CUSTOMIZING THE CONFD CLI: CONFIGURATION MODE

tail-f Tail-f is now part of Cisco. cisco

Table of Contents

1. Introduction

One of ConfD's data model driven features is the ability to automatically render a CLI from a YANG data mode. This powerful feature provides both development velocity and productivity benefits. The user can easily write a YANG data model, start ConfD, and log into a fully functional, industry-standard style CLI. However, the automatically rendered CLI maybe not be exactly what is desired. Therefore, ConfD provides a very extensive set of YANG extension statements which can be used to annotate a YANG data model in order to customize the auto rendering.

This application note focuses on the most important configuration mode CLI extensions that are commonly used when developing a CLI interface. For more information regarding these annotations please refer to the ConfD User Guide.

2. Background

YANG

YANG is a data modeling language for the definition of data sent over the NETCONF network configuration protocol. The data modeling language can be used to model both the configuration data as well as the state data and administrative actions of network elements. Furthermore, YANG can be used to define the format of event notifications which can be generated by network elements and also allows data modelers to define the signature of extension remote procedure calls (RPCs) which can be invoked on network elements via the NETCONF protocol.

ConfD

ConfD is an embedded management software framework that enables network element providers to quickly and inexpensively deliver world-class management functionality and programmability for their products. The network elements can be physical devices or virtual devices (VNFs, Virtual routers, etc.). ConfD is data model-driven and provides automatic rendering of all northbound interfaces including NETCONF, RESTCONF, CLI, JSON-RPC, and SNMP as well as C, Python, JAVA, and Erlang APIs for application integration.

ConfD

The CLI extensions are predefined annotations that are inserted into a YANG data model. The purpose of these annotations is to make the development of CLI (command line interface) easier and reduce the time to market for a given product. The CLI extensions can change the look and feel of the auto-rendered configuration commands, the operational commands, as well as the running configuration.

These YANG extension statements were written by Tail-f and are defined in a YANG data model which is included which the ConfD distribution. The CLI extension statements are defined in tailf-cli-extension.YANG which is a submodule of tailf-common.YANG. To make use of the CLI extensions in a YANG data model, simply import tailf-common.YANG:

```
import tailf-common {
  prefix tailf;
}
```

3. tailf:cli-add-mode

Definition: Creates a mode of the container.

Usage: Containers are usually used for grouping other types of data (leafs, lists, etc). Unlike lists, containers don't have a mode by default. This means that you can't type <cr> after configuring a container. A "presence true" YANG statement can resolve the issue but the configuration mode is absent. This is why this CLI extension is useful: It lets you type <cr> after configuring the container and also changes and creates a mode of the configuration in the CLI. Used for C-style and I-style CLI.

Example:

Case 1: (no CLI extension)

YANG model:

```
container routing {
  container ospf {
    leaf routerID {
      type string;
    }
  }
}
```

CLI output:

```
device(config)# routing ospf ?
Possible completions:
  routerID
device(config)# routing ospf   <CR>
-----------------------------------^
syntax error: incomplete path
device(config)#
```

Case 2: (with CLI extension)

YANG model:

```
container routing {
  container ospf {
    tailf:cli-add-mode;
    leaf routerID {
      type string;
    }
  }
}
```

CLI output:

```
device(config)# routing ospf ?
Possible completions:
  routerID
  <cr>
device(config)# routing ospf <CR>
device(config-ospf)#
```

Tip:

- Every time an operator wants to return to the previous menu in CLI, they have to type “exit”. The more add-mode statements there are, the more exit commands the operator has to type.
- Used in conjunction with the tailf:cli-full-command extension, this can force the operator to enter the mode with no other configuration being allowed on the same line.

4. tailf:cli-mode-name “value”

Definition: Specifies a custom mode name, instead of the default which is the name of the list or container node.

Can be used in configuration nodes only. If used in a container, the container must also have a tailf:cli-add-mode statement, and, if used in a list, the list must not also have a tailf:cli-suppress-mode statement.

Usage: This extension is used when you want to change the name of the mode for a specific element to make it more intuitive or when you want to reduce the size of the characters inside the configuration mode. Used C-style and I-style CLI.

Example 1: (Reduce the size of configuration mode characters)

Case 1: (no CLI extension)

YANG model:

```
container routing {
  container ospf-version-2 {
    tailf:cli-add-mode;
    leaf routerID {
      type string;
    }
  }
}
```

CLI output:

```
device(config)# routing ospf-version-2 <CR>
device(config-ospf-version-2)#
```

Case 2: (with CLI extension)

YANG model:

```
container routing {
  container ospf-version-2 {
    tailf:cli-add-mode;
    tailf:cli-mode-name "config-ospfv2";
    leaf routerID {
      type string;
    }
  }
}
```

CLI output:

```
device(config)# routing ospf-version-2 <CR>
device(config-ospfv2)#
```

Example 2: (Cosmetic improvement of the configuration mode. In this case, making it Unix style. The operator knows exactly that ospf it's under routing)

Case 1: (no CLI extension)

YANG model:

```
container routing {
  container ospf-version-2 {
    tailf:cli-add-mode;
    leaf routerID {
      type string;
    }
  }
}
```

CLI output:

```
device(config)# routing ospf-version-2 <CR>
device(config-ospf-version-2)#
```

Case 2: (with CLI extension)

YANG model:

```
container routing {
  container ospf-version-2 {
    tailf:cli-add-mode;
    tailf:cli-mode-name config(routing/ospf);
      leaf routerID {
      type string;
    }
  }
}
```

CLI output:

```
device(config)# routing ospf-version-2 <CR>
device(config(routing/ospf))#
```

Tip:

Make sure to preserve the same look and feel across the device. If you start using this command for cosmetic improvement (Unix like or other things like that) take into account that you have to do this for every element.

5. tailf:cli-drop-node-name

Definition: Specifies that the name of a node is not present in the CLI.

Usage: There are some cases in which you don't need the actual name of an element but the values underneath, when configuring from CLI. In some other situations you want to group some elements inside a container but wish the container name to be invisible. This is what this extension is for. Used for C-style and I-style CLI.

Example:

Case 1: (no CLI extension)

YANG model:

```
container system {
  container routing {
    container ospf {
      container timers {
        leaf timer1 {
          type uint16;
        }
        leaf timer2 {
          type uint16;
      }
    }
  }
}
```

CLI output:

```
device(config)# system routing ospf timers ?
Possible completions:
  timer1  timer2
device(config)# system routing ospf timers timer1 2 timer2
3 <CR>
device(config)#
```

Case 2: (with CLI extension)

YANG model:

```
container system {
  container routing {
    container ospf {
      container timers {
        leaf timer1 {
          tailf:cli-drop-node-name;
          type uint16;
        }
        leaf timer2 {
          tailf:cli-drop-node-name;
          type uint16;
        }
      }
    }
  }
}
```

CLI output:

```
device(config)# system routing ospf timers ?
Possible completions:
  <unsignedByte>  <unsignedShort>
device(config)# system routing ospf timers 2 3 <CR>
device(config)#
```

Tip:

If you want to see one element entry at a time use tailf:cli-sequence-commands in conjunction with tailf:cli-drop-node-name.

6. tailf:cli-expose-key-name

Definition: Force the user to enter the name of the key and display the key name when displaying the running configuration.

Usage: There are some cases in which you don't need the actual name of an element but the values underneath. In some other situations, you want to group some elements inside a container but wish the container name to be invisible. In these cases, using this extension is helpful. Used for C-style, I-style, and J-style CLI.

Example:

Case 1: (no CLI extension)

YANG model:

```
container system {
  list server {
    key ip-address;
    leaf ip-address {
      type inet:ipv4-address;
    }
    leaf port {
      type uint16;
    }
  }
}
```

CLI output:

```
device(config)# system server ?
Possible completions:
  <ip-address:IPv4 address>  range
device(config)# system server
```

Case 2: (with CLI extension)

YANG model:

```
container system {
  list server {
    key ip-address;
    leaf ip-address {
      tailf:cli-expose-key-name;
      type inet:ipv4-address;
    }
    leaf port {
      type uint16;
    }
  }
}
```

CLI output:

```
device(config)# system server ?
Possible completions:
  ip-address  range
device(config)# system server
```

7. tailf:cli-custom-error “text”

Definition: This statement specifies a custom error message to be displayed when the user enters an invalid value.

Usage: ConfD provides built in validation for all the types. However, if you want a custom error message to be displayed to the operator this extension is useful.

Example:

Case 1: (no CLI extension)

YANG model:

```
leaf port {
  type uint16;
}
```

CLI output:

```
device(config)# port -1  (enter)
---------------------------^
syntax error: "-1" is not a valid value.
```

Case 2: (with CLI extension)

YANG model:

```
leaf port {
  type uint16;
  tailf:cli-custom-error "Invalid port number."+
  "Valid range: 0-65535.";
}
```

CLI output:

```
device(config)# port -1  (enter)
---------------------------^
syntax error: Invalid port number. Valid range: 0-65535.
```

8. tailf:cli-flatten-container

Definition: Allows the CLI to exit the container and continue to input from the parent container when all leaves in the current container have been set.

Usage: The normal YANG behavior for containers is this: After you configure all the elements inside a container you cannot configure anything on the same line (you are stuck within the container). However, there are cases when this is not enough and there is a need to continue to configure the rest of the elements in the YANG model, exiting the container. In the following example, if we start configuring dhcp there will be no chance of configuring leaf active on the same line unless we use this extension. Used for C-style and I-style CLI.

Example:

Case 1: (no CLI extension)

YANG model:

```
container system {
  container dhcp {
    leaf port {
      type uint16;
    }
    leaf subnet {
      type inet:ipv4-prefix;
    }
  }
  leaf active {
    type boolean;
  }
}
```

CLI output:

```
device(config)# system dhcp port 10 subnet 192.168.1.0/24 ?
Possible completions:
  <cr>
device(config)# system dhcp port 10 subnet 192.168.1.0/24
```

Case 2: (with CLI extension)

YANG model:

```
container system {
  container dhcp {
    tailf:cli-flatten-container;
    leaf port {
      type uint16;
    }
    leaf subnet {
      type inet:ipv4-prefix;
    }
  }
  leaf active {
    type boolean;
  }
}
```

CLI output:

```
device(config)# system dhcp port 10 subnet 192.168.1.0/24 ?
Possible completions:
 active <cr>
device(config)# system dhcp port 10 subnet 192.168.1.0/24
```

9. tailf:cli-full-command

Definition: Specifies that an auto-rendered command should be considered complete, i.e., no additional leaves or containers can be entered on the same command line.

Usage: This extension is used when you want to force the operator to end a configuration command. After configuring that element the operator cannot continue with the next one. Used for C-style and I-style CLI.

Example:

Case 1: (no CLI extension)

YANG model:

```
container dhcp {
  leaf port {
    type uint16;
  }
  leaf subnet {
    type inet:ipv4-prefix;
  }
}
```

CLI output:

```
device(config)# dhcp port 10 ?
Possible completions:
  subnet <cr>
device(config)# dhcp port 10
```

Case 2: (with CLI extension)

YANG model:

```
container dhcp {
  leaf port {
    tailf:cli-full-command;
    type uint16;
  }
  leaf subnet {
    type inet:ipv4-prefix;
  }
}
```

CLI output:

```
device(config)# dhcp port 10 ?
Possible completions:
 <cr>
device(config)# dhcp port 10
```

Tip:

There are many network element manufacturers who discourage very large commands sent at a time. Most of them use this extension for every leaf of their model.

10. tailf:cli-incomplete-command

Definition: Specifies that an auto-rendered command should be considered incomplete.

Usage: Can be used to prevent <cr> from appearing in the completion list for optional internal nodes.

This extension is used when you want to lock the operator inside a configuration element. After configuring that element the operator must continue with the next one. Used for C-style and I-style CLI.

Example:

Case 1: (no CLI extension)

YANG model:

```
container dhcp {
  leaf port {
    type uint16;
  }
  leaf subnet {
    type inet:ipv4-prefix;
  }
}
```

CLI output:

```
device(config)# dhcp port 10 ?
Possible completions:
  subnet <cr>
device(config)# dhcp port 10
```

Case 2: (with CLI extension)

YANG model:

```
container dhcp {
  leaf port {
    tailf:cli-incomplete-command;
    type uint16;
  }
  leaf subnet {
    type inet:ipv4-prefix;
  }
}
```

CLI output:

```
device(config)# dhcp port 10 ?
Possible completions:
 subnet
device(config)# dhcp port 10
```

Tip:

Can be used in combination with cli-sequence-commands to makes sure that the user enters all the elements inside a container.

11. tailf:cli-key-format

Definition: The format string is used when parsing a key value and when generating a key value for an existing configuration. The key items are numbered from 1-N and the format string should indicate how they are related by using $(X) (where X is the key number).

Usage: There are special cases where the key leafs of a list need to be referenced using a symbol between them. One good example in the networking world is defining the interfaces of a switch or router. Some devices use chassis/slot/interface for referencing a specific interface which is part of a slot in a given chassis. Other devices use slot:port for the same situation. This is where this extension comes handy, because it allows the operator to reference the keys using a symbol between them. Used for C-style, I-style, and J-style CLI.

Example:

Case 1: (no CLI extension)

YANG model:

```
container system {
  list interface {
    key "chassis slot port"
    leaf chassis {
      type uint8;
    }
    leaf slot {
      type uint8;
    }
    leaf port {
      type uint8;
    }
    leaf ip {
      type inet:ipv4-address;
    }
    leaf mask {
      type inet:ipv4-address;
    }
  }
}
```

CLI output:

```
device(config)# system interface 1 1 1 ip 3.3.3.3 mask
255.255.255.0 <CR>
device(config-interface-1/1/1)# commit <CR>
Commit complete.
device(config-interface-1/1/1)# do show run | include
interface <CR>
system interface 1 1 1
```

Case 2: (with CLI extension)

YANG model:

```
container system {
  list interface {
    tailf:cli-key-format '$(1):$(2)/$(3)';
    key "chassis slot port"
    leaf chassis {
      type uint8;
    }
    leaf slot {
      type uint8;
    }
    leaf port {
      type uint8;
    }
    leaf ip {
      type inet:ipv4-address;
    }
    leaf mask {
      type inet:ipv4-address;
    }
  }
}
```

CLI output:

```
device(config)# system interface 1:1/1 ip 3.3.3.3 mask
255.255.255.0 <CR>
device(config-interface-1/1/1)# commit <CR>
Commit complete.
device(config-interface-1/1/1)# do show run | include
interface <CR>
system interface 1:1/1
```

12. tailf:cli-sequence-commands

13. tailf:cli-break-sequence-commands

Definition: The first extension specifies that an auto-rendered command should only accept arguments in the same order as they are specified in the YANG model. The second extension specifies that previous cli-sequence-command declaration should stop at this point. Only applicable when a cli-sequence-command declaration has been used in the parent container.

Usage: The most obvious usage of this extension is when you want to configure the elements in a container in the same order they are specified in the YANG model. That means that the elements are not shown all at once but one at a time. In the following example, the operator is forced to input the first two leafs in the same order as in the YANG while letting him choose the order for the last two leafs. Used for C-style, I-style, and J-style CLI.

Example:

Case 1: (no CLI extension)

YANG model:

```
container system {
  leaf id {
    type string;
  }
  leaf admin {
    type enumeration {
      enum up;
      enum down;
    }
  }
  leaf time {
    type string;
  }
  leaf user {
    type string;
  }
}
```

CLI output:

```
device(config)#  system ?
Possible completions:
  admin  id  time  user
device(config)#  system admin up ?
Possible completions:
  id  time  user  <cr>
device(config)#  system admin up id 3 ?
Possible completions:
  time  user  <cr>
device(config)#  system admin up id 3 time 12:00 ?
Possible completions:
  user  <cr>
device(config)#  system admin up id 3 time 12:00 user john
?
Possible completions:
  <cr>
device(config)#  system admin up id 3 time 12:00 user john
```

Case 2: (with CLI extension)

YANG model:

```
container system {
  tailf:cli-sequence-commands;
  leaf id {
    type string;
  }
  leaf admin {
    type enumeration {
      enum up;
      enum down;
    }
  }
  leaf time {
    tailf:cli-break-sequence-commands;
    type string;
  }
  leaf user {
    type string;
  }
}
```

CLI output:

```
device(config)#  system ?
Possible completions:
  id
device(config)#  system id 1 ?
device(config)#  system admin up ?
Possible completions:
  admin  <cr>
device(config)#  system id 1 admin up ?
Possible completions:
  time  user  <cr>
device(config)#  system id 1 admin up time 12:00 ?
Possible completions:
  user  <cr>
device(config)#  system admin up id 3 time 12:00 user john
?
Possible completions:
  <cr>
device(config)#  system admin up id 3 time 12:00 user john
```

14. Conclusion

This application note has discussed some of the most commonly used YANG extension statements which are used to customize the default automatic rendering of the ConfD CLI. While this application note has focused on the most commonly used CLI configuration mode customizations, there are many more customization statements available. There are also YANG extension statements for customizing the CLI operational mode. Additionally, the existing commands can be modified or added to using definitions called “clispec” and implemented using scripts or executable programs.

- For more information about ConfD, visit http://www.tail-f.com
- For more information about YANG, visit https://tools.ietf.org/html/rfc7950
- For more information about the ConfD CLI, see the “CLI Agent” chapter of the ConfD User Guide
- For more information about the ConfD CLI YANG extensions, see the tailf_yang_cli_extensions(5) man page
- To see more CLI customization options in action, see examples.confd/cli in your ConfD distribution

Tail-f is now part of Cisco.

www.tail-f.com
info@tail-f.com

Corporate Headquarters

Sveavagen 25
111 34 Stockholm
Sweden
+46 8 21 37 40