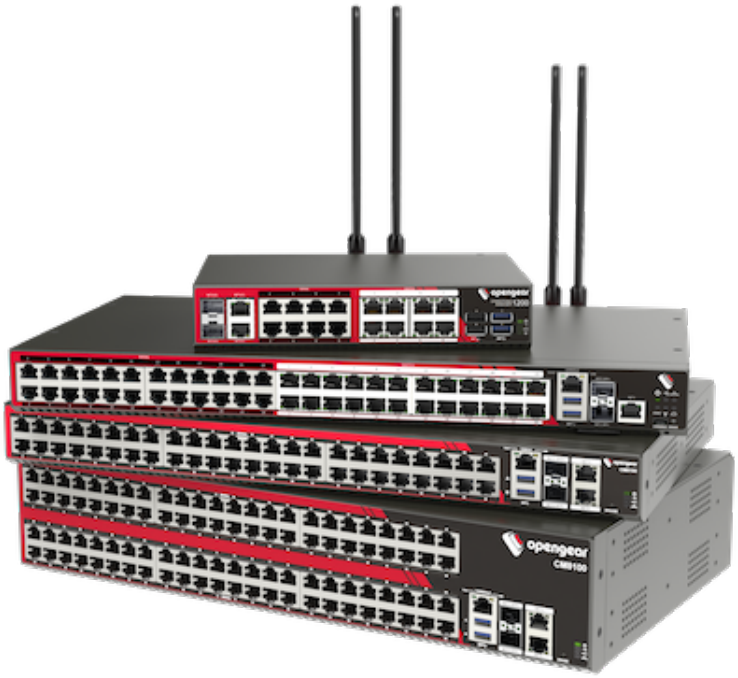
Opengear Config CLI
23.10



COPYRIGHT ©

Information in this document is subject to change without notice and does not represent a commitment on the part of Opengear. Opengear provides this document “as is,” without warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose.

Opengear may make improvements and/or changes in this manual or in the product (s) and/or the program(s) described in this manual at any time. This product could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes may be incorporated in new editions of the publication.

DOCUMENT REVISION HISTORY

Document Version Number	Revision Date	Description
22.11.0	November 2022	WIP for the first release of the Config CLI User Guide.
22.11.0	March 24 2023	Pre-release evaluation copy.
23.10.0	October 2023	Added Configuring a Single Session on a Port. Support for WireGuard

ABOUT THIS USER GUIDE

This user guide is up to date for the 23.10.0 release. When using a minor release there may or may not be a specific version of the user guide for that release.

CONFIG CLI

The Config Command Line Interface(CLI) provides users with an interactive and familiar environment similar to other networking devices that users may be familiar with. The result is a user-experience that feels like an Interactive CLI .

Advantages of the Config CLI are:

- Interactive CLI makes everyday operations such as configuration changes and troubleshooting activities easier for users.
- Items can be created or updated without being applied immediately.
- Items that are not applied are indicated by an asterisk (*) beside them when viewing information..
- Tab complete is supported for many commands.
- Built-in context sensitive help.
- Has a structured, tabular view when displaying lists of data.

CONFIG CLI GLOSSARY

Some new concepts are introduced in the initial release of Config CLI, these are described in this brief glossary of Config CLI terms used to define compound command lines.

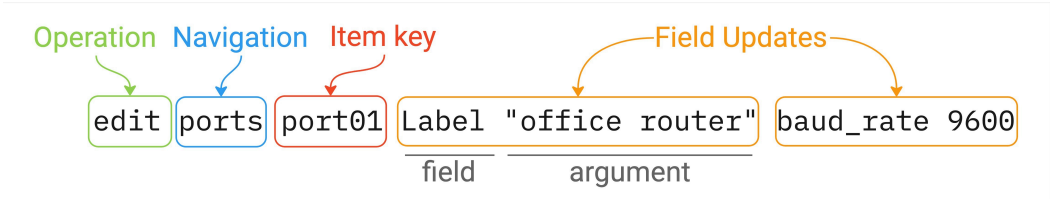
Context	The point in the configuration where the shell is active or focused.
Command Line	The whole command string that the user typed. Sometimes, the command line may directly translate to a command but sometimes may translate to multiple commands.
Command	The action that is to be executed in the current context.
Argument	The tokens that are passed to a command. These could be the values being assigned to fields in field updates but may also include array indexes or object keys.
Token	The words that make up a command line. These words can be space separated single words or quoted strings like "two words" or mult-iline strings.
Command Line Category	The overall description of the command line. There are four main types: 1. **simple:** simple command lines are compose of single command and with or without an argument, # Some examples: • config: user

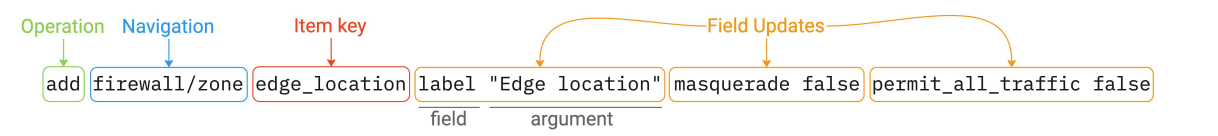
	• config: show • config: top
	2. **field update**: A command line composed of a field-update expression only.
	3. **config(user root):** `description "my description" enabled true` `navigation`: A command line made of a navigation expression only, meaning `config: user root`
	4. **compound:** A command line that starts with an OPERATION and is followed by an optional navigation expression then a field update expression. An operation is anyone of add edit show delete. Some examples are: • `config: add user bob description "my description" enabled true` • `config: show user bob` • `config: edit user bob description "new description"` • `config: delete user bob`

Complex field	A field that can be navigated into. These are either array or subobject type. See the Compound Command Line Structure illustration on the following page.

COMPOUND COMMAND LINE STRUCTURE

The following illustrations assist in visualizing the compound command line concept:





NAVIGATION IN CONFIG CLI

STARTING A SESSION IN CONFIG CLI

Start the config shell by typing `config` at a bash prompt. The bash prompt is presented to root and admin users when they log in via SSH or on the management or local console.

EXITING A CONFIG CLI SESSION

You can exit the Interactive CLI by in any of the following ways:

- Type `exit` to end the session.
- Send an EOF (Control+D).
- Send an INT (Control+C).

> **Note:**The session is prevented from exiting if there are un-committed changes, this condition is indicated by a message. However, you can force an exit by immediately executing an exit command again, any un-committed changes will be discarded.

NAVIGATING THE CONFIG CLI

The Config CLI operates using a hierarchy . Due to the variety of endpoints, there are several ways to get to a place where you may want to make changes.

- Starting at the root, enter endpoint names to descend down to lower endpoints.
- Similarly, type 'up' to ascend towards the root or type 'top' to reset to the root context.

Note: Every endpoint name is an operation that descends into that endpoint.

When using the config CLI, it is possible to navigate ‘downwards’ through multiple contexts with a single command line.

HIERARCHICAL IDENTIFIERS

This section outlines the identifiers needed to navigate the CLI.

Identifier	Description
Singleton endpoints	These require only the endpoint name to be uniquely identified.
List/item endpoints	The first level is the endpoint name, the second level is the item identifier (the identifier is the same identifier used by ogcli).
Multiple identifiers	A single endpoint (ssh/authorized_keys) requires an extra identifier. In this case, the hierarchy is: ssh/authorized_keys > userid > [key_id]
Nested fields	The interactive CLI treats nested fields as additional hierarchy levels. This applies both to arrays and maps. For arrays of complex values, each value shall also be a hierarchy level.

UNDERSTANDING FIELDS, ENTITIES AND CONTEXTS

The Config CLI allows you to configure the device settings through a number of required fields, which provide the settings for the device.

These fields are grouped in *entities* that describe a small set of functionality, for example, there is a ‘user’ entity which is used to access user settings. Entities can contain sub-entities as well as simple fields.

HOW CONTEXT OPERATES IN THE CONFIG CLI

Description

The *context* is the current entity that is the focus of the config shell. When the shell is first started, the context is a special parent context from which sub-entities can be seen. Within the config shell, a number of commands are available, depending on the current context.

When config shell is started the context is at the “top context” which lists all the entities when the show command is used. If the name of an entity is typed, then the context moves 'down' into that entity. When simple commands such as `show`, `help` or `apply` are used, they will act on the current context. The context can be moved down further by typing the name of an item.

Entities can contain sub-entities as well as simple fields. For example, there is a ‘user’ entity which is used to access user settings. Fields are grouped within entities that describe a small set of functionality.

Navigating Using Context

You select a context by typing the name of the target entity and pressing Enter/Return; the new context is shown in the prompt between brackets. In the following example, the ‘`user`’ context is accessed and then the ‘`john`’ sub-entity is accessed causing the context to become ‘user john’.

The ‘`show`’ command is used to list the entities and fields that descend from the current context.

```
config: user
config(user): show
Item names for entity user
  john matt myuser netgrp root
config(user): john
config(user john):
Entity user item john
  description
  enabled         true
  no_password     false
  password
  ssh_password_enabled true
  groups (array)
config(user john):
```

The following example will navigate the context to the root user object without first having to navigate to the user context:

```
config: user root
config(user root):
```

Sub-objects are supported. In the following example, power_supply_voltage_alert and syslog are nested sub-objects of the onitoring/alerts/power entity:

```
config: monitoring/alerts/power power_supply_voltage_alert syslog
config(monitoring/alerts/power power_supply_voltage_alert syslog):
```

GLOBAL & ENTITY-CONTEXT COMMANDS

GLOBAL CONTEXT COMMANDS

The table below lists commands available on any context:

Global Command	Description
`help (or '?')`	Show help which is context sensitive. It will list some special details about the current context, the list of sub entities (or fields) and a list of available commands.
`help <entity>`	Displays short-form help for the specific entity.
`show`	Lists the available entities and fields.
`<entity>`	Inputing the name of an entity changes the context to focus on the named entity.
`exit`	Exit the command shell.

ENTITY CONTEXT COMMANDS

In addition to the global context commands, once an entity context is selected then further, entity context, commands become available.

Entity Command	Description
`<field>`	Show the value of a field.
`help <entity>`	Displays short-form help for the specific entity.
`<field> <value>`	Set the value of a field.
`delete`	Deletes the current enitity. This is available when the context entity is an item in a list.
`add`	Append a sub-entity or field to the current entity. This is only available when the context entity is a list.

CONFIG CLI ENTITIES

The config shell allows the user to configure a number of fields which are the settings for the device. These fields are grouped in entities that describe a small set of functionality. For example, there is a ‘user’ entity which is used to access user settings. Entities can contain sub-entities as well as simple fields.

Once in the shell, a number of commands are available depending on the current context. The context is the current entity that is the focus of the config shell. When the shell is first started, the context is a special parent context from which sub-entities can be seen.

Once a context is selected by typing the name of the entity, it is shown in the prompt between brackets. e.g. In the following snippet, the ‘user’ context is accessed and then the ‘john’ sub-entity is accessed causing the context to become ‘user john’. The ‘show’ command is used to list the entities and fields that descend from the current context.

SUPPORTED ENTITIES

Entity	Definition
access_right	An access right is a permit that grants the holder access to a feature or collection of related features.
auth	Configure remote authentication, authorization, accounting (AAA) servers.
auto_response/beacon	Read and manipulate the Auto-Response beacons on the NetOps Console Server appliance.

auto_response/reaction	Read and manipulate the Auto-Response reactions on the NetOps Console Server appliance.
auto_response/status	Read the AutoResponse Status on the NetOps Console Server appliance.
auto_response/status/beacon-module	Read the AutoResponse Status of Beacon Modules on the NetOps Console Server appliance.
cellfw/info	Retrieve cellular modem version and related information.
cellmodem	Retrieve information about the cell modem.
cellmodem/sim	Cell modem SIM status.
conn	Read and manipulate the network connections on the NetOps Console Server appliance.
failover/settings	failover/settings endpoint is to check and update failover settings. When failover is enabled, this device will consume from 1MB to 1.6 MB of bandwidth per day on the probe_physif connection. If the probe addresses are unreachable, this device will take from 108 to 156 seconds to enter the failover state.
failover/status	failover/status endpoint is to check current failover status.
firewall/policy	A collection of policies defined for the NetOps Console Server appliance's firewall. A policy specifies which zones traffic is allowed to route between.
firewall/predefined_service	A collection of predefined services for the NetOps Con-

	sole Server appliance's firewall. A service is a named grouping of one or more TCP or UDP ports for a particular networking protocol. For example, the 'https' service refers to TCP port 443. This collection contains predefined services for common protocols and doesn't include the services added by the administrator.
firewall/service	A collection of custom services defined for the NetOps Console Server appliance's firewall. A service is a named grouping of one or more TCP or UDP ports for a particular networking protocol. For example, the 'https' service refers to TCP port 443. The appliance includes many predefined services for common protocols (see /firewall/predefined_services). This collection contains only custom services which have been defined by the administrator.
firewall/zone	Collection of zones defined for the NetOps Console Server appliance's firewall. A zone includes 1 or more interfaces.
group	Retrieve or update user group information
ip_passthrough	IP Passthrough endpoints are for retrieving / changing IP Passthrough settings.
ip_passthrough/status	The IP Passthrough status endpoint provides information about what part of the IP Passthrough connection process the device is currently at and information about the connected downstream device.
ipsec_tunnel	Read and manipulate the IPsec tunnels on the NetOps Console Server appliance.

lighthouse_enrollment	View and control enrollment to a lighthouse.
local_password_policy	Configure the password policy for local users. This includes expiry and complexity settings.
logs/portlog	None
logs/portlog_settings	Check and update port log settings.
managementport	Used for working with local management console information
monitor/brute_force_protection/ban	Used for monitoring addresses banned by Brute Force Protection.
monitor/lldp/chassis	Get the current status of the network discovery (LLDP/CDP) protocols on this device.
monitor/lldp/neighbor	Get the list of neighboring devices (peers) that have been discovered by the LLDP protocol.
monitor/static_routes/ status	Used for monitoring the status of static routes. Only IPv4 static routes are supported.
monitoring/ alerts/networking	Retrieve and configure Networking Alert Group settings.
monitoring/alerts/power	Retrieve and configure Power Alert Group settings.
monitoring/alerts/system	Retrieve and configure System Alert Group settings.

pdu	Configure, monitor and control PDUs connected to the device.
pdus/drivers	Read the PDU driver list.
physif	Read and manipulate the network physical interfaces on the NetOps Console Server appliance.
port	Configuring and viewing ports information
port_session	None
ports/ auto_discover/schedule	Manage Port Auto-Discovery Scheduling
ports/status_port	Provides information about the serial pin status and Tx & Rx counters for each of this device's serial ports
system/admin_info	Retrieve or change the Operations Manager appliance system's information (hostname, contact and location)
services/ brute_force_protection	Provides access to the Brute Force Protection configuration on the system. When this service is enabled, the system watches for multiple failed login attempts and temporarily bans the offending IP Address for the configured amount of time.
services/lldp	Provides access to the Network Discovery Protocols (LLDP/CDP) configuration.
services/ntp	Provides access to the NTP client configuration on the system.

services/routing	Retrieve and configure routing services on the NetOps Console Server appliance.
services/ snmp_alert_manager	SNMP Alert Managers are used to receive and log SNMP TRAP and INFORM messages sent by the NetOps Console Server. To receive SNMP alerts generated by the system at least one SNMP Alert Manager must be configured.
services/snmpd	Simple Network Management Protocol (SNMP) is an Internet Standard protocol for collecting and organizing information about managed devices on IP networks and for modifying that information to change device behaviour. This entity allows configuration of the SNMP service.
services/ssh	Configure the Secure Shell Protocol (SSH) service.
services/syslog_server	Provides access to the remote syslog server configuration.
services/tftp	Trivial File Transfer Protocol (TFTP) is a service that allows files to be transferred to or from the NetOps Console Server appliance. This entity provides access to the TFTP server configuration on the system.
single_session	Can be enabled on a given port to prevent multiple users from connecting to that port or limit the port to a single concurrent connection.
ssh/authorized_key	Configure the SSH authorized keys for a specific user.
static_route	Configuring and viewing static routes.
system/admin_info	Retrieve or change the NetOps Console Server appli-

	ance system's information (hostname, contact and location).
system/banner	Retrieve or change the Operations Manager appliance system's banner text
system/cell_reliability_test	None
system/cellular_logging	Cellular logging provides the ability to capture the RRC connection messages from the EM7565 cellular module. This entity allows configuration of cellular logging and is only to be used during compliance testing.
system/cloud_connect	Retrieve or change the Operations Manager appliance system's cloud connect configuration
system/diskspace	Retrieve the system's Disk Space usage.
system/info	Retrieve basic system information.
system/model_name	Retrieve the Operations Manager appliance's Model Name
system/serial_number	Retrieve the Operations Manager appliance's Serial Number
system/session_timeout	Retrieve or change the Operations Manager appliance session timeouts
system/ssh_port	The SSH port used in Direct SSH links
system/ system_authorized_key	Configure the SSH authorized keys for all users.

system/time	Retrieve and update the NetOps Console Server's time.
system/timezone	Retrieve and update the system's timezone.
system/version	Retrieve the Operations Manager's most recent firmware and REST API version
user	Retrieve and update user information

CONFIG CLI COMMANDS

Command	Definition
add	Add a new item for an entity.
apply	Apply changes on just the current entity.
changes	View a list of config areas with unapplied changes.
delete	Delete an item for an entity.
discard	Discard changes on just the current entity
edit	Making changes to configuration options without navigating through the hierarchy.
exit	Leave config mode without applying changes.
help / ?	Display the available options for the configuration section. Can be used in combination with a command or configuration option to access help documentation.
import/export	Copy a config file from a specific network location to the console server and run the file. The import/export commands operate in bash, ie. outside of config CLI.

	You must exit config to operate the import/export features.
show	Display information relevant to the configuration section, highlighting changes.
up/exit/ ..	Allows users to traverse the configuration hierarchy.

ADD

Description

The `add` command will add a new item for an entity. The add command requires a unique value to identify the record. This will be used for the entity's label field.

The **add** command can be used:

- Anywhere within the command structure to begin the process of progressively adding an element.
- As part of a single line command where an element is added and simple fields are set.

Parameters

`entity` - the entity to which the new item will be added

`label` - a unique value to identify the record

`field` - optional field to set for the item

`value` - optional value corresponding to the field

Syntax

```
add <entity> <optional-entity> <label> <optional-field> <optional-value
```

Example

```
add user aconsoleuser description "I am a console user"
```

APPLY

Description

The `apply` command allows users to stage configuration changes by allowing proposed changes to be held in memory, separate from active configuration until they are applied.

This may be considered from a user perspective like this:

"When I am adding users and realize that groups are missing, I can take a pause and add the groups without having to discard my work so far."

or

"When I am in the process of creating a new firewall zone but there is required service missing, I can go off and add the service and come back without losing changes."

Users can choose to apply changes in the following manner:

- Isolated changes that are specific to sections of configuration.
- Across all configurations.

Parameters

When no parameters are provided, the command will apply the changes in the current item context. For example, if the current context is `user consoleuser`, any changes to the `consoleuser`user will be saved. If the `apply` command is used outside of an item context, this will result in an error.

`apply all` – When the '`all`' parameter is added, the command will apply all changes to all items that have been changed in this session.

Syntax

`apply [all]`

Examples

Apply changes to a single item

These commands change a user. Then the apply command is used while still in the “user myuser” item context so only changes to this user are applied:

```
config: user myuser
config(user myuser): password secret123 description "This is my user"
config(user myuser): apply
```

Apply all changes

These commands add a new group and then change a port setting. At the end, the apply all command saves both the group and port items.

```
config: add group mygroup
config: group mygroup
config(group mygroup): access_rights
config(group mygroup access_rights): add pmshell
config(group mygroup access_rights): up
config(group mygroup): ports
config(group mygroup ports): add port01
config(group mygroup ports): top
config: port port01
config(port port01): label "Port for my group"
config(port port01): top
config: apply all
```

Apply changes to specific sections of configuration

From within a specific section of hierarchy. For example, if the user is in the

```
config users johnsmith
apply
```

This will apply any changes made specifically within the user's configuration section.

Apply changes from a different section in the hierarchy

For example, if changes have been made in

```
config users johnsmith
```

but the user has moved elsewhere in the hierarchy, the command:

```
apply users johnsmith
```

will apply any changes made specifically within the user's configuration section.

Alternatively, a user might choose to apply all changes in the user list using the following command:

```
apply users
```

Using `apply` across all configurations

```
apply
```

```
apply all
```

CHANGES

Description

The `changes` command allows users to view a list of config areas with unapplied changes.

This will be a list, ordered alphabetically. Users should be able to copy and paste items from the list and use it in conjunction with the show command to view details.

Parameters

none

Syntax

changes

Examples

The following example shows changes made to multiple users and a port:

```
config: edit user root description "New description"
config: add user newuser description "New User"
config: edit port port01 baudrate 115200
config: changes
Entity user item root (edit)
 description New description
Entity user item newuser (add)
 description New User
Entity port item port01 (edit)
 baudrate 115200
```

DELETE

Description

The `delete` command is used to delete an item or entity or remove a config section or sub-section. The command requires a unique value to identify the record. This will be used for the entity's label field.

Similar to the `add` command, `delete` makes the change in a temporary state and will affect configuration only once applied.

The `delete` command can be used on:

- Existing configuration
- Unapplied changes

When used on unapplied changes, this will behave in the same way as the `discard` command.

Parameters

`entity` - the entity from which to delete the item.

`Item-label` - the label identifying the item to delete.

Syntax

```
delete <entity> <optional-entity> <item-label>
```

Example

```
delete user aconsoleuser
config:
```

Removing an element

From the users context:

```
delete "username"
```

Single line command

```
delete user "username"
apply
```

Either of the above examples will result in exiting the context of an item being deleted.

Refer to the `apply` command for how this will behave.

DISCARD

Description

The discard command is used to remove unapplied changes.

This can be used to discard specific or configuration wide changes including:

- Updates to configuration items
- Unapplied additions
- Items designated for deletion

Parameters

`discard` - when used on its own discard the current item when in an item context, otherwise it will be an error.

`discard all` - when used with the 'all' command, then any changes staged in the current session will be dropped.

Syntax

```
discard [all]
```

Examples

The following commands create a user and then discard the user (it is never saved).

Note: The context changes to exit the ‘myuser’ item since it no longer exists.

```
config: add user myuser
```

```
config: user myuser
```

```
config(user myuser): discard
```

Discard changes

config(user):

The following commands discard changes to an existing item. The item isn’t removed in this case since it has been applied previously. The description field will revert back to whatever it was before.

```
config: user root
config(user root): description "Root user"
config(user root): discard
```

The following commands discard changes to multiple entities, the group and port entities. Both will be reverted:

```
config: edit group admin description "New group description"
config: edit port port01 label "New label"
config: discard all
```

Discard all changes

```
discard *
```

This will result in a confirmation being displayed.

Discard groups of changes

```
discard auth user "username"
```

- If "username" is an addition that has not been applied, it will result in the added user being discarded. In this case the user will be prompted to confirm before the command is implemented.
- If "username" is an existing user with unapplied configuration changes, this will result in any changes there being discarded. A confirmation will be required.
- If "username" is an existing user but with no changes, the user will be informed that there are no configuration changes to discard.

Discarding specific changes

```
port port01
discard
```

- If the entity has unapplied changes it will be discarded.
- If there are no unapplied changes an information message is displayed.

Confirmation

Discarding changes at a section, or configuration wide level will give a warning that multiple changes will be discarded.

EDIT

Description

The edit command is used when making changes to configuration options without navigating through the hierarchy.

Parameters

`entity` - the entity to be edited.

`item-label` - unique value that identifies the item.

`record field` - the field to set for the item.

`value`- the value corresponding to the field.

Syntax

edit <entity> <optional-entity> <item-label> <field>

<value>

Examples

Consider the following change to a port label:

```
config
port
port_01
label "Office-switch"
```

Alternatively, consider making the change from the root of configuration mode.

```
config
edit port port_01 label "Office-switch"
```

EXIT

Description

The `exit` command can be run at any level in the configuration structure and will allow you to leave config mode. If there are unapplied changes, you are informed and asked to confirm it you wish to proceed.

Parameters

There are no parameters applicable to the exit command.

Syntax

```
exit
```

Example

```
exit
```

HELP (OR ?)

Description

Note: Config mode will accept either `help` or a question mark `?` input.

Can be used in the following ways:

- A standalone command to view available options for the configuration section.
- In combination with a command to access help documentation.
- In combination with a configuration option to access help documentation and examples.

Parameters

The `help` command shows help for the current context.

`command` - shows help for the command.

`field` - shows help for the field.

Syntax

```
help <command or field>
```

```
<command or field> ?
```

Examples

The following will print help for the “port port01” context:

```
config(port port01): help
```

or

```
config(port port01): ?
```

The following will print help for the baudrate field when in the “port port01” context:

```
config(port port01): help baudrate
```

or

```
config(port port01): baudrate ?
```

Help command used standalone

When used by itself, `help` or `?` returns a list of available commands or configuration options.

Help used in conjunction with a command

```
apply ?
```

When used in conjunction with a command, `help` displays available sub-options.

For example, when running the `apply` command from the root config level, the `help` command notifies you that changes will traverse the configuration structure, however, when running the `help` command from within a configuration section, changes will apply to configuration options contained within.

```
add user ?
```

Displays help content including syntax and config items (mandatory and optional).

Help used with a configuration option

In the context of this example, the user is running the command from within the port configuration section and is wanting to get information on the available options.

```
pinout ?
```

This will display a list of available options.

```
label ?
```

This will display expected format and a sample.

IMPORT/EXPORT

Description

> **Note:**The import / export and associated commands operate in bash, ie. outside of config CLI. You must exit config to operate the import/export features.

The Import / Export feature allows you to export the current configuration to a file and import or restore the configuration from that file. An import will add configuration to the current configuration and restore will replace the current configuration with the contents of the configuration file.

Import

Running the import command (within bash, not in config:) will allow you to import a configuration script from an external source file. You should point the console server to a config file on specific network location. The file will be copied to the console server and run. Depending on how it has been set up, the changes can be automatically applied after the config file is run.

Export

Running the export command (within bash, not in config:) will allow you to generate a configuration script based on the existing configuration on the console server.

This command can be run at any level in the hierarchy and used to export either:

- The configuration across the node
- Configuration specific to the users's location in the hierarchy.

```
export all current config
```

Will display all config on the console server before it has been applied for copying.

```
export all saved config
```

Will display all saved config on the console server for copying.

```
export current config
```

Will display the config from the users's current position in the navigation hierarchy.

Parameters

Import and export are run from outside of the config shell. The config command is invoked from bash with different parameters to cause it to import or export the configuration without entering the config shell.

filename – The name of the file to be imported from or exported to. If omitted then `stdin` or `stdout` will be used.

Syntax

```
config export <optional filename>
```

```
config import <optional filename>
```

Examples

```
config export /tmp/console_server.config
```

```
config import /tmp/console_server.config
```

Positional arguments

{export,import,restore,merge,replace,get}

Positional Argument	Description
`export`	Export the current configuration.
`import`	Import config from a file.
`restore`	Restore config from a file.
`merge`	Merge a provided list with existing config.
`replace`	Replace a list or item.
`get`	Display an entity's associated values
Options	

`-h, --help`	Show this help message and exit.
`--show-config`	Display the entire configuration and exit.
`-d`	Increase debugging (up to 3 times).
`-j`	Export in json format.
`--entities`	Display entities and exit.

Exporting to a file

Note: The import/export and associated commands operate in bash, ie. outside of config CLI. You must exit config to operate the import/export features.

SHOW

Description

The `show` command displays information relevant to the configuration section, including the highlighting of changes. The context in which the command is run will determine what is displayed.

At `config root`, the `show` command will display system information.

Within a config section, for example from **config > auth > user**, this will display a flat list of available users.

Parameters

show	Used on its own, will display the fields of the current context. When used in the top context, it shows the list of all entities. When used in an entity context, it shows the list of items in that entity. When used in an item context, it shows the fields and values of the current item.
entity	The entity to display, or to show details of.
item	The item to display or show details of.
field	The field to show the value of.

Syntax

show <optional entity> <optional item> <optional field>

Context

Examples using context

The following examples show how the output of the show command changes in accordance with context as it may be used at the config, physif, net1 contexts:

show - at the config context:

```
config: show

Entities
========
access_right                          pdus/drivers
auth                                  physif
auto_response/beacon                  port
auto_response/reaction                port_session
auto_response/status                  ports/auto_discover/schedule
auto_response/status/beacon-module    ports/status_port
cellfw/info                           services/brute_force_protection
cellmodem                             services/lldp
cellmodem/sim                         services/ntp
conn                                  services/routing
failover/settings                     services/snmp_alert_manager
failover/status                       services/snmpd
firewall/policy                       services/ssh
firewall/predefined_service           services/syslog_server
firewall/service                      services/tftp
firewall/zone                         ssh/authorized_key
group                                 static_route
ip_passthrough                        system/admin_info
ip_passthrough/status                 system/banner
ipsec_tunnel                          system/cell_reliability_test
lighthouse_enrollment                 system/cellular_logging
local_password_policy                 system/cloud_connect
logs/portlog                          system/diskspace
logs/portlog_settings                 system/info
managementport                        system/model_name
monitor/brute_force_protection/ban    system/serial_number
monitor/lldp/chassis                  system/session_timeout
monitor/lldp/neighbor                 system/ssh_port
monitor/static_routes/status          system/system_authorized_key
monitoring/alerts/networking          system/time
monitoring/alerts/power               system/timezone
monitoring/alerts/system              system/version
pdu                                   user

config:
```

show - at the physif context:

```
config: physif
config(physif): show
Item names for entity physif
  net1
  net2

config(physif):
```

show - at the net1 context:

```
config(physif): net1
config(physif net1): show
Entity physif item net1
  description NET1 - 1G Copper/SFP
  enabled     true
  mtu         1500
  dns (object)
    nameservers (array)
    search_domains (array)
  ethernet_setting (object)
    link_speed auto

config(physif net1):
```

Examples using parameters

The following examples show the output of the show command when used with different parameters:

```
config: show physif
Item names for entity physif
  net1
  net2

config: show physif net1
Entity physif item net1
  description NET1 - 1G Copper/SFP
  enabled     true
  mtu         1500
  dns (object)
    nameservers (array)
    search_domains (array)
  ethernet_setting (object)
    link_speed auto

config:


config: show physif net1 description
NET1 - 1G Copper/SFP
config:
```

Config

You can view the content of all configuration in JSON format.

You can also view the config of a specific section of the hierarchy you are in.

```
show-config
```

Directed Usage

You will also be able to look into a config sections using the show command. For example:

```
show auth user
```

Will display a flat list of users.

```
show auth user “username”
```

Will display the configuration for the user specified.

UP / EXIT / ..

Description

These commands allow users to traverse the configuration hierarchy.

```
up
```

The position will move one level up in the hierarchy.

If used at the root configuration level, it should point trigger the exit command.

Parameters

No parameters.

Syntax

```
up
```

```
exit
```

Examples

If, as in this example, the context is a specific port, then the ports entity can be accessed by using the `up` command then moving into another port:

```
config: port port01
config(port port01): up
config(port): port02
config(port port02):
```

HOW CHANGES ARE APPLIED OR DISCARDED

When fields and entities are changed, the changes are not immediately applied to the system configuration but remain in a staged status. Items that are staged are indicated by an '*' (asterisk) when the 'show' command is used. In addition, the 'changes' command can be used to show what fields have been changed.

In the following example, the user 'john' has been changed to alter the description. The 'show ' command indicates the changed field with an '*'. The changes command lists the changed field.

```
config(user john): description "Admin"
config(user john): show
Entity user item john
 description Admin * enabled true
 no_password false password false
 password
 ssh_password_enabled true
 groups (array)
```

APPLYING OR DISCARDING CHANGES

Once fields and entities have been changed, they are not yet applied to the system configuration but are kept staged. Items that are staged are indicated with an ‘*' when the ‘show’ command is used. In addition, the 'changes’ command can be used to show what fields have been changed.

When any changes have been made to a single or multiple entities, the following commands become available. These commands are described in detail in the Config CLI Commands section:

Command	Description
changes	Show staged changes on all entities.
apply	Apply changes only on the current entity.
discard	Discard changes only on the current entity.
apply all	Apply changes on all entities.
discard all	Discard changes on all entities.

Example

In the following example, the user ‘john’ has been changed to alter the description. The ‘show’ command indicates the changed field with an asterisk '*'. The changes command lists the changed field.

```
config(user john): description "Scrum Master"
config(user john): show
Entity user item john
description Scrum Master *
enabled true
no_password false
password
ssh_password_enabled true
groups (array)
config(user john): changes
Entity user item john (edit)
description Scrum Master
config(user john):
```

MULTI-FIELD UPDATES

Description

Within config shell, it is possible to update multiple fields with one command line. This is restricted to ‘flat’ fields within the current context ie arrays and sub-objects cannot currently be updated all in one command line.

For example, the following port fields can all be changed in a single command: `baudrate, databits, escape_char, label, logging_level, mode, parity, pinout` and `stopbits`. Other complex fields such as `control_code` and `ip_alias` cannot be modified from the port item context in one commands (multiple commands are needed).

Example

The following command sets the `baudrate, escape_char` and `label` fields.

```
config(port port01): baudrate 115200 escape_char ! label "My Router"
```

The changes will be staged in config shell. Use the `apply` command to save the changes to config.

To further update the `control_codes` and `ip_aliases`, multiple commands are required as follows:

```
config(port port01): control_code
config(port port01 control_code): break b chooser c
config(port port01 control_code): up
config(port port01): ip_alias
config(port port01 ip_alias): add
config(port port01 ip_alias 1): interface net1 ipaddress 10.83.0.6/24
config(port port01 ip_alias 1): up
config(port port01 ip_alias): up
```

```
config(port port01): changes
Entity port item port01 (edit)
  control_code (object)
    break b
    chooser c
  ip_alias (array)
    1 (object)
      interface net1
      ipaddress 10.83.0.6/24
config(port port01):
```

If certain fields are hidden and only visible by first configuring other fields, these hidden fields need to be set in another line. For example, the `kernel_debug` field is only revealed by setting the field `mode` of a port to `localConsole`, so this is configured on the next line:

```
config: port port03
config(port port03): mode localConsole baudrate 115200 databits 7
label aaa
logging_level eventsOnly parity even
config(port port03): kernel_debug true
```

Error Messages

If there is an error while processing a multiple-fields command, the staged values in configuration will not be changed. If there were no staged changes on the item, then no staged changes will appear. If there were already staged changes, then those staged changes will not be affected.

In the following example, the user description was previously changed to “my user”

```
config(user consoleuser): show
Entity user item consoleuser
  description         my user *
  enabled              true
  no_password false
  password            ""
  ssh_password_enabled true
  groups (array)
    0 consoleuser
```

If a bad field name or value is supplied on the command line, then the existing staged value is retained. The bad field name is highlighted using a ^ marker.

```
config(user consoleuser): description "My console user" invalid true
                                                        ^
Invalid input detected at '^' marker.
config(user consoleuser):
```

If the field is missing a value, a different error message is displayed:

```
config(user consoleuser): description "My console user" enabled
Incomplete command.
config(user consoleuser): show
Entity user item consoleuser
  description         my user *
  enabled true
  no_password false
  password            ""
```

```
ssh_password_enabled true
groups (array)
  0 consoleuser
```

The bad value for the field is indicated by an error message hinting the expected type of the value:

```
config(user consoleuser): description "My console user" enabled bad
Value bad for field enabled cannot be parsed as a boolean.
config(user consoleuser): show
Entity user item consoleuser
  description my user *
  enabled              true
  no_password         false
  password             ""
  ssh_password_enabled true
  groups (array)
    0 consoleuser
```

Changes to previous functionality

With the new `show` command, some previous syntax has changed. Just typing a field name is now an error condition. Previously this would be equivalent to the show command.

```
config: user root
config(user root): description
Incomplete command.
config(user root):
```

ERROR MESSAGES

When an error is made in the command line an error message which identifies the error is returned. For example, if the first token of the command is mistyped, the unknown command message is displayed.

```
config: usear root
There is no command usear root.
Type 'help' to see the available commands.
config:
config: aaaaa
There is no command aaaaa.
Type 'help' to see the available commands.
config:
```

If only the first few tokens of the command can be parsed, an error message with a ^ marker is displayed showing which part of the command cannot be parsed. If a context navigation is mistyped on the command line, then the context remains unchanged. It does not partially navigate through multiple contexts. In the following example, the context remains at the top context because `roopt` is not a valid item context in the user entity context.

```
config: user roopt
             ^
invalid input detected at '^' marker.
config:
```

STRING VALUES IN CONFIG COMMANDS

Description

The syntax for the use of string values has changed. It was previously possible to enter values containing spaces without using quotes. Multiple fields can now be assigned in one command line, quotes are required to keep field values together.

Example

The following example shows setting multiple fields where the field value for the description has spaces. The first attempt doesn't work because the second part of the description is interpreted as a field name. The second attempt is the correct syntax:

> **Note:** In the example the syntax error in the first line in highlighted in **bold** for clarity; Ihe correct syntax is highlighted in bold in line four.

```
config(user consoleuser): description My console user enabled true
There is no command description My console user enabled true.
Type 'help' to see the available commands.
config(user consoleuser): description "My console user" enabled true
config(user consoleuser): changes
Entity user item consoleuser (edit)
  description My console user
  enabled true
config(user consoleuser):
```

If the value itself must contain quotes, there is a triple quote form for entering string values:

```
config(user consoleuser): description """My "console" user""" enabled true
config(user consoleuser): changes
Entity user item consoleuser (edit)
  description My "console" user
  enabled true
```

The triple quoted string is used for entering multi-line strings:

```
config(system/banner): banner """
This is a banner that has
multiple lines.
"""
config(system/banner):
```

Error Messages

If the multi-line command string cannot be tokenised, an error message will be displayed in the following form:

```
config(system/banner): banner """
aaa
""""
Invalid input. Tokens must be separated by whitespace.
Check your input and try again.
config(system/banner):
```

CONFIG CLI USE CASE EXAMPLES

ADDING A USER

The following is a fully worked example showing the adding of a new user.

Note: In the following examples, some commentary has been added, the commentary is denoted with a '//' prefix. Where sessions continue onto the next page, this is shown with the comment "// session continues here:"

```
# config
Welcome to the Opengear interactive config shell. Type ? or help for help.
// Move to the user entity

config: user
config(user): help add
Add a new item for entity user.

The add command requires a unique value to identify the record.
This will be used for the username field.

Description for the item:
  Retrieve and update information for a specific user.

// Create the new user

config(user): add matt
config(user matt): show
Entity user item matt
```

```
  description

// Session continues here:
  enabled              true
  no_password          false
password                    (required)
ssh_password_enabled true
username             matt
groups (array)

// Fill out some fields

config(user matt): password topsecretpassword
config(user matt): description scrum master
config(user matt): show
Entity user item matt
  description          scrum master *
  enabled              true
password               topsecretpassword *
ssh_password_enabled   true
username               matt
groups (array)

// Edit the groups

config(user matt): groups
config(user matt groups): show
Entity user item matt field groups
config(user matt groups): add // Tab completion to show available values
```

```
admin myuser netgrp
config(user matt groups): add admin
config(user matt groups): up // Exit the groups list
// Session continues here:
// Show and apply

config(user matt): show
Entity user item matt
  description          scrum master *
  enabled              true
  password             topsecretpassword *
  ssh_password_enabled true
  username             matt
  groups  (array)
    0 admin *
config(user matt): apply
Creating entity user item matt.
config(user matt):
```

CONFIGURING A PORT

```
config: port
config(port): help
You are here: entity port

Description for the entity:
   Configuring and viewing ports information
```

```
Names (type <name> or help <name>)
==================================
USB-A USB-E USB-front-lower port03 port07 port11 port15 port19 port23
USB-B USB-F USB-front-upper port04 port08 port12 port16 port20 port24
USB-C USB-G port01          port05 port09 port13 port17 port21
USB-D USB-H port02 port06   port10 port14 port18 port22

Commands (type help <command>)
==============================
exit help show up

config(port): port01
config(port port01): baudrate // tab completion
110 1200 150 19200 230400 300 4800 57600 75
115200 134 1800 200 2400 38400 50 600 9600
config(port port01): baudrate 57600
config(port port01): label Router
config(port port01): control_code
config(port port01 control_code): break a
config(port port01 control_code): up
config(port port01): show
// Session continues here:
Entity port item port01
  baudrate        57600 *
  databits        8
  escape_char     ~
  label Router    *
  logging_level   disabled
  mode            consoleServer
```

```
  parity          none
  pinout          X2
  stopbits        1
  control_code  (object)
    break a *
    chooser
    pmhelp
    portlog
    power
    quit
  ip_alias (array)
config(port port01): apply
Updating entity port item port01.
config(port port01):
```

CONFIGURE A SINGLE SESSION ON A PORT

The feature is enabled by typing `single_session true`, then apply the change.

```
config(port port01):      single_session true
config(port port01):      apply
Updating entity port      item port01.
config(port port01):      show
Entity port item              port01
    baudrate       9600
...
single_session                true
...
ip_alias (array)
```

CONFIGURE NET1 STATIC IPV4

```
conn default-conn-1 ipv4_static_settings
  address 192.168.2.54
  gateway 192.168.2.1
top
```

CONFIGURE NET2 STATIC IPV4

```
add conn net2-static-1 mode static physif net2
conn net2-static-1 ipv4_static_settings
  address 192.168.3.58
  gateway 192.168.3.1
  netmask 255.255.255.0
top
```

CONFIGURE NET3 STATIC IPV4 FOR OM2224-24E UNITS

```
add conn net3-static-1 mode static physif net3
conn net3-static-1 ipv4_static_settings
  address 192.168.4.58
  gateway 192.168.4.1
  netmask 255.255.255.0
top
```

CONFIGURE WIREGUARD THROUGH CONFIG SHELL

WireGuard is configured through Config Shell (or REST API). The minimum configuration of WireGuard is shown in the following:

1. Provide a name for the interface (wg0 in the example below).
2. Set enabled.
3. Set the private_key of your WireGuard interface.
4. Add an address (at least one) for your WireGuard interface (10.0.0.1/24 in this case).
5. Add a peer with the following parameters: endpoint_address, endpoint_port, public_key.
6. Add an allowed_ip for your peer. At least one - this is the WireGuard address(es) (as it can also accept an address range) of the other interface to which you are connected.

For example:

```
config: wireguard
config(wireguard): add wg0
config(wireguard wg0): private_key
AGiZvFHY+r/dD0rHSKU5ZCrHNdLM0W/h29VxobxWgFo=
config(wireguard wg0): enabled true
config(wireguard wg0): addresses
config(wireguard wg0 addresses): add 10.0.0.1/24
config(wireguard wg0 addresses): up
config(wireguard wg0): peers
config(wireguard wg0 peers): add
config(wireguard wg0 peers 0): public_key
```

```
o+quB4sbUAG2hEGSPpMNTnO0YSaQTP7dD+Q4IVjiCW8=
config(wireguard wg0 peers 0): allowed_ips
config(wireguard wg0 peers 0 allowed_ips): add 10.0.0.2/32
config(wireguard wg0 peers 0 allowed_ips): up
config(wireguard wg0 peers 0): endpoint_address 192.168.1.2
config(wireguard wg0 peers 0): endpoint_port 51820
config(wireguard wg0 peers 0): up
config(wireguard wg0 peers): top
```

ROOT USER PASSWORD - CLEARTEXT

```
edit user root password newpassword
```

ROOT USER PASSWORD = PASSOWRD VIA SHA256

openssl passwd -5 password

Note:* this operation is not available in config shell

DEFINE PASSWORD COMPLEXITY RULES

```
edit local_password_policy
  password_complexity_enabled true
  password_expiry_interval_enabled true
  edit local_password_policy
  password_disallow_username true
  password_must_contain_number true
  password_must_contain_special true
  password_must_contain_upper_case true
```

HOSTNAME

```
edit system/admin_info hostname "OM2216-1-lab"
```

CONTACT INFO

```
edit system/admin_info
  contact "fred.bloggs@opengear.com"
  hostname "om2216-1.lab"
  location "Happy Valley Lab"
```

TIME ZONE AND NTP

```
edit system/timezone timezone "America/New_York"


edit services/ntp enabled true
services/ntp servers
  add
  value "74.207.242.234"
top
```

CREATE ADMIN USER

```
add user admin
  description "admin"
  enabled true
  no_password false
  password "password"
  user admin groups
  add "admin"
top
```

CREATE BREAKGLASS USER (BELONGS TO NETGRP)

```
add user breakglass
  description "breakglass" enabled true
  no_password false
  password "password"
  user breakglass groups
  add "netgrp"
top
```

ENABLE NETGRP - SET TO CONSOLEUSER

```
edit group netgrp enabled true
group netgrp ports
  add port01
  add port02
  add port03
  add port04
top
group netgrp access_rights
  add web_ui
  add pmshell
  delete admin
top
```

CHANGE SSH DELIMIITER TO : DEFAULT IS +

```
edit services/ssh ssh_url_delimiter ":"
```

CHANGE PORT LABELS

```
edit port port01 label "cisco1"
edit port port02 label "cisco2"
edit port port03 label "cisco3"
edit port port04 label "cisco4"
```

ENABLE TACACS - SET MODE TO REMOTELOCAL

```
edit auth mode "tacacs"
edit auth tacacsMethod "pap" tacacs
Password "tac_tests"
policy "remotelocal"
tacacsService "raccess"
auth tacacsAuthenticationServers
  add
  hostname "192.168.2.220"
  port 49
top
```

ENABLE LLDP ON NET1 & NET2

```
edit services/lldp enabled true
services/lldp physifs
  add "net1"
  add "net2"
top
```

ENABLE TFTP

```
edit services/tftp enabled true
```

ENABLE BOOT MESSAGES

Displays on local console port.

```
edit managementport ttyS0 kerneldebug true
```

DEFINE SESSION TIMEOUTS

```
edit system/session_timeout cli_timeout 100 serial_port_timeout 100 webui_timeout 100
```

Note: The inactivity timer starts only after you exit config shell, ie. it begins the count when you have left config and are at the bash command prompt.

DEFINE MOTD

Enter banner text within quotations.

```
edit system/banner banner """
```

ENABLE SIMM 1 ENABLE AND ADD APN

```
edit physif wwan0 enabled true
physif wwan0 cellular_setting
  apn hologram
top
```

ENABLE SIMM 1 COMPLETE END POINTS

```
edit physif wwan0 enabled true
physif wwan0 cellular_setting
  active_sim 1
  apn hologram
  iptype IPv4v6
  sim_failback_disconnect_mode ping
  sim_failback_policy never
  sim_failover_disconnect_mode ping
  sim_failover_policy never
top
physif wwan0 cellular_setting sims 0
  fail_probe_address 8.8.8.8
  fail_probe_count 3
  fail_probe_interval 600
  fail_probe_threshold 1
  failback_delay 60
  iptype "IPv4v6"
  slot 1
top
physif wwan0 cellular_setting sims 1
  fail_probe_address 8.8.8.8
  fail_probe_count 3
  fail_probe_interval 600
  fail_probe_threshold 1
  failback_delay 60
```

```
  iptype IPv4v6
  slot 2
top
```

ENABLE FAILOVER

```
edit failover/settings enabled true probe_address 192.168.2.1 probe_physif net1
```

ADD A SYSLOG SERVER

```
services/syslog_server
  add server1
  address 192.168.34.113
  protocol TCP
  port 610
  description "my syslog server"
top
```

Add Five Syslog Servers

Note: Due to page width limitations, in the following example, some command lines break over two lines.

```
add services/syslog_server server0 address 192.168.34.112 min_severity notice
port 514 port_logging_enabled true protocol UDP
add services/syslog_server server1 address 192.168.34.113 min_severity notice
port 514 port_logging_enabled true protocol UDP
add services/syslog_server server2 address 192.168.34.114 min_severity notice
```

```
port 514 port_logging_enabled true protocol UDP
add services/syslog_server server3 address 192.168.34.116 min_severity info
port 514 port_logging_enabled true protocol UDP
add services/syslog_server server4 address 192.168.128.1 description
"lighthouse-remote-syslog" min_severity info port 514 port_logging_enabled
true protocol UDP
```

SET PORT LOGGING REMOTE SYSLOG SETTINGS

```
edit logs/portlog_settings facility daemon severity infoEnable system
monitor snmp traps
```

ENABLE SYSTEM MONITOR SNMP TRAPS

```
monitoring/alerts/power power_supply_voltage_alert
  millivolt_lower 11000
  millivolt_upper 13000
  snmp
    enabled true
  up
top
monitoring/alerts/networking cell_signal_strength_alert
  enabled true
  threshold_lower 33
  threshold_upper 66
top
monitoring/alerts/system
  authentication_alert
    enabled true
  up
  config_change_alert
    enabled true
  up
  temperature_alert
    enabled true
    threshold_lower 35
    threshold_upper 67
  up
top
```

ENABLE SNMP V2 SERVICE FOR POLLING

```
edit services/snmpd enable_legacy_versions true
enable_secure_snmp false enabled true port 161 protocol UDP
edit services/snmpd rocommunity
"TkcxJAAAABBfDsigaxdDf7whb3sxKQKnjtCuuy/0COC6rE3lUu9ghg=="
```

ENABLE 2 SNMP TRAPS AND TRAP SERVERS

Note: Due to page width limitations, in the following example, some command lines break over two lines.

```
add services/snmp_alert_manager "snmp trap server 1" address 10.1.1.199 port
162 protocol UDP version v2c
services/snmp_alert_manager "snmp trap server 1"
  community "TkcxJAAAABBfDsigaxdDf7whb3sxKQKnjtCuuy/0COC6rE3lUu9ghg==" msg_type TRAP
  top
apply all


services/snmp_alert_manager 10.1.1.199:162/UDP
  name "snmp trap server 1" privacy_password secret auth_password secret
  top
apply all
```

CREATE A STAIC ROUTE

Note: Due to page width limitations, in the following example, some command lines break over two lines.

```
add static_route "static route test" destination_address 10.0.0.0
destination_netmask 8 interface net2
```

EDIT LAN (NET2) FIREWALL ZONE

(allow only source address traffic)

```
firewall/zone lan custom_rules
  add
    description "source_net4-1"
    rule_content "rule family=ipv4 source address=192.168.3.0/24 accept"
  up
  add
    description "source_net4-2"
    rule_content "rule family=ipv4 source address=10.202.198.0/27 accept"
  up
top
```

EDIT WAN (NET1) FIREWALL ZONE

(allow only source address traffic)

```
firewall/zone wan custom_rules
  add
    description "source_net4-1"
    rule_content "rule family=ipv4 source address=192.168.2.0/24 accept"
  up
  add
    description "source_net4-2"
    rule_content "rule family=ipv4 source address=192.168.4.0/24 accept"
  up
top
```

CUSTOM_RULE EXAMPLE FOR PORT AND PROTOCOL

```
add firewall/service myports label "My Serial Ports"
firewall/service myports
  add
    port 3001
    protocol tcp
  up
  apply
top
firewall/zone wan address_filters
  add
    source_address 10.10.2.0/19
    services
      add myports
    up
  up
top
```

ENROLL INTO LIGHTHOUSE

```
add lighthouse_enrollment lh1 address 2.21.99.188 bundle om2216-1 token password
```