Nokia Service Router Linux

Release 24.3

System Management Guide

3HE 20232 AAAA TQZZA
Edition: 01
March 2024

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.

Configuration and command outputs shown in this guide are examples only; actual displays may differ depending on supported functionality and user configuration.

1.1 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.

1.2 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**.
- 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 What's new

Topic	Location
Configurable SR Linux CLI completion type setting	Configuring the SR Linux CLI completion type
gNMI service configuration using a gRPC server	gNMI service configuration
gNOI Packet Link Qualification service	gNOI Packet Link Qualification service
gNOI configuration using a gRPC server	gNOI configuration
gNSI	gNSI
Compressed JSON responses using gzip	Compressed JSON responses using gzip
SNMPv3 support	SNMP versions and configuration
Interactive CLI commands on non-interactive interfaces (for example, JSON-RPC's CLI method).	Interactive CLI commands
Show reports added for **show network-instance protocols bgp routes [ipv4-labeled-unicast | ipv6-labeled-unicast]**	ipv4-labeled-unicast ipv6-labeled-unicast

3 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.

3.1 CLI access and use

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

3.1.1 Accessing the CLI

About this task

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.

Procedure

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 }--[ ]--
```

3.1.2 Using the CLI help functions

About this task

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

Procedure

Enter a question mark (**?**) after a command to display the command usage.

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

Example: ?

In this 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

The following example shows the system output from the **help** command

```
# 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

3.1.3 Using the CLI auto-complete function

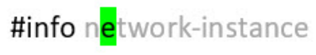
About this task

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

Procedure

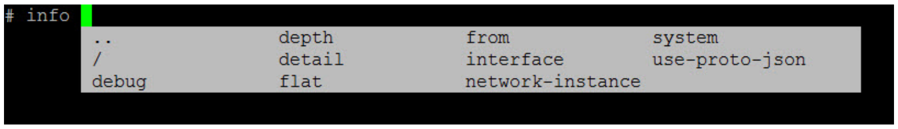
Enter a tab at any mode or level to auto-complete the next command level.

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 in a popup:



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

3.1.4 Wildcards and ranges

Rather than define a single value for a parameter, the SR Linux CLI allows you to enter a wildcard or a range to substitute for one or more values, such as a list of interfaces. The CLI engine automatically expands the specified wildcard or range into a list of individual values and executes the command for each value in that list.

3.1.4.1 Wildcards

In the SR Linux CLI, you can enter an asterisk (*****) to serve as a wildcard character in commands. You can use wildcards to represent any predefined values for a parameter. Note the following considerations when you use wildcards:

- Wildcards are valid in both **info** and configuration commands.
- Multiple wildcards are allowed in one command.
- Wildcards apply only to existing pre-configured objects (for non-configured objects, use ranges).
- Wildcards expand to YANG list keys, which must be valid in the context in which the wildcard is entered.
- Wildcards and ranges can be entered in the same command.

The following sections provide some examples of wildcards.

Wildcards in info commands

The following example uses a wildcard character as a substitute for the subinterface value in the **info interface** command to display all subinterfaces of ethernet-1/1:

Example: Display all subinterfaces using a wildcard

```
--{ * candidate shared default }--[  ]--
# info interface ethernet-1/1 subinterface *
    interface ethernet-1/1 {
        subinterface 0 {
            admin-state enable
        }
        subinterface 1 {
            admin-state enable
        }
        subinterface 2 {
            admin-state enable
        }
    }
```

You can also enter multiple wildcards to substitute for multiple parameters. The following example checks the active status for all IPv4 unicast BGP routes in the default network instance:

Example: Display state for all IPv4 unicast BGP routes using wildcards

```
--{ * candidate shared default }--[  ]--
```

```
# info from state network-instance default route-table ipv4-unicast route * id * route-type bgp route-
owner * origin-network-instance * active
    network-instance default {
        route-table {
            ipv4-unicast {
                route 10.0.0.2/32 id 0 route-type bgp route-owner bgp_mgr origin-network-instance
 default {
                    active true
                }
                route 10.0.0.3/32 id 0 route-type bgp route-owner bgp_mgr origin-network-instance
 default {
                    active true
                }
                route 10.0.0.4/32 id 0 route-type bgp route-owner bgp_mgr origin-network-instance
 default {
                    active true
                }
            }
        }
    }
```

Wildcards in configuration commands

You can also use wildcards in configuration commands. The following example adds **subinterface 0** to every configured interface in the system:

Example: Add subinterface 0 to all configured interfaces

```
--{ + candidate shared default }--[  ]--
# /interface * subinterface 0 description "created with a wildcard"
```

Wildcards in interface commands

Wildcards can expand YANG list keys, with the exception of interface names. However, within the interface name, you can use wildcards for either the linecard or port elements.

The following example uses a wildcard for the port value to display the admin-state for all interfaces on linecard 1:

Example: Display admin-state for all interfaces on linecard 1

```
--{ + candidate shared default }--[  ]--
# info interface ethernet-1/* admin-state
    interface ethernet-1/1 {
        admin-state enable
    }
    interface ethernet-1/3 {
        admin-state enable
    }
    interface ethernet-1/8 {
        admin-state enable
    }
```

Similarly, you can include a wildcard to specify one port on all linecards (for example: **interface ethernet-*/2**).

Wildcards in expanded contexts

Wildcards support the ability to enter an expanded context. Context-based configuration workflows with wildcards can eliminate copying and pasting by applying the configuration commands to all existing objects.

For example, to analyze the state of the configured interfaces, you can enter the context of all interfaces at once, as shown in the following example:

Example: Enter state mode for all interfaces

```
--{ + running }--[  ]--
# enter state
--{ + state }--[  ]--
# interface *
--{ + state }--[ interface * ]--
```

From the wildcarded context, you have access to a range of commands that are executed across all applicable interfaces. The following example lists the number of incoming unicast packets on all interfaces:

Example: List the number of incoming unicast packets on all interfaces

```
--{ + state }--[ interface * ]--
# info statistics in-unicast-packets
    interface ethernet-1/1 {
        statistics {
            in-unicast-packets 0
        }
    }
    interface ethernet-1/3 {
        statistics {
            in-unicast-packets 0
        }
    }
    ...
    interface mgmt0 {
        statistics {
            in-unicast-packets 919
        }
    }
```

You can also use wildcards in the context mode for configuration tasks. The following example adds VLAN tagging for **subinterface 0** on all applicable interfaces:

Example: Add VLAN tagging for subinterface 0 on all interfaces

```
# switching to candidate datastore
--{ + state }--[ interface * ]--
A:srl# enter candidate

# entering the wildcarded context
# of subinterface 0 of all interfaces
--{ + candidate shared default }--[  ]--
A:srl# interface * subinterface 0

# adding vlan tagging on all of them
--{ + candidate shared default }--[ interface * subinterface 0 ]--
A:srl# vlan encap single-tagged vlan-id any
```

As a result, the VLAN tagging configuration is applied to all subinterfaces:

Example: Confirm the VLAN tagging is applied (diff)

```
--{ +* candidate shared default }--[ interface * subinterface 0 ]--
# diff
      interface ethernet-1/1 {
          subinterface 0 {
              vlan {
                  encap {
+                     single-tagged {
+                         vlan-id any
+                     }
                  }
              }
          }
      }
      interface ethernet-1/3 {
          subinterface 0 {
              vlan {
                  encap {
+                     single-tagged {
+                         vlan-id any
+                     }
                  }
              }
          }
      }
...
```

Wildcards and strings

Wildcards can also be used with string-based keys. You can add a wildcard character anywhere in the string key to match any number of characters in that position. For example, on a system that has several VRFs with names beginning with `red`, you can match multiple VRFs as in the following example:

Example: Match multiple VRFs

```
--{ +* candidate shared default }--[  ]--
A:srl# info network-instance red*
    network-instance red {
        admin-state enable
    }
    network-instance red-a {
        admin-state enable
    }
    network-instance red-b {
        admin-state enable
    }
    network-instance red1 {
        admin-state enable
    }
```

In this case, **red*** expands to the **red**, **red-a**, **red-b**, and **red1** keys.

Wildcard expansion with string keys also provides a means to filter out objects that match a particular pattern, such as a customer name or location code.

Wildcard applicability and scope

It is important to note that wildcards expand to existing objects only. For example, if your candidate or running datastore has only two interfaces **ethernet-1/1** and **ethernet-1/5**, then the wildcard **ethernet-1/*** only matches these existing interfaces.

Also, to expand a list key, the list key must pertain to the context in which the wildcard is employed.

Consider the case where three interfaces are configured and you want to enable LLDP. First, you can ensure that the interfaces exist:

Example: List interface state

```
--{ + candidate shared default }--[  ]--
# info from running interface ethernet-1/* admin-state
    interface ethernet-1/1 {
        admin-state enable
    }
    interface ethernet-1/3 {
        admin-state enable
    }
    interface ethernet-1/8 {
        admin-state enable
    }
```

But if you try to enable LLDP on all interfaces using a wildcard, the operation fails:

Example: Attempt to enable LLDP using a wildcard

```
--{ + candidate shared default }--[  ]--
# system lldp interface ethernet-1/* admin-state enable
Error: Path '.system.lldp.interface{.name==ethernet-1/*}' does not specify any existing
 objects
```

This error occurs because the **/system/lldp/interface** list does not contain these interfaces within its own context. These interfaces are instead contained in the **/interface** list context, which the **/system/lldp/interface** list references. As a result of this referencing structure, these interfaces are not eligible for wildcard expansion in the **/system/lldp/interface** context.

In this case, ranges can be useful as they can create objects that do not exist yet.

3.1.4.2 Ranges

CLI ranges allow you to define a series of values for a particular list key. Unlike wildcards, ranges can generate new objects within the system, allowing for bulk object creation.

To express a range, enter a list of values separated by commas. Each value can be a single scalar value. You can also specify a continuous range of values using the double-dot (**..**) delimiter, or mix both formats within the same range specification.

Note the following considerations when you use ranges:

- Ranges are valid in both **info** and configuration commands.
- Multiple ranges are allowed in one command.
- Ranges apply to both existing and new objects.
- In a configuration command, if a range includes an object that already exists, the command overwrites the existing object.

- Wildcards and ranges can be entered in the same command.

The following table provides a few examples showing the syntax of different range patterns and how they translate to the list of elements:

Syntax	Result
{1,3}	1, 3
{2..5}	2, 3, 4, 5
{1,3..5,8}	1, 3, 4, 5,8

The following example shows how to display the administrative state of interfaces 1, 3, and 5 using a comma-separated list of elements in just one command:

Example: Display admin-state of interfaces 1,3, and 5

```
--{ + running }--[  ]--
# info interface ethernet-1/{1,3,5} admin-state
    interface ethernet-1/1 {
        admin-state enable
    }
    interface ethernet-1/3 {
        admin-state enable
    }
    interface ethernet-1/5 {
        admin-state enable
    }
```

To create a consecutive range of integer values, you can use double-dot (**..**) notation. The following example uses a range of **ethernet-1/{2..4}** to list all interfaces in the range between 2 and 4 (the range boundaries are included).

Example: Display consecutive range of interfaces

```
--{ + running }--[  ]--
# info interface ethernet-1/{2..4} admin-state
    interface ethernet-1/2 {
        admin-state enable
    }
    interface ethernet-1/3 {
        admin-state enable
    }
    interface ethernet-1/4 {
        admin-state enable
    }
```

The following example mixes the two range patterns, providing greater flexibility:

Example: Mix ranges and scalars

```
--{ + running }--[  ]--
# info interface ethernet-1/{1,3..5,8} admin-state
    interface ethernet-1/1 {
        admin-state enable
    }
    interface ethernet-1/2 {
        admin-state enable
    }
    interface ethernet-1/3 {
        admin-state enable
    }
    interface ethernet-1/4 {
        admin-state enable
    }
    interface ethernet-1/5 {
        admin-state enable
    }
    interface ethernet-1/8 {
        admin-state enable
    }
```

Objects and scoping

Ranges can generate new objects within the system, which is useful for bulk object creation. However, if the CLI range includes an object that already exists, the configuration command overwrites the existing object as well.

Conversely, in the case of an **info** command, if a range expands to a non-existing object, it is skipped.

To demonstrate this behavior, consider a freshly deployed system, where no **ethernet-1/*** interfaces are configured:

Example: Confirm no interfaces configured

```
--{ + running }--[  ]--
# info interface ethernet-1/*
--{ + running }--[  ]--
#
```

To create a set of interfaces, wildcards are not helpful because they cannot expand to undefined objects. Instead, you can define a range:

Example: Create new objects with ranges

```
--{ + running }--[  ]--
# enter candidate

--{ + candidate shared default }--[  ]--
# interface ethernet-1/{1..4} admin-state enable
```

By using ranges in candidate mode, four new interfaces are created on the system with a single command:

Example: Display configuration changes (diff)

```
--{ +* candidate shared default }--[  ]--
# diff
+     interface ethernet-1/1 {
+         admin-state enable
+     }
+     interface ethernet-1/2 {
+         admin-state enable
+     }
+     interface ethernet-1/3 {
+         admin-state enable
+     }
+     interface ethernet-1/4 {
+         admin-state enable
+     }
```

Because ranges can create new list elements, you can also successfully enable LLDP on multiple interfaces (unlike the LLDP example that failed with wildcards).

Example: Enable LLDP on multiple interfaces

```
--{ +* candidate shared default }--[  ]--
# system lldp interface ethernet-1/{1,2} admin-state enable
```

String key ranges

All the preceding range examples use integer keys, such as interface numbers or VLAN IDs. But ranges also support string-based keys, such as creating multiple named VRFs or ACLs.

Syntax	Result
{red,blue}	"red", "blue"
{red{1..2}}	"red1", "red2"
{red-{a..c}}	"red-a", "red-b", "red-c"

In its simplest form, string-based ranges operate on comma-separated list of strings. In the following example, two strings **red** and **blue** are included in the command to create two network instances.

Example: Create network instances red and blue

```
--{ +* candidate shared default }--[  ]--
A:srl# network-instance {red,blue} admin-state enable
```

The defined range expands to two elements and as a result two VRFs are created:

Example: Display configuration changes (diff)

```
--{ +* candidate shared default }--[  ]--
# diff
+     network-instance blue {
+         admin-state enable
+     }
+     network-instance red {
+         admin-state enable
+     }
```

Advanced templating syntax that involves nested ranges and a combination of both integer and string values is also supported:

Example: Create multiple red and blue VRFs

```
--{ +* candidate shared default }--[  ]--
# network-instance {red{1..2},blue{1..2}} admin-state enable
```

In this case, the defined range expands to two red and two blue VRFs:

Example: Display configuration changes (diff)

```
--{ +* candidate shared default }--[ network-instance {red{1..2},blue{1..2}} ]--
# diff
+     network-instance red1 {
+         admin-state enable
+     }
+     network-instance red2 {
+         admin-state enable
+     }
+     network-instance blue1 {
+         admin-state enable
+     }
+     network-instance blue2 {
+         admin-state enable
+     }
```

String-based ranges can create multiple named objects at once, and with nesting, you can construct even more intricate structures.

The **info** command can also take advantage of string-based ranges. The following example displays all ACLs that adhere to a specific naming pattern:

Example: Display ACL filters cust1-* and cust2-*

```
--{ * candidate shared default }--[  ]--
# info acl acl-filter {cust1-*,cust2-*} type ipv4 description
    acl {
        acl-filter cust1-filter1 type ipv4 {
            description somefilter
        }
        acl-filter cust1-filter2 type ipv4 {
            description somefilter
        }
        acl-filter cust2-filter1 type ipv4 {
            description somefilter
        }
        acl-filter cust2-filter2 type ipv4 {
            description somefilter
        }
    }
```

String-based ranges can also be useful in large scale deployments where for example named objects can encode customer or facility information.

3.1.4.3 Using wildcards and ranges in show route-table

Procedure

You can use ranges and wildcards in the **show network-instance route-table** command to display routes for a subset of prefixes or prefix lengths in one or more network instances.

For example, any of the following expressions can match route 192.168.0.1/32:

Table 1: Wildcard and range expressions to match 192.168.0.1/32

Expression	Description
*****	Full wildcard
***.168.0.1/32**	Wildcard for first octet
.168.0.1/	Wildcards for first octet and prefix length
1*.168.0.1/32	Partial wildcard in first octet

Expression	Description
{190..192}.*	Range for first octet and wildcard for final octets and prefix length

The following example displays routes for 172.18.0.x prefixes that have a prefix length of /24 (**172.18.0.*/24**) and for any prefixes that have a prefix length of /32 (***32**) in either the management (**m***) or default (**def***) network instances.

Example: Filtering route table output

```
--{ candidate shared default }--[  ]--
A:dut1# show network-instance {m*,def*} route-table ipv4-unicast prefix {172.18.0.*/24,*32}
-----------------------------------------------------------------------------------------------------
IPv4 unicast route table of network instance mgmt
-----------------------------------------------------------------------------------------------------
+-----------------+--+-----+--------+------+--------+------+----+-----------+----------+------+------+
| Prefix          |ID|Route| Route  |Active|Origin  |Metric|Pref|Next-hop   |Next-hop  |Backup|Backup|
|                 |  |Type | Owner  |      |Network |(Type)|    |           |Interface |Nexthp|Nexthp|
|                 |  |     |        |      |Instance|      |    |           |          |(Type)| If   |
+=================+==+=====+========+======+========+======+====+===========+==========+======+======+
| 172.18.0.9/32   |5 |host |net_inst| True | mgmt   | 0    | 0  |None       | None     |      |      |
|                 |  |     |_mgr    |      |        |      |    |(extract)  |          |      |      |
| 172.18.0.255/32 |5 |host |net_inst| True | mgmt   | 0    | 0  |None       |          |      |      |
|                 |  |     |_mgr    |      |        |      |    |(broadcast)|          |      |      |
| 172.18.0.0      |0 |linux|linux   | False| mgmt   | 0    | 5  |172.18.0.0 | mgmt0.0  |      |      |
|                 |  |     |_mgr    |      |        |      |    |(direct)   |          |      |      |
| 172.18.0.0      |5 |local|net_inst| True | mgmt   | 0    | 0  |172.18.0.9 | mgmt0.0  |      |      |
|                 |  |     |_mgr    |      |        |      |    |(direct)   |          |      |      |
+-----------------+--+-----+--------+------+--------+------+----+-----------+----------+------+------+
-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
IPv4 unicast route table of network instance default
-----------------------------------------------------------------------------------------------------

+-----------------+--+-----+--------+------+--------+------+----+-----------+----------+------+------+
| Prefix          |ID|Route| Route  |Active|Origin  |Metric|Pref|Next-hop   |Next-hop  |Backup|Backup|
|                 |  |Type | Owner  |      |Network |(Type)|    |           |Interface |Nexthp|Nexthp|
|                 |  |     |        |      |Instance|      |    |           |          |(Type)| If   |
+=================+==+=====+========+======+========+======+====+===========+==========+======+======+
| 10.1.1.1/32     |4 |host |net_inst| True |default | 0    | 0  |None       | None     |      |      |
|                 |  |     |_mgr    |      |        |      |    |(extract)  |          |      |      |
| 10.10.1.2/32    |2 |host |net_inst| True |default | 0    | 0  | None      | None     |      |      |
|                 |  |     |_mgr    |      |        |      |    |(extract)  |          |      |      |
| 10.1.1.255/32   |2 |host |net_inst| True |default | 0    | 0  | None      |          |      |      |
|                 |  |     |_mgr    |      |        |      |    |(broadcast)|          |      |      |
| 10.2.1.2/32     |3 |host |net_inst| True |default | 0    | 0  | None      | None     |      |      |
|                 |  |     |_mgr    |      |        |      |    |(extract)  |          |      |      |
| 10.2.1.255/32   |3 |host |net_inst| True |default | 0    | 0  |None       |          |      |      |
|                 |  |     |        |      |        |      |    |(broadcast)|          |      |      |
+-----------------+--+-----+--------+------+--------+------+----+-----------+----------+------+------+
-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
```

3.1.5 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 2: CLI keyboard shortcuts

Task	Keystroke
Move cursor to the beginning of the line	Ctrl-A
Move cursor to the end of the line	Ctrl-E
Move cursor one character to the right	Ctrl-F or Right arrow
Move cursor one character to the left	Ctrl-B or Left arrow
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

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**

3.1.6 Closing the CLI

Procedure

Close the CLI using one of the following methods:

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

3.2 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.

3.2.1 Configuration candidates

You can modify the candidate configuration in different modes:

- Exclusive
- Shared
- Private
- Name

3.2.1.1 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.

3.2.1.2 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.

3.2.1.3 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).

3.2.1.4 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.

3.2.2 Setting the configuration mode

Procedure

After logging in to the CLI, you are initially placed in running mode. To change between modes, use one of the commands in the following table.

Table 3: 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**

To enter this mode:	Type this command:
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}--[  ]--
```

3.2.3 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.

Procedure

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

- **baseline update** - 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.

3.2.4 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 4: 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.	**commit [checkpoint] [save] [stay] [comment] confirmed**

Option	Action	Permitted additional arguments
	Before the timer expires, the accept parameter explicitly confirms and applies the changes. With no timer running, the reject parameter explicitly rejects the changes.	

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}--[  ]--
#
```

3.2.5 Deleting configurations

Procedure

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 {
        }
    }
```

3.2.6 Annotating the configuration

Procedure

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 acl-filter ip_tcp type ipv4 !! "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 acl-filter ip-tcp type ipv4
--{ * candidate shared default}--[ acl acl-filter ip-tcp type ipv4  ]--
# 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 {
        acl-filter ip-tcp type ipv4 {
            !!! Filter TCP traffic
            entry 100 {
                action {
                    log true
                    drop {
                    }
                }
            }
            entry 110 {
                action {
                    log true
                    accept {
                    }
                }
            }
        }
    }
```

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

3.2.7 Discarding a configuration in candidate mode

Procedure

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 ]--
```

3.3 Administrative commands

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

3.3.1 Pinging a destination IP address

Procedure

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
```

3.3.2 Tracing the path to a destination

Procedure

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

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

3.3.3 Configuring the network-instance environment variable

About this task

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.

Procedure

To configure the network-instance environment variable, use the **environment network-instance** command

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

```
--{ 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
```

3.3.4 Using bash mode

Procedure

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

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

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

```
--{ 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 ###
```

3.3.5 Setting commands to execute periodically

Procedure

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)
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        |                      |                      |
```

3.3.6 Using the diff command

Procedure

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 {
+         acl-filter allow_sip_dip type ipv4 {
+             subinterface-specific output-only
+             entry 10 {
+                 match {
+                     ipv4 {
+                         destination-ip {
+                             prefix 100.1.2.2/32
+                         }
+                         source-ip {
+                             prefix 100.1.1.1/32
+                         }
+                     }
+                 }
+                 action {
+                     accept {
+                     }
+                 }
+             }
+             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
      }
```

3.3.7 Using the copy command

Procedure

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 {
                admin-state enable
                address 1.2.4.2/24 {
                }
            }
            ipv6 {
                admin-state enable
                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 acl-filter cpm type ipv4 entry 10
    acl {
        acl-filter cpm type ipv4 {
            entry 10 {
                description "cpm-filter description"
                match {
                    ipv4 {
                        protocol icmp
                        icmp {
                            type dest-unreachable
                            code [
                                0
                                1
                                2
                                3
                                4
                                13
                            ]
                        }
                    }
                }
                action {
                    accept {
                    }
                }
            }
        }
    }

--{ * candidate shared default }--[  ]--
# copy acl cpm-filter ipv4-filter entry 10 to interface ethernet-2/1
--{ +* candidate shared default }--[  ]--
# diff
+     interface ethernet-2/1 {
+         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
+     }
```

3.3.8 Using the replace command

Procedure

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 {
          ssh-server mgmt {
-             network-instance mgmt
+             network-instance test
          }
          grpc-server mgmt {
-             network-instance mgmt
+             network-instance test
          }
          json-rpc-server {
-             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 arguments 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 arguments is entered.

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

Related topics

Configuring CLI command aliases

3.3.9 Displaying configuration details

Procedure

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
    }
```

3.3.10 Displaying the command tree hierarchy

Procedure

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
+-- egress-mac-filtering
+-- acl-filter
|  +-- description
|  +-- subinterface-specific
|  +-- statistics-per-entry
|  +-- entry
|     +-- description
|     +-- match
|     |  +-- l2
|     |  |  +-- ethertype
|     |  |  +-- destination-mac
|     |  |  |  +-- address
|     |  |  |  +-- mask
|     |  |  +-- source-mac
|     |  |  |  +-- address
|     |  |  |  +-- mask
.
.
.
.
```

3.3.11 Displaying the state of the configuration

Procedure

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 }--[  ]--
```

3.3.12 Executing configuration statements from a file

Procedure

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). 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.

3.3.13 Collecting technical support data

Procedure

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 }--[  ]--
```

3.4 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

3.4.1 Specifying output format

Procedure

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 |
+----------+----------+-------+--------+--------+----------+----------+----------+
```

3.4.2 Using Linux output modifiers

Procedure

You can pipe output from SR Linux CLI commands to standard Linux tools using **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)
---------------------------------------------------------------------------------
```

3.4.3 Using output filters

Procedure

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"
        linecard 1
        forwarding-complex 0
        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 **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
        }
    }
```

3.4.4 Using the jq output modifier

About this task

The jq output modifier (see https://jqlang.github.io/jq/) is a JSON processor that can manipulate SR Linux **show**, **info**, and **info from state** output, provided the output is displayed in JSON format (using the **| as json** option). You can use jq to filter, extract, combine, and modify the JSON output. You can also use piping to combine jq arguments in various ways, such as feeding the results of one filter into another, or collecting the output into an array. The jq processor is supported as a CLI plug-in that is enabled by default.

Procedure

To modify the show or info output with the jq output modifier, include the **| as json** parameter in the command, followed by **| jq** and one or more jq arguments.

Note: SR Linux uses double quotes (**"**) rather than single quotes (**'**) to define jq filter expressions in the jq output modifier. Therefore, if a jq expression contains double quotes, they must be escaped using a backslash (**\"**).

Example: Modify info from state output

The following example modifies the output of the **info from state** command to list applications that have a status of **waiting-for-config**.

```
--{ running }--[  ]--
# info from state system app-management application * state | as json | jq ".system.\"app-
management\".application[] | select(.state == \"waiting-for-config\") | .name"
"dhcp_relay_mgr"
"dhcp_server_mgr"
"ethcfm_mgr"
"event_mgr"
"fhs_mgr"
"gribi_server"
"igmp_mgr"
"isis_mgr"
"l2_proxy_arp_nd_mgr"
"l2_static_mac_mgr"
"label_mgr"
"ldp_mgr"
"macsec_mgr"
"mirror_mgr"
"mpls_mgr"
"oam_mgr"
"oc_mgmt_server"
"ospf_mgr"
"p4rt_server"
"pim_mgr"
"segrt_mgr"
"static_route_mgr"
"te_mgr"
"tepolicy_mgr"
"twamp_mgr"
"vrrp_mgr"
```

3.4.5 Directing output to a file

Procedure

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%)
```

3.5 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.

3.5.1 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#
```

Procedure

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#
```

3.5.2 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
```

Procedure

You can configure the bottom toolbar using the **environment bottom-toolbar** command 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
```

3.5.3 Configuring the SR Linux CLI engine type

About this task

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.

Procedure

If necessary, you can use the **environment cli-engine type basic** command to configure 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

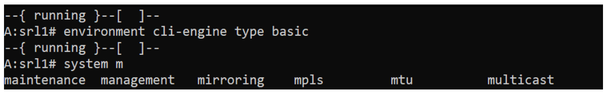
3.5.4 Configuring the SR Linux CLI completion type

About this task

When you enter a partial command at the CLI prompt and press the **Tab** key, SR Linux auto-completes the command if no other command at that level starts with those letters.

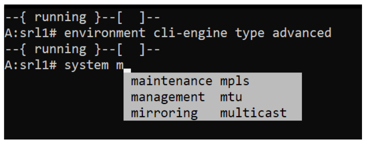
If multiple commands at that level start with the letters you typed, SR Linux displays a list of options that can complete the command.

- When the basic CLI engine type is configured, SR Linux displays the command options as a non-selectable list. For example:



```
--{ running }--[  ]--
A:srl1# environment cli-engine type basic
--{ running }--[  ]--
A:srl1# system m
maintenance  management   mirroring    mpls         mtu          multicast
```

- When the advanced CLI engine type is configured, SR Linux displays a popup with possible commands that are valid at that level. You can press the up or down arrows to select a command in the list. For example:



```
--{ running }--[  ]--
A:srl1# environment cli-engine type advanced
--{ running }--[  ]--
A:srl1# system m
                maintenance mpls
                management  mtu
                mirroring   multicast
```

The command options that appear when you press the **Tab** key depend on the SR Linux CLI completion type setting, which can be one of the following:

CLI completion type	Definition
prefix	The displayed options start with the letters you typed before pressing **Tab**. This is the only valid CLI completion type for the basic CLI engine.
smart	The displayed options contain the letters you typed before pressing **Tab**. Preference is given to the following: • Options that start with the letters you typed • Multi-word options where each word in the option starts with a letter you typed; for example, typing **ni** displays `network-instance` as an option. • Options that contain the letters you typed; for example, typing **inst** displays `network-instance` as an option. • Options that exist only at the current CLI level, rather than those that exist at every CLI level. This is the default CLI completion type for the advanced CLI engine.
substring	The displayed options contain the letters you typed as a string within the option name; for example, typing **stat** displays `modules-state`, `netconf-state`, and `statistics` as options.
fuzzy	SR Linux displays options that contain the letters you typed as a string, as well as options that are close; for example, typing **ntw** displays `network-instance` as an option.

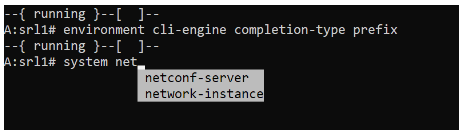
Procedure

To configure the SR Linux CLI completion type, use the **environment cli-engine completion-type** command.

Example: Set CLI completion type to prefix

The following example configures SR Linux to use **prefix** as the CLI completion type.

In this example, if you type **system net** and press **Tab**, SR Linux displays the command options at the system level that start with `net`.

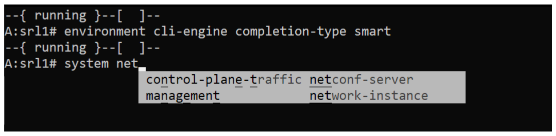


```
--{ running }--[  ]--
A:srl1# environment cli-engine completion-type prefix
--{ running }--[  ]--
A:srl1# system net
                  netconf-server
                  network-instance
```

Example: Set CLI completion type to smart

The following example configures SR Linux to use **smart** as the CLI completion type.

In this example, if you type **system net** and press **Tab**, SR Linux displays the command options at the system level that start with `net`, as well as those that include the letters n, e, and t.

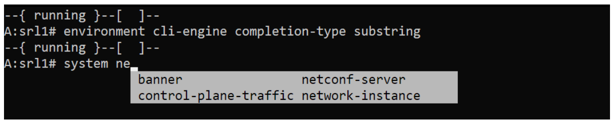


```
--{ running }--[  ]--
A:srl1# environment cli-engine completion-type smart
--{ running }--[  ]--
A:srl1# system net
                  control-plane-traffic netconf-server
                  management            network-instance
```

Example: Set CLI completion type to substring

The following example configures SR Linux to use **substring** as the CLI completion type.

In this example, if you type **system ne** and press **Tab**, SR Linux displays the command options at the system level that include the text string ne.

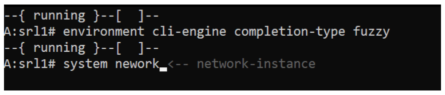


```
--{ running }--[  ]--
A:srl1# environment cli-engine completion-type substring
--{ running }--[  ]--
A:srl1# system ne
                 banner                netconf-server
                 control-plane-traffic network-instance
```

Example: Set CLI completion type to fuzzy

The following example configures SR Linux to use **fuzzy** as the CLI completion type.

In this example, if you type **system nework** and press **Tab**, SR Linux corrects the spelling of the command to **system network**, since **system network-instance** is a valid command at this CLI level.



```
--{ running }--[  ]--
A:srl1# environment cli-engine completion-type fuzzy
--{ running }--[  ]--
A:srl1# system nework <-- network-instance
```

3.5.5 Configuring the CLI output format

Procedure

You can configure the output of CLI commands using the **output-format** command 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"
  }
}
```

3.5.6 Configuring CLI command aliases

Procedure

As a shortcut for entering commands in the CLI, you can configure CLI command aliases using the **environment alias** command. 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      |
+---------------------+-------+-------------+
```

3.5.7 Configuring command auto-completion

About this task

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.

Procedure

To configure command auto-completion using the space bar, set the **environment complete-on-space** command to **true**.

Example: Set space bar to auto-complete commands

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

Related topics

Using the CLI auto-complete function

3.5.8 Displaying the CLI environment configuration

Procedure

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
```

3.5.9 Managing CLI environment settings

Procedure

You can save the current CLI environment settings to the default SR Linux configuration using the **environment save home** command, and you can load those settings using the **environment load home** command.

Alternatively, you can save the CLI environment settings to a file using the **environment save file** command and you can load those settings from the file using the **environment load file** command.

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: 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: 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 file

The following example loads CLI environment settings from a file:

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

4 YANG data models

SR Linux supports YANG data models for configuring the network element. YANG is a standards-based, data-modelling language that supports Remote Procedure Calls (RPCs). The SR Linux model-driven management interfaces are based on a common infrastructure that uses YANG models as the core definition for network element configuration, state, and operational actions. The model-driven interfaces (SR Linux CLI, gRPC server) take the underlying YANG modules and render them for the particular management interface.

SR Linux supports the following YANG data models:

- Nokia vendor-specific data models
- OpenConfig vendor-neutral data models

This chapter describes how SR Linux implements OpenConfig YANG data models and manages interactions with the Nokia YANG data models.

4.1 SR Linux data models

SR Linux makes extensive use of structured YANG data models to provide network operators with a single set of data models to configure and manage commonly-used network protocols, services, and devices. Each application that SR Linux supports has a Nokia vendor-specific YANG model that defines the application's configuration and state.

SR Linux exposes the YANG models to supported management APIs; for example, the CLI command tree is derived from the SR Linux YANG models loaded into the system. A gNMI client can use `Set` RPCs to configure an application based on the YANG model.

When you commit a configuration, the SR Linux management server validates the YANG models and translates them into protocol buffers for the impart database (IDB).

4.2 OpenConfig data models

OpenConfig is an informal working group that provides structured, vendor-neutral YANG data models to address the use requirements of networking applications and technologies by the community. OpenConfig data models support configuration and management of multivendor networks. They use standards-based, YANG data-modeling language, support Remote Procedure Calls (RPCs), and allow network operators to use a single set of data models to configure and manage commonly-used network protocols, services, and devices that support the OpenConfig initiative.

4.2.1 Implementation in SR Linux

SR Linux supports OpenConfig YANG data models for configuring and managing network elements. You can use the OpenConfig data models together with the SR Linux data models to configure network elements, using a CLI console or SSH connection or management-interface RPCs (gNMI) for communications between the clients and routers.

The SR Linux and OpenConfig data models are implemented through the management server binary. The SR Linux installation process installs the data models by default during the management server installation. The OpenConfig instance of the management server passes through authorization toward the native management server.

Note:
The system does not support multiple candidates within the OpenConfig instance of the management server.

See the *SR Linux Software Release Notes* for the supported OpenConfig modules in SR Linux releases.

4.2.2 Interaction with SR Linux data models

You can use the SR Linux vendor-specific and OpenConfig vendor-neutral YANG data models together to configure and manage network elements. The SR Linux data models offer a more complete representation of the capabilities of the SR Linux network elements, because they include vendor-specific features and functions that the OpenConfig data models do not describe.

The SR Linux configuration and operational statements map to path statements in the supported OpenConfig modules. The mappings are exposed in JSON files delivered with the software package. You can view the mapping files after installation, for vendor-specific deviations and augmentations.

Note: See the *SR Linux Software Release Notes* for information about the supported OpenConfig modules in SR Linux releases.

When using the gNMI service, the capabilities RPC can discover the capabilities of a specific gNMI server. The client sends a CapabilityRequest message to request capability information from the target. The target replies with a CapabilityResponse message that includes the gNMI service version, the supported data model versions, and the supported data encodings. Subsequent RPC messages from the client use this information to indicate the set of models used by the client and the encoding used for data.

The CapabilityResponse messages indicates each data model the target supports, based on the configuration in CLI (**system management openconfig admin-state**) . The advertised names include information about the model, organization, and version, for the respective YANG models, and Nokia deviations for the OpenConfig models. See gNMI Capabilities RPC for more information.

4.2.2.1 Viewing OpenConfig to SR Linux mapping files

Procedure

SR Linux exposes the OpenConfig to SR Linux data model module mappings in `oc-srl.json` files provided with the installation. You can locate and view the module mapping files after SR Linux installation in the `/opt/srlinux/mappings/openconfig` directory.

See the *SR Linux Software Release Notes* for information about the supported OpenConfig modules in SR Linux releases.

Example:

The following example shows partial content from the `oc-srl-lldp.json` mapping file.

```
"mapping": [
  {
    "oc": "/lldp",
    "srlinux": "/system/srl_nokia-lldp:lldp"
  },
  {
    "oc": "/lldp/{config,state}",
    "srlinux": ""
  },
  {
    "oc": "/lldp/{config,state}/enabled",
    "srlinux": "/system/srl_nokia-lldp:lldp/srl_nokia-lldp:admin-state",
    "transform":"bool-to-admin-state"
  },
  {
    "oc": "/lldp/{config,state}/hello-timer",
    "srlinux": "/system/srl_nokia-lldp:lldp/srl_nokia-lldp:hello-timer"
  },
  {
    "oc": "/lldp/{config,state}/suppress-tlv-advertisement",
    "supported": false
  },
  {
    "oc": "/lldp/config/system-name",
    "supported": false
  },
  {
    "oc": "/lldp/state/system-name",
    "srlinux": "/system/srl_nokia-lldp:lldp/srl_nokia-lldp:system-name"
  },
  {
    "oc": "/lldp/config/system-description",
    "supported": false
  },
  {
    "oc": "/lldp/state/system-description",
    "srlinux": "/system/srl_nokia-lldp:lldp/srl_nokia-lldp:system-description"
  },
  {
    "oc": "/lldp/config/chassis-id",
    "supported": false
  },
  {
    "oc": "/lldp/state/chassis-id",
    "srlinux": "/system/srl_nokia-lldp:lldp/srl_nokia-lldp:chassis-id"
  },
  {
    "oc": "/lldp/config/chassis-id-type",
    "supported": false
  },
  {
    "oc": "/lldp/state/chassis-id-type",
    "srlinux": "/system/srl_nokia-lldp:lldp/srl_nokia-lldp:chassis-id-type"
  },
  {
    "oc": "/lldp/state/counters",
    "srlinux": "/system/srl_nokia-lldp:lldp/srl_nokia-lldp:statistics"
  },
  {
    "oc": "/lldp/state/counters/frame-in",
       "srlinux": "/system/srl_nokia-lldp:lldp/srl_nokia-lldp:statistics/srl_nokia-
lldp:frame-in"
    },
    {
#
```

4.2.3 Enabling access to OpenConfig data models

Procedure

You can enable access to the OpenConfig data model using the **admin-state** in the **system management openconfig admin-state** context. When you validate and commit a configuration, the system verifies if **openconfig** is set to **true** and determines if there is access to OpenConfig data models. If the candidate contains OpenConfig configuration statements and **openconfig** is not enabled (**false**), the action fails with an error.

Note: The system does not support multiple candidates within the OpenConfig instance of the management server.

The following example shows the basic configuration required to enable the OpenConfig configuration modules and obtain state information.

Example:

```
A:dut2# system management openconfig admin-state enable
--{ [FACTORY] +* candidate shared default }--[ ]--
A:dut2# commit stay
All changes have been committed. Starting new transaction.
--{ [FACTORY] + candidate shared default }--[ ]--
A:dut2#
```

4.2.4 Specifying the data model mode

The SR Linux YANG data model is the default mode. The system supports different methods for switching between the default SR Linux data model and the OpenConfig data model.

4.2.4.1 Specifying the data model in SSH mode

Procedure

To specify the data model when accessing the system using SSH, use the **enter oc** or **enter srl** command. A special OpenConfig banner appears in OpenConfig mode.

Example: Specifying the data model in SSH mode

```
A:dut2# enter oc
--{ oc candidate shared default }--[ ]--
A:dut2# enter srl
--{ [FACTORY] + candidate shared default }--[ ]--
A:dut2#
```

4.2.4.2 Specifying the data model in Linux bash mode

Procedure

To specify the data model when logging in to the Linux bash, start the CLI using **sr cli --oc** or **sr cli --srl**. This enters directly into OpenConfig or SRL mode respectively.

Example: Specifying the data model in Linux bash mode

```
[linuxadmin@dut2 srl]$ sr_cli --oc
Using configuration file(s): ['/etc/opt/srlinux/srlinux.rc']
Welcome to the srlinux CLI.
Type 'help' (and press <ENTER>) if you need any help using this.
--{ oc running }--[ ]--
A:dut2# quit
[linuxadmin@dut2 srl]$ sr_cli --srl
Using configuration file(s): ['/etc/opt/srlinux/srlinux.rc']
Welcome to the srlinux CLI.
Type 'help' (and press <ENTER>) if you need any help using this.
--{ [FACTORY] + running }--[ ]--
A:dut2# quit
[linuxadmin@dut2 srl]$
```

4.2.4.3 Specifying the data model in CLI mode

Procedure

To specify the data model while working in the CLI environment, use the **environment yang-models** command.

Example: Specifying the data model in the CLI environment

```
A:dut2# environment yang-models oc
--{ [FACTORY] + running }--[  ]--
A:dut2# environment show
...
...
[yang-models]
value = "oc"

--{ [FACTORY] + running }--[  ]--
A:dut2#
```

4.2.5 Verifying the OpenConfig operational state

Procedure

You can verify the OpenConfig operational state using the command **system management openconfig oper-state**. The following example shows the operational state of the OpenConfig data model.

Example:

```
A:dut2# info from state system management openconfig
system {
    management {
        openconfig {
            admin-state enable
            oper-state up
        }
    }
}
--{ [FACTORY] + running }--[ ]--
A:dut2#
```

4.2.6 Specifying the data model mode for the gRPC server

Procedure

When using the gRPC server to send and accept edits, requests, and responses for the SR Linux or OpenConfig data models, you can specify the corresponding data model schema (**openconfig** or **native**) for the target in the **system grpc-server** *name* **yang-models** context. The following example shows this configuration.

Example: Yang data model selection for gRPC server

```
A:dut2# enter candidate
--{ [FACTORY] + candidate shared default }--[ ]--
A:dut2# system grpc-server mgmt yang-models openconfig
--{ [FACTORY] +* candidate shared default }--[ ]--
A:dut2# commit stay
All changes have been committed. Starting new transaction.
--{ [FACTORY] + candidate shared default }--[ ]--
A:dut2#
```

Alternatively, you can send a gNMI SetRequest with the origin prefix **openconfig** embedded in the request. This informs the system to use the OpenConfig data model, regardless of the value specified for **yang-models** under the gRPC server configuration.

Example: Prefix origin openconfig in gNMI SetRequest

```
SetRequest:
prefix: <
  origin: "openconfig"
>
```

5 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.

5.1 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.

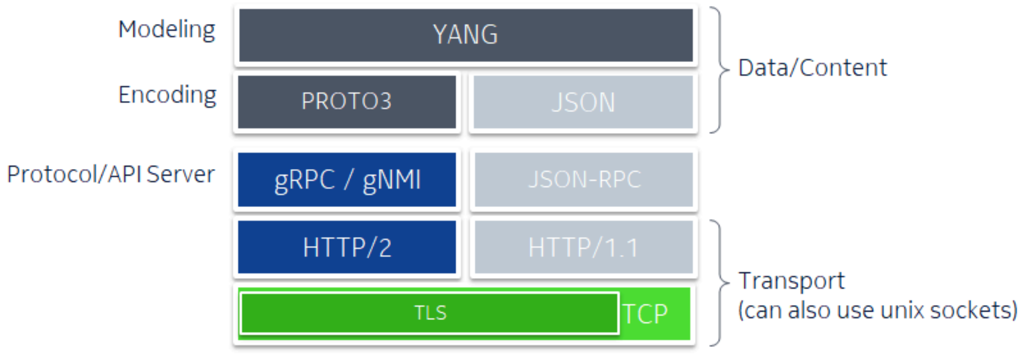
When RPCs operate in a server/client model, the client executes an application and requests some method be executed on a server. The server receives the RPC, executes the method, and sends the outcome back to the server. For this to occur, the following are needed:

- A server application that contains a set of methods (or service) which a client can call. For example, a gRPC 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



5.1.1 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]
```

5.2 Configuring a gRPC or JSON server

SR Linux can enable a gRPC server that allows external gRPC 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.

6 gNMI

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.

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

- Get
- Set
- Subscribe
- Capabilities

6.1 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 5: 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

6.1.1 Timestamps

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

6.1.2 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
  >
 >
>
```

6.1.3 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
```

6.1.4 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

6.1.4.1 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 6: Encoding for structured data

Data type	Description	Field
ASCII	An ASCII encoded string	ascii_val
PROTO	A Protobuf encoded message using protobuf.any	any_val
JSON_IETF	A JSON encoded string using JSON encoding compatible with RFC 7951	json_ietf_val

6.2 Data model selection using gNMI origin

Within a gNMI request, the origin extension allows clients to specify which data model to interact with: OpenConfig (**openconfig**), native (**native** or **srlinux_native**), or CLI (**cli** or **srlinux_cli**).

Clients can specify a value for the origin in the path prefix or in one or more paths, but not both at once. When a request specifies the origin in the path prefix, SR Linux uses the specified data model to process all operations within the transaction.

Note: If a request includes a path prefix, the prefix must specify any required origin.

If no origin is specified in a request, SR Linux handles the request based on the data model specified under **system grpc-server yang-models** (set to **native** by default).

SR Linux deviates from the gNMI specification in that it intrinsically links all origins such that a change in one data model produces the matching change in the other.

The following table describes the usage of the origin field in the message types where it is typically used.

Table 7: Purpose of origin field by message type

Message type	Purpose of `origin` field
SetRequest	Specifies the schema to use to modify the target configuration
GetRequest	Retrieves the contents of the specified schema
GetResponse	Indicates that the payload contains data from the specified <origin, path> schema
SubscribeRequest	Subscribes to paths within the specified schema
SubscribeResponse	Indicates the update corresponds to the specified <origin, path> tuple

Multiple path origins

If a SetRequest specifies more than one origin (for example, it contains two operations each with different path origins), the updates are processed as a single transaction. In this case, all operations must succeed for the SetResponse to return a success message. If any of the operations fail, the contents of all origins roll back, and the SetResponse returns an error.

You can specify mixed origins combining OpenConfig with native YANG or with CLI. In either case, SR Linux applies the OpenConfig origin first, followed by the native or CLI origin.

The following example shows the OpenConfig origin defined in the path prefix:

Example: Origin defined in path prefix

```
prefix {
  origin: "openconfig"
}
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "ethernet-1/1"
    }
  }
  elem {
    name: "state"
  }
  elem {
    name: "description"
  }
}
type: ALL
encoding: JSON_IETF
```

6.2.1 CLI configuration with gNMI origin

SR Linux supports setting the gNMI origin to CLI (**cli** or **srlinux_cli**), which allows gNMI clients to apply CLI configurations. In this case, SR Linux ignores any defined path and enters and commits the configuration included in the update or replace operation as CLI input.

The value defined in the CLI origin is encoded as ASCII and supports both the tree-based CLI commands as displayed using the SR Linux **info** command or the full-context **set /** commands as shown using the **info flat** command.

Delete operations are not supported with origin CLI.

If a SetRequest replace operation contains only the CLI origin, SR Linux treats the operation as a full device configuration replacement. The whole running configuration is replaced with the CLI commands applied over a blank configuration.

SR Linux supports only one CLI replace operation per SetRequest. However, one or more update operations can follow the replace operation, in which case the updates are appended to the contents of the replace operation.

The following example shows a SetRequest that uses CLI origin within the update operation:

Example: SetRequest with CLI origin

```
update: <
  path: <
    origin: "cli"
  >
  val: <
    ascii_val: "/interface ethernet-1/1 admin-state disable"
  >
>
```

6.3 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

6.3.1 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 8: 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

6.3.2 GetResponse message

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

Table 9: 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

6.4 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.

6.4.1 SetRequest message

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

Table 10: 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
union_replace	A set of update messages of mixed origin 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 considered 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.

6.4.2 SetResponse message

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

Table 11: 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

6.5 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.

6.5.1 SubscribeRequest message

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

Table 12: 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.

6.5.1.1 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 13: 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.

6.5.2 SubscribeResponse message

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

Table 14: 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

6.6 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.

6.6.1 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;
}
```

6.6.2 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;
}
```

6.7 Candidate mode

gNMI uses its own private exclusive candidate that restricts other users or services from making simultaneous changes to a configuration. If another exclusive session is already active, any attempted gNMI updates fail with an error.

The gNMI server uses the private exclusive candidate name **gnmirpc-**<*n*>, where <*n*> is a 32-bit number starting at 1 that increments for every request received, but resets on a gNMI server restart.

6.8 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

6.8.1 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"
     >
   >
>
```

6.8.2 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
```

6.8.3 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": {
    "acl-filter": {
      "allow_sip_dip": {
        "type": {
          "ipv4": {
            "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"
      }
    }
  }
}
.
.
.
```

6.8.4 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: PROTO
supported_encodings: 45
supported_encodings: 44
supported_encodings: 46
supported_encodings: 47
gNMI_version: "0.7.0"
```

6.9 gNMI service configuration

SR Linux supports a gRPC server that allows external gRPC clients, including gNMI clients, to connect to the device and modify the configuration and collect state information.

See the "Management servers" chapter in the *SR Linux Configuration Basics Guide* for information about how to configure the gRPC server for gNMI support.

7 gNOI

gRPC Network Operations Interface (gNOI) defines a set of gRPC-based services for executing operational commands on network devices. The individual RPCs and messages that perform the gNOI operations required on the node are defined at the following location: https://github.com/openconfig/gnoi. This repository also stores the various per-service protos in subdirectories.

SR Linux supports the following gNOI services:

- gNOI OS service
- gNOI FactoryReset service
- gNOI File service
- gNOI System service
- gNOI Healthz service
- gNOI Packet Link Qualification service

7.1 gNOI OS service

The gNOI OS service provides an interface to install an OS package on a target node. SR Linux supports the gNOI OS service on both the active and standby CPMs (referred to as supervisors in gNOI).

To perform the OS installation, the client progresses through the following three gNOI OS RPCs:

- Install RPC
- Activate RPC
- Verify RPC

The protos used to define the OS service were pulled from the following hash: https://github.com/openconfig/gnoi/commit/93cdd9ae9f35d8b4bc1599d0a727b294faeca352.

7.1.1 Install RPC

The Install RPC transfers the OS package to the target node. The target node first attempts to copy the specified OS package between the CPMs before it accepts the transfer from the client.

To refer to the OS version, SR Linux uses the same string in gNOI as the one used with ZTP (*version string-build number*) , for example: `v22.11.1-010`. The download folder for the OS is located at: `/var/run/srlinux/gnoi`. To validate that the transferred OS package is valid and bootable before installation, the platform performs a hash check against the md5sum that is embedded in the .bin file.

On a dual CPM node, only the active CPM runs the gNOI service. The Install RPC transfers the OS to the active CPM.

Note: SR Linux does not support the **standby_supervisor** option. On a dual CPM node, the transferred image is synced automatically to the standby CPM using ZTP.

One Install RPC is required for each CPM. Concurrent Install RPCs are not allowed on the same target node.

Install RPC structure

```
rpc Install(stream InstallRequest) returns (stream InstallResponse);
```

7.1.2 Activate RPC

The Activate RPC sets the requested OS version for the target node to use at the next reboot. It also reboots the target node if the `no_reboot` flag is not set.

Note: If the requested image fails to boot, SR Linux cannot attempt to boot a secondary image. In this case, the system can revert to the rescue image.

On a dual CPM node, if you perform this RPC on the active CPM, it triggers a switchover to the standby CPM before rebooting the previously active CPM.

Activate RPC structure

```
rpc Activate(ActivateRequest) returns (ActivateResponse);
```

7.1.3 Verify RPC

The Verify RPC checks the OS version running on the target node. The client can call this RPC multiple times while the target node boots until the activation is successful.

Note: The activation_fail_message is not supported because if the target node does not boot, it remains in a failure state and does not revert to a previous version of OS.

Verify RPC structure

```
rpc Verify(VerifyRequest) returns (VerifyResponse);
```

7.2 gNOI FactoryReset service

The FactoryReset service enables gNOI clients to reset a target node to boot using a golden image and to optionally format persistent storage.

One of the practical applications of this service is the ability to factory reset a device before performing Zero Touch Provisioning (ZTP).

SR Linux supports the following gNOI FactoryReset RPC:

- Start RPC

The protos used to define the FactoryReset service were pulled from the following hash: v0.1.0.

7.2.1 Start RPC

The Start RPC allows the client to instruct the target node to immediately clean all existing state data (including storage, configuration, logs, certificates, and licenses) and boot using the OS image configured as the golden image. The golden image is the image that the device resets to when a factory reset is performed. You can set the golden image using the **tools system boot golden-image** command. To view the available images to select from, use the **tools system boot available-images** command. If the golden image is not set, the system boots using the current OS image.

The Start RPC supports optional flags to:

- roll back to the OS configured as the golden image
- zero-fill any state data saved in persistent storage

If the golden image is configured and the **factory_reset** flag is set to **true**, SR Linux resets to the golden image. If the **factory_reset** flag is omitted or set to **false**, SR Linux boots using the current running image, but all existing state data is cleaned.

If any optional flags are set but not supported, the target node returns a gRPC Status message with code INVALID_ARGUMENT with the details value set to the appropriate ResetError message.

Start RPC structure

```
rpc Start(StartRequest) returns (StartResponse);
```

7.3 gNOI File service

The gNOI File service allows the client to transfer files to and from the target node. The main use for this service is extracting debugging information through the transfer of system logs and core files.

SR Linux supports the following gNOI File RPCs:

- Get RPC
- Put RPC
- Stat RPC
- Remove RPC

Note: The TransferToRemote RPC is not supported.

The protos used to define the gNOI File service were pulled from the following hash: https://github.com/openconfig/gnoi/commit/93cdd9ae9f35d8b4bc1599d0a727b294faeca352.

7.3.1 Get RPC

The Get RPC reads and streams the contents of a file from a target node to the client using sequential messages, and sends a final message containing the hash of the streamed data before closing the stream.

The target node returns an error if:

- An error occurs while reading the file.
- The file does not exist.

Get RPC structure

```
rpc Get(GetRequest) returns (stream GetResponse) {}
```

7.3.2 Put RPC

The Put RPC streams data to the target node and writes the data to a file. The client streams the file using sequential messages. The initial message contains information about the filename and permissions. The final message includes the hash of the streamed data.

The target node returns an error if:

- An error occurs while writing the data.
- The location does not exist.

Put RPC structure

```
rpc Put(stream PutRequest) returns (PutResponse) {}
```

7.3.3 Stat RPC

The Stat RPC returns metadata about files on the target node.

If the path specified in the StatRequest references a directory, the StatResponse returns the metadata for all files and folders, including the parent directory. If the path references a direct path to a file, the StatResponse returns metadata for the specified file only.

The target node returns an error if:

- The file does not exist.
- An error occurs while accessing the metadata.

Stat RPC structure

```
rpc Stat(StatRequest) returns (StatResponse) {}
```

7.3.4 Remove RPC

The Remove RPC removes the specified file from the target node.

The target node returns an error if:

- An error occurs during the remove operation (for example, permission denied).
- The file does not exist.
- The path references a directory instead of a file.

Remove RPC structure

```
rpc Remove(RemoveRequest) returns (RemoveResponse) {}
```

7.4 gNOI System service

The gNOI System service defines an interface that allows a client to perform operational tasks on target network nodes. SR Linux supports the following gNOI System RPCs:

- Ping RPC
- Traceroute RPC
- Time RPC
- SwitchControlProcessor RPC
- Reboot RPC
- CancelReboot RPC
- RebootStatus RPC
- KillProcess RPC

The protos used to define the gNOI System service were pulled from the following hash: https://github.com/openconfig/gnoi/commit/93cdd9ae9f35d8b4bc1599d0a727b294faeca352.

7.4.1 Ping RPC

The Ping RPC allows the client to execute the ping command on the target node. The target node streams the results back to the client. Some targets do not stream any results until they receive all results. If the RPC does not specify a packet count, the ping operation uses a default of five packets.

Note:

- The Ping RPC does not currently support specification of a network-instance. The ping is executed in the network-instance where the gNMI server is running.
- SR Linux does not support setting the interval field in the PingRequest to -1 (flood ping).

Ping RPC structure

```
rpc Ping(PingRequest) returns (stream PingResponse) {}
```

7.4.2 Traceroute RPC

The Traceroute RPC allows the client to execute the traceroute command on the target node. The target node streams the results back to the client. Some targets do not stream any results until they receive all results. If the RPC does not specify a hop count, the traceroute operation uses a default of 30.

Note:

- The Traceroute RPC does not currently support specification of a network-instance. The traceroute is executed in the network-instance where the gRPC server is running.
- In the TracerouteRequest, SR Linux does not support the TCP and UDP enum values for the l4protocol field. Only ICMP is supported.
- In the TracerouteResponse, SR Linux does not support the mpls and as_path fields.

Traceroute RPC structure

```
rpc Traceroute(TracerouteRequest) returns (stream TracerouteResponse) {}
```

7.4.3 Time RPC

The Time RPC returns the current time on the target node. It is typically used to test whether the target is currently responding.

Time RPC structure

```
rpc Time(TimeRequest) returns (TimeResponse) {}
```

7.4.4 SwitchControlProcessor RPC

The SwitchControlProcessor RPC switches the active control processing module (CPM) on the target node to the control slot (A or B) that is specified in the request message.

SwitchControlProcessor RPC structure

```
rpc SwitchControlProcessor(SwitchControlProcessorRequest)
    returns (SwitchControlProcessorResponse) {}
```

7.4.5 Reboot RPC

The Reboot RPC allows the client to reboot a target node, either immediately or at some time in the future. It triggers the reboot of the entire chassis. It also supports specification of a reboot method (for example, cold or warm reboot), however, if the target node does not support the specified reboot method, the Reboot RPC fails.

Note: SR Linux supports only the cold reboot method, and does not support rebooting of subcomponents.

If a reboot is pending on the active control processor, the service rejects all other reboot requests.

Reboot RPC structure

```
rpc Reboot(RebootRequest) returns (RebootResponse) {}
```

7.4.6 CancelReboot RPC

The CancelReboot RPC allows the client to cancel any pending reboot requests on the target node.

Note: SR Linux does not support canceling a reboot for a subcomponent.

CancelReboot RPC structure

```
message CancelRebootResponse { }
```

7.4.7 RebootStatus RPC

The RebootStatus RPC allows the client to query the status of a reboot on the target node.

Note: SR Linux does not support querying on a single component at a time.

RebootStatus RPC structure

```
rpc RebootStatus(RebootStatusRequest) returns (RebootStatusResponse) {}
```

7.4.8 KillProcess RPC

The KillProcess RPC allows a client to kill an OS process and optionally restart it on the target node.

To specify the process to kill, the RPC must match the application name referenced in the **tools system app-management application** <*name*> command.

Mapping of termination signals to SR Linux commands

The KillProcess RPC termination signals map to SR Linux commands as follows:

Table 15: Mapping of termination signals to SR Linux commands

Termination signal	SR Linux command	Command if restart field is true
SIGNAL_TERM	**stop**	**restart** (this option runs a warm or cold restart based on what is supported)
SIGNAL_KILL	**kill**	**restart cold**
SIGNAL_HUP	**reload**	—

KillProcess RPC structure

```
rpc KillProcess(KillProcessRequest) returns (KillProcessResponse) {}
```

7.5 gNOI Healthz service

To align with general design principles of distributed systems, the gNOI Healthz service allows system components to report their own health.

Debug commands can display details about the health and state of a component. The Healthz service exposes these interfaces as queryable endpoints. In doing so, it allows clients to validate the health of components and, if unhealthy, gather device-specific data to help triage or reproduce issues.

The Healthz service allows a client to initiate health checks on a target node using the Check RPC. Alternatively, the target node can self-initiate the check and report the results to the client.

A client can then use the List or Get RPC to retrieve health events associated with the affected component and subcomponents. These health events are included in ComponentStatus messages and can be helpful to debug or further root cause the reported fault.

As part of the event response, the List or Get RPC can identify specific artifacts associated with the event, which the client can then retrieve using the Artifact RPC. The client can also call the Acknowledge RPC to acknowledge the retrieval of an event (corresponding to a series of artifacts). By default, acknowledged events are no longer included in the list of events.

The SR Linux components that support some degree of Healthz are as follows (listed in native schema):

- `.platform.control{}`
- `.platform.linecard{}`
- `.platform.chassis`
- `.platform.fan-tray{}`
- `.platform.power-supply{}`
- `.platform.fabric{}`
- `.interface{}.transceiver`

This includes all control, linecard, and fabric modules, along with power supplies and fans, individual transceivers and the chassis itself. Software components, such as routing protocol daemons, are not yet supported with the gNOI Healthz service.

SR Linux supports the following gNOI Healthz RPCs:

- Check RPC
- Get RPC
- List RPC
- Acknowledge RPC
- Artifact RPC

SR Linux uses v1.3.0 of the gNOI Healthz service protos, pulled from the following hash: https://github.com/openconfig/gnoi/blob/4f5cb0885a26a52f9c30acc236d307192c665bd8/healthz/healthz.proto.

Collection of artifacts

The workflow for collecting gNOI Healthz artifacts is as follows:

- When a system component becomes unhealthy, the system transmits health state information via telemetry that indicates the `healthz/state/status` of the component has transitioned to UNHEALTHY.
- When the client observes the transition to UNHEALTHY, it can call the Get or List RPCs to collect the events that occurred on the component.
- As the collection of some artifacts can be service impacting, all artifacts are not always automatically collected for an event. In this case, the client can call the Check RPC to collect the additional service impacting artifacts. This provides an opportunity to coordinate the collection of these artifacts when the operational risk of doing so is minimized (for example, by first removing traffic from the target node). To refer to a previously reported event, the Check RPC request must populate the **event_id** field with the ID reported in a prior Get or List response. After this Check RPC call, the client can call the Get or List RPCs to obtain the additional artifacts collected for the specified event.
- If a component returns to a healthy status, the system sends updated telemetry information to ensure that the external clients are updated about the current health status, even if the clients make no additional Healthz calls to the system.

Healthz events persistence

Healthz events created for the components are written to disk and persist across restarts of the service or software components.

SR Linux saves Healthz events in the `/etc/opt/srlinux/gnoi/healthz/events` directory, and rotates the event files to prevent overflow of the partition size. The rotation limit is set to 10 MB for all events combined.

During an unexpected CPM failover Healthz event, the creation and storage of events and artifacts can be interrupted by a CPM switchover. In this case, defer any user-initiated CPM switchover while the system is still processing the Healthz events and artifacts.

Healthz events are written to disk in intervals (every minute) to mitigate high disk pressure for frequently changing events (for example, a flapping interface).

gNMI component paths

The Healthz service works in conjunction with telemetry streamed via gNMI. The system can stream OpenConfig or native YANG paths for a specific component when the component becomes unhealthy.

To maintain Healthz parameters, SR Linux includes a **healthz** container for each of the supported components. For example, the following container maintains Healthz data for the control module:

```
augment /srl-platform:platform/srl-platform-control:control:
    +--ro healthz
        +--ro status? enumeration
        +--ro last-unhealthy? srl-comm:date-and-time-delta
        +--ro unhealthy-count? srl-comm:zero-based-counter64
```

When the Healthz service references a gNMI path (`gnoi.types.Path`), it specifies the complete path to a component, for example: `/components/component[name=FOO]`.

7.5.1 Check RPC

The Check RPC allows a client to execute a set of validations against a component. As with other Healthz operations, the component is specified using its gNMI path.

The Check RPC produces a Healthz ComponentStatus message, which contains a list of the artifacts generated from the validation process.

While the system can initiate health checks itself, these checks are limited to operations that do not impact the device functionality. Checks that are potentially service impacting require use of the Check RPC.

Note: Nokia recommends the implementation of command authorization to restrict use of these commands to prevent running unauthorized Check RPCs during normal operations.

The CheckRequest message includes an optional **event_id** field. When populated, this field directs the system to perform the check for a prior event. In this case, the device collects artifacts that were not collected automatically when the event occurred (to prevent service impacts). The collected artifacts are returned in the artifact list for the event in subsequent Get or List RPC calls.

A CheckRequest for a previous **event_id** does not overwrite previous artifacts that were collected at the time of the event.

Check RPC structure

```
rpc Check(CheckRequest) returns (CheckResponse) {}
```

7.5.2 Get RPC

After a health check, the client can use the Get (or List) RPC to retrieve the health events that are associated with a component.

The Get RPC retrieves the latest health event for the specified component. Each event consists of a collection of data that you can use to debug or root cause the fault. Unlike the List RPC, the Get RPC returns only the latest event.

The GetResponse returns a ComponentStatus message that corresponds to the latest health event for the component and each of its subcomponents. As a result, the Get RPC can return multiple ComponentStatus messages for a single component.

Each ComponentStatus message includes a set of ArtifactHeader messages that correspond to the health event, and provide identifiers and types for the artifacts returned by the system. All artifacts listed within the same ComponentStatus message share the same acknowledgement state and expiry time.

When a client invokes a Get RPC on a path, this action is not recorded as an event for this path and no health checks are performed.

Get RPC structure

```
rpc Get(GetRequest) returns (GetResponse) {}
```

7.5.3 List RPC

As an alternative to the Get RPC, the client can use the List RPC to retrieve not just the latest but all health events for the specified component and its subcomponents. Similar to the Get RPC, the List RPC also returns a series of ComponentStatus messages, which have the same semantics as those returned by the Get RPC.

By default, events that are already acknowledged are not returned.

List RPC structure

```
rpc List(ListRequest) returns (ListResponse) {}
```

7.5.4 Acknowledge RPC

A client can use the Acknowledge RPC to indicate to the target node that the client retrieved a particular (component, event) tuple. To ensure that Healthz artifact storage does not cause resource exhaustion, SR Linux can remove saved artifacts, starting with acknowledged artifacts first.

Acknowledge RPC structure

```
rpc Acknowledge(AcknowledgeRequest) returns (AcknowledgeResponse) {}
```

7.5.5 Artifact RPC

The Artifact RPC allows a client to retrieve specific artifacts that are related to an event that the target node reported in a prior List or Get response.

Because these artifacts can be large, the Artifact RPC is implemented as a server-side streaming RPC. The Artifact RPC ensures that a target node sends these potentially large artifacts only when the client explicitly requests them.

Artifacts can be core files, dumps of state (**info from state** on the specified component), or other log files. The collection of **info from state** artifacts results in the capture of any failure reasons from either the **oper-reason** or **oper-down-reason** fields.

The client can acknowledge a retrieved event corresponding to a series of artifacts. Acknowledged events are no longer returned in the list of events by default.

Events persist across restarts of the system or its hardware and software components, and they are removed only for resource management purposes. SR Linux can use the acknowledged status to remove artifacts that are no longer relevant and, if necessary, remove artifacts that are not yet acknowledged.

Artifact RPC structure

```
rpc Artifact(ArtifactRequest) returns (stream ArtifactResponse) {}
```

7.5.6 gNOI Healthz CLI commands

To allow the gNOI Healthz events to be cleared using the CLI or by a gNMI or JSON-RPC client, SR Linux supports the following CLI command:

```
tools system grpc-server testing gnoi healthz [<component>] clear
```

You can omit the component value to clear all statistics for all components, or you can use one of the following parameters to clear statistics for the specified component only:

- **chassis**: chassis component
- **control slot** <*id*>: control module component
- **fabric slot** <*id*>: fabric module component
- **fan-tray id** <*id*>: fan component
- **linecard slot** <*id*>: line card component
- **power-supply id** <*id*>: power supply component
- **transceiver interface** <*name*>: transceiver component

7.6 gNOI Packet Link Qualification service

The gNOI Packet Link Qualification service allows a client to validate the quality of the link between interfaces on two devices. This service defines a protocol to support the generation of packets from one device (the generator) to a peer device (the reflector). The reflector unconditionally loops back any packets it receives from the generator, and the service validates that those packets are sent and received. After the qualification is complete, the interfaces are restored to their previous state.

SR Linux uses v1.1.0 of the gNOI Packet Link Qualification service protos, pulled from the following hash: https://github.com/openconfig/gnoi/blob/671c4286820310a6fa7b016124ea248bf5cfbe76/packet_link_qualification/packet_link_qualification.proto.

The service allows for different generation and reflection modes of operation based on the hardware capabilities of the devices, however SR Linux only supports a subset of those modes. Regardless of the modes selected, a standard report is generated that upstream services can use to aggregate network-wide link quality.

Platform support

The gNOI Packet Link Qualification service is supported on 7250 IXR platforms.

Generation modes

The service supports the following packet generation modes:

- **Packet Generators**: The preferred mode for the generation of frames. This mode provides the most flexibility regarding packet rates, packet sizes, and packet content.
- **Packet Injectors (not supported on SR Linux)**: For devices that do not support a built-in packet generator.

Reflector modes

The service supports the following packet reflector modes:

- **ASIC Loopback**: The preferred mode of reflection. This mode allows for the loopback of frames to occur in the forwarding path of the device. It enables the most comprehensive testing results as all counters may be available to the qualification.
- **PMD Loopback (not supported on SR Linux)**: For devices that do not support ASIC loopback mode.

Operational status of interfaces during qualification

During the link qualification, the operational status of the interfaces displays as `testing`. After the qualification is complete, the operational status is restored to the previous state. In SR Linux, a port in `testing` status cannot be configured. Upper layer protocols consider an interface in `testing` state as operationally down. Control plane packets stop generating on the interface, but resume when the operational status moves back to up. In the test mode, both generating and reflecting ports indicate their status using port LEDs normally, as when forwarding traffic.

Retention and persistence of qualification results

SR Linux stores up to ten of the latest qualification results per interface; the oldest results are automatically overwritten. The results do not persist across reboots of the device nor do they persist when the warm restart is issued. During warm or cold restarts the in-progress and scheduled tests are canceled.

Supported RPCs

SR Linux supports the following gNOI Packet Link Qualification RPCs:

- Capabilities RPC
- Create RPC
- Get RPC
- Delete RPC
- List RPC

7.6.1 Service Call Flow

The following steps show a typical call flow using the gNOI Packet Link Qualification RPCs. In these examples, the client is referred to as the orchestrator.

1. Determine peer capabilities using the Capabilities RPC

```
Orchestrator                                 Generator                                Reflector

| ------------- Capabilities() -------------> |                                             |

| <------------CapbilityResponse ------------ |                                             |

| ------------------------------------- Capabilities() ------------------------------> |

| <------------------------------------ CapabilityResponse() ------------------------- |
```

2. Specify generator and reflector types and interfaces to qualify using the Create RPC

```
Orchestrator                                 Generator                                Reflector

| -------------------- Create() ------------> |                                             |

| ---------------------------------------- Create() ---------------------------------> |
   *Generator and Reflector both validate the CreateRequest*
```

3. Loop Get RPC to return the status of the qualification ID until qualification is complete

```
Orchestrator                                 Generator                                 Reflector

| -------------------- Get() -------------> |                                                  |

| <---------------- GetResponse ------------ |                                                 |

| ---------------------------------------- Get() ------------------------------------> |

| <------------------------------------ GetResponse----------------------------------- |

   *Loop until qualification complete*
```

4. Retrieve results for the qualification ID using the List RPC

```
Orchestrator                                 Generator                                 Reflector

| -------------------- List() -------------> |                                                 |

| <---------------- ListResponse ------------ |                                                |

| ------------------------------------------ List() ---------------------------------> |

| <-------------------------------------- ListResponse-------------------------------- |
```

5. Delete results as required using the Delete RPC

```
Orchestrator                                 Generator                                 Reflector

| ------------------ Delete() -------------> |                                                 |

| ---------------------------------------- Delete() ---------------------------------> |
```

7.6.2 Capabilities RPC

The Capabilities RPC allows the client to determine the generation and reflection capabilities of target peers in preparation for a packet link qualification between the two devices.

Capabilities RPC structure

```
rpc Capabilities(CapabilitiesRequest) returns (CapabilitiesResponse);
```

7.6.3 Create RPC

The Create RPC initiates a qualification operation for each interface. If it fails to create the qualification, the RPC returns an error.

When the Create RPC is called, the device interfaces are put into a forwarding mode which must not contain other traffic, either control or data. This setting can cause the generator or reflector endpoint to become unreachable.

An interface that is configured but not present in the state datastore is not eligible for testing, and `interface not found` is returned. An interface that is present in state but not in the configuration is eligible for testing.

An attempt to create a test with a duplicate ID returns an `AlreadyExists` error. This is applicable both to a test that is still running or to a completed test that is not yet deleted using the Delete RPC.

Create RPC structure

```
rpc Create(CreateRequest) returns (CreateResponse);
```

7.6.4 Get RPC

The Get RPC allows the client to retrieve the status for the specified qualification IDs.

Get RPC structure

```
rpc Get(GetRequest) returns (GetResponse);
```

7.6.5 Delete RPC

The Delete RPC allows the client to remove the qualification results for the specified qualification IDs. If a qualification is still in progress, the qualification is first canceled before it is deleted. If a qualification cannot be stopped or deleted, the RPC returns an error.

A client can cancel a running qualification early for several reasons, such as:

- Determination that the errors have already exceeded a threshold
- Desire to change the rate or MTU for a running test
- Realization that the qualification was scheduled at the wrong time

Delete RPC structure

```
rpc Delete(DeleteRequest) returns (DeleteResponse);
```

7.6.6 List RPC

The List RPC allows the client to retrieve the current results for the specified qualification IDs from the generator and reflector nodes.

List RPC structure

```
rpc List(ListRequest) returns (ListResponse);
```

7.6.7 Packet Link Qualification CLI commands

To allow the gNOI Packet Link Qualification behavior to be invoked in the CLI or by a gNMI or JSON-RPC client, SR Linux supports the following CLI commands:

- **system packet-link-qualification profile** *<profile>*
- **tools interface** *<name>* **packet-link-qualification start id** *<id>* **qualification-profile** *<profile>*
- **info from state interface** *<name>* **packet-link-qualification result** *<id>*
- **tools interface** *<name>* **packet-link-qualification cancel id** *<id>*

7.6.7.1 Configuring a gNOI packet link qualification profile using the CLI

Procedure

As an alternative to using the gNOI RPCs, you can also configure a packet link qualification profile using the **system packet-link-qualification profile** CLI command.

Example: Configure a packet link qualification profile (generator)

```
--{ + candidate shared default }--[  ]--
# info system packet-link-qualification profile p1
    system {
        packet-link-qualification {
            profile p1 {
                rpc {
                    pre-sync-duration 10
                    setup-duration 20
                    duration 200
                    post-sync-duration 5
                    teardown-duration 15
                }
                packet-generator {
                    packet-rate 152737
                    packet-size 8184
                }
            }
        }
    }
```

Example: Configure a packet link qualification profile (reflector)

```
--{ + candidate shared default }--[  ]--
# info system packet-link-qualification profile p1
    system {
        packet-link-qualification {
            profile p1 {
                rpc {
                    pre-sync-duration 10
                    setup-duration 20
                    duration 200
                    post-sync-duration 20
                    teardown-duration 15
                }
                asic-loopback {
                }
            }
```

7.6.7.2 Running a packet link qualification test using the CLI

Procedure

You can run a gNOI packet link qualification test using the **tools interface packet-link-qualification start id qualification-profile** CLI command and obtain results of the test using the **info from state** command. To stop or cancel the test, use the **tools interface packet-link-qualification cancel id** command.

Example: Start a packet link qualification test on the generator

```
--{ + candidate shared default }--[  ]--
# tools interface ethernet-5/4 packet-link-qualification start id id1 qualification-
profile p1
```

Example: Start a packet link qualification test on the reflector

```
--{ + candidate shared default }--[  ]--
# tools interface ethernet-8/4 packet-link-qualification start id id1 qualification-
profile p1
```

Example: Obtain test results from the generator

```
--{ + candidate shared default }--[  ]--
# info from state interface ethernet-5/4 packet-link-qualification result 1
    interface ethernet-5/4 {
        packet-link-qualification {
            result 1 {
                oper-state completed
                packets-sent 30545942
                packets-received 30545942
                packets-error 0
                packets-dropped 0
                start-time "2024-02-20T16:05:03.050Z (an hour ago)"
                end-time "2024-02-20T16:08:23.050Z (an hour ago)"
                expected-rate 1249999608
                qualification-rate 1249940692
            }
        }
    }
```

Example: Obtain test results from the reflector

```
--{ + candidate shared default }--[  ]--
# info from state interface ethernet-8/4 packet-link-qualification result 1
    interface ethernet-8/4 {
        packet-link-qualification {
            result 1 {
                oper-state completed
                packets-sent 30599067
                packets-received 30599067
                packets-error 0
                packets-dropped 0
                start-time "2024-02-20T16:05:02.969Z (an hour ago)"
                end-time "2024-02-20T16:08:22.969Z (an hour ago)"
            }
        }
    }
```

Example: Stop the test on the generator

```
--{ + candidate shared default }--[  ]--
# tools interface ethernet-5/4 packet-link-qualification cancel id id1
```

Example: Stop the test on the reflector

```
--{ + candidate shared default }--[  ]--
# tools interface ethernet-8/4 packet-link-qualification cancel id id1
```

7.7 gNOI configuration

SR Linux supports gNOI services using the gRPC server configuration. To enable gNOI support, enable the gRPC server.

The session between the gNOI client and SR Linux must be encrypted using TLS.

See the "Management servers" chapter in the *SR Linux Configuration Basics Guide* for information about how to configure the gRPC server.

7.7.1 Configuring gNOI services

About this task

As part of the gRPC server configuration, you can also specify which individual gNOI services to enable.

Procedure

To enable gNOI services, use the **system grpc-server** <*network-instance*> **services** command.

Example: Enable gNOI services

```
# info system grpc-server mgmt

    system {
        grpc-server mgmt {
            admin-state enable
            timeout 7200
            rate-limit 1500
            session-limit 20
            metadata-authentication true
            yang-models native
            tls-profile tls-profile-1
            network-instance mgmt
            port 50052
            oper-state up
            services [
                gnoi.packet_link_qualification
            ]
            source-address [
                ::
            ]
            gnmi {
                commit-confirmed-timeout 0
                commit-save false
                include-defaults-in-config-only-responses false
            }
            unix-socket {
                admin-state disable
                socket-path ""
            }
        }
    }
```

8 gNSI

gRPC Network Security Interface (gNSI) is a set of RPCs that allow a client to define the security configuration of a device and to retrieve security information. gNSI is maintained by the OpenConfig project on Github, and contains a set of RPCs as well as extensions to gNMI and the OpenConfig YANG modules.

SR Linux supports the following gNSI services:

- gNSI Authz service
- gNSI Certz service

8.1 gNSI Authz service

The gNSI Authz service allows clients to configure and manage a gRPC-level authorization policy that defines which users can access which gRPCs on the target node. This policy is defined by a JSON string and controls access to all gRPC servers to prevent access from malicious actors.

Note: Only one policy is active for the entire system.

Unlike role-based access control, which provides authorization of system YANG paths, the Authz service authorizes the use of individual gRPCs; for example access to the `/gnmi.gNMI/Get` RPC, rather than to any individual path that the Get RPC can retrieve.

Typical Authz service steps

In the typical Authz service process, the client performs the following steps:

1. Upload the authorization policy by sending an UploadRequest to the target node using the Rotate RPC.
2. Verify that the authorized users can access the required gRPCs (and that non-authorized users are denied) by sending a ProbeRequest using the Probe RPC.
3. Lock in the updates by sending a FinalizeRequest using the Rotate RPC.
4. As required, retrieve details of the currently active policy by sending a GetRequest using the Get RPC.

8.1.1 Authz policy

The Authz policy consists of the following:

- **name**: name for the policy defined by the client
- **allow_rules list** (mandatory): list of allow rules for the policy
- **deny_rules list** (optional): list of deny rules for the policy

Allow and deny rules

Each allow or deny list is comprised of instances of rules. Each rule consists of the following:

- **name**: name for the rule defined by the client; similar to the policy name
- **source**: a holding container with a single source type of **principals**:
  - **principals**: a list of principals that must match against user objects, be it one of the users defined in **system aaa authentication user**, a remotely resolved user from a **system aaa server-group**, or the **admin**
- **request** contains:
  - **paths**: a list of gRPC RPC paths (in format `pkg.Service/Rpc`) that this policy applies to; for example: `/gnoi.os.OS/Install` (if no paths are provided, all paths are matched)
  - **headers**: a list of key/value pairs matching HTTP headers

    Note: SR Linux does not support matching on **headers**.

Matching source and request

Both the source and request fields must match in order for a rule to match. If both source and request are empty, the rule always matches. Empty source and request can serve as a wildcard in a **deny_rules** list to deny all RPCs or in an **allow_list** to accept all RPCs except those matching in the **deny_rules** list.

If only the source is empty, the paths provided in the request field apply to all users system wide. Similarly if only the request is empty then the action (deny or allow) is applied to all users provided in the source. If no paths are specified in the request, all paths are matched.

Example: Authz policy example

The following shows an example policy in which two administrators Alice and Bob have the ability to execute any of the gNSI Authz RPCs.

```
{
  "name": "gNSI.authz policy",
  "allow_rules": [{
    "name": "admin-access",
    "source": {
     "principals": [
      "spiffe://nokia.com/sa/alice",
      "spiffe://nokia.com/sa/bob"
      ]
    },
    "request": {
      "paths": [
        "/gnsi.authz.Authz/*"
      ]
    }
  }]
}
```

Authz policy persistence

As it is a requirement to persist a policy after it has been finalized, on finalization the current policy is written to `/etc/opt/srlinux/gnsi/authz-policy.json`. This policy is read by the SR Linux gNSI server on startup and exposed as the active policy.

It is an Authz requirement that if a policy is configured, no gRPC servers (this includes P4RT, gRIBI, gNOI, gNMI, and gNSI externally) can start until the policy is successfully activated. The Authz scope is to protect access to system RPCs from malicious entities, so a policy must be active before ports are opened.

Default Authz policy

As part of the default SR Linux configuration, the Authz service is running, and a default Authz policy controls access to every active gRPC service. The result of the default policy is that any *authenticated* user is permitted full access to any currently active service and gRPC. This does not overrule any subsequent role-based access control that exists on the system.

```
{
 "name": "Default policy",
 "deny_rules": [],
 "allow_rules": [
  {
   "name": "admin-access",
   "source": {
    "principals": [
     "*"
    ]
   },
   "request": {
    "paths": [
     "/*",
     ""
    ],
    "headers": []
   }
  }
 ]
}
```

8.1.2 Authz authorization logic and interactions

To validate a gRPC authorization, gNSI performs the following actions:

1. Check the user and RPC combination against any **deny_rules**. If a match is found, the request is denied. If no match is found or there are no **deny_rules**, execute the next step.
2. Check for a match against any **allow_rules**. If a match is found, the request is permitted. If no match is found, deny the request.

Note: The gNSI Authz behavior differs from role-based access control, in that, if any **deny_rules** are matched, then the request is denied, even if a more specific path exists in **allow_rules**.

Authz interactions with role-based access control and service authorization

The Authz service operates in addition to existing role-based access control and service authorization. As a result, to provide a user access to a gRPC, you must configure their access under role-based access control and service authorization (for example, **system aaa authorization role services gnmi**) in addition to enabling access using the Authz service. The **system aaa authorization role services** command also supports defining service authorization for **gnsi**.

Order of authorization

The order of authorization is as follows:

1. Check service authorization first; only if the user is authorized for the service are they admitted.
2. If the interface is a gRPC interface, check the Authz policy to see if the user is authorized for the specified RPC.
3. If the user is configuring or reading any paths, perform role-based access control.

8.1.3 Default TLS profile

To allow the gNMI (and gNSI) server to start as part of the default configuration, the factory default configuration includes a TLS profile named **__default__**. This default policy does not authenticate the identity of connecting clients. You can view the policy using the **info from state system tls server-profile __default__** command.

```
# info from state system tls server-profile __default__
   system {
        tls {
            server-profile __default__ {
                key $aes1$4lL6SVq5G8=$FqANWbk8TQmrGYB1cTuaplFUZTZlAP992Debrq4X2ZjfOde4wA
                ...
                certificate "-----BEGIN CERTIFICATE-----
MIIFLzCCAxegAwIBAgIUBQFYPrWyBalNUfTNCBUAMiWrk2kwDQYJKoZIhvcNAQEL
BQAwHTEOMAwGA1UEAwwFc3JsZDUxCzAJBgNVBAYTAlVTMCAXDTI0MDIyMDIzNTY0
...
-----END CERTIFICATE-----
"
                authenticate-client false
                dynamic true
                cipher-list [
                    ecdhe-ecdsa-aes256-gcm-sha384
                    ecdhe-ecdsa-aes128-gcm-sha256
                    ecdhe-rsa-aes256-gcm-sha384
                    ecdhe-rsa-aes128-gcm-sha256
                ]
            }
        }
    }
```

SR Linux also supports a default TLS configuration option for the gRPC server (**system grpc-server** *name* **default-tls-profile [true | false]**) that controls whether the system uses the default TLS profile if none are configured using the **tls-profile** option. If **default-tls-profile** is set to **false** (the default value), and no **tls-profile** is defined, then TLS is disabled entirely from the system.

8.1.4 Supported RPCs

SR Linux supports the following Authz service RPCs:

- Rotate RPC
- Probe RPC
- Get RPC

8.1.5 Rotate RPC

The Authz Rotate RPC allows a client to replace an existing gRPC authorization policy on the target node.

If any steps in the Rotate RPC process fail, the target node rolls back any changes made by the RPC.

Only one Rotate RPC can be in progress at a time. If a client attempts to call the RPC while another call is already in progress, the second call is rejected with a gRPC error of `UNAVAILABLE`.

Rotate RPC structure

```
rpc Rotate(stream RotateAuthzRequest) returns (stream RotateAuthzResponse);
```

8.1.6 Probe RPC

The Authz Probe RPC tests the currently active policy. It allows the Authz policy engine on the target node to provide a response to a specified gRPC call without actually performing the gRPC operation. The current policy can be one that has not been finalized, but this is not a requirement.

The Probe RPC expects a single ProbeRequest message, and returns a ProbeResponse. The ProbeRequest does not result in an actual gRPC call being made; it simply validates whether the call can succeed with the provided user and RPC information.

Probe RPC structure

```
rpc Probe(ProbeRequest) returns (ProbeResponse);
```

8.1.7 Get RPC

The Authz Get RPC returns the currently active gRPC authorization policy, as well as the policy version and created-on timestamp.

Get RPC structure

```
 rpc Get(GetRequest) returns (GetResponse);
```

8.1.8 gNSI Authz tools and state commands

To allow the gNSI Authz behavior to be invoked in the CLI or by a gNMI or JSON-RPC client, SR Linux supports the following tools commands:

- **tools system aaa authorization authz-policy clear**
- **tools system aaa authorization authz-policy probe**
- **tools system aaa authorization authz-policy remove**
- **tools system aaa authorization authz-policy rotate**

Note: Performing a **rotate** operation using the tools commands triggers an immediate finalization.

8.1.8.1 Clearing Authz policy counters using the tools command

Procedure

To clear Authz policy counters, use the **tools system aaa authorization authz-policy clear** command.

Example: Clear Authz policy counters

```
--{ candidate shared default }--[  ]--
# tools system aaa authorization authz-policy clear
```

8.1.8.2 Probing Authz policy counters using the tools command

Procedure

To probe an Authz policy, use the **tools system aaa authorization authz-policy probe** command. The **rpc** parameter specifies which RPC to test, and the **user** parameter specifies which user to use for the test.

Example: Probe Authz policy counters

```
--{ candidate shared default }--[  ]--
# /tools system aaa authorization authz-policy probe rpc /gnmi.gNMI/Get user admin
/system/aaa:
    Authz authorization policy version '1' action for user 'admin' and rpc '/gnmi.gNMI/
Get' is ACTION_PERMIT
```

8.1.8.3 Removing an Authz policy using the tools command

Procedure

To remove an Authz policy, use the **tools system aaa authorization authz-policy remove** command.

Given that there is only one system-wide gRPC authorization policy, this command reverts to the factory default authorization policy, which authorizes any gRPC call for every authenticate user.

Example: Remove Authz policy

```
--{ candidate shared default }--[  ]--
# tools system aaa authorization authz-policy remove
```

8.1.8.4 Rotating an Authz policy using the tools command

Procedure

To rotate an Authz policy, use the **tools system aaa authorization authz-policy rotate** command, which supports the following parameters:

- **created-on**: sets the created-on value for the new policy
- **policy**: specifies the gRPC authorization policy as a JSON-formatted string
- **version**: specifies a version string to store with the policy

Unlike the Rotate RPC, the **tools system aaa authorization authz-policy rotate** command triggers an immediate finalization.

Example: Rotate Authz policy

```
--{ candidate shared default }--[  ]--
# / tools system aaa authorization authz-policy rotate policy "{\"name\": \"foo\", \
"allow_rules\": { \"name\": \"dev\", \"source\": { \"principals\": [
\"admin\", \"foo\", \"bar\" ] }, \"request\": { \"pat hs\": [ \"/*\", \"\" ] } } }"
 created-on 10 version 1
/system/aaa:
    Authz authorization policy has been rotated and finalized (version '' created on
 '1970-01-01T00:00:10.000Z')
```

8.1.8.5 Displaying the currently installed Authz policy

Procedure

To display the currently installed Authz policy, use the **info from state system aaa authorization authz-policy** command.

Example: Display the currently installed Authz policy

```
# info from state system aaa authorization authz-policy
    system {
        aaa {
            authorization {
                authz-policy {
                    version 2023-06-01
                    created-on "2024-02-16T17:29:08.721Z (40 minutes ago)"
                    policy "{
 \"name\": \"Default policy\",
 \"deny_rules\": [],
 \"allow_rules\": [
  {
   \"name\": \"admin-access\",
   \"source\": {
    \"principals\": [
     \"*\"
    ]
   },
   \"request\": {
    \"paths\": [
     \"/*\",
     \"\"
    ],
    \"headers\": []
   }
  }
 ]
}
"
                }
            }
        }
    }
```

8.2 gNSI Certz service

The gNSI Certz service allows a client to replace a certificate, trust bundle, certificate revocation list (CRL), or some combination of these artifacts on a target node.

Typical Certz service steps

In the typical certificate rotation process, the client performs the following steps using the Rotate RPC:

1. Either generate the CSR on the client side, or direct the target node to generate the CSR using a GenerateCSRRequest.
2. Obtain a signature for the certificate from the CA (out of band of the RPC).
3. Upload the signed certificate to the target node using an UploadRequest.
4. Verify (out of band of the RPC) that the services that use the new certificate bundle continue to operate normally.
5. Send a FinalizeRequest to the target node to lock in the updates.

Entity messages

The Certz service uses Entity messages to define the Certz artifacts, which can be any of the following:

- **Certificate**: An X.509 certificate and optional private key that is either PEM or DER encoded. The specific encoding is indicated in fields carried in children of the Entity message. A certificate also specifies any intermediate parent certificates. It is equivalent to the certificate displayed using the **system tls server-profile** *name* **certificate** command.
- **Trust bundle**: The certificate bundle used to validate outbound connections, and inbound clients when mTLS is enabled. This is a typical system trust bundle available in `/etc/ssl/certs`. It is equivalent to the trust anchor displayed using the **system tls server-profile** *name* **trust-anchor** command.
- **CRL**: The CRL bundle, which lists certificates that have been revoked and can no longer be used to validate connections. It is equivalent to the CRL displayed using the **system tls server-profile** *name* **certificate-revocation-list** command.

All together, these artifacts make up a single profile. In a single Rotate RPC call, a client can rotate all artifacts, or only specific artifacts.

To improve performance on the target node, certificates can be ordered. Groups of chained certificates (namely the **trust-anchor** and **crl**) appear last, and within the group, the root certificate must be the last certificate in the chain; for example: CertA, CertB, CertB-Root, CertC, CertC-Intermediate, CertC-Root.

Note: SR Linux can support an unordered bundle.

Default TLS profile

The default TLS profile is available at system startup and uses system-generated self-signed certificates and private keys. The default profile is saved to disk with other TLS profiles at `/etc/opt/srlinux/tls`. This default profile allows the gRPC server to start as part of the default configuration to meet the requirements of the Authz service.

TLS profiles can be populated using the Certz service or configured via the CLI using the **system tls server-profile** command. If a conflict exists in the configuration, the CLI configuration takes precedence.

Certz state paths

Certz state paths are available under **system tls server-profile certz**. These paths indicate the certificate, trust bundle, and CRL in use by each specific instance of a gRPC server.

The Certz JSON files are stored in `/etc/opt/srlinux/gnsi`, while the CRL PEM files are stored in `/etc/opt/srlinux/tls`.

Only users in the **tls** group can read these artifacts, and only the **srlinux** user can modify them.

Supported RPCs

SR Linux supports the following Certz service RPCs:

- Rotate RPC
- CanGenerateCSR RPC
- AddProfile RPC
- DeleteProfile RPC
- GetProfileList RPC

8.2.1 Rotate RPC

The Certz Rotate RPC allows a client to replace an existing certificate, trust bundle, CRL, or some combination of these artifacts on the target node. Either the target node or the client can generate the CSR for the new device certificate. If the client generates the CSR, it must also provide the corresponding private key with the signed certificate.

Only one Rotate RPC can be active at once. If SR Linux detects an additional Rotate RPC, it returns an error.

The Rotate RPC accepts a stream of RotateCertificateRequest messages and returns a stream of RotateCertificateResponse messages in response. A client uses the RPC to upload a certificate by specifying the certificate in a **rotate_request** before testing the new certificate (out of band of the RPC) and finalizing the result with a **finalize_rotation**. Until the target node receives the **finalize_rotation**, the change is transient and not persistent on disk. If the RPC is canceled for any reason before a **finalize_rotation** is received, then the system reverts all services using TLS to the previous values.

If the stream is broken or any of the steps fail, the target node rolls back to the original state, reverting any changes to any artifact.

Rotate RPC structure

```
rpc Rotate(stream RotateCertificateRequest) returns (stream RotateCertificateResponse);
```

8.2.1.1 Rotate RPC use cases

The following sections describe a number of Rotate RPC use cases each presenting the expected sequence of message exchange.

Case 1: Client generates the CSR and gets it signed

1. The Rotate RPC stream begins.

```
Client <------          Rotate() RPC stream begins         ------> Target node
```

2. The client generates the CSR and gets the certificate signed by the CA (typically done before initiating the stream).
3. The client sends the signed certificate with private key to the target node.

   The client can optionally include a trust bundle, a CRL, or both.

```
Client ------>  UploadRequest(certificate, [trust_bundle],  ------> Target node
                       [certificate_revocation_list])

Client <------                UploadResponse                <------ Target node
```

4. To validate that the certificate works, the client attempts to create new connections to the target node using the new certificate. If the new connections fail, the client cancels the RPC, forcing the target node to roll back all certificates included in the RPC.

   If a new CRL bundle is included in the UploadRequest in step 3, the client also attempts to establish new connections to the target node using the revoked certificates. In this case, failed connections validate that the certificates have been revoked.
5. After successful validation, the client sends a final commit.

```
Client ------>                FinalizeRequest               ------> Target node
```

Case 2: Target node generates the CSR and the client gets it signed

1. Rotate RPC stream begins:

```
Client <------           Rotate() RPC stream begin          ------> Target node
```

2. The client sends a GenerateCSRRequest to the target node, and the target node provides the CSR in response:

```
Client ------>              GenerateCSRRequest              ------> Target node

Client <------              GenerateCSRResponse             <------ Target node
```

3. The client obtains a signature for the certificate from the CA.
4. The client sends the signed certificate without private key to the target node (the target node already has the private key because it generated the CSR).

   The client can optionally include a trust bundle, a CRL, or both.

```
Client ------>  UploadRequest(certificate, [trust_bundle],  ------> Target node
                      [certificate_revocation_list])

Client <------                 UploadResponse               <------ Target node
```

5. The client tests and validates the certificate, using the same validation step as in Case 1.

6. After successful validation, the client sends a final commit.

```
Client ------>                FinalizeRequest              ------> Target node
```

Case 3: Client changes only the trust bundle

1. The Rotate RPC stream begins:

```
Client <------             Rotate() RPC stream begin       ------> Target node
```

2. The client sends a Certificate Authority Bundle chain to the target node.

   The client can optionally include a CRL.

```
Client ------>           UploadRequest(trust_bundle,        ------> Target node
                        [certificate_revocation_list])

Client <------                   UploadResponse             <------ Target node
```

3. The client tests and validates the certificate, using the same validation step as in Case 1.
4. After successful validation, the client sends a final commit.

```
Client ------>                  FinalizeRequest              ------> Target node
```

Case 4: Client uploads a CRL

1. The Rotate RPC stream begins:

```
Client <------           Rotate() RPC stream begin           ------> Target node
```

2. The client sends a CRL bundle to the target node.

```
Client ------>  UploadRequest([certificate_revocation_list])   ------> Target node

Client <------                 UploadResponse                  <------ Target node
```

3. The client validates the CRL by attempting to establish a new connection to the target node using the revoked certificates. Failed connections validate that the certificates have been revoked.
4. After successful validation, the client sends a final commit.

```
Client ------>                 FinalizeRequest                  ------> Target node
```

8.2.2 CanGenerateCSR RPC

The Certz CanGenerateCSR RPC tests whether the target node is able to generate a CSR given provided parameters. The CanGenerateCSR RPC consists of a single CanGenerateCSRRequest message and returns a CanGenerateCSRResponse. The client includes the parameters that it wants the target node to test for the CSR generation, including:

- **key_type**: the algorithm used for generation of the key pair (for example, **KEY_TYPE_RSA**)
- **signature_algorithm_type**: the signature algorithm used to generate the key pair (for example: **SIGNATURE_ALGORITHM_SHA512_WITH_RSA**)

- **certificate_type**: the type of certificate to create (for example, **CERTIFICATE_TYPE_X509**)
- **key_size_bits**: a uint32, indicating the size of the key in bits (for example, **2048**)

CanGenerateCSR RPC structure

```
rpc CanGenerateCSR(CanGenerateCSRRequest) returns (CanGenerateCSRResponse);
```

8.2.3 AddProfile RPC

The Certz AddProfile RPC allows a client to add a new TLS profile to the target node. All elements of the added profile (certificate, CA trust bundle, and CRLs) are initially empty. The client must subsequently populate the artifacts using a Rotate RPC.

If a client attempts to add a pre-existing profile, the attempt is rejected with an error.

AddProfile RPC structure

```
rpc AddProfile(AddProfileRequest) returns (AddProfileResponse);
```

8.2.4 DeleteProfile RPC

The Certz DeleteProfile allows a client to remove an existing TLS profile.

The **__default__** profile used by the gRPC server cannot be deleted. If a client attempts to delete the **__default__** profile, the attempt is rejected with an error.

DeleteProfile RPC structure

```
rpc DeleteProfile(DeleteProfileRequest) returns (DeleteProfileResponse);
```

8.2.5 GetProfileList RPC

The Certz GetProfileList RPC allows a client to retrieve a list of the TLS profile IDs that are present on a target node.

GetProfileList RPC structure

```
rpc GetProfileList(GetProfileListRequest) returns (GetProfileListResponse);
```

8.2.6 gNSI Certz tools commands

To allow the gNSI Certz RPC behavior to be invoked in the CLI or by a gNMI or JSON-RPC client, SR Linux supports the following tools commands:

- **tools system tls server-profile** <*name*> **certz rotate**
- **tools system tls server-profile** <*name*> **certz remove**

8.2.6.1 Rotating a Certz profile using the tools command

Procedure

To rotate a profile on the system, use the **tools system tls server-profile** <*name*> **certz rotate** command, which supports the following parameters:

- **certificate**: specifies the new certificate to use
- **created-on**: sets the created on value for the new policy
- **crl**: specifies a bundle of certificates to add to the CRL
- **key**: specifies the new private key to use
- **trust-anchor**: specifies a certificate chain to use as a trust anchor
- **version**: specifies the version string to store with the policy

Example: Rotate Certz profile using tools command

The following example rotates the **test-certz-profile** with certificate **test-cert**, key **test-key**, CRL **test-crl**, trust anchor **test-anchor**, and assigned version of **1**.

```
--{ candidate shared default }--[  ]--
# tools system tls server-profile test-certz-profile certz rotate certificate test-cert
 key test-key crl test-crl trust-anchor test-anchor version 1
/system/tls/server-profile[name=test-certz-profile]:
    Certz SSL profile 'test-certz-profile' has been added

/system/tls/server-profile[name=test-certz-profile]:
    Certz SSL profile 'test-certz-profile' has been rotated and finalized (version '1'
 created on '2024-02-16T16:20:03.000Z')
```

8.2.6.2 Removing a Certz profile using the tools command

Procedure

To remove a profile from the system, use the **tools system tls server-profile certz remove** command.

Example: Remove a Certz SSL profile using tools command

```
--{ candidate shared default }--[  ]--
# tools system tls server-profile test-certz-profile certz remove
/system/tls/server-profile[name=test-certz-profile]:
    Certz SSL profile 'test-certz-profile' has been removed
```

8.2.7 gNSI Certz state commands

Certz state paths are available under **system tls server-profile**. These paths indicate the certificate, trust bundle, and CRL in use by each specific instance of a gRPC server.

To display the currently installed Certz policy artifacts, use the following commands:

- **info from state system tls server-profile** <*name*> **certz certificate**
- **info from state system tls server-profile** <*name*> **certz crl**

- **info from state system tls server-profile** <*name*> **certz ssl-profile-id**
- **info from state system tls server-profile** <*name*> **certz trust-anchor**

Example: Info from state for empty TLS server profile dyn

```
--{ running }--[  ]--
# info from state system tls server-profile dyn
    system {
        tls {
            server-profile dyn {
                dynamic true
                certz {
                    ssl-profile-id dyn
                }
            }
        }
    }
```

Example: Info from state for rotated TLS server profile dyn

```
--{ running }--[  ]--
# info from state system tls server-profile dyn
    system {
        tls {
            server-profile dyn {
                key $aes1$ATQxUhdz4QGeu28=$HrE...
                certificate "-----BEGIN CERTIFICATE----- MIIGW...  ----END CERTIFICATE----
-"
                authenticate-client true
                trust-anchor "-----BEGIN CERTIFICATE----- MIIFe...  -----END CERTIFICATE--
---"
                dynamic true
                cipher-list [
                    ecdhe-ecdsa-aes256-gcm-sha384
                    ecdhe-ecdsa-aes128-gcm-sha256
                    ecdhe-rsa-aes256-gcm-sha384
                    ecdhe-rsa-aes128-gcm-sha256
                ]
                certz {
                    ssl-profile-id dyn
                    certificate {
                        version 7
                        created-on "2023-08-19T08:56:45.000Z (6 months ago)"
                    }
                    trust-anchor {
                        version 9
                        created-on "2023-08-19T14:53:09.000Z (6 months ago)"
                    }
                }
            }
        }
    }
```

8.3 gNSI configuration

SR Linux supports gNSI services using the gRPC server configuration. To enable gNSI support, enable the gRPC server and specify gNSI as one of the enabled gRPC services for a specified network instance (enabled by default on the mgmt network instance).

Like gNMI, the session between the gNSI client and SR Linux can be encrypted using TLS.

gNMI and gNSI servers enabled by default

By default, the gNMI and gNSI servers are enabled in the mgmt network instance.

```
# info system grpc-server mgmt    system {
        grpc-server mgmt {
            admin-state enable
            default-tls-profile true
            network-instance mgmt
            services [
                gnmi
                gnsi
            ]
        }
    }
```

8.3.1 Configuring gNSI services

About this task

As part of the gRPC server configuration, you can also specify which individual gNSI services to enable.

Procedure

To enable gNSI services, use the **system grpc-server** <*name*> **services** command.

Note: If you enter **services [gnsi]** all gNSI services are enabled. To enable an individual service only, enter for example **services [gnsi.authz]** or **services [gnsi.certz]**, and omit the **[gnsi]** parameter.

Example: Enable gNSI services

```
# info system grpc-server mgmt
    system {
        grpc-server mgmt {
            admin-state enable
            network-instance mgmt
            services [
                gnsi
            ]
        }
    }
```

See the "Management servers" chapter in the SR Linux Configuration Basics Guide for related information about how to configure the gRPC server.

9 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.

9.1 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 16: 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

9.1.1 method options

The following table defines supported JSON RPC method options.

Table 17: 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.

9.1.2 params options

The following table defines valid JSON RPC params options.

Table 18: 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.

9.2 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.

Compressed JSON responses using gzip

If a client request specifies to use HTTP compression, the JSON RPC server compresses the responses using gzip.

9.3 Candidate mode

JSON uses its own private exclusive candidate that restricts other users or services from making simultaneous changes to a configuration. If another exclusive session is already active, any commits to the configuration using the JSON-RPC set method fail with an error.

The JSON-RPC server uses the private exclusive candidate name **jsonrpc-**<*n*>, where <*n*> is a 32-bit number starting at 1 that increments for every request received, but resets on a JSON-RPC server restart.

9.4 Logical expressions

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

9.5 Interactive CLI commands

Interactive CLI commands that either run until receiving an indication to end the process, or that require interactive user input typically cannot be called using non-interactive interfaces such as JSON-RPC's CLI method.

To allow you to perform critical tasks related to service validation and network testing using programmable interfaces, SR Linux allows non-interactive interfaces to call the following interactive CLI commands:

- **ping** and **ping6** when the **-c** argument is included to ensure finite execution time
- **traceroute** and **traceroute6**
- **bash cat** <*file path*>
- **bash echo** <*content*> <*file*>

9.6 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

9.6.1 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"
}
```

9.6.2 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"
}
```

9.6.3 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"
}
```

9.6.4 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"
}
```

9.6.5 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"
}
```

10 SNMP

SR Linux supports the Simple Network Management Protocol (SNMP) versions SNMPv2c and SNMPv3, which allow SNMP managers to read information about the system for device monitoring.

10.1 SNMP architecture

SNMP is an application-layer protocol that enables communication between managers (the management system) and agents (the network devices). It provides a standard framework to monitor devices in a network from a central location.

An SNMP manager can get a value from an SNMP agent. The manager uses definitions in the management information base (MIB) to perform operations on the managed device such as retrieving values from variables and processing traps.

The following actions can occur between the agent and the manager:

- The manager gets information from the agent.
- The agent sends traps to notify the manager of significant events that occur on the system.

10.2 Management information base

A management information base (MIB) is a formal specifications document with definitions of management information used to remotely monitor, configure, and control a managed device or network system. The agent’s management information consists of a set of network objects that can be managed with SNMP. Object identifiers are unique object names that are organized in a hierarchical tree structure. The main branches are defined by the Internet Engineering Task Force (IETF). When requested, the Internet Assigned Numbers Authority (IANA) assigns a unique branch for use by a private organization or company. The branch assigned to Nokia (TiMetra) is 1.3.6.1.4.1.6527.

The SNMP agent provides management information to support a collection of IETF specified MIBs and a number of MIBs defined to manage devices and network data unique to the Nokia router.

MIB files are packaged with each release and are available on the Nokia support portal or in `/opt/srlinux/snmp/MIBs.zip`.

10.3 SNMP network instance configuration

About this task

The SNMP agent must first be configured to run in each network instance used to monitor the system. Then, access groups can be configured to read information or trap groups can be configured to send traps.

To configure the SNMP agent, use the **system snmp** commands.

Example: SNMP network instance configuration

In the following example, the SNMP agent is running in the **default** network instance.

```
--{ * candidate shared default }--[  ]--
# info system snmp
    system {
        snmp {
            network-instance default {
                admin-state enable
            }
        }
    }
```

10.4 SNMP versions and configuration

The SNMP agent supports two versions of the SNMP protocol:

- SNMPv2c is a community-based administrative framework for SNMPv2. SNMPv2c uses a community string for authentication.
- SNMPv3 uses the User-based Security Model (USM) for user authentication with passwords.

10.4.1 SNMPv3 authentication and privacy protocols

The User-based Security Model (USM) for the authentication, encryption, and decryption of SNMPv3 packets is supported with configurable authentication and privacy protocols.

SNMPv3 authentication protocols

The following SNMPv3 authentication protocols are supported:

- HMAC-MD5-96
- HMAC-SHA-96
- HMAC-SHA-224
- HMAC-SHA-256
- HMAC-SHA-384
- HMAC-SHA-512

SNMPv3 privacy protocols

The following SNMPv3 privacy protocols are supported:

- CBC-DES
- CFB128-AES-128
- CFB128-AES-192
- CFB128-AES-256

SNMPv3 authentication and privacy protocol combinations

The following combinations of authentication and privacy protocols are not allowed because the hash does not produce enough bytes to use as a key:

- HMAC-MD5-96 (16 bytes) and CFB128-AES-192 (24 bytes)
- HMAC-MD5-96 (16 bytes) and CFB128-AES-256 (32 bytes)
- HMAC-SHA1-96 (20 bytes) and CFB128-AES-192 (24 bytes)
- HMAC-SHA1-96 (20 bytes) and CFB128-AES-256 (32 bytes)
- HMAC-SHA2-224 (28 bytes) and CFB128-AES-256 (32 bytes)

10.4.2 Configuring SNMPv2c

About this task

SR Linux supports SNMPv2c, which allows SNMP managers to read information about the system for device monitoring.

Procedure

To configure the SNMP agent, use the **system snmp** commands.

Example: SNMPv2c access group configuration

In SNMPv2c, the **community** value is mandatory and cannot contain spaces.

Optionally, the **prefix-list** value defines which managers can use the community (both IPv4 and IPv6 addresses) and is only supported in SNMPv2c.

The **community-entry** value cannot be the same as the **community** value because this reveals the plaintext value of the **community**.

In the following example, the SNMPv2c agent uses an access group for get requests. The minimum security level is configured.

```
--{ * candidate shared default }--[  ]--
# info system snmp
    system {
        snmp {
            access-group ag1 {
                admin-state enable
                security-level no-auth-no-priv
                community-entry ce1 {
                    community $aes1$AW/5wLmAOcTPhG8=$aFJfMhdHwSGTplCfsDgBPA==
                    prefix-list [
                        10.1.1.1/32
                    ]
                }
            }
            network-instance default {
                admin-state enable
            }
        }
    }
```

Example: SNMPv2c trap group configuration

In the following example, the SNMPv2c agent uses a trap group within the **default** network instance. The minimum security level is configured. The SNMPv2c **community** value is configured using the **community-entry** parameter.

```
--{ * candidate shared default }--[  ]--
# info system snmp
    system {
        snmp {
            trap-group tg1 {
                admin-state enable
                network-instance default
                destination destination1 {
                    admin-state enable
                    address 10.2.2.2
                    security-level no-auth-no-priv
                    community-entry ce1 {
                        community $aes1$AWOTWOQo41n22m8=$XD4pX1F7pWJFtTdgwjf23w==
                    }
                }
            }
            network-instance default {
                admin-state enable
            }
        }
    }
```

10.4.3 Configuring SNMPv3

About this task

SR Linux supports SNMPv3, which allows SNMP managers to read information about the system for device monitoring.

Procedure

To configure the SNMP agent, use the **system snmp** commands.

Example: SNMPv3 access group configuration

In the following example, the SNMPv3 agent uses an access group for get requests. The SNMPv3 user authentication and privacy protocols are configured using the **security-entry** parameter. The value of **password** cannot contain spaces.

```
--{ * candidate shared default }--[  ]--
# info system snmp
    system {
        snmp {
            access-group ag1 {
                admin-state enable
                security-level auth-priv
                security-entry se1 {
                    user user1 {
                    authentication {
                        protocol hmac-md5-96
                        password $aes1$AW8qEdNV+4KmIm8=$F2zgIDAO4DkcFh+6oLyd2w==
                    }
                    privacy {
                        protocol cbc-des
                        password $aes1$AW+ZudVoGPQP5W8=$1UMEKehkoPqo8zGFE0KzxA==
                    }
                }
            }
            network-instance default {
                admin-state enable
            }
        }
    }
```

Example: SNMPv3 trap group configuration

In the following example, the SNMPv3 agent uses a trap group within the **default** network instance. The SNMPv3 user authentication and privacy protocols are configured using the **security-entry** parameter. The value of **password** cannot contain spaces.

```
--{ * candidate shared default }--[  ]--
# info system snmp
    system {
        snmp {
            trap-group tg1 {
                admin-state enable
                network-instance default
                destination destination1 {
                    admin-state enable
                    address 10.2.2.2
                    security-level auth-priv
                    security-entry se1 {
                        user user1 {
                        authentication {
                            protocol hmac-md5-96
                            password $aes1$AW/ZYq/e/AbNS28=$Dw3ipXdBawX9P10lfe/zAw==
                        }
                        privacy {
                            protocol cbc-des
                            password $aes1$AW9Bvh9EbJcvwm8=$V214DJcpHCvdxkWbewDVuQ==
                        }
                    }
                }
            }
            network-instance default {
                admin-state enable
            }
        }
    }
```

11 General and operational commands

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

Table 19: 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.
echo	Echoes text back to a session

Command	Description
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) • home (use `~/.srlinuxrc` configuration file)

Command	Description
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)
file cat	The **file** commands allow you to interact with files and directories on the system. These commands are passed through to the Linux binaries that perform file actions, for example **cat** and **cp**. The **file** commands are supported in candidate, running, and state modes. **file cat** displays files contents Usage: **cat** *<path (1-255 times)>* **[-v] [-T] [-t] [-s] [-n] [-E] [-e] [-A] [--version]** • *path* - Specifies the path to a file or directory • **-v** - Verbose, explains what is being done • **-T** - Shows tabs, displaying TAB characters as `^I` • **-t** - Equivalent to **-vT** • **-s** - Suppresses repeated empty output lines • **-n** - Numbers all output lines

Command	Description
	• **-E** - Displays $ at the end of each line • **-e** - Equivalent to **-vE** • **-A** - Shows all, equivalent to **-vET** • **--version** - Outputs version information and exits
file cd	Changes working directory Usage: **cd [**<*path*>**]** • *path* - Specifies the path to a directory
file cp	Copies files and directories Usage: **cp** <*path (2 times)*> **[-a] [-f] [-i] [-l] [-n] [-r] [-p] [-R] [-s] [-u] [--version] [-v]** • *path* - Specifies the path to a file or directory • **-a** - Specifies archive mode, preserving file ownership, permissions, and timestamps • **-f** - If an existing destination file cannot be opened, removes it and tries again (this option is ignored when the **-n** option is also used) • **-i** - Interactively prompts before overwrite (overrides a previous **-n** option) • **-l** - Hard links files instead of copying • **-n** - Does not overwrite an existing file (overrides a previous **-i** option) • **-r** - Copies directories recursively • **-p** - Preserves mode, ownership, and timestamps • **-R** - Copies directories recursively (alias of **-r**) • **-s** - Makes symbolic links instead of copying • **-u** - Copies only when the source file is newer than the destination or when the destination is missing • **--version** - Outputs version information and exits • **-v** - Verbose, explains what is being done
file ls	Lists directory contents Usage: **ls [**<*path (0-255 times)*>**] [-a] [-A] [-C] [--color <always|auto|never>] [-d] [-F] [-h] [-i] [-l] [-r] [-R] [-s] [-S] [--time-style <long-iso|full-iso|iso|locale>] [-t] [-u] [-U] [-v] [-x] [-X] [-1] [--version]** • *path* - Specifies the path to a file or directory • **-a** - Lists all, including entries starting with . • **-A** - List almost all, excluding implied . and ..

Command	Description
	• **-C** - Lists entries by columns • **--color** - Colorizes the output • **-d** - Lists directories themselves, not their contents • **-F** - Classifies, appending indicators (one of */=>@|) to entries • **-h** - Human readable, with **-l** or **-s**, prints human readable sizes (for example, 1K, 234M, 2G) • **-i** - Inode, prints the index number of each file • **-l** - Uses a long listing format • **-r** - Reverses order while sorting • **-R** - Lists subdirectories recursively • **-s** - Prints the allocated size of each file, in blocks • **-S** - Sorts by file size, largest first • **--time-style** - With **-l**, shows times using a specific style • **-t** - Sorts by modification time, newest first • **-u** - With **-lt**, sorts by, and shows access time; with **-l** shows access time and sorts by name; otherwise, sorts by access time, newest first • **-U** - Does not sort; lists entries in directory order • **-v** - Verbose, explains what is being done • **-x** - Lists entries by lines instead of by columns • **-X** - Sorts alphabetically by entry extension • **-1** - Lists one file per line • **--version** - Outputs version information and exits
file md5sum	Calculates and verifies checksums. Usage: **md5sum [**<*path (0-255 times)*>**] [-c] [--ignore-missing] [--quiet] [--status] [--version]** • **-c** - Reads checksums from the files and checks them • **--ignore-missing** - Does not fail or report the status for missing files • **--quiet** - Does not print OK for each successfully verified file • **--status** - Does not output anything, status code shows success • **--version** - Outputs version information and exits
file mkdir	Creates directories. Usage: **mkdir** <*path (1-255 times)*> **[-m** <*value*>**] [-p] [--version] [-v]**

Command	Description
	• *path* - Specifies the path to a file or directory • **-m** - Sets the file mode (as in chmod), not a=rwx - umask • **-p** - No error if existing, makes parent directories as needed • **--version** - Outputs version information and exits • **-v** - Verbose, explains what is being done
file mv	Moves or renames files and directories Usage: **mv** *<path (2 times)>* **[-f] [-i] [-n] [-u] [--version] [-v]** • *path* - Specifies the path to a file or directory • **-f** - Force, does not prompt before overwriting • **-i** - Interactive, prompts before overwrite • **-n** - No clobber, does not overwrite an existing file if you specify more than one of **-i**, **-f**, **-n**, only the final one takes effect • **-u** - Update, moves only when the source is newer than the destination or when the destination is missing • **--version** - Outputs version information and exits • **-v** - Verbose, explains what is being done
file pwd	Prints the working directory Usage: **pwd [**<*path*>**] [--version]** • *path* - Specifies the path to a file or directory • **--version** - Outputs version information and exits
file rm	Removes files or directories Usage: **rm** *<path (1-255 times)>* **[-f] [-i] [-I] [-r] [-R] [-d] [--version] [-v]** • *path* - Specifies the path to a file or directory • **-f** - Force, ignores nonexistent files and arguments, never prompts before removal • **-i** - Interactive, prompts before every removal • **-I** - Interactive, prompts once before removing more than three files, or when removing recursively; less intrusive than **-i**, while still giving protection against most mistakes • **-r** - Removes directories and their contents recursively • **-R** - Removes directories and their contents recursively (alias of **-r**) • **-d** - Directories, removes empty directories • **--version** - Outputs version information and exits

Command	Description
	• **-v** - Verbose, explains what is being done
file touch	Changes file timestamps Usage: **touch** <*path (1-255 times)*> **[-a] [-c] [-d** <*value*>**] [-m] [-r** <*value*>**] [-t** <*value*> **] [--version]** **-a** - Changes only the access time **-c** - Does not create any files **-d** - Parses this string and uses it instead of the current time **-m** - Changes only the modification time **-r** - Uses this file's times instead of the current time **-t** - Uses [[CC]YY]MMDDhhmm[.ss] instead of the current time **--version** - Outputs version information and exit
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
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.

Command	Description
	Usage: **ping** <*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 • **-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; see Pre-defined show reports.

Command	Description
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
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.

Command	Description
	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
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).

insert	Inserts the provided value into the specified user-ordered list.
	
	Usage: **insert <***path-to-list***> <***value***> <first|before|after|last>< ***value***>**
	
	The provided value can be a single entry or a list. Use square brackets when specifying a list. In these examples, the provided values get added to an example `community-set` list named *sample*.
	
	`insert community-set sample member [ 1:1 1:2 ]`
	
	`insert community-set sample member 1:1`
	
	• **first** - Insert a value at the beginning of the user-ordered list
	• **before** - Insert a value in the user-ordered list before another specified value

Command	Description
	• **after** - Insert a value in the user-ordered list after another specified value • **last** - Insert a value at the end of the user-ordered list The optional **after** or **before** parameters, or the **first** and the **last** parameters, are mutually exclusive to place the value or values at an exact location in the list. **Note:** These parameters cannot be used for a system-ordered list.

12 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*.

12.1 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>
        — mac-filter <filter name>
            — entry <value>
            — subinterface <subinterface value>
        — summary
```

12.1.1 ACL descriptions

Context	Description
show acl	Container for ACL show reports.
show acl capture-filter *filter type*	Show a list of ACL capture filters. Specify either ipv4-filter or ipv6-filter.
show acl capture-filter ipv4-filter	Show a list of ACL capture IPv4 filters.
show acl capture-filter ipv4-filter entry *number*	Show a specific ACL capture IPv4 filter.
show acl capture-filter ipv6-filter	Show a list of ACL capture IPv6 filters.

Context	Description
show acl capture-filter ipv6-filter entry *number*	Show a specific ACL capture IPv6 filter.
show acl cpm-filter *filter type*	Show a list of ACL CPM filters. Specify either ipv4-filter or ipv6-filter.
show acl cpm-filter ipv4-filter	Show a list of ACL CPM IPv4 filters.
show acl cpm-filter ipv4-filter entry *number*	Show a specific ACL CPM IPv4 filter.
show acl cpm-filter ipv6-filter	Show a list of ACL CPM IPv6 filters.
show acl cpm-filter ipv6-filter entry *number*	Show a specific ACL CPM IPv6 filter.
show acl ipv4-filter *name*	Show ACL IPv4 filter policies.
show acl ipv4-filter *name* **entry** *number*	Show ACL IPv4 filter policies by entry ID.
show acl ipv4-filter *name* **subinterface** *string*	Show ACL IPv4 filter policies by subinterface.
show acl ipv6-filter *name*	Show ACL IPv6 filter policies.
show acl ipv6-filter *name* **entry** *number*	Show ACL IPv6 filter policies by entry ID.
show acl ipv6-filter *name* **subinterface** *string*	Show ACL IPv6 filter policies by subinterface.
show acl mac-filter *name*	Show ACL MAC filter policies.
show acl mac-filter *name* **entry** *number*	Show ACL MAC filter policies by entry ID.
show acl mac-filter *name* **subinterface** *string*	Show ACL MAC filter policies by subinterface.
show acl summary	Show a summarized list of all ACL filters.

12.2 ARPND show reports

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

12.2.1 ARPND descriptions

Context	Description
show arpnd	Container for ARPND show reports.

Context	Description
show arpnd neighbors	Show all ARPND neighbors including associated interfaces, subinterfaces, origin, and state.
show arpnd neighbors interface *interface name*	Show ARPND neighbor for a specified interface name.
show arpnd arp-entries	Show all ARPND entries including associated interfaces, subinterfaces, origin, and expiry.
show arpnd arp-entries interface *interface name*	Show ARPND entries for a specified interface name.

12.3 Interface show reports

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

12.3.1 Interface descriptions

Context	Description
show interface *interface port name*	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.
show interface *interface port name* **brief**	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.
show interface *interface port name* **all**	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.
show interface *interface port name* **detail**	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.

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

12.4 LAG show reports

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

12.4.1 LAG descriptions

Context	Description
show lag *lag instance*	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.
show lag *lag instance* **brief**	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.
show lag *lag instance* **detail**	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.
show lag *lag instance* **lacp-state**	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.).
show lag *lag instance* **lacp-statistics**	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.
show lag *lag instance* **member-statistics**	Show a LAG report that displays all member-statistics.
show lag *lag instance* **queue-detail**	Show a LAG report that displays aggregated queue statistics for all LAG link members.

12.5 MPLS show reports

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

12.5.1 MPLS descriptions

Context	Description
show mpls-aft	Container for MPLS reports. Can be used to show all MPLS abstract forwarding tables.
show mpls-aft network-instance *network-instance name*	Show an MPLS abstract forwarding table for a specified network instance.

12.6 Network-instance show reports

```
show
    — network-instance <name>
        — bridge-table
            — mac-duplication duplication-entries
            — mac-table
                — all
                — mac <address>
                — summary
            — proxy-arp
                — all
                — ip-duplication duplicate-entries
                — neighbor <address>
                — summary
            —proxy-nd
                — all
                — ip-duplication duplicate-entries
                — neighbor <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
                    – ipv4-labeled-unicast
                        – prefix <IP prefix>
                        – summary
                    – ipv6-labeled-unicast
                        – prefix <IP prefix>
                        – summary
                    – l3vpn-ipv4-unicast
                        – summary
                    – l3vpn-ipv6-unicast
                        – 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
            – ldp
                – interface
                – ipv4
                    – address
                        – advertised
                        – all
                        – received
                    – fec
                        – advertised
                        – all
                        – received
                    – nexthop
                – neighbor
                – session
                    – detail
                    – statistics
                – statistics
                – summary
            – ospf
                – area
                    – detail
                – database
                    – type
                – instance <name>
                – interface
                    – detail
                – neighbor
                    – detail
                – statistics
```

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

12.6.1 Network-instance descriptions

Context	Description
show network-instance *name*	Show network-instance for specified name or additional command (bridge-table, interfaces, protocols, route-table, or summary)
show network-instance *name* **bridge-table**	Show MAC bridge tables. Specify mac-duplication duplication-entries or mac-table.
show network-instance *name* **bridge-table mac-duplication duplication-entries**	Show all MAC learned entries.
show network-instance *name* **bridge-table mac-table**	Show a MAC table report. Specify option of all, mac, or summary.
show network-instance *name* **bridge-table mac-table all**	Show all MAC table entries
show network-instance *name* **bridge-table mac-table mac** *address*	Show MAC table entry for specific address.
show network-instance *name* **bridge-table mac-table summary**	Show a MAC table summary report.
show network-instance *name* **bridge-table proxy-arp**	Show proxy ARP entries report.
show network-instance *name* **bridge-table proxy-arp all**	Show all proxy ARP table entries.
show network-instance *name* **bridge-table proxy-arp ip-duplication duplicate-entries**	Show proxy ARP IP duplication report.
show network-instance *name* **bridge-table proxy-arp neighbor** *<address>*	Show proxy ARP IP table entry.

Context	Description
show network-instance *name* **bridge-table proxy-arp summary**	Show proxy ARP IP table summary.
show network-instance *name* **bridge-table proxy-nd**	Show proxy ND entries report.
show network-instance *name* **bridge-table proxy-nd all**	Show all proxy ND table entries.
show network-instance *name* **bridge-table proxy-nd ip-duplication duplicate-entries**	Show proxy ND IP duplication report.
show network-instance *name* **bridge-table proxy-nd neighbor** *<address>*	Show proxy ND table entry.
show network-instance *name* **bridge-table proxy-nd summary**	Show proxy ND table summary.
show network-instance *name* **interfaces** *interface name*	Show all network-instance interfaces.
show network-instance *name* **protocols**	Show network-instance protocol details. Specify BGP, IS-IS, or OSPF.
show network-instance *name* **protocols bgp**	Show BGP status report. Specify neighbor, routes, or summary.
show network-instance *name* **protocols bgp neighbor** *IP address*	Show a BGP neighbor report. Specify a peer address to see a report for the specified address.
show network-instance *name* **protocols bgp neighbor** *IP address* **advertised-routes**	Show a BGP neighbor advertised routes report. The network-instance name and neighbor peer IP address must be specified.
show network-instance *name* **protocols bgp neighbor** *IP address* **advertised-routes evpn**	Show a BGP neighbor advertised routes report for EVPN. The network-instance name and neighbor peer IP address must be specified.
show network-instance *name* **protocols bgp neighbor** *IP address* **advertised-routes ipv4**	Show a BGP neighbor advertised routes report for IPv4. The network-instance name and neighbor peer IP address must be specified.
show network-instance *name* **protocols bgp neighbor** *IP address* **advertised-routes ipv6**	Show a BGP neighbor advertised routes report for IPv6. The network-instance name and neighbor peer IP address must be specified.
show network-instance *name* **protocols bgp neighbor** *IP address* **detail**	Show a BGP neighbor detailed report. network-instance name and neighbor peer IP address must be specified.
show network-instance *name* **protocols bgp neighbor** *IP address* **maintenance**	Show a BGP neighbor report that shows the maintenance mode and group.
show network-instance *name* **protocols bgp neighbor** *IP address* **received-routes**	Show a BGP neighbor received routes report. network-instance name and neighbor peer IP address must be specified.

Context	Description
show network-instance *name* **protocols bgp neighbor** *IP address* **received-routes evpn**	Show a BGP neighbor received routes report for EVPN. The network-instance name and neighbor peer IP address must be specified.
show network-instance *name* **protocols bgp neighbor** *IP address* **received-routes ipv4**	Show a BGP neighbor received routes report for IPv4. The network-instance name and neighbor peer IP address must be specified.
show network-instance *name* **protocols bgp neighbor** *IP address* **received-routes ipv6**	Show a BGP neighbor received routes report for IPv6. The network-instance name and neighbor peer IP address must be specified.
show network-instance *name* **protocols bgp routes** *IP family*	Show BGP route report for an IP family (EVPN, IPv4 or IPv6).
show network-instance *name* **protocols bgp routes evpn**	Show a BGP EVPN route report. Additional parameters can define a report for a specific route type, and a detailed verses summary report.
show network-instance *name* **protocols bgp routes evpn route-type** *route type*	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.
show network-instance *name* **protocols bgp routes evpn route-type** *route type* **detail**	Show a detailed BGP EVPN route report for a specific route type.
show network-instance *name* **protocols bgp routes evpn route-type** *route type* **summary**	Show a summarized BGP EVPN route report for a specific route type.
show network-instance *name* **protocols bgp routes ipv4**	Show a BGP IPv4 route report. Specify an IP prefix or summary.
show network-instance *name* **protocols bgp routes ipv4 prefix** *IP prefix*	Show a BGP IPv4 route report for a specified IP prefix.
show network-instance *name* **protocols bgp routes ipv4 summary**	Show a BGP IPv4 route summary report.
show network-instance *name* **protocols bgp routes ipv6**	Show a BGP IPv6 route report. Specify an IP prefix or summary.
show network-instance *name* **protocols bgp routes ipv6 prefix** *IP prefix*	Show a BGP IPv6 route report for a specified IP prefix.
show network-instance *name* **protocols bgp routes ipv6 summary**	Show a BGP IPv6 route summary report.
show network-instance *name* **protocols bgp routes ipv4-labeled-unicast**	Show a BGP IPv4 labeled unicast route report. Specify an IP prefix or summary.
show network-instance *name* **protocols bgp routes ipv4-labeled-unicast prefix** *IP prefix*	Show a BGP IPv4 labeled unicast route report for a specified IP prefix.

Context	Description
show network-instance *name* **protocols bgp routes ipv4-labled-unicast summary**	Show a BGP IPv4 labeled unicast route summary report.
show network-instance *name* **protocols bgp routes ipv6-labeled-unicast**	Show a BGP IPv6 labeled unicast route report. Specify an IP prefix or summary.
show network-instance *name* **protocols bgp routes ipv6-labeled-unicast prefix** *IP prefix*	Show a BGP IPv6 labeled unicast route report for a specified IP prefix.
show network-instance *name* **protocols bgp routes ipv6-labeled-unicast summary**	Show a BGP IPv6 labeled unicast route summary report.
show network-instance *name* **protocols bgp routes l3vpn-ipv4-unicast**	Show a BGP IPv4 unicast routes report for Layer 3 VPN.
show network-instance *name* **protocols bgp routes l3vpn-ipv4-unicast summary**	Show a summary report of BGP IPv4 unicast routes for Layer 3 VPN.
show network-instance *name* **protocols bgp routes l3vpn-ipv6-unicast**	Show a BGP IPv6 unicast routes report for Layer 3 VPN.
show network-instance *name* **protocols bgp routes l3vpn-ipv4-unicast summary**	Show a summary report of BGP IPv6 unicast routes for Layer 3 VPN.
show network-instance *name* **protocols bgp summary**	Show a BGP summary report.
show network-instance *name* **protocols bgp-evpn**	Show a BGP-EVPN status report.
show network-instance *name* **protocols bgp-evpn bgp-instance** *instance*	Show a BGP-EVPN status report for a specific bgp instance.
show network-instance *name* **protocols bgp-vpn**	Show a BGP-VPN status report.
show network-instance *name* **protocols bgp-vpn bgp-instance** *instance*	Show a BGP-VPN status report for a specific bgp instance.
show network-instance *name* **protocols isis**	Show IS-IS status report. Specify adjacency, database, hostnames, interface, or summary.
show network-instance *name* **protocols isis adjacency** *interface_name*	Show a list of IS-IS adjacencies formed through this interface.
show network-instance *name* **protocols isis adjacency** *interface_name* **neighbor-system-id** *ID*	Show a list of IS-IS adjacencies for a specified system ID of a neighbor router.
show network-instance *name* **protocols isis adjacency** *interface_name* **neighbor-system-id** *ID* **adjacency-level** *value*	Show a list of IS-IS adjacencies for a specified adjacency level.
show network-instance *name* **protocols isis adjacency** *interface_name* **neighbor-system-id** *ID* **adjacency-level** *value* **detail**	Show a detailed IS-IS adjacency report.

Context	Description
show network-instance *name* **protocols isis database**	Show an IS-IS database report.
show network-instance *name* **protocols isis database lsp-id** *ID*	Show an IS-IS 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).
show network-instance *name* **protocols isis database detail**	Show a detailed IS-IS database report.
show network-instance *name* **protocols isis interface** *name*	Show an IS-IS interface report for all interfaces or optionally specify an interface name.
show network-instance *name* **protocols isis interface** *name* **detail**	Show a detailed IS-IS interface report for all interfaces or optionally specify an interface name.
show network-instance *name* **protocols isis hostnames**	Show an IS-IS hostname report.
show network-instance *name* **protocols isis hostnames detail**	Show a detailed IS-IS hostname report.
show network-instance *name* **protocols isis summary**	Show a summarized IS-IS report.
show network-instance *name* **protocols ldp**	Container for LDP show reports.
show network-instance *name* **protocols ldp interface** *name* **[detail]**	Show the LDP report for an interface.
show network-instance *name* **protocols ldp ipv4**	Container for IPv4 LDP reports.
show network-instance *name* **protocols ldp ipv4 address**	Show the LDP report for IPv4 addresses.
show network-instance *name* **protocols ldp ipv4 address advertised [table|summary] [lsr-id** *value***] [label-space-id***value***]**	Show the LDP report for advertised IPv4 addresses.
show network-instance *name* **protocols ldp ipv4 address all [table|summary] [lsr-id** *value***] [label-space-id***value***]**	Show the LDP report for all IPv4 addresses.
show network-instance *name* **protocols ldp ipv4 address received [table|summary] [lsr-id** *value***] [label-space-id***value***]**	Show the LDP report for received IPv4 addresses.
show network-instance *name* **protocols ldp ipv4 fec**	Show the LDP report for IPv4 FECs.
show network-instance *name* **protocols ldp ipv4 fec advertised [table|summary|**	Show the LDP report for advertised IPv4 FECs.

Context	Description
detail] [prefix *value* **[lsr-id** *value*] **[label-space-id***value***]**	
show network-instance *name* **protocols ldp ipv4 fec all [table|summary|detail] [prefix** *value* **[lsr-id** *value*] **[label-space-id***value***]**	Show the LDP report for all IPv4 FECs.
show network-instance *name* **protocols ldp ipv4 fec received [table|summary|detail] [prefix** *value* **[lsr-id** *value*] **[label-space-id***value***]**	Show the LDP report for received IPv4 FECs.
show network-instance *name* **protocols ldp ipv4 nexthop [fec-prefix** *value* **[lsr-id** *value*] **[label-space-id***value***]**	Show the LDP report for IPv4 next-hops.
show network-instance *name* **protocols ldp neighbor**	Show the LDP neighbor report.
show network-instance *name* **protocols ldp session [lsr-id** *value*] **[label-space-id** *value*]	Show the LDP session report.
show network-instance *name* **protocols ldp session [lsr-id** *value*] **[label-space-id** *value*] **detail**	Show the detailed LDP session report.
show network-instance *name* **protocols ldp session [lsr-id** *value*] **[label-space-id** *value*] **statistics**	Show the LDP session statistics report.
show network-instance *name* **protocols ldp statistics**	Show the LDP statistics report.
show network-instance *name* **protocols ldp summary**	Show the LDP summary report.
show network-instance *name* **protocols ospf**	Show OSPF status report. Specify area, interface, neighbor, or status.
show network-instance *name* **protocols ospf area**	Show OSPF report for areas where the local system exists.
show network-instance *name* **protocols ospf area detail**	Show detailed OSPF report for areas where the local system exists.
show network-instance *name* **protocols ospf database**	Show OSPF database report.
show network-instance *name* **protocols ospf database type**	Show OSPF database report for a specific database type. For example: router-lsa
show network-instance *name* **protocols ospf instance** *instance name*	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.

Context	Description
show network-instance *name* **protocols ospf interface**	Show OSPF report for OSPF interfaces that act as a connection between a router and one of its attached networks.
show network-instance *name* **protocols ospf interface detail**	Show detailed OSPF report for OSPF interfaces that act as a connection between a router and one of its attached networks.
show network-instance *name* **protocols ospf neighbor**	Show OSPF report for OSPF neighbors.
show network-instance *name* **protocols ospf neighbor detail**	Show detailed OSPF report for OSPF neighbors.
show network-instance *name* **protocols ospf statistics**	Show OSPF Rx, Tx, and packet statistics report.
show network-instance *name* **protocols ospf status**	Show an overall status report for OSPF (router IDs, version, admin state, backbone and area border router settings, and so on.).
show network-instance *name* **route-table**	Show a network-instance route table report. Specify all, ipv4-unicast, ipv6-unicast, next-hop, or, summary.
show network-instance *name* **route-table all**	Show a report of all routes that includes prefix, ID, active status and next-hop details.
show network-instance *name* **route-table summary**	Show a summary report of active routes by name and protocol.
show network-instance *name* **route-table ipv4-unicast**	Show a IPv4 route table report. Specify a summary report or by prefix ID.
show network-instance *name* **route-table ipv4-unicast prefix** < *ID* > **detail**	Show a IPv4 route table report for a specified prefix. Report includes destination, ID, owner, active status and details when it was last changed.
show network-instance *name* **route-table ipv4-unicast summary**	Show a IPv4 route table summary report that includes prefix, IDs, active status, and next-hop details.
show network-instance *name* **route-table ipv6-unicast**	Show a IPv6 route table report. Specify a summary report or by prefix ID.
show network-instance *name* **route-table ipv6-unicast prefix** < *ID* > **detail**	Show a IPv6 route table report for a specified prefix. Report includes destination, ID, owner, active status and details when it was last changed.
show network-instance *name* **route-table ipv6-unicast summary**	Show a IPv6 route table summary report that includes prefix, IDs, active status, and next-hop details.
show network-instance *name* **route-table mpls**	Show MPLS table report that includes labels, next-hop IP, next-hop subinterface, and next-hop labels.
show network-instance *name* **route-table next-hop** *index*	Show a report of all next-hop route tables or optionally provide an index number to show a report for a specified route table.
show network-instance *name* **summary**	Show a summary report for all network-instances, or specify a network-name to refine the report.

Context	Description
show network-instance *name* **static-mpls**	Show static MPLS entries.
show network-instance *name* **tunnel-table**	Context for all tunnels in a tunnel table. Use with options all, IPv4, or IPv6.
show network-instance *name* **tunnel-table all**	Shows a network instance tunnel report that includes the prefix, type, ID, metric, preference, and last updated fields.
show network-instance *name* **tunnel-table ipv4 ip address type**	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).
show network-instance *name* **tunnel-table ipv6 ip address type**	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).
show network-instance *name* **vxlan-inteface** *name*	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 "*").

12.7 Platform show reports

```
show
    — platform
        — chassis
        — control <slot ID>
        — environment
        — fabric <slot number>
        — fan-tray <ID number>
        — linecard <slot number>
        — power-supply <ID number>
        — redundancy
        — resource-monitoring
        — trust
            — secure-boot
                — control
                    — detail
            — tpm
                — certificate
                — pcr-bank
```

12.7.1 Platform descriptions

Context	Description
show platform	Show a state report for all components.
show platform chassis	Show a report for the chassis, including type, MAC address, number of slots, and operating status.

Context	Description
show platform control *slot ID*	Show a report for the control module configuration and state. Shows all modules unless a specific slot is provided.
show platform environment	Show a report for the platform environment (state and temperature) for all components.
show platform fabric *slot ID*	Show a report for the fabric module configuration and state. Shows all modules unless a specific slot is provided.
show platform fan-tray *slot ID*	Show a report for the fan tray configuration and state. Shows all modules unless a specific slot is provided.
show platform linecard *slot ID*	Show a report for the linecard configuration and state. Shows all modules unless a specific slot is provided.
show platform power-supply *ID*	Show a report for the power-supply configuration and state. Shows all modules unless a specific ID is provided.
show platform redundancy	Show a platform redundancy report.
show platform resource-monitoring	Show a resource-monitoring report for areas such as ACL, TCAM, IP MPLS forwarding, etc.
show platform trust	Show platform trust information.
show platform trust secure-boot	Show platform secure boot information.
show platform trust secure-boot control *id*	Show secure boot information for control modules.
show platform trust secure-boot control *id* **detail**	Show detailed secure boot information for control modules.
show platform trust tpm *tpm-id*	Show TPM information.
show platform trust tpm *tpm-id* **certificate** *certificate*	Show certificates from TPM NV Memory.
show platform trust tpm *tpm-id* **pcr-bank** *pcr-bank*	Show TPM PCR allocation.

12.8 System show reports

```
show
    — 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
```

12.8.1 System descriptions

Context	Description
show system	Container for system management show reports.
show system aaa authentication session *session ID*	Show a list of all active sessions in the system or for a specific session ID user if an ID is specified.
show system application *application name*	Show a list of all applications and their status in the system or for a specific application if a name is specified.
show system lldp neighbor	Show a report of all LLDP neighbors that includes system name, chassis ID, first message, last update, port and related BGP details.
show system lldp neighbor interface *interface name*	Show a report of a specific LLDP neighbors by specifying an interface name.
show system logging	Shows a list of system logs, type, and directory path.
show system logging buffer	Shows a report of type buffer log files maintained in memory (non-persistent across system reboots).
show system logging buffer messages	Show a report of the messages associated with the system logging buffer (/var/log/srlinux/buffer/ messages)
show system logging buffer system	Show a report of the system related information in the system logging buffer (/var/log/srlinux/buffer/system)
show system logging file	Shows a report of log files that have been directed to a specific file (/var/log/srlinux/file).
show system logging file messages	Show a report of the messages associated with logs in a specific file (/var/log/srlinux/file/messages)
show system network-instance ethernet-segments name	Show a report that defines ethernet-segments details such as ESI, Alg, peers, and associated network instances.
show system sflow status	Show an sflow status report that includes the state, and sample size and rate.

12.9 Tunnel show reports

```
show
    — tunnel
        — vxlan-tunnel
            — all
            — vtep <ip address>
```

12.9.1 Tunnel descriptions

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

12.10 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
```

12.10.1 Tunnel-interface descriptions

Context	Description
show tunnel-interface *interface*	Enter the context for tunnel interface (reports for EVPN Layer-2). Specify a tunnel-interface name or an asterisk (*) for all interfaces.
show tunnel-interface *interface* **vxlan-interface** *interface*	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.
show tunnel-interface *interface* **vxlan-interface** *interface* **bridge-table**	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.
show tunnel-interface *interface* **vxlan-interface** *interface* **bridge-table mac-table**	Show a mac-table vxlan interface report for EVPN-VXLAN for layer-2. Report includes the address, destination, VNI, index, type, and last updated.

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

12.11 Version show reports

```
show
    — version
```

12.11.1 Version descriptions

Context	Description
show version	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