Nokia Service Router Linux

SYSTEM MANAGEMENT GUIDE RELEASE 21.11

3HE 17900 AAAA TQZZA
Issue 1
December 2021

Table of contents

1 About this guide

This document describes the interfaces used with the Nokia Service Router Linux (SR Linux). These include:

- CLI
- gNMI
- JSON

This document also provides an overview to CLI plug-ins and details Nokia defined general and operation commands, and show commands.

This document is intended for network technicians, administrators, operators, service providers, and others who need to understand how to interface with the SR Linux.

Note:

This manual covers the current release and may also contain some content that will be released in later maintenance loads. See the for *SR Linux Release Notes* information about features supported in each load.

1.1 What's new

There have been no updates in this document since it was last released.

1.2 Precautionary and information messages

The following are information symbols used in the documentation.

DANGER: Danger warns that the described activity or situation may result in serious personal injury or death. An electric shock hazard could exist. Before you begin work on this equipment, be aware of hazards involving electrical circuitry, be familiar with networking environments, and implement accident prevention procedures.

WARNING: Warning indicates that the described activity or situation may, or will, cause equipment damage, serious performance problems, or loss of data.

Caution: Caution indicates that the described activity or situation may reduce your component or system performance.

Note: Note provides additional operational information.

Tip: Tip provides suggestions for use or best practices.

1.3 Conventions

Nokia SR Linux documentation uses the following command conventions.

- **Bold** type indicates a command that the user must enter.
- Input and output examples are displayed in `Courier` text.
- An open right-angle bracket indicates a progression of menu choices or simple command sequence (often selected from a user interface). Example: **start** > **connect to**.
- Angle brackets (< >) indicate an item that is not used verbatim. For example, for the command **show ethernet <***name***>**, *name* should be replaced with the name of the interface.
- A vertical bar (|) indicates a mutually exclusive argument.
- Square brackets ([ ]) indicate optional elements.
- Braces ({ }) indicate a required choice. When braces are contained within square brackets, they indicate a required choice within an optional element.
- *Italic* type indicates a variable.

Generic IP addresses are used in examples. Replace these with the appropriate IP addresses used in the system.

2 CLI interface

The CLI is an interface for configuring, monitoring, and maintaining the SR Linux. This chapter describes basic features of the CLI and how to use them.

2.1 CLI access and use

The following sections describe how to access and use the CLI.

2.1.1 Accessing the CLI

After the SR Linux device is initialized, you can access the CLI using a console or SSH connection. See the SR Linux hardware documentation for information about establishing a console connection and enabling and connecting to an SSH server.

Use the following command to connect to the SR Linux and open the CLI using SSH:

ssh admin@*<IP Address>*

Example:

```
$ ssh admin@172.16.0.3
Hello admin,
Welcome to the srlinux CLI.
Type 'help' (and press <ENTER>) if you need any help using this.
--{ running }--[ ]--
```

2.1.2 Using the CLI help functions

The CLI help functions (**?** and **help**) can assist in understanding command usage and indicate which configuration mode you are in.

Example: ?

Enter a question mark (**?**) after a command to display the command usage. For example, entering a question mark after the command **network-instance**, shows its usage.

```
# network-instance ?
usage: network-instance <name>
Network instances configured on the local system
Positional arguments:
  name              [string] A unique name identifying the network instance
```

Example: help

Enter **help** at the top level to show the current configuration mode and details on other configuration modes.

```
# help
```

```
--------------------------------------------------------------------------------
You are now in the running mode.
Here you can navigate and query the running configuration.
This configuration has been validated, committed and send to the applications.
There are multiple modes you can enter while using the CLI.
Each mode offers its own set of capabilities:
    - running mode: Allows traversing and inspecting of the running configuration.
    - state mode: Allows traversing and inspecting of the state.
    - candidate mode: Allows editing and inspecting of the configuration.
    - show mode: Allows traversing and executing of custom show routines.
To switch mode use 'enter <mode-name>', e.g. 'enter candidate'
To navigate around, you can simply type the node name to enter a context, while
'exit [all]' is used to navigate up.
'{' and '}' are an alternative way to enter and leave contexts.
'?' can be used to see all possible commands, or alternatively,
'<TAB>' can be used to trigger auto-completion.
'tree' displays the tree of possible nodes you can enter.
--------------------------------------------------------------------------------
--{ * running }--[  ]--
```

Related topics

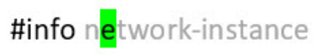
Configuration modes

2.1.3 Using the CLI auto-complete function

To reduce keystrokes or aid in remembering a command name, use the CLI auto-complete function.

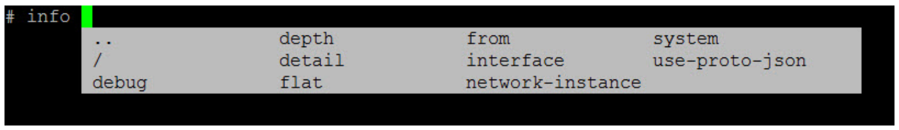
Enter a tab at any mode or level to auto-complete the next command level. If multiple options are available, a popup appears.

When a command is partially entered, the remainder of the command appears ahead of the prompt in lighter text. Press the Tab key to complete the command.



```
#info network-instance
```

When the Tab key is pressed and there are multiple options, the options are shown:



```
# info
       ..                depth            from             system
       /                 detail           interface        use-proto-json
       debug             flat             network-instance
```

2.1.4 CLI keyboard shortcuts

Use shortcuts to move the cursor on the command line, complete commands, and recall commands previously entered. Shortcuts can also make syntax correction easier. The following table lists common shortcuts.

Table 1: CLI keyboard shortcuts

Task	Keystroke
Move cursor to the beginning of the line	Ctrl+A

Task	Keystroke
Move cursor to the end of the line	Ctrl+E
Move cursor one character to the right	Ctrl+F or Right arrow key
Move cursor one character to the left	Ctrl+B or Left arrow key
Move cursor forward one word	Esc F
Move cursor back one word	Esc B
Transpose the character to the left of the cursor with the character the cursor is placed on	Ctrl+T
Complete a partial command	Enter the first few letters, then press the Tab key
Recall previous entry in the buffer	Page up key

Navigate one level up within a context. For example:
 `--{running}--[interface ethernet-1/1 subinterface 1]--`
 `# exit`
 `--{running}--[interface ethernet-1/1]--`
	Type: **exit**

Return to the root context. For example:
 `--{running}--[interface ethernet-1/1 subinterface 1]--`
 `# exit all`
 `--{running}--[ ]--`
	Type: **exit all**

2.1.5 Closing the CLI

Close the CLI using one of the following methods:

- Press **Ctrl+D**.
- Enter the **quit** command at the CLI prompt.

2.2 Configuration modes

Configuration modes define how the system is running when transactions are performed. Supported modes are the following:

- **Candidate** – Use this mode to modify a configuration. Modifications are not applied to the running system until a **commit** command is issued. When committed, the changes are copied to the running configuration and become active.

  There are different types of configuration candidates. See Configuration candidates.

- **Running** – Use this mode to display the currently running or active configuration. Configurations cannot be edited in this mode.
- **State** – Use this mode to display the configuration and operational states. The state mode displays more information than show mode.
- **Show** – Use this mode to display configured features and operational states. The show mode displays less information than state mode.

2.2.1 Configuration candidates

You can modify the candidate configuration in different modes:

- Exclusive
- Shared
- Private
- Name

2.2.1.1 Exclusive mode

When entering candidate mode, if you specify the **exclusive** keyword, it locks out other users from making changes to the candidate configuration.

You can enter candidate exclusive mode only under the following conditions:

- The current shared candidate configuration has not been modified.
- There are no other users in candidate shared mode.
- No other users have entered candidate exclusive mode.

2.2.1.2 Shared mode

By default, the candidate configuration is in shared mode. This allows multiple users to modify the candidate configuration concurrently. When the configuration is committed, the changes from all of the users are applied.

A default candidate is defined per user (see Name mode).

Use caution when allowing multiple users access to the candidate configuration at the same time. If one user commits the configuration, that commits the changes made by any other users who have modified the current candidate configuration.

2.2.1.3 Private mode

A private candidate allows multiple users to modify a configuration; however when a user commits their changes, only the changes from that user are committed. When a private candidate is created, private datastores are created and a snapshot is taken from the running database to create a baseline.

When starting a private candidate, a default candidate is defined per user with the name 'private-<username>' unless a unique name is defined (see Name mode).

2.2.1.4 Name mode

Candidate types support an optional name. This allows multiple users to share environments.

When a candidate is created with a name specified, a new entry is created in the /system/configuration/candidate[] list. Other users can enter the same candidate (if the candidate type is shared) using this name.

If no name is specified, then the name **default** is used. This means the global shared candidate can be accessed by entering **enter candidate name default** or just **enter candidate**, For private candidates the name **default** is not used (because private candidates share a list with shared candidates). Instead, private candidates are created with the name **private-**<*username*>.

Named candidates are automatically deleted when there are no active sessions present and they are empty, or after 7 days of no session activity. This 7-day default is configurable in the **/system/configuration/idle-timeout** field.

2.2.2 Setting the configuration mode

After logging in to the CLI, you are initially placed in running mode. The following table describes the commands to change between modes.

Table 2: Commands to change configuration mode

To enter this mode:	Type this command:
Candidate shared	**enter candidate**
Candidate mode for named shared candidate	**enter candidate name** <*name*>
Candidate private	**enter candidate private**
Candidate mode for named private candidate	**enter candidate private name** <*name*>
Candidate exclusive	**enter candidate exclusive**
Exclusive mode for named candidate	**enter candidate exclusive name** <*name*>
Running	**enter running**
State	**enter state**
Show	**enter show**

Example: Change from running to shared mode

To change from running to a shared candidate mode (using the default)’:

```
--{ running }--[  ]--
# enter candidate
--{ * candidate shared default}--[  ]--
```

The asterisk (*) next to the mode name indicates that the candidate configuration has changes that have not yet been committed.

Example: Switch between shared and candidate exclusive modes

To switch between candidate shared and candidate exclusive modes, you must first switch to a different configuration mode (for example, running mode) before entering candidate shared or exclusive mode. For example:

```
--{ running }--[  ]--
# enter candidate exclusive
--{ candidate exclusive }--[  ]--
Are you sure? (y/[n]):
# enter running
--{ running }--[  ]--
# enter candidate
--{ candidate shared default}--[  ]--
```

Example: Enter candidate mode for named candidate

To enter candidate mode for a named configuration candidate, you specify the name of the configuration candidate. For example:

```
--{ running }--[  ]--
# enter candidate name cand1
--{ candidate shared cand1}--[  ]--
```

2.2.3 Managing configuration conflicts

About this task

When a user enters candidate mode, the system creates two copies of the running datastore: one is modifiable by the user, and the other serves as a baseline. The modifiable datastore and the baseline datastore are collectively known as a configuration candidate.

You can use the **baseline** command to assist in managing conflicts. It uses the following arguments:

- **baseline udpate** - Performs an update of the complete baseline datastore, pulling in any changes that occurred in the running datastore since the baseline snapshot was taken.
- **baseline diff**- Shows baseline configuration changes with options to refine by area.
- **baseline check** - Performs a dry-run baseline check, and if conflicts are detected, an informational or warning message is generated.

2.2.4 Committing a configuration in candidate mode

About this task

Changes made during a configuration modification session do not take effect until a **commit** command is issued. Use the **commit** command in candidate mode only.

Procedure

Step 1. Enter candidate mode:

```
# enter candidate
```

Step 2. Enter configuration commands.

Step 3. Enter the **commit** command when with the required option.

Table 3: commit command options

Option	Action	Permitted additional arguments
commit now	Apply the changes, exit candidate mode, and enter running mode.	NA
commit stay	Apply the changes and then remain in candidate mode.	**commit stay [save] [comment] [confirmed]**
commit save	Apply the changes and automatically save the commit to the startup configuration. Can be used other arguments except **now** (for example, **commit stay save**).	**commit [stay] [checkpoint] save [confirmed] [comment]**
commit checkpoint	Apply the changes and cause an automatic checkpoint after the commit succeeds.	**commit [stay] [now] checkpoint [save] [confirmed]**
commit validate	Verify that a propose configuration change passes a management server validation.	NA
commit comment *<comment>*	Use with other keywords (except **validate**) to add a user comment (for example, **commit stay comment** *<comment>* where *<comment>* is a quoted string, 1-255 characters.	**commit [stay] [save] [checkpoint] [confirmed] comment**
commit confirmed **commit confirmed [timeout]** **commit confirmed [accept | reject]**	Apply the changes, but requires an explicit confirmation to become permanent. If the explicit confirmation is not issued within a specified time period, all changes are automatically reverted. The timeout period default is 600 seconds (10 mins.), or can be provisioned with a value of 1-86400 sec.). The timeout parameter cannot be used with the accept or reject parameter. Before the timer expires, the accept parameter explicitly confirms and applies the changes. With no timer running, the reject parameter explicitly rejects the changes.	**commit [checkpoint] [save] [stay] [comment] confirmed**

Example: commit stay and commit confirmed

This example shows the **commit stay** option:

```
# enter candidate
--{ candidate shared default}--[  ]--
# interface ethernet-1/1 subinterface 1
--{ * candidate shared default}--[ interface ethernet-1/1 subinterface 1 ]--

# commit stay
All changes have been committed. Starting new transaction.
```

```
--{ candidate shared default}--[ interface ethernet-1/1 subinterface 1 ]--
```

This example shows the **commit confirmed** option with a custom timeout followed by an accept action.

```
--{ * candidate shared default}--[  ]--
# commit confirmed timeout 86400
Commit confirmed (automatic rollback in a day)
All changes have been committed. Leaving candidate mode.
--{ running }--[  ]--

# commit confirmed accept
Info: Commit confirmed, automatic rollback cancelled
--{ candidate shared default}--[  ]--
#
```

2.2.5 Deleting configurations

Use the **delete** command to delete configurations while in candidate mode.

Example: Delete configuration

The following example displays the system banner configuration, deletes the configured banner, then displays the resulting system banner configuration:

```
--{ candidate shared default}--[  ]--
# info system banner
    system {
        banner {
            login-banner "Welcome to SRLinux!"
        }
    }
--{ candidate shared default}--[  ]--
# delete system banner
--{ candidate shared default}--[  ]--
# info system banner
    system {
        banner {
        }
    }
```

2.2.6 Annotating the configuration

To aid in reading a configuration, you can add comments or descriptive annotations. The annotations are indicated by `!!!` in displayed output.

You can enter a comment either directly from the command line or by navigating to a CLI context and entering the comment in annotate mode.

Example: Add a comment

The following example adds a comment to an ACL configuration. If there is already a comment in the configuration, the new comment is appended to the existing comment.

```
--{ candidate shared default}--[  ]--
# acl ipv4-filter ip_tcp !! "Filter TCP traffic"
```

To replace the existing comment, use **!!!** instead of **!!** in the command.

Example: Add a comment in annotate mode

The following example adds the same comment to the ACL by navigating to the context for the ACL and entering the comment in annotate mode:

```
--{ * candidate shared default}--[  ]--
# acl ipv4-filter ip_tcp
--{ * candidate shared default}--[ acl ipv4-filter ip_tcp ]--
# annotate
Press [Meta+enter] or [Esc] followed by [Enter] to finish
-> Filter TCP traffic
```

You can enter multiple lines in annotate mode. To exit annotate mode, press Esc, then the Enter key.

In CLI output, the comment is displayed in the context it was entered. For example:

```
--{ running }--[  ]--
# info acl
    acl {
        ipv4-filter ip_tcp {
            !!! Filter TCP traffic
            entry 100 {
                action {
                    drop {
                        log true
                    }
                }
            }
            entry 110 {
                action {
                    accept {
                        log true
                    }
                }
            }
        }
    }
```

To remove a comment, enter annotate mode for the context and press Esc then Enter without entering any text.

2.2.7 Discarding a configuration in candidate mode

You can discard previously applied configurations with the **discard** command. Use the **discard** command in candidate mode only.

- To discard the changes and remain in candidate mode with a new candidate session, enter **discard stay**.
- To discard the changes, exit candidate mode, and enter running mode, enter **discard now**.

Example: Discard changes and remain in candidate mode

```
All changes have been committed. Starting new transaction.

--{ candidate shared }--[ interface ethernet-1/1 subinterface 1 ]--
# discard stay
--{ candidate shared }--[ interface ethernet-1/1 subinterface 1 ]--
```

2.3 Administrative commands

The common administrative CLI commands in this section can help you understand a current configuration and perform routine configuration tasks.

2.3.1 Pinging a destination IP address

Use the **ping** (IPv4) or **ping6** (IPv6) command to contact an IP address. Use this command in any mode.

Example: ping for IPV4

```
--{ running }--[  ]--
# ping 192.168.1.1 network-instance default
Pinging 192.168.1.1 in srbase-default
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.027 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.032 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.030 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 6165ms
rtt min/avg/max/mdev = 0.027/0.030/0.033/0.005 ms
```

2.3.2 Tracing the path to a destination

To display the path a packet takes to a destination, use the **traceroute** (IPv4) or **traceroute6** (IPv6) command.

Example: traceroute for IPv4

```
--{ running }--[  ]--
# traceroute 1.1.1.1  network-instance mgmt
Using network instance srbase-mgmt
traceroute to 1.1.1.1 (1.1.1.1), 30 hops max, 60 byte packets
 1  172.18.18.1 (172.18.18.1)  1.268 ms  1.260 ms  1.256 ms
 2  172.21.40.1 (172.21.40.1)  1.253 ms  1.848 ms  1.851 ms
 3  172.22.35.230 (172.22.35.230)  1.835 ms  1.834 ms  1.828 ms
 4  66.201.62.1 (66.201.62.1)  3.222 ms  3.222 ms  3.216 ms
 5  66.201.34.17 (66.201.34.17)  5.474 ms  5.475 ms  5.480 ms
 6  * * *
 7  206.81.81.10 (206.81.81.10)  32.577 ms  32.542 ms  32.400 ms
 8  1.1.1.1 (1.1.1.1)  22.627 ms  22.637 ms  22.638 ms
```

To trace the route using TCP SYN packets instead of UDP or ICMP echo packets, use the **tcptraceroute** command.

2.3.3 Configuring the network-instance environment variable

When you enter administrative commands such as **ping** and **traceroute**, you can specify the network-instance to be used with the command. If you do not specify a network-instance, then the network-instance configured in the network-instance environment variable is used.

If you do not specify a network-instance, or the network-instance cannot be inferred from the CLI context, then the <*base*> network instance is used when no network-instance is specified in a **ping** or **traceroute** command; for example:

Example: ping with no network-instance specified

```
--{ running }--[  ]--
# ping 192.168.1.1
Using network instance <base>
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.027 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.032 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.030 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 6165ms
rtt min/avg/max/mdev = 0.027/0.030/0.033/0.005 ms
```

Example: Configure the environment network-instance

To configure the network-instance environment variable, use the **environment network-instance** command; for example:

```
--{ candidate shared default }--[  ]--
# environment network-instance red
```

Example: ping using implied network-instance

In this example, the network-instance environment variable is set to "red", so network instance red is implied when the **ping** or **traceroute** command is entered without a network-instance name; for example:

```
--{ running }--[  ]--
# ping 192.168.1.1
Using network instance srbase-red
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.027 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.032 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.030 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 6165ms
rtt min/avg/max/mdev = 0.027/0.030/0.033/0.005 ms
```

2.3.4 Using bash mode

From within the CLI, use the **bash** command to enter the bash shell. Use this command in any mode. To exit the bash shell and return to the CLI, enter **exit**.

Example: Using bash shell

```
--{ running }--[  ]--
# bash
[root@3-node_srlinux-A /]# ls
anaconda-post.log  dev           etc   lib    media  opt   root  sbin  sys   tmp  var
bin                entrypoint.sh home  lib64  mnt    proc  run   srv   tini  usr
[root@3-node_srlinux-A /]# exit
logout
--{ running }--[  ]--
```

```
#
```

Example: Enter Linux commands using bash

You can use the **bash** command to enter Linux commands directly from the SR Linux CLI prompt. For example:

```
--{ running }--[  ]--
# bash cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.18.5     3-node-srlinux-A
### SRLINUX - ANYTHING MODIFIED BEYOND THIS POINT WILL BE OVERWRITTEN ###
127.0.0.1 3-node-srlinux-A
### SRLINUX FOOTER ###
```

2.3.5 Setting commands to execute periodically

To set a command to execute periodically, use the **watch** command. The **watch** command can be used with any valid command with the exception of interactive commands (for example, **ping**, **monitor**, **packet-trace**, **bash**, and bash oneliners such as **bash cat /etc/hosts**). An error message is generated if an interactive command is used. Output redirection is supported.

When used, the first page of output displays. The command then re-executes every 2 seconds by default.

The watch command uses the following format:

watch <*arguments*> <*command to watch*>

All arguments must precede the command being watched. Arguments for the **watch** command are:

Arguments	Definition
interval	Sets an interval for the command to re-execute. Range: 1 - 3600 seconds. Default: 2 seconds
differences	For output that is actively changing, highlights the differences between the previous and current execution
cumulative-differences	For output that is actively changing, highlights the differences between the first execution and the current execution
no-colors	When used with differences or cumulative-differences, markers are used to show differences (verses colors)
no-limit	Allows the full output of each execution to display (verses limiting to the height of the terminal)

Arguments	Definition
no-paging	Provides continuous output (verses redrawing the terminal with each execution)
no-title	Turns off the header that shows the interval, command, number of executions that have occurred, and the time

The command being watched does not require double quotes. In addition, the auto complete function can be used to complete a valid command.

To exit the **watch** command, enter Ctrl-C.

Example: Watch interface statistics

In the following example, the watch command is used to show interface statistics (showing in-octets and out-octets in table format). The interval is modified to 10 seconds.

```
# watch interval 10 info from state interface * statistics | as table |filter fields in-octets
 out-octets

Every 10.0s: info from state interface * statistics | as table | filter fields in-octets out-
octets (Executions 2,
Thu 04:47:34PM)
+---------------------+---------------------+---------------------+
|      Interface      |      In-octets      |      Out-octets     |
+=====================+=====================+=====================+
| ethernet-1/1        |              812755 |              812626 |
| ethernet-1/2        |              811411 |              810362 |
| ethernet-1/3        |                     |                     |
| ethernet-1/4        |                     |                     |
| ethernet-1/5        |                     |                     |
| ethernet-1/6        |                     |                     |
| ethernet-1/7        |                     |                     |
| ethernet-1/8        |                     |                     |
| ethernet-1/9        |                     |                     |
| ethernet-1/10       |                     |                     |
| ethernet-1/11       |                     |                     |
| ethernet-1/12       |                     |                     |
| ethernet-1/13       |                     |                     |
| ethernet-1/14       |                     |                     |
| ethernet-1/15       |                     |                     |
```

Example: Show differences in interface statistics

In the following example, the watch command is used to show differences in interface statistics (with in-octets and out-octets in table format). In addition, the no-color argument is used to show the differences between previous and current execution with markers verses color indicators.

```
# watch differences no-colors info from state interface * statistics | as table |filter fields
 in-octets out-octets

Every 10.0s: info from state interface * statistics | as table | filter fields in-octets out-
octets (Executions 4,
 Thu 04:54:48PM)
  +---------------------+---------------------+---------------------+
  |      Interface      |      In-octets      |      Out-octets     |
  +=====================+=====================+=====================+
- | ethernet-1/1        |              817889 |              817760 |
?                                        --                     ^ ^
```

```
+ | ethernet-1/1        |              817975 |              817865 |
?                                          ++                   ^ ^
- | ethernet-1/2        |              816328 |              815088 |
?                                         ^^                    ^^
+ | ethernet-1/2        |              816585 |              815278 |
?                                         ^ +                   ^^
  | ethernet-1/3        |                     |                     |
  | ethernet-1/4        |                     |                     |
  | ethernet-1/5        |                     |                     |
  | ethernet-1/6        |                     |                     |
  | ethernet-1/7        |                     |                     |
  | ethernet-1/8        |                     |                     |
  | ethernet-1/9        |                     |                     |
```

2.3.6 Using the diff command

Use the **diff** command to get a comparison of configuration changes. Optional arguments can be used to indicate the source and destination datastore. The following use rules apply:

- If no arguments are specified, the diff is performed from the candidate to the baseline of the candidate.
- If a single argument is specified, the diff is performed from the current candidate to the specified candidate.
- If two arguments are specified, the first is treated as the source, and the second as the destination.

Global arguments include: baseline, candidate, checkpoint, factory, file, from, rescue, running, and startup.

The **diff** command can be used outside of candidate mode, but only if used with arguments.

Example: Basic diff command with no arguments

The following shows a basic **diff** command without arguments. In this example, the description and admin state of an interface are changed and the differences shown:

```
--{ candidate shared default }--[  ]--
# interface ethernet-1/1 admin-state disable
--{ * candidate shared default }--[  ]--
# interface ethernet-1/2 description "updated"
--{ * candidate shared default }--[  ]--
# diff
      interface ethernet-1/1 {
+         admin-state disable
      }
+     interface ethernet-1/2 {
+         description updated
+     }
```

Example: diff command with single argument

The following shows a diff with a single argument. In this example, the comparison occurs to and from a factory configuration.

```
--{ candidate shared default }--[  ]--
# diff factory
      acl {
-     }
+         ipv4-filter allow_sip_dip {
+             subinterface-specific output-only
+             entry 10 {
+                 action {
```

```
+                     accept {
+                     }
+                 }
+                 match {
+                     destination-ip {
+                         prefix 100.1.2.2/32
+                     }
+                     source-ip {
+                         prefix 100.1.1.1/32
+                     }
+                 }
+             }
+             entry 100 {
+                 action {
+                     accept {
+                     }
+                 }
+             }
```

Example: diff between two checkpoints

The following shows an example where the comparison occurs between two checkpoints.

```
--{ candidate shared default }--[  ]--
# save checkpoint name current
/system:
    Generated checkpoint '/etc/opt/srlinux/checkpoint/checkpoint-0.json' with name 'current'
 and comment ''

--{ candidate shared default }--[  ]--
# interface ethernet-1/1 description "changed-in-next-checkpoint"
--{ candidate shared default }--[  ]--
# commit stay
All changes have been committed. Starting new transaction.
--{ candidate shared default }--[  ]--
# save checkpoint name newer
/system:
    Generated checkpoint '/etc/opt/srlinux/checkpoint/checkpoint-0.json' with name 'newer' and
 comment ''

--{ candidate shared default }--[  ]--
# diff checkpoint newer checkpoint current
      interface ethernet-1/1 {
-         description changed-in-next-checkpoint
+         description dut1-dut2-1
      }
```

2.3.7 Using the copy command

Use the **copy** command to copy a specific context to another context. For example, you can use this command to copy a container or any configuration hierarchy and give it another name (such as an interface).

The **copy** command merges the existing context instead of replacing it. If a replace is required, you can delete the target node first. When using the **copy** command, matching fields populate to the destination node; fields that do not match are ignored. The use of ranges for both source and destination is permitted.

Use this command in candidate mode. Nokia also recommends that you use the **diff** command to verify changes when using complex copy operations such as multiple ranges in both the source and destination.

Example: Copy context of one interface to another

The following shows the context of two existing interfaces (ethernet-1/1 and ethernet-2/1):

```
--{ candidate shared default }--[  ]--
# info interface ethernet-1/1
    interface ethernet-1/1 {
        description dut1-dut2-1
        ethernet {
            aggregate-id lag1
        }
    }
--{ candidate shared default }--[  ]--
# info interface ethernet-2/1
    interface ethernet-2/1 {
        description dut2-dut4-1
        subinterface 1 {
            ipv4 {
                address 1.2.4.2/24 {
                }
            }
            ipv6 {
                address 3ffe:0:2:4::2/64 {
                }
            }
        }
    }
```

To copy the context of ethernet-1/1 to ethernet-2/1, enter the following:

```
--{ candidate shared default }--[  ]--
# copy interface ethernet-1/1 to interface ethernet-2/1
--{ * candidate shared default }--[  ]--
# diff
      interface ethernet-2/1 {
-         description dut2-dut4-1
+         description dut1-dut2-1
          ethernet {
+             aggregate-id lag1
          }
      }
```

Example: Copy matching fields only

The following shows an example where only some fields match. Fields that do not match are ignored:

```
--{ * candidate shared default }--[  ]--
# info acl cpm-filter ipv4-filter entry 10
    acl {
        cpm-filter {
            ipv4-filter {
                entry 10 {
                    description "cpm-filter description"
                    action {
                        accept {
                        }
                    }
                    match {
                        protocol icmp
                        icmp {
                            type dest-unreachable
                            code [
                                0
                                1
                                2
                                3
                                4
                                13
                            ]
                        }
                    }
                }
            }
        }
    }
--{ * candidate shared default }--[  ]--
# copy acl cpm-filter ipv4-filter entry 10 to interface ethernet-2/1
--{ * candidate shared default }--[  ]--
A:dev-dut2# diff
      interface ethernet-2/1 {
-         description dut2-dut4-1
+         description "cpm-filter description"
      }
```

Example: Copy with ranges

The following shows an example that uses ranges. The system expands both the source and destination to determine which set has the largest number of keys, and copies 1-to-1 until the larger of the two ranges finishes.

```
--{ * candidate shared default }--[  ]--
# copy interface ethernet-1/{1,2} to interface ethernet-{2,3}/{1..10}
--{ * candidate shared default }--[  ]--
# diff
      interface ethernet-2/1 {
-         description dut2-dut4-1
+         description dut1-dut2-1
+         ethernet {
+             aggregate-id lag1
+         }
      }
      interface ethernet-2/2 {
-         description dut2-dut3-1
+         description "second description"
+         mtu 9000
+     }
+     interface ethernet-2/3 {
+         description dut1-dut2-1
+         ethernet {
+             aggregate-id lag1
+         }
      }

+     interface ethernet-3/1 {
+         description dut1-dut2-1
+         ethernet {
+             aggregate-id lag1
+         }
+     }
+     interface ethernet-3/2 {
+         description "second description"
+         mtu 9000
+     }
+     interface ethernet-3/3 {
+         description dut1-dut2-1
+         ethernet {
+             aggregate-id lag1
+         }
+     }
+     interface ethernet-3/4 {
+         description "second description"
+         mtu 9000
+     }

+     interface ethernet-3/9 {
+         description dut1-dut2-1
+         ethernet {
+             aggregate-id lag1
+         }
+     }
+     interface ethernet-3/10 {
+         description "second description"
+         mtu 9000
+     }
```

2.3.8 Using the replace command

Use the **replace** command to replace a source context with a destination context (including all of its children). Both the source and destination can be text. This command can also be used to create a destination that does not currently exist by creating a new key that uses the source configuration.

The **replace** command has the following usage: **replace** <*original text*> **with** <*replacement text*>

Use this command in candidate mode.

Example: replace network-instance

The following example replaces all instances of the "mgmt" network-instance with a "test" network-instance:

```
--{ candidate shared default }--[  ]--
# replace "network-instance mgmt" with "network-instance test"
--{ * candidate shared default }--[  ]--
# diff
      system {
          gnmi-server {
-             network-instance mgmt {
+             network-instance test {
                  admin-state enable
                  port 50052
                  tls-profile tls-profile-1
              }
          }
          json-rpc-server {
-             network-instance mgmt {
+             network-instance test {
                  http {
                      admin-state enable
                      use-authentication false
                      port 4000
                  }
                  https {
                      admin-state enable
                      port 4001
                      tls-profile tls-profile-1
                  }
              }
          }
          dns {
-             network-instance mgmt
```

```
+              network-instance test
```

Example: replace interface

The following example shows a command line that can be used to replaces all instances of "interface lag3" or "aggregate-id lag3" with "interface lag2" or "aggregate-id lag2":

```
--{ candidate shared default }--[  ]--
# replace "(interface |aggregate-id )lag3" with \1lag2
```

Example: replace with environment alias

The **replace** command can also be used with the **environment alias** command so that multiple agruments can be used. In the following, an alias named ‘ifrename’ is created to replace the name of an interface:

```
--{ candidate shared default }--[  ]--
# environment alias ifrename "replace \"(interface |aggregate-id ){}\b\" with \1{} /"
```

When the alias is created, the alias with the agruments is entered.

```
--{ candidate shared default }--[  ]--
# ifrename lag2 lag9
```

Related topics

Configuring CLI command aliases

2.3.9 Displaying configuration details

Use the **info** command to display the configuration. Entering the **info** command from the root context displays the entire configuration, or the configuration for a specified context. Entering the command from within a context limits the display to the configuration under that context. Use this command in candidate or running mode.

Example: info from root context

To display the entire configuration, enter **info** from the root context:

```
--{ candidate shared default}--[  ]--
# info
<all the configuration is displayed>
--{ candidate }--[  ]--
```

Example: info for specific context

To display the configuration for a specific context, enter **info** and specify the context:

```
--{ candidate shared default}--[  ]--
# info system lldp
    system {
        lldp {
            admin-state enable
            hello-timer 600
            management-address mgmt0.0 {
                type [
                    IPv4
                ]
            }
            interface mgmt0 {
                admin-state disable
            }
        }
    }
--{ candidate }--[  ]--
```

Example: info within specific context

From a context, use the **info** command to display the configuration under that context:

```
--{ candidate shared default}--[  ]--
# system lldp
--{ candidate }--[ system lldp ]--
# info
    admin-state enable
    hello-timer 600
    management-address mgmt0.0 {
        type [
            IPv4
        ]
    }
    interface mgmt0 {
        admin-state disable
    }
--{ candidate }--[ system lldp ]--
```

Example: info within specific context with JSON-formatted output

Use the **as-json** option to display JSON-formatted output:

```
--{ candidate }--[ system lldp ]--
# info | as json
{
  "admin-state": "enable",
  "hello-timer": "600",
  "management-address": [
    {
      "subinterface": "mgmt0.0",
      "type": [
        "IPv4"
      ]
    }
  ],
  "interface": [
    {
      "name": "mgmt0",
      "admin-state": "disable"
    }
  ]
}
```

Example: info detail

Use the **detail** option to display values for all parameters, including those not specifically configured:

```
--{ candidate shared default}--[ system lldp ]--
# info detail
    admin-state enable
    hello-timer 600
    hold-multiplier 4
    management-address mgmt0.0 {
        type [
            IPv4
        ]
    }
    interface mgmt0 {
        admin-state disable
    }
```

Example: info flat

Use the **flat** option to display the output as a series of **set** statements, omitting indentation for any sub-contexts:

```
--{ candidate shared default}--[ system lldp ]--
# info flat
set / system lldp admin-state enable
set / system lldp hello-timer 600
set / system lldp management-address mgmt0.0
set / system lldp management-address mgmt0.0 type [ IPv4 ]
set / system lldp interface mgmt0
set / system lldp interface mgmt0 admin-state disable
```

Example: info depth

Use the **depth** option to display parameters with a specified number of sub-context levels:

```
--{ candidate shared default}--[ system lldp ]--
# info depth 0
    admin-state enable
    hello-timer 600
```

```
--{ candidate shared default}--[ system lldp ]--
# info depth 1
    admin-state enable
    hello-timer 600
    management-address mgmt0.0 {
        type [
            IPv4
        ]
    }
    interface mgmt0 {
        admin-state disable
    }
```

2.3.10 Displaying the command tree hierarchy

Use the **tree** command to display the tree hierarchy for all available nodes you can enter. Entering the **tree** command from the root context displays the entire tree hierarchy. Entering the command from a context limits the display to the nodes under that context. Use this command in candidate or running mode.

Example: tree command

```
--{ candidate shared default}--[ ]--
# tree
<root>
+-- acl
|   +-- ipv4-filter
|   |   +-- description
|   |   +-- subinterface-specific
|   |   +-- statistics-per-entry
|   |   +-- entry
|   |       +-- description
|   |       +-- action
|   |       |   +-- accept
|   |       |   |   +-- log
|   |       |   +-- drop
|   |       |       +-- log
|   |       +-- match
|   |           +-- destination-address
|   |           +-- fragment
|   |           +-- first-fragment
|   |           +-- protocol
|   |           +-- source-address
.
.
.
.
```

2.3.11 Displaying the state of the configuration

To display the state of the configuration, enter the **info from state** command in candidate or running mode, or the **info** command in state mode.

Example: info from state command

To display state information for a specified context from modes that are not state mode:

```
--{ candidate shared default}--[  ]--
# info from state routing-policy policy bgp-export-policy
    routing-policy {
        policy bgp-export-policy {
            statement 999 {
                action {
                    accept {
                    }
                }
            }
        }
    }
--{ candidate }--[  ]--
```

Example: info command from state mode

To display state information for a specified context from state mode:

```
--{ candidate shared shared default}--[  ]--
# enter state
--{ state }--[  ]--
# info routing-policy policy bgp-export-policy
    routing-policy {
        policy bgp-export-policy {
            statement 999 {
                action {
                    accept {
                    }
                }
            }
        }
    }
--{ state }--[  ]--
```

2.3.12 Executing configuration statements from a file

You can execute configuration statements from a source file consisting of **set** statements such as those generated by the **info flat** command (see Displaying configuration details). The SR Linux reads the file and executes each configuration statement line-by-line. You can optionally commit the configuration automatically after the file is read.

Example: Execute configuration from a file

The following example executes a configuration from a specified file:

```
--{ running }--[  ]--
# source config.cfg
Sourcing commands from 'config.cfg'
Executed 20 lines in 1.6541 seconds from file config.cfg
```

Use the **auto-commit** option to commit the configuration after the commands in the source file are executed.

2.3.13 Collecting technical support data

Collect technical support data using the **tech-support** command. Use this command in any configuration mode.

Example: Collect technical support data

```
--{ candidate shared default}--[  ]--
# tech-support
Waiting 5 seconds for apps to dump the reports in /tmp/admintech-report-2019_06_19_20_26_05.zip
Finished collecting in 5s
Admin tech report has been generated at /tmp/admintech-report-2019_06_19_20_26_05.zip
--{ candidate }--[  ]--
```

Example: Access technical support file from bash shell

You can access the saved file from the bash shell. For example:

```
--{ running }--[  ]--
# bash
[root@3-node_srlinux-A /]# cd /tmp
[root@3-node_srlinux-A tmp]# ls -l
total 6012
-rw-r--r-- 1 root root 6110160 Jun 19 20:26 admintech-report-2019_06_19_20_26_05.zip
[root@3-node_srlinux-A tmp]# exit
logout
--{ running }--[  ]--
```

2.4 CLI output formatting and filtering

Several ways exist to format and filter CLI output. These include:

- Specifying the display output (text, JSON, or table format)
- Filtering the output using Linux tools such as **grep**

- Using an output filter
- Directing filtered output to a specified file in a specified format

2.4.1 Specifying output format

You can display output from SR Linux CLI commands as lines of text, in JSON format, or in a table. By default, output is displayed as lines of text, but you can configure output to be displayed in JSON format by default. See Configuring the CLI output format.

Example: show version | as text

The following example displays the output of the **show version** command as lines of text:

```
--{ running }--[  ]--
# show version | as text
--------------------------------------------------------------------------------
Hostname          : 3-node-srlinux-A
Chassis Type      : 7250 IXR-10
Part Number       : Sim Part No.
Serial Number     : Sim Serial No.
System MAC Address: 12:12:02:FF:00:00
Software Version  : v19.11.1
Build Number      : 291-g4664705
Architecture      : x86_64
Last Booted       : 2019-12-07T00:34:48.942Z
Total Memory      : 16396536 kB
Free Memory       : 5321932 kB
--------------------------------------------------------------------------------
```

Example: show version | as json

The following example displays the output of the **show version** command in JSON format:

```
--{ running }--[  ]--
# show version | as json
{
  "basic system info": {
    "Hostname": "3-node-srlinux-A",
    "Chassis Type": "7250 IXR-10",
    "Part Number": "Sim Part No.",
    "Serial Number": "Sim Serial No.",
    "System MAC Address": "12:12:02:FF:00:00",
    "Software Version": "v19.11.1",
    "Build Number": "291-g4664705",
    "Architecture": "x86_64",
    "Last Booted": "2019-12-07T00:34:48.942Z",
    "Total Memory": "16396536 kB",
    "Free Memory": "5319448 kB"
  }
}
```

Example: show version | as table

The following example displays the output of the **show version** command as a table:

```
--{ running }--[  ]--
# show version | as table
+----------+----------+-------+--------+--------+----------+----------+----------+
| Hostname | Chassis  | Part  | Serial | System | Software | Architec |   Last   |
|          |   Type   | Number| Number |  MAC   | Version  |   ture   |  Booted  |
|          |          |       |        | Address|          |          |          |
+==========+==========+=======+========+========+==========+==========+=========+
| 3-node-s | 7250     | Sim Pa| Sim    |12:12:05| v19.11.1 | x86_64   | 2019-12- |
| rlinux-A | IXR-10   |rt No. | Serial |:FF:00:0|          |          | 07T00:34 |
|          |          |       | No.    |0       |          |          | :48.942Z |
+----------+----------+-------+--------+--------+----------+----------+----------+
```

2.4.2 Using Linux output modifiers

Output from SR Linux CLI commands can be piped to standard Linux tools, including **grep**, **more**, **head**, and **tail**.

Example: show interface | grep

The following example pipes the output of the **show interface** command to **grep** so that only lines with `mgmt0` appear in the output:

```
--{ running }--[  ]--
# show interface | grep mgmt0
mgmt0 is up, speed None, type None
  mgmt0.0 is up
```

Example: show interface | more

The following example pipes the output of the **show interface** command to **more** to display one page of output at a time:

```
--{ running }--[  ]--
# show interface | more
==================================================================================
ethernet-1/10 is up, speed 100G, type 100GBASE-CR4 CA-L
  ethernet-1/10.1 is up
    Encapsulation: null
    IPv4 addr    : 192.35.1.0/31 (static)
    IPv6 addr    : 2001:192:35:1::/127 (static, preferred)
    IPv6 addr    : fe80::22e0:9cff:fe78:e2ea/64 (link-layer, preferred)
----------------------------------------------------------------------------------
ethernet-1/21 is up, speed 100G, type 100GBASE-CR4 CA-L
  ethernet-1/21.1 is up
    Encapsulation: null
    IPv4 addr    : 192.45.1.254/31 (static)
    IPv6 addr    : 2001:192:45:1::fe/127 (static, preferred)
    IPv6 addr    : fe80::22e0:9cff:fe78:e2f5/64 (link-layer, preferred)
----------------------------------------------------------------------------------
ethernet-1/22 is up, speed 100G, type 100GBASE-CR4 CA-L
  ethernet-1/22.1 is up
    Encapsulation: null
    IPv4 addr    : 192.45.3.254/31 (static)
    IPv6 addr    : 2001:192:45:3::fe/127 (static, preferred)
    IPv6 addr    : fe80::22e0:9cff:fe78:e2f6/64 (link-layer, preferred)
----------------------------------------------------------------------------------
ethernet-1/3 is up, speed 100G, type 100GBASE-CR4 CA-L
  ethernet-1/3.1 is up
    Encapsulation: null
    IPv4 addr    : 192.57.1.1/31 (static)
    IPv6 addr    : 2001:192:57:1::1/127 (static, preferred)
    IPv6 addr    : fe80::22e0:9cff:fe78:e2e3/64 (link-layer, preferred)
----------------------------------------------------------------------------------
--More--
```

Example: show interface | head

The following example pipes the output of the **show interface** command to **head** to display only the first 8 lines:

```
--{ running }--[  ]--
# show interface | head --lines 8
=================================================================================
ethernet-1/10 is up, speed 100G, type 100GBASE-CR4 CA-L
  ethernet-1/10.1 is up
    Encapsulation: null
    IPv4 addr    : 192.35.1.0/31 (static)
    IPv6 addr    : 2001:192:35:1::/127 (static, preferred)
    IPv6 addr    : fe80::22e0:9cff:fe78:e2ea/64 (link-layer, preferred)
---------------------------------------------------------------------------------
```

2.4.3 Using output filters

You can use the **filter** option to limit the output of **show** and **info from state** commands. Filter options include:

- **node** - Defines a specific node to filter on
- **depth** - Filters out sub-nodes that are deeper than the specified depth
- **fields** - Specifies a specific field or fields to filter on
- **keys-only** - Hides all fields
- **non-zero** - Filters integer fields set to 0 and empty strings

The **show** or **info from state** command is piped to the **filter** command with the following usage: **filter [<*node*>] [depth <*value*>] [fields <*value*>] [keys-only] [non-zero]**

Example: Filter info from state command

The following example directs the output of the **info from state** command to filter on the Ethernet node.

```
--{ running }--[  ]--
# info from state interface ethernet-1/1 | filter ethernet
    interface ethernet-1/1 {
        description dut1-dut2-1
        admin-state enable
        mtu 9232
        loopback-mode false
        ifindex 54
        oper-state up
        last-change "an hour ago"
        vlan-tagging false
        ethernet {
            port-speed 10G
            hw-mac-address 00:01:02:FF:00:00
            flow-control {
                receive false
            }
            statistics {
                in-mac-pause-frames 0
                in-oversize-frames 0
                in-jabber-frames 0
                in-fragment-frames 0
                in-crc-error-frames 0
                out-mac-pause-frames 0
                in-64b-frames 0
                in-65b-to-127b-frames 0
                in-128b-to-255b-frames 0
                in-256b-to-511b-frames 0
                in-512b-to-1023b-frames 0
                in-1024b-to-1518b-frames 0
                in-1519b-or-longer-frames 0
                out-64b-frames 0
                out-65b-to-127b-frames 0
                out-128b-to-255b-frames 0
                out-256b-to-511b-frames 0
                out-512b-to-1023b-frames 0
                out-1024b-to-1518b-frames 0
                out-1519b-or-longer-frames 0
            }
        }
    }
```

Example: Apply filter to a specific field

Specify the field option to further filter the output to a specific field. For example:

```
--{ running }--[  ]--
# info from state interface ethernet-1/1 | filter fields ethernet/port-speed
    interface ethernet-1/1 {
        ethernet {
            port-speed 10G
        }
    }
```

Example: Apply non-zero filter

The following example shows how you can use the the **non-zero** option to display only non-zero values to eliminate non-useful information. The use of the non-zero option only filters integer fields that are currently set to 0 and empty strings.

```
--{ running }--[  ]--
# info from state interface ethernet-1/* statistics | filter non-zero
    interface ethernet-1/1 {
        statistics {
            in-octets 1106761
            in-unicast-packets 1592
            in-broadcast-packets 32
            in-multicast-packets 10736
            in-error-packets 5
            out-octets 2859625
            out-unicast-packets 31576
            out-broadcast-packets 18
            out-multicast-packets 243
            carrier-transitions 3
        }
    }
    interface ethernet-1/2 {
        statistics {
            in-octets 1399961
            in-unicast-packets 129
            in-broadcast-packets 11
            in-multicast-packets 14772
            in-error-packets 1
            out-octets 2474023
            out-unicast-packets 26126
            out-broadcast-packets 19
            out-multicast-packets 173
            carrier-transitions 3
        }
    }
    interface ethernet-1/20 {
        statistics {
            in-octets 1443012
            in-unicast-packets 350
            in-broadcast-packets 12
            in-multicast-packets 15639
            in-error-packets 3
            out-octets 2319629
            out-unicast-packets 25558
            out-broadcast-packets 17
            out-multicast-packets 199
            carrier-transitions 3
        }
    }
```

2.4.4 Directing output to a file

You can direct the output of SR Linux CLI commands to a specified file. The output can be saved as text, in table format, or in JSON format.

Example: Direct show interface output to file

The following example directs the output of the **show interface** command to a file. The output is saved in JSON format.

```
--{ running }--[  ]--
# show interface | as json > show_interface.json
```

Use **>** to create a new file with the specified filename; if the file already exists, it is replaced. Use **>** to append the output to the specified file if it exists.

Example: Access output file in bash mode

You can access the file using bash mode. For example:

```
--{ * running }--[  ]--
# bash
[bob@3-node-srlinux-A /]# more show_interface.json
{
  "interfaces": [
    {
      "Interface": "ethernet-1/1",
      "subinterfaces": [
        {
          "Subinterface": "ethernet-1/1.1",
          "Oper": "up",
          "IPv4 Addresses": "192.168.11.1/30",
          "IPv6 Addresses": "2001:1::192:168:11:1/126, fe80::1012:5ff:feff:0/64"
        }
      ]
    },
    {
--More--(42%)
```

2.5 CLI environment customization

You can optionally configure the SR Linux CLI environment to change settings such as the command prompt, contents of the bottom toolbar, and the default format for displayed output. You can create aliases for CLI commands. The CLI environment settings can be saved to the default SR Linux configuration or to a specified file and subsequently loaded and applied to the current CLI session.

2.5.1 Configuring the CLI prompt

About this task

By default, the SR Linux CLI prompt consists of two lines of text, indicating with an asterisk whether the configuration has been modified, the current mode and session type, the current CLI context, and the host name of the SR Linux device, in the following format:

```
--{ modified? mode_and_session_type }--[ context ]--
hostname#
```

For example:

```
--{ * candidate shared }--[ acl ]--
3-node-srlinux-A#
```

You can configure the SR Linux prompt to include information such as the username or session ID of the CLI session, the number of changes made to the configuration, and the current local time.

Example: Add local time and session username to CLI prompt

The following example adds the local time and session username to the SR Linux CLI prompt.

```
--{ candidate shared default}--[  ]--
# environment prompt "--{{ {modified}{mode_and_session_type} }}--[ {pwc} ]--{time}--\
n{user}@{host}# "
```

In the example, the local time is configured with the **{time}** keyword, and the session username is configured with the **{user}** keyword. The line break is configured with **\n**. Use the **environment prompt ?** command to display the keywords that you can configure in the SR Linux CLI prompt.

After you enter this command, the CLI prompt looks like the following:

```
--{ * candidate shared default}--[ acl ]--Wed 03:07PM--
bob@3-node-srlinux-A#
```

2.5.2 Configuring the bottom toolbar text

About this task

By default, the text that appears at the bottom of the terminal window in SR Linux CLI sessions displays the current mode and session type, whether the configuration has been modified, the username and session ID of the current AAA session, and the local time, in the following format:

```
Current mode: modified? mode_and_session_type aaa_user (aaa_session_id) time
```

For example:

```
Current mode: * candidate shared                          root (36)  Wed 09:52PM
```

You can configure the bottom toolbar to include information such as the number of changes made to the configuration and the host name.

Example: Add number of changes and host name to toolbar

The following example adds the number of changes made to the configuration and the host name to the bottom toolbar.

```
--{ candidate shared default}--[  ]--
# environment bottom-toolbar "Current mode: {modified_with_change_count}{mode_and_session_type}
 | {aaa_user}@{host} ({aaa_session_id})  {time}"
```

In the example, the number of configuration changes is configured with the **{modified_with_change_count}** keyword, and the host name is configured with the **{host}** keyword. Use the **environment bottom-toolbar ?** command to display the keywords that you can configure in the bottom toolbar.

After you enter this command, the bottom toolbar looks like the following:

```
Current mode: *10 candidate shared       root@3-node-srlinux-A (36)  Wed 10:18PM
```

2.5.3 Configuring the SR Linux CLI engine type

The SR Linux features two versions of the CLI engine: advanced and basic. The advanced CLI engine is enabled by default; it includes the following features:

- Displays a toolbar at the bottom of the terminal window.
- Includes the command auto-complete feature.
- Allows you to select command options by pressing the **Tab** key to display the options in a popup, then using the arrow keys to select an option.
- Displays descriptions of command options when you press the **?** key.

The basic CLI engine includes a limited set of features compared to the advanced version:

- Omits the bottom toolbar.
- Does not present a selectable list of options when you press the **Tab** key.
- Displays descriptions of command options when you press the **?** key (and press **Enter**), but only at the # prompt for the current context.

If necessary, you can configure the SR Linux to use the basic CLI engine instead of the advanced CLI engine for CLI sessions.

Example: Set CLI engine type to basic

The following example configures the SR Linux to use the basic CLI engine for CLI sessions:

```
--{ candidate shared default}--[  ]--
# environment cli-engine type basic
```

Related topics

Configuring the bottom toolbar text

Using the CLI auto-complete function

2.5.4 Configuring the CLI output format

You can configure the output of CLI commands to be displayed as either text or in JSON format.

Example: Set output to JSON format

The following example configures CLI command output to be displayed in JSON format.

```
--{ candidate shared default}--[  ]--
# environment output-format json
```

Subsequent command output is displayed in JSON format by default. For example:

```
--{ running }--[  ]--
# show version
{
  "basic system info": {
    "Hostname": "3-node-srlinux-A",
    "Chassis Type": "7250 IXR-10",
    "Part Number": "Sim Part No.",
    "Serial Number": "Sim Serial No.",
    "System MAC Address": "12:12:02:FF:00:00",
    "Software Version": "v19.11.1",
    "Build Number": "291-g4664705",
    "Architecture": "x86_64",
    "Last Booted": "2019-12-07T00:34:48.942Z",
    "Total Memory": "16396536 kB",
    "Free Memory": "5319448 kB"
  }
}
```

2.5.5 Configuring CLI command aliases

As a shortcut for entering commands in the CLI, you can configure CLI command aliases. The alias can include one or more CLI command keywords and arguments.

Example: Set alias for info interface command

The following example configures the alias **display interface** for the SR Linux command **info interface** <*name*>**subinterface** <*index*> **| as table**:

```
--{ candidate shared default}--[  ]--
# environment alias "display interface" "info / interface {} subinterface {subinterface} | as
 table"
```

In the example, the **display interface** alias consists of the keywords **info interface**, arguments to specify an interface name and subinterface number, and keywords to display the output as a table. The alias name and aliased command are each enclosed in quotes. The arguments are enclosed in braces (**{ }**). The argument **{}** creates an optional unnamed variable for the interface name, and the argument **{subinterface}** creates an optional parameter named `subinterface`.

When you enter the alias at the CLI prompt, the output of the aliased command is displayed. For example:

```
# display interface ethernet-1/1 subinterface 1
+---------------------+-------+------------+
|      Interface      | Index | Admin-state |
+=====================+=======+=============+
| ethernet-1/1        |     1 | enable      |
+---------------------+-------+-------------+
```

If you omit the optional parameters for the interface and subinterface names, they are treated as wildcards. For example:

```
# display interface
+---------------------+-------+-------------+
|      Interface      | Index | Admin-state |
+=====================+=======+=============+
| ethernet-1/1        |     1 | enable      |
| ethernet-1/2        |     1 | enable      |
| lo0                 |     1 | enable      |
| mgmt0               |     0 | enable      |
+---------------------+-------+-------------+
```

2.5.6 Configuring command auto-completion

By default, if you enter a tab at any mode or level to auto-complete the next command level, a popup appears that displays the available options for that command.

You can optionally configure the space bar to provide the same function as the **Tab** key, so that pressing the space bar auto-completes the command.

Example: Set space bar to auto-complete commands

The following example configures the space bar to auto-complete commands in SR Linux CLI sessions:

```
--{ candidate shared default}--[  ]--
# environment complete-on-space true
```

Related topics

Using the CLI auto-complete function

2.5.7 Displaying the CLI environment configuration

To display the CLI environment configuration, including any CLI command aliases, use the **environment show** command.

Example: environment show

```
--{ candidate shared default}--[  ]--
# environment show
[alias]
"show system configuration session" = "info from state / system configuration session {} | as
 table | filter fields username type started"
"show system configuration checkpoint" = "info from state / system configuration checkpoint {}
 | as table | filter fields name comment username created"
"display interface" = "info / interface {} subinterface {subinterface} | as table

[bottom-toolbar]
value = "Current mode: {modified}{mode_and_session_type} | {aaa_user} ({aaa_session_id})
 {time}"

[cli-engine]
```

```
type = "advanced"

[output-display-format]
value = "text"

[prompt]
value = "--{{ {modified}{mode_and_session_type} }}--[ {pwc} ]--\n{host}# "

[space-completion]
enabled = false
```

2.5.8 Managing CLI environment settings

You can save the current CLI environment settings to the default SR Linux configuration or to a file, and you can load CLI environment settings from the default SR Linux configuration or from a file.

Example: Save CLI environment settings to default configuration

The following example saves the current CLI environment settings to the default SR Linux configuration:

```
--{ candidate shared default}--[  ]--
# environment save home
Saved configuration to /root/.srlinuxrc
```

Example: Save CLI environment settings to file

The following example saves the current CLI environment settings to a file:

```
--{ candidate shared default}--[  ]--
# environment save file env.cfg
Saved configuration to env.cfg
```

Example: Load CLI environment settings from default

The following example loads CLI environment settings from the default SR Linux configuration:

```
--{ candidate shared default}--[  ]--
# environment load home
Loaded configuration from /root/.srlinuxrc
```

Example: Load CLI environment settings from file

The following example loads CLI environment settings from a file:

```
--{ candidate shared default}--[  ]--
# environment load file env.cfg
Loaded configuration from env.cfg
```

3 RPC overview and supporting interfaces

SR Linux supports the gNMI and JSON interfaces that use the Remote Procedure Call (RPC) protocol for the modification and retrieval of a configuration from a target device as if it were a local object. This chapter provides a general RPC overview and defines how SR Linux implements the gNMI and JSON serving RPCs.

3.1 RPC overview

An RPC executes a procedure or method on a remote device in a way similar to one executed locally. Using an RPC should look and feel like a local procedure call and achieve similar results.

When RPCs operate in a server/client model, the client executes an application and requests some method be executed on a server. The server receives the RPC, executes the method, and sends the outcome back to the server. For this to occur, the following are needed:

- A server application that contains a set of methods (or service) which a client can call. For example, a gNMI server or JSON-RPC server. Each needs to receive an RPC from a client for a specific method or service, execute the requested methods/service, and return data.
- A common way of describing which method to execute and associated data.

  This is referred to as encoding/decoding or serialization/deserialization. For example, proto buffers or JSON.
- A transport mechanism between the two devices. In the case of gNMI and JSON-RPC this is done using HTTP/2 and HTTP1.1 over TCP secured by TLS.

For RPCs to relate to network applications and a network operating system, there needs to be a way to define the data model of the network constructs which an RPC must perform. In the SR Linux, all applications are modeled in YANG, as shown in the following figure.

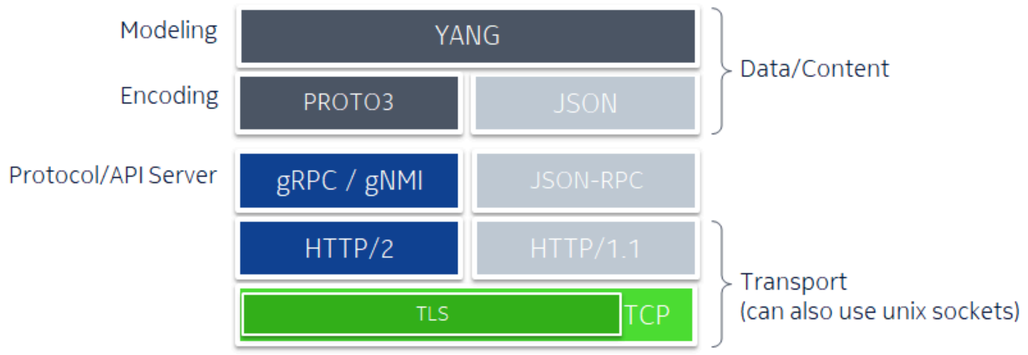


Figure 1: RPC-based interfaces in SR Linux

3.1.1 gNMI path convention

Because SR Linux is modeled in YANG, for RPCs to retrieve or configure an SR Linux device (set) the data location or path to the data within the YANG model must be specified to the RPC. For example, to configure a description under a BGP peer, the RPC server must be able to reference this specific data and its location in the SR Linux data model.

Both the JSON-RPC and gNMI use the gNMI path convention to describe the location or path in SR Linux. The gNMI path convention is commonly referred to as the "path" and defines the data structure of a path to reference a config or state leaf within SR Linux.

A path is an ordered list of path elements with each element containing an element name and one or more key value pairs associated with the element. For example:

```
path: <
  elem: <
    name: "interface"
    key: <
      key: "name"
      value: "ethernet-1/20"
    >
  >
  elem: <
    name: "subinterface"
    key: <
      key: "index"
      value: "1"
    >
```

A path can also be displayed as a string which is more human readable. The rules for building this string type are:

- Each path element is separated by a "/" character and listed in sequence.
- A path element which contains one or more key/value pairs is represented by the path element name followed the key/value wrapped in brackets ([ ]).
- The root of the data tree is represented by a single "/".

The following is a human readable path for the previous example:

```
/interface=ethernet-1/20/subinterface[index=1]
```

3.2 Configuring a gNMI or JSON server

The SR Linux can enable a gNMI server that allows external gNMI clients to connect to the device and modify the configuration and collect state information. You can also enable a JSON-RPC server on the SR Linux device, which allows you issue JSON-formatted requests to the device to retrieve and set configuration and state.

See the Management Servers chapter in the *SR Linux Configuration Basics Guide* for details on how to configure these servers.

4 gNMI interface

A gRPC Network Management Interface (gNMI) is a gRPC based protocol that defines a service or set of services or RPC methods used to configure and retrieve data from network devices.

The SR Linux provides a gNMI-based RPC for the modification and retrieval of a configuration. Supported RPCs are:

- get
- set
- subscribe
- capabilities

4.1 Common notification messages

When the SR Linux gNMI process communicates data to a client, it uses common notification messages. Notification messages use the fields shown in the following table.

Table 4: Common notification fields

Field	Definition
timestamp	Time data was collected
prefix	Prefix applied to all path fields included in the notification message. The paths expressed within the message are formed by the concatenation of prefix + path.
update	List of update messages that indicate changes in the underlying data. Subfields are: • path • val (value)
delete	List of paths indicating the deletion of data nodes

4.1.1 Timestamps

Timestamp values are represented in nanoseconds. The value is encoded as a signed 64-bit integer (int64).

4.1.2 Path prefix

A prefix can be specified to reduce the lengths of path fields within a message. The absolute path is a concatenation of the path elements representing the prefix and the list of path elements in the path field. For example:

Example: Path Prefix

```
notification: <
timestamp: (timestamp) // timestamp as int64
prefix: <
  elem: <
      name: "a"
    >
  elem: <
     name: "b"
     key: <
        key: "name"
        value: "b1"
       >
     >
  elem: <
     name: "c"
  >
>
update: <
  path: <
    elem: <
      name: "d"
    >
  >
  value: <
    val: <
      json_val: "AStringValue"
    >
  >
>
update: <
  path: <
    elem: <
      name: "e"
    >
  >
  val: <
    json_val: 10042 // converted to int representation
  >
 >
>
```

4.1.3 Paths

Paths are represented according to gNMI path conventions. Each path is represented by an ordered list of PathElem messages, starting at the root node, and ending at the most specific path element (versus a single string with a "/" character separating each element). Each PathElem message contains the name of the node within the data tree, along with any associated keys and attributes that may be required. For example:

Example

```
path: <
```

```
  elem: <
    name: "a"
  >
  elem: <
    name: "e"
    key: <
      key: "key"
      value: "k1"
    >
  >
  elem: <
    name: "f"
  >
  elem: <
    name: "g"
  >
>
```

Multiple paths are supported. Multiple notification messages are triggered in response to each path. For example:

Example

```
path <
  elem <
    name: "interface"
    key <
      key: "name"
      value: "mgmt0"
    >
  >
>
path <
  elem <
    name: "system"
  >
>
type: 1
encoding: JSON_IETF
```

4.1.4 Data node values

The value of a data node can be the following:

- scalar types, such as a string in the string_val field, int64 in the int_val field, unit64 in the uint_val field, bool in the bool_val field, bytes, and float in the float_val field
- additional types used in some schema languages, such as decimal64 in the decimal_val field and ScalarArray in the leaflist_val field
- structured data types

4.1.4.1 Structured data types

When structured data is sent in an update message, it is serialized according to supported encoding, as shown in the following table.

Table 5: Encoding for structured data

Data type	Description	Field
ASCII	An ASCII encoded string	ascii_val
JSON_IETF	A JSON encoded string using JSON encoding compatible with RFC 7951	json_ietf_val

4.2 gNMI get RPC

The get RPC allows you to obtain a view of the existing state. A GetRequest message is sent to the target (SR Linux gNMI process gnmi_mgr) that specifies the data to retrieve. A GetResponse message is returned that reflects the values of specified leafs at the collection time.

The get RPC is recommended for retrieving small data sets. For larger data sets, the gNMI subscribe RPC is recommended, using the ONCE mode.

Related topics

gNMI subscribe RPC

4.2.1 GetRequest message

A GetRequest message retrieves a view of data from the server. A Get RPC requests the server retrieve a subset of the data tree as specified by the paths included in the message and serializes this using the specified encoding.The GetRequest message uses the fields shown in the following table.

Table 6: GetRequest fields

Field	Definition
path	Path (or set of paths) for the requested data view. Wildcards are permitted.
type	Type of data requested. Supported options are: • CONFIG (configurable read/write data) • STATE (non-configurable read-only data)
encoding	Encoding that the target should use (ASCII or JSON_IETF). If not specified, JSON is the default.
extension	Repeated field to carry gNMI extensions

4.2.2 GetResponse message

The GetResponse message uses the fields shown in the following table.

Table 7: GetResponse fields

Field	Definition
notification	Set of notification messages for each path specified in the Get Request.
extension	Repeated field to carry gNMI extensions

Related topics
Common notification messages

4.3 gNMI set RPC

The set RPC allows you to modify an existing state. A SetRequest message is sent to the target (SR Linux gNMI process gnmi_mgr) that specifies the required modifications. The server deletes, replaces, and updates paths based on the order they are listed. For each operation designated in the SetRequest message, an UpdateResult message is included in the SetResponse message.

4.3.1 SetRequest message

The SetRequest message uses the fields shown in the following table.

Table 8: SetRequest fields

Field	Definition
prefix	A specified prefix is applied to all defined paths within each field
delete	A set of paths to be removed from the data tree
replace	A set of update messages that defines content to replace
update	A set of update messages that defines content to update
extension	Repeated field to carry gNMI extensions

An update message indicates changes to paths where a new value is required. Update messages contain the following:

- path - the path of the element to be modified
- value - a value to apply to the specified node

All changes to the state included in a SetRequest message are consider part of a transaction. Either all modifications are applied or changes are rolled back to reflect the original state. For changes to be applied together, they must be in a single SetRequest message.

For replace operations, the behavior of omitted data elements depends on whether they are non-default values (set by a previous SetRequest message) or unmodified defaults. When the replace operation omits values that have been previously set, they are deleted from the data tree. Otherwise, omitted data elements are created with their default values.

For update operations, only the value of the data elements explicitly specified are changed.

4.3.2 SetResponse message

The SetResponse message uses the fields shown in the following table.

Table 9: SetResponse fields

Field	Definition
prefix	The prefix specified for all paths
response	A list of responses (one per operation). Each response consists of an UpdateResult message with the following: • timestamp - time when the SetRequest message was accepted • path - path defined in SetRequest message. A prefix may be present to reduce repetition of path elements. • op - the operation performed on the path (delete, replace, or update) • message - a status message
extension	Repeated field to carry gNMI extensions

4.4 gNMI subscribe RPC

The subscribe RPC allows you to receive updates relating to the state of data instances. The user creates a subscription using the subscribe RPC with the desired subscription mode. The defined mode triggers how and when the data is sent to the client.

A SubscribeRequest message is sent to the target (SR Linux gNMI process gnmi_mgr) to request updates for one or more paths. A SubscribeReponse message is sent to the client over an established RPC.

4.4.1 SubscribeRequest message

The SubscribeRequest message uses the fields shown in the following table.

Table 10: SubscribeRequest fields

Field	Definition
subscribe	A SubscriptionList message specifying a new set of paths to subscribe to
extension	Repeated field to carry gNMI extensions

Subscriptions are set once and cannot be modified. A new subscribe RPC call must be created for new paths. To end an existing subscription, the client must cancel the subscribe RPC that relates to the subscription.

4.4.1.1 SubscriptionList message

A SubscriptionList message indicates a set of paths where common subscription behavior is required. The SubscriptionList message uses the fields shown in the following table.

Table 11: SubscriptionList fields

Field	Definition
subscription	A set of subscription messages indicating the paths associated with the subscription
mode	Type of subscription to create: • ONCE • STREAM (default) – ON_CHANGE – SAMPLE • sample_interval – TARGET_DEFINED
extension	Repeated field to carry gNMI extensions

ONCE subscriptions are one-time requests. A ONCE subscription is created by sending a SubscribeRequest message with the subscribe field containing a SubscriptionList, with the mode type set to ONCE. The relevant update messages are sent and the RPC channel is closed.

STEAM subscriptions are long-lived and transmit updates indefinitely. A STREAM subscription is created by sending a SubscribeRequest message with the subscribe field containing a SubscriptionList, with the mode type set to STREAM. The STEAM mode subscription message also specifies a mode.

- ON_CHANGE - Data updates are only sent when the value of the data item changes.
- SAMPLE - Data is sent at specified intervals as specified in the sample_interval field. The maximum sample rate is a 64-bit integer in nanoseconds and minimum is 0.

- TARGET_DEFINED - The target determines the best subscription type to create on a per-leaf basis. For example, if the path specified refers to leaves that are event-driven, then an ON_CHANGE subscription may be created. If the data represents counters values, a SAMPLE subscription may be created.

4.4.2 SubscribeResponse message

The SubscribeResponse message uses the fields shown in the following table.

Table 12: SubscribeResponse fields

Field	Definition
update OR sync_response	A response field. Only one type can be specified per message: • update - message providing an update value for a subscribed data entity • sync_response - a Boolean field indicating that all data values corresponding to the paths have been transmitted at least once (not used with ONCE mode subscriptions)
extension	Repeated field to carry gNMI extensions

4.5 gNMI capabilities RPC

The capabilities RPC allows you to discover the capabilities of a specific gNMI server.

A CapabilityRequest message is sent by the client to request capability information from the target. The target replies with a CapabilityResponse message that includes its gNMI service version, the versioned data models it supports, and the supported data encodings.

This information is used in subsequent RPC messages from the client to indicate the set of models that the client uses, and the encoding used for data.

4.5.1 CapabilityRequest message

The CapabilityRequest message is sent by the client to request capability information from the target. The CapabilityRequest message carries a single repeated extension field which can be used to carry gNMI extensions.

Example: CapabilityRequest message

```
message CapabilityRequest {
 repeated gnmi_ext.Extension extension = 1;
}
```

4.5.2 CapabilityResponse message

A CapabilityResponse message is sent from the target and includes the following fields:

- supported_models - a set of ModelData messages describing each model supported by the target
- supported_encodings - an enumerated field describing the data encodings supported by the target (ASCII and JSON_IETF are supported)
- gNMI_version - the version of the gNMI service supported by the target
- encoding - a repeated field for gNMI extensions

Example: CapabilityResponse message

```
message CapabilityResponse {
 repeated ModelData supported_models = 1;
 repeated Encoding supported_encodings = 2;
 string gNMIversion = 3;
 repeated gnmi_ext.Extension extension = 4;
}
```

4.6 Candidate mode

The gNMI uses its own private candidate that allows multiple users or services to make simultaneous changes to a configuration.

4.7 gNMI examples

Open source clients can be used to run GetRequests, SetRequests, subscriptions, and capabilities. The examples that follow show requests and responses using the following clients although any client that conforms to gNMI specifications can be used:

- gnmi_get — used for simple GetRequests
- gnmi_set — used for simple SetRequests
- gnmi_cli — used for SubscribeRequests, and advanced GetRequests and SetRequests
- gnmi_capabilities — used for CapabilityRequests

4.7.1 gnmi_get examples

The get gNMI-RPC allows you to retrieve state and configuration from a datastore. The following examples are shown:

- get all request
- get interface with wildcard key request

Example: get all request

```
# gnmi_get -target_addr 172.18.0.6:50052 -insecure -xpath '/'
== getRequest:
path: <
>
encoding: JSON_IETF
```

Response (get all)

```
notification: <
  timestamp: 1565672122888042050
  update: <
    path: <
    >
    val: <
      json_ietf_val: "{\n \"srl_nokia-acl:acl\": {\n \"ipv4-
      filter\" ---- snip ---- ]\n }\n}\n"
    >
  >
>
```

Example: get interface with wildcard key request

```
# gnmi_get -target_addr 172.18.0.6:50052 -insecure -xpath
'/interface[name=mgmt0]/subinterface[index=*]'
== getRequest:
path: <
  elem: <
    name: "interface"
    key: <
      key: "name"
      value: "mgmt0"
    >
  >
  elem: <
    name: "subinterface"
    key: <
      key: "index"
      value: "*"
    >
  >
>
encoding: JSON_IETF
```

Response (get interface with wildcard key)

```
notification: <
  timestamp: 1565671919030747121
  update: <
    path: <
      elem: <
        name: "srl_nokia-interfaces:interface"
        key: <
          key: "name"
          value: "mgmt0"
        >
      >
    >
    val: <
      json_ietf_val: "{\n \"name\": \"mgmt0\",\n \"subinterface\":
      [\n {\n \"index\": 0,\n \"admin-state\": \"enable\",\n
      \"ip-mtu\": 1500,\n \"ifindex\": 524288000,\n \"operstate\":
      \"up\",\n \"last-change\": \"2019-08-
      11T17:21:48.366Z\",\n \"ipv4\": {\n \"allow-directedbroadcast\":
      false,\n \"dhcp-client\": true,\n
      \"address\": [\n {\n \"ip-prefix\":in
      \"172.18.0.6/24\",\n \"origin\": \"dhcp\"\n }\n
      ],\n \"srl_nokia-interfaces-nbr:arp\": {\n
      \"timeout\": 14400,\n \"neighbor\": [\n {\n
      \"ipv4-address\": \"172.18.0.1\",\n \"link-layeraddress\":
```

```
      \"02:42:45:9D:DB:FC\",\n \"origin\":
      \"dynamic\",\n \"expiration-time\": \"2019-08-
      13T07:14:34.707Z\"\n },\n {\n
      \"ipv4-address\": \"172.18.0.2\",\n \"link-layeraddress\":
      \"02:42:AC:12:00:02\",\n \"origin\":
      \"dynamic\",\n \"expiration-time\": \"2019-08-
      13T05:17:51.893Z\"\n }\n ]\n }\n },\n
      \"ipv6\": {\n \"dhcp-client\": true,\n \"address\": [\n
      {\n \"ip-prefix\": \"2001:172:18::6/80\",\n
      \"origin\": \"dhcp\",\n \"status\": \"preferred\"\n
      },\n {\n \"ip-prefix\":
      \"fe80::42:acff:fe12:6/64\",\n \"origin\": \"linklayer\",\
      n \"status\": \"preferred\"\n }\n
      ],\n \"srl_nokia-interfaces-nbr:neighbor-discovery\": {\n
      \"dup-addr-detect\": true,\n \"reachable-time\": 30,\n
      \"stale-time\": 14400\n }\n },\n \"statistics\": {\n
      \"in-pkts\": \"5136\",\n \"in-octets\": \"438953\",\n
      \"in-error-pkts\": \"0\",\n \"in-discarded-pkts\": \"0\",\n
      \"in-terminated-pkts\": \"5136\",\n \"in-terminated-octets\":
      \"438953\",\n \"in-forwarded-pkts\": \"0\",\n \"inforwarded-
      octets\": \"0\",\n \"out-forwarded-pkts\":
      \"6062\",\n \"out-forwarded-octets\": \"2746613\",\n
      \"out-error-pkts\": \"0\",\n \"out-discarded-pkts\": \"0\",\n
      \"out-pkts\": \"6062\",\n \"out-octets\": \"2746520\"\n
      },\n \"srl_nokia-qos:qos\": {\n \"input\": {\n
      \"classifiers\": {\n \"ipv4-dscp\": \"default\",\n
      \"ipv6-dscp\": \"default\",\n \"mpls-tc\": \"default\"\n
      }\n }\n }\n }\n ]\n}\n"
    >
  >
>
```

4.7.2 gnmi_set examples

The set gNMI-RPC allows you to modify the state. The following examples are shown:

- set delete request
- set update all request

Example: set delete request

```
# gnmi_set -target_addr 172.18.0.3:50052 -username admin -password
admin -insecure -delete /system/name/host-name
== setRequest:
delete: <
  elem: <
    name: "system"
  >
  elem: <
    name: "name"
  >
  elem: <
    name: "host-name"
  >
>
```

Response (set delete)

```
response: <
  path: <
    elem: <
```

```
      name: "system"
    >
    elem: <
      name: "name"
    >
    elem: <
      name: "host-name"
    >
  >
  op: DELETE
>
timestamp: 1567203341816078044
```

Example: set an update all request

```
# gnmi_set -target_addr 172.18.0.3:50052 -username admin -password
admin -insecure -update /system/name/host-name:replaced-host -replace
/system/name/domain-name:replaced-domain
== setRequest:
replace: <
  path: <
    elem: <
      name: "system"
    >
    elem: <
      name: "name"
    >
    elem: <
      name: "domain-name"
    >
  >
  val: <
    string_val: "replaced-domain"
  >
>
update: <
  path: <
    elem: <
      name: "system"
    >
    elem: <
      name: "name"
    >
    elem: <
      name: "host-name"
    >
  >
  val: <
    string_val: "replaced-host"
  >
>
```

Response (set update all)

```
response: <
  path: <
    elem: <
      name: "system"
    >
    elem: <
      name: "name"
    >
    elem: <
      name: "domain-name"
    >
  >
  op: REPLACE
response: <
  path: <
    elem: <
      name: "system"
    >
    elem: <
      name: "name"
    >
    elem: <
      name: "host-name"
    >
  >
  op: UPDATE
>
timestamp: 1567204165851469784
```

4.7.3 gnmi_cli examples

The cli gNMI-RPC allows you to subscribe and receive updates on the state of a data instance. The following examples are shown:

- Subscribe - ONCE for all (one-time subscription) request
- Subscribe - STREAM ON_CHANGE interface (long term subscription) request

In these examples, `-qt` specifies the subscription type. ONCE mode is the default and therefore is not shown in the first example.

Example: Subscribe ONCE for all request

```
# gnmi_cli -a 172.18.0.6:50052 -insecure -q '/'
```

Response (subscribe ONCE for all)

```
{
  "acl": {
    "ipv4-filter": {
      "allow_sip_dip": {
        "entry": {
          "10": {
            "action": {
              "accept": {
                "log": "false"
              }
-- Snip –
}
```

Example: Subscribe STREAM ON_CHANGE interface request

```
# gnmi_cli -a 172.18.0.6:50052 -insecure --qt streaming -q
'/interface[name=mgmt0]'
```

Response (Subscribe STREAM ON_CHANGE interface)

```
{
  "interface": {
    "mgmt0": {
      "admin-state": "enable",
      "ethernet": {
      "flow-control": {
        "receive": "false"
      },
      "hw-mac-address": "02:42:AC:12:00:06",
      "statistics": {
        "in-crc-errors": "0",
        "in-fragment-frames": "0",
        "in-jabber-frames": "0",
        "in-mac-pause-frames": "0",
        "in-oversize-frames": "0",
        "out-mac-pause-frames": "0"
      }
    },
    "ifindex": "524304383",
    "last-change": "2019-08-30T18:44:45.490Z",
    "mtu": "1514",
    "oper-state": "up",
    "statistics": {
      "carrier-transitions": "1",
      "in-broadcast-pkts": "5",
      "in-errors": "0",
      "in-fcs-errors": "0",
      "in-multicast-pkts": "1356",
      "in-octets": "612022",
      "in-unicast-pkts": "4662",
      "out-broadcast-pkts": "1",
      "out-errors": "0",
      "out-multicast-pkts": "456",
      "out-octets": "2724476",
      "out-unicast-pkts": "5505"
    },
    "subinterface": {
      "0": {
        "admin-state": "enable",
        "ifindex": "524288000",
        "ip-mtu": "1500",
        "ipv4": {
          "address": {
            "172.18.0.6/24": {
              "origin": "dhcp"
            }
          },
          "allow-directed-broadcast": "false",
          "arp": {
            "neighbor": {
              "172.18.0.1": {
                "expiration-time": "2019-08-31T01:13:22.987Z",
                "link-layer-address": "02:42:45:9D:DB:FC",
                "origin": "dynamic"
              },
              "172.18.0.2": {
                "expiration-time": "2019-08-30T22:44:54.422Z",
                "link-layer-address": "02:42:AC:12:00:02",
                "origin": "dynamic"
              }
            },
            "timeout": "14400"
          },
          "dhcp-client": "true"
        },
        "ipv6": {
          "address": {
            "2001:172:18::6/80": {
              "origin": "dhcp",
              "status": "preferred"
            },
           "fe80::42:acff:fe12:6/64": {
              "origin": "link-layer",
              "status": "preferred"
            }
          },
          "dhcp-client": "true",
          "neighbor-discovery": {
            "dup-addr-detect": "true",
            "reachable-time": "30",
            "stale-time": "14400"
          }
        },
        "last-change": "2019-08-30T18:44:45.490Z",
        "oper-state": "up",
        "qos": {
          "input": {
            "classifiers": {
              "ipv4-dscp": "default",
              "ipv6-dscp": "default",
              "mpls-tc": "default"
            }
          }
        },
        "statistics": {
          "in-discarded-pkts": "0",
          "in-error-pkts": "0",
          "in-forwarded-octets": "0",
          "in-forwarded-pkts": "0",
          "in-octets": "404380",
          "in-pkts": "4679",
          "in-terminated-octets": "404380",
          "in-terminated-pkts": "4679",
          "out-discarded-pkts": "0",
          "out-error-pkts": "0",
          "out-forwarded-octets": "2409995",
          "out-forwarded-pkts": "5511",
          "out-octets": "2409995",
          "out-pkts": "5511"
        }
      }
    },
    "vlan-tagging": "false"
  }
}
{
  "interface": {
    "mgmt0": {
      "statistics": {
        "in-octets": "615366"
      }
    }
  }
}
{
  "interface": {
    "mgmt0": {
      "statistics": {
        "in-unicast-pkts": "4693"
      }
    }
  }
}
{
  "interface": {
    "mgmt0": {
      "statistics": {
        "out-octets": "2736287"
      }
    }
  }
}
.
.
.
```

4.7.4 gnmi_capabilities example

The capabilities gNMI-RPC allows you to discover the capabilities of a specific gNMI server. The following example shows a request to obtain model, data encodings, and version for a specified server.

Example: Request server capabilities for specified server

```
gnmi_capabilities   -username admin -password admin --target_addr [172.18.0.8]:50264
```

Response (request server capabilities)

```
supported_models: <
 name: "urn:srl_nokia/aaa:srl_nokia-aaa"
 organization: "Nokia"
 version: "2020-12-31"
>

supported_models: <
 name: "urn:srl_nokia/aaa-types:srl_nokia-aaa-types"
 organization: "Nokia"
 version: "2019-11-30"
>
.
.
.
supported_encodings: JSON_IETF
supported_encodings: ASCII
supported_encodings: 45
supported_encodings: 44
supported_encodings: 46
supported_encodings: 47
gNMI_version: "0.7.0"
```

5 JSON interface

The SR Linux provides a JSON-based Remote Procedure Call (RPC) for both CLI commands and configuration. The JSON API allows the operator to retrieve and set the configuration and state, and provide a response in JSON format. This JSON-RPC API models the CLI implemented on the system.

If output from a command cannot be displayed in JSON, the text output is wrapped in JSON to allow the application calling the API to retrieve the output. During configuration, if a TCP port is in use when the JSON-RPC server attempts to bind to it, the commit fails. The JSON-RPC supports both normal paths, as well as XPATHs.

5.1 JSON message structure

The JSON RPC requires a specific message structure. The jsonrpc version, ID, method, and params are required in all requests. For example:

```
{
"jsonrpc": "2.0",
"id": 0,
"method": "get",
"params": {
}
}
```

Within these required elements can be additional mandatory and conditional elements, as described in the following table.

Table 13: Required JSON request structure

Required request elements	Description
jsonrpc	Version, which must be “2.0”. No other JSON RPC versions are currently supported.
id	Client-provided integer. The JSON RPC responds with the same ID, which allows the client to match requests to responses when there are concurrent requests.
method	Defines the method accessed with the JSON RPC. Supported options are get, set, and cli.
params	Defines a container for any parameters related to the request. The type of parameter is dependent on the method used.

Related topics
method options
params options

5.1.1 method options

The following table defines supported JSON RPC method options.

Table 14: JSON RPC method options

Method option	Description
get	Used to retrieve configuration and state details from the system. The get method can be used with candidate, running, and state datastores, but cannot be used with the tools datastore.
set	Used to set a configuration or run operational transaction. The set method can be used with the candidate and tools datastores.
validate	Used to verify that the system accepts a configuration transaction before applying it to the system.
cli	Used to run CLI commands. The get and set methods are restricted to accessing data structures via the YANG models, but the cli method can access any commands added to the system via python plug-ins or aliases.

5.1.2 params options

The following table defines valid JSON RPC params options.

Table 15: JSON RPC params options

params option	Descriptions
commands - Mandatory. List of commands used to execute against the called method. Multiple commands can be executed with a single request. Supported commands are: • **action** • **path** • **path-keywords** • **datastore** • **recursive**	**action** command - Conditional mandatory; used with the set and validate methods. Supported options are: • **replace** — Replaces the entire configuration within a specific context with the supplied configuration; equivalent to a delete/update. • **update** — Updates a leaf or container with the specified value. • **delete** — Deletes a leaf or container. All children beneath the parent are removed from the system. Note: When the action command is used with the tools datastore, update is the only supported option.

params option	Descriptions
• **include-field-defaults**	**path** command - Mandatory with the get, set and validate methods. This value is a string that follows the gNMI path specification[1] in human-readable format: • "/" separates nodes • keys are specified using [<key-name>=<key-value>] • if key-value contains "]", it must be escaped ("\" before the "]" • fields end with "." or ",value>" Example: /interface[name=mgmt0]
	path-keywords command - Optional; used to substitute named parameters with the path field. More than one keyword can be used with each path.
	datastore command - Optional; selects the datastore to perform the method against. Supported options are: • **candidate** — Used to change the configuration of the system with the get, set, and validate methods; default datastore is used if the datastore parameter is not provided. • **running** — Used to retrieve the active configuration with the get method. • **state** — Used to retrieve the running (active) configuration along with the operational state. • **tools** — Used to perform operational tasks on the system; only supported with the update action command and the set method.
	recursive command - Optional; a Boolean used to retrieve children underneath the specific path. The default = true.
	include-field-defaults command - Optional; a Boolean used to show all fields, regardless if they have a directory configured or are operating at their default setting. The default = false.
output-format - Optional. Defines the output format as: • json • text • table	Output defaults to JSON if not specified.

5.2 JSON responses

The JSON RPC returns one entry for each command that was executed. For methods that contain non-response (other than acknowledging the command), a response is returned, but with an empty list.

1 gNMI path specification reference: https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-path-conventions.md

When the cli method is used, each executed command returns an individual response.

5.3 Candidate mode

JSON uses its own private candidate that allows multiple users or services to make simultaneous changes to a configuration.

5.4 Logical expressions

For logical expressions, support is provided for the asterisk ("*") which can be used to reference all keys within a list.

5.5 JSON examples

The following provides JSON examples (both requests and responses) for these method options:

- JSON get examples
- JSON set examples
- JSON delete example
- JSON validate example
- JSON CLI example

5.5.1 JSON get examples

The get method allows you to retrieve configuration and state details from the system. The following examples are shown:

- get method using the path, datastore, and recursive commands with the recursive option set to true (to retrieve children underneath the specific path)
- multiple get request where multiple commands are executed with a single request

Example: Single get with recursive request

```
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "get",
  "params": {
    "commands": [
      {
        "path": "/interface[name=mgmt0]",
        "datastore": "state",
        "recursive": true
      }
    ]
  }
}
```

Response (single get with recursive)

```
{
  "result": [
    {
      "name": "mgmt0",
      "admin-state": "enable",
      "mtu": 1514,
      "ifindex": 524304383,
      "oper-state": "up",
      "last-change": "2019-07-12T16:53:39.291Z",
      "statistics": {
        "in-octets": "4545395",
        "in-unicast-pkts": "1178",
        "in-broadcast-pkts": "130",
        "in-multicast-pkts": "27560",
        "in-discards": "0",
        "in-errors": "0",
        "in-unknown-protos": "0",
        "in-fcs-errors": "0",
        "out-octets": "1735990",
        "out-unicast-pkts": "1125",
        "out-broadcast-pkts": "38",
        "out-multicast-pkts": "9187",
        "out-discards": "0",
        "out-errors": "0",
        "carrier-transitions": "1"
      },
      "ethernet": {
        "hw-mac-address": "02:42:AC:12:00:05",
        "statistics": {
          "in-mac-control-frames": "0",
          "in-mac-pause-frames": "0",
          "in-oversize-frames": "0",
          "in-jabber-frames": "0",
          "in-fragment-frames": "0",
          "in-crc-errors": "0",
          "out-mac-control-frames": "0",
          "out-mac-pause-frames": "0"
        }
      },
      "subinterface": [
        {
          "index": 0,
          "admin-state": "enable",
          "ip-mtu": 1500,
          "ifindex": 524288000,
          "oper-state": "up",
          "last-change": "2019-07-12T16:53:39.291Z",
          "ipv4": {
            "allow-directed-broadcast": false,
            "dhcp-client": true,
            "address": [
              {
                "ip-prefix": "172.18.0.5/24",
                "origin": "dhcp"
              }
            ],
            "srl_nokia-interfaces-nbr:arp": {
              "timeout": 14400,
              "neighbor": [
                {
                  "ipv4-address": "172.18.0.1",
                  "link-layer-address": "02:42:90:06:D7:26",
                  "origin": "dynamic",
                  "expiration-time": "2019-07-15T21:37:28.987Z"
                },
                {
                  "ipv4-address": "172.18.0.2",
                  "link-layer-address": "02:42:AC:12:00:02",
                  "origin": "dynamic",
                  "expiration-time": "2019-07-16T00:44:17.673Z"
                },
                {
                  "ipv4-address": "172.18.0.3",
                  "link-layer-address": "02:42:AC:12:00:03",
                  "origin": "dynamic",
                  "expiration-time": "2019-07-16T01:20:48.600Z"
                },
                {
                  "ipv4-address": "172.18.0.4",
                  "link-layer-address": "02:42:AC:12:00:04",
                  "origin": "dynamic",
                  "expiration-time": "2019-07-16T01:20:48.597Z"
                },
                {
                  "ipv4-address": "172.18.0.6",
                  "link-layer-address": "02:42:AC:12:00:06",
                  "origin": "dynamic",
                  "expiration-time": "2019-07-16T01:20:48.599Z"
                }
              ]
            }
          },
          "ipv6": {
            "dhcp-client": true,
            "address": [
              {
                "ip-prefix": "2001:172:18::5/80",
                "origin": "dhcp",
                "status": "preferred"
              },
              {
                "ip-prefix": "fe80::42:acff:fe12:5/64",
                "origin": "link-layer",
                "status": "preferred"
              }
            ],
            "srl_nokia-interfaces-nbr:neighbor-discovery": {
              "dup-addr-detect": true,
              "reachable-time": 30,
              "stale-time": 14400
            }
          }
        }
      ]
    }
  ],
  "id": 0,
  "jsonrpc": "2.0"
}
```

Example: Multiple get request

```
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "get",
  "params": {
    "commands": [
      {
        "path": "/interface[name=mgmt0]",
        "datastore": "state",
        "recursive": false
      },
      {
        "path": "/interface[name=ethernet-1/10]",
        "datastore": "state",
        "recursive": false
      }
    ]
  }
}
```

Response (multiple get)

```
{
  "result": [
    {
      "name": "mgmt0",
      "admin-state": "enable",
      "mtu": 1514,
      "ifindex": 524304383,
      "oper-state": "up",
      "last-change": "2019-07-12T16:53:39.291Z"
    },
    {
      "name": "ethernet-1/10",
      "description": "dut2-dut4-2",
      "admin-state": "enable",
      "mtu": 9232,
      "ifindex": 180223,
      "oper-state": "up",
      "last-change": "2019-07-12T16:54:15.602Z"
    }
  ],
  "id": 0,
  "jsonrpc": "2.0"
}
```

5.5.2 JSON set examples

The set method allows you to set a configuration or run operational commands. The following examples are shown:

- multiple set method using update and replace
- set method using **path-keywords**
- set method using an alternative update

Example: Multiple set with update and replace request

```
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "set",
  "params": {
     "commands": [
        {
           "action": "update",
           "path": "/interface[name=mgmt0]/description:my-description"
        },
        {
           "action": "replace",
           "path": "/interface[name=mgmt0]/subinterface[index=0]/description:my-subdescription"
        }
     ]
  }
}
```

Response (multiple set)

```
{
  "result": {},
  "id": 0,
  "jsonrpc": "2.0"
}
```

Example: set using path keywords request

```
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "set",
  "params": {
    "commands": [
      {
        "action": "update",
        "path": "/interface[name={name}]/description:my-description",
        "path-keywords": {
          "name": "mgmt0"
        }
      },
      {
        "action": "replace",
        "path": "/interface[name={name}]/subinterface[index={index}]/description:my-
subdescription",
        "path-keywords": {
          "name": "mgmt0",
          "index": "0"
        }
      }
    ]
  }
}
```

Response (set using path-keywords)

```
{
  "result": {},
  "id": 0,
  "jsonrpc": "2.0"
}
```

Example: set using alternative update (specifying a value) request

```
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "set",
  "params": {
    "commands": [
      {
        "action": "update",
        "path": "/interface[name=mgmt0]",
        "value": {
          "description": "my-description",
          "subinterface": {
            "index": "0",
            "description": "my-subdescription"
          }
        }
      }
    ]
  }
}
```

Response (set using alternative update)

```
{
  "result": {},
  "id": 0,
  "jsonrpc": "2.0"
}
```

5.5.3 JSON delete example

The set method allows you to use an action delete command to delete nodes or leafs within a configuration.

The following example shows a multiple set delete request to delete the specified paths.

Example: Multiple set method delete request

```
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "set",
  "params": {
    "commands": [
      {
        "action": "delete",
        "path": "/interface[name=mgmt0]/description"
      },
      {
        "action": "delete",
        "path": "/interface[name=mgmt0]/subinterface[index=0]/description"
      }
    ]
  }
}
```

Response (multiple set delete)

```
{
  "result": {},
  "id": 0,
  "jsonrpc": "2.0"
}
```

5.5.4 JSON validate example

The validate method allows you to verify that the system will accept a configuration transaction before applying it to the system. The ‘delete’, ‘replace’, and ‘update’ actions can be used with the validate method. The following examples are shown:

- validate request to delete a specified path.
- validate request to update and replace a specified path.

Example: validate a delete request

```
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "validate",
  "params": {
    "commands": [
      {
        "action": "delete",
        "path": "/interface[name=mgmt0]/description"
      }
    ],
    "datastore": "candidate"
  }
}
```

Response (validate delete)

```
{
  "result": {},
  "id": 0,
  "jsonrpc": "2.0"
}
```

Example: validate an update and replace request

```
{
    "jsonrpc": "2.0",
    "id": 0,
    "method": "validate",
    "params": {
        "commands": [
            {
                "action": "update",
                "path": "/interface[name=mgmt0]/description:my-description"
            },
            {
                "action": "replace",
                "path": "/interface[name=mgmt0]/subinterface[index=0]
 /description:my-subdescription"
            }
        ]
    }
}
```

Response (validate update and delete)

```
{
  "result": {},
  "id": 0,
  "jsonrpc": "2.0"
}
```

5.5.5 JSON CLI example

The cli method allows you to run CLI commands. While get and set methods are restricted to accessing data structures in the YANG models, the cli method can access commands that have been added to the system using python plug-ins.

The following examples are shown:

- JSON cli command request.
- JSON cli command requesting the output format as text.

  You can optionally define these output formats: json, text, or table. JSON is the default.

Example: CLI method input request

```
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "cli",
  "params": {
    "commands": [
      "enter candidate",
      "info interface mgmt0"
    ]
  }
}
```

Response (CLI input)

```
{
  "result": [
    {},
    {
      "interface": [
        {
          "name": "mgmt0",
          "description": "my-description",
          "admin-state": "enable",
          "subinterface": [
            {
              "index": 0,
              "description": "my-subdescription",
              "admin-state": "enable",
              "ipv4": {
                "dhcp-client": true
              },
              "ipv6": {
                "dhcp-client": true
              }
            }
          ]
        }
      ]
    }
  ],
  "id": 0,
  "jsonrpc": "2.0"
}
```

Example: CLI method input request with output formatting

```
{
  "jsonrpc": "2.0",
  "id": 0,
  "method": "cli",
  "params": {
    "commands": [
      "enter candidate",
      "info interface mgmt0"
    ],
    "output-format": "text"
  }
}
```

Response (CLI with output formatting)

```
{
  "result": [
    "",
    "    interface mgmt0 {\n      description my-description\n
admin-state enable\n              subinterface 0 {\n
description my-subdescription\n   admin-state enable\n       ipv4 {\n
dhcp-client true\n                }\n                        ipv6 {\n
dhcp-client true\n                }\n                        }\n      }\n"
  ],
  "id": 0,
  "jsonrpc": "2.0"
}
```

6 Configuring SNMP

The SR Linux device supports SNMPv2. To allow an SNMP client to read information about the system as an aid in monitoring the device, the SR Linux supports the OIDs listed in Supported SNMP OIDs. The MIB file that covers these OIDs is packaged with each release.

Example: SNMP server configuration

In the following example, an SNMP server is running within the mgmt and default network-instances, and the configuration specifies the IP addresses where the device listens for SNMP client connections:

```
--{ candidate shared default }--[  ]--
# info system snmp
    system {
        snmp {
            community test1 {
                permission r
                version v2c
            }
            network-instance mgmt {
                admin-state enable
                source-address [
                    1.1.1.1
                ]
            }
            network-instance default {
                admin-state enable
                source-address [
                    3.3.3.3
                ]
            }
        }
    }
```

6.1 Supported SNMP OIDs

The SR Linux supports the following OIDs. Details about each OID are listed in the linked sections.

- sysName
- sysObjectId
- sysContact
- sysLocation
- sysDescr
- ifIndex
- ifDescr
- ifOperStatus
- ifAdminStatus
- ifType

- ifMtu
- ifSpeed
- ifPhysAddress
- ifName
- ifHCInOctets
- ifHCOutOctets
- ifHighSpeed
- ifPromiscuousMode
- ifConnectorPresent
- ifAlias

6.1.1 sysName

The sysName OID is located within the SNMP MIB-2 subtree, specifically at 1.3.6.1.2.1.1.5. This is generally the device FQDN. The SR Linux device reads the sysName from the configured system host name.

If the host name does not contain a dot (.) character, the configured domain name is also read, and is appended to the host name before being returned to the client.

6.1.2 sysObjectId

The sysObjectId is located within the SNMP MIB-2 subtree, specifically at 1.3.6.1.2.1.1.2. This identifies the kind of device being managed. SR Linux uses the subtree at 1.3.6.1.4.1.6527.

6.1.3 sysContact

The sysContact OID is located within the SNMP MIB-2 subtree, specifically at 1.3.6.1.2.1.1.4. This identifies the contact person for this managed node, together with information about how to contact this person. SR Linux propagates the information configured for /system/information/contact for the sysContact OID.

6.1.4 sysLocation

The sysLocation OID is located within the SNMP MIB-2 subtree, specifically at 1.3.6.1.2.1.1.6. This identifies the physical location of the device. SR Linux propagates the information configured for `/system/information/location` for the sysLocation OID.

6.1.5 sysDescr

The sysDescr OID is located within the SNMP MIB-2 subtree, specifically at 1.3.6.1.2.1.1.1. This value can include the name and version for the system hardware, software operating system, and networking software. This must contain only printable ASCII characters.

The SR Linux uses a fixed sysDescr field, based on this template:

```
<SRLinux-version (from IDB)> Nokia <Platform Type> right (c) 2000-2019 Nokia.
Kernel <kernel-version> <Linux-version> <SNMP-version> <kernel build date>.
All rights reserved. All use subject to applicable license agreements.
```

6.1.6 ifIndex

The ifIndex is located within the SNMP IF-MIB ifEntry subtree, specifically at 1.3.6.1.2.1.2.2.1.1. This is a unique value, greater than zero, for each interface. SR Linux propagates the `/interface[]/ifindex` leaf into the ifIndex OID. One ifEntry is created for each ifIndex found in the system. If an interface does not have an ifIndex, it is not populated into SNMP. An ifIndex is generated only for physical interfaces (excluding management).

6.1.7 ifDescr

The ifDescr OID is located within the SNMP IF-MIB ifEntry subtree, specifically at 1.3.6.1.2.1.2.2.1.2. This is a string containing information about the interface. This string can include the name of the manufacturer, product name, and version of the interface hardware/software. SR Linux propagates the `/interface[]/description` leaf into the ifDescr OID. If the description passed from `/interface[]/description` exceeds 255 characters, it is truncated.

6.1.8 ifOperStatus

The ifOperStatus OID is located within the SNMP IF-MIB ifEntry subtree, specifically at 1.3.6.1.2.1.2.2.1.8. This indicates the current operational state of the interface. SR Linux propagates the `/interface[]/oper-state` leaf into the ifOperStatus OID using the mapping shown in the following table.

Table 16: Mapping between ifOperStatus OID and /interface[]/oper-state leaf

ifOperStatus OID value	ID	/interface[]/oper-state enum	/interface[]/oper-state value
up	1	up	1
down	2	down	2
testing	3		
unknown	4		
dormant	5		
notPresent	6	empty	3
lowerLayerDown	7	down	2

6.1.9 ifAdminStatus

The ifAdminStatus is located within the SNMP IF-MIB ifEntry subtree, specifically at 1.3.6.1.2.1.2.2.1.7. This is the desired state of the interface. The testing (3) state indicates that no operational packets can be passed. When a managed system initializes, all interfaces start with ifAdminStatus in the down (2) state. Based on either explicit management action or configuration information retained by the managed system, ifAdminStatus is then changed to either the up (1) or testing (3) states (or remains in the down(2) state).

SR Linux propagates the `/interface[]/admin-state` leaf into the ifAdminStatus OID using the mapping shown in the following table.

Table 17: Mapping between ifAdminStatus OID and /interface[]/admin-state leaf

ifAdminStatus OID value	ID	/interface[]/admin-state enum	/interface[]/admin-state value
up	1	enable	1
down	2	disable	2
testing	3		

6.1.10 ifType

The ifType OID is located within the SNMP IF-MIB ifEntry subtree, specifically at 1.3.6.1.2.1.2.2.1.3. This indicates the type of interface. SR Linux propagates the ifType based on the interface name using the mapping shown in the following table.

Table 18: Mapping between ifType OID and /interface leaf

ID	ifType OID value	/interface[name=<>] value	/interface[]/subinterface[]/type value
6	ethernetCsmacd	ethernet-*	
6	ethernetCsmacd	*	bridged
24	softwareLoopback	lo*	
142	ipForward	*	routed
161	ieee8023adLag	lag*	

6.1.11 ifMtu

The ifMtu is located within the SNMP IF-MIB ifEntry subtree, specifically at 1.3.6.1.2.1.2.2.1.4. This indicates the size of the largest packet that can be sent/received on the interface, specified in octets.

When used with Ethernet/bridged interfaces, this includes Ethernet overhead, but does not include the 4-byte FCS. When used with routed interfaces, this includes the IP header, but excludes any Ethernet encapsulation.

SR Linux propagates the `/interface[]/mtu` leaf into the ifMtu OID when dealing with physical or top level interfaces. For subinterfaces, the `/interface[]/subinterface[]/l2-mtu` should be used for type=bridged, and `/interface[]/subinterface[]/l3-mtu` should be used for type=routed.

6.1.12 ifSpeed

The ifSpeed OID is located within the SNMP IF-MIB ifEntry subtree, specifically at 1.3.6.1.2.1.2.2.1.5. This is an estimate of the interface's current bandwidth in bits per second. For interfaces that do not vary in bandwidth, or for those where no accurate estimation can be made, this object contains the nominal bandwidth.

If the bandwidth of the interface is greater than the maximum value reportable by this object, then this object reports its maximum value (4,294,967,295), and ifHighSpeed is used to report the interface's speed using the mapping shown in the following table.

Table 19: Mapping between ifSpeed OID, ifHighSpeed, and port-speed leaf

ifSpeed OID value	ifHighSpeed OID value	/interface[]/ethernet/port-speed value
10000000	10	10M
100000000	100	100M
1000000000	1000	1G
4294967295	10000	10G
4294967295	25000	25G
4294967295	40000	40G
4294967295	50000	50G
4294967295	100000	100G
4294967295	200000	200G
4294967295	400000	400G
4294967295	800000	800G
4294967295	1000000	1T

6.1.13 ifPhysAddress

The ifPhysAddress OID is located within the SNMP IF-MIB ifEntry subtree, specifically at 1.3.6.1.2.1.2.2.1.6.

This is the interface address at its protocol sublayer. For example, for an 802.x interface, this object normally contains a MAC address. The interface's media-specific MIB defines the bit and byte ordering and the format of the value of this object. For interfaces that do not have such an address (for example, a serial line), this object contains an octet string of zero length.

SR Linux propagates the `/interface[]/ethernet/hw-mac-address` leaf into the ifPhysAddress OID, with transformations to conform to the SNMPv2 IF-MIB. The transformations are to remove the ":" separators. For example, the MAC address 00:25:ba:51:c7:0a, is transformed into 0025ba51c70a. For subinterfaces, the parent interface MAC is used, and for any interfaces which do not use MAC addresses, the field contains an octet string of zero length.

6.1.14 ifName

The ifName OID is located within the SNMP IF-MIB2 ifXEntry subtree, specifically at 1.3.6.1.2.1.31.1.1.1.1.

This is the textual name of the interface. The value of this object is the name of the interface as assigned by the local device, and is suitable for use in commands entered at the device's console. This can be a text name, such as le0 or a simple port number, such as 1, depending on the interface naming syntax of the device. If several entries in the ifTable together represent a single interface as named by the device, then each has the same value of ifName. Note that for an agent that responds to SNMP queries concerning an interface on some other (proxied) device, then the value of ifName for such an interface is the proxied device's local name for it.

If there is no local name, or this object is otherwise not applicable, then this object contains a zero-length string.

SR Linux propagates `/interface[]/name` (for physical or top-level interfaces) leaf or for subinterfaces the `/interface[]/subinterface[]/name` leaf into the ifName OID.

6.1.15 ifHCInOctets

The ifHCInOctets OID is located within the SNMP IF-MIB2 ifXEntry subtree, specifically at 1.3.6.1.2.1.31.1.1.1.6.

This is the total number of octets received on the interface, including framing characters. This object is a 64-bit version of ifInOctets.

Discontinuities in the value of this counter can occur at reinitialization of the management system, and at other times as indicated by the value of ifCounterDiscontinuityTime.

SR Linux propagates `/interface[]/statistics/in-octets` (for physical or top-level interfaces) leaf or for subinterfaces the `/interface[]/subinterface[]/statistics/in-octets` leaf into the ifHCInOctets OID.

6.1.16 ifHCOutOctets

The ifHCOutOctets OID is located within the SNMP IF-MIB2 ifXEntry subtree, specifically at 1.3.6.1.2.1.31.1.1.1.10.

This is the total number of octets transmitted out of the interface, including framing characters. This object is a 64-bit version of ifOutOctets.

Discontinuities in the value of this counter can occur at reinitialization of the management system, and at other times as indicated by the value of ifCounterDiscontinuityTime.

SR Linux propagates `/interface[]/statistics/out-octets` (for physical or top-level interfaces) leaf, or for subinterfaces the `/interface[]/subinterface[]/statistics/out-octets` leaf, into the ifHCOutOctets OID.

6.1.17 ifHighSpeed

The ifHighSpeed OID is located within the SNMP IF-MIB2 ifXEntry subtree, specifically at 1.3.6.1.2.1.31.1.1.1.15.

An estimate of the interfaces current bandwidth in units of 1,000,000 bits per second. If this object reports a value of `n then the speed of the interface is somewhere in the range of `n-500,000 to `n+499,999. For interfaces that do not vary in bandwidth, or for those where no accurate estimation can be made, this object contains the nominal bandwidth. For a sublayer that has no concept of bandwidth, this object is zero.

See Table 19: Mapping between ifSpeed OID, ifHighSpeed, and port-speed leaf for the use of ifHighSpeed.

6.1.18 ifPromiscuousMode

The ifPromiscuousMode is located within the SNMP IF-MIB2 ifXEntry subtree, specifically at 1.3.6.1.2.1.31.1.1.1.16.

This object has a value of false (2) if this interface only accepts packets/frames that are addressed to this station. This object has a value of true (1) when the station accepts all packets/frames transmitted on the media. The value true (1) is only legal for specific types of media. If legal, setting this object to a value of true (1) may require the interface to be reset before becoming effective.

The value of ifPromiscuousMode does not affect the reception of broadcast and multicast packets/frames by the interface.

This field is set to false (2) for all interfaces in SR Linux.

6.1.19 ifConnectorPresent

The ifConnectorPresent OID is located within the SNMP IF-MIB2 ifXEntry subtree, specifically at 1.3.6.1.2.1.31.1.1.1.17.

This object has the value true (1) if the interface sublayer has a physical connector and the value false (2) otherwise.

This field is set to false (2) for all non-physical interfaces. For physical interfaces with non-removable optics, this is set to true (1), and for physical interfaces with optics present, this is also set to true (1). A physical interface without an optic is set to false (2).

6.1.20 ifAlias

The ifAlias OID is located within the SNMP IF-MIB2 ifXEntry subtree, specifically at 1.3.6.1.2.1.31.1.1.1.18.

This object is an alias name for the interface as specified by a network manager, and provides a non-volatile identifier for the interface.

On the first instantiation of an interface, the value of ifAlias associated with that interface is the zero-length string. When a value is written into an instance of ifAlias through a network management set operation, the agent retains the supplied value in the ifAlias instance associated with the same interface for as long as that interface remains instantiated, including across all re-initializations/reboots of the network management system, including those that result in a change of the interface's ifIndex value.

An example of the value that a network manager may store in this object for a WAN interface is the circuit number/identifier of the interface.

Some agents may support write access only for interfaces having particular values of ifType. An agent that supports write access to this object is required to keep the value in non-volatile storage, but it may limit the length of new values depending on how much storage is already occupied by the current values for other interfaces.

SR Linux propagates the `/interface[]/description` (for physical or top-level interfaces) leaf, or for subinterfaces the `/interface[]/subinterface[]/description` leaf, into the ifAlias OID.

7 General and operational commands

The following table defines general and operational commands that can be entered at any point in the CLI.

Table 20: General and operational commands

Command	Description
Tab	Auto-completes a command
/	Moves to the root; can also be used to reset the context to the root for a specific command (for example: **info from state /**)
?	Displays context-based help
back	Returns to the context before executing the last command
baseline <*argument*>	Arguments: • check - check baseline update for current candidates • diff - show baseline configuration changes • update - update baseline for current candidate
bash	Opens a bash session
bash <*command*>	Executes a command without entering a bash session
cls	Clears the screen
diff [flat]	Compares the current candidate against the running configuration. Specifying **flat** provides a copy/paste format showing inserts and deletes. Global arguments include: • **baseline** - configuration candidate baseline • **candidate** - configuration candidate • **checkpoint** - configuration checkpoint • **factory** - factory configuration • **file** - file with configuration stored in .json format • **from** - source datastore to compare with • **rescue** - rescue configuration • **running** - running datastore • **startup** - startup configuration If arguments are not used, this command must be used in candidate mode.

Command	Description
echo	Echoes text back to a session
enter	Switches to a different mode
enter candidate	Enters the shared candidate mode. Shared candidate mode is the default.
enter candidate exclusive	Enters the exclusive candidate mode To switch between shared and exclusive, you must switch to a different datastore (for example. running).
enter candidate exclusive name *<name>*	Enters the exclusive mode for a named candidate
enter candidate name *<name>*	Enters the shared candidate mode for a named shared candidate
enter candidate private	Enters the candidate private mode
enter candidate private name *<name>*	Enters the candidate private mode for a named candidate
enter running	Enters the running mode (default)
enter show	Enters the show mode (used with show CLI plug-ins)
enter state	Enters the state mode (all configuration and operational states)
environment	Configures and displays environment variables
environment alias	Creates or overwrites an alias
environment bottom-toolbar	Changes the text displayed in the bottom toolbar
environment cli-engine type	Sets cli engine type for interactive logins: • basic • advanced
environment complete-on-space	Triggers auto-completion when a space is typed (default is to explicitly require a <TAB>)
environment delete	Resets and removes environment settings
environment key-completer-limit *<limit>*	Number of keys limited in auto-completion
environment load	Loads the environment settings from a file: • file (path of the configuration file)

Command	Description
	• home (use `~/.srlinuxrc` configuration file)
environment output-format	Allows the default output format to change between text and JSON
environment prompt	Changes the prompt displayed before every input line
environment save	Saves the current environment
environment save home	Saves the current environment to the user home directory
environment save file *<file>*	Saves the current environment to a specific file
environment show	Shows the currently active environment settings
exit	Exits to a previous context
exit to *<ancestor>*	Exits to a specific ancestor of the current context
exit all	Exits to the root
filter	Filters output for show and info commands. Usage: `filter [<node>] [depth <value>] [fields <value>] [keys-only] [non-zero]` • **node** - Positional node to filter on • **depth** - Filter out sub-nodes that are more than *n* levels deeper • **fields** - Fields to include • **keys-only** - Hide all fields • **non-zero** - Show only non-zero values (numeric values only)
help	Displays mode-related help
history	Displays the command history list with line numbers
history hot	Displays the top 5 most frequently used commands
history clear	Clears the history
info [*path***]**	Shows the value of all nodes and fields under the current context; optionally made more specific by a path
info depth *<n>*	Filters out sub-nodes that are deeper than the specified depth
info detail	Shows also default values for unset fields

Command	Description
info flat	Shows each node or field as a single line
info use-proto-json	Shows the output as the JSON used for the protobuf messages
info from *<mode>*	Executes the info command from within the specified mode (either candidate, running, or state). The current context can be retrieved without a from argument.
list	Show the keys of all nodes under the current context
monitor [*path***]**	Monitors state changes within the current context; optionally made more specific by a path
monitor recursive [*path***]**	Includes children when monitoring
monitor sample	Uses sampling instead of on-change monitoring (interval in seconds)
ping	Sends IPv4 ICMPv4 echo requests to network hosts. Usage: **ping6** *<destination>* **[-I** *<value>***] [-M** *<value>***] [-Q** *<value>***] [-c** *<value>***] [-i** *<value>***] [-s** *<value>***] [-t** *<value>***] [network-instance** *<value>***]** • **-I** - Source interface/IP • **-M** - Path MTU discovery strategy • **-M- pmtudisc_options** - sets df-bit set • **-Q** - tos • **-c** - Number of ping requests • **-i** - Wait interval in seconds between each packet • **-s** - Packet size • **-t** - TTL (Time-To-Live)
ping6	Sends IPv6 ICMPv6 echo requests to network hosts. Usage: **ping6** *<destination>* **[-I** *<value>***] [-M** *<value>***] [-Q** *<value>***] [-c** *<value>***] [-i** *<value>***] [-s** *<value>***] [-t** *<value>***] [network-instance** *<value>***]** • **-I** - Source interface/IP • **-M** - Path MTU discovery strategy • **-M- pmtudisc_options** - sets df-bit set • **-Q** - tos • **-c** - Number of ping requests • **-i** - Wait interval seconds between each packet • **-s** - Packet size

Command	Description
	• **-t** - TTL (Time-To-Live)
pwc	Prints the current working context
quit	Closes the CLI session
save	Saves the current datastore to the specified file in JSON format. Usage: **save [detail] file** *<value>* **[from <running|state>] [text]** • **detail** - Additionally saves the default values for unset fields • **file** - Output filename • **from** - Datastore used to retrieve data from • **text** - Use CLI (text) format instead of JSON
show	Displays plug-in style show commands; seePre-defined show reports.
source *<file>*	Executes a set of commands from a file
tech-support	Generates a technical support file. Usage: **tech-support [ignore-host-keys <***value***>] [max-time <***value***>] [network-instance <***value***>] [no-core] [scp-to <***value***>]** • **ignore-host-keys** - Skip security verification of remote SSH server key • **max-time** - Maximum time • **network-instance** - Network-instance for scp command • **no-core** - Skip inclusion of core files • **scp-to** - scp location, for example, user@server-ip:/tmp/
tools	Executes a tool command
traceroute	Prints the route packets trace to network host Usage: **traceroute** *<destination>*
traceroute6	Prints the route IPv6 packets trace to network host Usage: **traceroute6** *<destination>*
tcptraceroute	tcptraceroute compatible wrapper for traceroute Usage: **tcptraceroute** *<destination>*
tree [*path***]**	Shows the tree structure in the current context; optionally made more specific with a path

Command	Description
tree flat	Shows each structure on a single line
tree from [*mode*]	Retrieves the tree from the current context in another mode
watch	Execute a program periodically
Within candidate mode only, the following apply	
!!	Appends a line to the annotation of the current node
!!!	Replaces the annotation of the current node
commit now	Applies the changes, exits candidate mode, and enters running mode
commit stay	Applies the changes and then remains in candidate mode. Permitted additional arguments: commit stay [save] [comment] [confirmed]
commit save	Applies the changes and then remains in candidate mode. Permitted additional arguments: commit [stay] [checkpoint] save [confirmed] [comment]
commit checkpoint	Causes an automatic checkpoint after the commit succeeds. Permitted additional arguments: commit [stay] [now] checkpoint [save] [confirmed]
commit validate	Verifies that a propose configuration change passes a management server validation
commit comment <*comment*>	Used with other arguments (except **validate**) to add a user comment where comment is a quoted string, 1-255 characters. Permitted additional arguments: commit [stay] [save] [checkpoint] [confirmed] comment
commit confirmed **commit confirmed [timeout** <*1-86400*>**]** **commit confirmed [accept|reject]**	Applies the changes, but requires an explicit confirmation to become permanent. The timeout period default is 600 seconds (10 mins.), or can be provisioned with a value of 1-86400 sec.). The timeout argument cannot be used with the accept or reject parameter. Before the timer expires, the accept argument explicitly confirms and applies the changes. With no timer running, the reject argument explicitly rejects the changes.
annotate	Sets the annotation of the current node

Command	Description
discard [now|stay]	Discards all uncommitted changes; requires either a **now** or **stay** option (to stop unintended commit). When **stay** is used, the mode remains in candidate mode (opening a new transaction).

8 Pre-defined show reports

The SR Linux CLI is a python application that can load dynamic libraries from other applications.This flexibility allows users to create their own show commands, and also allows each application to own its own show command tree. In addition to custom show commands, Nokia provides pre-defined plug-ins.

The following sections provide the commands and syntax for pre-defined python plug-ins.

Users can also create their own customer reports using CLI Plug-ins. For more information, see the *SR Linux CLI Plug-In Guide*.

8.1 ACL show reports

```
show
    — acl
        — capture-filter <filter name>
            — ipv4-filter <filter name>
                — entry <value>
            — ipv6-filter <filter name>
                — entry <value>
        — cpm-filter <filter name>
            — ipv4-filter <filter name>
                — entry <value>
            — ipv6-filter <filter name>
                — entry <entry number>
        — ipv4-filter <filter name>
            — entry <value>
            — subinterface <subinterface value>
        — ipv6-filter <filter name>
            — entry <value>
            — subinterface <subinterface value>
        — summary
```

8.1.1 ACL descriptions

show

Context	show
Tree	show
Description	Container for show reports.

acl

Context	show acl
Tree	acl
Description	Container for ACL show reports.

capture-filter

Context	show acl capture-filter *filter type*
Tree	capture-filter
Description	Show a list of ACL capture filters. Specify either ipv4-filter or ipv6-filter.
Configurable	False

ipv4-filter

Context	show acl capture-filter ipv4-filter
Tree	ipv4-filter
Description	Show a list of ACL capture IPv4 filters.

entry

Context	show acl capture-filter ipv4-filter entry *number*
Tree	entry
Description	Show a specific ACL capture IPv4 filter.

ipv6-filter

Context	show acl capture-filter ipv6-filter
Tree	ipv6-filter
Description	Show a list of ACL capture IPv6 filters.

entry

Context	show acl capture-filter ipv6-filter entry *number*
Tree	entry
Description	Show a specific ACL capture IPv6 filter.

cpm-filter

Context	show acl cpm-filter *filter type*
Tree	cpm-filter
Description	Show a list of ACL CPM filters. Specify either ipv4-filter or ipv6-filter.

ipv4-filter

Context	show acl cpm-filter ipv4-filter
Tree	ipv4-filter
Description	Show a list of ACL CPM IPv4 filters.

entry

Context	show acl cpm-filter ipv4-filter entry *number*
Tree	entry
Description	Show a specific ACL CPM IPv4 filter.

ipv6-filter

Context	show acl cpm-filter ipv6-filter
Tree	ipv6-filter
Description	Show a list of ACL CPM IPv6 filters.

entry

Context	show acl cpm-filter ipv6-filter entry *number*
Tree	entry
Description	Show a specific ACL CPM IPv6 filter.

ipv4-filter

Context	show acl ipv4-filter *name*
Tree	ipv4-filter
Description	Show ACL IPv4 filter policies.
Configurable	False

entry

Context	show acl ipv4-filter *name* entry *number*
Tree	entry
Description	Show ACL IPv4 filter policies by entry ID.

subinterface

Context	show acl ipv4-filter *name* subinterface *string*
Tree	subinterface
Description	Show ACL IPv4 filter policies by subinterface.

ipv6-filter

Context	show acl ipv6-filter *name*
Tree	ipv6-filter
Description	Show ACL IPv6 filter policies.
Configurable	False

entry

Context	show acl ipv6-filter *name* entry *number*
Tree	entry
Description	Show ACL IPv6 filter policies by entry ID.

subinterface

Context	show acl ipv6-filter *name* subinterface *string*
Tree	subinterface
Description	Show ACL IPv6 filter policies by subinterface.

summary

Context	show acl summary
Tree	summary
Description	Show a summarized list of all ACL filters (capture, CPM, IPv4, and IPv6).

8.2 ARPND show reports

```
show
    — arpnd
        — neighbors
            — interface <interface name>
        — arp-entries
            — interface <interface name>
```

8.2.1 ARPND descriptions

show

Context	show
Tree	show
Description	Container for show reports.

arpnd

Context	show arpnd
Tree	arpnd
Description	Container for ARPND show reports.

neighbors

Context	show arpnd neighbors

Tree	neighbors
Description	Show a report of all ARPND neighbors including associated interfaces, subinterfaces, origin, and state.

interface

Context	show arpnd neighbors interface *interface name*
Tree	interface
Description	Show a ARPND neighbor report for a specified interface name.

arp-entries

Context	show arpnd arp-entries
Tree	arp-entries
Description	Show a report of all ARPND entries including associated interfaces, subinterfaces, origin, and expiry.

interface

Context	show arpnd arp-entries interface *interface name*
Tree	interface
Description	Show a ARPND entries report for a specified interface name.

8.3 Interface show reports

```
show
    — interface <interface port name>
        — brief
        — all
        — detail
        — queue-detail
```

8.3.1 Interface descriptions

show

Context	show
Tree	show
Description	Container for show reports.

interface

Context	show interface *interface port name*
Tree	interface

Description	Container for Interface show reports. Can be used to show a report of all interfaces and subinterfaces with an operational state of up, or for a specified interface if an optional name is entered. Interfaces that are operationally down are not displayed; use the all option to display both up and down interfaces.

brief

Context	show interface *interface port name* brief
Tree	brief
Description	Show a report that provides a summarized view of all interfaces and subinterfaces, or for a specified interface if an optional name is entered. Displays the administrative state, operational state, speed, and type.

all

Context	show interface *interface port name* all
Tree	all
Description	Show a report that displays all up and down interfaces and subinterfaces, or for a specified interface if an optional name is entered. For interfaces with an operational state of up, IP addresses are displayed in addition to speed and type. For interfaces with an operational state of down, a reason for this state is provided.

detail

Context	show interface *interface port name* detail
Tree	detail
Description	Show a detail report for all up and down interfaces and subinterfaces, or for a specified interface if an optional name is entered. In addition to operational state, last changed, flow control and MAC address details, this report includes queue parameters, statistics, and transceiver/transceiver channel details.

queue-detail

Context	show interface *interface port name* queue-detail
Tree	queue-detail
Description	Show an egress queues and VOQs report for all interfaces and subinterfaces, or for a specified interface if an optional name is entered.

8.4 LAG show reports

```
show
    — lag <lag instance>
        — brief
        — detail
        — lacp-state
        — lacp-statistics
        — member-statistics
        — queue-detail
```

8.4.1 LAG descriptions

show

Context	show
Tree	show
Description	Container for show reports.

lag

Context	show lag *lag instance*
Tree	lag
Description	Container for lag show reports. Can be used to show a report of all LAGs, or for a specified LAG instance if an optional ID is entered.This includes a summary report of all operationally up and down LAGs.

brief

Context	show lag *lag instance* brief
Tree	brief
Description	Show a report that provides a summarized view of all LAGs, or for a specified LAG if an optional ID is entered. Displays the administrative state, operational state, aggregate speed, and min and active links.

detail

Context	show lag *lag instance* detail
Tree	detail
Description	Show a detail report for all LAGs, or for a specified LAG if an optional ID is entered. This report includes the operational status of the LAG, list of member links and their operational status, aggregated queue statistics, aggregated traffic/error statistics, and aggregated traffic rate.

lacp-state

Context	show lag *lag instance* lacp-state
Tree	lacp-state
Description	Show a LAG report that details all LAGs or a specific LAG it is enabled. Report also includes the LACP configuration mode and per member link statistics state details (operation state, activity, timeout, and so on.).

lacp-statistics

Context	show lag *lag instance* lacp-statistics

Tree	lacp-statistics
Description	Show a LAG report that details all LAGs or specific LAGs statistics. Report can include per member link statistics such as LACP in-pkts, out-pkts, rx-errors, tx-errors, unknown errors, and LACP errors.

member-statistics

Context	show lag *lag instance* member-statistics
Tree	member-statistics
Description	Show a LAG report that displays all member-statistics.

queue-detail

Context	show lag *lag instance* queue-detail
Tree	queue-detail
Description	Show a LAG report that displays aggregated queue statistics for all LAG link members.

8.5 MPLS show reports

```
show
    — mpls-aft
        — network-instance <instance name>
```

8.5.1 MPLS descriptions

show

Context	show
Tree	show
Description	Container for show reports.

mpls-aft

Context	show mpls-aft
Tree	mpls-aft
Description	Container for MPLS reports. Can be used to show all MPLS abstract forwarding tables.

network-instance

Context	show mpls-aft network-instance *network-instance name*
Tree	network-instance
Description	Show an MPLS abstract forwarding table for a specified network instance.

8.6 Network-instance show reports

```
show
    – network-instance <name>
        – bridge-table
            – mac-duplication duplication-entries
            – mac-table
                – all
                – mac <address>
                – summary
        – interfaces <interface name>
        – protocols
            – bgp
                – neighbor [<IP-address>]
                    – advertised-routes
                        – evpn
                        – ipv4
                        – ipv6
                    – detail
                    – maintenance
                    – received-routes
                        – evpn
                        – ipv4
                        – ipv6
                – routes [<IP family>]
                    – evpn
                        – route-type <route type>
                            – detail
                            – summary
                    – ipv4
                        – prefix <IP prefix>
                        – summary
                    – ipv6
                        – prefix <IP prefix>
                        – summary
                – summary
            – bgp-evpn
                – bgp-instance
            – bgp-vpn
                – bgp-instance
            – isis
                – adjacency <interface_name>
                    – neighbor-system-id <ID>
                        – adjacency-level <value>
                            – detail
                – database
                    – lsp-id <ID>
                    – detail
                – interface <name>
                    – detail
                – hostnames
                    – detail
                – summary
            – ospf
                – area
                    – detail
                – database
                    – type
                – instance <name>
                – interface
                    – detail
                – neighbor
                    – detail
```

```
                  — statistics
                  — status
          — route-table
              — all
              — summary
              — ipv4-unicast
                  — prefix <ID> detail
                  — summary
              — ipv6-unicast
                  — prefix <ID> detail
                  — summary
              — next-hop <index>
          — summary
          — tunnel-table
              — all
              — ipv4 ip address type
              — ipv6 ip address type
          — vxlan-interface <name>
```

8.6.1 Network-instance descriptions

show

Context	show
Tree	show
Description	Container for show reports.

network-instance

Context	show network-instance *name*
Tree	network-instance
Description	Show network-instance for specified name or additional command (bridge-table, interfaces, protocols, route-table, or summary)

bridge-table

Context	show network-instance *name* bridge-table
Tree	bridge-table
Description	Show MAC bridge tables. Specify mac-duplication duplication-entries or mac-table.

mac-duplication duplication-entries

Context	show network-instance *name* bridge-table mac-duplication duplication-entries
Tree	mac-duplication duplication-entries
Description	Show all MAC learned entries.

mac-table

Context	show network-instance *name* bridge-table mac-table
Tree	mac-table

Description	Show a MAC table report. Specify option of all, mac, or summary.

all

Context	show network-instance *name* bridge-table mac-table all
Tree	all
Description	Show all MAC table entries

mac

Context	show network-instance *name* bridge-table mac-table mac *address*
Tree	mac
Description	Show MAC table entry for specific address.

summary

Context	show network-instance *name* bridge-table mac-table summary
Tree	summary
Description	Show a MAC table summary report.

interfaces

Context	show network-instance *name* interfaces *interface name*
Tree	interfaces
Description	Show all network-instance interfaces.

protocols

Context	show network-instance *name* protocols
Tree	protocols
Description	Show network-instance protocol details. Specify bgp, isis, or ospf.

bgp

Context	show network-instance *name* protocols bgp
Tree	bgp
Description	Show bgp status report. Specify neighbor, routes, or summary.

neighbor

Context	show network-instance *name* protocols bgp neighbor *IP address*
Tree	neighbor
Description	Show a bgp neighbor report. Specify a peer address to see a report for the specified address.

advertised-routes

Context	show network-instance *name* protocols bgp neighbor *IP address* advertised-routes
Tree	advertised-routes
Description	Show a bgp neighbor advertised routes report. The network-instance name and neighbor peer IP address must be specified.

evpn

Context	show network-instance *name* protocols bgp neighbor *IP address* advertised-routes evpn
Tree	evpn
Description	Show a bgp neighbor advertised routes report for EVPN. The network-instance name and neighbor peer IP address must be specified.

ipv4

Context	show network-instance *name* protocols bgp neighbor *IP address* advertised-routes ipv4
Tree	ipv4
Description	Show a bgp neighbor advertised routes report for IPv4. The network-instance name and neighbor peer IP address must be specified.

ipv6

Context	show network-instance *name* protocols bgp neighbor *IP address* advertised-routes ipv6
Tree	ipv6
Description	Show a bgp neighbor advertised routes report for IPv6. The network-instance name and neighbor peer IP address must be specified.

maintenance

Context	show network-instance *name* protocols bgp neighbor *IP address* maintenance
Tree	maintenance
Description	Show a bgp neighbor report that shows the maintenance mode and group.

received-routes

Context	show network-instance *name* protocols bgp neighbor *IP address* received-routes
Tree	received-routes
Description	Show a bgp neighbor received routes report. network-instance name and neighbor peer IP address must be specified.

evpn

Context	show network-instance *name* protocols bgp neighbor *IP address* received-routes evpn
Tree	evpn

Description	Show a bgp neighbor received routes report for EVPN. The network-instance name and neighbor peer IP address must be specified.

ipv4

Context	show network-instance *name* protocols bgp neighbor *IP address* received-routes ipv4
Tree	ipv4
Description	Show a bgp neighbor received routes report for IPv4. The network-instance name and neighbor peer IP address must be specified.

ipv6

Context	show network-instance *name* protocols bgp neighbor *IP address* received-routes ipv6
Tree	ipv6
Description	Show a bgp neighbor received routes report for IPv6. The network-instance name and neighbor peer IP address must be specified.

detail

Context	show network-instance *name* protocols bgp neighbor *IP address* detail
Tree	detail
Description	Show a bgp neighbor detailed report. network-instance name and neighbor peer IP address must be specified.

routes

Context	show network-instance *name* protocols bgp routes
Tree	routes
Description	Show bgp route report for an IP family (EVPN, IPv4 or IPv6).

evpn

Context	show network-instance *name* protocols bgp routes evpn
Tree	evpn
Description	Show a bgp EVPN route report. Additional parameters can define a report for a specific route type, and a detailed verses summary report.

route-type

Context	show network-instance *name* protocols bgp routes evpn route-type *route type*
Tree	route-type
Description	Show a bgp EVPN route report for a specific route type. Route type can be numeric or of type esi, mac-address, and originating-router.

detail

Context	show network-instance *name* protocols bgp routes evpn route-type *route type* detail
Tree	detail
Description	Show a detailed bgp EVPN route report for a specific route type.

summary

Context	show network-instance *name* protocols bgp routes evpn route-type *route type* summary
Tree	summary
Description	Show a summarized bgp EVPN route report for a specific route type.

ipv4

Context	show network-instance *name* protocols bgp routes ipv4
Tree	ipv4
Description	Show a bgp IPv4 route report. Specify an IP prefix or summary.

prefix

Context	show network-instance *name* protocols bgp routes ipv4 prefix I*P prefix*
Tree	prefix
Description	Show a bgp IPv4 route report for a specified IP prefix.

summary

Context	show network-instance *name* protocols bgp routes ipv4 summary
Tree	summary
Description	Show a bgp IPv4 route summary report.

ipv6

Context	show network-instance *name* protocols bgp routes ipv6
Tree	ipv6
Description	Show a bgp IPv6 route report. Specify an IP prefix or summary.

prefix

Context	show network-instance *name* protocols bgp routes ipv6 prefix *IP prefix*
Tree	prefix
Description	Show a bgp IPv6 route report for a specified IP prefix.

summary

Context	show network-instance *name* protocols bgp routes ipv6 summary

Tree	summary
Description	Show a bgp IPv6 route summary report.

summary

Context	show network-instance *name* protocols bgp summary
Tree	summary
Description	Show a bgp summary report.

bgp-evpn

Context	show network-instance *name* protocols bgp-evpn
Tree	bgp-evpn
Description	Show a bgp-evpn status report.

bgp-instance

Context	show network-instance *name* protocols bgp-evpn bgp-instance *instance*
Tree	bgp-instance
Description	Show a bgp-evpn status report for a specific bgp instance.

bgp-vpn

Context	show network-instance *name* protocols bgp-vpn
Tree	bgp-vpn
Description	Show a bgp-vpn status report.

bgp-instance

Context	show network-instance *name* protocols bgp-vpn bgp-instance *instance*
Tree	bgp-instance
Description	Show a bgp-vpn status report for a specific bgp instance.

isis

Context	show network-instance *name* protocols isis
Tree	isis
Description	Show ISIS status report. Specify adjacency, database, hostnames, interface, or summary.

adjacency

Context	show network-instance *name* protocols isis adjacency *interface_name*
Tree	adjacency
Description	Show a list of ISIS adjacencies formed through this interface.

neighbor-system-id

Context	show network-instance *name* protocols isis adjacency *interface_name* neighbor-system-id *ID*
Tree	neighbor-system-id
Description	Show a list of ISIS adjacencies for a specified system ID of a neighbor router.

adjacency-level

Context	show network-instance *name* protocols isis adjacency *interface_name* neighbor-system-id *ID* adjacency-level *value*
Tree	adjacency-level
Description	Show a list of ISIS adjacencies for a specified adjacency level.

detail

Context	show network-instance *name* protocols isis adjacency *interface_name* neighbor-system-id *ID* adjacency-level *value* detail
Tree	detail
Description	Show a detailed ISIS adjacency report.

database

Context	show network-instance *name* protocols isis database
Tree	database
Description	Show an ISIS database report.

lsp-id

Context	show network-instance *name* protocols isis database lsp-id *ID*
Tree	lsp-id
Description	Show an ISIS database report for a specified lsp-id (ID format: 6 octets of adjacency system-id followed by 1 octet Lan-ID and 1 octet LSP Number).

detail

Context	show network-instance *name* protocols isis database detail
Tree	detail
Description	Show a detailed ISIS database report.

interface

Context	show network-instance *name* protocols isis interface *name*
Tree	interface
Description	Show an ISIS interface report for all interfaces or optionally specify an interface name.

detail

Context	show network-instance *name* protocols isis interface *name* detail
Tree	detail
Description	Show a detailed ISIS interface report for all interfaces or optionally specify an interface name.

hostnames

Context	show network-instance *name* protocols isis hostnames
Tree	hostnames
Description	Show an ISIS hostname report.

detail

Context	show network-instance *name* protocols isis hostnames detail
Tree	detail
Description	Show a detailed ISIS hostname report.

summary

Context	show network-instance *name* protocols isis summary
Tree	summary
Description	Show a summarized ISIS report.

ospf

Context	show network-instance *name* protocols ospf
Tree	ospf
Description	Show OSPF status report. Specify area, interface, neighbor, or status.

area

Context	show network-instance *name* protocols ospf area
Tree	area
Description	Show OSPF report for areas where the local system exists.

detail

Context	show network-instance *name* protocols ospf area detail
Tree	detail
Description	Show detailed OSPF report for areas where the local system exists.

database

Context	show network-instance *name* protocols ospf database

Tree	database
Description	Show OSPF database report.

type

Context	show network-instance *name* protocols ospf database type
Tree	type
Description	Show OSPF database report for a specific database type. For example: router-lsa

instance

Context	show network-instance *name* protocols ospf instance *instance name*
Tree	instance
Description	Show OSPF report for a specific instance. After the instance name, a more detailed report can be generated by specifying area, database, interface, neighbor, statistics, or status after the instance name.

interface

Context	show network-instance *name* protocols ospf interface
Tree	interface
Description	Show OSPF report for OSPF interfaces that act as a connection between a router and one ofits attached networks.

detail

Context	show network-instance *name* protocols ospf interface detail
Tree	detail
Description	Show detailed OSPF report for OSPF interfaces that act as a connection between a router and one of its attached networks.

neighbor

Context	show network-instance *name* protocols ospf neighbor
Tree	neighbor
Description	Show OSPF report for OSPF neighbors.

detail

Context	show network-instance *name* protocols ospf neighbor detail
Tree	detail
Description	Show detailed OSPF report for OSPF neighbors.

statistics

Context	show network-instance *name* protocols ospf statistics

Tree	statistics
Description	Show OSPF Rx, Tx, and packet statistics report.

status

Context	show network-instance *name* protocols ospf status
Tree	status
Description	Show an overall status report for OSPF (router IDs, version, admin state, backbone and area border router settings, and so on.).

route-table

Context	show network-instance *name* route-table
Tree	route-table
Description	Show a network-instance route table report. Specify all, ipv4-unicast, ipv6-unicast, next-hop, or, summary.

all

Context	show network-instance *name* route-table all
Tree	all
Description	Show a report of all routes that includes prefix, ID, active status and next-hop details.

summary

Context	show network-instance *name* route-table summary
Tree	summary
Description	Show a summary report of active routes by name and protocol.

ipv4-unicast

Context	show network-instance *name* route-table ipv4-unicast
Tree	ipv4-unicast
Description	Show a IPv4 route table report. Specify a summary report or by prefix ID.

prefix <*ID*> detail

Context	show network-instance *name* route-table ipv4-unicast prefix <ID> detail
Tree	prefix ID detail
Description	Show a IPv4 route table report for a specified prefix. Report includes destination, ID, owner, active status and details when it was last changed.

summary

Context	show network-instance *name* route-table ipv4-unicast summary

Tree	summary
Description	Show a IPv4 route table summary report that includes prefix, IDs, active status, and next-hop details.

ipv6-unicast

Context	show network-instance *name* route-table ipv6-unicast
Tree	ipv6-unicast
Description	Show a IPv6 route table report. Specify a summary report or by prefix ID.

prefix <*ID*> detail

Context	show network-instance *name* route-table ipv6-unicast prefix <ID> detail
Tree	prefix ID detail
Description	Show a IPv6 route table report for a specified prefix. Report includes destination, ID, owner, active status and details when it was last changed.

summary

Context	show network-instance *name* route-table ipv6-unicast summary
Tree	summary
Description	Show a IPv6 route table summary report that includes prefix, IDs, active status, and next-hop details.

next-hop

Context	show network-instance *name* route-table next-hop *index*
Tree	next-hop
Description	Show a report of all next-hop route tables or optionally provide an index number to show a report for a specified route table.

summary

Context	show network-instance *name* summary
Tree	summary
Description	Show a summary report for all network-instances, or specify a network-name to refine the report.

tunnel-table

Context	show network-instance *name* tunnel-table
Tree	tunnel-table
Description	Context for all tunnels in a tunnel table. Use with options all, IPv4, or IPv6.

all

Context show network-instance *name* tunnel-table all

Tree all

Description Shows a network instance tunnel report that includes the prefix, type, ID, metric, preference, and last updated fields.

ipv4 *ip address* type

Context show network-instance *name* tunnel-table ipv4 ip address type

Tree ipv4 ip address type

Description Show a network instance IPv4 tunnel report that includes the prefix, type, ID, metric, preference, and last updated fields. Report can be defined further by specifying the type (such as vxlan, ldp, and * (all types).

ipv6 *ip address* type

Context show network-instance *name* tunnel-table ipv6 ip address type

Tree ipv6 ip address type

Description Show a network instance IPv6 tunnel report that includes the prefix, type, ID, metric, preference, and last updated fields. Report can be defined further by specifying the type (such as vxlan, ldp, and * (all types).

vxlan-interface

Context show network-instance *name* vxlan-interface *name*

Tree vxlan-interface

Description Show a network instance vxlan-interface report that defines the type, operation state and operation down reason for a specified interface or all interfaces (when the vxlan-interface name is “*”).

8.7 Platform show reports

```
show
    — platform
        — control <slot ID>
        — environment
        — fabric <slot number>
        — fan-tray <ID number>
        — linecard <slot number>
        — power-supply <ID number>
        — redundancy
        — resource-monitoring
```

8.7.1 Platform descriptions

show

Context	show
Tree	show
Description	Container for show reports.

platform

Context	show platform
Tree	platform
Description	Show a state report for all components.

control

Context	show platform control *slot ID*
Tree	control
Description	Show a report for the control module configuration and state. Shows all modules unless a specific slot is provided.

environment

Context	show platform environment
Tree	environment
Description	Show a report for the platform environment (state and temperature) for all components.

fabric

Context	show platform fabric *slot ID*
Tree	fabric
Description	Show a report for the fabric module configuration and state. Shows all modules unless a specific slot is provided.

fan-tray

Context	show platform fan-tray *slot ID*
Tree	fan-tray
Description	Show a report for the fan tray configuration and state. Shows all modules unless a specific slot is provided.

linecard

Context	show platform linecard *slot ID*
Tree	linecard

Description	Show a report for the linecard configuration and state. Shows all modules unless a specific slot is provided.

power-supply

Context	show platform power-supply *ID*
Tree	power-supply
Description	Show a report for the power-supply configuration and state. Shows all modules unless a specific ID is provided.

redundancy

Context	show platform redundancy
Tree	redundancy
Description	Show a platform redundancy report.

resource-monitoring

Context	show platform resource-monitoring
Tree	resource-monitoring
Description	Show a resource-monitoring report for areas such as ACL, TCAM, IP MPLS forwarding, etc.

8.8 System show reports

```
show
    — system
        — aaa authentication session <session ID number>
        — application <application name>
        — lldp neighbor
            — interface <interface name>
        — logging
            — buffer
                — messages
                — system
            — file
                — messages
        — network-instance ethernet-segments <name>
        — sflow status
```

8.8.1 System descriptions

show

Context	show
Tree	show
Description	Container for show reports.

system

Context	show system
Tree	system
Description	Container for system management show reports.

aaa authentication session

Context	show system aaa authentication session *session ID*
Tree	aaa authentication session
Description	Show a list of all active sessions in the system or for a specific session ID user if an ID is specified.

application

Context	show system application *application name*
Tree	application
Description	Show a list of all applications and their status in the system or for a specific application if a name is specified.

lldp neighbor

Context	show system lldp neighbor
Tree	lldp neighbor
Description	Show a report of all LLDP neighbors that includes system name, chassis ID, first message, last update, port and related BGP details.

interface

Context	show system lldp neighbor interface *interface name*
Tree	interface
Description	Show a report of a specific LLDP neighbors by specifying an interface name.

logging

Context	show system logging
Tree	logging
Description	Shows a list of system logs, type, and directory path.

buffer

Context	show system logging buffer
Tree	buffer
Description	Shows a report of type buffer log files maintained in memory (non-persistent across system reboots).

messages

Context	show system logging buffer messages
Tree	messages
Description	Show a report of the messages associated with the system logging buffer (/var/log/srlinux/buffer/messages)

system

Context	show system logging buffer system
Tree	system
Description	Show a report of the system related information in the system logging buffer (/var/log/srlinux/buffer/system)

file

Context	show system logging file
Tree	file
Description	Shows a report of log files that have been directed to a specific file (/var/log/srlinux/file).

messages

Context	show system logging file messages
Tree	messages
Description	Show a report of the messages associated with logs in a specific file (/var/log/srlinux/file/messages)

network-instance ethernet-segments

Context	show system network-instance ethernet-segments *name*
Tree	network-instance ethernet-segments
Description	Show a report that defines ethernet-segments details such as ESI, Alg, peers, and associated network instances.

sflow status

Context	show system sflow status
Tree	sflow status
Description	Show an sflow status report that includes the state, and sample size and rate.

8.9 Tunnel show reports

```
show
    — tunnel
        — vxlan-tunnel
```

```
              — all
              — vtep <ip address>
```

8.9.1 Tunnel descriptions

show

Context	show
Tree	show
Description	Container for show reports.

tunnel

Context	show tunnel
Tree	tunnel
Description	Enter the tunnel context.

vxlan-tunnel

Context	show tunnel vxlan-tunnel
Tree	vxlan-tunnel
Description	Show a tunnel report specific to VXLAN tunnels. Report includes the VXLAN tunnel endpoint address, index and when last updated. Specify a specific VTEP ip address or 'all' to display all vxlan tunnels.

all

Context	show tunnel vxlan-tunnel all
Tree	all
Description	Show a tunnel report specific to all VXLAN tunnels. Report includes the VXLAN tunnel endpoint address, index, and when last updated.

vtep

Context	show tunnel vxlan-tunnel vtep *ip address*
Tree	vtep
Description	Show a tunnel report for a specific VXLAN tunnel. Report includes the VXLAN tunnel endpoint address, index, and when last updated.

8.10 Tunnel-interface show reports

```
show
    — tunnel-interface <interface>
        — vxlan-interface <interface>
            — bridge-table
```

```
                – mac-table
                – multicast-destinations
                    – destination | VNI
                – unicast-destinations
                    – destination | VNI
            – brief
            – detail
```

8.10.1 Tunnel-interface descriptions

show

Context	show
Tree	show
Description	Container for show reports.

tunnel-interface

Context	show tunnel-interface *interface*
Tree	tunnel-interface
Description	Enter the context for tunnel interface (reports for EVPN Layer-2). Specify a tunnel-interface name or an asterisk (*) for all interfaces.

vxlan-interface

Context	show tunnel-interface *interface* vxlan-interface *interface*
Tree	vxlan-interface
Description	Show a vxlan interface report for EVPN-VXLAN for layer-2. Specify an vxlan-interface name or an asterisk (*) for all interfaces. Report options include brief, detail, or bridge-table.

bridge-table

Context	show tunnel-interface *interface* vxlan-interface *interface* bridge-table
Tree	bridge-table
Description	Show a bridge-table vxlan interface report for EVPN-VXLAN for layer-2. Options to use with bridge table include mac-table, multicast-destinations, and unicast-destination.

mac-table

Context	show tunnel-interface *interface* vxlan-interface *interface* bridge-table mac-table
Tree	mac-table
Description	Show a mac-table vxlan interface report for EVPN-VXLAN for layer-2. Report includes the address, desination, VNI, index, type, and last updated.

multicast-destinations

Context	show tunnel-interface *interface* vxlan-interface *interface* bridge-table multicast-destinations
Tree	multicast-destinations
Description	Show a multicast-destinations vxlan interface report for EVPN-VXLAN for layer-2. Use with the destination or VNI options.

destination \| VNI

Context	show tunnel-interface *interface* vxlan-interface *interface* bridge-table multicast-destinations destination | VNI
Tree	destination | VNI
Description	Show a unicast-destinations vxlan interface report for EVPN-VXLAN for layer-2 for a specific destination or VNI or use the asterisk after either option to see all destinations/ VNIs.

unicast-destinations

Context	show tunnel-interface *interface* vxlan-interface *interface* bridge-table unicast-destinations
Tree	unicast-destinations
Description	Show a unicast-destinations vxlan interface report for EVPN-VXLAN for layer-2. Use with the destination or VNI options.

destination \| VNI

Context	show tunnel-interface *interface* vxlan-interface *interface* bridge-table unicast-destinations destination | VNI
Tree	destination | VNI
Description	Show a unicast-destinations vxlan interface report for EVPN-VXLAN for layer-2 for a specific destination or VNI or use the asterisk after either option to see all destinations/ VNIs.

brief

Context	show tunnel-interface *interface* vxlan-interface *interface* brief
Tree	brief
Description	Show a brief vxlan interface report for EVPN-VXLAN for layer-2. The brief report includes the tunnel-interface, VXLAN-interface, type (bridged/routed), ingress VNI, and egress source-ip.

detail

Context	show tunnel-interface *interface* vxlan-interface *interface* detail
Tree	detail

Description	Show a detailed vxlan interface report for EVPN-VXLAN for layer-2. The detailed report includes the same details as the brief report (tunnel-interface, VXLAN-interface, type (bridged/routed), ingress VNI, and egress source-ip), plus bridge table and summary details.

8.11 Version show reports

```
show
    — version
```

8.11.1 Version descriptions

show

Context	show
Tree	show
Description	Container for show reports.

version

Context	show version
Tree	version
Description	Show a version report that contains the currently running software version, last booted, and memory details.

Customer document and product support

Customer documentation

Customer documentation welcome page

Technical support

Product support portal

Documentation feedback

Customer documentation feedback