7 auth-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.7 snmp-group
virl_test
# wae nimos network-access network-access virl_test node-access 198.51.100.7 ip-manage
192.0.2.137
```

Configure XTC agents.

```
# wae agents xtc xtc virl enabled xtc-host-ip 192.0.2.131
```

Configure the BGP network (topo-bgpls-xtc-nimo).

```
# networks network virl_bgpls nimo topo-bgpls-xtc-nimo xtc-host virl igp-protocol isis
extended-topology-discovery true backup-xtc-host virl network-access virl_test advanced
nodes remove-node-suffix virl.info
```

Configure the LSP PCEP network (lsp-pcep-xtc-nimo).

```
# networks network virl_pcep_lsp nimo lsp-pcep-xtc-nimo xtc-hosts virl xtc-host virl
# networks network virl_pcep_lsp nimo lsp-pcep-xtc-nimo source-network virl_bgpls advanced
 sr-use-signaled-name true
```

Set the network that the aggregator will write to.

```
# networks network virl_final_model
```

Configure continuous polling (traffic-poll-nimo).

```
# networks network virl_cp nimo traffic-poll-nimo network-access virl_max source-network
virl_dare iface-traffic-poller enabled
# networks network virl_cp nimo traffic-poll-nimo lsp-traffic-poller enabled
# networks network virl_cp nimo traffic-poll-nimo advanced snmp-traffic-population
```

```
scheduler-interval 0
```

Configure the aggregator to subscribe to source networks.

```
# wae components aggregators aggregator virl_final_model sources source virl_bgpls
# wae components aggregators aggregator virl_final_model sources source virl_pcep_lsp
```

Configure WMD. In this example, WMD is set up to run demand mesh and demand deduction for all applications using WMD. So, when the continuous poller updates WMD, WMD triggers demand deduction.

```
# wae components wmd config network-name virl_final_model dare dare-destination
virl_final_model
# wae components wmd config network-name virl_final_model demands add-demands true
demand-mesh-config dest-equals-source true
```

Configure the XATP module. In this example, the `connection-attempts` parameter is set to `0`. This means if WMD connection fails, the application will keep trying to connect to WMD until the connection is successful.

```
# wae components xatp config xtc-agent virl dare aggregator-network virl_dare
pcep-lsp-xtc-nimo-network virl_pcep_lsp topo-bgpls-xtc-nimo-network virl_bgpls
wae components xatp config wmd connection-attempts 0
```

Enable WMD and XATP.

```
# wae components wmd config enable true
# wae components xatp config enable true
# commit
# exit
```

Run NIMOs (network collection).

```
networks network virl_bgpls nimo topo-bgpls-xtc-nimo run-xtc-collection
networks network virl_pcep_lsp nimo lsp-pcep-xtc-nimo run-collection
```

Configure Bandwidth on Demand.

```
# configure
# wae components bw-on-demand config xtc-host 192.0.2.131 xtc-port 8080 util-threshold 90.0
# wae components bw-on-demand config advanced lsp-traffic max-simulated-requested
primary-objective min-metric private-new-lsps true
# commit
# exit
```

Open base network model using WAE Design (**WAE Design** > **File** > **Open From** > **WAE Modeling Daemon**) to compare the resulting network model (after SR policies have been configured and the Bandwidth on Demand application has been executed).

Configure SR policies on a device.

```
# configure
# segment-routing
# traffic-eng
# policy BWOD_2TO3_IGP
# bandwidth 1000
# color 100 end-point ipv4 192.0.2.132
# candidate-paths
# preference 10
# dynamic mpls
# pce
# address ipv4 192.0.2.130
# exit
# metric
# type igp
# commit
# end
```

After the SR policy configuration is committed, WMD is updated and the Bandwidth on Demand application calculates the best path given the congestion restraint and IGP metric. Open the resulting network model using WAE Design (**WAE Design** > **File** > **Open From** > **WAE Modeling Daemon**) to compare the baseline network model to the new network model.

Shut Down Bandwidth on Demand

To properly shut down the Bandwidth on Demand application, the following steps must be done in order:

Step 1 Stop Bandwidth on Demand.

```
# wae components bw-on-demand config enable false
# commit
```

Step 2 Stop the XTC Agent to Patch module.

```
# wae components xatp config enable false
# commit
```

Step 3 Stop the WAE Modeling Daemon.

```
# wae components wmd config enable false
# commit
```

Step 4 Stop XTC agents.

```
# wae agents xtc xtc <network_name> disable xtc-host-ip <xtc_ip_address>
# commit
```

Bandwidth Optimization Application Workflow

The Bandwidth Optimization application is designed to react and manage traffic as the state of the network changes. It determines if any changes in the network state will cause congestion. If so, the Bandwidth Optimization application computes the LSPs and sends them to XTC for deployment.

This workflow describes the high-level configuration steps needed to configure the Bandwidth Optimization application and other components.

Steps	For more information, see ...
1. Configure device authgroups and SNMP groups	Configure Device Access Using the Expert Mode, on page 19.
2. Configure a network access profile	Configure Network Access, on page 20

Steps	For more information, see ...
3. Configure the XTC agent	Configuring XTC Agents Using the Expert Mode, on page 22
4. Configure the aggregator	NIMO Collection Consolidation, on page 51
5. Configure the WAE Modeling Daemon (WMD)	Configure the WAE Modeling Daemon (WMD), on page 73
6. Configure the XTC Agent to Patch (XATP) module	Configure the XTC Agent to Patch Module, on page 74
7. Run topology and additional NIMOs.	Network Interface Modules (NIMOs), on page 43
8. Configure the Bandwidth Optimization application	Configure Bandwidth Optimization, on page 113
9. Open the network model	You can get a visual network model layout using WAE Design. From WAE Design, navigate to **File** > **Open From** > **WAE Modeling Daemon** and select the final network model. **Note** After initial configuration, you may run NIMOs anytime and the Bandwidth on Demand application will update the network model.

Configure Bandwidth Optimization

This procedure describes the configuration options for the Bandwidth Optimization application. For the complete configuration workflow, see Bandwidth Optimization Application Workflow, on page 112.

Before you begin

The Bandwidth Optimization application cannot run concurrently with the Bandwidth on Demand application. You must disable (`enable = false`) one of the applications in order to use the other. For information on how to properly shut down the Bandwidth on Demand application, see Shut Down Bandwidth on Demand, on page 112.

Step 1 From the Expert Mode, navigate to **/wae:wae/components/bw-opt** and click the **config** tab.

Step 2 Enter the following values:

- **enable**—Select true to enable the Bandwidth Optimization application.
- **util-threshold**—Enter a percentage that must be exceeded if optimization is to occur. The default is 100%.
- **xtc-host**—Enter the hostname or IP address of the XTC host.
- **xtc-port**—Enter the XTC host port.
- **color**—Color representing the SR policy for XTC. For more information, contact your Cisco WAE representative.

Step 3 Click **Commit** to save the configuration.

Step 4 After the tool runs, you can click **created-lsps** to view the SR LSPs that wer created for optimized routing..

Step 5 Open the resulting network model using WAE Design (**WAE Design** > **File** > **Open From** > **WAE Modeling Daemon**).

Example

CLI (in config mode) example:

```
# wae components bw-opt config color 2000 enable false threshold 90 xtc-host 192.0.2.131
xtc-port 8080
```

WAE SR Policy Limitations

When using the latest IOS-XR SR features associated with the SR Policy, the following WAE limitations exist:

- If 2 paths are given under the `candidate-paths` option, then only the first path will be considered.
- If an SR LSP is created through WAE, a default color will be set to an SR LSP.
- Multiple LSPs cannot have the same color, source, and destination.

Shut Down Bandwidth Optimization

To properly shut down the Bandwidth Optimization application, the following steps must be done in order:

Step 1 Stop Bandwidth Optimization.

```
# wae components bw-opt config enable false
# commit
```

Step 2 Stop the XTC Agent to Patch module.

```
# wae components xatp config enable false
# commit
```

Step 3 Stop the WAE Modeling Daemon.

```
# wae components wmd config enable false
# commit
```

Step 4 Stop XTC agents.

```
# wae agents xtc xtc <network_name> disable xtc-host-ip <xtc_ip_address>
# commit
```

CHAPTER 10

Scheduler Configuration

This section provides examples and instructions on how to schedule cron and subscription jobs.

Scheduler Overview

The Scheduler performs two types of scheduling jobs:

- Cron job—A time-based job scheduler that allows certain operations to run at a specific date and time. For example, you can schedule collections to run periodically.
- Subscription job—An event-based job scheduler that listens to event notifications defined by triggers from specified sources; for example, network model changes that are pushed to the Scheduler.

Configure the Scheduler

This procedure describes how to schedule cron and subscription-based jobs using the Expert Mode.

Note Scheduler configurations using the WAE Expert Mode or WAE CLI will not appear in the WAE UI. To schedule network collections using the WAE UI, see Schedule Network Collections, on page 14.

Before you begin

Configuration of any dependent actions or events must be completed before adding them to the Scheduler. For example, if you want to schedule a network topology collection to run at a specific interval or time, that network collection must already be configured before proceeding with this task.

Step 1 From the Expert Mode, navigate to **wae:wae** > **components** tab > **scheduler** > **task**.

Step 2 Click the plus (+) icon and enter a Scheduler job name.

Step 3 Click **Add**.

Step 4 Define what action to take when the Scheduler job is enabled:

a) From the **action** tab, click the plus (+) icon and enter the action name.
b) Click **Add**.
c) From the **Choice-action** drop-down list, choose **rpc**.
d) Click **rpc** and enter the path name. The path name designates the operation to invoke. For example, to invoke a network collection, enter the following path:
`/wae:networks/network{`*`<network_model_name>`*`}/nimo/`*`<nimo_name>`*`/run-collection`
e) (Optional) If certain parameters must be met to invoke the action, click the **params** tab and add the parameters in order of requirement.

Step 5 Identify what type of event(s) trigger the action. (If there are multiple triggers, the action is executed if any trigger is invoked.)

a) From the **trigger** tab, click the plus (+) icon and enter the trigger name.
b) Click **Add**.
c) From the **Choice-trigger-spec** drop-down list, choose the trigger type: **cron** or **subscription**.

Step 6 If you are configuring a subscription-based job, click the **subscription** link and do the following:

a) Enter the source path of the trigger. For example, if the source of the event is a circuit change, enter **`/wae:networks/network{`*`<network_model_name>`*`}/model/circuits/circuit`**
b) From the **subscription-type** drop-down list, choose one of the following:

- **operational**—Applies to any operational (read-only) changes, such as traffic polling. These changes are not user initiated.
- **configuration**—Applies to any configuration changes (addition, deletion, or modification in the network), such as user-initiated LSP configuration changes.

Step 7 If you are configuring a cron-based job, click the **cron** link and enter the applicable parameters that define when to run the action.

Step 8 To add more triggers, repeat the preceding steps. (If there are multiple triggers, the action is executed if any trigger is invoked.)

Step 9 Click **Commit**.

Configure a Trigger for Topology Collection to Run Example

This example configures a subscription-based job that triggers the topology collection to run. The following procedure configures the Scheduler to run BGP-LS collection when a change occurs in the network model. For more information, see BGP-LS Topology Collection Using XTC, on page 48.

Step 1 From the Expert Mode, navigate to **wae:wae** > **components** tab > **scheduler** > **task**.

Step 2 Click the plus (+) icon and enter **run-topo-bgpls** as the Scheduler job name.

Step 3 Click **Add**.

Step 4 Define what action to take when the Scheduler job is enabled:

a) From the **action** tab, click the plus (+) icon and enter **run-xtc-topo** as the action name.
b) Click **Add**.
c) From the **Choice-action** drop-down list, choose **rpc**.

d) Click **rpc** and enter the path name. The path name designates the operation to invoke. For example, to invoke a network collection, enter the following path:
`/wae:networks/network{NetworkABC_topo-bgpls-xtc-nimo}/nimo/topo-bgpls-xtc-nimo/run-collection`

Step 5 Identify what type of event triggers this action:

a) From the **trigger** tab, click the plus (+) icon and enter **xtc-objects** as the trigger name.
b) Click **Add**.
c) From the **Choice-trigger-spec** drop-down list, choose **subscription**.

Step 6 Click the **subscription** link and do the following:

a) Enter the source path (in this example, it is where the XTC link status changes):
`/wae/agents/xtc/xtc{TTE-xtc11}/pce/xtc-topology-objects/xtc-links`
b) From the **subscription-type** drop-down list, choose **operational**.

Step 7 Click **Commit**.

Example

If using the WAE CLI (in config mode), enter:

```
# wae components scheduler tasks task run-topo-bgpls action run-xtc-topo rpc path
"/wae:networks/network{NetworkABC_topo-bgpls-xtc-nimo}/nimo/topo-bgpls-xtc-nimo/run-xtc-collection"
# wae components scheduler tasks task run-topo-bgpls triggers trigger xtc-objects subscription
 node "/wae/agents/xtc/xtc{TTE-xtc11}/pce/xtc-topology-objects/xtc-links"
# wae components scheduler tasks task run-topo-bgpls triggers trigger xtc-objects subscription
 subscription-type operational
# commit
```

CHAPTER 11

WAE Administration

This section contains the following topics:

Manage Users

In WAE 7.0, all users have the administrator role. The following procedure describes how to create and delete users.

Step 1 From the WAE UI, choose **System** > **User Manager**.

Step 2 To add a user, click **+Add User** and fill in all applicable fields.

Step 3 To change a user's password:

a) From the user row, click the pencil icon.
b) Update the password fields.
c) Click **Save**.

Step 4 To delete a user:

a) From the user row, click the trash icon.

Configure Aging

By default, when a circuit, port, node, or link disappears from a network, it is permanently removed and must be rediscovered. To configure how long WAE retains these elements that have disappeared before they are permanently removed from the network, complete the following steps.

Note This is a global option that will be configured for all networks.

Step 1 From the Expert Mode, navigate to **/wae:wae** and click the **nimos** tab.

Step 2 Click **aging**.

Step 3 Enter the number of minutes that WAE must retain the elements (circuit, node, port, or link) in the appropriate fields.

Step 4 Click **Commit**.

wae.conf

`wae.conf` is an XML configuration file that is formally defined by a YANG model, `tailf-ncsconfig.yang`. This YANG file is included in the WAE distribution, as is a commented `wae.conf.example` file.

The `wae.conf` file controls the baseline of the WAE run time. You can change certain configuration parameters in the `wae.conf` file; for example, you can change the default port that WAE runs on (port 8080) to another port.

Whenever you start or reload the WAE daemon, it reads its configuration from `./wae.conf` or *<waeruntime-directory>*`/etc/wae.conf`.

The following example shows *<waeruntime-directory>*`/etc/wae.conf` contents:

```
<!-- -*- nxml -*- -->
<!-- Example configuration file for wae. -->

<ncs-config xmlns="http://tail-f.com/yang/tailf-ncs-config">

  <!-- WAE can be configured to restrict access for incoming connections -->
  <!-- to the IPC listener sockets. The access check requires that -->
  <!-- connecting clients prove possession of a shared secret. -->
  <ncs-ipc-access-check>
    <enabled>false</enabled>
    <filename>${NCS_DIR}/etc/ncs/ipc_access</filename>
  </ncs-ipc-access-check>

  <!-- Where to look for .fxs and snmp .bin files to load -->

  <load-path>
    <dir>./packages</dir>
    <dir>${NCS_DIR}/etc/ncs</dir>

    <!-- To disable northbound snmp altogether -->
    <!-- comment out the path below -->
    <dir>${NCS_DIR}/etc/ncs/snmp</dir>
  </load-path>

  <!-- Plug and play scripting -->
  <scripts>
    <dir>./scripts</dir>
    <dir>${NCS_DIR}/scripts</dir>
  </scripts>

  <state-dir>./state</state-dir>

  <notifications>
    <event-streams>

      <!-- This is the builtin stream used by WAE to generate northbound -->
      <!-- notifications whenever the alarm table is changed. -->
      <!-- See tailf-ncs-alarms.yang -->
      <!-- If you are not interested in WAE northbound netconf notifications -->
      <!-- remove this item since it does consume some CPU -->
      <stream>
        <name>wae-alarms</name>
        <description>WAE alarms according to tailf-ncs-alarms.yang</description>
        <replay-support>false</replay-support>
        <builtin-replay-store>
          <enabled>false</enabled>
          <dir>./state</dir>
          <max-size>S10M</max-size>
          <max-files>50</max-files>
        </builtin-replay-store>
      </stream>

      <!-- This is the builtin stream used by WAE to generate northbound -->
      <!-- notifications for internal events. -->
      <!-- See tailf-ncs-devices.yang -->
      <!-- Required for cluster mode. -->
      <stream>
        <name>wae-events</name>
        <description>WAE event according to tailf-ncs-devices.yang</description>
        <replay-support>true</replay-support>
        <builtin-replay-store>
          <enabled>true</enabled>
          <dir>./state</dir>
          <max-size>S10M</max-size>
          <max-files>50</max-files>
        </builtin-replay-store>
      </stream>

      <!-- This is the builtin stream used by WAE to generate northbound -->
      <!-- notifications forwarded from devices. -->
      <!-- See tailf-event-forwarding.yang -->
      <stream>
        <name>device-notifications</name>
        <description>WAE events forwarded from devices</description>
        <replay-support>true</replay-support>
        <builtin-replay-store>
          <enabled>true</enabled>
          <dir>./state</dir>
          <max-size>S10M</max-size>
          <max-files>50</max-files>
        </builtin-replay-store>
      </stream>

      <!-- This is the builtin stream used by WAE to generate northbound -->
      <!-- notifications for plan state transitions. -->
      <!-- See tailf-ncs-plan.yang -->
      <stream>
        <name>service-state-changes</name>
        <description>Plan state transitions according to
        tailf-ncs-plan.yang</description>
        <replay-support>false</replay-support>
        <builtin-replay-store>
          <enabled>false</enabled>
          <dir>./state</dir>
          <max-size>S10M</max-size>
          <max-files>50</max-files>
        </builtin-replay-store>
      </stream>
      <stream>
        <name>XtcNotifications</name>
        <description>Xtc object change notifications</description>
        <replay-support>false</replay-support>
      </stream>
    </event-streams>
  </notifications>

  <!-- Where the database (and init XML) files are kept -->
  <cdb>
    <db-dir>./ncs-cdb</db-dir>
    <!-- Always bring in the good system defaults -->
    <init-path>
      <dir>${NCS_DIR}/var/ncs/cdb</dir>
    </init-path>
  </cdb>


  <!--
      These keys are used to encrypt values of the types
      tailf:des3-cbc-encrypted-string and tailf:aes-cfb-128-encrypted-string.
      For a deployment install it is highly recommended to change
      these numbers to something random (done by WAE "system install")
  -->
  <encrypted-strings>
    <DES3CBC>
      <key1>0123456789abcdef</key1>
      <key2>0123456789abcdef</key2>
      <key3>0123456789abcdef</key3>
      <initVector>0123456789abcdef</initVector>
    </DES3CBC>

    <AESCFB128>
      <key>0123456789abcdef0123456789abcdef</key>
      <initVector>0123456789abcdef0123456789abcdef</initVector>
    </AESCFB128>
  </encrypted-strings>


  <logs>
    <syslog-config>
      <facility>daemon</facility>
      <udp>
        <enabled>false</enabled>
        <host>syslogsrv.example.com</host>
      </udp>
    </syslog-config>

    <ncs-log>
      <enabled>true</enabled>
      <file>
```

```
        <name>./logs/wae.log</name>
        <enabled>true</enabled>
      </file>
      <syslog>
        <enabled>true</enabled>
      </syslog>
    </ncs-log>

    <developer-log>
      <enabled>true</enabled>
      <file>
        <name>./logs/devel.log</name>
        <enabled>true</enabled>
      </file>
    </developer-log>
    <developer-log-level>trace</developer-log-level>

    <audit-log>
      <enabled>true</enabled>
      <file>
        <name>./logs/audit.log</name>
        <enabled>true</enabled>
      </file>
    </audit-log>

    <netconf-log>
      <enabled>true</enabled>
      <file>
        <name>./logs/netconf.log</name>
        <enabled>true</enabled>
      </file>
    </netconf-log>

    <snmp-log>
      <enabled>true</enabled>
      <file>
        <name>./logs/snmp.log</name>
        <enabled>true</enabled>
      </file>
    </snmp-log>

    <webui-browser-log>
      <enabled>true</enabled>
      <filename>./logs/webui-browser.log</filename>
    </webui-browser-log>

    <webui-access-log>
      <enabled>true</enabled>
      <dir>./logs</dir>
    </webui-access-log>


     <!-- This log is disabled by default if wae is installed using -->
    <!-- the 'system-install' flag. It consumes a lot of CPU power -->
    <!-- to have this log turned on, OTOH it is the best tool to -->
    <!-- debug must expressions in YANG models -->

    <xpath-trace-log>
      <enabled>false</enabled>
      <filename>./logs/xpath.trace</filename>
    </xpath-trace-log>

    <error-log>
      <enabled>true</enabled>
      <filename>./logs/wae-err.log</filename>
    </error-log>

  </logs>

  <ssh>
    <algorithms>
      <mac>hmac-sha1,hmac-sha2-256,hmac-sha2-512</mac>
      <encryption>aes128-ctr,aes192-ctr,aes256-ctr</encryption>
    </algorithms>
  </ssh>

  <aaa>
    <ssh-server-key-dir>${NCS_DIR}/etc/ncs/ssh</ssh-server-key-dir>

    <!-- Depending on OS - and also depending on user requirements -->
    <!-- the pam service value value must be tuned. -->

    <pam>
      <enabled>true</enabled>
      <service>common-auth</service>
    </pam>
    <external-authentication>
      <enabled>false</enabled>
      <executable>my-test-auth.sh</executable>
    </external-authentication>

    <local-authentication>
      <enabled>true</enabled>
    </local-authentication>

  </aaa>

  <!-- Hash algorithm used when setting leafs of type ianach:crypt-hash, -->
  <!-- e.g. /aaa/authentication/users/user/password -->
  <crypt-hash>
    <algorithm>sha-512</algorithm>
  </crypt-hash>

  <!-- Disable this for performance critical applications, enabling -->
  <!-- rollbacks means additional disk IO for each transaction -->
  <rollback>
    <enabled>true</enabled>
    <directory>./logs</directory>
    <history-size>50</history-size>
  </rollback>


  <cli>
    <enabled>true</enabled>

    <!-- Use the builtin SSH server -->
    <ssh>
      <enabled>true</enabled>
      <ip>0.0.0.0</ip>
      <port>2024</port>
    </ssh>

    <prompt1>\u@wae> </prompt1>
    <prompt2>\u@wae% </prompt2>

    <c-prompt1>\u@wae# </c-prompt1>
    <c-prompt2>\u@wae(\m)# </c-prompt2>
```

```
    <show-log-directory>./logs</show-log-directory>
    <show-commit-progress>true</show-commit-progress>
    <suppress-commit-message-context>maapi</suppress-commit-message-context>
    <suppress-commit-message-context>system</suppress-commit-message-context>
  </cli>

  <webui>
    <absolute-timeout>PT0M</absolute-timeout>
    <idle-timeout>PT30M</idle-timeout>
    <enabled>true</enabled>
    <transport>
      <tcp>
        <enabled>true</enabled>
        <ip>0.0.0.0</ip>
        <port>8080</port>
       <redirect>https://@HOST@:8443</redirect>
      </tcp>
      <ssl>
        <enabled>true</enabled>
        <ip>0.0.0.0</ip>
        <port>8443</port>
        <key-file>${NCS_DIR}/var/ncs/webui/cert/host.key</key-file>
        <cert-file>${NCS_DIR}/var/ncs/webui/cert/host.cert</cert-file>
      </ssl>
    </transport>

    <cgi>
      <enabled>true</enabled>
      <php>
        <enabled>false</enabled>
      </php>
    </cgi>
  </webui>

  <rest>
    <enabled>true</enabled>
  </rest>

  <restconf>
    <enabled>true</enabled>
  </restconf>

  <netconf-north-bound>
    <enabled>true</enabled>

    <transport>
      <ssh>
        <enabled>true</enabled>
        <ip>0.0.0.0</ip>
        <port>2022</port>
      </ssh>
      <tcp>
        <enabled>false</enabled>
        <ip>127.0.0.1</ip>
        <port>2023</port>
      </tcp>
    </transport>
  </netconf-north-bound>

  <!-- <ha> -->
  <!--   <enabled>true</enabled> -->
  <!-- </ha> -->

  <large-scale>
```

```
    <lsa>
      <!-- Enable Layered Service Architecture, LSA. This requires
           a separate Cisco Smart License.
      -->
      <enabled>true</enabled>
    </lsa>
  </large-scale>

</ncs-config>
```

The default values for many configuration parameters are defined in the YANG file. See wae.conf Configuration Parameters, on page 150.

LSA Configuration

The basic idea of layered service architecture (LSA) is to split a service into an upper layer and one or several lower level parts. This can be viewed as splitting the service into a customer-facing (CFS) and a resource-facing (RFS) part. The CFS code (upper-level) runs in one (or several) NSO cfs-nodes, and the RFS code (lower-level) runs in one of many NSO rfs-nodes. The rfs-nodes have each a portion of the managed devices mounted in their / devices tree and the cfs-node(s) have the NSO rfs-nodes mounted in their /devices tree.

This section assumes the following:

- You have an understanding of Cisco Network Service Orchestrator (NSO) with regards to deployment, device configuration, and LSA. See the NSO 4.4 documentation for details. Specifically, see the *NSO Getting Started* and the *NSO Layered Service Architecture* guides.
- There are one or more NSO installations (for each LSA resource-facing (RFS) node) on the machine. The device option configuration for each instance is noted in the procedure.

LSA Configuration Workflow

This workflow describes the high-level steps to configure LSA in WAE.

Step	For more information, see...
1. Install additional packages for WAE LSA configuration.	Install LSA Packages, on page 126
2. Configure WAE to support LSA.	Configure WAE for LSA, on page 127
3. Bootstrap the RFS model.	Bootstrap the RFS Model for LSA, on page 128

Install LSA Packages

Prior to configuring WAE to support layered service architecture (LSA), you must install additional WAE NSO packages and enable LSA in the ncs.conf file.

Step 1 From `<wae_installation_directory>/packages/lsa`, copy `cisco-wae-rfs` to the run/packages directory on the NSO node.

Step 2 (Optional) If using the lsp-config-nimo for LSP collection, do the following:

a) From `<wae_installation_directory>/packages/lsa`, copy `cisco-wae-lsp-rfs` to the NSO run/packages directory on the NSO node.

b) Copy the required RFS NED to the NSO node. For example:

`<wae_installation_directory>/packages/cisco-wae-lsp-rfs/cisco-wae-lsp-rfs-iosxr.`

Step 3 Ensure that the RFS LSP CONFIG NIMO package is installed. Copy `<wae_installation_directory>/cisco-wae-lsp-config-nimo/cisco-wae-lsp-config-nimo-rfs` to `<wae_run_time_directory>/packages directory on the WAE node.`

Step 4 Enable the LSA feature in the ncs.conf file:

```
<large-scale>
    <lsa>
      <!-- Enable Layered Service Architecture, LSA. This requires
           a separate Cisco Smart License.
      -->
      <enabled>true</enabled>
    </lsa>
  </large-scale>
```

Step 5 Click the **Commit** button.

Step 6 Complete the steps documented in Configure WAE for LSA, on page 127.

Configure WAE for LSA

To configure WAE to support LSA, you must configure certain device options.

This procedure assumes the following:

- You have an understanding of Cisco Network Service Orchestrator (NSO) with regards to deployment, device configuration, and LSA. See the NSO 4.4 documentation for details. Specifically, see the *NSO Getting Started* and the *NSO Layered Service Architecture* guides.
- There are one or more NSO installations (for each LSA resource-facing (RFS) node) on the machine. The device option configuration for each instance is noted in the procedure.

Step 1 In WAE, create an LSA NETCONF device for each LSA RFS node with the following configuration options:

From the Expert mode, navigate to **`/ncs:devices/device/`*`rfs_node`*** to easily view options.

- From the device tab:
  - address—IP address of LSA RFS node.
  - port—The netconf-north-bound port configured on the LSA RFS node in ncs.conf. The default is 2022, but each LSA RFS node instance could be configured to have a different port number.
  - use-lsa—Check this box to enable LSA.
- From the device-type tab:
  - netconf— Confirm that check box is checked.
  - ned-id— Select **ned:lsa-netconf.**

- From the ssh tab, click **fetch-host-keys**.
- From the state tab, select **unlocked** under the admin-state drop-down list.
- From the device tab, click **sync-from**.

Step 2 Click the **Commit** button.

Step 3 From the Expert mode, navigate to **`/wae:wae/lsa`** and add all the LSA RFS nodes you just configured in the rfs-node drop-down list.

Step 4 Click the **Commit** button.

Bootstrap the RFS Model for LSA

Bootstrapping the RFS model populates the nodes of the RFS model from available devices. It includes the vendor, operating system, and management-IP information (not LSP-related information). The information can be populated from the RFS node by invoking the `wae-rfs services wae-rfs run-collection` command. This updates the list of nodes (possibly removing non-existent nodes).

Step 1 Invoke the RFS collection from the RFS node:

```
admin@nso# wae-rfs services wae-rfs run-collection
status true
message Wae-rfs Collection done
admin@nso#
```

Step 2 Perform a device sync (`devices sync-from`) to update WAE with the new RFS model and confirm that the RFS model is populated with devices that are present on the LSA RFS node:

```
admin@wae# show running-config devices device rfs1 config wae-rfs:wae-rfs rfs-model nodes node XRv-2
devices device rfs1
 config
  wae-rfs:wae-rfs rfs-model nodes node XRv-2
   ip-manage 192.0.2.24
   platform vendor Cisco
   platform os "IOS XR"
  !
 !
!
```

Note The device model on the NSO instance is assumed to be in-sync. Sync devices (**devices sync-from**) to avoid device sync errors during deployment.

Step 3 (For LSP deployments) Do the following to populate the LSP portion of RFS model:

a) For convenience, you can populate the RFS LSP model from the CFS node through a remote action, but this may result in NED read timeout errors. To avoid the errors, disable the lsp-config-nimo `perform-sync-from` advanced option.

b) Populate the RFS LSP information of the RFS model directly from the RFS node using the following command:

```
wae-rfs services lsp-rfs run-collection
```

c) Sync devices (**devices sync-from**) again to update WAE.

What to do next

You can perform one of the following:

- LSP Collection Using NSO NEDs, on page 55
- Multi-Layer Collection, on page 77

Troubleshoot LSA

Note You can view logs in <*wae_runtime_directory*>/logs.

After LSA configuration is completed and LSP collection (lsp-config-nimo) is invoked, the following occurs:

1. The source network is copied.
2. The LSP collection from the LSA RFS network model is populated.
   a. The LSA RFS model is updated with LSP information (**`wae-rfs services lsp-rfs run-collection`**).
   b. A device sync (**`devices device`** *`<rfs_device_name>`* `sync-from`) updates the WAE device model of the RFS.
   c. The LSP portion of the WAE network model is populated from the RFS model.
   d. The newly populated LSP portion of the WAE network model is committed.
3. The service meta-data for the RFS model is updated (**`re-deploy reconcile`**).

The processing required in the above steps can trigger timeouts. Out-of-sync errors in the RFS model and timeout errors in the wae-java-vm.log file often indicate timeout errors. In general, the starting timeout values should be at least 4 seconds per device on each RFS node. If you suspect that timeout times are causing errors, change the following values as you deem necessary:

- **`lsp-config-nimo run-collection`** action timeout—The time allocated by the YANG runtime (YRT) for **`lsp-config-action`** to complete. This value is potentially specified for all actions in the YRT instance in the wae.conf file. You can edit this value: **`networks network`** *`<network_name>`* **`nimo lsp-config-nimo advanced action-timeout`** *`<timeout_value>`*
- **`wae-rfs services lsp-rfs run-collection`** action timeout—Time allocated by the RFS model for the **`lsp-rfs`** action to complete. The value is potentially specified for RFS NSO instances in the ncs.conf file. You can edit this value: **`wae-rfs services lsp-rfs advanced action-timeout`***`<timeout_value>`*

Example: Setting the timeout for 5 minutes in YRT

```
admin@wae# config
Entering configuration mode terminal
Current configuration users:
admin tcp (maapi from 127.0.0.1) on since 2017-08-09 09:46:21 terminal mode
admin@wae(config)# networks network lsp-network nimo lsp-config-nimo advanced action-timeout
 5
admin@wae(config-network-lsp-network)# top
```

```
admin@wae(config)# devices device rfs config wae-rfs:wae-rfs services lsp-rfs advanced
action-timeout 5
admin@wae(config-config)# commit
Commit complete.
admin@wae(config-config)#
```

Understand WAE CLI Logging

WAE has extensive logging functionality. WAE logs to the directory specified in the `wae.conf` file. The following are the most useful log files:

- `wae.log`—WAE daemon log; can be configured to syslog.
- `wae_err.log.1`, `wae_err.log.idx`, `wae_err.log.siz`—If the WAE daemon has a problem, this log contains debug information for support. Display the content with the command **wae --printlog wae_err.log**.
- `audit.log`—Central audit log that covers all northbound interfaces; can be configured to syslog.
- `localhost:8080.access`—All http requests to the daemon. This is an access log for the embedded web server. This file adheres to the Common Log Format, as defined by Apache and others. This log is disabled by default and is not rotated; that is, use **logrotate(8)**.
- `devel.log`—Debug log for troubleshooting user-written code. This log is enabled by default and is not rotated; that is, use **logrotate(8)**. Use this log with the `java-vm` or `python-vm` logs. The user code logs in the `vm` logs and the corresponding library logs in `devel.log`. Disable this log in production systems. Can be configured to syslog.
- `wae-java-vm.log`, `wae-python-vm.log`—Log for code running in Java or Python VMs, such as service applications. Developers writing Java and Python code use this log (in combination with `devel.log`) for debugging.
- `netconf.log`, `snmp.log`—Log for northbound agents; can be configured to syslog.
- `rollback`*NNNNN*—All WAE commits generate a corresponding rollback file. You can configure the maximum number of rollback files and file numbering in `wae.conf`.
- `xpath.trace`—XPATH is used in many places, such as XML templates. This log file shows the evaluation of all XPATH expressions. To debug XPATH for a template, use the pipe-target debug in the CLI instead.
- `ned-cisco-ios-xr-pe1.trace`—If device trace is turned on, a trace file is created for each device. The file location is not configured in `wae.conf` but is configured when device trace is turned on, such as in the CLI.

Syslog

Using BSD or IETF syslog format (RFC5424), WAE can syslog to a local or remote syslog server. You can use the `wae.conf` file to choose which logs to save to syslog: `ncs.log`, `devel.log`, `netconf.log`, or `snmp.log`.

The following example shows a common syslog configuration:

```
<syslog-config>
```

```
    <facility>daemon</facility>

    <udp>
       <enabled>false</enabled>
       <host>127.0.0.1</host>
       <port>895</port>
    </udp>

    <syslog-servers>
      <server>
        <host>127.0.0.2</host>
        <version>1</version>
      </server>
      <server>
        <host>127.0.0.3</host>
        <port>7900</port>
        <facility>local4</facility>
        </server>
     </syslog-servers>
    </syslog-config>

  <ncs-log>
    <enabled>true</enabled>
    <file>
     <name>./logs/ncs.log</name>
     <enabled>true</enabled>
    </file>
    <syslog>
      <enabled>true</enabled>
    </syslog>
  </ncs-log>
```

Syslog Messages and Formats

The following table lists WAE syslog messages and formats.

Symbol	Format String	Comment
DAEMON_DIED	"Daemon ~s died"	An external database daemon closed its control socket.
DAEMON_TIMEOUT	"Daemon ~s timed out"	An external database daemon did not respond to a query.
NO_CALLPOINT	"no registration found for callpoint ~s of type=~s"	ConfD tried to populate an XML tree, but no code had registered under the relevant callpoint.
CDB_DB_LOST	"CDB: lost DB, deleting old config"	CDB found its data schema file but not its data files. CDB recovered by starting from an empty database.
CDB_CONFIG_LOST	"CDB: lost config, deleting DB"	CDB found its data files but not its schema file. CDB recovered by starting from an empty database.

Symbol	Format String	Comment
CDB_UPGRADE_FAILED	"CDB: Upgrade failed: ~s"	Automatic CDB upgrade failed, meaning the data model was changed in a way that is not supported.
CDB_INIT_LOAD	"CDB load: processing file: ~s"	CDB is processing an initialization file.
CDB_OP_INIT	"CDB: Operational DB re-initialized"	The operational database was deleted and reinitialized because of an upgrade or a corrupt file.
CDB_CLIENT_TIMEOUT	"CDB client (~s) timed out, waiting for ~s"	A CDB client failed to answer within the timeout period and was disconnected.
INTERNAL_ERROR	"Internal error: ~s"	A ConfD internal error occurred and should be reported to Cisco technical support.
AAA_LOAD_FAIL	"Failed to load AAA: ~s"	Failed to load the AAA data because the external database is misbehaving or AAA is mounted or populated badly.
EXTAUTH_BAD_RET	"External auth program (user=~s) ret bad output: ~s"	Authentication is external and the external program returned badly formatted data.
BRIDGE_DIED	"confd_aaa_bridge died - ~s"	ConfD is configured to start the confd_aaa_bridge and the C program died.
PHASE0_STARTED	"ConfD phase0 started"	ConfD has started its start phase 0.
PHASE1_STARTED	"ConfD phase1 started"	ConfD has started its start phase 1.
STARTED	"ConfD started vsn: ~s"	ConfD has started.
UPGRADE_INIT_STARTED	"Upgrade init started"	In-service upgrade initialization started.
UPGRADE_INIT_SUCCEEDED	"Upgrade init succeeded"	In-service upgrade initialization succeeded.
UPGRADE_PERFORMED	"Upgrade performed"	In-service upgrade was performed but not yet committed.
UPGRADE_COMMITTED	"Upgrade committed"	In-service upgrade was committed.
UPGRADE_ABORTED	"Upgrade aborted"	In-service upgrade was aborted.
CONSULT_FILE	"Consulting daemon configuration file ~s"	ConfD is reading its configuration file.
STOPPING	"ConfD stopping (~s)"	ConfD is stopping (for example, due to **confd --stop**).
RELOAD	"Reloading daemon configuration"	Initiated daemon configuration reload.
BADCONFIG	"Bad configuration: ~s:~s: ~s"	confd.conf contains bad data.
WRITE_STATE_FILE_FAILED	"Writing state file failed: ~s: ~s (~s)"	Failed to write a state file.

Symbol	Format String	Comment
READ_STATE_FILE_FAILED	"Reading state file failed: ~s: ~s (~s)"	Failed to read a state file.
SSH_SUBSYS_ERR	"ssh protocol subsys - ~s"	Client did not send the \"**subsystem**\" command correctly.
SESSION_LIMIT	"Session limit of type '~s' reached, rejected new session request"	Session limit reached; new session request was rejected.
CONFIG_TRANSACTION_LIMIT	"Configuration transaction limit of type '~s' reached, rejected new transaction request"	Configuration transaction limit reached; new transaction request was rejected.
ABORT_CAND_COMMIT	"Aborting candidate commit, request from user, reverting configuration"	Aborting candidate commit due to user request. Reverting the configuration.
ABORT_CAND_COMMIT_TIMER	"Candidate commit timer expired, reverting configuration"	Candidate commit timer expired; reverting configuration.
ABORT_CAND_COMMIT_TERM	"Candidate commit session terminated, reverting configuration"	Candidate commit session terminated; reverting configuration.
ROLLBACK_REMOVE	"Found half created rollback0 file - removing and creating new"	Removing and recreating a rollback0 file that was found only half created.
ROLLBACK_REPAIR	"Found half created rollback0 file - repairing"	Repairing a rollback0 file that was found only half created.
ROLLBACK_FAIL_REPAIR	"Failed to repair rollback files"	Failed to repair a rollback file.
ROLLBACK_FAIL_CREATE	"Error while creating rollback file: ~s: ~s"	An error occurred while creating a rollback file.
ROLLBACK_FAIL_RENAME	"Failed to rename rollback file ~s to ~s: ~s"	Failed to rename a rollback file.
NS_LOAD_ERR	"Failed to process namespace ~s: ~s"	System failed to process a loaded namespace.
NS_LOAD_ERR2	"Failed to process namespaces: ~s"	System failed to process a loaded namespace.
FILE_LOAD_ERR	"Failed to load file ~s: ~s"	System failed to load a file in its load path.
FILE_LOADING	"Loading file ~s"	System is starting to load a file.
SKIP_FILE_LOADING	"Skipping file ~s: ~s"	System skipped a file.
FILE_LOAD	"Loaded file ~s"	System loaded a file.
LISTENER_INFO	"~s to listen for ~s on ~s:~s"	ConfD starts or stops to listen for incoming connections.
NETCONF_HDR_ERR	"Got bad NETCONF TCP header"	The clear text header that indicates users and groups was formatted badly.

Symbol	Format String	Comment
LIB_BAD_VSN	"Got library connect from wrong version (~s, expected ~s)"	An application connecting to ConfD used a library version that does not match the ConfD version (for example, an old version of the client library).
LIB_BAD_SIZES	"Got connect from library with insufficient keypath depth/keys support (~s/ ~s, needs ~s/~s)"	An application connecting to ConfD used a library version that cannot handle the depth and the number of keys used by the data model.
LIB_NO_ACCESS	"Got library connect with failed access check: ~s"	An access check failure occurred when an application connected to ConfD.
SNMP_NOT_A_TRAP	"SNMP gateway: Non-trap received from ~s"	A UDP package was received on the trap receiving port, but it's not an SNMP trap.
SNMP_TRAP_V1	"SNMP gateway: V1 trap received from ~s"	An SNMPv1 trap was received on the trap receiving port, but forwarding v1 traps is not supported.
SNMP_TRAP_NOT_FORWARDED	"SNMP gateway: Can't forward trap from ~s; ~s"	An SNMP trap was not forwarded.
SNMP_TRAP_UNKNOWN_ SENDER	"SNMP gateway: Not forwarding trap from ~s; the sender is not recognized"	An SNMP trap was supposed to be forwarded, but the sender was not listed in confd.conf.
SNMP_TRAP_OPEN_PORT	"SNMP gateway: Can't open trap listening port ~s: ~s"	Could not open the port for listening to SNMP traps.
SNMP_TRAP_NOT_RECOGNIZED	"SNMP gateway: Can't forward trap with OID ~s from ~s; There is no notification with this OID in the loaded models"	An SNMP trap was received on the trap receiving port, but its definition is unknown.
XPATH_EVAL_ERROR1	"XPath evaluation error: ~s for ~s"	An error occurred while evaluating an xpath expression.
XPATH_EVAL_ERROR2	"XPath evaluation error: '~s' resulted in ~s for ~s"	An error occurred while evaluating an xpath expression.
CANDIDATE_BAD_FILE_FORMAT	"Bad format found in candidate db file ~s; resetting candidate"	The candidate database file has a bad format. The candidate database is reset to an empty database.
CANDIDATE_CORRUPT_FILE	"Corrupt candidate db file ~s; resetting candidate"	The candidate database file is corrupt and cannot be read. The candidate database is reset to an empty database.
MISSING_DES3CBC_ SETTINGS	"DES3CBC keys were not found in confd.conf"	DES3CBC keys were not found in confd.conf.

Symbol	Format String	Comment
MISSING_AESCFB128_ SETTINGS	"AESCFB128 keys were not found in confd.conf"	AESCFB128 keys were not found in confd.conf.
SNMP_MIB_LOADING	"Loading MIB: ~s"	The SNMP agent is loading a MIB file.
SNMP_CANT_LOAD_MIB	"Can't load MIB file: ~s"	The SNMP agent failed to load a MIB file.
SNMP_WRITE_STATE_ FILE_FAILED	"Write state file failed: ~s: ~s"	Failed to write the SNMP agent state file.
SNMP_READ_STATE_ FILE_FAILED	"Read state file failed: ~s: ~s"	Failed to read the SNMP agent state file.
SNMP_REQUIRES_CDB	"Can't start SNMP. CDB is not enabled"	CDB must be enabled before the SNMP agent can start.
FXS_MISMATCH	"Fxs mismatch, slave is not allowed"	A slave connected to a master with different fxs files.
TOKEN_MISMATCH	"Token mismatch, slave is not allowed"	A slave connected to a master with a bad authorization token.
HA_SLAVE_KILLED	"Slave ~s killed due to no ticks"	A slave node did not produce its ticks.
HA_DUPLICATE_NODEID	"Nodeid ~s already exists"	A slave arrived with a node ID that already exists.
HA_FAILED_CONNECT	"Failed to connect to master: ~s"	An attempted library to become a slave call failed because the slave could not connect to the master.
HA_BAD_VSN	"Incompatible HA version (~s, expected ~s), slave is not allowed"	A slave connected to a master with an incompatible HA protocol version.
NETCONF	"~s"	NETCONF traffic log message.
DEVEL_WEBUI	"~s"	Developer web UI log message.
DEVEL_AAA	"~s"	Developer AAA log message.
DEVEL_CAPI	"~s"	Developer C API log message.
DEVEL_CDB	"~s"	Developer CDB log message.
DEVEL_CONFD	"~s"	Developer ConfD log message.
DEVEL_SNMPGW	"~s"	Developer SNMP gateway log message.
DEVEL_SNMPA	"~s"	Developer SNMP agent log message.
NOTIFICATION_REPLAY_ STORE_FAILURE	"~_s"	A failure occurred in the built-in notification replay store.
EVENT_SOCKET_TIMEOUT	"Event notification subscriber with bitmask ~s timed out, waiting for ~s"	An event notification subscriber did not reply within the configured timeout period.

Symbol	Format String	Comment
EVENT_SOCKET_WRITE_BLOCK	"~s"	Write on an event socket was blocked for too long.
COMMIT_UN_SYNCED_DEV	"Committed data towards device ~s which is out of sync"	Data was committed toward a device with a bad or unknown sync state.
NCS_SNMP_INIT_ERR	"Failed to locate snmp_init.xml in loadpath ~s"	Failed to locate snmp_init.xml in the load path.
NCS_JAVA_VM_START	"Starting the NCS Java VM"	Starting the NCS Java VM.
NCS_JAVA_VM_FAIL	"The NCS Java VM ~s"	An NCS Java VM failure or timeout occurred.
NCS_PACKAGE_SYNTAX_ERROR	"Failed to load NCS package: ~s; syntax error in package file"	Syntax error in package file.
NCS_PACKAGE_DUPLICATE	"Failed to load duplicate NCS package ~s: (~s)"	Duplicate package found.
NCS_PACKAGE_COPYING	"Copying NCS package from ~s to ~s"	A package was copied from the load path to a private directory.
NCS_PACKAGE_UPGRADE_ ABORTED	"NCS package upgrade failed with reason '~s'"	The CDB upgrade was aborted, implying that the CDB is untouched. However, the package state changed.
NCS_PACKAGE_BAD_ NCS_VERSION	"Failed to load NCS package: ~s; requires NCS version ~s"	Bad NCS version for the package.
NCS_PACKAGE_BAD_ DEPENDENCY	"Failed to load NCS package: ~s; required package ~s of version ~s is not present (found ~s)"	Bad NCS package dependency.
NCS_PACKAGE_CIRCULAR_ DEPENDENCY	"Failed to load NCS package: ~s; circular dependency found"	Circular NCS package dependency.
CLI_CMD	"CLI '~s'"	User executed a CLI command.
CLI_DENIED	"CLI denied '~s'"	Due to permissions, a user was denied from executing a CLI command.
BAD_LOCAL_PASS	"Provided bad password"	A locally configured user provided a bad password.
NO_SUCH_LOCAL_USER	"no such local user"	A non existing local user tried to log in.
PAM_LOGIN_FAILED	"pam phase ~s failed to login through PAM: ~s"	A user failed to log in through PAM.
PAM_NO_LOGIN	"failed to login through PAM: ~s"	A user failed to log in through PAM.

Symbol	Format String	Comment
EXT_LOGIN	"Logged in over ~s using externalauth, member of groups: ~s~s"	An externally authenticated user logged in.
EXT_NO_LOGIN	"failed to login using externalauth: ~s"	External authentication failed for a user.
GROUP_ASSIGN	"assigned to groups: ~s"	A user was assigned to a set of groups.
GROUP_NO_ASSIGN	"Not assigned to any groups - all access is denied"	A user was logged in but was not assigned to any groups.
MAAPI_LOGOUT	"Logged out from maapi ctx=~s (~s)"	A management agent API (MAAPI) user was logged out.
SSH_LOGIN	"logged in over ssh from ~s with authmeth:~s"	A user logged into ConfD's built-in SSH server.
SSH_LOGOUT	"Logged out ssh <~s> user"	A user was logged out from ConfD's built-in SSH server.
SSH_NO_LOGIN	"Failed to login over ssh: ~s"	A user failed to log in to ConfD's built-in SSH server.
NOAAA_CLI_LOGIN	"logged in from the CLI with aaa disabled"	A user used the --noaaa flag to confd_cli.
WEB_LOGIN	"logged in through Web UI from ~s"	A user logged in through the web UI.
WEB_LOGOUT	"logged out from Web UI"	A web UI user logged out.
WEB_CMD	"WebUI cmd '~s'"	A user executed a web UI command.
WEB_ACTION	"WebUI action '~s'"	A user executed a web UI action.
WEB_COMMIT	"WebUI commit ~s"	A user performed a web UI commit.
SNMP_AUTHENTICATION_FAIL	"ESDNMP authentication failed: ~s"	An SNMP authentication failed.
LOGIN_REJECTED	"~s"	Authentication for a user was rejected by application callback.
COMMIT_INFO	"commit ~s"	Information about configuration changes committed to the running data store.
CLI_CMD_DONE	"CLI done"	CLI command finished successfully.
CLI_CMD_ABORTED	"CLI aborted"	CLI command aborted.
NCS_DEVICE_OUT_OF_SYNC	"NCS device-out-of-sync Device '~s' Info '~s'"	A check-sync action reported out-of-sync for a device.
NCS_SERVICE_OUT_OF_SYNC	"NCS service-out-ofsync Service '~s' Info '~s'"	A check-sync action reported out-of-sync for a service.
NCS_PYTHON_VM_START	"Starting the NCS Python VM"	Starting the NCS Python VM.

Symbol	Format String	Comment
NCS_PYTHON_VM_FAIL	"The NCS Python VM ~s"	The NCS Python VM failed or timed out.
NCS_SET_PLATFORM_ DATA_ERRORS	"NCS Device '~s' failed to set platform data Info '~s'"	The device failed to set the platform operational data at connect.
NCS_SMART_LICENSING_ START	"Starting the NCS Smart Licensing Java VM"	Starting the NCS Smart Licensing Java VM.
NCS_SMART_LICENSING_FAIL	"The NCS Smart Licensing Java VM ~s"	The NCS Smart Licensing Java VM failed or timed out.
NCS_SMART_LICENSING_ GLOBAL_NOTIFICATION	"Smart Licensing Global Notification: ~s"	Smart Licensing global notification.
NCS_SMART_LICENSING_ ENTITLEMENT_NOTIFICATION	"Smart Licensing Entitlement Notification: ~s"	Smart Licensing entitlement notification.
NCS_SMART_LICENSING_ EVALUATION_COUNTDOWN	"Smart Licensing evaluation time remaining: ~s"	Smart Licensing evaluation time remaining.
DEVEL_SLS	"~s"	Developer Smart Licensing API log message.
JSONRPC_REQUEST	"JSON-RPC: '~s' with JSON params ~s"	JSON-RPC method requested.
DEVEL_ECONFD	"~s"	Developer econfd API log message.
CDB_FATAL_ERROR	"fatal error in CDB: ~s"	CDB encountered an unrecoverable error.
LOGGING_STARTED	"Daemon logging started"	Logging subsystem started.
LOGGING_SHUTDOWN	"Daemon logging terminating, reason: ~s"	Logging subsystem terminated.
REOPEN_LOGS	"Logging subsystem, reopening log files"	Logging subsystem reopened log files.
OPEN_LOGFILE	"Logging subsystem, opening log file '~s' for ~s"	Indicate target file for certain type of logging.
LOGGING_STARTED_TO	"Writing ~s log to ~s"	Write logs for a subsystem to a specific file.
LOGGING_DEST_CHANGED	"Changing destination of ~s log to ~s"	The target log file will change to another file.
LOGGING_STATUS_CHANGED	"~s ~s log"	Notify a change of logging status (enabled/disabled) for a subsystem.
ERRLOG_SIZE_CHANGED	"Changing size of error log (~s) to ~s (was ~s)"	Notify a change of log size for an error log.
CGI_REQUEST	"CGI: '~s' script with method ~s"	CGI script requested.
MMAP_SCHEMA_FAIL	"Failed to setup the shared memory schema"	Failed to set up the shared memory schema.

Symbol	Format String	Comment
KICKER_MISSING_SCHEMA	"Failed to load kicker schema"	Failed to load the kicker schema.
JSONRPC_REQUEST_ IDLE_TIMEOUT	"Stopping session due to idle timeout: ~s"	JSON-RPC idle timeout.
JSONRPC_REQUEST_ ABSOLUTE_TIMEOUT	"Stopping session due to absolute timeout: ~s"	JSON-RPC absolute timeout.

Database Locking

This section explains the different locks that exist in WAE and how they interact.

Global Locks

The WAE management backplane keeps a lock on the data store: *running*. This lock is known as the global lock and provides a mechanism to grant exclusive access to the data store. The global lock is the only lock that can explicitly be taken through a northbound agent—for example, by the NETCONF <lock> operation—or by calling Maapi.lock().

A global lock can be taken for the entire data store, or it can be a partial lock (for a subset of the data model). Partial locks are exposed through NETCONF and MAAPI.

An agent can request a global lock to ensure that it has exclusive write access. When an agent holds a global lock, no one else can write to that data store. This behavior is enforced by the transaction engine. A global lock on running is granted to an agent if there are no other lock holders (including partial locks), and if all data providers approve the lock request. Each data provider (CDB or external data provider) has its lock() callback invoked to refuse or accept the lock. The output of **ncs --status** includes the lock status.

Transaction Locks

A northbound agent starts a user session towards the WAE management backplane. Each user session can then start multiple transactions. A transaction is either read/write or read-only.

The transaction engine has its internal locks toward the running data store. These transaction locks exist to serialize configuration updates toward the data store and are separate from global locks.

When a northbound agent wants to update the running data store with a new configuration, it implicitly grabs and releases the transactional lock. The transaction engine manages the lock as it moves through the transaction state machine. No API exposes the transactional lock to the northbound agent.

When the transaction engine wants to take a lock for a transaction (for example, when entering the validate state), it first checks that no other transaction has the lock. It then checks that no user session has a global lock on that data store. Finally, it invokes each data provider with a transLock() callback.

Northbound Agents and Global Locks

In contrast to implicit transactional locks, some northbound agents expose explicit access to global locks. The management API exposes global locks by providing Maapi.lock() and Maapi.unlock() methods (and the corresponding Maapi.lockPartial() Maapi.unlockPartial() for partial locking). Once a user session is established (or attached to), these functions can be called.

In the CLI, global locks are taken when entering different configure modes, as follows:

- **config exclusive**—Takes the running data store global lock.
- **config terminal**—Does not grab any locks.

The CLI keeps the global lock until the configure mode is exited.

The Expert Mode behaves in the same way as the CLI: it has edit tabs called **Edit private** and **Edit exclusive**, which correspond to the CLI modes described above.

The NETCONF agent translates the <lock> operation into a request for a global lock for the requested data store. Partial locks are also exposed through the partial-lock rpc.

External Data Providers and CDB

An external data provider is not required to implement the lock() and unlock() callbacks. WAE never tries to initiate the transLock() state transition toward a data provider while a global lock is taken. The reason for a data provider to implement the locking callbacks is if someone else can write to the data provider's database.

CDB ignores the lock() and unlock() callbacks (because the data provider interface is the only write interface towards it).

CDB has its own internal locks on the database. The running data store has a single write lock and multiple read locks. It is not possible to grab the write lock on a data store while there are active read locks on it. The locks in CDB exist to ensure that a reader always gets a consistent view of the data. (Confusion occurs if another user deletes configuration nodes in between calls to getNext() on YANG list entries.)

During a transaction transLock() takes a CDB read lock toward the transaction's data store and writeStart() tries to release the read lock and grab the write lock instead. A CDB external reader client implicitly takes a CDB read lock between Cdb.startSession() and Cdb.endSession(). This means that while a CDB client is reading, a transaction cannot pass through writeStart(). Conversely, a CDB reader cannot start while a transaction is in between writeStart() and commit() or abort().

The operational store in CDB does not have any locks; WAE's transaction engine can only read from it. CDB client writes are atomic per write operation.

Lock Impact on User Sessions

When a session tries to modify a data store that is locked, it fails. For example, the CLI might print:

```
admin@wae(config)# commit
Aborted: the configuration database is locked
```

Because some locks are short-lived (such as a CDB read lock), WAE is configured by default to retry the failed operation for a configurable length of time. If the data store remains locked after this time, the operation fails.

To configure the retry timeout, set the `/ncs-config/commit-retry-timeout` value in `wae.conf`.

Security

WAE requires privileges to perform certain tasks. Depending on the target system, the following tasks might require root privileges:

- Binding to privileged ports. The `wae.conf` configuration file specifies which port numbers WAE should bind(2) to. If a port number is lower than 1024, WAE usually requires root privileges unless the target operating system allows WAE to bind to these ports as a non-root user.
- If PAM is used for authentication, the program installed as `$NCS_DIR/lib/ncs/priv/pam/epam` acts as a PAM client. Depending on the local PAM configuration, this program might require root privileges. If PAM is configured to read the local `passwd` file, the program must either run as root, or be setuid root. If the local PAM configuration instructs WAE to run for example pam_radius_auth, root privileges might not be required, depending on the local PAM installation.
- If the CLI is used to create CLI commands that run executables, modify the permissions of the `$NCS_DIR/lib/ncs/priv/ncs/cmdptywrapper` program.

To run an executable as root or as a specific user, make cmdptywrapper setuid root:

```
# chown root cmdptywrapper
# chmod u+s cmdptywrapper
```

Failing that, all programs are executed as the user running the WAE daemon. If that user is root, you need not perform the chmod operations above.

Failing that, all programs are executed as the user running the confd daemon. If that user is root, you need not perform the preceding chmod operations.

For executables that run via actions, modify the permissions of the `$NCS_DIR/lib/ncs/priv/ncs/cmdwrapper` program:

```
# chown root cmdwrapper
# chmod u+s cmdwrapper
```

WAE can be instructed to terminate NETCONF over clear text TCP, which is useful for debugging (NETCONF traffic can be captured and analyzed) and when providing a local proprietary transport mechanism other than SSH. Clear text TCP termination is not authenticated; the clear text client simply tells WAE which user the session should run as. The assumption is that authentication is already done by an external entity, such as an SSH server. If clear text TCP is enabled, WAE must bind to localhost (127.0.0.1) for these connections.

Client libraries connect to WAE. For example, the CDB API is TCP-based and a CDB client connects to WAE. WAE learns which address to use for these connections through the `wae.conf` parameters `/ncs-config/ncs-ipc-address/ip` (the default address is 127.0.0.1) and `/ncs-config/ncs-ipcaddress/port` (the default port is 4565).

WAE multiplexes different kinds of connections on the same socket (IP and port combination). The following programs connect on the socket:

- Remote commands, such as **ncs --reload**.
- CDB clients.

- External database API clients.
- Management agent API (MAAPI) clients.
- The ncs_cli program.

By default, the preceding programs are considered trusted. MAAPI clients and the ncs_cli authenticate users before connecting to WAE. CDB clients and external database API clients are considered trusted and do not have to authenticate.

Because the ncs-ipc-address socket allows full, unauthenticated access to the system, it is important to ensure that the socket is not accessible from untrusted networks. You can also restrict access to the ncs-ipc-address socket by means of an access check. See Restrict Access to the IPC Port, on page 142.

Restrict Access to the IPC Port

By default, clients connecting to the IPC port are considered trusted; no authentication is required. To prevent remote access, WAE relies on the use of 127.0.0.1 for `/ncs-config/ncs-ipc-address/ip`. However, you can restrict access to the IPC port by configuring an access check.

To enable the access check, set the `wae.conf` element `/ncs-config/ncs-ipc-accesscheck/enabled` to **true**, and specify a filename for `/ncs-config/ncs-ipc-accesscheck/filename`. The file should contain a shared secret (a random-character string). Clients connecting to the IPC port must provide a challenge handshake before they are granted access to WAE functions.

Note The access permissions on this file must be restricted via OS file permissions, such that the file can only be read by the WAE daemon and client processes that are allowed to connect to the IPC port. For example, if both the daemon and the clients run as root, the file can be owned by root and have only "read by owner" permission (mode 0400). Another possibility is to create a group that only the daemon and the clients belong to, set the group ID of the file to that group, and have only "read by group" permission (mode 040).

To provide the secret to the client libraries and instruct them to use the access check handshake, set the environment variable NCS_IPC_ACCESS_FILE to the full path name of the file that contains the secret. This is sufficient for all clients mentioned above; there is no need to change the application code to enable this check.

Note The access check must be either enabled or disabled for both the daemon and the clients. For example, if the `wae.conf` element `/ncsconfig/ncs-ipc-access-check/enabled` is not set to **true**, but clients are started with the environment variable NCS_IPC_ACCESS_FILE pointing to a file with a secret, the client connections fail.

Clear WAE Operational Data

To clear WAE operational data from the database, you must delete the model, l1-model, from the respective NIMO network models. Then, delete the device tree. If your NIMO network model has layouts, delete those layouts from the NIMO network models.

The following example commands show how to clear operational data from the as64002 network model and device tree:

```
delete networks network as64002 model
delete networks network as64002 layouts
delete networks network as64002 l1-model
delete devices device *
commit
```

Back Up and Restore the WAE Configuration

With the YANG run-time framework, you can easily back up and restore the WAE configuration. We recommend that you back up the WAE configuration before starting any collection (that is, before any operational data is populated).

- To back up a WAE configuration:

  ```
  admin@wae% save /home/wae/wae-backup.cfg
  ```

  The preceding command backs up both the configuration data and the operational data. To back up only the configuration data, you must clear the operational data from the database as described in Clear WAE Operational Data, on page 143. Be careful before clearing operational data in a production environment, because all operational data is deleted.

- To restore a WAE configuration:

  ```
  [wae@wae ~]$ ncs_load -l -m -F j wae-backup.cfg
  ```

CHAPTER 12

Security

Core Security Concepts

If you are an administrator and are looking to optimize the security of your product, you should have a good understanding of the following security concepts.

HTTPS

Hypertext Transfer Protocol Secure (HTTPS) uses Secure Sockets Layer (SSL) or its subsequent standardization, Transport Layer Security (TLS), to encrypt the data transmitted over a channel. Several vulnerabilities have been found in SSL, so now supports TLS only.

Note TLS is loosely referred to as SSL often, so we will also follow this convention.

SSL employs a mix of privacy, authentication, and data integrity to secure the transmission of data between a client and a server. To enable these security mechanisms, SSL relies upon certificates, private-public key exchange pairs, and Diffie-Hellman key agreement parameters.

SSL Certificates

SSL certificates and private-public key pairs are a form of digital identification for user authentication and the verification of a communication partner's identity. Certificate Authorities (CAs), such as VeriSign and Thawte, issue certificates to identify an entity (either a server or a client). A client or server certificate includes the name of the issuing authority and digital signature, the serial number, the name of the client or server that the certificate was issued for, the public key, and the certificate's expiration date. A CA uses one or more signing certificates to create SSL certificates. Each signing certificate has a matching private key that is used to create the CA signature. The CA makes signed certificates (with the public key embedded) readily available, enabling anyone to use them to verify that an SSL certificate was actually signed by a specific CA.

In general, setting up certificates involve the following steps:

1. Generating an identity certificate for a server.
2. Installing the identity certificate on the server.

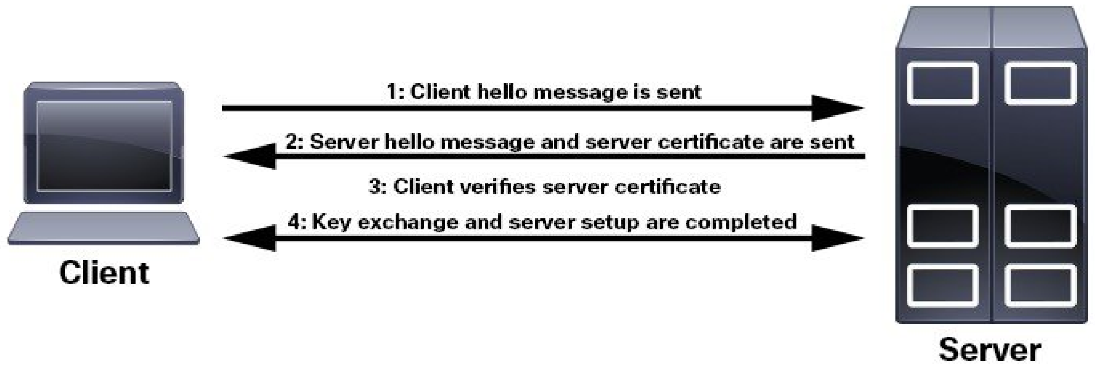
3. Installing the corresponding root certificate on your client or browser.

The specific tasks you need to complete will vary, depending on your environment.

1-Way SSL Authentication

This authentication method is used when a client needs assurance that it is connecting to the right server (and not an intermediary server), making it suitable for public resources like online banking websites. Authentication begins when a client requests access to a resource on a server. The server on which the resource resides then sends its server certificate (also known as an SSL certificate) to the client in order to verify its identity. The client then verifies the server certificate against another trusted object: a server root certificate, which must be installed on the client or browser. After the server has been verified, an encrypted (and therefore secure) communication channel is established. At this point, the server prompts for the entry of a valid username and password in an HTML form. Entering user credentials after an SSL connection is established protects them from being intercepted by an unauthorized party. Finally, after the username and password have been accepted, access is granted to the resource residing on the server.

Note A client might need to store multiple server certificates to enable interaction with multiple servers.


1: Client hello message is sent
2: Server hello message and server certificate are sent
3: Client verifies server certificate
4: Key exchange and server setup are completed
Client
Server

To determine whether you need to install a root certificate on your client, look for a lock icon in your browser's URL field. If you see this icon, this generally indicates that the necessary root certificate has already been installed. This is usually the case for server certificates signed by one of the bigger Certifying Authorities (CAs), because root certificates from these CAs are included with popular browsers.

If your client does not recognize the CA that signed a server certificate, it will indicate that the connection is not secure. This is not necessarily a bad thing. It just indicates that the identity of the server you want to connect has not been verified. At this point, you can do one of two things: First, you can install the necessary root certificate on your client or browser. A lock icon in your browser's URL field will indicate the certificate was installed successfully. And second, you can install a self-signed certificate on your client. Unlike a root certificate, which is signed by a trusted CA, a self-signed certificate is signed by the person or entity that created it. While you can use a self-signed certificate to create an encrypted channel, understand that it carries an inherent amount of risk because the identity of the server you are connected with has not been verified.

APPENDIX A

Additional WAE CLI Commands

This section contains the following topics:

Commit Flags

Commit flags modify transaction semantics. Use a commit flag when issuing a **commit** command:

```
commit <flag>
```

The following table lists some of commonly used flags.

Command	Description
and-quit	Exits to CLI operational mode after a commit.
bypass-commit-queue	Attempts to commit directly, bypassing the commit queue. This flag is relevant only when the commit queue is used by default (by the configuration item `/devices/global-settings/commit-queue/enabled-bydefault`). The operation fails if the commit queue contains entries that affect the same device(s) as the transaction to be committed.
check	Validates the pending configuration changes. Equivalent to the **validate** command.
comment | label	Adds a commit comment or label the is visible in compliance reports, rollback files, and so on.
dry-run	Validates and displays the configuration changes, but does not perform the actual commit. Neither CDB nor devices are affected. Various output formats are supported.
no-networking	Validates the configuration changes and updates the CDB, but does not update the actual devices. This is equivalent to first setting the admin state-state to southbound locked, then issuing a standard commit. In both cases the configuration changes are not committed to actual devices. If the commit implies changes, it makes the device out-of-sync.

`no-out-of-sync-check`	Commits even if the device is out-of-sync. This flag can be used in scenarios where you know the change is not in conflict with what is on the device, and you don't want to perform a **sync-from** first. Use **device compare-config** to verify the result. If the commit implies changes, it makes the device out-of-sync.
`no-revision-drop`	Fails if devices have obsolete device models. When WAE connects to a NETCONF device, the version of the device data model is discovered. Different devices in the network might have different versions. When WAE sends configurations to devices, by default it drops any configuration that only exists in later models than the device supports.
`through-commit-queue`	Although the configuration change is committed to CDB immediately, it is not committed to the actual device. Instead, to increase transaction throughput, the config change is queued for eventual commit. This enables the use of the commit queue feature for individual commit commands without enabling it by default.

All WAE command can have pipe commands. For example, the **details** pipe command provides feedback on the steps performed in the commit:

```
wae% commit | details
```

To enable debugging on all templates, use the **debug** pipe command:

```
wae% commit | debug template
```

If you use many templates during configuration, the debug output can be overwhelming. You can limit debug information to just one template, as shown in the following example for a template named *l3vpn*:

```
wae% commit | debug template l3vpn
```

Device Actions

Actions for devices can be performed globally on the `/devices` path, and for individual devices on `/devices/device/name`. Many actions are also available on device groups and device ranges.

The following table lists device actions.

Command	Description
`check-sync`	Checks if the WAE copy of the device configuration is in sync with the actual device configuration. This operation compares only a signature of the configuration from the device; it does not compare the entire configuration. The signature is implemented as a transaction-id, time-stamp, or hash-sum. The corresponding NED must support the capability. If the output says unsupported, you must use a full **`device compare-config`** command.
`check-yang-modules`	Checks if WAE and the devices have compatible YANG modules.
`clear-trace`	Clears all trace files.

`commit-queues`	Displays a list of queued commits.
`connect`	Sets up sessions to unlocked devices. This action is not used in real operational scenarios, because WAE automatically establishes connections on demand. However, this action is useful for test purposes when installing new NEDs, adding devices, and so on.
`disconnect`	Closes the session to the device.
`sync-from`	Synchronizes the WAE copy of the device configuration by reading the actual device configuration. The change is committed immediately to WAE and cannot be rolled back. If any service created a configuration on the device, the corresponding service might be out of sync. To reconcile this discrepancy, use the commands **service check-sync** and **service re-deploy**.
`sync-to`	Synchronizes the device configuration by pushing the WAE copy to the devices. (This action cannot be rolled back.)

Service Actions

Many of the preceding device operations can be combined with the option **no-networking**, which performs all updates only in the configuration database and makes the devices out of sync. The updates can be pushed to the network later. (This action is the same as setting the devices in admin-state southbound-locked.)

The following table lists service actions.

Command	Description
`check-sync`	Verifies that the service and the associated device configuration is in sync. Any differences are displayed in a chosen out-format. If configuration changes were made out-of-band, a **deep-check-sync** is required to detect an out-of-sync condition.
`deep-check-sync`	Validates whether the actual devices are configured according to the service. Use **re-deploy** to reconcile the service.
`get-modifications`	Gets the configuration data created by the service.
`re-deploy`	Reruns the service logic—taking into account all service data—and generates a diff using the device configuration in the configuration database. Sends the configuration diff to the devices. This action is useful when: • A **device sync-from** action has been performed to incorporate an out-of-band change. • Data referenced by the service—topology information, QoS policy definitions, and so on—has changed. This action is idempotent. If no configuration diff exists, nothing needs to be done. The WAE general principle of minimum change applies.
`un-deploy`	Undoes the effects of the service on the network. This action removes the configuration from the actual devices and from the WAE configuration database.

wae.conf Configuration Parameters

The following table lists the `wae.conf` configuration parameters and their type (in parentheses) and default values (in brackets). Parameters are written using a path notation to make it easier to see how they relate to each other.

Parameter	Description
/ncs-config	WAE configuration.
/ncs-config/db-mode (running) [running]	This feature is deprecated; WAE supports only running db-mode. It is not a requirement to set this leaf; it is retained only for backward compatibility.
/ncs-config/ncs-ipc-address	WAE listens by default on 127.0.0.1:4569 for incoming TCP connections from WAE client libraries, such as CDB, MAAPI, the CLI, the external database API, as well as commands from the ncs script (such as 'ncs --reload'). The IP address and port can be changed. If they are changed, all clients using MAAPI, CDB, and so on must be recompiled to handle this. **Caution** There are severe security implications involved if WAE is instructed to bind(2) to anything but localhost. Use the IP 0.0.0.0 if you want WAE to listen(2) on all IPv4 addresses.
/ncs-config/ncs-ipc-address/ip (ipv4-address | ipv6-address) [127.0.0.1]	The IP address that WAE listens on for incoming connections from the Java library.
/ncs-config/ncs-ipc-address/port (port-number) [4569]	The port number that WAE listens on for incoming connections from the Java library.
/ncs-config/ncs-ipc-extra-listen-ip (ipv4-address | ipv6-address)	This parameter can be given multiple times. It lists additional IPs to which to bind the WAE IPC listener. This is useful if you don't want to use the wildcard 0.0.0.0 address in order to never expose the WAE IPC to certain interfaces.
/ncs-config/ncs-ipc-access-check	WAE can be configured to restrict access for incoming connections to the IPC listener sockets. The access check requires that connecting clients prove possession of a shared secret.
/ncs-config/ncs-ipc-access-check/enabled (boolean) [false]	If 'true', the access check for IPC connections is enabled.
/ncs-config/ncs-ipc-access-check/filename (string)	This parameter is mandatory. *filename* is the full path to a file containing the shared secret for the IPC access check. The file should be protected via OS file permissions, such that it can only be read by the WAE daemon and client processes that are allowed to connect to the IPC listener sockets.
/ncs-config/enable-shared-memory-schema (boolean) [true]	*enabled* is either true or false. If true, a C program starts and loads the schema into shared memory (which can then be accessed by Python, for example).
/ncs-config/load-path	—

Parameter	Description
`/ncs-config/load-path/dir (string)`	This parameter can be given multiple times. The *load-path* element contains any number of *dir* elements. Each *dir* element points to a directory path on disk that is searched for compiled and imported YANG files (.fxs files) and compiled clispec files (.ccl files) during daemon startup. WAE also searches the load path for packages at initial startup, or when requested by the /packages/reload action.
`/ncs-config/state-dir (string)`	This parameter is mandatory. This is where WAE writes persistent state data. It stores a private copy of all packages found in the load path, in a directory tree rooted at 'packages-in-use.cur' (also referenced by a symlink 'packages-in-use'). It is also used for the state file 'running.invalid', which exists only if the running database status is invalid, which occurs if one of the database implementations fails during the two-phase commit protocol. It is also used for 'global.data', which is used to store data that needs to be retained across reboots.
`/ncs-config/commit-retry-timeout (xs:duration | infinity) [infinity]`	Commit timeout in the WAE back plane. This timeout controls how long the commit operation in the CLI and the JSON-RPC API try to complete the operation when another entity is locking the database; for example, when another commit is in progress or when a managed object is locking the database.
`/ncs-config/max-validation-errors (uint32 | unbounded) [1]`	Controls how many validation errors are collected and presented to the user at a time.
`/ncs-config/notifications`	Defines NETCONF northbound notification settings.
`/ncs-config/notifications/event-streams`	Lists all available notification event streams.
`/ncs-config/notifications/event-streams/stream`	Parameters for a single notification event stream.
`/ncs-config/notifications/event-streams/stream/name (string)`	The name attached to a specific event stream.
`/ncs-config/notifications/event-streams/stream/description (string)`	This parameter is mandatory. Descriptive text attached to a specific event stream.
`/ncs-config/notifications/event-streams/stream/replay-support (boolean)`	This parameter is mandatory. Signals if replay support is available for a specific event stream.
`/ncs-config/notifications/event-streams/stream/builtin-replay-store`	Parameters for the built-in replay store for this event stream. If replay support is enabled, WAE automatically stores all notifications on disk, ready to be replayed if a NETCONF manager asks for logged notifications. The replay store uses a set of wrapping log files on disk (of a certain number and size) to store the notifications. To achieve fast replay of notifications in a certain time range, the max size of each wrap log file should not be too large. If possible, use a larger number of wrap log files instead. If in doubt, use the recommended settings (see below).

Parameter	Description
/ncs-config/notifications/event-streams/ stream/builtin-replay-store/ enabled (boolean) [false]	If 'false', the application must implement its own replay support.
/ncs-config/notifications/event-streams/ stream/builtin-replay-store/dir (string)	This parameter is mandatory. The disk location for the wrapping log files.
/ncs-config/notifications/event-streams/ stream/builtin-replay-store/ max-size (tailf:size)	This parameter is mandatory. The max size of each log wrap file. The recommended setting is approximately S10M.
/ncs-config/notifications/event-streams/ stream/builtin-replay-store/ max-files (int64)	This parameter is mandatory. The max number of log wrap files. The recommended setting is around 50 files.
/ncs-config/opcache	Controls the behavior of the operational data cache.
/ncs-config/opcache/enabled (boolean) [false]	If 'true', the cache is enabled.
/ncs-config/opcache/timeout (uint64)	This parameter is mandatory. The amount of time to keep data in the cache, in seconds.
/ncs-config/hide-group	Lists any hide groups that can be unhidden. There can be zero, one, or many hide-group entries in the configuration. If a hide group does not have a hide-group entry, it cannot be unhidden using the CLI 'unhide' command. However, it is possible to add a hide-group entry to the ncs.conf file and then use **ncs -- reload** to make it available in the CLI. This can be useful to enable, for example, a diagnostics hide group that you do not want accessible even using a password.
/ncs-config/hide-group/name (string)	Name of the hide group, which should correspond to a hide group name defined in a YANG module with 'tailf:hidden'.
/ncs-config/hide-group/ password (tailf:md5-digest-string) []	A password can optionally be specified for a hide group. If no password or callback is given, the hide group can be unhidden without giving a password. If a password is specified, the hide group cannot be enabled unless the password is entered. To completely disable a hide group (that is, make it impossible to unhide it), remove the entire hide-group container for that hide group.
/ncs-config/hide-group/ callback (string)	A callback can optionally be specified for a hide group. If no callback or password is given, the hide group can be unhidden without giving a password. If a callback is specified, the hide group cannot be enabled unless a password is entered and verified. The callback receives the name of the hide group, the name of the user issuing the unhide command, and the password. Callbacks make it possible to have short-lived unhide passwords and per-user unhide passwords.

Parameter	Description
`/ncs-config/cdb`	—
`/ncs-config/cdb/db-dir (string)`	*db-dir* is the directory on disk that CDB uses for its storage and any temporary files. It is also the directory where CDB searches for initialization files.
`/ncs-config/cdb/init-path`	—
`/ncs-config/cdb/init-path/dir (string)`	This parameter can be given multiple times. The *init-path* can contain any number of *dir* elements. Each *dir* element points to a directory path that CDB searches for .xml files before looking in *db-dir*. The directories are searched in the order in which they are listed.
`/ncs-config/cdb/client-timeout (xs:duration | infinity) [infinity]`	Specifies how long CDB waits for a response before considering a client unresponsive. If a client fails to call Cdb.syncSubscriptionSocket() within the timeout period, CDB logs a syslog of this failure and then, considering the client dead, closes the socket and proceeds with the subscription notifications. If set to infinity, CDB never times out waiting for a response from a client.
`/ncs-config/cdb/subscription-replay`	—
`/ncs-config/cdb/subscription-replay/enabled (boolean) [false]`	If enabled, it is possible to request a replay of the previous subscription notification to a new CDB subscriber.
`/ncs-config/cdb/replication (async | sync) [sync]`	When CDB replication is enabled (which it is when high-availability mode is enabled; see /ncs-config/ha), the CDB configuration stores can be replicated asynchronously or synchronously. With asynchronous replication, a transaction updating the configuration is allowed to complete as soon as the updates are sent to the connected slaves. With the default synchronous replication, the transaction is suspended until the updates are completely propagated to the slaves, and the subscribers on the slaves (if any) have acknowledged their subscription notifications.
`/ncs-config/cdb/journal-compaction (automatic | manual) [automatic]`	Controls the way the CDB configuration store does its journal compaction. Never set to anything but the default 'automatic' unless there is an external mechanism that controls the compaction using the cdb_initiate_journal_compaction() API call.
`/ncs-config/cdb/operational`	Operational data can either be implemented by external callbacks, or stored in CDB (or a combination of both). The operational data store is used when data is to be stored in CDB.
`/ncs-config/cdb/operational/ db-dir (string)`	*db-dir* is the directory on disk that CDB operational uses for its storage and any temporary files. If left unset (default), the same directory as *db-dir* for CDB is used.
`/ncs-config/encrypted-strings`	*encrypted-strings* defines keys used to encrypt strings that adhere to the types tailf:des3-cbc-encryptedstring and tailf:aes-cfb-128-encrypted-string.
`/ncs-config/encrypted-strings/DES3CBC`	With DES3CBC, three 64-bit (8-byte) keys and a random initial vector are used to encrypt the string. The initVector leaf is only used when upgrading from earlier versions, but is retained for backward compatibility.

Parameter	Description
`/ncs-config/encrypted-strings/DES3CBC/key1 (hex8-value-type)`	This parameter is mandatory.
`/ncs-config/encrypted-strings/DES3CBC/key2 (hex8-value-type)`	This parameter is mandatory.
`/ncs-config/encrypted-strings/DES3CBC/key3 (hex8-value-type)`	This parameter is mandatory.
`/ncs-config/encrypted-strings/DES3CBC/initVector (hex8-value-type)`	—
`/ncs-config/encrypted-strings/AESCFB128`	With AESCFB128, one 128-bit (16-byte) key and a random initial vector are used to encrypt the string. The initVector leaf is only used when upgrading from earlier versions, but is retained for backward compatibility.
`/ncs-config/encrypted-strings/AESCFB128/key (hex16-value-type)`	This parameter is mandatory.
`/ncs-config/encrypted-strings/AESCFB128/initVector (hex16-value-type)`	—
`/ncs-config/crypt-hash`	*crypt-hash* specifies how clear-text values should be hashed for leafs of the types ianach:crypt-hash, tailf:sha-256-digest-string, and tailf:sha-512-digest-string.
`/ncs-config/crypt-hash/algorithm (md5 | sha-256 | sha-512) [md5]`	*algorithm* can be set to one of the values 'md5', 'sha-256', or 'sha-512', to choose the corresponding hash algorithm for hashing of clear-text input for the ianach:crypt-hash type.
`/ncs-config/crypt-hash/rounds (crypt-hash-rounds-type) [5000]`	For the 'sha-256' and 'sha-512' algorithms for the ianach:crypt-hash type, and for the tailf:sha-256-digest-string and tailf:sha-512-digest-string types, *rounds* specifies how many times the hashing loop should be executed. If a value other than the default 5000 is specified, the hashed format has 'rounds=N$', where N is the specified value, prepended to the salt. This parameter is ignored for the 'md5' algorithm for ianach:crypt-hash.
`/ncs-config/logs`	—
`/ncs-config/logs/syslog-config`	Shared settings for how to log to syslog. Logs can be configured to log to file or syslog. If a log is configured to log to syslog, the settings under /ncs-config/logs/syslog-config are used.
`/ncs-config/logs/syslog-config/version (bsd | 1) [bsd]`	*version* is either 'bsd' (traditional syslog) or '1' (new IETF syslog format: RFC 5424). '1' implies that /ncs-config/logs/syslog-config/udp/enabled must be set to true.

Parameter	Description
/ncs-config/logs/syslog-config/facility (daemon | authpriv | local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 | uint32) [daemon]	This facility setting is the default facility. It is also possible to set individual facilities in the different logs.
/ncs-config/logs/syslog-config/udp	—
/ncs-config/logs/syslog-config/udp/enabled (boolean) [false]	If 'false', messages are sent to the local syslog daemon.
/ncs-config/logs/syslog-config/udp/host (string | ipv4-address | ipv6-address)	This parameter is mandatory. *host* is either a domain name or an IPv4/IPv6 network address. UDP syslog messages are sent to this host.
/ncs-config/logs/syslog-config/udp/port (port-number) [514]	*port* is a valid port number to be used in combination with /ncs-config/logs/syslog-config/udp/host.
/ncs-config/logs/syslog-config/syslog-servers	This is an alternative way of specifying UDP syslog servers. If you configure the /ncs-config/logs/syslog-config/udp container, any configuration in this container is ignored.
/ncs-config/logs/syslog-config/syslog-servers/server	A set of syslog servers that get a copy of all syslog messages.
/ncs-config/logs/syslog-config/syslog-servers/server/host (string | ipv4-address | ipv6-address)	*host* is either a domain name or an IPv4/IPv6 network address. UDP syslog messages are sent to this host.
/ncs-config/logs/syslog-config/syslog-servers/server/port (port-number) [514]	*port* is the UDP port number where this syslog server is listening.
/ncs-config/logs/syslog-config/syslog-servers/server/version (bsd | 1) [bsd]	*version* is either 'bsd' (traditional syslog) or '1' (new IETF syslog format: RFC 5424).
/ncs-config/logs/syslog-config/syslog-servers/server/facility (daemon | authpriv | local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 | uint32) [daemon]	—
/ncs-config/logs/syslog-config/syslog-servers/server/enabled (boolean) [true]	If 'false', this syslog server does not get any UDP messages.
/ncs-config/logs/ncs-log	ncs-log is WAE's daemon log. Check this log for startup problems of the WAE daemon itself. This log is not rotated; use logrotate(8).

Parameter	Description
`/ncs-config/logs/ncs-log/ enabled (boolean) [true]`	If 'true', the log is enabled.
`/ncs-config/logs/ncs-log/file`	—
`/ncs-config/logs/ncs-log/ file/name (string)`	*name* is the full path to the actual log file.
`/ncs-config/logs/ncs-log/file/ enabled (boolean) [false]`	If 'true', file logging is enabled.
`/ncs-config/logs/ncs-log/syslog`	—
`/ncs-config/logs/ncs-log/ syslog/enabled (boolean) [false]`	If 'true', syslog messages are sent.
`/ncs-config/logs/ncs-log/ syslog/facility (daemon | authpriv | local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 | uint32)`	This optional value overrides the /ncs-config/logs/syslog-config/facility for the specified log.
`/ncs-config/logs/developer-log`	*developer-log* is a debug log for troubleshooting user-written Java code. Enable and check this log for problems with validation code. This log is enabled by default. In all other regards it can be configured as ncs-log. This log is not rotated; use logrotate(8).
`/ncs-config/logs/developer-log/ enabled (boolean) [true]`	If 'true', the log is enabled.
`/ncs-config/logs/developer-log/ file`	—
`/ncs-config/logs/developer-log/ file/name (string)`	*name* is the full path to the actual log file.
`/ncs-config/logs/developer-log/ file/enabled (boolean) [false]`	If 'true', file logging is enabled.
`/ncs-config/logs/developer-log/ syslog`	—
`/ncs-config/logs/developer-log/ syslog/enabled (boolean) [false]`	If 'true', syslog messages are sent.
`/ncs-config/logs/developer-log/ syslog/facility (daemon | authpriv | local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 | uint32)`	This optional value overrides the /ncs-config/logs/syslog-config/facility for the specified log.

Parameter	Description
`/ncs-config/logs/developer-log-level (error | info | trace) [info]`	Controls the level of developer messages to print in the developer log.
`/ncs-config/logs/audit-log`	*audit-log* is an audit log that records successful and failed logins to the WAE back plane. This log is enabled by default. In all other regards it can be configured as /ncs-config/logs/ncs-log. This log is not rotated; use logrotate(8).
`/ncs-config/logs/audit-log/ enabled (boolean) [true]`	If 'true', the log is enabled.
`/ncs-config/logs/audit-log/file`	—
`/ncs-config/logs/audit-log/ file/name (string)`	*name* is the full path to the actual log file.
`/ncs-config/logs/audit-log/ file/enabled (boolean) [false]`	If 'true', file logging is enabled.
`/ncs-config/logs/audit-log/ syslog`	—
`/ncs-config/logs/audit-log/ syslog/enabled (boolean) [false]`	If 'true', syslog messages are sent.
`/ncs-config/logs/audit-log/ syslog/facility (daemon | authpriv | local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 | uint32)`	This optional value overrides the /ncs-config/logs/syslog-config/facility for the specified log.
`/ncs-config/logs/audit-log-commit (boolean) [false]`	Controls whether the audit log should include messages about the resulting configuration changes for each commit to the running data store.
`/ncs-config/logs/netconf-log`	*netconf-log* is a log for troubleshooting northbound NETCONF operations, such as checking why a filter operation didn't return the data requested. This log is enabled by default. In all other regards it can be configured as /ncs-config/logs/ncs-log. This log is not rotated; use logrotate(8).
`/ncs-config/logs/netconf-log/ enabled (boolean) [true]`	If 'true', the log is enabled.
`/ncs-config/logs/netconf-log/ file`	—
`/ncs-config/logs/netconf-log/ file/name (string)`	*name* is the full path to the actual log file.
`/ncs-config/logs/netconf-log/ file/enabled (boolean) [false]`	If 'true', file logging is enabled.
`/ncs-config/logs/netconf-log/syslog`	—
`/ncs-config/logs/netconf-log/ syslog/enabled (boolean) [false]`	If 'true', syslog messages are sent.

Parameter	Description
`/ncs-config/logs/netconf-log/ syslog/facility (daemon | authpriv | local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 | uint32)`	This optional value overrides the /ncs-config/logs/syslog-config/facility for the specified log.
`/ncs-config/logs/snmp-log`	—
`/ncs-config/logs/snmp-log/ enabled (boolean) [true]`	If 'true', the log is enabled.
`/ncs-config/logs/snmp-log/file`	—
`/ncs-config/logs/snmp-log/ file/name (string)`	*name* is the full path to the actual log file.
`/ncs-config/logs/snmp-log/ file/enabled (boolean) [false]`	If 'true', file logging is enabled.
`/ncs-config/logs/snmp-log/ syslog`	—
`/ncs-config/logs/snmp-log/ syslog/enabled (boolean) [false]`	If 'true', syslog messages are sent.
`/ncs-config/logs/snmp-log/ syslog/facility (daemon | authpriv | local0 | local1 | local2 | local3 | local4 | local5 | local6 | local7 | uint32)`	This optional value overrides the /ncs-config/logs/syslog-config/facility for the specified log.
`/ncs-config/logs/snmp-log-level (error | info) [info]`	Controls which level of SNMP PDUs are printed in the SNMP log. The value 'error' means that only PDUs with error-status not equal to 'noError' are printed.
`/ncs-config/logs/webui-browser-log`	*webui-browser-log* makes it possible to log Java script errors/exceptions in a log file on the target device instead of just in the browser's error console. This log is not enabled by default and is not rotated; use logrotate(8).
`/ncs-config/logs/webui-browser-log/ enabled (boolean) [false]`	If 'true', the browser log is used.
`/ncs-config/logs/webui-browser-log/filename (string)`	This parameter is mandatory. The path to the filename where browser log entries are written.
`/ncs-config/logs/webui-access-log`	*webui-access-log* is an access log for the embedded WAE web server. This file adheres to the Common Log Format, as defined by Apache and others. This log is not enabled by default and is not rotated; use logrotate(8).
`/ncs-config/logs/webui-access-log/ enabled (boolean) [false]`	If 'true', the access log is used.

Parameter	Description
`/ncs-config/logs/webui-access-log/ traffic-log (boolean) [false]`	If 'true', all HTTP(S) traffic towards the embedded web server is logged in a log file named traffic.trace. This log is not enabled by default and is not rotated; use logrotate(8). **Caution** Do not use this log in a production setting.
`/ncs-config/logs/webui-access-log/ dir (string)`	This parameter is mandatory. The path to the directory where the access log is written.
`/ncs-config/logs/netconf-trace-log`	*netconf-trace-log* is a log for understanding and troubleshooting northbound NETCONF protocol interactions. When this log is enabled, all NETCONF traffic to and from WAE is stored to a file. By default, all XML is pretty-printed. This slows down the NETCONF server, so be careful when enabling this log. This log is not rotated; use logrotate(8).
`/ncs-config/logs/netconf-trace-log/ enabled (boolean) [false]`	If 'true', all NETCONF traffic is logged.
`/ncs-config/logs/netconf-trace-log/ filename (string)`	This parameter is mandatory. The name of the file where the NETCONF traffic trace log is written.
`/ncs-config/logs/netconf-trace-log/ format (pretty | raw) [pretty]`	The value 'pretty' means that the XML data is pretty-printed. The value 'raw' means that it is not pretty-printed.
`/ncs-config/logs/xpath-trace-log`	*xpath-trace-log* is a log for understanding and troubleshooting xpath evaluations. When this log is enabled, all xpath queries evaluated by WAE are logged to a file. This slows down WAE, so be careful when enabling this log. This log is not rotated; use logrotate(8).
`/ncs-config/logs/xpath-trace-log/ enabled (boolean) [false]`	If 'true', all xpath execution is logged.
`/ncs-config/logs/xpath-trace-log/ filename (string)`	This parameter is mandatory. The name of the file where the xpath trace log is written.
`/ncs-config/logs/error-log`	*error-log* is an error log used for internal logging from the WAE daemon. It is used for troubleshooting the WAE daemon itself, and should normally be disabled. This log is rotated by the WAE daemon.
`/ncs-config/logs/error-log/ enabled (boolean) [false]`	If 'true', error logging is performed.
`/ncs-config/logs/error-log/ filename (string)`	This parameter is mandatory. *filename* is the full path to the actual log file. This parameter must be set if the error log is enabled.
`/ncs-config/logs/error-log/max-size (tailf:size) [S1M]`	*max-size* is the maximum size of an individual log file before it is rotated. Log filenames are reused when five logs have been exhausted.
`/ncs-config/logs/error-log/ debug`	—
`/ncs-config/logs/error-log/ debug/enabled (boolean) [false]`	—

Parameter	Description
`/ncs-config/logs/error-log/debug/level (uint16) [2]`	—
`/ncs-config/logs/error-log/debug/tag (string)`	This parameter can be given multiple times.
`/ncs-config/candidate`	—
`/ncs-config/candidate/ filename (string)`	The candidate db-mode has been removed; this leaf no longer affects the WAE configuration. This leaf and the candidate container are retained for backward compatibility.
`/ncs-config/sort-transactions (boolean) [true]`	This parameter controls how WAE lists newly created, not yet committed list entries. If this value is set to 'false', WAE lists all new elements before listing existing data. If this value is set to 'true', WAE merges new and existing entries, and provides one sorted view of the data. This behavior works well when CDB is used to store configuration data, but if an external data provider is used, WAE does not know the sort order and cannot merge the new entries correctly. If an external data provider is used for configuration data, and if the sort order differs from CDB's sort order, this parameter should be set to 'false'.
`/ncs-config/enable-attributes (boolean) [true]`	This parameter controls whether WAE's attribute feature is enabled. There are two attributes: annotations and tags. These are available in northbound interfaces (the annotate command in the CLI, and the annotation XML attribute in NETCONF), but to be useful they need support from the underlying configuration data provider. CDB supports attributes, but if an external data provider is used for configuration data, and if it does not support the attribute callbacks, this parameter should be set to 'false'.
`/ncs-config/enable-inactive (boolean) [true]`	This parameter controls whether WAE's inactive feature is enabled. This feature also requires enableAttributes to be enabled. When WAE is used to control Juniper routers, this feature is required.
`/ncs-config/session-limits`	Limits concurrent access to WAE.
`/ncs-config/session-limits/max-sessions (uint32 | unbounded) [unbounded]`	Limits the total number of concurrent sessions to WAE.
`/ncs-config/session-limits/session-limit`	Limits concurrent access for a specific context to WAE. There can be multiple instances of this container element, each one specifying parameters for a specific context.
`/ncs-config/session-limits/session-limit/context (string)`	The context is cli, netconf, webui, snmp, or any other context string defined through the use of MAAPI. For example, if you use MAAPI to implement a CORBA interface to WAE, the MAAPI program could send the string 'corba' as context.
`/ncs-config/session-limits/session-limit/max-sessions (uint32 | unbounded)`	This parameter is mandatory. Limits the total number of concurrent sessions to WAE.

Parameter	Description
/ncs-config/session-limits/ max-config-sessions (uint32 | unbounded) [unbounded]	Limits the total number of concurrent configuration sessions to WAE.
/ncs-config/session-limits/ config-session-limit	Limits concurrent read-write transactions for a specific context to WAE. There can be multiple instances of this container element, each one specifying parameters for a specific context.
/ncs-config/session-limits/ config-session-limit/context (string)	The context is cli, netconf, webui, snmp, or any other context string defined through the use of MAAPI. For example, if you use MAAPI to implement a CORBA interface to WAE, the MAAPI program could send the string 'corba' as context.
/ncs-config/session-limits/ config-session-limit/max-sessions (uint32 |unbounded)	This parameter is mandatory. Limits the total number of concurrent configuration sessions to WAE for the corresponding context.
/ncs-config/aaa	—
/ncs-config/aaa/ssh-login-grace-time (xs:duration) [PT10M]	WAE servers close SSH connections after this time if the client has not successfully authenticated itself. If the value is 0, there is no time limit for client authentication. This is a global value for all SSH servers in WAE. Changing this value affects only SSH connections that are established after the change is made.
/ncs-config/aaa/ssh-max-auth-tries (uint32 | unbounded) [unbounded]	WAE servers close SSH connections when the client has made this number of unsuccessful authentication attempts. This is a global value for all SSH servers in WAE. Changing this value affects only SSH connections that are established after the change is made.
/ncs-config/aaa/ssh-server-key-dir (string)	*ssh-server-key-dir* is the directory file path where the keys used by the WAE SSH daemon are found. This parameter must be set if SSH is enabled for NETCONF or the CLI. If SSH is enabled, the server keys used by WAE are on the same format as the server keys used by openssh (that is, the same format as generated by 'ssh-keygen'). Only DSA- and RSA-type keys can be used with the WAE SSH daemon, as generated by 'ssh-keygen' with the '-t dsa' and '-t rsa' switches, respectively. The key must be stored with an empty passphrase, and with the name 'ssh_host_dsa_key' if it is a DSA-type key, and with the name 'ssh_host_rsa_key' if it is an RSA-type key. The SSH server advertises support for those key types for which there is a key file available and for which the required algorithm is enabled. See the /ncs-config/ssh/algorithms/server-host-key leaf.

Parameter	Description
`/ncs-config/aaa/ssh-pubkey-authentication (none | local | system) [system]`	Controls how the WAE SSH daemon locates the user keys for public key authentication. If set to 'none', public key authentication is disabled. If set to 'local', and the user exists in /aaa/authentication/users, the keys in the user's 'ssh_keydir' directory are used. If set to 'system', the user is first looked up in /aaa/authentication/users, but only if /ncs-config/aaa/local-authentication/enabled is set to 'true'. If local-authentication is disabled, or if the user does not exist in /aaa/authentication/users but does exist in the OS password database, the keys in the user's $HOME/.ssh directory are used.
`/ncs-config/aaa/default-group (string)`	If the user group cannot be found in the AAA subsystem, a logged-in user ends up as a member of the default group (if specified). If a user logs in and the group membership cannot be established, the user has zero access rights.
`/ncs-config/aaa/auth-order (string)`	The default order for authentication is 'local-authentication pam external-authentication'. It is possible to change this order through this parameter.
`/ncs-config/aaa/expiration-warning (ignore | display | prompt) [ignore]`	When PAM or external authentication is used, the authentication mechanism might give a warning that the user's password is about to expire. This parameter controls how the WAE daemon processes that warning message. If set to 'ignore', the warning is ignored. If set to 'display', interactive user interfaces display the warning message at login. If set to 'prompt', interactive user interfaces display the warning message at login. The user must acknowledge the message before proceeding.
`/ncs-config/aaa/audit-user-name (always | known | never) [known]`	Controls the logging of the username when a failed authentication attempt is logged to the audit log. If set to "always", the username is always logged. If set to "known", the username is only logged when it is known to be valid (that is, when attempting local-authentication and the user exists in /aaa/authentication/users). Otherwise, it is logged as "[withheld]". If set to "never", the username is always logged as "[withheld]".
`/ncs-config/aaa/pam`	If PAM is used for login, the WAE daemon typically must run as root.
`/ncs-config/aaa/pam/enabled (boolean) [false]`	When set to 'true', WAE uses PAM for authentication.
`/ncs-config/aaa/pam/service (string) [common-auth]`	The PAM service to use for the login NETCONF/SSH CLI procedure. This can be any service installed in the /etc/pam.d directory. Different unices have different services installed under /etc/pam.d. Choose an existing service or create a new one.

Parameter	Description
`/ncs-config/aaa/pam/timeout (xs:duration) [PT10S]`	The maximum time that authentication waits for a reply from PAM. If the timeout is reached, the PAM authentication fails, but authentication attempts are made with other mechanisms as configured for /ncs-config/aaa/authOrder. The default is PT10S (10 seconds).
`/ncs-config/aaa/external-authentication`	—
`/ncs-config/aaa/external-authentication/enabled (boolean) [false]`	When set to 'true', external authentication is used.
`/ncs-config/aaa/external-authentication/executable (string)`	If external authentication is enabled, an executable on the local host can be launched to authenticate a user. The executable receives the username and the clear-text password on its standard input. The format is '[${USER};${PASS};]\n'. For example, if user is 'bob' and password is 'secret', the executable receives the line '[bob;secret;]' followed by a new line on its standard input. The program must parse this line. The task of the external program is to authenticate the user and also provide the user-to-groups mapping. If 'bob' is a member of the 'oper' and the 'lamers' groups, the program should echo 'accept oper lamers' on its standard output. If the user fails to authenticate, the program should echo 'reject ${reason}' on its standard output.
`/ncs-config/aaa/external-authentication/use-base64 (boolean) [false]`	When set to 'true', ${USER} and ${PASS} in the data passed to the executable are base64-encoded, allowing the password to contain ';' characters. For example, if user is 'bob' and password is 'secret', the executable receives the string '[Ym9i;c2VjcmV0;]' followed by a new line.
`/ncs-config/aaa/external-authentication/include-extra (boolean) [false]`	When set to 'true', additional information items are provided to the executable: source IP address and port, context, and protocol. The complete format is '[${USER};${PASS};${IP};$ {PORT};${CONTEXT};${PROTO};]\n'. Example: '[bob;secret;192.168.1.1;12345;cli;ssh;]\n'.
`/ncs-config/aaa/local-authentication`	—
`/ncs-config/aaa/local-authentication/enabled (boolean) [true]`	When set to 'true', WAE uses local authentication. The user data kept in the aaa namespace is used to authenticate users. When set to 'false', another authentication mechanism (such as PAM or external authentication) is used.
`/ncs-config/aaa/authentication-callback`	—
`/ncs-config/aaa/authentication-callback/enabled (boolean) [false]`	When set to 'true', WAE invokes an application callback when authentication succeeds or fails. The callback might reject an otherwise successful authentication. If the callback has not been registered, all authentication attempts fail.
`/ncs-config/aaa/authorization`	—

Parameter	Description
/ncs-config/aaa/authorization/enabled (boolean) [true]	When set to 'false', all authorization checks are turned off, similar to the -noaaa flag in ncs_cli.
/ncs-config/aaa/authorization/ callback	—
/ncs-config/aaa/authorization/callback/enabled (boolean) [false]	When set to 'true', WAE invokes application callbacks for authorization. If the callbacks have not been registered, all authorization checks are rejected.
/ncs-config/aaa/namespace (string) [http://tail-f.com/ns/aaa/1.1]	To move the AAA data into another user-defined namespace, indicate that namespace here.
/ncs-config/aaa/prefix (string) [/]	To move the AAA data into another user-defined namespace, indicate the prefix path in that namespace where the WAE AAA namespace is mounted.
/ncs-config/rollback	Settings that control if and where rollback files are created. A rollback file contains a copy of the system configuration. The current running configuration is always stored in rollback0, the previous version in rollback1, and so on. The oldest saved configuration has the highest suffix.
/ncs-config/rollback/ enabled (boolean) [false]	When set to 'true', a rollback file is created whenever the running configuration is modified.
/ncs-config/rollback/ directory (string)	This parameter is mandatory. The location where rollback files are created.
/ncs-config/rollback/ history-size (uint32) [35]	The number of old configurations to save.
/ncs-config/rollback/ type (delta) [delta]	This parameter is deprecated. WAE supports only type 'delta'. It is not necessary to set a value for this parameter; it is retained only for backward compatibility. Type 'delta' means that only the changes are stored in the rollback file. Rollback file 0 contains the changes from the last configuration commit. This is space and time efficient for large configurations.
/ncs-config/rollback/ rollback-numbering (rolling | fixed) [fixed]	*rollback-numbering* is either 'fixed' or 'rolling'. If set to 'rolling', rollback file '0' always contains the last commit. If set to 'fixed', each rollback gets a unique increasing number.
/ncs-config/ssh	Controls the behavior of the SSH server built into WAE.
/ncs-config/ssh/idle-connection-timeout (xs:duration) [PT10M]	The maximum time that an authenticated connection to the SSH server is allowed to exist without open channels. If the timeout is reached, the SSH server closes the connection. The default is PT10M (10 minutes). A value of 0 means there is no timeout.
/ncs-config/ssh/algorithms	Defines custom lists of algorithms to be usable with the built-in SSH implementation. For each type of algorithm, an empty value means that all supported algorithms should be usable. A non-empty value (a comma-separated list of algorithm names) means that the intersection of the supported algorithms and the configured algorithms should be usable.

Parameter	Description
`/ncs-config/ssh/algorithms/server-host-key (string) []`	The supported serverHostKey algorithms (if implemented in libcrypto) are "ssh-dss" and "ssh-rsa", but for any SSH server, it is limited to those algorithms for which there is a host key installed in the directory given by /ncs-config/aaa/ssh-server-key-dir. To limit the usable serverHostKey algorithms to "ssh-dss", set this value to "ssh-dss" or avoid installing a key of any other type than ssh-dss in the sshServerKeyDir.
`/ncs-config/ssh/algorithms/kex (string) []`	The supported key exchange algorithms (as long as their hash functions are implemented in libcrypto) are "diffie-hellman-group-exchange-sha256", "diffie-hellman-group-exchange-sha1", "diffie-hellmangroup14-sha1", and "diffie-hellman-group1-sha1". To limit the usable key exchange algorithms to "diffie-hellman-group14-sha1" and "diffie-hellmangroup-exchange-sha256" (in that order), set this value to "diffie-hellman-group14-sha1, diffie-hellmangroup-exchange-sha256".
`/ncs-config/ssh/algorithms/dh-group`	The range of allowed group size the SSH server responds to the client during a "diffie-hellman-groupexchange". The range is the intersection of what the client requests. If there is none, the key exchange is aborted.
`/ncs-config/ssh/algorithms/ dh-group/min-size (dh-group-size-type) [2048]`	Minimum size of p, in bits.
`/ncs-config/ssh/algorithms/dh-group/max-size (dh-group-size-type) [4096]`	Maximum size of p, in bits.
`/ncs-config/ssh/algorithms/mac (string) []`	The supported mac algorithms (if implemented in libcrypto) are "hmac-md5", "hmac-sha1", "hmacsha2-256", "hmac-sha2-512", "hmac-sha1-96", and "hmac-md5-96".
`/ncs-config/ssh/algorithms/encryption (string) []`	The supported encryption algorithms (if implemented in libcrypto) are "aes128-ctr", "aes192-ctr", "aes256-ctr", "aes128-cbc", "aes256-cbc", and "3des-cbc".
`/ncs-config/ssh/client-alive-interval (xs:duration | infinity) [infinity]`	If no data has been received from a connected client for this long, a request that requires a response from the client is sent over the SSH transport.
`/ncs-config/ssh/client-alive-count-max (uint32) [3]`	If no data has been received from the client after this many consecutive client-alive-intervals have passed, the connection drops.
`/ncs-config/cli`	CLI parameters.
`/ncs-config/cli/enabled (boolean) [true]`	If 'true', the CLI server is started.
`/ncs-config/cli/allow-implicit-wildcard (boolean) [true]`	If 'true', users do not need to explicitly type * in the place of keys in lists, in order to see all list instances. If 'false', users must explicitly type * to see all list instances.
`/ncs-config/cli/completion-show-max (cli-max) [100]`	The maximum number of possible alternatives to present when doing completion.

Parameter	Description
`/ncs-config/cli/style (j | c)`	Style is either 'j' or 'c'. If set to 'j', the CLI is presented as a Juniper-style CLI. If 'c', the CLI appears as Cisco XR style.
`/ncs-config/cli/ssh`	—
`/ncs-config/cli/ssh/enabled (boolean) [true]`	*enabled* is either 'true' or 'false'. If 'true', the WAE CLI uses the built-in SSH server.
`/ncs-config/cli/ssh/ip (ipv4-address | ipv6-address) [0.0.0.0]`	*ip* is an IP address that the WAE CLI listens on for SSH connections. 0.0.0.0 means that it listens on the port (/ncs-config/cli/ssh/port) for all IPv4 addresses on the machine.
`/ncs-config/cli/ssh/port (port-number) [2024]`	The port number for CLI SSH.
`/ncs-config/cli/ssh/banner (string) []`	*banner* is a string that is presented to the client before authenticating when logging in to the CLI via the built-in SSH server.
`/ncs-config/cli/ssh/banner-file (string) []`	*banner-file* is the name of a file whose contents are presented (after any string given by the banner directive) to the client before authenticating when logging in to the CLI via the built-in SSH server.
`/ncs-config/cli/ssh/extra-listen`	A list of additional IP address and port pairs that the WAE CLI listens on for SSH connections.
`/ncs-config/cli/ssh/extra-listen/ip (ipv4-address | ipv6-address)`	—
`/ncs-config/cli/ssh/extra-listen/port (port-number)`	—
`/ncs-config/cli/top-level-cmds-in-sub-mode (boolean) [false]`	topLevelCmdsInSubMode is 'true' or 'false'. If 'true', all top-level commands in I and C style CLI are available in submodes.
`/ncs-config/cli/completion-meta-info (false | alt1 | alt2) [false]`	completionMetaInfo is 'false', 'alt1', or 'alt2'. If set to 'alt1', the alternatives shown for possible completions are prefixed as follows: containers with > lists with + leaf-lists + For example: Possible completions: ... > applications + apply-groups ... + dns-servers ... If set to 'alt2', possible completions are prefixed as follows: containers with > lists with children with +> lists without children + For example: Possible completions: ... > applications +>apply-groups ... + dns-servers ...
`/ncs-config/cli/allow-abbrev-keys (boolean) [false]`	allowAbbrevKeys is 'true' or 'false'. If 'false', key elements are not allowed to be abbreviated in the CLI. This is relevant in the J-style CLI when using the commands 'delete' and 'edit'. This is relevant in the C/I-style CLIs when using the commands 'no', 'show configuration', and for commands to enter submodes.

Parameter	Description
`/ncs-config/cli/j-align-leaf-values (boolean) [true]`	j-align-leaf-values is 'true' or 'false'. If 'true', the leaf values of all siblings in a container or list are aligned.
`/ncs-config/cli/enter-submode-on-leaf (boolean) [true]`	enterSubmodeOnLeaf is 'true' or 'false'. If 'true' (the default), setting a leaf in a submode from a parent mode results in entering the submode after the command has completed. If 'false', an explicit command for entering the submode is required—for example, if running the command **interface FastEthernet 1/1/1 mtu 1400** from the top level in config mode. If enterSubmodeOnLeaf is 'true', the CLI ends up in the 'interface FastEthernet 1/1/1' submode after the command execution. If 'false', the CLI remains at the top level. To enter the submode when set to 'false', the command **interface FastEthernet 1/1/1** is required. Applied to the C-style CLI.
`/ncs-config/cli/table-look-ahead (int64) [50]`	The tableLookAhead element tells confd how many rows to pre-fetch when displaying a table. The prefetched rows are used to calculate the required column widths for the table. If set to a small number, you should explicitly configure the column widths in the clispec file.
`/ncs-config/cli/more-buffer-lines (uint32 | unbounded) [unbounded]`	moreBufferLines is used to limit the buffering done by the more process. It can be 'unbounded' or a positive integer that describes the maximum number of lines to buffer.
`/ncs-config/cli/show-all-ns (boolean) [false]`	If showAllNs is 'true', all elem names are prefixed with the namespace prefix in the CLI. This is visible when setting values and when showing the configuration.
`/ncs-config/cli/suppress-fast-show (boolean) [false]`	suppressFastShow is 'true' or 'false'. If 'true', the fast show optimization is suppressed in the C-style CLI. The fast show optimization is somewhat experimental and might break certain operations.
`/ncs-config/cli/use-expose-ns-prefix (boolean) [true]`	If 'true', all nodes annotated with the tailf:cli-expose-ns-prefix result in the namespace prefix being shown/required. If 'false', the tailf:cli-expose-ns-prefix annotation is ignored. The container /devices/device/config has this annotation.
`/ncs-config/cli/show-defaults (boolean) [false]`	*show-defaults* is 'true' or 'false'. If 'true', default values are shown when displaying the configuration. The default value is shown inside a comment on the same line as the value. Showing default values can also be enabled in the CLI per session using the operational mode command **set show defaults true**.
`/ncs-config/cli/default-prefix (string) []`	*default-prefix* is a string that is placed in front of the default value when a configuration is shown with default values as comments.
`/ncs-config/cli/commit-retry-timeout (xs:duration | infinity) [PT0S]`	The commit timeout in the CLI. This timeout controls for how long the commit operation tries to complete the operation when some other entity is locking the database. A similar configuration parameter, /ncs-config/commit-retry-timeout, sets a timeout for WAE transactions in the JSON-RPC API.
`/ncs-config/cli/timezone (utc | local) [local]`	Time in the CLI can be local (as configured on the host) or UTC.

Parameter	Description
`/ncs-config/cli/with-defaults (boolean) [false]`	*with-defaults* is 'true' or 'false'. If 'false', leaf nodes that have their default values are not shown when the user displays the configuration, unless the user gives the 'details' option to the 'show' command. This is useful when there are many settings that are seldom used. If 'false', only the values actually modified by the user are shown.
`/ncs-config/cli/banner (string) []`	Banner shown to the user when the CLI is started. The default is empty.
`/ncs-config/cli/banner-file (string) []`	File whose contents are shown to the user (after any string set by the 'banner' directive) when the CLI is started. The default is empty.
`/ncs-config/cli/prompt1 (string) [\u@\h\M> ]`	Prompt used in operational mode. The string might contain a number of backslash-escaped special characters that are decoded as follows: • \d—Date in 'YYYY-MM-DD' format (for example, '2006-01-18'). • \h—Hostname up to the first '.' (or delimiter as defined by promptHostnameDelimiter). • \H—Current time in 24-hour HH:MM:SS format. • \T—Current time in 12-hour HH:MM:SS format. • \@—Current time in 12-hour am/pm format. • \A—Current time in 24-hour HH:MM format. • \u—Username of the current user. • \m—Mode name (only used in XR style). • \M—Mode name inside parenthesis if in a mode.
`/ncs-config/cli/prompt2 (string) [\u@\h\M% ]`	Prompt used in configuration mode. The string might contain a number of backslash-escaped special characters that are decoded as described for prompt1.
`/ncs-config/cli/c-prompt1 (string) [\u@\h\M> ]`	Prompt used in operational mode in the Cisco XR-style CLI. The string might contain a number of backslash-escaped special characters that are decoded as described for prompt1.
`/ncs-config/cli/c-prompt2 (string) [\u@\h\M% ]`	Prompt used in configuration mode in the Cisco XR-style CLI. The string might contain a number of backslash-escaped special characters that are decoded as described for prompt1.
`/ncs-config/cli/prompt-hostname-delimiter (string) [.]`	When the \h token is used in a prompt, the first part of the hostname up until the first occurrence of the promptHostnameDelimiter is used.
`/ncs-config/cli/show-log-directory (string) [/var/log]`	Location where the **show log** command looks for log files.
`/ncs-config/cli/idle-timeout (xs:duration) [PT30M]`	Maximum idle time before terminating a CLI session. The default is PT30M (30 minutes).

Parameter	Description
`/ncs-config/cli/prompt-sessions-cli (boolean) [false]`	promptSessionsCLI is 'true' or 'false'. If 'true', only the current CLI sessions are displayed when the user tries to start a new CLI session and the maximum number of sessions has been reached. Note that MAAPI sessions with their context set to 'cli' are regarded as CLI sessions and are listed as such.
`/ncs-config/cli/suppress-ned-errors (boolean) [false]`	Suppress errors from NED devices. Make log-communication between WAE and its devices more silent. Be careful with this option, because it might suppress interesting errors as well.
`/ncs-config/cli/disable-idle-timeout-on-cmd (boolean) [true]`	*disable-idle-timeout-on-cmd* is 'true' or 'false'. If 'false', the idle timeout triggers even when a command is running in the CLI. If 'true', the idle timeout only triggers if the user is idling at the CLI prompt.
`/ncs-config/cli/command-timeout (xs:duration | infinity) [infinity]`	Global command timeout: terminate the command unless the command has completed within the timeout. We do not recommend using this feature because it might have undesirable effects in a loaded system where normal commands take longer to complete. This timeout can be overridden by a command-specific timeout specified in the ncs.cli file.
`/ncs-config/cli/space-completion`	—
`/ncs-config/cli/space-completion/enabled (boolean)`	—
`/ncs-config/cli/ignore-leading-whitespace (boolean)`	If 'false', the CLI shows completion help when you enter TAB or SPACE as the first characters on a row. If 'true', leading SPACE and TAB are ignored. Enter '?' for a list of possible alternatives. Setting the value to 'true' makes it easier to paste scripts into the CLI.
`/ncs-config/cli/auto-wizard`	The default value for autowizard in the CLI. Users can always enable or disable the autowizard in each session; this controls the initial session value.
`/ncs-config/cli/auto-wizard/enabled (boolean) [true]`	**`enabled`** is 'true' or 'false'. If 'true', the CLI prompts the user for required attributes when a new identifier is created.
`/ncs-config/cli/restricted-file-access (boolean) [false]`	*restricted-file-access* is 'true' or 'false'. If 'true', a CLI user cannot access files and directories outside the home directory tree.
`/ncs-config/cli/restricted-file-regexp (string) []`	*restricted-file-regexp* is either an empty string or a regular expression (AWK style). If not empty, all files and directories created or accessed must match the regular expression. This can be used to ensure that certain symbols do not occur in created files.
`/ncs-config/cli/history-save (boolean) [true]`	If 'true', the CLI history is saved between CLI sessions. The history is stored in the state directory.
`/ncs-config/cli/history-remove-duplicates (boolean) [false]`	If 'true', repeated commands in the CLI are only stored once in the history. Each invocation of the command only updates the date of the last entry. If 'false', duplicates are stored in the history.
`/ncs-config/cli/history-max-size (int64) [1000]`	Sets the maximum configurable history size.

Parameter	Description
`/ncs-config/cli/message-max-size (int64) [10000]`	Sets the maximum size of user messages.
`/ncs-config/cli/show-commit-progress (boolean) [true]`	*show-commit-progress* is 'true' or 'false'. If 'true', the commit operation in the CLI provides progress information.
`/ncs-config/cli/commit-message (boolean) [true]`	CLI prints a message when a commit is executed.
`/ncs-config/cli/use-double-dot-ranges (boolean) [true]`	*use-double-dot-ranges* is 'true' or 'false'. If 'true', range expressions are given as 1..3. If 'false', ranges are given as 1-3.
`/ncs-config/cli/allow-range-expression-all-types (boolean) [true]`	*allow-range-expression-all-types* is 'true' or 'false'. If 'true', range expressions are allowed for all key values regardless of type.
`/ncs-config/cli/suppress-range-keyword (boolean) [false]`	*suppress-range-keyword* is 'true' or 'false'. If 'true', the 'range' keyword is not allowed in C- and I-style for range expressions.
`/ncs-config/cli/commit-message-format (string) [ System message at $(time)... Commit performed by $(user) via $(proto) using $(ctx). ]`	The format of the CLI commit messages.
`/ncs-config/cli/suppress-commit-message-context (string)`	This parameter can be given multiple times. A list of contexts for which a commit message is not displayed. A good value is [system], which makes all system-generated commits go unnoticed in the CLI. A context is either the name of an agent (CLI, web UI, NETCONF, SNMP) or a free-form text string if the transaction is initiated from MAAPI.
`/ncs-config/cli/show-subsystem-messages (boolean) [true]`	*show-subsystem-messages* is 'true' or 'false'. If 'true', the CLI displays a system message whenever a connected daemon starts or stops.
`/ncs-config/cli/show-editors (boolean) [true]`	*show-editors* is 'true' or 'false'. If 'true', a list of current editors is displayed when a user enters configure mode.
`/ncs-config/cli/rollback-aaa (boolean) [false]`	If 'true', AAA rules are applied when a rollback file is loaded. Rollback might not be possible if other users made changes that the current user does not have access privileges to.
`/ncs-config/cli/rollback-numbering (rolling | fixed) [fixed]`	*rollback-numbering* is 'fixed' or 'rolling'. If 'rolling', rollback file '0' always contains the last commit. If 'fixed', each rollback gets a unique increasing number.
`/ncs-config/cli/show-service-meta-data (boolean) [false]`	If 'true', backpointers and refcounts are displayed by default when showing the configuration. The default can be overridden by the pipe flags 'display service-meta' and 'hide service-meta'.
`/ncs-config/rest`	Controls how the embedded WAE web server should behave with respect to TCP and SSL.

Parameter	Description
`/ncs-config/rest/enabled (boolean) [false]`	*enabled* is 'true' or 'false'. If 'true', the web server is started.
`/ncs-config/rest/custom-headers`	—
`/ncs-config/rest/custom-headers/header`	—
`/ncs-config/rest/custom-headers/header/name (string)`	—
`/ncs-config/rest/custom-headers/header/value (string)`	This parameter is mandatory.
`/ncs-config/restconf`	Controls settings for the RESTCONF API.
`/ncs-config/restconf/enabled (boolean) [false]`	*enabled* is 'true' or 'false'. If 'true', the RESTCONF API is enabled on the web server used by the web UI. Note that the web UI must also be enabled.
`/ncs-config/restconf/root-resource (string) [restconf]`	The RESTCONF root resource path.
`/ncs-config/webui`	Controls how the embedded WAE web server should behave with respect to TCP and SSL.
`/ncs-config/webui/custom-headers`	*custom-headers* contains any number of header elements, with a valid header-field as defined in RFC7230. The headers are part of HTTP responses on '/login.html', '/index.html', and '/jsonrpc'.
`/ncs-config/webui/custom-headers/header`	—
`/ncs-config/webui/custom-headers/header/name (string)`	—
`/ncs-config/webui/custom-headers/header/value (string)`	This parameter is mandatory.
`/ncs-config/webui/enabled (boolean) [false]`	*enabled* is 'true' or 'false'. If 'true', the web server is started.
`/ncs-config/webui/server-name (string) [localhost]`	The hostname that the web server serves.
`/ncs-config/webui/match-host-name (boolean) [false]`	Specifies whether the web server should only serve URLs that adhere to the server-name defined above. By default, the server-name is 'localhost' and match-host-name is 'false'; any server name can be given in the URL. If you want the server to only accept URLs that adhere to the server-name, enable this setting.
`/ncs-config/webui/cache-refresh-secs (uint64) [0]`	The WAE web server uses a RAM cache for static content. An entry sits in the cache for a number of seconds before it is reread from disk (on access). The default is 0.

Parameter	Description
`/ncs-config/webui/max-ref-entries (uint64) [100]`	Leafref and keyref entries are represented as drop-down menus in the automatically generated web UI. By default, no more than 100 entries are fetched. This element makes this number configurable.
`/ncs-config/webui/docroot (string)`	The location of the document root on disk. If this configurable is omitted, the docroot points instead to the next generation docroot in the WAE distribution.
`/ncs-config/webui/login-dir (string)`	*login-dir* points out an alternative login directory that contains the HTML code used to log in to the web UI. This directory is mapped to https://<*ip-address*>/login. If this element is not specified, the default login/ directory in the docroot is used instead.
`/ncs-config/webui/X-Frame-Options (DENY | SAMEORIGIN | ALLOW-FROM) [DENY]`	By default the *X-Frame-Options* header is set to DENY for the /login.html and /index.html pages. With this header, you can set it to SAMEORIGIN or ALLOW-FROM instead.
`/ncs-config/webui/disable-auth`	—
`/ncs-config/webui/disable-auth/dir (string)`	This parameter can be given multiple times. The *disable-auth* element contains any number of *dir* elements. Each *dir* element points to a directory path in the docroot that should not be restricted by the AAA engine. If no *dir* elements are specified, the following directories and files are not restricted by the AAA engine: '/login' and '/login.html'.
`/ncs-config/webui/allow-symlinks (boolean) [true]`	Allows symlinks in the docroot directory.
`/ncs-config/webui/transport`	Controls which transport services (for example, TCP or SSL) the web server should listen on.
`/ncs-config/webui/transport/tcp`	Controls how the web server TCP transport service should behave.
`/ncs-config/webui/transport/tcp/enabled (boolean) [true]`	*enabled* is 'true' or 'false'. If 'true', the web server uses clear text TCP as a transport service.
`/ncs-config/webui/transport/tcp/ redirect (string)`	Redirects the user to the specified URL. Two macros can be specified: @HOST@ and @PORT@. For example: https://@HOST@:443 or https://192.12.4.3:@PORT@
`/ncs-config/webui/transport/ tcp/ip (ipv4-address | ipv6-address) [0.0.0.0]`	The IP address that the web server should listen on. 0.0.0.0 means that it listens on the port (/ncsconfig/webui/transport/tcp/port) for all IPv4 addresses on the machine.
`/ncs-config/webui/transport/ tcp/port (port-number) [8008]`	*port* is a valid port number to use in combination with the address in /ncs-config/webui/transport/tcp/ip.
`/ncs-config/webui/transport/tcp/extra-listen`	A list of additional IP address and port pairs that the web server should also listen on.

Parameter	Description
`/ncs-config/webui/ transport/tcp/extra-listen/ip (ipv4-address | ipv6-address)`	—
`/ncs-config/webui/ transport/tcp/extra-listen/port (port-number)`	—
`/ncs-config/webui/ transport/ssl`	Controls how the web server SSL transport service should behave. SSL is widely deployed on the Internet; virtually all online shopping and bank transactions are done with SSL encryption. There are many good sources that describe SSL in detail; for example, http://www.tldp.org/HOWTO/SSL-Certificates-HOWTO/ describes how to manage certificates and keys.
`/ncs-config/webui/ transport/ssl/enabled (boolean) [false]`	*enabled* is 'true' or 'false'. If 'true', the web server uses SSL as a transport service.
`/ncs-config/webui/transport/ ssl/redirect (string)`	Redirects the user to the specified URL. Two macros can be specified: @HOST@ and @PORT@. For example: http://@HOST@:80 or http://192.12.4.3:@PORT@
`/ncs-config/webui/transport/ssl/ip (ipv4-address | ipv6-address) [0.0.0.0]`	The IP address on which the web server listens for incoming SSL connections. 0.0.0.0 means that it listens on the port (/ncs-config/webui/transport/ssl/port) for all IPv4 addresses on the machine.
`/ncs-config/webui/ transport/ssl/port (port-number) [8888]`	*port* is a valid port number to use in combination with /ncs-config/webui/transport/ssl/ip.
`/ncs-config/webui/transport/ssl/extra-listen`	A list of additional IP address and port pairs on which the web server listens for incoming SSL connections.
`/ncs-config/webui/ transport/ssl/extra-listen/ip (ipv4-address | ipv6-address)`	—
`/ncs-config/webui/ transport/ssl/extra-listen/port (port-number)`	—
`/ncs-config/webui/transport/ ssl/key-file (string)`	Specifies the file that contains the private key for the certificate. Read more about certificates in /ncs-config/webui/ transport/ssl/cert-file. If this configurable is omitted, the keyFile points instead to a built-in, self-signed certificate/key in the WAE distribution. Note: Only use this certificate/key for test purposes.

Parameter	Description
`/ncs-config/webui/transport/ssl/cert-file (string)`	Specifies the file that contains the server certificate. The certificate is either a self-signed test certificate or a genuine, validated certificate bought from a certificate authority (CA). If this configurable is omitted, the keyFile points instead to a built-in, self-signed certificate/key in the WAE distribution. Note: Only use this certificate/key for test purposes. The WAE distribution comes with a server certificate that can be used for testing (${NCS_DIR}/var/ncs/webui/ cert/host.{cert,key}). This server certificate has been generated using a local CA certificate: $ openssl OpenSSL> genrsa -out ca.key 4096 OpenSSL> req -new -x509 -days 3650 -key ca.key - out ca.cert OpenSSL> genrsa -out host.key 4096 OpenSSL> req -new -key host.key -out host.csr OpenSSL> x509 -req -days 365 -in host.csr -CA ca.cert \ -CAkey ca.key -set_serial 01 -out host.cert
`/ncs-config/webui/transport/ssl/ca-cert-file (string)`	Specifies the file that contains the trusted certificates to use during client authentication and to use when attempting to build the server certificate chain. The list is also used in the list of acceptable CA certificates passed to the client when a certificate is requested. The WAE distribution comes with a CA certificate that can be used for testing (${NCS_DIR}/var/ncs/ webui/ca_cert/ca.cert). This CA certificate has been generated as shown above.

`/ncs-config/webui/transport/ssl/verify (1 | 2 | 3) [1]`	Specifies the level of verification the server does on client certificates:
	• 1—No verification.
	• 2—The server asks the client for a certificate but does not fail if the client does not supply one.
	• 3—The server requires the client to supply a client certificate.
	
	If ca-cert-file has been set to the ca.cert file generated above, you can verify that it works by using:
	
	$ openssl s_client -connect 127.0.0.1:8888 \ -cert client.cert -key client.key
	
	For this to work, client.cert must have been generated using the ca.cert from above:
	
	$ openssl OpenSSL> genrsa -out client.key 4096 OpenSSL> req -new -key client.key -out client.csr OpenSSL> x509 -req -days 3650 -in client.csr -CA ca.cert \ -CAkey ca.key -set_serial 01 -out client.cert

`/ncs-config/webui/transport/ssl/depth (uint64) [1]`	Specifies the depth of certificate chains the server is prepared to follow when verifying client certificates.

Parameter	Description
`/ncs-config/webui/transport/ ssl/ciphers (string) [DEFAULT]`	Specifies the cipher suites for the server to use. The ciphers are a colon-separated list from the following set: ECDHEECDSA-AES256-SHA384, ECDHE-RSA-AES256-SHA384, ECDH-ECDSA-AES256-SHA384, ECDH-RSA-AES256-SHA384, DHE-RSA-AES256-SHA256, DHE-DSS-AES256-SHA256, AES256-SHA256, ECDHE-ECDSA-AES128-SHA256, ECDHE-RSA-AES128-SHA256, ECDHECDSA-AES128-SHA256, ECDH-RSA-AES128-SHA256, DHE-RSA-AES128-SHA256, DHEDSS-AES128-SHA256, AES128-SHA256, ECDHE-ECDSA-AES256-SHA, ECDHE-RSA-AES256-SHA, DHE-RSA-AES256-SHA, DHE-DSS-AES256-SHA, ECDH-ECDSA-AES256-SHA, ECDHRSA-AES256-SHA, AES256-SHA, ECDHE-ECDSA-DES-CBC3-SHA, ECDHE-RSA-DES-CBC3-SHA, EDH-RSA-DES-CBC3-SHA, EDH-DSS-DES-CBC3-SHA, ECDH-ECDSA-DES-CBC3-SHA, ECDH-RSA-DES-CBC3-SHA, DES-CBC3-SHA, ECDHE-ECDSA-AES128-SHA, ECDHE-RSAAES128-SHA, DHE-RSA-AES128-SHA, DHE-DSS-AES128-SHA, ECDH-ECDSA-AES128-SHA, ECDH-RSA-AES128-SHA, AES128-SHA, ECDHE-ECDSA-RC4-SHA, ECDHE-RSA-RC4-SHA, RC4-SHA, RC4-MD5, EDH-RSA-DES-CBC-SHA, ECDH-ECDSA-RC4-SHA, ECDH-RSA-RC4-SHA, and DES-CBC-SHA, or the word "DEFAULT" (use the listed set except the suites using DES, RC4, or MD5 algorithms) See the OpenSSL manual page ciphers(1) for the definition of the cipher suites. Note: The general cipher list syntax described in ciphers(1) is not supported.
`/ncs-config/webui/transport/ ssl/protocols (string) [DEFAULT]`	Specifies the SSL/TLS protocol versions for the server to use as a whitespace-separated list from the set sslv3 tlsv1 tlsv1.1 tlsv1.2, or the word "DEFAULT" (use all supported protocol versions except sslv3).
`/ncs-config/webui/cgi`	CGI-script support.
`/ncs-config/webui/cgi/ enabled (boolean) [false]`	*enabled* is 'true' or 'false'. If 'true', CGI-script support is enabled.
`/ncs-config/webui/cgi/ dir (string) [cgi-bin]`	The directory path to the location of the CGI-scripts.
`/ncs-config/webui/cgi/ request-filter (string)`	Specifies that characters not specified in the regexp should be filtered out silently.
`/ncs-config/webui/cgi/ max-request-length (uint16)`	Specifies the maximum number of characters in a request. All characters that exceed this limit are silently ignored.
`/ncs-config/webui/cgi/php`	PHP support.
`/ncs-config/webui/cgi/php/ enabled (boolean) [false]`	*enabled* is 'true' or 'false'. If 'true', PHP support is enabled.
`/ncs-config/webui/ idle-timeout (xs:duration) [PT30M]`	The maximum idle time before terminating a web UI session. PT0M means no timeout. The default is PT30M (30 minutes).

Parameter	Description
/ncs-config/webui/ absolute-timeout (xs:duration) [PT60M]	The maximum absolute time before terminating a web UI session. PT0M means no timeout. The default is PT60M (60 minutes).
/ncs-config/webui/ rate-limiting (uint64) [1000000]	The maximum number of JSON-RPC requests allowed every hour. 0 means infinity. The default is 1 million.
/ncs-config/webui/ audit (boolean) [true]	*audit* is 'true' or 'false'. If 'true', JSON-RPC/CGI requests are logged to the audit log.
/ncs-config/japi	Java-API parameters.
/ncs-config/japi/new-session-timeout (xs:duration) [PT30S]	The timeout for a data provider to respond to a control socket request; see DpTrans. If the Dp fails to respond within the given time, it is disconnected.
/ncs-config/japi/query-timeout (xs:duration) [PT120S]	The timeout for a data provider to respond to a worker socket query; see DpTrans. If the Dp fails to respond within the given time, it is disconnected.
/ncs-config/japi/connect-timeout (xs:duration) [PT60S]	The timeout for a data provider to send an initial message after connecting the socket to the WAE server. If the Dp fails to initiate the connection within the given time, it is disconnected.
/ncs-config/japi/object-cache-timeout (xs:duration) [PT2S]	The timeout for the cache used by the getObject() and iterator(),nextObject() callback requests. WAE caches the result of these calls and serves getElem() requests from northbound agents from the cache. Setting this timeout too low causes the callbacks to be non-functional. For example, getObject() can be invoked for each getElem() request from a northbound agent.
/ncs-config/japi/event-reply-timeout (xs:duration) [PT120S]	The timeout for the reply from an event notification subscriber for a notification that requires a reply; see the Notif class. If the subscriber fails to reply within the given time, the event notification socket is closed.
/ncs-config/netconf-north-bound	Controls how the NETCONF agent should behave with respect to NETCONF and SSH.
/ncs-config/netconf-north-bound/ enabled (boolean) [true]	*enabled* is 'true' or 'false'. If 'true', the NETCONF agent is started.
/ncs-config/netconf-north-bound/ transport	Controls which transport services (TCP or SSH) the NETCONF agent should listen on.
/ncs-config/netconf-north-bound/ transport/ssh	Controls how the NETCONF SSH transport service should behave.
/ncs-config/netconf-north-bound/ transport/ssh/enabled (boolean) [true]	*enabled* is 'true' or 'false'. If 'true', the NETCONF agent uses SSH as a transport service.
/ncs-config/netconf-north-bound/ transport/ssh/ip (ipv4-address | ipv6-address) [0.0.0.0]	*ip* is an IP address that the WAE NETCONF agent listens on. 0.0.0.0 means that it listens on the port (/ncs-config/netconf-north-bound/transport/ssh/port) for all IPv4 addresses on the machine.

Parameter	Description
/ncs-config/netconf-north-bound/transport/ssh/port (port-number) [2022]	*port* is a valid port number to use in combination with /ncs-config/netconf-north-bound/transport/ssh/ip. The standard port for NETCONF over SSH is 830.
/ncs-config/netconf-north-bound/transport/ssh/extra-listen	A list of additional IP address and port pairs that the WAE NETCONF agent listens on.
/ncs-config/netconf-north-bound/transport/ssh/extra-listen/ip (ipv4-address | ipv6-address)	—
/ncs-config/netconf-north-bound/transport/ssh/extra-listen/port (port-number)	—
/ncs-config/netconf-north-bound/transport/tcp	NETCONF over TCP is not standardized, but it can be useful during development (for example, to use netcat for scripting). It is also useful when using your own proprietary transport. You can set up the NETCONF agent to listen on localhost and then proxy it from your transport service module.
/ncs-config/netconf-north-bound/transport/tcp/enabled (boolean) [false]	*enabled* is 'true' or 'false'. If 'true', the NETCONF agent uses clear text TCP as a transport service.
/ncs-config/netconf-north-bound/transport/tcp/ip (ipv4-address | ipv6-address) [0.0.0.0]	*ip* is an IP address that the WAE NETCONF agent listens on. 0.0.0.0 means that it listens on the port (/ncs-config/netconf-north-bound/transport/tcp/port) for all IPv4 addresses on the machine.
/ncs-config/netconf-north-bound/transport/tcp/port (port-number) [2023]	*port* is a valid port number to use in combination with /ncs-config/netconf-north-bound/transport/tcp/ip.
/ncs-config/netconf-north-bound/transport/tcp/extra-listen	A list of additional IP address and port pairs that the WAE NETCONF agent listens on.
/ncs-config/netconf-north-bound/transport/tcp/extra-listen/ip (ipv4-address | ipv6-address)	—
/ncs-config/netconf-north-bound/transport/tcp/extra-listen/port (portnumber)	—
/ncs-config/netconf-north-bound/extended-sessions (boolean) [false]	If extended-sessions are enabled, all WAE sessions can be terminated using <kill-session>. Not only can other NETCONF sessions be terminated, but also CLI sessions, web UI sessions, and so on. If a session holds a lock, its session ID is returned in the <lock-denied>, instead of '0'. This extension is not covered by the NETCONF specification; therefore, it is false by default.

Parameter	Description
`/ncs-config/netconf-north-bound/idle-timeout (xs:duration) [PT0S]`	The maximum idle time before terminating a NETCONF session. If the session is waiting for notification or has a pending confirmed commit, the idle timeout is not used. The default value is 0, which means no timeout.
`/ncs-config/netconf-north-bound/rpc-errors (close | inline) [close]`	If *rpc-errors* is 'inline' and an error occurs during the processing of a <get> or <get-config> request when WAE tries to fetch data from a data provider, WAE generates an rpc-error element in the faulty element, and continue to process the next element. If an error occurs and *rpc-errors* is 'close', WAE closes the NETCONF transport.
`/ncs-config/netconf-north-bound/max-batch-processes (uint32 | unbounded) [unbounded]`	Controls the number of concurrent NETCONF batch processes. A batch process can be started by the agent if a new NETCONF operation is implemented as a batch operation.
`/ncs-config/netconf-north-bound/capabilities`	Controls which NETCONF capabilities to enable.
`/ncs-config/netconf-north-bound/capabilities/url`	Turns on the URL capability options to support.
`/ncs-config/netconf-north-bound/capabilities/url/enabled (boolean) [false]`	*enabled* is 'true' or 'false'. If 'true', the URL NETCONF capability is enabled.
`/ncs-config/netconf-north-bound/capabilities/url/file`	Controls how the URL file support should behave.
`/ncs-config/netconf-north-bound/capabilities/url/file/enabled (boolean) [true]`	*enabled* is 'true' or 'false'. If 'true', the URL file scheme is enabled.
`/ncs-config/netconf-north-bound/capabilities/url/file/root-dir (string)`	*root-dir* is a directory path on disk where ConfD stores the result from an NETCONF operation using the URL capability. This parameter must be set if the file URL scheme is enabled.
`/ncs-config/netconf-north-bound/capabilities/url/ftp`	Controls how the URL FTP scheme should behave.
`/ncs-config/netconf-north-bound/capabilities/url/ftp/enabled (boolean) [true]`	*enabled* is 'true' or 'false'. If 'true', the URL FTP scheme is enabled.
`/ncs-config/netconf-north-bound/capabilities/url/sftp`	Controls how the URL SFTP scheme should behave.
`/ncs-config/netconf-north-bound/capabilities/url/sftp/enabled (boolean) [true]`	*enabled* is 'true' or 'false'. If 'true', the URL SFTP scheme is enabled.
`/ncs-config/netconf-north-bound/capabilities/inactive`	Controls the inactive capability option.

Parameter	Description
`/ncs-config/netconf-north-bound/capabilities/inactive/enabled (boolean) [true]`	*enabled* is 'true' or 'false'. If 'true', the 'http://tail-f.com/ns/netconf/inactive/1.0' capability is enabled.
`/ncs-config/southbound-source-address`	Specifies the source address to use for southbound connections from WAE to devices. In most cases the source address assignment is best left to the TCP/IP stack in the OS, because an incorrect address might result in connection failures. However, if the stack could choose more than one address, and you need to restrict the choice to one address, these settings can be used.
`/ncs-config/southbound-source-address/ipv4 (ipv4-address)`	The source address to use for southbound IPv4 connections. If not set, the source address is assigned by the OS.
`/ncs-config/southbound-source-address/ipv6 (ipv6-address)`	The source address to use for southbound IPv6 connections. If not set, the source address is assigned by the OS.
`/ncs-config/ha`	—
`/ncs-config/ha/enabled (boolean) [false]`	If 'true', HA mode is enabled.
`/ncs-config/ha/ip (ipv4-address | ipv6-address) [0.0.0.0]`	The IP address that WAE listens to for incoming connections from other HA nodes.
`/ncs-config/ha/port (port-number) [4570]`	The port number that WAE listens to for incoming connections from other HA nodes.
`/ncs-config/ha/tick-timeout (xs:duration) [PT20S]`	Defines the timeout between keepalive ticks sent between HA nodes. The value 'PT0' means that no keepalive ticks are ever sent.
`/ncs-config/scripts`	It is possible to add scripts to control various things in WAE, such as post-commit callbacks. New CLI commands can also be added. The scripts must be stored under /ncs-config/scripts/dir, where there is a subdirectory for each script category. For some script categories it suffices to add a script in the correct subdirectory to enable the script. For others some configuration must be done.
`/ncs-config/scripts/dir (string)`	This parameter can be given multiple times. The directory path to the location of plug-and-play scripts. The scripts directory must have the following subdirectories: scripts/command/ post-commit/
`/ncs-config/large-scale`	—
`/ncs-config/large-scale/lsa`	—
`/ncs-config/large-scale/lsa/enabled (boolean) [false]`	Enables Layered Service Architecture (LSA), which requires a separate Cisco Smart License.