ERICSSON

COM Command-Line Interface

Common Operation and Maintenance

INTERWORK DESCRIPTION

4/155 19-CAA 901 2587/6 Uen A1

Contents

1 Introduction

This document describes the Ericsson Command-Line Interface (ECLI), which is a proprietary Ericsson CLI standard.

The CLI is a terminal-based CLI, that allows the user to monitor and manage the Managed Element (ME). The CLI enables the user to interact with the Management Information Base (MIB) through common, generic-purpose commands.

1.1 Key Features of CLI

The key features of the CLI are described in Table 1.

Table 1 Key Features of the CLI

Feature	Description
Access control	CLI commands accessing or manipulating the MIB are subject to authorization. If the user has no permission to access an MO instance or attribute, then operations behave as if the MO instance does not exist. For details on access control, see the product-specific Security Management documentation.
Auto-completion	By pressing **Tab**, all possible CLI command completions are displayed and unique completions are added to the command line. For details, see Section 2.8 AutoCompletion on page 21.
Case correction	Auto-completed items entered by the user in the CLI commands are automatically changed to the case-defined in the model. For details, see Section 2.9 Case Correction on page 27.
CLI modes	2 CLI modes are supported. Exec mode is intended for observation and executing actions, and Config mode for configuration scenarios. For details, see Section 2.3 CLI Mode on page 8.
CLI prompt	The CLI prompt is configurable and provides information on the CLI mode and CLI position. For details, see Section 2.6 CLI Prompt on page 15.
Configurable CLI properties	CLI properties (command history, page break, script mode, width, and prompt configuration) can be changed for the CLI session.

Feature	Description
Configuration export and import	With **`show-config`**, the system configuration can be displayed in a format that is also a valid input for the CLI. Thus, copy/paste or terminal input/output redirection allow configuration copy. For details, see Section 3.4 Display MO Instances on page 71.
Context-sensitive help	By pressing the **?** key, a description on the CLI command element is displayed. For details, see Section 2.7 CLI Help on page 16.
Model driven	The CLI command elements and their properties are defined in the MOM as MOCs, attributes, and actions.
Navigation	The position in the MO tree can be changed. The position determines the context of the CLI command. For details, see Section 2.5 CLI Position on page 13
Security	A CLI session is running securely over SSH.
Scripting	CLI scripts can be created by feeding the CLI commands to, and parsing the output from, the SSH client.
Transactions	Configuration changes are applied through atomic transactions. Thus, it is ensured that all or none of the operations are executed. For details, see Section 2.4 Transaction on page 9.

Note: The examples in this document are based on models that are subject of change.

1.2 Prerequisites

This section describes the prerequisites, which must be fulfilled before using the CLI.

1.2.1 Conditions

The following conditions must apply:

- A standard Secure Shell (SSH) version 2 client is available. Terminal type VT100 is supported. For information on terminal types, see Section 4.1 Terminal Types on page 113.
- The user has IP access to the Operation and Maintenance (O&M) address of the ME.
- The user has a valid O&M user account.
- Suitable firewall settings enable access to the CLI port.

- The user has prior knowledge of generic O&M concepts. For more information, refer to *Glossary of Terms and Acronyms* and *COM Management Guide*.

2 CLI Concepts

This section describes the CLI concepts.

2.1 CLI Session

This section describes how to start and close CLI sessions.

Multiple sessions can exist at a time. The maximal number of parallel sessions is 256.

2.1.1 Start a CLI Session

Log on to CLI Session over SSH (Deprecated)

Note: **INSTRUCTION FOR DOCUMENT REUSE:** Use this section only if the deprecated method is supported by the product.

To log on to the CLI:

1. Start an SSH session with the following options:
   - **port number** – TCP port `22` is default.
   - **subsystem** – Use subsystem `cli`.
   - **tty allocation** – Force `pseudo-tty` allocation.
   - **credentials** – Username and password.

The following is an example of logon with an OpenSSH client:

```
ssh <user>@<target_host> -p 22 -t -s cli
```

Example 1 Logon with an OpenSSH Client

Log on to CLI Session over SSH

Note: **INSTRUCTION FOR DOCUMENT REUSE:** Use this section only if the SSH management feature is supported by the product.

For more information on COM Secure Shell Daemon (SSHD) Management, refer to *COM System Architecture Description*.

To log on to the CLI:

1. Start SSH session with the following options:

- **port number** – TCP port `22` is default.
- **credentials** – Username and password.

Note: **INSTRUCTION FOR DOCUMENT REUSE:** Add product specific options and default values.

Root user access is denied.

The following is an example of logon with an OpenSSH client:

```
ssh <user>@<target_host> -p 22
```

Example 2 Logon with OpenSSH Client

The following is the same for both previous cases.

Logon Succeeded

When logon is successful, the CLI startup message is printed. The CLI session starts in Exec mode, and the CLI prompt is displayed.

Unsuccessful Logon

Logon can fail because of any of the reasons specified in Table 2.

Table 2 Logon Error Messages

Error Message	Recommended Action
`SSH session timeout`	Check the IP address and port accessibility.
`Invalid Credentials`	Get a valid pair of user account and password from the system administrator.
`Connection to COM failed (<socket_error>)`	Check if the CLI service is running. For details, refer to the socket error messages.
`subsystem request failed on channel 0`	Check the SSH server state and verify that its configuration is valid.

2.1.2 CLI Session Inactivity Timer

When a predefined time has passed without any activity in the CLI, the system automatically ends any ongoing transaction and closes the session without notice. Activity in a CLI session means any operation that results in data exchange between the terminal and server.

The inactivity timer value is configured on system level. The default value is 120 seconds.

2.1.3 CLI Session End

A CLI session is closed either as a result of command **`exit`**, or when the session inactivity timer expires.

2.2 CLI Version

The ECLI version can be displayed using the **`version`** command. For more information, see Section 3.16 Display CLI Version on page 111.

The ECLI version is described using three sequence numbers: `<major>.<minor>.<revision>`:

- The major number is incremented when a significant and non-backward compatible change is made, for instance, when a command is deleted, or when the syntax or behavior of a command is changed.
- The minor number is incremented when a significant but backward compatible change is made, for instance, when a new command is added or when new options are added to an existing command.
- The revision number is incremented when a minor change that does not affect the described behavior is made, for instance, when a fault that made a documented function unusable is corrected.

The version number is meant to inform CLI users, including scripts and programs, about what behavior to expect from the ECLI commands. Usability enhancing features, like auto-completion and help, are not considered in the version number.

Commands labeled as optional in this document are also not considered in the version number.

This document describes ECLI 1.2.1.

Note: The ECLI version number is not directly connected to a COM version number. A new version of COM can be released without the ECLI version number being changed, if the COM release does not contain any significant changes to the CLI.

2.2.1 Changes between Ericsson CLI Versions 1.2.0 and 1.2.1

ECLI 1.2.1 changes the name of the `reset` command to `reinit`.

2.2.2 Changes between Ericsson CLI Versions 1.1.1 and 1.2.0

ECLI 1.2.0 adds the command `reset`, to set an attribute or an MO instance back to its default state.

ECLI 1.2.0 adds the command `back`, to navigate back to a previously visited CLI position.

ECLI 1.2.0 adds support for recursive searching with search conditions in the `show` and `show-table` commands, and support for searching for an MO instance to navigate to in the `dn` command.

ECLI 1.2.0 adds support for additional escape sequences in string values.

ECLI 1.2.0 adds support for passphrase strings.

ECLI 1.2.0 adds support for addressing individual values in sequences of simple types using indexes.

ECLI 1.2.0 adds support for TAB completion of multiple attribute assignments on the same command line, and of MO reference values.

2.2.3 Changes between Ericsson CLI Versions 1.1.0 and 1.1.1

ECLI 1.1.1 adds the `--sort|-s` option to the `show`, `show-config`, `show-mib` and `show-table` commands. With this option, MO instances are sorted according to their instance names.

2.2.4 Changes between Ericsson CLI Versions 1.0.0 and 1.1.0

ECLI 1.1.0 adds support for floating point numbers.

ECLI 1.1.0 also adds support for Managed Object (MO) key attributes of integer and enumeration types. While this does not affect the CLI syntax, new error messages specific to these types can be returned when attempting to create an MO instance.

ECLI 1.1.0 also adds support for the `canCreate` and `canDelete` properties on MO relations. While this does not affect the CLI syntax, new error messages specific to these properties can be returned when attempting to create or delete an MO instance.

2.3 CLI Mode

The CLI provides the following two modes:

Exec mode Displays the status of the ME. In this mode, the user enters commands to monitor the ME, display its configuration, and execute actions.

Config mode Used to change the ME configuration. In this mode, the user starts a configuration transaction to the MIB, enters commands to change the ME configuration, and commits the changes.

The CLI mode determines what CLI commands are available. Change between two modes is possible using the **`configure`**, **`abort`**, **`end`** and **`commit`** commands, as shown in Figure 1. Detailed information about what commands are available in what mode is included in Table 12.

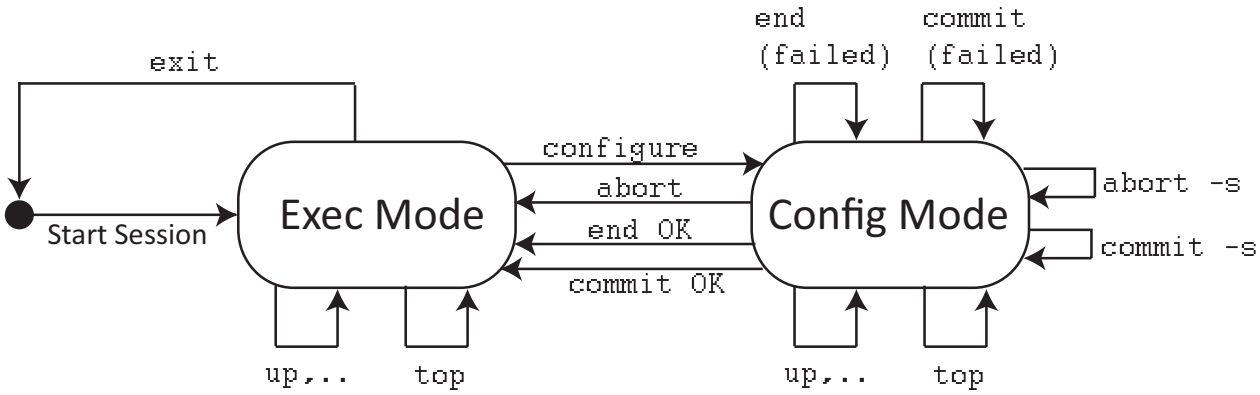

Figure 1 CLI Mode

2.4 Transaction

Changes to configuration information are performed in an atomic way in transactions. If an action executed in Exec mode changes the configuration, the change is automatically committed when the action is executed. In Config mode, all configuration changes, whether from configuration commands or from actions, are committed with the **`commit`** command.

2.4.1 Start

The CLI session starts in Exec mode and can be changed to Config mode by command **`configure`**.

A new transaction starts automatically if the CLI mode is changed to Config mode.

2.4.2 Commit

Command **`commit [-s|--stay]`** validates the transaction and, on success, commits the configuration changes. The CLI position is unchanged, unless the MO instance in the CLI position no longer exists; in this case the CLI position moves to the closest instantiated parent MO location.

Command **commit** without any argument, shown in Example 3, performs a commit and the CLI mode changes to Exec mode.

```
(config-ManagedElement=<node_name>)>userLabel="sweden"
(config-ManagedElement=<node_name>)>commit
(ManagedElement=<node_name>)>
```

Example 3 Command commit

Command **commit -s** or **commit --stay** performs a commit, the CLI remains in Config mode and starts a new transaction instead of returning to Exec mode, as shown in Example 4,.

```
(config-ManagedElement=<node_name>)>userLabel="sweden"
(config-ManagedElement=<node_name>)>commit -s
(config-ManagedElement=<node_name>)>
```

Example 4 Command commit -s

From the CLI perspective, a successful commit command consists of three steps: validate the data in the transaction against model restrictions (for example, multiplicity, cardinality), request validation of the transaction from the middleware, and finally request transaction commit from the middleware. If commit is completed without errors, nothing is displayed by the system. If any of the three steps fail, an error message is displayed and the command does not proceed to the next step.

Note: Use command `validate` before `commit` to verify that the entered changes are valid.

2.4.2.1 Model Validation Phase Error Messages

When the `commit` command fails during the model validation, it means that model restrictions have been violated by the data in the current transaction. Two possible error printouts are displayed by the system:

```
ERROR: Unable to commit incomplete object (<MO_name>)
<error specific information why the validation failed>

ERROR: Current cardinality of <current cardinality>
for class-instance of <MOC_name> is < <min cardinality value>
(lower-limit) for <DN of validated MO>
```

The error-specific part of the first message gives detailed information about why the validation failed. When the model validation is not successful, the transaction is still valid and its state is the same as before the `commit` command was executed.

2.4.2.2 Middleware Validation Phase Error Messages

When the `commit` command fails because the validation does not pass in the middleware, the following error printout is displayed by the system:

```
ERROR: Transaction not committed due to validation errors
Transaction validation failed!
<error specific information why the validation failed,
when available>
```

When this message is displayed, the CLI is unaware of the reason why the validation did not pass but prints any error-specific information that is available through the middleware. The transaction is still valid and the state is the same as before the commit command was executed.

When the `commit` command fails because the validation fails in an expected way in the middleware, the following error printout is displayed by the system:

```
ERROR: Transaction validation failed with error code:
<error code>
```

The error code that is returned from the middleware is displayed and its meaning can be interpreted with help of Table 14. The transaction is still valid and its state is the same as before the commit command was executed unless the error code definition states otherwise.

2.4.2.3 Commit Phase Error Messages

When the command fails during the commit phase the following error printout is displayed by the system:

```
ERROR: Transaction commit failed and uncommitted changes
have been lost
```

In this case, it is not possible to recover the transaction. The transaction and all the changes associated with it are lost.

Warning printout `Invalid location. Cursor points to uninstantiated MO.` notifies the user that the prompt is changed because of cursor points to `uninstantiated` MO. This can happens if commit fails and the MO that the cursor points to was created in the transaction that was lost.

2.4.3 Abort

Command **`abort [-s|--stay]`** discards the changes in the transaction and terminates the transaction.

Command **`abort`** without any argument, shown in Example 5, performs an end and returns to Exec mode.

```
(config-ManagedElement=<node_name>)>userLabel="Ericsson"
(config-ManagedElement=<node_name>)>abort
(ManagedElement=<node_name>)>
```

Example 5 Command abort

Command **abort -s** or **abort --stay**, shown in Example 6, performs an end and starts a new transaction. The CLI position is unchanged, unless the MO instance in the CLI position no longer exists; in this case, the CLI position moves to the closest instantiated parent MO location.

```
(config-ManagedElement=<node_name>)>userLabel="Ericsson"
(config-ManagedElement=<node_name>)>abort -s
(config-ManagedElement=<node_name>)>
```

Example 6 Command abort -s

2.4.4 End

Command **end**, shown in Example 7, returns from Config mode to Exec mode when there are no changes in the configuration transaction. The CLI position is unchanged, unless the MO instance in the CLI position no longer exists; in this case, the CLI position moves to the closest instantiated parent MO location.

Use command **abort** or **commit** to return to the Exec mode after entering configuration changes.

```
(config-ManagedElement=<node_name>)>end
(ManagedElement=<node_name>)>
```

Example 7 Command end

If transactional changes have been made, command **end** returns error message `Configuration changes have been made in the current transaction, use 'abort' or 'commit' to leave config mode.`

2.4.5 Time-Out

At transaction inactivity time-out, the following events are triggered:

- Transaction end that results in removal of transaction content and MO locks.
- State of the transaction is changed from active to timed out.
- Start of transaction cleanup timer that triggers transaction ID and state removal after 3600 seconds, by default.

The first operations entered in a timed out, but not removed, transaction are replied with the following transaction time-out error message:

```
ERROR: Transaction broken, possibly because of a timeout.
Uncommited changes have been lost
```

If the session inactivity timer occurs before transaction time-out, the session is closed and the transaction is immediately ended.

2.5 CLI Position

The CLI position is a Distinguished Name (DN) of an MO instance that identifies a position in the MO tree. The CLI position is a session property that can be changed by navigation commands, like directory change commands in a directory tree of a file system.

2.5.1 Distinguished Name

The DN identifies an MO instance with a comma-separated sequence of `<class_name>=<key_value>` name value pairs.

2.5.2 Local Distinguished Name

Local Distinguished Name (LDN) is a DN that starts at LDN root and provides address of MO instance through its sequence of parent MOs. For COM-based nodes, the LDN root is `ManagedElement=<MEID>`.

The MOs are organized in a hierarchical structure. Each MO instance is uniquely identified in the node by its LDN. The highest MO in a node, the so called root MO, is the `ManagedElement`. This MO represents the whole node.

If an MO is located further down in the MO tree, the LDN must contain the MO classes that identities all parents of that MO, in a sequence going from the root MO down to the MO in question, as shown in Example 8:

```
ManagedElement=<node_name>
ManagedElement=<node_name>,SystemFunctions=1
ManagedElement=<node_name>,SystemFunctions=1,Pm=1
ManagedElement=<node_name>,SystemFunctions=1,Pm=1,PmMeasurementCapabilities=1
```

Example 8 Local Distinguished Name

In Example 8, the `ManagedElement` has a child called `SystemFunctions=1`, which has a child called `Pm=1` representing the performance management, which has a child called `PmMeasurementCapabilities=1` representing the `PmMeasurementCapabilities`. The LDN of the lowest MO, that is, the one called `PmMeasurementCapabilities=1`, contains the address of all successive parents of that MO all the way up to the `ManagedElement`.

2.5.3 Relative Distinguished Name

Relative Distinguished Name (RDN) is the address of an MO instance in relation to its closest parent. It is a single `<name=value>` pair of DNs.

`PmMeasurementCapabilities=1` is the RDN for LDN `ManagedElement=<node_name>,SystemFunctions=1,Pm=1,PmMeasurementCapabilities=1`.

Pm=1 is the RDN for LDN ManagedElement=<node_name>,SystemFunctions=1,Pm=1.

2.5.4 Path

A CLI path can consist of either LDN or RDN.

2.5.5 Valid Positions in Tree

Considering LDN ManagedElement=<node_name>,SystemFunctions=1,SysM=1, the valid positions in the tree are described in Table 3.

Table 3 Valid Positions in Tree

Position in Tree	Description
Root position	The root of the MO tree that contains ManagedElement. Indicated in the DN and <path> field of the CLI prompt as an empty string.
Position at ManagedElement=<node_name>	The position at ManagedElement. Indicated in the DN and RDN field of the CLI prompt as ManagedElement=<node_name>.
Position at SystemFunctions=1	The position at SystemFunctions. Indicated in the RDN field of the CLI prompt as SystemFunctions=1. Indicated in the DN field of the CLI prompt as ManagedElement=<node_name>,SystemFunctions=1.
Position at SysM=1	The position at SysM. Indicated in the RDN field of the CLI prompt as SysM=1. Indicated in the DN field of the CLI prompt as ManagedElement=<node_name>,SystemFunctions=1,SysM=1.

2.5.6 Interpretation of CLI Commands

CLI commands are interpreted in the context of the CLI position, that is, that attribute names, action names, and RDNs are prepended by the DN indicating the CLI position. For example, the following operations show the same attribute from different CLI positions:

```
(config)>show ManagedElement=<node_name>,
SystemFunctions=1,SysM=1,userLabel
```

```
(config-ManagedElement=<node_name>)>show
SystemFunctions=1,SysM=1,userLabel
```

```
(config-SysM=1)>show userLabel
```

```
(ManagedElement=<node_name>)>show
SystemFunctions=1,SysM=1,userLabel
```

```
(SysM=1)>show userLabel
```

2.6 CLI Prompt

The CLI prompt provides information on the status and context of the CLI session in the following format:

- In Exec mode, the default prompt is "`>`". Prompt is changed in execution mode, based on RDN value during navigation, to "`(RDN)>`".
- In Config mode, the default prompt is "`(config)>`". Prompt is changed in configuration mode, based on RDN value during creation or navigation, to "`(config-RDN)>`".

After session start, the prompt configuration can be changed by CLI command `prompt` in any of the CLI modes for the lifetime of the session to contain any elements shown in Table 4.

Table 4 CLI Prompt Configuration Elements

Element	Description
`$default`	The `$default` value is equivalent to `$mode-$rdn` in Config mode and an empty string in Exec mode.
`$dn`	LDN indicating the CLI position relative to the root position.
`$mode`	CLI mode, that is, `config` or `exec`.
`$hostname`	The name of the host the CLI service runs on.
`$nodename`	The key value of the `ManagedElement` MO.
`$rdn`	RDN indicating the CLI position relative to the parent MO.

Element	Description
`$user`	User account name.
<Value: Any user-defined string>	Value can be any user-defined string. It can also contain special characters in escaped hexadecimal format. Supported escaped characters and escape sequences are **\"**, **\'**, **\t**, **\n**, **\b**, **\f**, **\r**, and **\v**. Special characters with escaped hexadecimal form are also supported, in form `\x`, followed by the ASCII code in hexadecimal in two digits. For example, the hash character `#` must be specified as `\x23` and is displayed in the CLI prompt as `#`.

If the CLI prompt is longer than the terminal width, it is continued on the next line.

2.7 CLI Help

The CLI provides online and context-sensitive help. It enables the user to access information and learn about the commands, the MIB, and the Managed Object Model (MOM) without relying on the documentation library.

By pressing the **?** key, the user can request context-sensitive help on the CLI operations and MOM elements (MOCs, attributes, and actions) that are available for the following:

- CLI mode
- CLI position
- CLI command

After the help text, a new prompt without the **?** character is printed. If no help text is available, only a new prompt is printed.

Note: CLI only supports US-ASCII characters. If any non-US-ASCII character is to be displayed in help text, it is displayed as a question mark (`?`) in the CLI terminal.

Verbose Help is provided if the context uniquely identifies single mode element, otherwise only Brief Help on all available elements is listed, as a summary.

Verbose Help is provided when pressing the **?** key on a single element (CLI command or model element).

No help is provided when pressing **?** key on partial command (if context does not find any hit for model elements).

CLI help is not available for the position in key values of the CLI path.

For example, no CLI help is provided in the following cases:

- On an attribute in the **show-config***<path>*,*<attribute_name>* command if the attribute is defined as read-only or restricted.
- On a Managed Object Class (MOC) in the **show-config[***<path>*],*<class_name>* command if the class is defined as system-created and has no MO instance created.
- On a MOC in the **show [***<path>*],*<class_name>* command if the class is defined as not system-created and has no MO instance created.

2.7.1 Brief Help

A description on the Brief Help printout content and format is provided in Table 5.

Table 5 Printout Syntax for Single-Line Help

CLI Element	Printout Syntax
Action	*<action_name>*()*<spaces><truncated_action_description>*
Action Parameter	*<parameter_name><spaces><truncated_parameter_description>*
Class	+*<class_name><spaces><truncated_class_description>*
CLI operation parameter	*<parameter_name><spaces><parameter_description>*
Single-valued attribute, structure member	*<attribute_name><spaces><truncated_attribute_description>*
Structure (multi-valued attribute)	*<attribute_name>*[]*<spaces><truncated_attribute_description>*

The class, attribute, and action descriptions in the printout are displayed in a truncated form (first sentence only). Truncation is not indicated in the printout. The text-formatting characters (tab, new line) are deleted from the description printouts.

For example, by pressing **?** in the ManagedElement=*<node_name>* CLI position in Config mode, the Brief Help information on the ManagedElement MO attributes and the child MOCs are displayed, as shown in Example 9 and Example 10.

```
(config-ManagedElement=<node_name>)>?
dnPrefix                It provides naming context allowing the managed
                        objects to be partitioned into logical domains.
networkManagedElementId Replaces the value component of the RDN in the
                        COM Northbound Interface.
siteLocation            A freetext attribute describing the geographic
                        location of a Managed Element.
userLabel               A freetext string for additional information to
                        assist Managed Element identification.
productIdentity[]       Contains product information for the Managed
                        Element and its Managed Function(s).
+SystemFunctions        This model has a structural purpose to group the
                        management of the system functions of the Managed
                        Element.
+TestRootMoc            Root of CLI test tree.
```

Example 9 Brief Help

```
(config-CallableThing=1)>addNumbers<space>?
--num1                    Number one
--num2                    Number two
```

Example 10 Brief Help on Action Parameters

Indicators and Keywords

The following indicators, also shown in Example 11, are used to indicate the CLI user what syntax is used to interact with the data type:

- `()` action or command that can be executed
- `+` MO instance or struct that can be navigated to

```
config-ManagedElement=<node_name>)>?
actionA() first line of help text
myMultiValueAttr[] first line of help text
+B first line of help text
...

(config-ManagedElement=<node_name>)>
```

Example 11 Indicators in Help Text

2.7.2 Verbose Help

A description on the Verbose Help printout content and format is provided in Table 6.

Table 6 Printout Syntax for Verbose Help

Item	Printout Syntax
Action	`<action_name>() [Return Type]` `<action_description>`

Action parameter	`<parameter_name><parameter_type> [passphrase] [optional/default=<default_value>]`
	
	`<parameter_description>`

Class	`<class_name> MO type [singleton][optional]` `<class_description>`(1)
CLI operation	`<CLI_operation_name>() Command` `<operation_description>`
CLI operation parameter	`<parameter_name>` `<parameter_description>`
Single-valued attribute, structure member	`<attribute_name> <attribute_type> [passphrase] [optional/read only/default=<default_value>/exclusive]`(2) `<attribute_description>`(2) `<attribute_description>`

(1) The `[singleton]` class specifier is present if exactly one instance of this class can exist as a child MO of its actual parent. The `[optional]` class specifier is present if zero is the minimal number of instances of this class as a child MO of its actual parent.
(2) The `[exclusive]` specifier can only be present for struct members, and means that the struct `isExclusive` property is set. See Section 3.2.1 on page 50

The descriptions of class, attribute, and action in the printout are displayed in a complete form without truncating the text and the text formatting characters (tab, new line) are kept.

Examples of Verbose Help are shown in Example 12 through Example 14.

```
>show ManagedElement=<node_name>,SystemFunctions ?
SystemFunctions   MO Type [singleton] [optional]
This model has a structural purpose to group the management of the system functions
of the Managed Element.
```

Example 12 Verbose Help on MOC

```
(config)>ManagedElement=<node_name>,userLabel ?
userLabel   String [optional]
A freetext string for additional information to assist Managed Element identification.
```

Example 13 Verbose Help on Attribute of Type String

```
(config-ManagedElement=<node_name>)>sitelocation ?
siteLocation   String [optional]
A freetext attribute describing the geographic location of a Managed Element.
```

Example 14 Verbose Help on String Attribute

If the help is triggered directly after (without space) the CLI operation or action name, then Verbose Help is provided on the operation, as shown in Example 15.

```
(config-ManagedElement=<node_name>)>show?
show   Command
Display information
```

Example 15 Verbose Help on CLI Operation

If the help is triggered separated by space from the command or action name, then Verbose Help is provided on the operation parameter, as shown in Example 16 through Example 21.

```
(config)>show ?
--moc                          Option to select a specific Child MOC under the current DN
--recursive                    Display all information
--sort                         Sort the MO instances in numerical/alphabetical order
--verbose                      Display verbose information
-m                             Option to select a specific Child MOC under the current DN
-r                             Display all information
-s                             Sort the MO instances in numerical/alphabetical order
-v                             Display verbose information
+ManagedElement                The top-level class in the Common Information Model
                               is Managed Element root Managed Object Class.
```

Example 16 Verbose Help on CLI Command

```
(config-CallableThing=1)>concatString_defValues --str1 ?
--str1                         String [optional/default=com]
String one
```

Example 17 Verbose Help on Action Parameters

```
(config)>show-mib ?
--sort                         Sort the MO instances in numerical/alphabetical order
--verbose                      Display full path of MO instance information
-s                             Sort the MO instances in numerical/alphabetical order
-v                             Display full path of MO instance information
+ManagedElement                The top-level class in the Common Information Model
                               is Managed Element root Managed Object Class.
```

Example 18 Verbose Help on Show-Mib

```
(config)>show --verbose ?
--recursive                    Display all information
--sort                         Sort the MO instances in numerical/alphabetical order
-r                             Display all information
-s                             Sort the MO instances in numerical/alphabetical order
+ManagedElement                The top-level class in the Common Information Model
                               is Managed Element root Managed Object Class.
```

Example 19 Verbose Help on Show Verbose

```
(config>show-table ?
--moc                          Option to select a specific Child MOC under the current DN
--recursive                    Display all information
-m                             Option to select a specific Child MOC under the current DN
-r                             Display all information
+ManagedElement                The top-level class in the Common Information Model
                               is Managed Element root Managed Object Class.
```

Example 20 Verbose Help on Show-Table

```
(config)>show | filter  ?
--after                         Print number of lines of trailing context after matching lines
--before                        Print number of lines of leading context before matching lines
--ignore                        Ignore case distinctions in both the pattern and the input
--invert                        Invert the sense of matching, to select non-matching lines
-A                              Print number of lines of trailing context after matching lines
-B                              Print number of lines of leading context before matching lines
-i                              Ignore case distinctions in both the pattern and the input
-v                              Invert the sense of matching, to select non-matching lines
<pattern>                       The text used for pattern matching
```

Example 21 Verbose Help on Filter

2.8 AutoCompletion

By pressing **Tab** at any position of the CLI command, completion can be requested on all possible continuations of the CLI command and model-defined elements to which the user is authorized.

Depending on the entered command, the response can be as shown in Table 7.

Table 7 Responses at Auto-Completion

Scenario	Response
Multiple valid continuations exist	All valid continuations of the CLI command are printed in `Completion Possibility List` with the following content and listing order: • CLI operation name, for example, **`show`** or **`history`**. • CLI operation parameter name, for example, **`-v`** or **`--verbose`** for operation **`show`**. • MOC name. • Attribute name and structure member name. • Action name. • Action Parameter Name. • Value of key attribute of existing MOs. • `<new>` indicates, in Config mode, that a key attribute value of a new MO can be specified. • `<value>` indicates, in Config mode, that a new value for an attribute (not `readOnly`) can be specified. • `"` indicates in Config mode that a string attribute (not `readOnly`) value can be specified in quotation marks. • `[` indicates in Config mode that sequence attribute (not `readOnly`) values can be specified within square brackets. • Comma (`,`) indicates that the MO identified by the RDN can have an attribute, a child MO, or an action. • `<cr>` indicates that the entered CLI command is valid by itself, and command execution can be requested by pressing the **CR** key or the **Enter** key. • `<space>` indicates that a space can be entered after an action parameter name or the value.

Table 7 Responses at Auto-Completion

Scenario	Response
Exactly one valid continuation (unique match) exists	Auto-completion automatically adds them, including the following elements: • Equal sign (=), comma (,) and space (“ ”) characters. • MOC and action names. • Attribute names. • Action Parameter Names. • Value of single-valued attributes (enumeration, string, integer, and boolean). • Structure member names. • Struct member names in sequence of struct if the struct has the key member. • Enum structure member values.
The entered CLI command is invalid	Auto-completion or a completion possibility list are not provided.

Auto-completion is provided only for valid CLI commands for which the user is authorized, in a context-sensitive way, for example, as follows:

- Completion is provided for DNs as parameter of the **`show`** operation if the MO exists and the user has read privilege to the MO.
- Completion is provided for attribute names as parameter of the **`show`** operation if the attribute has a value assigned and the user has read privilege to the attribute.
- Completion is provided for an attribute name in an attribute value change operation if the attribute is not defined as `readOnly` or restricted, and if the user has write privileges to the attribute.
- Read-only and restricted attributes are not auto-completed as parameters of **`show-config`**.
- Completion is provided for parameters of an action if the parameter exists.

2.8.1 Keys Activating Auto-Completion

The keys in Table 8 activate auto-completion in the listed contexts if the entered partial name uniquely identifies the name of the command or its parameter.

Table 8 Keys Activating Auto-Completion

Key	Description
Comma (,)	Completes DNs if they can be followed by a further MOC name or DN.
Enter (CR)	Completes operation, attribute, action names, and DNs and then executes the completed command, if there is a unique match.
Equal sign (=)	Completes DN and attribute names if they can be followed by an = character.
Space (“ ”)	Completes command names that can be followed by parameters.
Tab	Provides auto-completion or a completion possibility list. Or both.

Note: Comma (,), equal sign (=) and space (“ ”) have to be removed from the set of keys activating auto-completion in the next version of the CLI.

2.8.2 Example of Completion Possibility List

Examples of a completion possibility list is shown in Example 22 through Example 25.

```
(config)>show<TAB>
show-config
show-dn
show-mib
show-table
<cr>
<space>
```

Example 22 Completion Possibility List of Show without Space

```
(config-CallableThing=1)>add<TAB>
addNumbers
addNumbers_defValues
addNumbersNameClash
addsoc
```

Example 23 Completion Possibility List for Actions without Space

```
config-CallableThing=1)>addNumbers --n<TAB>
--num1
--num2
(config-CallableThing=1)>addNumbers --num
```

Example 24 Completion Possibility List for Action Parameters

```
(config)>show <TAB>
--moc
--recursive
--sort
--verbose
-m
-r
-s
-v
ManagedElement=<node_name>
<cr>
```

Example 25 Completion Possibility List of Show with Space

```
(config-ManagedElement=<node_name>)>show | filter   <TAB>
--after
--before
--ignore
--invert
-A
-B
-i
-v
<pattern>
```

Example 26 Completion Possibility List for Filter Parameters

2.8.3 Examples of Auto-Completion

Examples of auto-completion are shown in Table 9.

Table 9 Examples of Auto-Completion

Function	Input	Result
Unique Match for DN with Tab	Press **Tab** after **`show M`**	Triggers auto-completion to **`show ManagedElement=<node_name>`**
Unique Match for Partial DN with Tab	Press **Tab** after **`>show ManagedElement=<node_name>,SystemFunctions=1,FileM=1,LogicalFs=1,FileGroup=SysMMimSchemas,FileInformation=C`**	Triggers auto-completion to **`show ManagedElement=<node_name>,SystemFunctions=1,FileM=1,LogicalFs=1,FileGroup=SysMMimSchemas,FileInformation=Com`** if the **`FileInformation=ComFm.xml`** and **`FileInformation=ComFileM.xml`** MOs exist
Unique Match for DN with Comma	Press **,** after **`show m`**	Triggers auto-completion to **`show ManagedElement=<node_name>,`**
Unique Match for Operation Name with Enter	Press **Enter** after **`c`** in Exec mode	Triggers **`config`** operations
Unique Match for DN with Enter	Press **Enter** after **`show m`**	Triggers auto-completion to and execution of **`show ManagedElement=<node_name>`**
Unique Match for DN with Enter	Press **Enter** with empty command line in Config mode in `Root Position`	Results in navigation to **`ManagedElement=<node_name>`**
Unique Match for MOC with Equal Sign	Press **=** after **`show m`**	Triggers auto-completion to **`show ManagedElement=`**

Table 9 Examples of Auto-Completion

Function	Input	Result
Unique Match for Attribute with Equal Sign	Press **=** after **ManagedElement=*<node_name>*,u**	Triggers auto-completion to **ManagedElement=*<node_name>*,userLabel=**
Unique Match for Operation Name with Space	As an example for the operations: press **Space** after **show-c**	Triggers auto-completion to **show-config**
Multiple Matches for Parameter Names of an Action with Tab	Press **Tab** after **addNumbers** with space	Triggers auto-completion to **(config-CallableThing=*1*)>addNumbers** **--num1** **--num2**
Unique Match for Value of a Parameter of an Action with Tab	Press **Tab** after **addNumbers --num1**	Triggers auto-completion to **(config-CallableThing=1)>addNumbers --num1** ***<value>*** **(config-CallableThing=1)>addNumbers --num1*<*space*>**
Unique Match for Parameter Name of an Action with Tab	Press **Tab** after **addNumbers --num1 20** with space	Triggers auto-completion to **(config-CallableThing=1)>addNumbers --num1 20*<space>*** ***--num2***
Multiple Matches for *<space>*, *<cr>*, and *<value>* of a Parameter with Value of an Action with Tab	Press **Tab** after **addNumbers_defValues --num1 23**	Triggers auto-completion to **(config-CallableThing=1)>addNumbers_defValues --num1 23** ***<value>*** ***<space>*** ***<cr>***

2.8.4 Auto-Completion of MO Reference Type

The input must be quoted to trigger completion for MO references.

Assume a model with MO `D` under `C`, MO `C` under `B`, MO `B` under the root `ManagedElement=1` and the `MoReferenceAttribute` under `C`. This creates the following example output:

```
ManagedElement=1,B=1,C=1>moReferenceAttribute="<tab>
"ManagedElement=1
"D
"..
```

A relative path is always relative to the CLI position.

Assume a model with MO `C1` and `C2` under `B`, MO `B` under the root `ManagedElement=1` and the `MoReferenceAttribute` under `C1`. This creates the following example output:

```
A=1,B=1>C1=1,moReferenceAttribute="<tab>
"ManagedElement=1
"C1
"C2
```

". .

2.9 Case Correction

A unique match of auto-completion automatically triggers case correction (for example, `ntpSERVER` to `NtpServer`) of one CLI command element at a time for the following types:

- MOC name
- Attribute name, struct name, and struct member attribute name
- Enumeration value, that is, enumeration member or literal name
- Action name
- CLI operation and parameter name
- Action Parameter Name

Case correction for string attribute values is not supported.

2.10 Escaping of Special Characters

The CLI supports the US-ASCII character set. In both input and output strings, escape sequences are used to represent non-printable characters, non-US-ASCII and characters with a special meaning in the CLI, as show in Table 10.

Note: There is a slight difference between the escape sequences used in attributes, struct members and action parameters, and those used in MO instance names.

Table 10 Escape Sequences in Strings

Escape Sequence	Description
\n	New line
\r	Carriage return
\t	Tab
\"	Quotation mark
****	Escape character
\?	Question mark (this character does not need to be escaped in quoted input strings)

Escape Sequence	Description
\!	Exclamation mark (this character does not need to be escaped in quoted input strings)
\#	Hash (this character does not need to be escaped in quoted input strings)
\xNN	Any character in hexadecimal format (2 hexadecimal digits) NN is any character "0" through "9" or "A" through "F" or "a" through "f". The sequence **\x00**, which translates to the string termination character in C is not allowed. Non US-ASCII characters are not allowed (**NN** is greater than **\x7F**).
\NN	In MO instance names only: Any character in hexadecimal format (2 hexadecimal digits). NN is any character "0" through "9" or "A" through "F" or "a" through "f". This format is mandated by 3GPP TS 32.300, Naming convention for MO. This escaping format is always used when DNs are displayed.

```
(config-ManagedElement=<node_name>)>userLabel=hello\nworld
(config-ManagedElement=<node_name>)>show userLabel
userLabel="hello\nworld"
(config-ManagedElement=<node_name>)>
```

Example 27 Set String Attribute with Special Character

```
(config-aSimpleStruct)>str1=hello\!world
(config-aSimpleStruct)>show str1
str1="hello!world"
(config-aSimpleStruct)>
```

Example 28 Set struct Member with Special Character

If a non-supported escape sequence is entered, then CLI displays an error message.

```
(config-ManagedElement=<node_name>)>userLabel=temp\76value
ERROR: Invalid value 'temp\76value' for attribute 'userLabel'.
This is not a valid escape sequence
(config-ManagedElement=<node_name>)>
```

Example 29 Set String Attribute with Non-Special Character

Entered escape sequences are internally converted into the characters they represent before being stored. When a string value is printed by the CLI,

characters are escaped when needed. For example, a quotation character within a string is printed as `\"`.

The exception is DNs, where an escape sequence is only converted into the character it represents when the character is allowed according to 3GPP TS 32.300. Otherwise escaping is translated to the `\NN` format (as mandated by 32.300).

2.11 Visibility Levels

The behavior of CLI command output depends on the following visibility levels returned for model elements. For more details on life cycle controller, refer to *Glossary of Terms and Acronyms*.

There are different visibility levels applicable for model elements, as follows:

- **visible** – specifies that user has full access to all model elements.
- **not-visible** – specifies that user cannot display or access model elements.
- **accessible** – specifies that user can only display or access model elements by providing the full name of model element.

The behavior of the visibility for model elements in CLI is shown in Table 11.

Table 11 Visibility Behavior

Resulting Entity Visibility Level	Normal Show Output	`show --verbose [--recursive |-r] or show -v [--recursive |-r]`	`show-config [--verbose| -v]`	Auto-Completion	Help on Empty Command Line (?)	Help on Element Name	Navigation
visible	Displayed	Displayed	Displayed	Displayed	Displayed	Displayed	Supported

Table 11 Visibility Behavior

Resulting Entity Visibility Level	Normal Show Output	`show --verbose [--recursive |-r] or show -v [--recursive |-r]`	`show-config [--verbose| -v]`	Auto-Completion	Help on Empty Command Line (?)	Help on Element Name	Navigation
accessible	Displayed only in the following cases: • when user is located in accessible element • when user runs **`show -r`** from accessible parent element • when user runs **`show`** using DN to location Not displayed when running **`show -r`** from visible parent element	Displayed In this case, status value is added, for example ***<deprecated>***	Displayed	Supported only in following cases: • when user is located in accessible element • when user is located in accessible parent element Not supported when user is located in visible parent element	Supported only in following cases: • when user is located in accessible element • when user is located in accessible parent element Not supported when user is located in visible parent element	Supported	Supported
not-visible	Not displayed If **`show`** is run on DN, error message `Element not visible` is returned.	Not displayed If **`show`** is run on DN, error message `Element not visible` is returned.	Not displayed	Not supported	Not supported	Not supported	Not supported

The default visibility level for all model elements is visible, meaning that obsolete, preliminary, and deprecated elements are set to visible in visibility configuration file for backward compatibility.

Note: The examples are executed with deprecated element set to accessible, obsolete, and the preliminary elements are set to not- visible.

In the examples, consider the visibility level for the following MOC instances:

- `YCurrentThing` MOC is visible
- `YDeprecatedThing` MOC is accessible
- `YObsoleteThing` MOC is not-visible

2.11.1 Auto-Completion

Auto-completion for accessible element `YDeprecatedThing` is not supported as the current location is visible element `YCurrentThing`, as shown in Example 30.

```
(config-YCurrentThing=1)>YDepre<TAB>
```

Example 30 Auto-Completion of Accessible Element

Auto-completion for accessible element `YSampleDeprecatedThing` is supported as the current location is accessible element `YDeprecatedThing`, as shown in Example 31.

```
(config-YDeprecatedThing=1)YSample<TAB>
YSampleDeprecatedThing=1
```

Example 31 Auto-Completion of Accessible Element

2.11.2 Display Information

When **`show`** command is executed from the visible current location, only visible elements, such as `YCurrentThing`, are displayed, as shown in Example 32.

```
(config-YCurrentThing=1)>show
YCurrentThing=1
 userLabel=”UserLabel”
```

Example 32 Command show

When the **`show`** command is executed with **`verbose`** option from visible current location, all visible elements, such as `YCurrentThing`, and accessible elements, such as `YDeprecatedThing`, are displayed. When the life cycle status of an element is anything else than current, the status is included as a tag in verbose mode, as shown in Example 33.

```
(config-YCurrentThing=1)>show --verbose
YCurrentThing=1
  rwattr1=[] <empty>
  userLabel="UserLabelValue"
  yCurrentThingId="1"
  YDeprecatedThing=1 <deprecated>
```

Example 33 show --verbose

When the **`show-config`** command is executed from visible current location, all visible elements, such as `YCurrentThing`, and accessible elements, such as `YDeprecatedThing`, are displayed, as shown in Example 34.

```
(config-YThing=1)>show-config
YThing=1
  restrictedattr=4
  rwattr1="RW-One"
  rwattr2=2
  rwattr3=3
  YCurrentThing=1
     userLabel="UserLabelValue"
     YDeprecatedThing=1
        depAttr1=UNLOCKED
        depAttr2=16
        deprecatedStruct
           bool1=true
           int1=132
           str1="StringValue1"
           str2="StringValue2"
           up
        depStructWithKeyAttribute="StringValue1"
           bool1=true
           int1=132
           str2="StringValue2"
           up
        YSampleDeprecatedThing=1
           rwattr2=2
           up
        up
     up
  up
```

Example 34 show-config

2.11.3 Help

For accessible elements, help information is displayed when the user gives a full element name, but not when only a partial element name is given.

When the life cycle status of an element is anything else than current, the status is included as a tag in the help information, as shown in Example 35.

```
(config-YCurrentThing=1)>YDeprecatedThing?
YDeprecatedThing MO Type  [optional] [deprecated]
Test MOC
```

Example 35 Help for Accessible Element

For not-visible elements, help is not supported.

2.12 CLI Limitations

The COM CLI functionality is provided with the following limitations:

- CLI session properties (prompt, width, length, script mode) changed in a CLI session are not stored persistently and are lost after session end.
- The maximum line length supported by CLI is 2048 characters.

2.13 Automatic Correction of ManagedElement ID

The `ManagedElement` ID is typically reflecting the name of the node. That means that different nodes in a network can have different names.

To facilitate creation of generic CLI scripts and other tools to be used to several nodes, the CLI accepts any 3GPP compliant `ManagedElement` ID as part of a DN. It can automatically correct this to the actual/correct ID.

This applies for all commands and command parameters containing a DN.

3 CLI Commands

This section describes the basic CLI commands.

The following commands are used to browse information:

- `show`
- `show-config`
- `show-counters`
- `show-dn`
- `show-mib`
- `show-table`

The following navigation commands are used to navigate in the MIB without changing it:

- `dn`
- `top`
- `up`
- `back`

The following commands are used to modify information in the MIB. They require a transaction and are therefore only available in Config mode:

- `insert`
- `no`
- `reinit`

The following transaction commands are used to interact with a transaction:

- `abort`
- `commit`
- `configure`
- `validate`
- `end`

The following miscellaneous commands are used to modify and request information about the behavior of the CLI:

- `help`
- `history`
- `length`
- `prompt`
- `scriptmode`
- `version`
- `width`

The following pipe commands exist:

- `filter`

The CLI commands are described in Table 12.

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
?	Yes	Yes	Displays context-sensitive help on available CLI command elements. For details, see Section 2.7 CLI Help on page 16.
abort [-s | --stay]	No	Yes	Discards changes entered in the current transaction, closes the transaction, and creates a new one. For details, see Section 2.4.3 Abort on page 11.
back [-h | --history]	Yes	Yes	Navigates back to the previous position in the MO tree. Parameter `-h | --history` lists the previous positions in the MO tree without changing the current position. The root position in the navigation history is not stored in the navigation history because the `top` command can be used to navigate there. Only the last 10 positions are kept in this navigation history.

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
commit [-s | --stay]	No	Yes	Commits configuration changes of the current transaction and automatically starts a new transaction. For details, see Section 2.4.2 Commit on page 9.
configure	Yes	No	Changes the CLI mode to Config mode.
dn <LDN> or dn –m | --moc *<moc-name>* [--condition | -c *<condition>*]	Yes	Yes	Navigates to any existing location in the MIB. --moc or -m – Specify the MO class to navigate to. --condition or –c can be used to define criteria for the MO to navigate to. This can be used if there are more than one instances of the selected MO class. For more informations, see Section 3.2.10.1 Parameters to Filter MO Information on page 60.
end	No	Yes	Changes the CLI mode to Exec mode, if there are no changes in the current transaction. For details, see Section 2.4.4 End on page 12.
exit	Yes	No	Exits the CLI session.
help	Yes	Yes	Provides introduction help on current CLI mode and available commands in this mode. For details, see Section 3.13 Static Help in CLI on page 110.

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
history [-s|--size] [<*number*>] | [<*number*>]	Yes	Yes	Without parameter, prints the command history of the CLI session in chronological order in format <*sequence_number*> <*date*> <*time*> <*command*>. Parameter **-s|--size** specifies the size. Parameter **number** is the number of lines to be displayed. Default is 100. This command is limited to 100 latest commands. If the <*number*> specified is greater than 100, an additional text "The command history is limited to the 100 latest commands" is displayed at the end of command output.
insert [<*path*>,]<*attribute_name*>'['<*attribute_value*> | @<*index*>']'=<*value*>	No	Yes	Inserts a sequence element in a sequence attribute before the specified element or at the specified position. For details, see Section 3.5.3.3 Insert Simple Type Element in Sequence on page 79 and Section 3.5.3.4 Insert Simple Type Element in Sequence Using Index on page 79.
[<*path*>,]<*attribute_name*>'['<*attribute_value*> | @<*index*>']'=<*value*>	No	Yes	Replaces the attribute value, which is enclosed by brackets, with another value or value at specified position. For more details, see Section 3.5.3.5 Change Sequence Element on page 80 and Section 3.5.3.6 Change or Add Sequence Element Using Index on page 81.

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
length [*<Length>*]	Yes	Yes	Without parameter, displays the number of CLI output rows printed until `--More--` is printed and print is suspended. The printout is continued by pressing **Space** or **Enter** and it is discarded by pressing **Q**. Parameter **Length** is a number in the range 0–2147483647, except 1. Default is zero, which indicates no output break.
no *<path>* or **no** [*<path>*,] [*<attribute_name>*[=attribute_value> | '['@*<index>*']']](1)	No	Yes	Deletes an MO instance, an attribute, or an element from a sequence. A sequence element can be selected either by its value or by its positional index (starting from 1). If an attribute with a default value is deleted, the attribute becomes empty.
prompt [*<prompt_specifier>*]	Yes	Yes	Customizes the prompt using variables `$default`, `$dn`, `$mode`, `$nodename`, `$rdn`, and `$user` or any desired string or combination of these variables for the lifetime of the CLI session. For details, see Table 4.
<RDN>	Yes	Yes	Changes the CLI position to the RDN. If the MO does not exist, the MO is created in Config mode. See Section 3.6 Create MO on page 95.
[*<path>*,] [.,]*<action_name>* [--*<action_parameter_name>* *<action_parameter_value>*]*	Yes	Yes	Requests action execution.
[*<path>*,] <attribute_name>= *<attribute_value>*	No	Yes	Assigns value to an attribute. See Section 3.5.1 Change Single-Valued Attribute on page 75.

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
scriptmode [--*on* | --*off*]	Yes	Yes	Without parameter, prints the present state of the `scriptmode` in the ongoing CLI session. Parameter **`--on`** turns on `scriptmode`, in which help function, auto-completion, case correction, and page break is disabled in ongoing CLI session if not done. Parameter **`--off`** turns off `scriptmode`, in which help function, auto-completion, case correction, and page break is enabled back in ongoing CLI session if not done already.

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
show [--recursive | -r] [--sort | -s] [--verbose | -v] [[<path> [,<*attribute_name*>['['@<*index*>']']] | <*attribute_name*>['['@<*index*>']']][1] or **show [--sort | -s] [<*path*>] --moc | -m <*moc-name*> [--property | -p <*attribute_name*> [,<*attribute_name*>]*] [--condition | -c <*condition*>]**	Yes	Yes	Displays the system configuration and state information as MO properties with the following options: • **--recursive** or **-r** – Displays child MO instances in a recursive manner. • **--verbose** or **-v** – Displays attributes that are not set and attribute set to their default values. • ***<path>*** – <path> is either an RDN or DN. For details, see Section 2.5.2 Local Distinguished Name on page 13 and Section 2.5.3 Relative Distinguished Name on page 13. • ***<index>*** – Specifies a sequence element in a sequence attribute. The index starts from 1. • **--moc** or **-m** – Specify the MOC name whose instances must be filtered. For more details, see Section 3.2.10.1 Parameters to Filter MO Information on page 60. • **--sort** or **-s** – Displays child MO instances sorted according to their instance names. • Without options, only those attributes are displayed that have a value assigned and not a default value. For details, see Section 3.2.1 on page 50

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
show-config [--verbose | -v] [--sort | -s] [<*path*>]	Yes	Yes	Displays the output in configuration format. Configuration also automatically enables the **--recursive** or **-r** parameter. For details, see Section 3.2.2 Display Configurational Information on page 52.
show-table [-r | --recursive] [<*path*>] -m | --moc <*moc-name*> [-p | --property <*attribute_name*> [:<*column width*>][,<*attribute_name*> [:<*column width*>]]*] [-c | --condition <*condition*>][-s | --sort]	Yes	Yes	Displays information in table format. For details, see Section 3.2.10.3 Display MO Information in Tabular Format on page 64. `--recursive` or `-r` implies that the displayed instances can be anywhere below the current position. `--moc` or `-m` – Specify the MOC name whose instances to be displayed. For more information, see Section 3.2.10.1 Parameters to Filter MO Information on page 60.
show-dn	Yes	Yes	Displays the current location of the user in the MOM.
show-mib [--verbose | -v] [--sort | -s] [<*path*>]	Yes	Yes	Displays MO instance information. For details, see Section 3.2.9 Display MO Instance Information on page 59.

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
show-counters [dn] [-j | --pmJob jobId] [-v | --verbose] [-c | --counters counter[,counter]*]	Yes	Yes	**show-counters** is a command that is not supported on all MEs. It displays real-time values of PM for 1 MO instance: • dn selects the MO instance to show counters from. If omitted, counters from the current MO are displayed. • **--pmJobId** or **-j** – displays only counters associated to one active PM job. The parameter is the ID value of 1 MO instance of the MO class PmJob. • **--verbose** or **-v** – displays verbose information. • **--counters** or **-c** – displays only the selected counters. **show-counters** is an optional command. It is available only if the ME supports displaying measurements through the CLI. For more information, see Section 3.2.11 Display PM Measurements on page 67.
top	Yes	Yes	Changes the CLI position to the root position.
up	Yes	Yes	Changes the CLI position to the parent MO.
validate	No	Yes	Validates the configuration changes in a transaction. Returns validation error or message Transaction is valid!.
version	Yes	Yes	Displays the CLI version. For more information, see Section 3.16 Display CLI Version on page 111.

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
width [<*Width*>]	Yes	Yes	Without parameter, displays the number of CLI output characters printed on a line until the line is broken. Parameter `Width` is a number in the range 0–2147483647. Default is zero, which indicates no line break for non-tabular output, and that tabular output uses the actual terminal window size. The `Width` command is primarily intended for informing the CLI about the actual terminal width when the CLI cannot determine it by itself. Setting the width to a value other than the actual terminal width is not recommended.

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
<*command/Action*> | **filter [-i | --ignore] [-v | --invert] [{-A | --after} <*value*>] [{-B | --before} <*value*>] <*pattern*>**	Yes	Yes	Display the lines that match the specified pattern with the following options. For more information, see Section 3.12.1 on page 108. • **-i | --ignore** : Ignore case distinctions in both the pattern and the input. • **-v | --invert** : Invert the sense of matching, to select non-matching lines. • **-A | --after** : Print number of lines of trailing context after matching lines. • **-B | --before** : Print number of lines of leading context before matching lines. • **<value>** : The value for number lines to be printed after leading / trailing context. • **<pattern>** : An input pattern is a regular expression used for filtering the output of a command/action. It can have alphanumeric characters and special characters supported by POSIX® regular expressions.

Table 12 Summary on CLI Commands

Command	Exec Mode	Config Mode	Description
passwd	Yes(2)	Yes(2)	Changes the CLI user password. The `passwd` is an optional command. It is available only if the ME supports changing user password through the CLI. For more information, see Section 3.17 Change Password Command on page 111.
reinit [. | *<attribute_name>*]	No	Yes	Reset an attribute or an MO to its original state. For details, see Section 3.7 Reinitialize MO on page 98.

(1) The syntax '['@<index>']' means here that @<index> is mandatory in the command.
(2) The `passwd` command is only available in root position.

3.1 Success and Error Indications

The conditions for successful command completion are checked in multiple steps, as follows:

- The user must have the proper authorization.
- The command must comply to CLI syntax rules.
- The command must comply to the constraints defined in the information model.
- The command must comply to semantic rules, that is, the system state must be valid after the command completion.
- The system must be in a state to be able to process the command.

If the CLI command completes with success, no printout is provided. If the operation fails, an error message is displayed in the following format:

```
ERROR: <generic_error_message>[<specific_error_message>]
```

Generic error message

Error text using the same message format or template for error indication for all the MOCs and attributes of the same properties in the same error situation.

Examples of generic error messages are shown in Table 13.

Specific error messages

Context-specific details provided by the model implementation.

As an exception, the `validate` operation returns a printout on success, and actions can return printout if the action return value is not void.

Table 13 Examples of Generic Error Messages

Error Message	Error Semantics
`ERROR: Command not found`	Indicates that the syntax of the command is not valid.
`ERROR: Can not instantiate system created object`	The CLI user attempts to create a MOC that is defined as `system created`.
`ERROR: Can not delete system created object`	The CLI user attempts to delete a MOC that is defined as `system created`.
`ERROR: Attribute '<attribute_name>' is read-only`	The CLI user attempts to modify an MO attribute that is defined as `read-only`.
`ERROR: Attribute '<attribute_name>' is read-only (can't be deleted)`	The CLI user attempts to delete an MO attribute that is defined as `read-only`.
`ERROR: Parent '<parent_DN>' does not exist`	The CLI user attempts to create an MO whose parent does not exist.
`ERROR: Attribute '<attribute_name>' is restricted`	The CLI user attempts to modify an MO attribute that is defined as `restricted`.

`ERROR: Call command failed, error code: <error_reason>`	Depending on the command and the MO instance, as described in the product-specific documentation, the error reason can be one of the following:
	
	`ComAborted`
	`ComAlreadyExist`
	`ComCommitFailed`
	`ComFailure`
	`ComInvalidArgument`
	`ComNoResources`
	`ComNotActive`
	`ComNotExist`
	`ComObjectLocked`
	`ComPrepareFailed`
	`ComTimeOut`
	`ComTryAgain`
	`ComValidationFailed`
	`Unknown return code`

Table 13 Examples of Generic Error Messages

Error Message	Error Semantics
ERROR: Invalid value *<issued command with arguments>* for integer parameter *<issued command with arguments>*. Values are in range [min, max]	Indicates that the value of the parameter is invalid and that allowed values are in specified range [min, max].
ERROR: Element not visible	Indicates that the user cannot access not-visible elements.
ERROR: Instances of '*<moc_name>*' are not creatable	The CLI user attempts to create an MO instance that is not possible to create through the NBI.
ERROR: Instances of '*<moc_name>*' are not deletable	The CLI user attempts to delete an MO instance that is not possible to delete through the NBI.
ERROR: Invalid index '0'. The lowest index in a sequence is '1'	The CLI user attempts to select a sequence element in a sequence attribute using the invalid index 0. Index values start from 1.
ERROR: Multiple MO classes with same name in different paths: *<MO class path>* *<MO class path>* A class path is required	This error is shown when there is more than one MO class with the same name when using the -moc or -m parameter. The MO class path is a comma separated list of the class names, for example ManagedElement,Moc1,Test. This MO class path can be used as argument to the -moc and the -m parameter.

3.1.1 Error Codes

Table 14 shows the meaning of the possible error codes that can be seen in CLI error messages.

Table 14 Error Codes

Error Code	Description
ComTryAgain	The function cannot provide any service currently. The problem that occurred is temporary, and the caller can retry later.
ComNotActive	The function cannot provide any service since the service is not started.
ComFailure	The function call failed, an error has occurred that is specific for the function implementation.
ComNotExist	The function call failed since something sought after did not exist.

Table 14 Error Codes

Error Code	Description
ComAlreadyExist	The function call failed since something that was to be created exists.
ComAborted	The function call failed and was ended. The function did not change any persistent data.
ComObjectLocked	The function call failed since an object is locked.
ComPrepareFailed	The function call failed in the prepare phase of the transaction. The transaction has been ended.
ComCommitFailed	The function call failed in the commit phase. Some participants failed to commit and the total transactional result can be inconsistent. A human can be needed to resolve the situation.
ComInvalidArgument	The function call failed since an argument is invalid.
ComValidationFailed	The function call failed since the data did not validate. The only time this error code is used is as return code from the commit operation.
ComNoResources	The function call failed since there was no available resource, such as memory.

3.2 Display Information

This section describes the displaying of the following informations:

- Single-valued attributes, see section Section 3.2.1 Display Single-Valued Attribute on page 50.
- `show-config`, see section Section 3.2.2 Display Configurational Information on page 52.
- Struct, see section Section 3.2.3 Display Struct on page 54.
- Struct member, see section Section 3.2.4 Display Structure Member on page 55.
- Sequence of single value, see section Section 3.2.5 Display Sequence of Simple Type on page 56.
- Sequence of struct, see section Section 3.2.7 Display Sequence of Struct on page 57.

- Sequence element of sequence of struct, see section Section 3.2.8 Display Sequence Element of Sequence of Struct on page 58.
- `show-mib`, see section Section 3.2.9 Display MO Instance Information on page 59.
- `show-table`, see section Section 3.2.10.3 Display MO Information in Tabular Format on page 64.
- `show-counters`, see section Section 3.2.11 Display PM Measurements on page 67.

3.2.1 Display Single-Valued Attribute

One single-valued attribute can be displayed by `show [--recursive|-r] [--verbose|-v] [<path>,]<attribute_name>`.

A summary on the printout syntax and the displayed information with various attribute types are provided in Table 15.

Table 15 Show Single-Valued Attribute Matrix

Model Property(1)	**show [*<path>*,]*<attribute_name>***	**show --verbose [*<path>*,]*<attribute_name>* or show -v [*<path>*,]*<attribute_name>***(2)
Read-write attribute with value that is not `defaultValue`	*<attribute_name>*=*<attribute_value>*	*<attribute_name>*=*<attribute_value>*(3)
Read-only or restricted attribute with value that is not `defaultValue`	*<attribute_name>*=*<attribute_value>*	*<attribute_name>*=*<attribute_value>*(4)
Attribute with value equal to `defaultValue`	*<attribute_name>*=*<attribute_value>*	*<attribute_name>*=*<attribute_value>* <default>(5)
Attributes without value(6)	*<attribute_name>*=[]	*<attribute_name>*=[] *<empty>*(3)

(1) The description is valid both for attributes and struct members.
(2) The <key> printout is present if the struct member is key.
(3) The <passphrase> printout is present if the attribute is a passphrase string.
(4) The <read-only> printout is present if the attribute is read-only.
(5) The <read-only> printout is also present if the attribute is read-only.
(6) That is, optional or nillable attributes that have no value assigned.

The display formats of supported attribute value data types are shown in Table 16.

Table 16 Display Formats of Supported Attribute Value Data Types

Attribute Value Data Type	Description
`bool`	Displayed as `false` or `true`

Attribute Value Data Type	Description
enum	The name of the enumeration member is displayed. Example: administrativeState=LOCKED
int8, int16, int32, int64, uint8, uint16, uint32, uint64	Displayed as integer numbers.
string	Displayed in double quotation marks. Example: "ABC"
moRef	MO reference is displayed as string that contains the LDN of the referred MO. Example: "ManagedElement=*<node_name>*,SystemFunctions=1,MyMo=42"
password	Displayed as string in encrypted form. Password change is supported in a special way, see Section 3.5.5 Change Attribute Defined as Password or Passphrase String on page 91.
float	Displayed as decimal numbers. Scientific notation is used when the lexical representation of the value is too long
passphrase string	Displayed as a masked value "********", or in encrypted form, depending on how the ME is configured. For more information, see Section 3.5.5 Change Attribute Defined as Password or Passphrase String on page 91.

The command and printout syntax is shown in Example 36, Example 37, Example 38, and Example 39.

```
>show ManagedElement=<node_name>,userLabel
userLabel="BTS#21 in Zone C"
```

Example 36 Display Single-Valued Attribute

```
>show ManagedElement=<node_name>,myEmptyAttribute
myEmptyAttribute
```

Example 37 Display Single-Valued Attribute for EcimEmpty

```
>show -r ManagementElement=<node_name>,TestRootMoc=1,C
razyThing=1,anAttrwWithIntDerivedType
anAttrWithIntDerivedType=16
```

Example 38 Display Single-Valued Attribute by Show Recursive

```
(TestRootMoc=1)>show ManagedElement=<node_name
>,TestRootMoc=1,userLabel
userLabel="UserLabelValue"
```

Example 39 Displaying Single-Valued Attribute by Using LDN

3.2.2 Display Configurational Information

Command **show-config** displays the output in configuration format, see Example 40. **show-config** also automatically enables the `--recursive` or `-r` parameter.

Command **show-config** with verbose option displays the attributes that are not set and the attributes set to their default values, see Example 41.

The difference between **show-config** and **show-config** with **-v** or **--verbose** option when copying the output into a CLI, is the following:

- With **--verbose** or **-v** option parameter the command overwrites, hence deletes non-set (in the input) optional values and sets non-set default values back to default.

- Without **--verbose** or **-v** option parameter the command appends, hence only includes values that are set.

Its recommended to use **show-config** with **-v** or **--verbose** option when copying the output into a CLI.

With **-s** or **--sort** option parameter, MO instances are sorted according to their instance names.

Syntax: **show-config [--sort | -s] [--verbose | -v] [*<path>*]**

```
(config-CrazyThing=1)>show-config
CrazyThing=1
administrativeState=UNLOCKED
aManagedObject="ManagedElement=<node_name>,TestRootMoc=1,
BasicThing=1"
anAttrWithIntDerivedType=16
anAttrWithStringDerivedType="xx6BBBBBBBBBBxx"
aSimpleStruct
  bool1=true
  int1=132
  str1="StringValue1"
  str2="StringValue2"
  up
 aStructWithDefValues
  up
 aStructWithKeyAttribute="StringValue1"
  bool1=true
  int1=132
  str2="StringValue2"
  up
up
```

Example 40 Show-Config

```
(config-CrazyThing=1)>show-config -v
CrazyThing=1
administrativeState=UNLOCKED
aManagedObject="ManagedElement=<node_name>,TestRootMoc=1,BasicThing=1"
anAttrWithIntDerivedType=16
anAttrWithStringDerivedType="xx6BBBBBBBBBBxx"
   no anAttrWithStringDerivedType_noDefault
   no anEcimEmpty
   no anyManagedObject
   no DateTimeTestValue
   no DateTimeWithoutOffsetTestValue
   defaultValue=0
   no DifferenceFromUTCTestValue
   no ipDNSAddressTestValue
   no noDontAutocompleteOnNo
   no ProblemCauseTestValue
   no rebelObject
   no RuleDataTypeTestValue
   no TimeoutTestValue
   no UnsignedRangeDataTestValue
   no anExclusiveStruct
   no anOptionalStruct
   no aPassphraseStruct
aSimpleStruct
   bool1=true
   int1=132
   int2=42
   str1="StringValue1"
   str2="StringValue2"
   up
aStructWithDefValues
   bool1=false
   int1=1
   str1="HoggaBogga"
   up
   no aStructWithHiddenAttribute
aStructWithKeyAttribute="StringValue1"
   bool1=true
   no int0
   int1=132
   int2=42
   str1="StringValue1"
   str2="StringValue2"
   up
no aStructWithMoRefs
no ecimStructArray
no MultiValueStructWithKeyAttr
up
```

Example 41 Display Configuration Output with Verbose Information

Command **show-config** for sequence of keyless structs displays the output as shown in Example 42.

```
(config-Snmp=1)>show-config
Snmp=1
agentAddress
   host="0.0.0.0"
   port=26343
   up
agentAddress
   host="1.1.1.1"
   port=9999
   up
```

Example 42 Show-config for Sequence of Keyless Structs

Command **show-config** with verbose option displays configuration with explicit position for sequence of keyless structs, see Example 43.

```
(config-Snmp=1)>show-config -v
Snmp=1
  administrativeState=UNLOCKED
  no nodeCredential
  no trustCategory
  agentAddress[@1]
     host="0.0.0.0"
     port=26343
     up
  agentAddress[@2]
     host="1.1.1.1"
     port=9999
     up
```

Example 43 Show-config with Verbose Option

3.2.3 Display Struct

Attributes defined as structure can be displayed by **show [--verbose|-v] [<*path*>,]<*attribute_name*>**.

The printout has the following format:

```
<attribute_name>
 <structure_member_name>=<value>
   <structure_member_name>=<value>
   <structure_member_name>=[]
   ...
   <structure_member_name>=<value>
```

Note: The nillable attributes are displayed as the ordinary attributes. See Table 15.

The command and printout syntax is shown in Example 44, Example 45, and Example 46.

```
(config-CrazyThing=1)>show aSimpleStruct
aSimpleStruct
   bool1=true
   int1=132
   str1="StringValue1"
   str2="StringValue2"
```

Example 44 Display Struct

```
(config-CrazyThing=1)>show --verbose aSimpleStruct
aSimpleStruct
   bool1=true
   int1=132
   int2=42 <default>
   str1="StringValue1"
   str2="StringValue2"
```

Example 45 Display Struct with CLI Operation Parameter

```
(config-CrazyThing=1)>show -r aSimpleStruct
aSimpleStruct
   bool1=true
   int1=132
   str1="StringValue1"
   str2="StringValue2"
```

Example 46 Display Struct with CLI Operation Parameter

3.2.4 Display Structure Member

This section provides information about displaying structure member.

3.2.4.1 Display Single-Valued Structure Member

Single-valued structure members can be displayed by command **show [--verbose|-v] [*<path>*,]*<attribute_name>*,*<structure_member_name>***.

The printout is according to the data type of the structure member.

The command and printout syntax is shown in Example 47 and Example 48.

```
(config-CrazyThing=1)>show aSimpleStruct,str1
str1="string"
```

Example 47 Display Structure Member

```
(config-CrazyThing=1)>show --verbose aSimpleStruct,int2
int2=42 <default>
```

Example 48 Display Structure Member with CLI Operation Parameter

3.2.4.2 Display Sequence Structure Member

Structure members defined as sequence (also know as multi-valued struct members) can be displayed by command **show [--verbose|-v] [*<path>*,]*<attribute_name>*,*<structure_member_name>***.

The command and printout syntax is shown in Example 49 and Example 50.

```
(config-structWithMultivalueMembers)>show intMultivalueMember
intMultivalueMember
  42
  43
  44
```

Example 49 Display Sequence Structure Member

```
(config-structWithMultivalueMembers)>show --verbose intMultivalueMember
intMultivalueMember <default>
  42
  43
  44
```

Example 50 Display Sequence Structure Member with CLI Operation Parameter

3.2.4.3 Display Sequence Element of Sequence Structure Member

A sequence element of sequence structure member can be displayed by command **show [--verbose|-v] [*<path>*,]*<attribute_name>*,*<structure_member_name>*'['@*<index>*']'**.

The command and printout syntax is shown in Example 51.

```
(config-structWithMultivalueMembers)>show intMultivalueMember[@2]
intMultivalueMember
  43
```

Example 51 Display Sequence Structure Member Using Positional Index

3.2.5 Display Sequence of Simple Type

Attributes defined as sequence (also know as multi-valued attributes) can be displayed by command **show [--recursive|-r][--verbose|-v][*<path>*,]*<sequence_attribute_name>***.

The printout has the following format if the sequence contains elements of the same type:

```
<sequence_attribute_name>
   <sequence_element_value>
   <sequence_element_value>
   ...
   <sequence_element_value>
```

The command and printout syntax is shown in Example 52, Example 53, and Example 54.

```
(config-MultivalueThing=1)>show PlainMultivalueAttrs=1,
stringMultivalueAttr
stringMultivalueAttr
   "STRING"
   "SWEDEN"
```

Example 52 Sequence of Strings

```
(config-MultivalueThing=1)>show -r PlainMultivalueAttrs=1,
stringMultivalueAttr
stringMultivalueAttr
     "STRING"
     "SWEDEN"
```

Example 53 Sequence of Strings with CLI Operation Parameter

```
(config-PlainMultivalueAttrs=1)>show --verbose
intMultivalueAttr
intMultivalueAttr <default>
   42
   43
   44
(config-PlainMultivalueAttrs=1)>
```

Example 54 Sequence of Integers

3.2.6 Display Sequence Element of Sequence of Simple Type

It is possible to display a selected sequence element from a sequence of simple type using positional index by command **show [--verbose | -v] [*<path>*,]*<sequence_attribute_name>*'['@*<index>*']'**.

The positional index starts from 1.

The command and printout syntax is shown in Example 55.

```
(config-PlainMultivalueAttr=1)>show intMultivalueAttr
intMultivalueAttr
   42
   43
   44
(config-PlainMultivalueAttr=1)>show intMultivalueAttr[@1]
intMultivalueAttr
   42
```

Example 55 Display Sequence Element of Sequence of Simple Type

3.2.7 Display Sequence of Struct

Attributes defined as sequence of struct can be displayed by **show [--recursive|-r] [--verbose|-v] [*<path>*,]*<attribute_name>***.

The printout has the following format:

```
<attribute_name>
   <structure_member_name>=<value>
   <structure_member_name>=<value>
   <structure_member_name>=[]
   ...
   <structure_member_name>=<value>
<attribute_name>
   <structure_member_name>=<value>
   <structure_member_name>=<value>
   <structure_member_name>=[]
...
```

The command and printout syntax is shown in Example 56 and Example 57.

```
>show ManagedElement=<node_name>,SystemFunctions=1,
SysM=1,Snmp=1,agentAddress
agentAddress
   host="1.1.1.1"
   port=111
agentAddress
   host="2.2.2.2"
   port=222
```

Example 56 Sequence of Structs without Key

```
>show -v ManagedElement=<node_name>,SystemFunctions=1,
SysM=1,Snmp=1,agentAddress
agentAddress[@1]
   host="1.1.1.1"
   port=111
agentAddress[@2]
   host="2.2.2.2"
   port=222
```

Example 57 Sequence of Structs without Key with CLI Operation Parameter

3.2.8 Display Sequence Element of Sequence of Struct

It is possible to display selected sequence elements if they contain struct with key member by command **show [--recursive|-r] [--verbose|-v] [*<path>*,]*<attribute_name>*,*<key_struct_member_name>*=*<key_struct_member_value>***.

The printout has the following format:

```
<key_struct_member_name>=<key_struct_member_value>
   <struct_member_name>=<struct_member_value>
   <struct_member_name>=<struct_member_value>
   ...
   <struct_member_name>=<struct_member_value>
```

The command and printout syntax is shown in Example 58, Example 59, and Example 60.

In Example 58, attribute `multiValueStructWithIntId` is defined as a sequence of structure and the structure has the following members:

- ‘‘id’’ – Defined as `int64`, that is, the key member of the struct.
- ‘‘name’’ – Defined as string with default value ‘‘countryName’’.
- ‘‘capital’’ – Defined as string.
- ‘‘population’’ – Defined as `int64` with default value 0.

```
(config-PlainMultivalueAttrs=1)show --verbose
multiValueStructWithIntId
multiValueStructWithIntId=2
   capital="kol"
   id=2 <key>
   name="countryName" <default>
   population=0 <default>
multiValueStructWithIntId=3
   capital="xyz"
   id=3 <key>
   name="countryName" <default>
   population=0 <default>
multiValueStructWithIntId=1
   capital="delhi"
   id=1 <key>
   name="countryName" <default>
   population=0 <default>
(config-MultivalueThing=1)>show --verbose PlainMultivalueAttrs=1,
multiValueStructWithIntId=1
multiValueStructWithIntId=1
   capital="delhi"
   id=1 <key>
   name="countryName" <default>
   population=0 <default>
```

Example 58 Sequence Element of Sequence of Structs with Key

```
>show ManagedElement=<node_name>,SystemFunctions=1,SysM=1,Snmp=1,
agentAddress,host
ERROR: Multi-value struct attribute 'agentAddress' is missing a
key value or an index.
```

Example 59 Sequence Element of Sequence of Structs without Key

```
(config-PlainMultivalueAttrs=1)>show -r multiValueStructWithIntId
multiValueStructWithIntId=1
   capital="Sweden"
   population=20000000
multiValueStructWithIntId=2
   capital="Norway"
```

Example 60 Sequence Element of Sequence of Structs with Key with CLI Operation Parameter

3.2.9 Display MO Instance Information

Command **show-mib** displays the list of instance information without their contents. It displays MO instance name recursively.

Command **show-mib** with verbose parameter displays complete path of the DN.

With **-s** or **--sort** option parameter, MO instances are sorted according to their instance names.

Syntax: **show-mib [--sort|-s] [--verbose|-v] [*<path>*]**

The command is shown in Example 61 and Example 62.

```
 (config-TestRootMoc=1)>show-mib
TestRootMoc=1
   BasicThing=1
   CallableThing=1
   CrazyThing=1
   HiddenAttrThing=1
   XThing=1
      XXThing=1
   YThing=1
```

Example 61 Display MO Instance Recursively

```
 (config-ManagedElement=<node_name>)>show-mib -v TestRootMoc=1
ManagedElement=<node_name>,TestRootMoc=1
 ManagedElement=<node_name>,TestRootMoc=1,BasicThing=1
 ManagedElement=<node_name>,TestRootMoc=1,CallableThing=1
 ManagedElement=<node_name>,TestRootMoc=1,CrazyThing=1
 ManagedElement=<node_name>,TestRootMoc=1,HiddenAttThing=1
 ManagedElement=<node_name>,TestRootMoc=1,XThing=1
 ManagedElement=<node_name>,TestRootMoc=1,XThing=1,XXThing=1
 ManagedElement=<node_name>,TestRootMoc=1,YThing=1
```

Example 62 Show-Mib with CLI Operation Parameter

3.2.10 Filter MO Information

The MO information can be reduced with the filter options added to commands **show** and **show-table**. The filtered information can be displayed in tree structure and in tabular format.

3.2.10.1 Parameters to Filter MO Information

Mandatory parameter `-m` or `--moc` option signifies the name of a child class (MOC) under the current context where the user is present. All MO instances of this MOC are displayed.

If there is more than one MO class with the same name, the name must be qualified by prepending the class name with the parent class name plus a comma. As an example, consider if there is a class named “Xyz” under `SystemFunctions` and another class with the same name under `ManagedElement`. Then these classes can be addressed by `SystemFunctions,Xyz` resp. `ManagedElement,Xyz`.

Optional parameters are the following:

- `-p` or `--property` signifies the attributes that the user wants to print specifically under the MOC, specified in the `-m` or `--moc` option. The attribute list can be specified by `[-p<attribute_name>]` where attributes must be separated by a comma. There can be zero to many attributes in this list, where zero attributes means that no attribute is printed.

- For command `show-table`, which shows MO information in tabular format, the column width can optionally be specified for attributes. This is done by using syntax `[-p <attribute_name:column_width>`.

- MO Information can be filtered and displayed based on a condition. This can be achieved by using the `-c` or `--condition` option. There can be a maximum of two conditions that can be specified for this option. The type of attributes that can be used as part of the condition are: integer, string, boolean and enum.

  If the attribute used in the condition is of the type string, then the value specified for that attribute must be quoted, otherwise an error message saying `Condition String not quoted` is shown.

  The syntax for condition is `[Condition]=[<attribute><operator><value>]`

  The valid operators are:

  – "==" or "=" for left hand side is equal to the right hand side.

  – ">=" for left hand side is greater than or equal to the right hand side.

  – "<=" for left hand side is less than or equal to the right hand side.

  – ">" for left hand side is greater than the right hand side.

  – "<" for left hand side is smaller than the right hand side.

  – "<>" for left hand side is greater or less than (not equal) the right hand side.

  In case of multiple conditions to be specified, the conditions must be separated by logical "OR (||)" or the logical "AND (&&)" operator. An expression can be preceded by a NOT predicate to negate the expression. Parentheses can be used for grouping of expressions.

  The multiple condition syntax is as follows:

```
[Condition]=((condition)&&(condition))
           =((condition)||(condition))
           =NOT(condition)
```

3.2.10.2 Search for Information in the MO Tree Structure

The `show` command can be used with filtering parameters: `-m` or `--moc`, `-p` or `--property`, and `-c` or `--condition`.

It is also described in Section 3 on page 35.

The `-r` or `--recursive` parameter can be used to search recursively in the whole MO tree. When the parameter is used, the DN for each matching MO instance is displayed. Omitting the parameter implies searching of one level and the RDN for each matching MO instance is displayed.

With `-s` or `--sort` option parameter, MO instances are sorted according to their instance names.

The syntax is as follows:

```
show [-s | --sort ][<path>] --moc | -m <moc-name> [-p | --property
[<attribute_name> [,<attribute_name>]*]+] [-r | --recursive]
[-v | --verbose] [--condition | -c <condition>]
```

Example 63 shows how to display specific MO type in tree structure.

```
>show SysM=1 -m Schema -r
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ECIM_CommonLibrary
   baseModelIdentifier="ECIM_CommonLibrary"
   baseModelVersion="1.2"
   identifier="ECIM_CommonLibrary"
   version="1.2"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComTop
   baseModelIdentifier="ECIM_Top"
   baseModelVersion="2.1.0"
   identifier="ComTop"
   version="10.10.0"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComSecM
   baseModelIdentifier="ECIM_Security_Management"
   baseModelVersion="2.0"
   identifier="ComSecM"
   version="11.0.1"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComLocalAuthorization
   baseModelIdentifier="ECIM_Local_Authorization"
   baseModelVersion="2.0.0"
   identifier="ComLocalAuthorization"
   version="0.11.1"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComLdapAuthentication
   baseModelIdentifier="ECIM_LDAP_Authentication"
   baseModelVersion="2.0"
   identifier="ComLdapAuthentication"
   version="11.0.0"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComSysM
   baseModelIdentifier="ECIM_SysM"
   baseModelVersion="3.1.0"
   identifier="ComSysM"
   version="3.1.0"
```

Example 63 Display Specific MO Type in Tree Structure

Display Specific Attributes, or No Attributes, for Found MO Instances

Command **show** with the `--property` or `-p` parameter displays specific attributes under all the MO instances of the specified MOC. See Example 64 and Example 65.

```
>show SysM=1 -m Schema -r -p
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ECIM_CommonLibrary
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComTop
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComSecM
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComLocalAuthorization
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComLdapAuthentication
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComSysM
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComFm
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComSnmp
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComFileM
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=CmwPm
```

Example 64 Display only Found Instances, no Attributes

```
(config-SystemFunctions=1)>show -r SysM=1 -m Schema -p identifier,baseModelVersion
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ECIM_CommonLibrary
   identifier="ECIM_CommonLibrary"
   baseModelVersion="1.2"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComTop
   identifier="ComTop"
   baseModelVersion="2.1.0"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComSecM
   identifier="ComSecM"
   baseModelVersion="2.0"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComLocalAuthorization
   identifier="ComLocalAuthorization"
   baseModelVersion="2.0.0"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComLdapAuthentication
   identifier="ComLdapAuthentication"
   baseModelVersion="2.0"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComSysM
   identifier="ComSysM"
   baseModelVersion="3.1.0"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComFm
   identifier="ComFm"
   baseModelVersion="4.0.0"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComSnmp
   identifier="ComSnmp"
   baseModelVersion="1.2"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComFileM
   identifier="ComFileM"
   baseModelVersion="3.1.0"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=CmwPm
   identifier="CmwPm"
   baseModelVersion="1.2"
```

Example 65 Display Specific Attributes under All Existing MO Instances for a MOC

Display Attributes Based on a Condition

Command **show** with `--condition` or `-c` filters the existing MO instances and displays information for MO instances with attributes fulfilling the specified condition. See Example 66, Example 67, and Example 68.

```
>show --recursive --moc Schema --condition version==11.0.0
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComLdapAuthentication
   baseModelIdentifier="ECIM_LDAP_Authentication"
   baseModelVersion="2.0"
   identifier="ComLdapAuthentication"
   version="11.0.0"
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=ComFileM
   baseModelIdentifier="ECIM_FileM"
   baseModelVersion="3.1.0"
   identifier="ComFileM"
   version="11.0.0"
```

Example 66 Specify Condition to Filter MO Information

```
(config-SystemFunctions=1)>show SysM=1 --moc Schema --property identifier,
baseModelIdentifier --condition version==11.0.0
Schema=ComLdapAuthentication
   identifier="ComLdapAuthentication"
   baseModelIdentifier="ECIM_LDAP_Authentication"
Schema=ComFileM
   identifier="ComFileM"
   baseModelIdentifier="ECIM_FileM"
```

Example 67 Specify Attributes and Condition for Filtering MO Information

```
(config-SysM=1)>show -m Schema -p identifier,baseModelIdentifier -c
version==3.1.0 && baseModelVersion==3.1.0
Schema=ComSysM
   identifier="ComSysM"
   baseModelIdentifier="ECIM_SysM"
```

Example 68 Specify Attributes and Multiple Conditions for Filtering MO Information

3.2.10.3 Display MO Information in Tabular Format

Command **show-table** displays the MO information based on MOC in a tabular format. It displays the show verbose printout without metadata, for example, no tags are displayed in the table.

The user can choose information based on attribute selection and can also specify the column width.

Command **show-table** do not display struct attributes and multivalued attributes.

If the MOC contains hidden attributes, command **show-table** do not display them. To display hidden attributes specifically, include the hidden attributes along with the `-p` or `--property` option.

With `-s` or `--sort` option parameter, MO instances are sorted according to their instance names.

Syntax: **show-table [-r | --recursive] [*<path>*] -m | --moc *<moc-name>* [-p | --property *<attribute_name>* [:*<column width>*] [,*<attribute_name>* [:*<column_width>*]]*] [-c | --condition *<condition>*] [-s | --sort]**

Example 69 shows how to display MO information in table.

```
(config-SystemFunctions=1)>show-table SysM=1 -m Schema
=============================================================================================================
| schemaId                | identifier              | baseModelIdentifier          | version | baseModel..
=============================================================================================================
| ECIM_CommonLibrary      | ECIM_CommonLibrary      | ECIM_CommonLibrary           | 1.2     | 1.2        |
| ComTop                  | ComTop                  | ECIM_Top                     | 10.10.0 | 2.1.0      |
| ComSecM                 | ComSecM                 | ECIM_Security_Management     | 11.0.1  | 2.0        |
| ComLocalAuthorization   | ComLocalAuthorization   | ECIM_Local_Authorization     | 0.11.1  | 2.0.0      |
| ComLdapAuthentication   | ComLdapAuthentication   | ECIM_LDAP_Authentication     | 11.0.0  | 2.0        |
| ComSysM                 | ComSysM                 | ECIM_SysM                    | 3.1.0   | 3.1.0      |
| ComFm                   | ComFm                   | ECIM_FM                      | 12.0.0  | 4.0.0      |
| ComSnmp                 | ComSnmp                 | ECIM_SNMP                    | 1.2     | 1.2        |
| ComFileM                | ComFileM                | ECIM_FileM                   | 11.0.0  | 3.1.0      |
| CmwPm                   | CmwPm                   | ECIM_PM                      | 1.0     | 1.2        |
=============================================================================================================
```

Example 69 Display MO Information in Table

When using the `-r` or `--recursive` parameter, a recursive search is done for the MO class. The parent MO LDN is printed above the table. If there are instances under different parents, a new table is printed for each parent. See Example 70.

```
>show-table -r -m Schema
ManagedElement=1,SystemFunctions=1,SysM=1
=============================================================================================================
| schemaId                | identifier              | baseModelIdentifier          | version | baseModel..
=============================================================================================================
| ECIM_CommonLibrary      | ECIM_CommonLibrary      | ECIM_CommonLibrary           | 1.2     | 1.2        |
| ComTop                  | ComTop                  | ECIM_Top                     | 10.10.0 | 2.1.0      |
| ComSecM                 | ComSecM                 | ECIM_Security_Management     | 11.0.1  | 2.0        |
| ComLocalAuthorization   | ComLocalAuthorization   | ECIM_Local_Authorization     | 0.11.1  | 2.0.0      |
| ComLdapAuthentication   | ComLdapAuthentication   | ECIM_LDAP_Authentication     | 11.0.0  | 2.0        |
| ComSysM                 | ComSysM                 | ECIM_SysM                    | 3.1.0   | 3.1.0      |
| ComFm                   | ComFm                   | ECIM_FM                      | 12.0.0  | 4.0.0      |
| ComSnmp                 | ComSnmp                 | ECIM_SNMP                    | 1.2     | 1.2        |
| ComFileM                | ComFileM                | ECIM_FileM                   | 11.0.0  | 3.1.0      |
| CmwPm                   | CmwPm                   | ECIM_PM                      | 1.0     | 1.2        |
=============================================================================================================
```

Example 70 Display MO Information in Table, recursive

Display Attributes with Show-Table

Command **`show-table`** with `--property` or `-p` displays the specific attributes under the MOC.

Width of the column can be specified by the user, see Example 71 and Example 72.

```
(config-ManagedElement=<node_name>)>show-table -m TestRootMoc
-p userLabel:20
==================
|userLabel       |
==================
|UserLabelValue  |
==================
```

Example 71 Display Single Attribute with Show-Table

```
(config-ManagedElement=<node_name>)>show-table -m TestRootMoc
-p testRootMocId, userLabel
==================================
|testRootMocId  | userLabel      |
==================================
|1              | UserLabelValue |
==================================
```

Example 72 Display Multiple Attributes with Show-Table

If no value is available for attribute it displays as empty, as shown in Example 73.

```
(config-SystemFunctions=1)>show-table -m FileM
=======================
|fileMId | userLabel |
=======================
|1       |           |
=======================
```

Example 73 Display Empty Attribute with Show-Table

Display Attributes in Tabular Format Based on Condition.

Example 74 shows how to display attributes based on condition.

```
(config-TestRootMoc=1)>show-table -m CrazyThing -p administrativeState,
defaultValue -c (defaultValue==0)
=====================================
| administrativeState | defaultValue |
=====================================
| UNLOCKED            | 0            |
=====================================
```

Example 74 Display Attributes Based on Condition with Show-Table

Show-Table Width

Command **`show-table`** automatically sets its width based on the terminal width and the column width specified by the user. If the column width is not specified by the user, the column width is set based on the largest item in the column.

3.2.10.4 Auto-Completion of Attributes under Filter Options

Command **`show`** and **`show-table`** with `-p` or `--property` option auto-completes the list of attributes under MOC, specified with the `-m` or `--moc` option. The hidden attributes cannot be displayed in the auto-completion list. See Example 75 and Example 76.

```
(config-SysM=1)>show -m Snmp -p <TAB>
administrativeState
agentAddress
operationalState
snmpId
(config-SysM=1)>show -m Snmp -p agentAddress, <TAB>
administrativeState
operationalState
snmpId
```

Example 75 Completion Possibility List for Attributes for Show Command

For command **`show-table`**, the multi-value and struct attributes is not be listed in the auto-completion list under the `--property` or `-p` parameter.

```
(config-SysM=1)>show-table -m Snmp -p <TAB>
administrativeState
operationalState
snmpId
(config-SysM=1)>show-table -m Snmp -p administrativeState, <TAB>
operationalState
snmpId
```

Example 76 Completion Possibility List for Attribute for Show-Table

Note: Auto-completion for attributes under the `--condition` or `-c` option is not supported in CLI for commands show and show-table.

3.2.11 Display PM Measurements

This is an optional command. It is available only if the ME supports displaying measurements through the CLI.

The `show-counters` command displays active PM counters for an MO instance. The command works analogous to the other `show` commands, but displays PM counters associated with MO instances instead of attributes. See Example 77.

```
(MeasObj=1)>show-counters
MeasObj=1
  intCounterActive=123
  multivalueIntCounter
     12
     14 <suspect>
     16
  floatCounterActive=123.456 <suspect>
  multivalueFloatCounter
     1.345
     2.456
     3.5678 <suspect>
(MeasObj=1)>show-counters --verbose
MeasObj=1
  intCounterActive=123 <PmJob=1> <Gp=15 min>
  multivalueIntCounter <PmJob=1> <Gp=15 min>
     12
     14 <suspect>
     16
  floatCounterActive=123.456 <PmJob=1> <Gp=15 min> <suspect>
  multivalueFloatCounter <PmJob=1> <Gp=15 min>
     1.345
     2.456
     3.5678
  intCounterNonActive=[] <empty>
  floatCounterNonActive=[] <empty>
(MeasObj=1)>
```

Example 77 show-counters Command Execution and show-counters Command Execution with Verbose Option

It is possible to decrease the number of displayed counters by filtering out specific counters by name or from a selected PM Job. See Example 78.

```
(MeasObj=1)>show-counters --counters floatCounterNonActive
floatCounterNonActive=[]
(MeasObj=1)>show-counters --verbose --counters floatCounterActive,
someOtherCounter --pmJob=aJob
floatCounterActive=123.456 <PmJob=aJob> <Gp=15 min>
(MeasObj=1)>
```

Example 78 show-counters Filtering

If no measurements are related to the DN/MO, error string "`ERROR: Measured object for '<Specific DN>' does not exist`" is shown. See Example 79.

```
(SystemFunctions=1)>show-counters
ERROR: Measured object for 'ManagedElement=<node_name>,SystemFunctions=1'
does not exist
```

Example 79 Error Message

3.3 Change and Display the Position in MO Tree

The actual MO position can be changed by navigation commands either in Exec mode or Config mode as follows:

- By entering the *<path>*. For example, to change position from root to `ManagedElement` and then to `SysM`:

```
(config)>ManagedElement=<node_name>
(config-ManagedElement=<node_name>)>SystemFunctions=1,
SysM=1
(config-SysM=1)
```

By entering RDNs, the CLI position can only be changed to child MO instances, not to parent MO instances, because RDNs are interpreted relative to the CLI position.

- By typing **..** or **up**, to return to the superior MO in the tree. For example, to change the current location to the parent MO instance:

```
(config-SystemFunctions=1)>up
(config-ManagedElement=<node_name>)>
```

The **..** element can be part of the RDN navigation command, for example:

```
(config-SysM=1)>..,Fm=1
(config-Fm=1)>
```

- By typing **top** to change the current location to the root position, for example:

```
(config-SystemFunctions=1)>top
(config)>
```

- By entering the LDN. For example, to change position from `SysM` to `SystemFunctions`:

```
(config-SysM=1)ManagedElement=<node_name>,Sys
temFunctions=1
(config-SystemFunctions=1)>
```

- By entering the `dn` command followed by an LDN, to navigate to any existing MO instance. The difference between the `dn` command and navigation by just entering an LDN is that the `dn` command never creates an MO instance that does not already exist.

```
(config-Snmp=1)>dn ManagedElement=<node_name>,
SystemFunctions=1,SecM=1
(config-SecM=1)>
```

- By entering the `dn` command followed by `-moc`/`-m` and an MO classname to navigate to the instance of the given class. This is only applicable where is one instance. If there are several instances of the given type, the navigation can be qualified further by using the `-condition`/`-c` parameter.

```
(config-Snmp=1)>dn –m Fm
ManagedElement=1,SystemFunctions=1,Fm=1
(Fm=1)>

>dn -m Schema -c schemaId=CmwPm
ManagedElement=1,SystemFunctions=1,SysM=1,Schema=CmwPm
(Schema=CmwPm)>
```

- By entering the `back` command to navigate back to the previous location in the MO tree.

  ```
  (config-SecM=1)>back
  (config-Snmp=1)>
  ```

  The `show-dn` command can be used to display the LDN of the current location in the MO tree.

  ```
  (config-Fm=1)>show-dn
  ManagedElement=<node_name>,SystemFunctions=1,FM=1
  ```

  The `prompt` command can be used to show the current location continuously.

  ```
  (config-Fm=1)>prompt $dn
  ManagedElement=1,SystemFunctions=1,Fm=1>
  ```

The navigation errors listed in Table 17.

Table 17 Navigation Errors

Error Message	Semantics
`ERROR: Navigation history is empty`	It is not possible to navigate back because the navigation history is empty.
`ERROR: Unable to exit from incomplete object <Current MO instance>`	It is not possible to navigate away from an MO instance that violates model constraints. The error text is followed by the identified problem that must be corrected.
`ERROR: Cannot navigate back to: <MO LDN>`	It is no longer possible to navigate back to the MO instance. The reason can be for instance that the MO has been deleted or the access rules have been changed.

Error Message	Semantics
`ERROR: Multiple MO instances found: <LDN1><LDN2>`	When the `dn` command is used with the `-moc/-m` parameter and there are several instances of the given class. The LDNs are listed in the message.
`ERROR: No instance found`	When the `dn` command is used with the `-moc/-m` parameter and there is no instance of the given class that the user can navigate to.

3.4 Display MO Instances

This section describes displaying of MO instances.

3.4.1 MO Instance Sorting

By default, MO instances are displayed in a Managed Element dependent order; for instance in creation order. The MO instances can also be displayed in a sorted order, by adding the `--sort|-s` parameter.

The sort order is:

- First, all instances with numeric names are displayed in increasing numeric order.
- Then, all other instances are displayed in alphabetical order.

Note: Sorting MO instances increases the execution time of the command. Only MO instances are sorted, not elements of sequence attributes.

3.4.2 Display Single MO

A single MO can be displayed by **`show [--sort|-s][--verbose|-v][<path>]`**.

The printout has the following format:

```
<MOC_name>=<key_value>
   <attribute>
   <attribute>
...
   <structure>
   <structure>
...
  <child_MOC_name>=<key_value>
  <child_MOC_name>=<key_value>
...
```

The list of actions for a class is deleted from the `show` output.

The command and printout syntax is shown in Example 80.

```
>show ManagedElement=<node_name>,SystemFunctions=1,SysM=1,Snmp=1
Snmp=1
   agentAddress
      host="1.1.1.1"
      port=1111
```

Example 80 Display Single MO

3.4.3 Display Single MO and Its Child MOs

A single MO and its child MOs can be displayed by **show -r [--sort|-s] [--verbose|-v] [*<path>*]** or **show --recursive [--sort|-s] [--verbose|-v] [*<path>*]**.

The printout is according to the single MO printout for the root MO and also for the child MOs. That is, all child MOs with all their properties, excluding the actions, are displayed in a recursive way.

The command and printout syntax is shown in Example 81.

```
>show --recursive ManagedElement=<node_name>,SystemFun
ctions=1,SysM=1,Snmp=1
Snmp=1
   agentAddress
      host="1.1.1.1"
      port=1111
   SnmpTargetV2C=1
      address="127.0.0.1"
      community="private"
```

Example 81 Display Single MO and Its Child MO

3.4.4 Display Single MO and Its Child MOs in Configuration Printout Format

A single MO and its child MOs can be displayed in configuration printout format by **show-config [--sort|-s] [--verbose|-v] [*<path>*]**.

The printout is according to the recursive MO printout, with the addition of navigation command **up** in specific printout positions. The resulted printout forms a valid CLI command sequence that can be input for the CLI. This printout allows configuration data export and import by copy/paste.

The command and printout syntax is shown in Example 82.

```
>show-config ManagedElement=<node_name>,⇒
SystemFunctions=1,SysM=1,Snmp=1
Snmp=1
   agentAddress
      host="1.1.1.1"
      port=1111
      up
   SnmpTargetV2C=1
      address="127.0.0.1"
      community="private"
      up
   up
```

Example 82 Configuration Printout on MOs

3.4.5 Summary of Displayed MO Properties

A summary of the elements displayed by different display options is provided in Table 18.

Table 18 Show MO Instance Matrix

Model Property	**show** *<path>*	**show --recursive** *<path>* **or show -r** *<path>*	**show-config** *<path>*	**show --verbose** *<path>* **or show -v** *<path>*	**show-config --verbose** *<path>* **or show-config -v** *<path>*
System-created child MOs	Yes	Yes	No(1)	Yes	No(1)
Child MOs recursively	No	Yes	Yes	No	Yes
Read-write attribute with value that is not `defaultValue`	Yes	Yes	Yes	Yes	Yes
Read-only or restricted attribute with value that is not `defaultValue`	Yes	Yes	No	Yes	No
Restricted attribute with value that is not `defaultValue`	No(2)	No	No	Yes	No

Table 18 Show MO Instance Matrix

Model Property	**show *<path>***	**show --recursive *<path>* or show -r *<path>***	**show-config *<path>***	**show --verbose *<path>* or show -v *<path>***	**show-config --verbose *<path>* or show-config -v *<path>***
Attribute with value equal to `defaultValue`	No	No	No	Yes	Yes
Attributes without value[3]	No	No	No	Yes	Yes
Action names	No	No	No	No	No
Not creatable child MOs	Yes	Yes	No[1]	Yes	No[1]

(1) Yes, if the MO subtree has configurable elements.
(2) This is present in the `show` attribute printout.
(3) That is, optional or nillable attributes with no values assigned.

The command is atomic and does not include non-committed changes in other transactions.

The command and printout syntax is shown in Example 83.

```
>show --verbose ManagedElement=<node_name>,SystemFunctions=1,Fm=1
Fm=1
   fmId="1"
   heartbeatInterval=30
   lastChanged="1970-01-01T01:00:00Z" <read-only>
   lastSequenceNo=0 <read-only>
   sumCritical=0 <read-only>
   sumMajor=0 <read-only>
   sumMinor=0 <read-only>
   sumWarning=0 <read-only>
   totalActive=0 <read-only>
   FmAlarmModel=FM_TEST
>
```

Example 83 Verbose Printout on MO

3.5 Change MO Attribute

This section describes how to change an MO attribute. MO attributes can be changed in Config mode only. If the changes are entered without error (that is, no printout is provided), then the change is added to the transaction and the

changed MO is locked. The changes are applied after successful commit of the transaction. The lock is released either by transaction **abort** or **commit**, as initiated by the CLI commands or the session time-out.

This section describes how to change an attribute of the following data types:

- Single-valued attribute, see Section 3.5.1 Change Single-Valued Attribute on page 75
- Single-valued attribute value deletion, see Section 3.5.2 Delete Value of Single-Valued Attribute on page 77
- Sequence, see Section 3.5.3 Change Attribute Defined as Sequence on page 78
- Struct, see Section 3.5.4 Change Attribute Defined as Struct on page 90
- Passwords, see Section 3.5.5 Change Attribute Defined as Password or Passphrase String on page 91
- MO reference, see Section 3.5.6 Change Attribute Defined as MO Reference on page 93

3.5.1 Change Single-Valued Attribute

Changing a single-valued attribute is performed by **[***<path>*, **]***<attribute_name>*=*<attribute_value>*. As a result, if no printout is provided, the operation is verified against the data type specific rules defined in the model, the attribute change is added to the transaction, and the changed MO is locked. An error printout is displayed if the operation fails, see Table 19 for example errors.

The changes are applied after the **commit** operation and the locks are released on success.

The input syntax for the attribute data types is identical to the printout syntax, see Section 3.2.1 on page 50, with the following exceptions:

- Strings can be entered with or without double quotation marks (″). A string including characters with a special meaning in the CLI (**[,=[]"]**) must be entered with quotation mark.
- Booleans are interpreted in case-insensitive way (for example, “TRUE”, “true”, and “True”) are displayed in lower case.

Attributes defined as `EcimEmpty` cannot have any value, it conveys information by its presence or absence.

Table 19 Attribute Value Change Errors

Error Message	Semantics	Data Type
`ERROR: Attribute not writable`	The user has read permission but not write permission	Any data type
`ERROR: Invalid value '<attribute_value>' for attribute '<attribute_name>'. Valid values are in range : [<min>, <max>]`	Indicates that the value specified for the attribute is out of range. The allowed values are in the specified range.	`int8`, `int16`, `int32`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`
`ERROR: Invalid value '<attribute_value>' for attribute '<attribute_name>'. Valid values are strings in a specified format. Type: '<attribute_name>?' for more information on the format to use`	The specified attribute value is incorrect and must be according to a predefined regular expression pattern. By typing **`<attribute_name>?`**, help information on the correct format to be used is displayed.	Derived String
`ERROR: Invalid value '<stringattribute_value>' for attribute '<attribute_name>'. This is not a valid escape sequence`	The specified attribute value is incorrect since it does not have a valid escape sequence. The supported escape sequences are mentioned in Table 10.	String
`ERROR: Invalid value "<attribute_value>" for attribute "<attribute_name>". Valid values are: true, false`	The specified attribute value is incorrect and the allowed values are `true` or `false`.	Boolean
`ERROR: Invalid value '<structmember_value>' for struct member '<structmember_name>'. This is not a valid escape sequence`	The specified struct member value is incorrect since it does not have a valid escape sequence. The supported escape sequences are mentioned inTable 10.	String
`ERROR: Invalid value '<attribute_value>' for parameter '<attribute_name>'.`	Indicates that the syntax of the command is not valid.	String

The command and printout syntax is shown in Example 84 through Example 88.

```
(config-ManagedElement=<node_name>)>dnPrefix=test
(config-ManagedElement=<node_name>)>commit
```

Example 84 Change Single-Valued Attribute for String

```
(config-agentAddress)>port=xyz
ERROR: Invalid value 'xyz' for attribute 'port'.
Valid values are in range : [0,4294967295]
```

Example 85 Change Single-Valued Attribute for Integer

```
(config-Snmp=1)>operationalState=DISABLED
ERROR: Attribute 'operationalState' is read-only.
(config-Snmp=1)>
```

Example 86 Change Single-Valued Attribute for Enumeration

```
(config-agentAddress)>host="$4546"
ERROR: Invalid value '$4546' for attribute 'host'.
Valid values are strings in a specified format.
Type: 'host?' for more information on the format to use>
```

Example 87 Change Single-Valued Attribute

```
(config-ManagedElement=<node_name>)>myEmptyAttribute
(config-ManagedElement=<node_name>)>commit -s
(config-ManagedElement=<node_name>)>
```

Example 88 Change Single-Valued Attribute for EcimEmpty

3.5.2 Delete Value of Single-Valued Attribute

Deleting an attribute value is performed by **no** ***<nillable_attribute_name>***.

For examples on related verification rules and errors, see Table 20.

Table 20 Attribute Value Delete Errors

Error Message	Semantics
ERROR: Attribute *<attribute_name>* is read-only (can't be deleted)	A read-only attribute cannot be deleted
ERROR: Delete only works for attribute and object types, type unrecognized for ("*<unknownType>*")	A delete operation can be performed only on attributes and object types

The command and printout syntax is shown in Example 89 and Example 90.

```
(config-ManagedElement=<node_name>)>show userLabel
userLabel=[]
(config-ManagedElement=<node_name>)>userLabel=xyz
(config-ManagedElement=<node_name>)>commit -s
(config-ManagedElement=<node_name>)>show userLabel
userLabel="xyz"
(config-ManagedElement=<node_name>)>no userLabel
(config-ManagedElement=<node_name>)>commit -s
(config-ManagedElement=<node_name>)>show userLabel
userLabel=[]
(config-ManagedElement=<node_name>)>
```

Example 89 Delete Value of Single-Valued Attribute

```
(config-Snmp=1)>no operationalState
ERROR: Attribute 'operationalState' is read-only (can't
be deleted).
(config-Snmp=1)>
```

Example 90 Delete Value of Single-Valued Attribute

3.5.3 Change Attribute Defined as Sequence

This section describes how to change a sequence.

3.5.3.1 Initialize Sequence

Initializing a sequence is performed by ***<sequenceName>*=[sequenceElement, sequenceElement, ...]**.

For examples on related verification rules and errors, see Table 24.

The command and printout syntax is shown in Example 91.

```
(config-PlainMultivalueAttrs=1)>ddtStringMultivalueAttr=[ALABAMA,
ALASKA,ARIZONA]
(config-PlainMultivalueAttrs=1)>commit -s
(config-PlainMultivalueAttrs=1)>show ddtStringMultivalueAttr
ddtStringMultivalueAttr
   "ALABAMA"
   "ALASKA"
   "ARIZONA"
```

Example 91 Initialize Sequence

3.5.3.2 Add Simple Type Element to Sequence

A new element can be added to the last position of a sequence by ***<attribute_name>=<sequence_element_value>***

For examples on related verification rules and errors, see Table 24.

The command and printout syntax is shown in Example 92.

```
(config-PlainMultivalueAttrs=1)>ddtStringMultivalueAttr=[ALABAMA,
ALASKA,ARIZONA]
(config-PlainMultivalueAttrs=1)>ddtStringMultivalueAttr=FLORIDA
(config-PlainMultivalueAttrs=1)>commit -s
(config-PlainMultivalueAttrs=1)>show ddtStringMultivalueAttr
ddtStringMultivalueAttr
   "ALABAMA"
   "ALASKA"
   "ARIZONA"
   "FLORIDA"
```

Example 92 Add Single-Valued Element to Sequence

3.5.3.3 Insert Simple Type Element in Sequence

A new sequence element can be added before the selected value of a sequence by **insert** *<attribute_name>*[*<existing_sequence_element>*]=*<new_sequence_element>*

For examples on related verification rules and errors, see Table 24.

The command and printout syntax is shown in Example 93.

```
(config-PlainMultivalueAttrs=1)>show ddtStringMultivalueAttr
ddtStringMultivalueAttr
   "ALABAMA"
   "ALASKA"
   "ARIZONA"
(config-PlainMultivalueAttrs=1)>insert ddtStringMultivalueAttr
[ALASKA]=FLORIDA
(config-PlainMultivalueAttrs=1)>show ddtStringMultivalueAttr
ddtStringMultivalueAttr
   "ALABAMA"
   "FLORIDA"
   "ALASKA"
   "ARIZONA"
```

Example 93 Add Single-Valued Elements to Sequence by Insert

3.5.3.4 Insert Simple Type Element in Sequence Using Index

A new sequence element can be inserted into a sequence with existing values by **insert** *<attribute_name>*'['@*<index>*']'=*<new_value>*.

The positional index starts from 1. The index must address an existing position in the sequence; it is not possible to add values after the last existing values.

Assume that a sequence attribute contains two values, see Example 94.

```
(config-PlainMultivalueAttr=1)>show intMultivalueAttr
intMultivalueAttr
   42
   43
(config-PlainMultivalueAttr=1)>insert intMultivalueAttr[@1]=34
(config-PlainMultivalueAttr=1)>show intMultivalueAttr
intMultivalueAttr
   34
   42
   43
```

Example 94 Insert Simple Type Element in Sequence Using Index

For examples on related verification rules and errors, see Table 24 and Table 21.

Table 21 Insert Simple Type Element in Sequence Using Index Errors

Error Message	Error Semantics
ERROR: Invalid index '*<invalid_index>*' to access a sequence containing *<no_of_set_values>* values. Use an index <=*<no_of_set_values>*	The index value is too large.
ERROR: Invalid index '*<invalid_index>*' to access a sequence containing 1 values. Use an index = 1	An index value larger than 1 is used to address a non-sequence attribute.

3.5.3.5 Change Sequence Element

A sequence element can be changed by ***<attribute_name>*[*<index>*]=*<sequence_element_value>***.

For examples on related verification rules and errors, see Table 24.

The command and printout syntax is shown in Example 95.

```
(config-PlainMultivalueAttrs=1)>show ddtStringMultivalueAttr
ddtStringMultivalueAttr
   "ALASKA"
   "ALABAMA"
   "ARIZONA"
(config-PlainMultivalueAttrs=1)>ddtStringMultivalueAttr[ALABAMA]=FLORIDA
(config-PlainMultivalueAttrs=1)>commit -s
(config-PlainMultivalueAttrs=1)>show ddtStringMultivalueAttr
ddtStringMultivalueAttr
   "ALASKA"
   "FLORIDA"
   "ARIZONA"
```

Example 95 Change Sequence Element

3.5.3.6 Change or Add Sequence Element Using Index

A sequence element can be changed or added by ***<attribute_name>'['@<index>']'=<new_value>***.

The positional index starts from 1. If the index addresses an existing sequence element, that value is replaced. If the index is one greater than the current number of values, the new value is added to the sequence.

Assume that a sequence attribute contains two values, see Example 96 and Example 97.

```
(config-PlainMultivalueAttr=1)>show intMultivalueAttr
intMultivalueAttr
   42
   43
(config-PlainMultivalueAttr=1)>intMultivalueAttr[@1]=34
(config-PlainMultivalueAttr=1)>show intMultivalueAttr
intMultivalueAttr
   34
   43
```

Example 96 Change Sequence Element Using Index

```
(config-PlainMultivalueAttr=1)>show intMultivalueAttr
intMultivalueAttr
   42
   43
(config-PlainMultivalueAttr=1)>intMultivalueAttr[@3]=34
(config-PlainMultivalueAttr=1)>show intMultivalueAttr
intMultivalueAttr
   42
   43
   34
```

Example 97 Add Sequence Element Using Index

For examples on related verification rules and errors, see Table 24 and Table 22.

Table 22 Change and Add Sequence Elements Using Index Errors

Error Message	Error Semantics
ERROR: Invalid index '*<invalid_index>*' to access a sequence containing *<no_of_set_values>* values. Use an index <= *<no_of_set_values>* for replace or use an index = *<no_of_set_values+1>* for append.	The index value is too large. It must address either an existing value, or the index after the last existing value. In the latter case, a value can be appended to the sequence.
ERROR: Invalid index '*<invalid_index>*' to access a sequence containing 1 value. Use an index = 1 for replace or use an index = 2 for append.	The index value is too large for an attribute having only one value. It must address either an existing value, or the index after the last existing value. In the latter case, a value can be appended to the sequence.

Error Message	Error Semantics
ERROR: Invalid index '*<invalid_index>*' to access a sequence containing *<no_of_set_values>* values. Use an index <= *<no_of_set_values>* for replace.	The index value is too large. It must address an existing value. No further values can be appended to the sequence.
ERROR: Invalid index '*<invalid_index>*' to access a sequence containing 1 value. Use an index = 1 for replace.	An index value larger than 1 is used to address a non-sequence attribute.
ERROR: Invalid index '*<invalid_index>*' to access a sequence containing 0 values. Use an index = 1 for append.	An index value larger than 1 is used to address a non-sequence attribute.

When changing a sequence attribute of type string and where the value to be changed starts with the character "@", it is necessary to provide the value to be changed within quotes to address it by value rather than by index.

For example:

```
(config-TestRootMoc=1)>show userLabel
userLabel
   "AB"
   "@1"
(config-TestRooMoc=1)>userLabel["@1"]=Swift
(config-TestRootMoc=1)>show userLabel
userLabel
   "AB"
   "Swift"
```

3.5.3.7 Delete Named Element from Sequence

Deleting a named element from a sequence is performed by command **no *<attribute_name>=<sequence_element_value>***.

For examples on related verification rules and errors, see Table 24.

The command and printout syntax is shown in Example 98.

```
(config-PlainMultivalueAttrs=1)>show ddtStringMultivalueAttr
ddtStringMultivalueAttr
   "ALASKA"
   "ALABAMA"
   "ARIZONA"
config-PlainMultivalueAttrs=1)>no ddtStringMultivalueAttr=ALABAMA
(config-PlainMultivalueAttrs=1)>commit -s
(config-PlainMultivalueAttrs=1)>show ddtStringMultivalueAttr
ddtStringMultivalueAttr
   "ALASKA"
   "ARIZONA"
```

Example 98 Delete Named Element from Sequence

3.5.3.8 Delete Element from Sequence Using Index

A sequence element can be removed from a sequence by command **no** ***<attribute_name>*****'['@*<index>*']'**.

The positional index starts from 1. The index must address an existing position in the sequence.

Assume that a sequence attribute contains two values, see Example 99

```
(config-PlainMultivalueAttr=1)>show intMultivalueAttr
intMultivalueAttr
   42
   43
(config-PlainMultivalueAttr=1)>no intMultivalueAttr[@1]
(config-PlainMultivalueAttr=1)>show intMultivalueAttr
intMultivalueAttr
   43
```

Example 99 Delete Element from Sequence Using Index

For examples on related verification rules and errors, see Table 24 and Table 23.

Table 23 Delete Element from Sequence Using Index Errors

Error Message	Error Semantics
ERROR: Invalid index '*<invalid_index>*' to access a sequence containing *<no_of_set_values>* values. Use an index <= *<no_of_set_values>*	The index value is too large. It must address an existing value.
ERROR: Invalid index '*<invalid_index>*' to access a sequence containing 1 values. Use an index = 1	The index value is too large. It must address an existing value for an attribute which contain only one value.

3.5.3.9 Delete All Elements from Sequence

Deleting all elements from a sequence is performed by command **no** ***<sequenceName>***.

For examples on related verification rules and errors, see Table 24.

The command and printout syntax is shown in Example 100.

```
(config-PlainMultivalueAttrs=1)>show ddtStringMultivalueAttr
ddtStringMultivalueAttr
   "ALASKA"
   "ALABAMA"
   "ARIZONA"
(config-PlainMultivalueAttrs=1)>no ddtStringMultivalueAttr
(config-PlainMultivalueAttrs=1)>commit -s
(config-PlainMultivalueAttrs=1)>show ddtStringMultivalueAttr
ddtStringMultivalueAttr=[]
```

Example 100 Delete All Elements from Sequence

3.5.3.10 Change Element in Sequence of Struct without Key

If an attribute is defined as a sequence of structure and the structure has no key member, then a sequence element can be identified using the positional index. Thus, an element in the sequence can be modified by addressing that element with the positional index. The positional index is supported only for sequence of structures. The positional index is identified by the ‘‘@’’ character for all the struct elements in the sequence. The index numbering starts from 1.

The command and printout syntax is shown in Example 101.

```
(config-Snmp=1)>show -v agentAddress
agentAddress[@1]
  host="1.1.1.1"
  port=26343
agentAddress[@2]
host=”2.2.2.2”
port=23154
(config-Snmp=1)>agentAddress[@2],host=4.4.4.4 port=4444
(config-Snmp=1)>show -v agentAddress
agentAddress[@1]
  host="1.1.1.1"
  port=26343
agentAddress[@2]
  host="4.4.4.4"
  port=4444
```

Example 101 Change Element in Sequence of Struct without Key

3.5.3.10.1 Add a New Keyless Struct Element to the Sequence

A new element can be appended to the end of the sequence, as follows:

1. Without specifying positional index:

   `[<path>,]<noKeyStructSequence>`

2. Setting member without specifying positional index:

   `[<path>,]<noKeyStructSequence>,<member_Name>=<member_value>`

3. Specifying positional index:

   `[<path>,]<noKeyStructSequence>[@<N>]`

4. Setting member by specifying positional index:

   `[<path>,]<noKeyStructSequence>[@<N>],<member_Name>=<member_value>`

A new struct instance is added to the end of the sequence and the user navigates to the newly created instance, if the multiplicity for the attribute has not reached the maximum value.

If the struct members are set as part of the append operation, then the specified values are set to the struct members.

The command and printout syntax for previous cases are shown in Example 102 through Example 106.

```
(config-Snmp=1)>agentAddress
(config-agentAddress[@2])>up
(config-Snmp=1)>show -v agentAddress
agentAddress[@1]
   host="0.0.0.0"
   port=26343
agentAddress [@2]
```

Example 102 Add a New Keyless Struct Element to the Sequence without Index

```
(config-Snmp=1)>agentAddress[@2]
(config-agentAddress[@2])>up
(config-Snmp=1)>show -v agentAddress
agentAddress[@1]
  host="0.0.0.0"
  port=26343
agentAddress[@2]
```

Example 103 Add a New Keyless Struct Element to the Sequence with Index

```
(config-Snmp=1)>agentAddress,port=999
(config-Snmp=1)>show agentAddress
agentAddress
  host="0.0.0.0"
  port=26343
agentAddress
  port=999
```

Example 104 Add a New Struct Element to the Sequence by Setting Struct Member, without Index

```
(config-Snmp=1)>agentAddress[@3],host=9.9.9.9
(config-Snmp=1)>show agentAddress
agentAddress
  host="0.0.0.0"
  port=26343
agentAddress
  port=999
agentAddress
  host="9.9.9.9"
```

Example 105 Add a New Struct Element to the Sequence by Setting Struct Member, with Index

```
(config-Snmp=1)>agentAddress[@5]
ERROR: Invalid index '5' for a sequence containing 1
values. Use an index <=2
```

Example 106 Add a New Keyless Struct Element to the Sequence with Invalid Positional Index

3.5.3.10.2 Delete Keyless Struct Sequence

An element at position *<N>* in a sequence of keyless struct can be deleted by `no [<path>,]<noKeyStructSequence>[@<N>]`.

Struct element at position *<N>* in the sequence is deleted. All instances after position `N` are shifted up by one position.

All the struct elements in the sequence can be deleted by command `no <noKeyStructSequence>`.

The command and printout syntax for these cases are shown in Example 107 through Example 109.

```
(config-Snmp=1)>show -v agentAddress
agentAddress[@1]
  host="0.0.0.0"
  port=26222
agentAddress[@2]
  host="9.9.9.9"
  port=9999
agentAddress[@3]
  host="2.2.2.2"
  port=2222
(config-Snmp=1)>no agentAddress[@2]
(config-Snmp=1)>show -v agentAddress
agentAddress[@1]
  host="0.0.0.0"
  port=26222
agentAddress[@2]
  host="2.2.2.2"
  port=2222
```

Example 107 Delete Keyless Struct Element at Positional Index <N>

```
(config-Snmp=1)>show -v agentAddress
agentAddress[@1]
  host="0.0.0.0"
  port=26222
agentAddress[@2]
  host="9.9.9.9"
  port=9999
agentAddress[@3]
  host="2.2.2.2"
  port=2222
(config-Snmp=1)>no agentAddress[@2],host
(config-Snmp=1)>show -v agentAddress
agentAddress[@1]
  host="0.0.0.0"
  port=26222
agentAddress[@2]
  port=2222
```

Example 108 Delete a Struct Member of a Struct Element in a Sequence of Keyless Struct

```
(config-Snmp=1)>show agentAddress
agentAddress
  host="0.0.0.0"
  port=26222
agentAddress
  host="9.9.9.9"
  port=9999
agentAddress
  host="2.2.2.2"
  port=2222
(config-Snmp=1)>no agentAddress
(config-Snmp=1)>show agentAddress
agentAddress=[]
```

Example 109 Delete Whole Sequence of Keyless Struct

3.5.3.10.3 Insert Element to Sequence of Keyless Struct

A new struct element can be inserted at a particular position for a sequence of keyless structs by using the **insert** command.

Syntax: **insert [*<path>*,]*<noKeyStructSequence>*[@*<N>*]**

A struct instance is created at position *<N>* and the existing struct elements in the sequence after position *<N>* are moved down by one position.

The command and printout syntax is shown in Example 110.

```
(config-Snmp=1)>insert agentAddress[@2]
(config-agentAddress[@2])>host=9.9.9.9
(config-agentAddress[@2])>port=9999
(config-agentAddress[@2])>up
(config-Snmp=1)>show -v agentAddress
agentAddress[@1]
  host="0.0.0.0"
  port=26222
agentAddress[@2]
  host="9.9.9.9"
  port=9999
agentAddress[@3]
  host="2.2.2.2"
  port=2222
```

Example 110 Insert Element to Sequence of Keyless Struct

3.5.3.11 Change Element in Sequence of Keyed Struct

If the attribute is defined as a sequence of struct and the struct has a key member, then it is possible to identify a sequence element and it is possible to change the CLI position in the sequence.

As a result, existing sequence elements can be changed and new elements can be added without changing the existing element, as shown in Example 111.

```
(config-PlainMultivalueAttrs=1)>countries=Sweden
(config-countries=Sweden)>capital=Stockholm
(config-countries=Sweden)>population=2000000
(config-countries=Sweden)>up
(config-PlainMultivalueAttrs=1)>countries=Norway
(config-countries=Norway)>capital=Oslo
(config-countries=Norway)>up
(config-PlainMultivalueAttrs=1)>show countries
countries="Sweden"
  capital="Stockholm"
  population=2000000
countries="Norway"
  capital="Oslo"
(config-PlainMultivalueAttrs=1)>show countries=Sweden
countries="Sweden"
  capital="Stockholm"
  population=2000000
(config-PlainMultivalueAttrs=1)>no countries=Sweden
(config-PlainMultivalueAttrs=1)>show countries
countries="Norway"
  capital="Oslo"
```

Example 111 Chang Element in Sequence of Keyed Struct

The countries attribute of class `PlainMultivalueAttrs` is defined as a sequence of struct. The struct has the following members:

- “name”, defined as string, that is, the key of the struct
- “capital”, defined as a string
- `population`, defined as `int64` with default value `0`

3.5.3.12 Common Error Messages in Sequence Operations

Common error messages in sequence operations are shown in Table 24.

Table 24 Common Error Messages in Sequence Operations

Error Message	Semantics
`ERROR: Value must be unique`	The sequence elements are set to be unique, that is, the `nonUnique` property is not present
`ERROR: Multiplicity of the attribute ("<attribute_name>") at max limit`	The sequence element `maxLength` property is set and the number of elements exceed this limit

Error Message	Semantics
ERROR: Multiplicity of the attribute (" *<attribute_name>* ") at minimum limit	The sequence element minLength property is set and the number of elements is lower than this limit
ERROR: Invalid index *<positional_Index>* to access a sequence containing *<existing number of values>* values. Use an index <= *<valid_values>*	The keyless struct element at the specified positional index for a keyless struct sequence does not exist and the allowed values are lesser than or equal to the *<valid_values>*.

3.5.4 Change Attribute Defined as Struct

A struct can be changed by ***<structName,structElement>=<structElement_value>***. As a result, if no printout is provided, the changed struct is added to the transaction and the changed MO is locked. An error printout is displayed if the operation fails, see Table 25 for example errors.

If the struct isExclusive property is set, then setting one struct member means that all other struct members are automatically unset.

The changes are applied after operation **commit** and the locks are released on success. It is only possible to navigate away from the MO if all members of a struct inside the MO are set correctly.

Table 25 Common Error Message in Struct Operations

Error Message	Semantics
ERROR: Failed to set struct key attribute '*<attribute_name>*' to '*<attribute_value>*'. Key must be unique.	The key value is not unique

The command and printout syntax is shown in Example 112.

```
(config-ManagedElement=<node_name>)>productIdentity=1,
productDesignation=xyz
(config-ManagedElement=<node_name>)>productIdentity=1,
productNumber=1234
(config-ManagedElement=<node_name>)>productIdentity=1,
productRevision=1.1
(config-ManagedElement=<node_name>)>commit -s
(config-ManagedElement=<node_name>)>
```

Example 112 Change Struct

3.5.4.1 Struct of Sequence

A struct can have a sequence as its element. Initialization, adding, and modifying values of such struct elements is similar, see Section 3.5.3 Change Attribute Defined as Sequence on page 78.

3.5.5 Change Attribute Defined as Password or Passphrase String

Two different types of passwords/passphrases exist, and they are treated slightly differently in the CLI. In this section, the two terms legacy password and passphrase string are used to distinguish the types.

Legacy passwords can only appear as single valued attributes, whereas `passphrase` strings can appear both as single valued and as sequences and both as attributes and in struct members. Verbose help shows a legacy Password as type `password` and a passphrase string as type string with the tag `passphrase`.

The way to enter values of both types interactively depend on how the ME is configured. Either they are entered visibly, similar to how other attribute types are assigned, or they are entered hidden using special prompts. For details, see Section 3.5.5.1 Visible Entry of Values on page 91 and Section 3.5.5.2 Hidden Entry of Values on page 92.

When entered non-interactively, for instance in scriptmode or when commands are pasted into the CLI, the values are always entered visibly.

3.5.5.1 Visible Entry of Values

A legacy password is entered the following ways:

- An encrypted value can be assigned to a password attribute by *<attribute_name>=<attribute_value>*.
- A cleartext value can be assigned to a password attribute by *<attribute_name>=<attribute_value>* `cleartext`.

A `passphrase` string value is always assigned by *<attribute_name>=<attribute_value>*. The *<attribute_value>* must be cleartext, unless the ME is configured to accept encrypted values. The information on whether a value is encrypted or not is in this case encoded within the value itself.

For an encrypted value to be valid, it must be taken from the output of a CLI `show` type command. An encrypted value is normally only valid on one ME, but MEs can be configured so that values can be copied between MEs. The encrypted values of legacy passwords and passphrase strings are only valid as input of the respective type; it is not possible to copy encrypted values between the two types.

Cleartext values of both types become encrypted when they are entered. When displaying legacy passwords, the values are always shown encrypted. When

displaying passphrase strings, the values are normally not displayed at all, but an ME can be configured to display encrypted values.

The command and printout syntax is shown in Example 113, Example 114, Example 115, and Example 116.

```
(config-A=1)>myPassword="foobarmypasswd398" cleartext
(config-A=1)>show myPassword
myPassword="ei28fwisgieatge5646i"
```

Example 113 Non-Encrypted Legacy Password

```
(config-A=1)>myPassword="34sfsSFGargy5ghyj124"
(config-A=1)>show myPassword
myPassword="34sfsSFGargy5ghyj124"
```

Example 114 Already Encrypted Legacy Password

```
(config-A=1)>myPassphrase="foobarmypasswd398"
(config-A=1)>show myPassphrase
myPassword="********"
```

Example 115 Passphrase String, Encrypted Input and Output is not Allowed

```
(config-A=1)>myPassphrase="foobarmypasswd398"
(config-A=1)>show myPassphrase
myPassphrase="encrypted:hdg4jdldf8gfk5jd"
(config-A=1)>myPassphrase="encrypted:hdg4jdldf8gfk5jd"
(config-A=1)>show myPassphrase
myPassphrase="hdg4jdldf8gfk5jd"
```

Example 116 Passphrase String, Encrypted Input and Output is Allowed

3.5.5.2 Hidden Entry of Values

When interactively entering a legacy password or a passphrase string, the following applies:

- When the equals sign is entered after the name of an attribute or struct member, or when the **Tab** key is used to complete the name of such an attribute or struct member, the cursor is moved to a new line where a special prompt is presented for the entry of the value. The characters entered at this prompt are not echoed.

- When the **CR** key or the **Enter** key is pressed, a second prompt is presented for the repeated entry of the value. As the **CR** key or the **Enter** key is pressed at this prompt, and if the two entered values were identical, the normal command line is again presented, where a sequence of eight asterisks is representing the value. See Example 116.

Example 117 show an entry of hidden legacy password or `passphrase` string.

```
(config-A=1)>myPassword=
Enter myPassword:
Repeat myPassword:
(config-A=1)>myPassword=********
```

Example 117 Entry of Hidden Legacy Password or Passphrase String

It is not possible to edit the contents of the command line where the value is represented by asterisks. If the type is legacy password, the two choices here are to either press the **CR** key or the **Enter** key to change the attribute member, or to press **Ctrl+C** to clear the line. If the type is passphrase string, it is possible to change further attributes or struct members on the same line.

If the two entered values were not identical, an error message is shown and the first password prompt is redisplayed, as shown in Example 118.

```
(config-A=1)>myPassword=
Enter myPassword:
Repeat myPassword:
ERROR: The inputs do not match
Enter myPassword:
```

Example 118 Mismatching Values

It is possible to cancel the entry of the password strings by pressing **Ctrl+C**.

Note: It is not possible to enter an encrypted legacy password in this way. The entered values are automatically treated as cleartext. If a legacy password attribute must be changed to an already encrypted value, this must be done in scriptmode or by pasting a full line including the encrypted value into the CLI. Encrypted passphrase string values, on the other hand, are possible to enter at these special prompts.

3.5.6 Change Attribute Defined as MO Reference

Changing an attribute of MO reference type is the same as changing a string attribute, with the following special conditions:

- The MO reference attribute value is the LDN of the referred MO.
- If the definition of the MO reference attribute contains a certain MOC type, the referred MO must be an instance of this MOC.
- The referred MO instance does not have to exist. If the `ManagedElement` ID does not match the one defined in the system, the CLI automatically corrects it.

As an alternative to providing the full LDN, the CLI also allows the value to be an MO path relative to the current position, using the same rules as when navigating to an MO instance. This makes it possible to, for instance, navigate to the MO instance that is to be referred to, and then give the MO reference attribute the value ".". When assigning the value, the CLI automatically translates the relative path to an LDN.

A MO reference value can, as any string, be entered unquoted as long as it does not contain any space characters. If a MO instance name contains a space, it is possible to quote only the MO instance name within the DN. It is however recommended to always quote the full MO reference values, as syntactical ambiguities can arise, in particular in commands changing sequences of MO references.

3.5.7 Change Multiple Attributes on One Command Line

It is possible to change multiple attributes belonging to the same MO instance on one single command line, by separating the subcommands with space characters.

The supported modifications are as follows:

- Change single-value attribute, see Section 3.5.1 Change Single-Valued Attribute on page 75.
- Initialize sequence, see Section 3.5.3.1 Initialize Sequence on page 78.
- Add single-value element to sequence, see Section 3.5.3.2 Add Simple Type Element to Sequence on page 78.

The CLI handles all subcommands on the same line as one atomic command, meaning that the result is either that all the subcommands on the command line are executed successfully or no changes are made.

Example 119 shows how to change multiple attributes on one command line.

```
(config-MocA=1)>intMultiValueAttr=1 stringAttr="abc" enumMultiValueAttr=[ONE, TWO]
(config-MocA=1)>show
MocA=1
   intMultiValueAttr=1
   enumMultivalueAttr
      ONE
      TWO
   stringAttr="abc"
(config-MocA=1)>
```

Example 119 Change Multiple Attributes on One Command Line

Attributes outside of the current MO location can be changed by giving the path to the MO where the attributes are located followed by a comma character and then list the attributes changes.

Example 120 shows how to change multiple attributes on one command line from outside MO.

```
(config-MocC=1)>MocB=1,MocA=1,intMultiValueAttr=1 stringAttr="abc" enumMultiValueAttr=[ONE, TWO]
(config-MocC=1)>show MocB=1,MocA=1
MocA=1
   intMultiValueAttr=1
   enumMultivalueAttr
      ONE
      TWO
   stringAttr="abc"
(config-MocC=1)>
```

Example 120 Change Multiple Attributes on One Command Line from Outside MO

Multiple struct members can also be set on one command line according to the principles that applies for attributes.

3.6 Create MO

MO instances can be created in an atomic way in transaction, that is, in Config Mode only. As a consequence, the configuration changes are not applied by entering the changes, but after successful **commit** of the transaction.

To create an MO:

1. Enter Config mode:

   configure

2. Check if the MO instance exists:

   show *<path>*

   Note: This step is needed, because the CLI command for MO creation is identical to the CLI command for changing the CLI position to an existing MO. The successes of both operations are indicated in the same way, by providing no printout.

3. Select action according to the possible results as follows:

   - No error message is displayed, but the requested MO is displayed. In this case, the MO exists. Continue the operation by changing the MO attributes, see Section 3.5 Change MO Attribute on page 74, according to the required changes.

   - The error message `ERROR: Specific element not found` indicates that the MO does not exists. Continue to Step 4.

4. If the MO does not exist, enter the name of the MO according to the MO naming rules of the key attribute (see Section 3.6.1 MO Naming Rules on page 97) to create the desired MO.

   Example of key attribute of type string:

   ```
   ManagedElement=<node_name>,SystemFunctions=1,S
   ysM=1,NtpServer=myServer
   ```

   If the MO key attribute is an enumeration type, press the **Tab** key to list the available values with information about which values that are not yet instantiated. Example:

   ```
   (config-KeyAttrMoc=1)>EnumKeyAttrMoc=<TAB>
   ONE
   THREE <new>
   TWO <new>
   (config-KeyAttrMoc=1)>EnumKeyAttrMoc=
   ```

   - An error message is provided if verification fails against any available constraint. In this case of error, modify the DN to comply to the constraints. For examples of error messages, see Table 26.

- If no error printout is provided, then the operation succeeded and the CLI position are changed to the new MO. The parent of the newly created MO, and the MO itself, is locked. Example:
  `(config-NtpServer=myServer)>`

  Note: If the MO is created and the value for a mandatory attribute is not assigned, then the CLI position change is rejected by an error indication, which lists the names of the mandatory attributes. Example: `ERROR: Following mandatory attributes are not set for the parent (NtpServer=new): < serverAddress >.`

5. Press the **Tab** key to list the available optional and mandatory attributes to be set. Example:

```
(config-NtpServer=testServer)><TAB>
administrativeState
serverAddress
userLabel
```

6. Request Verbose Help to determine if the attribute is mandatory and if it has a default value. Example:

```
(config-NtpServer=testServer)>administrativeState ?
administrativeState BasicAdmState <LOCKED|UNLOCKED>
[optional]
Locks or unlocks the operation of the NTP client
function.
This is a convenience function to permit some or all
NtpServer instances to be temporarily locked without
having to delete the object.
```

7. Set the mandatory attributes and the needed optional attributes:

```
(config-NtpServer=myServer)>serverAddress="22.22.22.22"
```

8. Commit the operation:

```
(config-NtpServer=myServer)>commit -s
```

9. Verify the operation result, as follows:

   - If the MO creation failed, then the error message is printed on the error reason. In this case, act according to the error indication.

   - If no result message is printed, the MO creation succeeded, the CLI position is in the new MO position in Config mode, and a new transaction is created automatically. The lock on parent of the newly created MO is released.

Table 26 MO Creation Errors

Error Message	Semantics
ERROR: Parent '*<parent_DN>*' does not exist	The parent DN does not exist
ERROR: Can not instantiate system created object	Instantiation of system-created objects is not allowed
ERROR: Cardinality is at upper limit for class (*<MO_class_name>*), cannot create object	An MO instance cannot be created because of a restriction on cardinality
ERROR: MO creation failed for classname: *<MO_class_name>*, error code: *<error_code>*	Other implementation-specific cases
ERROR: No create permission for '*<DN>*'	The user has read permission but not create permission
ERROR: Command not found	The user has not read or write permission
ERROR: Instances of '*<moc_name>*' are not creatable	Creation of MO is not possible through the NBI

The command and printout syntax is shown in Example 121.

```
(config-Snmp=1)>SnmpTargetV1=OSS-42
(config-SnmpTargetV1=OSS-42)><TAB>
address
administrativeState
community
isMibWritable
port
(config-SnmpTargetV1=OSS-42)>address=1.2.3.4
(config-SnmpTargetV1=OSS-42)>administrativeState=UNLOCKED
(config-SnmpTargetV1=OSS-42)>community=zoneA
(config-SnmpTargetV1=OSS-42)>isMibWritable=Tab
(config-SnmpTargetV1=OSS-42)>isMibWritable=true
(config-SnmpTargetV1=OSS-42)>port=777
(config-SnmpTargetV1=OSS-42)>commit -s
(config-SnmpTargetV1=OSS-42)>
```

Example 121 Create MO

3.6.1 MO Naming Rules

The MO naming rules depend on the type of the key attribute.

3.6.1.1 String

The unsupported characters are as follows:

- All ASCII characters in range 32–126 are supported except the following restricted characters:
  - comma (**,**)
  - equals sign (**=**)
  - plus sign (**+**)
  - less-than sign (**<**)
  - greater-than sign (**>**)
  - number sign (**#**)
  - semicolon (**;**)
  - reverse slash (****)
  - quotation mark (**"**)
  - asterisk (*****)
- All other characters, including restricted characters, must be entered as **\XX**, where XX is the two-digit hexadecimal ASCII code of the character.

3.6.1.2 Integer

The value must be numeric and inside the range of the underlying type, which can be int8, int16, int32, int64, uint8, uint16, uint32, or uint64.

3.6.1.3 Enumeration

The value must be the name of a member of enumeration.

3.7 Reinitialize MO

An MO instance, a single attribute, a struct instance within a sequence of structs, or a single struct member can be reset to its initial state using the `reinit` command. The `reinit` command can be used only in Config mode.

The syntax is: **reinit [.** | ***<attribute_name>*** **]**, where *<attribute_name>* indicates an attribute, a struct, a struct instance within a sequence of structs based on key or index and struct member.

The initial state of an attribute is what it has when an MO instance is newly created. That is, if an attribute has a default value, the `reinit` command sets its value to the default value, and otherwise it sets it to empty.

The initial state of an MO instance is what it has when all its attributes have their default values, or if they lack default values, no values. The `reinit` command only affects one single MO instance, it does not reinitialize or delete any child MO instances.

Read-only attributes cannot be reinitialized, as their values are assigned by the system. Key attributes and key struct members also cannot be reinitialized. Restricted attributes can be reinitialized only in newly created and not yet committed MO instances. Such non-resettable attributes and struct members are ignored when a whole MO or struct instance is reinitialized.

Note: When an attribute, that is, a sequence of structs, is reinitialized, all existing struct instances are deleted, and if the minimum multiplicity is larger than zero, a new struct instance is created. To keep and reinitialize all existing struct instances, the instances must be reinitialized one by one.

When an MO instance or an attribute is reinitialized, the attribute minimum multiplicity constraints cannot be fulfilled. In this case, a warning message is printed and it is not possible to navigate out of the MO instance until the condition is corrected. The warning messages have the format minimum multiplicity of *<min>* is violated for attribute *<attribute_name>*.

It is not possible to reinitialize an MO instance or an attribute unless the current position is inside the appropriate MO instance.

Table 27 shows the reinitialized errors.

Table 27 Reinitialized Errors

Error Message	Error Semantic
`ERROR: Cannot reinitialize key attribute <attribute_name>.`	Indicates that CLI user attempts to reinitialize key attribute
`ERROR: Cannot reinitialize read-only attribute <attribute_name>.`	Indicates that CLI user attempts to reinitialize read-only attribute
`ERROR: Cannot reinitialize restricted attribute <attribute_name>.`	Indicates that CLI user attempts to reinitialize restricted attribute
`ERROR: Cannot reinitialize the key member <struct_member_name>` of the struct `<struct_id>.`	Indicates the CLI user attempts to reinitialize key member of struct.

Error Message	Error Semantic
ERROR: *<attribute_name>* does not have any instances.	Indicates that CLI user attempts to reinitialize the struct that does not have any instance.
ERROR: Cannot reinitialize attribute, no write permission for *<attribute_name>*.	Indicates that CLI user is trying to reinitialize an attribute that does not have write permission.

Examples:

Assume an attribute `attrWithDefaultValue` has default value “abc”, and another attribute, `attrWithoutDefaultValue`, has been assigned value 45. The `reinit` command resets the value of `attrWithoutDefaultValue` to empty as shown in Example 122.

```
(config-TestRootMoc=1)>show -v
TestRootMoc=1
   attrWithDefaultValue=”abc” <default>
   attrWithoutDefaultValue=45
   aSimpleStruct
       memberWithDefaultValue=123 <default>
       memberWithoutDefaultValue=23
(config-TestRootMoc=1)>reinit
(config-TestRootMoc=1)>show -v
TestRootMoc=1
   attrWithDefaultValue=”abc” <default>
   attrWithoutDefaultValue=[] <empty>
   aSimpleStruct
       memberWithDefaultValue=123 <default>
       memberWithoutDefaultValue=[] <empty>
```

Example 122 Reset MO

Assume an attribute `attrWithDefaultValue` has default value “abc” and the value 123 has been assigned to the attribute. The `reinit` command resets `attrWithDefaultValue` to its default value as shown in Example 123.

```
(config-TestRootMoc=1)>show -v  attrWithDefaultValue
attrWithDefaultValue=”abc” <default>
(config-TestRootMoc=1)>attrWithDefaultValue=123
(config-TestRootMoc=1)>show -v  attrWithDefaultValue
attrWithDefaultValue=”123”
(config-TestRootMoc=1)>reinit  attrWithDefaultValue
(config-TestRootMoc=1)>show -v attrWithDefaultValue
attrWithDefaultValue=”abc” <default>
```

Example 123 Reset Attribute

Assume an attribute `mandatoryAttr` has minimum multiplicity 1 and no default value. The `reinit` command resets the attribute value and a warning message is printed on violation of minimum multiplicity constraints as shown in Example 124.

```
(config-TestRootMoc=1)>show -v
TestRootMoc=1
   mandatoryAttr=123
 (config-TestRootMoc=1)>reinit  mandatoryAttr
WARNING:  Minimum multiplicity of 1 is violated for attribute (mandatoryAttr)
(config-TestRootMoc=1)>show -v  mandatoryAttr
mandatoryAttr=[] <empty>
```

Example 124 Reset Command with Minimum Multiplicity Constraints Violation

The `reinit` command deletes all instances in a sequence of structs and creates one instance when only the `struct` attribute name is given as parameter, as shown in Example 125. Otherwise, when a struct key value or an index is given, only the members of the specified instance are reinitialize, excluding any key member, as shown in Example 126 and Example 127.

```
(config-PlainMultivalueAttrs=1)>show -v aKeylessStruct
aKeylessStruct[@1]
   int1=[] <empty>
   str1=[] <empty>
aKeylessStruct[@2]
   int1=3
   str1=[] <empty>
(config-PlainMultivalueAttrs=1)>reinit aKeylessStruct
WARNING: Minimum multiplicity of 1 is violated for attribute (aKeylessStruct)
struct member: str1
WARNING: Minimum multiplicity of 1 is violated for attribute (aKeylessStruct)
struct member: int1
(config-PlainMultivalueAttrs=1)>show -v aKeylessStruct
aKeylessStruct[@1]
   int1=[] <empty>
   str1=[] <empty>
(config-PlainMultivalueAttrs=1)>
```

Example 125 Reset struct

```
(config-ResetTestMocWithMultiValueAttr=1)>show -v mandatoryKeyedStruct
mandatoryKeyedStruct
   capital=[] <empty>
   name=”1” <key>
   population=0 <default>
mandatoryKeyedStruct="2"
   capital="1"
   name="2" <key>
   population=0 <default>
(config-ResetTestMocWithMultiValueAttr=1)>reinit mandatoryKeyedStruct=2
(config-ResetTestMocWithMultiValueAttr=1)>show -v mandatoryKeyedStruct
mandatoryKeyedStruct
   capital=[] <empty>
   name=”1” <key>
   population=0 <default>
mandatoryKeyedStruct="2"
   capital=[] <empty>
   name="2" <key>
   population=0 <default>
(config-ResetTestMocWithMultiValueAttr=1)>
```

Example 126 Reset a Specific Instance of a struct with a Key Member

```
(config-ResetTestMocWithMultiValueAttr=1)>show -v sequenceOfKeylessStruct
sequenceOfKeylessStruct[@1]
   int1=3
   str1=”3”
(config-ResetTestMocWithMultiValueAttr=1)>reinit sequenceOfKeylessStruct[@1]
(config-ResetTestMocWithMultiValueAttr=1)>show -v sequenceOfKeylessStruct
sequenceOfKeylessStruct[@1]
   int1=2 <default>
   str1=[] <empty>
(config-ResetTestMocWithMultiValueAttr=1)>
```

Example 127 Reset a Specific Instance of a struct without a Key Member

3.8 Delete MO

The MO can be deleted in Config mode only. The configuration changes are not applied by entering the changes, but after a successful commit of the transaction.

To delete an MO:

1. Enter Config mode:

   ```
   configure
   ```

2. Delete the MO by executing command **no** with the RDN of the MO. Example:

   ```
   no ManagedElement=<node_name>,SystemFunctions=1,SysM=
   1,NtpServer=myServer,NtpServer=myServer
   ```

   The possible results are as follows:

   - An error message is provided if verification failed against any available constraint. In this case, modify the DN to comply to the constraints. For examples of error messages, see Table 28.
   - If no printout is provided, then the operation succeeded and the deleted MO, its parent MO, and its children MOs are locked.

3. Commit the operation:

   ```
   commit -s
   ```

4. Verify the operation result, as follows:

   - If the MO deletion failed, an error message with error reason is printed. In this case, act according to the error indication.
   - If no result message is printed, the MO deletion succeeded, the CLI positions is not changed, the locks are released, and a transaction is created automatically.

Note: An MO can be deleted even if it is pointed by an MO reference.

Table 28 MO Deletion Errors

Error Message	Semantics
`ERROR: Cardinality is at lower limit for class ("<MO_class_name>"), cannot delete object`	The object cannot be deleted because of cardinality constraint
`ERROR: Can not delete system created object`	Deletion of system-created object is not allowed

Error Message	Semantics
`ERROR: No delete permission for '<DN>'`	The user has read permission but not delete permission
`ERROR: Command not found`	The user has not read or write permission
`ERROR: Instances of '<moc_name>' are not deletable`	Deletion of MO is not possible through the NBI

The command and printout syntax is shown in Example 128.

```
(config)>no ManagedElement=<node_name>
ERROR: Can not delete system created object
(config)>
```

Example 128 Delete System-Created MO

3.9 MO Actions

Actions are executed in Config mode and Exec mode on existing MOs, but not on those that are created in a transaction and not yet committed (in the Config mode case).

The following cases describe when an MO is locked:

- If an attribute change is requested, the MO that contains the attribute is locked.
- If an MO creation is requested, the MO and its parent MO are locked.
- If an MO deletion is requested, the MO, its parent MO, and its children MOs are locked.
- If an action execution is requested, the MO that contains the action is locked.

The locks are released if the transaction is committed or ended, or if the result of action execution is received.

3.9.1 Action Request

The action execution can be requested by **`[<path>,] [.,]<action> [--<action_parameter_name> <action_parameter_value>]*`**

Optional parameters are supported with this format. A parameter is considered optional if the parameter has a default value in the MOM or if its minimum multiplicity is zero.

Values must be specified for all the parameters that are not optional. Action parameters defined as struct (for example, the `EcimPassword` struct) or sequence multi-valued attribute are not supported.

If an action name is conflicting with any of the existing command name, then request for action must give as: `'.,<action_name>'` if action must be executed from the current DN.

Example 129 shows the syntax for executing an action, which has name conflicting with the existing command name (`show`) from the current DN.

```
(config-PrecedenceThing=2)>.,show
```

Example 129 Syntax for Executing an Action

3.9.1.1 Alternative Hidden Password Entry

An ME can be configured to hide the characters of entered passwords.

If that is the case, and if scriptmode is off, the following applies:

- When the **Space** key is pressed after the name of an action parameter defined as a password, or when the **Tab** key is used to complete the name of such a parameter, the cursor is moved to a new line where a special prompt is presented for the entry of the password string. The characters entered at this prompt are not echoed.
- When the **CR** key or the **Enter** key is pressed, the normal command line is again presented, where a sequence of 8 asterisks is representing the password. See Example 130.

```
Schema=1)>export --password
Enter password:
(Schema=1)>export --password ********
```

Example 130 Entry of Hidden Password

It is not possible to edit the contents of the command line where the password is represented by asterisks, but it is possible to add additional parameters.

It is possible to cancel the entry of the password string by pressing **Ctrl+C**.

3.9.2 Response to Action Request

If the action request is completed with error, then the error reason is indicated in printout `ERROR: <error_text>`. For examples on error messages, see Table 29.

If an action request is completed without error, the following apply:

- No printout is provided if the action return value is defined as `void`.

- Otherwise, a printout is provided according to the action return type. The action printout format is identical to **show [*<path>*,]*<attribute>*** of the same data type, see Section 3.2 Display Information on page 49.

Note: Error indication in an exception parameter is not supported.

3.9.3 Action Start

Depending on the semantics, the success of an action request can mean that the action execution is completed or started, which results in asynchronous or synchronous action execution.

The start of the actual action execution can be bound to various conditions depending on the action type, for example, as follows:

- Immediate Start – The action execution starts immediately after the request is entered.
- Start by Commit – The action execution starts after the request is entered and the related transaction is committed.
- Confirmed Start – To start the action execution, additional confirmation is needed.
- Start with Timer – The action execution starts after a predefined time-out. This can be combined with confirmed start.
- Any action-specific preconditions, for example, internal states and parallel activities.

For the action semantics and start conditions, see the action description in the `CLI Help` or in the model description.

3.9.4 Action Result

Table 29 Action Error Messages

Error Message	Semantics
`ERROR: No execute permission for action '<action_name>'`	The user has no execute permissions but the action is visible for the user.
`ERROR: Too many arguments for action (<action_name>) that takes <number_of_parameters> parameters <paramname1, paramname2, ...>`	The number of parameters are higher than expected.

Error Message	Semantics
ERROR: Too few arguments for action (*<action_name>*) that takes *<number_of_parameters>* parameters *<paramname1, paramname2, ...>*	The number of parameters are fewer than expected.
ERROR: Call command failed, error code *<error_code>*	Other implementation-specific cases.
ERROR: Invalid GNU Style Syntax for parameter : *<param_name>*	Indicates that the parameter is not in GNU style.
ERROR: No parameter found with name : *<param_name>*	Indicates the parameter with the name specified is not found.
ERROR: Duplicate parameters not allowed : *<param_name>*	Indicates that the parameter with name has more than an occurrence in the command.
ERROR: No value found for parameter : *<param_name>*	Indicates that there is no value specified for given parameter name.
ERROR: Invalid value '*<param_name>*' for parameter '*<paramname>*'. Valid values are: ' LOCKED UNLOCKED SHUTTING_DOWN'. Type: '*<action_name>*?' for more information on the values to use.	Indicates that the value specified for the action parameter is incorrect. By typing ***<action_name>?***, help information on the allowed values is displayed.
ERROR: Invalid value "*<param_name>*" for parameter "*<paramname>*". Valid values are: true, false	Indicates that the value specified for the action parameter is incorrect. By typing ***<action_name>?***, help information on the allowed values is displayed
ERROR: Invalid value '*<param_name>*' for parameter '*<paramname>*'. Valid values are in range : [*<min>*, *<max>*]	Indicates that the value specified for the action parameter is out of range. The allowed values are in the specified range.
ERROR: Invalid value '*<parameter_value>*' for parameter '*<parameter_name>*'. This is not a valid escape sequence	The specified parameter value is incorrect since it does not have a valid escape sequence. The supported escape sequences are mentioned in Table 10.
ERROR: Invalid value '*<parameter_value>*' for parameter '*<parameter_name>*'.	Indicates that the syntax of the command is not valid.

The command and printout syntax is shown in the following two examples:

Example: Immediate Action Start

1. Navigate to the desired MO where the action is present:

```
>ManagedElement=<node_name>,TestRootMoc=1,CallableThin
g=1
```

2. Trigger the desired action:

```
(config-CallableThing=1)>addNumbers --num1 23 --num2 54
77
(config-CallableThing=1)>concatString_defValues
comuser
(config-CallableThing=1)>boolenAdder --flag2 True
false
```

Example: Action Start by Commit

1. Enter Config mode:

```
>configure
```

2. Navigate to the desired MO where the action is present:

```
(config)>ManagedElement=<node_name>,SystemFunctions=1,Fi
leM=1,LogicalFs=1,FileGroup=SysMMimSchemas
```

3. Request the action execution:

```
(config-FileGroup=SysMMimSchemas)>removeFile xyz.txt
true
(config-FileGroup=SysMMimSchemas)>
```

4. Trigger the actual action start:

```
(config-FileGroup=SysMMimSchemas)>commit
```

3.10 Deprecated Actions

The following is the deprecated syntax for action execution:

```
[<path>]<action_name>[<action_ parameter_value>[<action_parame
ter_value>]*]
```

Auto-completion and help text for the previous syntax is not supported.

3.11 Deprecated Options

Options **all** and **verbose** are deprecated, and are replaced by new options **--recursive** or **-r** and **--verbose** or **-v** respectively.

The option configuration is replaced by new command **show-config**. The functionality of new options is the same as the deprecated options.

Auto-completion and help request for the deprecated options is not supported but command execution works.

Example 131 and Example 132 shows deprecated option.

```
>show all <TAB>
>show config?
```

Example 131 Auto-Completion and Help Request of Deprecated Option Is Not Supported

```
>show verbose
ManagedElement=<node_name>
```

Example 132 Command Execution for Deprecated Option

3.12 Pipe Utility Commands

COM supports filtering of CLI and Subshell command output with the help of PIPE extension command modules. Pipe symbol "|" is used to indicate that the output of CLI and Subshell commands is sent as input for pipe extension commands.

3.12.1 Filter Command

The filter command is a pipe extension command to filter the output of any command or action. There are optional parameters supported by the filter command, as shown in Example 133.

```
<command/Action> | filter [-i | --ignore] [-v | --invert]
[{-A | --after} <value>] [{-B | --before} <value>]
<pattern>
```

Example 133 Filter Command Syntax

The filter command parameter are listed in Table 30.

Table 30 Filter Command Parameters

Parameter	Description
i or **--ignore**	Ignore case distinctions in both the pattern and the input.
v or **--invert**	Invert the sense of matching, to select non-matching lines.
A or **--after**	Print number of lines of trailing context after matching lines.
B or **--before**	Print number of lines of leading context before matching lines.
<value>	The value for number lines to be printed after leading/trailing context.
<pattern>	An input pattern is a regular expression used for filtering the output of a command/action. It can have alphanumeric characters and special characters supported by POSIX regular expressions.

Examples of filter command are shown in Example 134, Example 135, Example 136, Example 137, and Example 138.

```
(config-SystemFunctions=1)>show | filter m
SystemFunctions=1
  Fm=1
  Pm=1
```

Example 134 Filter Output of Show Command Matching a Pattern

```
(config-SystemFunctions=1)>show | filter -i s
SystemFunctions=1
  SecM=1
  SysM=1
```

Example 135 Filter Output with Case Insensitive Match

```
(config-SystemFunctions=1)>show | filter --after 2 Fm
  Fm=1
  SecM=1
  SysM=1
```

Example 136 Filter Output with --after Option

```
(config-SystemFunctions=1)>show | filter -A 3 Fil |
filter -B 10 --ignore S
  FileM=1
  Fm=1
  SecM=1
  SysM=1
```

Example 137 Filter the Output of Another Filter Command

```
(config-TestRootMoc=1)>userLabel="@#$%^&*(+?>./=}["
(config-TestRootMoc=1)>show
TestRootMoc=1
  userLabel="@#$%^&*(+?>./=}["
  BasicThing=1
  CallableThing=1
  CrazyThing=1
  HiddenAttrThing=1
  XThing=1
  YThing=1
(config-TestRootMoc=1)>show | filter \@
  userLabel="@#$%^&*(+?>./=}["
(config-TestRootMoc=1)>show | filter \#
  userLabel="@#$%^&*(+?>./=}["
(config-TestRootMoc=1)>show | filter \$
  userLabel="@#$%^&*(+?>./=}["
(config-TestRootMoc=1)>show | filter \%
  userLabel="@#$%^&*(+?>./=}["
(config-TestRootMoc=1)>show | filter \^
  userLabel="@#$%^&*(+?>./=}["
```

Example 138 Filter Command with Special Characters in Pattern

3.13 Static Help in CLI

Static help for any CLI element can be obtained by command **help**. This command can be executed in both Exec and Config modes in CLI. Along with the help provided on the CLI elements, the CLI state diagram is also displayed to the user when help command is executed.

The output of command **help** has the CLI commands classified based on their purpose of use and operation. The commands are classified under categories; GENERAL, NAVIGATION, CONFIGURATION, and TRANSACTION.

Command **help** also provides information on the current mode, displays a CLI state machine diagram, see Figure 1, and gives information on the various key bindings in CLI.

Syntax: **>help**

3.14 CLI Commands Limitations

The COM CLI command functionality is provided with the following limitations:

- Owing to the modeling constraint, do not use the `show-config` output without `--verbose|-v` option, to copy MOs that contain mandatory keyless structs.
- If there are sequences of strings, or sequences of structs with string key attributes, and the "@" character is displayed as the first character in these strings, the commands used to address these strings or structs must be performed using quoted strings in all such cases.

3.15 Copy and Paste Configuration Data

CLI supports pasting of large lines of valid configuration data into the CLISS terminal. There is no limitation on the number of configuration lines pasted. To paste configuration data, copy a valid configuration from the `show-config` command output and paste it into the CLISS terminal.

To make it possible to copy data from one node to another with a different name, the CLI automatically corrects the entered node names in all commands (`ManagedElement` ID).

Note: If any of the pasted input commands result in errors, the rest of the pasted input cannot be successfully accepted. Also this feature cannot work as expected on some environments based on the terminal settings or the CPU speed.

3.16 Display CLI Version

Command **`version`** displays the ECLI version. For more information, see Section 2.2 CLI Version on page 7.

```
>version
Ericsson CLI 1.2.0
```

Example 139 Display ECLI Version

3.17 Change Password Command

`change` is an optional command. It is available only if the ME supports changing user password through the CLI.

The password of the logged on user can be changed using command **`passwd`**. This command uses the built-in password changing utility of the ME, which means that the exact behavior depends on the ME type.

This command is only available in root position.

```
>passwd
Changing password for cliuser.
Old Password:
New Password:
Reenter New Password:
Password changed.
>
```

Example 140 Password Command Execution Successful Change of Password

```
>passwd
Changing password for cliuser.
Old Password:
passwd: Authentication failure
>passwd
Changing password for cliuser.
Old Password:
New Password:
Bad password: it is based on a dictionary word
New Password:
Bad password: too simple
New Password:
Bad password: too similar
passwd: Have exhausted maximum number of retries for service
>
```

Example 141 Error Message

4 Terminal Properties

This section describes the terminal properties.

4.1 Terminal Types

Some aspects of how the CLI interacts with the attached terminal or terminal emulator can be controlled by setting the terminal type. The CLI recognizes all terminal types supported by the operating system of the management system and also the following special terminal types:

dumb — The CLI acts as if there is no terminal attached, but instead a script being fed to it on `stdin`. No prompts are printed and the input command lines are not echoed back. Only the command output is printed.

ossrc-eam — The CLI acts as if the attached terminal is a `vt100`, with the following exception:

A `vt100`, as well as terminals and terminal emulators compatible with it, do not show the cursor when an input line reaches the end of the terminal width. Therefore the CLI for most terminal types echoes an extra `space` followed by a `backspace` as an input line reaches this length. For terminal type `ossrc-eam`, these extra characters are not echoed.

The terminal type is set by assigning a value to the `TERM` environment variable in the environment where `cliss` starts. The SSH protocol allows `TERM` to be set, but the methods depend on the client being used.

4.2 Default Key Bindings

Apart from the standard line edition keys such as **Left arrow**, **Right arrow**, **Backspace**, and **Delete**, the CLI supports several of key combinations that can be used for editing the commands.

Support for key bindings is terminal type-specific and the terminal settings override any key or key combination.

This section provides examples on key combinations.

The CLI is aligned with Libtecla library, which defines each key binding as follows:

- **<CTRL>** Press the **Ctrl** key and enter the next key simultaneously.

- **<ESC>** Press the **Esc** key on the terminal followed by the next key, or press the **Meta** key simultaneously with the next key.

4.2.1 Move Cursor

The key combinations for moving the cursor are shown in Table 31.

Table 31 Move Cursor

Action	Key Combination
Move the cursor back one character	**Ctrl+B** or **Left arrow**
Move the cursor back one word	**Esc+B** or **Alt+B**
Move the cursor forward one character	**Ctrl+F** or **Right arrow**
Move the cursor forward one word	**Esc+F** or **Alt+F**
Move the cursor to the beginning of the command line	**Ctrl+A**
Move the cursor to the end of the command line	**Ctrl+E**

4.2.2 Delete Characters

The key combinations for deleting text are shown in Table 32.

Table 32 Delete Characters

Action	Key Combination
Delete the character before the cursor	**Ctrl+H**
Delete or backspace delete the character following the cursor When the cursor is within the line, it deletes the cursor character. When began at the end of the line, it displays all possible completions then redisplays the line. When started on an empty line, it terminates the session.	**Ctrl+D**
Delete all characters from the cursor to the end of the line	**Ctrl+K**
Delete the whole line	**Ctrl+U**
Delete the word before the cursor	**Esc+Backspace** or **Alt+Backspace**
Delete the characters between the last mark that was set, and the cursor.	**Ctrl+W**
Delete the word after the cursor	**Esc+D** or **Alt+D**

4.2.3 Insert Recently Deleted Text

The key combination for inserting text is shown in Table 33.

Table 33 Insert Recently Deleted Text

Action	Key Combination
Insert the most recently deleted text at the cursor	**Ctrl+Y**

4.2.4 Display Previous Command Lines

The key combinations for recalling previous commands are shown in Table 34.

Table 34 Display Previous Command Lines

Action	Key Combination
Scroll backward through the command history	**Ctrl+P** or **Up arrow**
Scroll forward through the command history	**Ctrl+N** or **Down arrow**

4.2.5 Capitalization

The key combinations for changing capitalization are shown in Table 35.

Table 35 Capitalization

Action	Key Combination
Capitalize the word at the cursor, that is, make the first character uppercase and the rest of the word lower case	**Esc+C**
Convert all the characters of the word which follows the cursor to lower case	**Esc+L**
Convert all the characters of the word which follows the cursor to upper case	**Esc+U**

4.2.6 Special Actions

Additional key combinations are shown in Table 36.

Table 36 Special Actions

Action	Key Combination
End a command or clear line	**Ctrl+C**

Action	Key Combination
Arrange for the next character to be treated as a normal character. This allows control characters to be entered.	**Ctrl+V**
Clear the terminal, then redisplay the current line	**Ctrl+L**
Swap the character under the cursor with the character just before the cursor.	**Ctrl+T**
Redisplay the line	**Ctrl+R**