Network Model Configuration—Cisco 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)`	Terminates or confirms a pending confirming commit. A pending confirming commit is terminated if the CLI session is canceled 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.
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. From Expert Mode, **Configuration editor**, 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`**.	`# 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 SR-PCE (topo-bgpls-xtc-nimo). **Note** This example assumes that the network access and an SR-PCE agent have been configured and are running.	1. From Expert Mode, **Configuration editor**, 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: **Field** / **User Input**: **network-access**: **`as54001`** **xtc-host**: **`xt11`** **backup-xtc-host**: **`xt12`** **asn**: **`54001`** **igp-protocol**: **`isis`** **extended-topology-discovery**: **`true`**	`# 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`**

Configuration Type	Expert Mode	CLI Equivalent

Consolidate network models.	1. From Expert Mode, **Configuration editor**, 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**.
		`# networks network `**`as54001`**` nimo aggregator sources `**`as54001_topo`**
		`# networks network `**`as54001`**` nimo aggregator sources `**`as54001_xtclsp`**
		`# networks network `**`as54001`**` nimo aggregator sources `**`as54001_conflsp`**
		`# 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 12
2. Configure a network access profile.	Configure a Network Access Profile, on page 14
3. Configure agents. **Note** This step is only required for collecting SR-PCE or multi-layer information.	• Configuring SR-PCE Agents Using the Expert Mode • Configure the Configuration Parsing Agent Using the Expert Mode
4. Create a network and collect basic topology data.	Create a Network Model, on page 14
5. Configure additional data collections or NIMO capabilities.	Configure Additional NIMOs, on page 16
6. (Optional) Configure the Scheduler.	Scheduler Configuration
7. Configure and view plan archives.	Configure the Archive Using the WAE CLI, on page 16

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.

```
# 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.

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

For SNMPv3, you can set the following options:

```
# 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
```

Note Even though there are options to select authentication and encryption with *no-auth-no-priv*, these values are not used in the backend and are optional.

Example

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

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

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

user@wae(config)# 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)# 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)# 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 12.

Step 1 Enter the following commands:

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

```
# 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>
```

Note Node filter does not work with `ip-manage`.

Step 3 Commit the configuration:

```
# commit
```

Example

For example:

```
# wae nimos network-access network-access as64001 auth-group ABCgroup
# wae nimos network-access network-access as64001 snmp-group snmp_v3_01
# wae nimos network-access network-access as64001 node-access 10.1.1.1 ip-manage 10.18.20.121
# wae nimos network-access network-access as64001 node-access 10.2.2.2 ip-manage 10.18.20.122
# commit

# wae nimos network-access network-access netaccess_01 auth-group ABCgroup
# 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 Topology Collection Using the IGP Database and Topology Collection Using SR-PCE.

Note You have the option to load an existing plan file to create a network model. See Load Plan Files, on page 15.

Before you begin

- Device access and network access must be configured.

- If creating a network running SR-PCE, confirm that SR-PCE agents have been configured. For more information, see Configuring SR-PCE Agents Using the Expert Mode.

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.

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.

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>
```

Configure the Archive Using the WAE CLI

You can also Configure the Archive Using the Network Model Composer.

Step 1 Launch the WAE CLI and enter configuration mode.

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

Step 2 Configure the archive directory and select whether to get archive data from a file.

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

For example:

```
(config)# networks network Network_123 plan-archive archive-dir /archive/planfiles/Network_123
(config)# networks network <network_model_name> plan-archive source filename
(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.

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 Using the WAE CLI, on page 16.

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 15.