Benjamin Rautio

[brautio@juniper.net](mailto:brautio@juniper.net)

12/3/19

**Intern Project**

**Problem:**

JTAC must support 10+ chipsets on MX and PTX, and each chipset has common features, blocks, and counters serving the same function, yet they have unique names. Escalation engineers responsible to train JTAC on PFEs are facing challenges with PFE training materials, causing debugging PFEs to become a soft spot in JTAC.

A picture containing screenshot

Description automatically generated

There are many counters that are specific to certain types of PFEs and have unique names, which makes debugging PFEs and training JTAC very difficult

**Solution:**

We wish to someday have a single PFE CLI which will encompass all ASICs by creating common counter names (chip agnostic counters) and commands for ASICs. This would simplify documentation, training, and automation.

For example, we could create a custom chip agnostic CLI command for the xm and mqss ASICs to display these similar counters:

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

show chip-agnostic stats

time now is 2019-12-12 12:05:13.520804

     fpc4 chip instance 0

     fpc type MPC5E 3D Q 2CGE+4XGE

[..]

          CChipFiStatsTable

               cchip-fi-packets-dropped:                 0

               cchip-fi-packets-with-error:              0

               cchip-fi-packets-received:                19028731401

               cchip-fi-packets-sent-ok:                 19028731401

**My Project**

Github: <https://css-git.juniper.net/brautio/Intern_Project>

I created a custom CLI command by creating a YANG module and an action script that uses a python package I created in order to get the chip specific data for each ASIC.

The YANG module creates a custom CLI command and RPC that invokes an action script. The action script gets the data from each online FPC’s ASICs and generates XML output for the command specified in the YANG module by using HealthBot chip-agnostic YAML table/views. The YANG module then renders the XML output to the CLI.

**Setup:**

Router requirements:

* MX platform
* Junos >= 17.3R1
* Command must be on master RE

Clone repository onto local device:

git clone <https://css-git.juniper.net/brautio/Intern_Project.git>

Configure router to allow python scripts:

[edit]

root@router-re0# set system scripts language python

Easy Setup:

Install Docker:

<https://docs.docker.com/v17.09/engine/installation/>

Go into cloned repository:

cd Intern\_Project

Edit Dockerfile to include router hostname and root credentials:

In Dockerfile lines 6-8:

ENV host router-re0.hostname.net <<< Change to your router hostname

ENV username root

ENV password JuniperLab <<< Change to your root password

Build Docker Image (must be in Intern\_Project directory):

docker build -t easy\_setup .

Run Docker container:

docker run easy\_setup:latest

Now the following commands should be on the router:

root@router> show center\_chip stats | display xml

(Shows XML output)

and,

root@router> show chip-agnostic stats

(shows cli output)

Manual Setup:

The following dependencies must be on the router:

* PyEZ 2.3
* transitions
* yamlordereddictloader

Copy chip\_agnostic\_command\_package from repository onto router:

scp -r Intern\_project/chip\_agnostic\_command\_package user@remoterouter:/var/db/scripts/action

Create YANG package on router CLI:

request system yang add package chip\_agnostic module /var/db/scripts/action/ chip\_agnostic\_command\_package/chip\_agnostic\_command/yang/chip\_agnostic\_command.yang action-script /var/db/scripts/action/chip\_agnostic\_command\_package/chip\_agnostic\_command/action\_scripts/chip\_agnostic\_command.py

Run command:

show center\_chip stats

Sample CLI Output:

show chip-agnostic stats

time now is 2019-12-12 12:05:13.520804

fpc4 chip instance 0

fpc type MPC5E 3D Q 2CGE+4XGE

CChipLoStatsTable

cchip-lookup-out-errors: 0

CChipDRDErrTable

cchip-drd-wan-timeouts: 0

cchip-drd-fab-timeouts: 0

cchip-drd-fab-errors: 0

cchip-drd-wan-errors: 0

CChipFOStatsTable

cchip-fo-packets-sent: 19097303140

CChipWiStatsTable

cchip\_wi\_stall\_0: True

cchip\_wi\_stall\_1: True

cchip-total-oversubscription-drop: 0

CChipPTStatTable

cchip-free-internal-packet-table-fabric: 4096

cchip-free-internal-packet-table-wan: 4096

CChipWoStatsTable

cchip-wo-packets-sent: 15911761

CChipHostDropTable

cchip-host-path-drops: 0

CChipFiErrTable

cchip-fi-cell-timeout: 32

cchip-fi-malloc-drops: 0

cchip-fi-crc-error-packets: 0

cchip-fi-error-cells: 0

cchip-fi-late-cells: 0

CChipLiInterruptStatsTable

cchip-errors-from-lookup-chip: 0

CChipFiStatsTable

cchip-fi-packets-dropped: 0

cchip-fi-packets-with-error: 0

cchip-fi-packets-received: 19028731401

cchip-fi-packets-sent-ok: 19028731401

fpc4 chip instance 1

fpc type MPC5E 3D Q 2CGE+4XGE

CChipWiStatsTable

cchip\_wi\_stall\_0: True

cchip\_wi\_stall\_1: True

cchip-total-oversubscription-drop: 0

CChipDRDErrTable

cchip-drd-wan-timeouts: 0

cchip-drd-fab-timeouts: 0

cchip-drd-fab-errors: 0

cchip-drd-wan-errors: 0

CChipFiErrTable

cchip-fi-cell-timeout: 32

cchip-fi-malloc-drops: 0

cchip-fi-crc-error-packets: 0

cchip-fi-error-cells: 0

cchip-fi-late-cells: 0

CChipHostDropTable

cchip-host-path-drops: 0

CChipWoStatsTable

cchip-wo-packets-sent: 9473985

CChipPTStatTable

cchip-free-internal-packet-table-fabric: 4096

cchip-free-internal-packet-table-wan: 4096

CChipLoStatsTable

cchip-lookup-out-errors: 0

CChipFOStatsTable

cchip-fo-packets-sent: 19097303140

CChipLiInterruptStatsTable

cchip-errors-from-lookup-chip: 0

CChipFiStatsTable

cchip-fi-packets-dropped: 0

cchip-fi-packets-with-error: 0

cchip-fi-packets-received: 19028731401

cchip-fi-packets-sent-ok: 19028731401

fpc5 chip instance 0

fpc type MPC7E 3D 40XGE

CChipDRDErrTable

cchip-drd-wan-timeouts: 0

cchip-drd-fab-timeouts: 0

cchip-drd-fab-errors: 0

cchip-drd-wan-errors: 0

CChipLoStatsTable

cchip-lookup-out-errors: 0

CChipWoStatsTable

cchip-wo-packets-sent: 7927947

CChipFOStatsTable

plane\_0

cchip-fo-total-packets-sent: 19073232093

plane\_1

cchip-fo-total-packets-sent: 19073232093

plane\_2

cchip-fo-total-packets-sent: 19073232093

plane\_3

cchip-fo-total-packets-sent: 19073232093

plane\_4

cchip-fo-total-packets-sent: 19073232093

plane\_5

cchip-fo-total-packets-sent: 19073232093

plane\_6

cchip-fo-total-packets-sent: 19073232093

plane\_7

cchip-fo-total-packets-sent: 19073232093

plane\_8

cchip-fo-total-packets-sent: 19073232093

plane\_9

cchip-fo-total-packets-sent: 19073232093

plane\_10

cchip-fo-total-packets-sent: 19073232093

plane\_11

cchip-fo-total-packets-sent: 19073232093

plane\_12

cchip-fo-total-packets-sent: 19073232093

plane\_13

cchip-fo-total-packets-sent: 19073232093

plane\_14

cchip-fo-total-packets-sent: 19073232093

plane\_15

cchip-fo-total-packets-sent: 19073232093

plane\_16

cchip-fo-total-packets-sent: 19073232093

plane\_17

cchip-fo-total-packets-sent: 19073232093

plane\_18

cchip-fo-total-packets-sent: 19073232093

plane\_19

cchip-fo-total-packets-sent: 19073232093

plane\_20

cchip-fo-total-packets-sent: 19073232093

plane\_21

cchip-fo-total-packets-sent: 19073232093

plane\_22

cchip-fo-total-packets-sent: 19073232093

plane\_23

cchip-fo-total-packets-sent: 19073232093

CChipWiStatsTable

cchip-wi-stall cchip-wi-received-packets: 19540140

cchip-wi-stall cchip-wi-stall-0: 0

cchip-wi-stall cchip-wi-stall-1: 0

cchip-total-drop: 0

CChipPTStatTable

cchip-free-internal-packet-table-fabric cnt: 2656

cchip-free-internal-packet-table-fabric uo\_cnt: 32

cchip-free-internal-packet-table-wan cnt: 2656

cchip-free-internal-packet-table-wan uo\_cnt: 32

CChipFiErrTable

CChipLiInterruptStatsTable

cchip-errors-from-lookup-chip: 0

CChipHostDropTable

cchip-host-path-drops: 0

CChipFiStatsTable

cchip-fi-packets-dropped: 0

cchip-fi-packets-with-error: 0

cchip-fi-packets-received: 19028897968

cchip-fi-packets-sent-ok: 19028897980

fpc5 chip instance 1

fpc type MPC7E 3D 40XGE

CChipFOStatsTable

plane\_0

cchip-fo-total-packets-sent: 19067422770

plane\_1

cchip-fo-total-packets-sent: 19067422770

plane\_2

cchip-fo-total-packets-sent: 19067422770

plane\_3

cchip-fo-total-packets-sent: 19067422770

plane\_4

cchip-fo-total-packets-sent: 19067422770

plane\_5

cchip-fo-total-packets-sent: 19067422770

plane\_6

cchip-fo-total-packets-sent: 19067422770

plane\_7

cchip-fo-total-packets-sent: 19067422770

plane\_8

cchip-fo-total-packets-sent: 19067422770

plane\_9

cchip-fo-total-packets-sent: 19067422770

plane\_10

cchip-fo-total-packets-sent: 19067422770

plane\_11

cchip-fo-total-packets-sent: 19067422770

plane\_12

cchip-fo-total-packets-sent: 19067422770

plane\_13

cchip-fo-total-packets-sent: 19067422770

plane\_14

cchip-fo-total-packets-sent: 19067422770

plane\_15

cchip-fo-total-packets-sent: 19067422770

plane\_16

cchip-fo-total-packets-sent: 19067422770

plane\_17

cchip-fo-total-packets-sent: 19067422770

plane\_18

cchip-fo-total-packets-sent: 19067422770

plane\_19

cchip-fo-total-packets-sent: 19067422770

plane\_20

cchip-fo-total-packets-sent: 19067422770

plane\_21

cchip-fo-total-packets-sent: 19067422770

plane\_22

cchip-fo-total-packets-sent: 19067422770

plane\_23

cchip-fo-total-packets-sent: 19067422777

CChipLiInterruptStatsTable

cchip-errors-from-lookup-chip: 0

CChipDRDErrTable

cchip-drd-wan-timeouts: 0

cchip-drd-fab-timeouts: 0

cchip-drd-fab-errors: 0

cchip-drd-wan-errors: 0

CChipWoStatsTable

cchip-wo-packets-sent: 1894658

CChipFiErrTable

CChipLoStatsTable

cchip-lookup-out-errors: 0

CChipFiStatsTable

cchip-fi-packets-dropped: 0

cchip-fi-packets-with-error: 0

cchip-fi-packets-received: 19028918057

cchip-fi-packets-sent-ok: 19028918069

CChipPTStatTable

cchip-free-internal-packet-table-fabric cnt: 2656

cchip-free-internal-packet-table-fabric uo\_cnt: 32

cchip-free-internal-packet-table-wan cnt: 2656

cchip-free-internal-packet-table-wan uo\_cnt: 32

CChipWiStatsTable

cchip-wi-stall cchip-wi-received-packets: 13143937

cchip-wi-stall cchip-wi-stall-0: 0

cchip-wi-stall cchip-wi-stall-1: 0

cchip-total-drop: 0

CChipHostDropTable

cchip-host-path-drops:

**XML Output:**

root@lisbon-re0> show center\_chip stats | display xml

<rpc-reply xmlns:junos="http://xml.juniper.net/junos/18.3R1/junos">

<all-data>

<date>

2019-12-12 12:36:19.818696

</date>

<fpc>

<fpc-num>

fpc4

</fpc-num>

<fpc-type>

MPC5E 3D Q 2CGE+4XGE

</fpc-type>

<center-chip>

<chip-instance>

<chip-instance-num>

0

</chip-instance-num>

<tables>

<CChipDRDErrTable>

<cchip-drd-fab-errors>

cchip-drd-fab-errors:0

</cchip-drd-fab-errors>

<cchip-drd-fab-timeouts>

cchip-drd-fab-timeouts:0

</cchip-drd-fab-timeouts>

<cchip-drd-wan-errors>

cchip-drd-wan-errors:0

</cchip-drd-wan-errors>

<cchip-drd-wan-timeouts>

cchip-drd-wan-timeouts:0

</cchip-drd-wan-timeouts>

</CChipDRDErrTable>

<CChipFOStatsTable>

<cchip-fo-packets-sent>

cchip-fo-packets-sent:19115990372

</cchip-fo-packets-sent>

</CChipFOStatsTable>

<CChipFiErrTable>

<cchip-fi-cell-timeout>

cchip-fi-cell-timeout:32

</cchip-fi-cell-timeout>

<cchip-fi-crc-error-packets>

cchip-fi-crc-error-packets:0

</cchip-fi-crc-error-packets>

<cchip-fi-error-cells>

cchip-fi-error-cells:0

</cchip-fi-error-cells>

<cchip-fi-late-cells>

cchip-fi-late-cells:0

</cchip-fi-late-cells>

<cchip-fi-malloc-drops>

cchip-fi-malloc-drops:0

</cchip-fi-malloc-drops>

</CChipFiErrTable>

<CChipFiStatsTable>

<cchip-fi-packets-dropped>

cchip-fi-packets-dropped:0

</cchip-fi-packets-dropped>

<cchip-fi-packets-received>

cchip-fi-packets-received:19047351541

</cchip-fi-packets-received>

<cchip-fi-packets-sent-ok>

cchip-fi-packets-sent-ok:19047351542

</cchip-fi-packets-sent-ok>

<cchip-fi-packets-with-error>

cchip-fi-packets-with-error:0

</cchip-fi-packets-with-error>

</CChipFiStatsTable>

<CChipHostDropTable>

<cchip-host-path-drops>

cchip-host-path-drops:0

</cchip-host-path-drops>

</CChipHostDropTable>

<CChipLiInterruptStatsTable>

<cchip-errors-from-lookup-chip>

cchip-errors-from-lookup-chip:0

</cchip-errors-from-lookup-chip>

</CChipLiInterruptStatsTable>

<CChipLoStatsTable>

<cchip-lookup-out-errors>

cchip-lookup-out-errors:0

</cchip-lookup-out-errors>

</CChipLoStatsTable>

<CChipPTStatTable>

<cchip-free-internal-packet-table-fabric>

cchip-free-internal-packet-table-fabric:4096

</cchip-free-internal-packet-table-fabric>

<cchip-free-internal-packet-table-wan>

cchip-free-internal-packet-table-wan:4096

</cchip-free-internal-packet-table-wan>

</CChipPTStatTable>

<CChipWiStatsTable>

<cchip-total-oversubscription-drop>

cchip-total-oversubscription-drop:0

</cchip-total-oversubscription-drop>

<cchip\_wi\_stall\_0>

cchip\_wi\_stall\_0:True

</cchip\_wi\_stall\_0>

<cchip\_wi\_stall\_1>

cchip\_wi\_stall\_1:True

</cchip\_wi\_stall\_1>

</CChipWiStatsTable>

<CChipWoStatsTable>

<cchip-wo-packets-sent>

cchip-wo-packets-sent:15927362

</cchip-wo-packets-sent>

</CChipWoStatsTable>

</tables>

</chip-instance>

<chip-instance>

<chip-instance-num>

1

</chip-instance-num>

<tables>

<CChipDRDErrTable>

<cchip-drd-fab-errors>

cchip-drd-fab-errors:0

</cchip-drd-fab-errors>

<cchip-drd-fab-timeouts>

cchip-drd-fab-timeouts:0

</cchip-drd-fab-timeouts>

<cchip-drd-wan-errors>

cchip-drd-wan-errors:0

</cchip-drd-wan-errors>

<cchip-drd-wan-timeouts>

cchip-drd-wan-timeouts:0

</cchip-drd-wan-timeouts>

</CChipDRDErrTable>

<CChipFOStatsTable>

<cchip-fo-packets-sent>

cchip-fo-packets-sent:19116010538

</cchip-fo-packets-sent>

</CChipFOStatsTable>

<CChipFiErrTable>

<cchip-fi-cell-timeout>

cchip-fi-cell-timeout:32

</cchip-fi-cell-timeout>

<cchip-fi-crc-error-packets>

cchip-fi-crc-error-packets:0

</cchip-fi-crc-error-packets>

<cchip-fi-error-cells>

cchip-fi-error-cells:0

</cchip-fi-error-cells>

<cchip-fi-late-cells>

cchip-fi-late-cells:0

</cchip-fi-late-cells>

<cchip-fi-malloc-drops>

cchip-fi-malloc-drops:0

</cchip-fi-malloc-drops>

</CChipFiErrTable>

<CChipFiStatsTable>

<cchip-fi-packets-dropped>

cchip-fi-packets-dropped:0

</cchip-fi-packets-dropped>

<cchip-fi-packets-received>

cchip-fi-packets-received:19047371632

</cchip-fi-packets-received>

<cchip-fi-packets-sent-ok>

cchip-fi-packets-sent-ok:19047371632

</cchip-fi-packets-sent-ok>

<cchip-fi-packets-with-error>

cchip-fi-packets-with-error:0

</cchip-fi-packets-with-error>

</CChipFiStatsTable>

<CChipHostDropTable>

<cchip-host-path-drops>

cchip-host-path-drops:0

</cchip-host-path-drops>

</CChipHostDropTable>

<CChipLiInterruptStatsTable>

<cchip-errors-from-lookup-chip>

cchip-errors-from-lookup-chip:0

</cchip-errors-from-lookup-chip>

</CChipLiInterruptStatsTable>

<CChipLoStatsTable>

<cchip-lookup-out-errors>

cchip-lookup-out-errors:0

</cchip-lookup-out-errors>

</CChipLoStatsTable>

<CChipPTStatTable>

<cchip-free-internal-packet-table-fabric>

cchip-free-internal-packet-table-fabric:4096

</cchip-free-internal-packet-table-fabric>

<cchip-free-internal-packet-table-wan>

cchip-free-internal-packet-table-wan:4096

</cchip-free-internal-packet-table-wan>

</CChipPTStatTable>

<CChipWiStatsTable>

<cchip-total-oversubscription-drop>

cchip-total-oversubscription-drop:0

</cchip-total-oversubscription-drop>

<cchip\_wi\_stall\_0>

cchip\_wi\_stall\_0:True

</cchip\_wi\_stall\_0>

<cchip\_wi\_stall\_1>

cchip\_wi\_stall\_1:True

</cchip\_wi\_stall\_1>

</CChipWiStatsTable>

<CChipWoStatsTable>

<cchip-wo-packets-sent>

cchip-wo-packets-sent:9483255

</cchip-wo-packets-sent>

</CChipWoStatsTable>

</tables>

</chip-instance>

</center-chip>

</fpc>

<fpc>

<fpc-num>

fpc5

</fpc-num>

<fpc-type>

MPC7E 3D 40XGE

</fpc-type>

<center-chip>

<chip-instance>

<chip-instance-num>

0

</chip-instance-num>

<tables>

<CChipDRDErrTable>

<cchip-drd-fab-errors>

cchip-drd-fab-errors:0

</cchip-drd-fab-errors>

<cchip-drd-fab-timeouts>

cchip-drd-fab-timeouts:0

</cchip-drd-fab-timeouts>

<cchip-drd-wan-errors>

cchip-drd-wan-errors:0

</cchip-drd-wan-errors>

<cchip-drd-wan-timeouts>

cchip-drd-wan-timeouts:0

</cchip-drd-wan-timeouts>

</CChipDRDErrTable>

<CChipFOStatsTable>

<plane-0>

cchip-fo-total-packets-sent:19091907184

</plane-0>

<plane-1>

cchip-fo-total-packets-sent:19091907184

</plane-1>

<plane-2>

cchip-fo-total-packets-sent:19091907184

</plane-2>

<plane-3>

cchip-fo-total-packets-sent:19091907184

</plane-3>

<plane-4>

cchip-fo-total-packets-sent:19091907184

</plane-4>

<plane-5>

cchip-fo-total-packets-sent:19091907184

</plane-5>

<plane-6>

cchip-fo-total-packets-sent:19091907184

</plane-6>

<plane-7>

cchip-fo-total-packets-sent:19091907184

</plane-7>

<plane-8>

cchip-fo-total-packets-sent:19091907184

</plane-8>

<plane-9>

cchip-fo-total-packets-sent:19091907184

</plane-9>

<plane-10>

cchip-fo-total-packets-sent:19091907184

</plane-10>

<plane-11>

cchip-fo-total-packets-sent:19091907184

</plane-11>

<plane-12>

cchip-fo-total-packets-sent:19091907184

</plane-12>

<plane-13>

cchip-fo-total-packets-sent:19091907184

</plane-13>

<plane-14>

cchip-fo-total-packets-sent:19091907184

</plane-14>

<plane-15>

cchip-fo-total-packets-sent:19091907184

</plane-15>

<plane-16>

cchip-fo-total-packets-sent:19091907184

</plane-16>

<plane-17>

cchip-fo-total-packets-sent:19091907184

</plane-17>

<plane-18>

cchip-fo-total-packets-sent:19091907184

</plane-18>

<plane-19>

cchip-fo-total-packets-sent:19091907184

</plane-19>

<plane-20>

cchip-fo-total-packets-sent:19091907184

</plane-20>

<plane-21>

cchip-fo-total-packets-sent:19091907184

</plane-21>

<plane-22>

cchip-fo-total-packets-sent:19091907184

</plane-22>

<plane-23>

cchip-fo-total-packets-sent:19091907184

</plane-23>

</CChipFOStatsTable>

<CChipFiErrTable/>

<CChipFiStatsTable>

<cchip-fi-packets-dropped>

cchip-fi-packets-dropped:0

</cchip-fi-packets-dropped>

<cchip-fi-packets-received>

cchip-fi-packets-received:19047529654

</cchip-fi-packets-received>

<cchip-fi-packets-sent-ok>

cchip-fi-packets-sent-ok:19047529666

</cchip-fi-packets-sent-ok>

<cchip-fi-packets-with-error>

cchip-fi-packets-with-error:0

</cchip-fi-packets-with-error>

</CChipFiStatsTable>

<CChipHostDropTable>

<cchip-host-path-drops>

cchip-host-path-drops:0

</cchip-host-path-drops>

</CChipHostDropTable>

<CChipLiInterruptStatsTable>

<cchip-errors-from-lookup-chip>

cchip-errors-from-lookup-chip:0

</cchip-errors-from-lookup-chip>

</CChipLiInterruptStatsTable>

<CChipLoStatsTable>

<cchip-lookup-out-errors>

cchip-lookup-out-errors:0

</cchip-lookup-out-errors>

</CChipLoStatsTable>

<CChipPTStatTable>

<cchip-free-internal-packet-table-fabric\_cnt>

cchip-free-internal-packet-table-fabric cnt:2656

</cchip-free-internal-packet-table-fabric\_cnt>

<cchip-free-internal-packet-table-fabric\_uo\_cnt>

cchip-free-internal-packet-table-fabric uo\_cnt:32

</cchip-free-internal-packet-table-fabric\_uo\_cnt>

<cchip-free-internal-packet-table-wan\_cnt>

cchip-free-internal-packet-table-wan cnt:2656

</cchip-free-internal-packet-table-wan\_cnt>

<cchip-free-internal-packet-table-wan\_uo\_cnt>

cchip-free-internal-packet-table-wan uo\_cnt:32

</cchip-free-internal-packet-table-wan\_uo\_cnt>

</CChipPTStatTable>

<CChipWiStatsTable>

<cchip-total-drop>

cchip-total-drop:0

</cchip-total-drop>

<cchip-wi-stall\_cchip-wi-received-packets>

cchip-wi-stall cchip-wi-received-packets:19559281

</cchip-wi-stall\_cchip-wi-received-packets>

<cchip-wi-stall\_cchip-wi-stall-0>

cchip-wi-stall cchip-wi-stall-0:0

</cchip-wi-stall\_cchip-wi-stall-0>

<cchip-wi-stall\_cchip-wi-stall-1>

cchip-wi-stall cchip-wi-stall-1:0

</cchip-wi-stall\_cchip-wi-stall-1>

</CChipWiStatsTable>

<CChipWoStatsTable>

<cchip-wo-packets-sent>

cchip-wo-packets-sent:7935531

</cchip-wo-packets-sent>

</CChipWoStatsTable>

</tables>

</chip-instance>

<chip-instance>

<chip-instance-num>

1

</chip-instance-num>

<tables>

<CChipDRDErrTable>

<cchip-drd-fab-errors>

cchip-drd-fab-errors:0

</cchip-drd-fab-errors>

<cchip-drd-fab-timeouts>

cchip-drd-fab-timeouts:0

</cchip-drd-fab-timeouts>

<cchip-drd-wan-errors>

cchip-drd-wan-errors:0

</cchip-drd-wan-errors>

<cchip-drd-wan-timeouts>

cchip-drd-wan-timeouts:0

</cchip-drd-wan-timeouts>

</CChipDRDErrTable>

<CChipFOStatsTable>

<plane-0>

cchip-fo-total-packets-sent:19086082075

</plane-0>

<plane-1>

cchip-fo-total-packets-sent:19086082075

</plane-1>

<plane-2>

cchip-fo-total-packets-sent:19086082075

</plane-2>

<plane-3>

cchip-fo-total-packets-sent:19086082075

</plane-3>

<plane-4>

cchip-fo-total-packets-sent:19086082075

</plane-4>

<plane-5>

cchip-fo-total-packets-sent:19086082075

</plane-5>

<plane-6>

cchip-fo-total-packets-sent:19086082075

</plane-6>

<plane-7>

cchip-fo-total-packets-sent:19086082075

</plane-7>

<plane-8>

cchip-fo-total-packets-sent:19086082075

</plane-8>

<plane-9>

cchip-fo-total-packets-sent:19086082075

</plane-9>

<plane-10>

cchip-fo-total-packets-sent:19086082075

</plane-10>

<plane-11>

cchip-fo-total-packets-sent:19086082075

</plane-11>

<plane-12>

cchip-fo-total-packets-sent:19086082075

</plane-12>

<plane-13>

cchip-fo-total-packets-sent:19086082075

</plane-13>

<plane-14>

cchip-fo-total-packets-sent:19086082075

</plane-14>

<plane-15>

cchip-fo-total-packets-sent:19086082075

</plane-15>

<plane-16>

cchip-fo-total-packets-sent:19086082075

</plane-16>

<plane-17>

cchip-fo-total-packets-sent:19086082075

</plane-17>

<plane-18>

cchip-fo-total-packets-sent:19086082075

</plane-18>

<plane-19>

cchip-fo-total-packets-sent:19086082075

</plane-19>

<plane-20>

cchip-fo-total-packets-sent:19086082075

</plane-20>

<plane-21>

cchip-fo-total-packets-sent:19086082075

</plane-21>

<plane-22>

cchip-fo-total-packets-sent:19086082075

</plane-22>

<plane-23>

cchip-fo-total-packets-sent:19086082075

</plane-23>

</CChipFOStatsTable>

<CChipFiErrTable/>

<CChipFiStatsTable>

<cchip-fi-packets-dropped>

cchip-fi-packets-dropped:0

</cchip-fi-packets-dropped>

<cchip-fi-packets-received>

cchip-fi-packets-received:19047539699

</cchip-fi-packets-received>

<cchip-fi-packets-sent-ok>

cchip-fi-packets-sent-ok:19047539712

</cchip-fi-packets-sent-ok>

<cchip-fi-packets-with-error>

cchip-fi-packets-with-error:0

</cchip-fi-packets-with-error>

</CChipFiStatsTable>

<CChipHostDropTable>

<cchip-host-path-drops>

cchip-host-path-drops:0

</cchip-host-path-drops>

</CChipHostDropTable>

<CChipLiInterruptStatsTable>

<cchip-errors-from-lookup-chip>

cchip-errors-from-lookup-chip:0

</cchip-errors-from-lookup-chip>

</CChipLiInterruptStatsTable>

<CChipLoStatsTable>

<cchip-lookup-out-errors>

cchip-lookup-out-errors:0

</cchip-lookup-out-errors>

</CChipLoStatsTable>

<CChipPTStatTable>

<cchip-free-internal-packet-table-fabric\_cnt>

cchip-free-internal-packet-table-fabric cnt:2656

</cchip-free-internal-packet-table-fabric\_cnt>

<cchip-free-internal-packet-table-fabric\_uo\_cnt>

cchip-free-internal-packet-table-fabric uo\_cnt:32

</cchip-free-internal-packet-table-fabric\_uo\_cnt>

<cchip-free-internal-packet-table-wan\_cnt>

cchip-free-internal-packet-table-wan cnt:2656

</cchip-free-internal-packet-table-wan\_cnt>

<cchip-free-internal-packet-table-wan\_uo\_cnt>

cchip-free-internal-packet-table-wan uo\_cnt:32

</cchip-free-internal-packet-table-wan\_uo\_cnt>

</CChipPTStatTable>

<CChipWiStatsTable>

<cchip-total-drop>

cchip-total-drop:0

</cchip-total-drop>

<cchip-wi-stall\_cchip-wi-received-packets>

cchip-wi-stall cchip-wi-received-packets:13156807

</cchip-wi-stall\_cchip-wi-received-packets>

<cchip-wi-stall\_cchip-wi-stall-0>

cchip-wi-stall cchip-wi-stall-0:0

</cchip-wi-stall\_cchip-wi-stall-0>

<cchip-wi-stall\_cchip-wi-stall-1>

cchip-wi-stall cchip-wi-stall-1:0

</cchip-wi-stall\_cchip-wi-stall-1>

</CChipWiStatsTable>

<CChipWoStatsTable>

<cchip-wo-packets-sent>

cchip-wo-packets-sent:1896513

</cchip-wo-packets-sent>

</CChipWoStatsTable>

</tables>

</chip-instance>

</center-chip>

</fpc>

</all-data>

<cli>

<banner></banner>

</cli>

</rpc-reply>

**Remaining Issues:**

Junos-odl:

PR:  <https://gnats.juniper.net/web/default/1478225#description_tab>

There is an issue with the Junos-odl yang extension when used within choice/case statements in the YANG module. Therefore, there is currently no CLI output when the command is issued, only the XML.

I am trying to use junos-odl with the YANG module like this:

container CChipFOStatsTable {

choice FO-stats-xm {

case FO-stats-xm {

leaf cchip-fo-packets-sent {

type string;

}

junos-odl:format FO-xm {

junos-odl:line {

junos-odl:field "cchip-fo-packets-sent";

}

}

}

But this error happens during TLV generation:

root@thunder-re0> ...e/chip\_agnostic\_command/action\_scripts/chip\_agnostic\_command.py module /var/db/scripts/action/chip\_agnostic\_command\_package/chip\_agnostic\_command/yang/chip\_agnostic\_command.yang

YANG modules validation : START

YANG modules validation : SUCCESS

Scripts syntax validation : START

Scripts syntax validation : SUCCESS

TLV generation: START

Traceback (most recent call last):

File "/usr/libexec/ui/pyang", line 444, in

There is no further traceback given.

Execution Time and YAML files:

Execution time is very slow since the “get” method for each YAML table/view take 3-5 seconds to execute**.**

Currently the EA YAML file forCChipFiErrTable has a bug and doesn’t work.

**YANG module chip\_agnostic\_command.yang:**

The YANG module is used to specify a custom RPC and CLI command. It also specifies the action script that is run by the custom command. In order to create the custom RPC and command, the “junos-extension” YANG extension is needed for the module:

//This extenion is needed to specify a custom CLI command and action script

import junos-extension {

prefix junos;

}

In order to render XML output to the CLI, the “junos-extension-odl” YANG extension is required:

//This extension is needed to format the output from the action script when //it gets rendered back from the CLI

import junos-extension-odl {

prefix junos-odl;

}

The custom RPC is then defined by using the “RPC” keyword. Inside this block, you also specify the CLI command using “junos-extension” and the “command” statement. Inside the “command” block, you must specify and action script that is invoked by the command:

//creates RPC

rpc get\_chip\_agnostic\_stats {

description "command for CA output";

//Specify CLI command with the YANG extension prefix

junos:command "show center\_chip stats" {

//Specify action script

junos:action-execute {

junos:script "chip\_agnostic\_command.py";

}

}

[…]

In order to render data from the XML output generated by the action script, you must create an “output” block inside of the “RPC” block and define the structure of the XML correctly:

rpc get\_chip\_agnostic\_stats {

description "command for CA output";

//Specify CLI command

junos:command "show center\_chip stats" {

//Specify action script

junos:action-execute {

junos:script "chip\_agnostic\_command.py";

}

}

output {

container all-data {

leaf date {

type string;

}

list fpc {

key "fpc-num";

leaf fpc-num {

type string;

}

leaf fpc-type {

type string;

}

container center-chip {

list chip-instance {

key "chip-instance-num";

[…]

It is important to model the data output correctly in order to render the correct data to the CLI with “junos-odl”. This can be done with “leaf”, “container” and “list” statements, as in the above code snippet. For example:

The XML output for the command:

<all-data>

<date>

2019-11-21 07:51:05.797913

</date>

<fpc>

<fpc-num>

fpc4

</fpc-num>

<fpc-type>

MPC5E 3D Q 2CGE+4XGE

</fpc-type>

<center-chip>

<chip-instance>

<chip-instance-num>

0

</chip-instance-num>

<tables>

<CChipDRDErrTable>

[…]

Maps to the YANG model here:

container all-data {

leaf date {

type string;

}

list fpc {

key "fpc-num";

leaf fpc-num {

type string;

}

leaf fpc-type {

type string;

}

container center-chip {

list chip-instance {

key "chip-instance-num";

leaf chip-instance-num {

type string;

}

container tables {

container CChipDRDErrTable {

[…]

The different statements in YANG I use to model the output are well explained in RFC 7950. Once the output is modeled correctly, it is possible to use junos-odl in order to render the data modeled from each leaf to the CLI.

**Action script chip\_agnostic\_command.py:**

The action script uses chip\_agnostic\_command\_package in order to get data from the YAML files for each center chip and then structure the data into XML:

#tested when ran out of action directory

from chip\_agnostic\_command\_package.chip\_agnostic\_command import chassis\_class

from chip\_agnostic\_command\_package.chip\_agnostic\_command import xml\_builder

obj = Chassis()

xml = xml\_builder(obj)

xml.print\_xml()

The Chassis class extends the Device class and finds all of the online FPCs and the FPC’s center chips. A Chassis instance is a list of FPC objects, which represent FPCs that are online for the router. An FPC has the properties fpc\_num, which is the slot number of the FPC, fpc\_type, the model number of the FPC, and center\_chips, which is a list of center\_chip objects on the FPC. Each center\_chip object has the properties chip\_type, which represents the type of ASIC the center chip is, and chip\_instance, which represents the ASIC number. For each center chip, you can run and capture data from ten different FPC commands using the YAML files inside the yamls sub-package.

**The Chassis class:**

\_\_init\_\_:

Scripts that use this package and the Chassis class are meant to be run on-box, because Chassis class connects to the router locally.

Chassis inherits the constructor from the Device class and connects to the router locally. It then sets the \_\_online\_fpcs property:

class Chassis(Device):

def \_\_init\_\_(self):

Device.\_\_init\_\_(self)

self.\_\_set\_fpc\_online()

\_\_online\_fpcs has double underscores because it should not be referenced directly, only through the getter method get\_online\_fpcs(self) because this property should only be modified or set through \_\_set\_fpc\_online(self), which actually issues the RPC to the router in order to find the online FPCs. If this property is accessed directly, users could modify the online FPC list to potentially include FPCs that aren’t actually online, which could potentially break the objects which make up a Chassis instance.

\_\_set\_fpc\_online():

In order to set the online\_fpcs property, I used a private method which is invoked whenever a Chassis instance is created. In order to get the online FPCs, a connection to the router is opened and an RPC is issued. Then an xpath expression is used to get the online FPCs from the full RPC reply:

self.open()

op = self.rpc.get\_fpc\_information()

online = op.xpath('./fpc[normalize-space(state) = "Online"]/slot')

The online FPCs are then added to a list, which is used to create FPC objects. This list of FPC objects is then used for the \_\_online\_fpcs property, and then the connection is closed:

for i in online:

if i.text is not None:

online\_fpcs.append('fpc'+i.text)

chassis\_components = []

for online\_fpc in online\_fpcs:

chassis\_fpc = FPC(online\_fpc)

chassis\_components.append(chassis\_fpc)

self.\_\_online\_fpcs = chassis\_components

self.close()

**The FPC Class:**

An FPC object represents the online FPC inside the chassis.

\_\_init\_\_:

An FPC instance has three properties: fpc\_num, fpc\_type, and center\_chips. fpc\_num is the slot number of the FPC, which is passed to \_\_init\_\_ from the Chassis class in \_\_set\_fpc\_online. fpc\_type is the FPC model number which is computed using the \_\_set\_fpc\_type method. center\_chips is a list of center\_chip objects which represent the center chip ASICs for the FPC.

An FPC is only initialized if it is online, which is checked upon initialization of an FPC instance using the \_\_get\_fpcs\_online static method, which is similar to the \_\_set\_fpc\_online method in Chassis, except it returns the full list of online FPCs:

def \_\_init\_\_(self, fpc\_num):

if fpc\_num in self.\_\_get\_fpcs\_online():

self.fpc\_num = fpc\_num

self.\_\_set\_fpc\_type()

self.\_\_set\_center\_chips()

else:

print "FPC isn't online"

raise ValueError("This FPC is not online")

\_\_set\_fpc\_type:

This method sets the fpc\_type property. It does this using the get\_chassis\_inventory rpc, and then using an XPath expression to navigate the RPC reply to find the FPC’s model number:

def \_\_set\_fpc\_type(self):

dev = Device()

dev.open()

#If xpath 1.0, must do lower case like this. If 2.0, can use lower-case()

op = dev.rpc.get\_chassis\_inventory()

fpc = '"{}"'.format(self.fpc\_num)

desc = op.xpath('//chassis-module[translate(translate\

(normalize-space(name), "ABCDEFGHIJKLMNOPQRSTUVWXYZ", \

"abcdefghijklmnopqrstuvwxyz"), " ","") = '+fpc+']/description')

for i in desc:

if i.text is not None:

fpc\_type = i.text

self.fpc\_type = fpc\_type

dev.close()

The XPath expression uses translate first to parse the FPC slot number as lower-case, and then to get rid of the spaces. Then the way that we have already formatted the string stored in the fpc\_num property can be used to find the correct FPC model number.

We must format the “fpc” string used in the XPath expression this way because there must be a string on the other side of the equal comparison, which means that it must be inside double quotes.

\_\_set\_center\_chips:

This method sets the center\_chips property. It uses a dictionary that is returned from the chip\_types\_dict function inside \_\_chip\_types module. This dictionary maps FPC models to their center chip types, so we can use the FPC model number to find the FPC’s center chip types.

In order to get the FPC model, the same RPC and XPath is used as in \_\_set\_fpc\_type:

def \_\_set\_center\_chips(self):

dev = Device()

dev.open()

chips = chip\_types\_dict()

op = dev.rpc.get\_chassis\_inventory()

fpc = '"{}"'.format(self.fpc\_num)

desc = op.xpath('//chassis-module[translate(translate\

(normalize-space(name), "ABCDEFGHIJKLMNOPQRSTUVWXYZ", \

"abcdefghijklmnopqrstuvwxyz"), " ","") = '+fpc+']/description')

for i in desc:

if i.text is not None:

fpc\_type = i.text

center\_chip\_type = chips["ChipTypeMap"][fpc\_type]["center\_chip"]

[…]

Next, a loop is used in order to create the proper center chip instances for the FPC using the center\_chip\_type dictionary. The amount of center\_chip objects created for the FPC depends on the amount of chip instances there are, which is also inside the dictionary. The property fpc\_num, the ASIC type (key), and chip instance (i) are needed to create a center chip instance:

for key, val in center\_chip\_type.items():

for i in range(val):

chip\_obj = center\_chip(self.fpc\_num, key, i)

list\_of\_center\_chips.append(chip\_obj)

self.center\_chips = list\_of\_center\_chips

dev.close()

**The center\_chip Class:**

Instances of center\_chip are created by an FPC instance in order to represent that FPC’s center chips.

\_\_init\_\_:

A center\_chip instance has three properties: \_\_fpc\_num, chip\_type, and chip\_instance. They are all passed to the constructor by an FPC instance in the \_\_set\_center\_chips method. \_\_fpc\_num is the slot number of the FPC which the center chip is on. This property is only required to run FPC commands with the chip agnostic YAML files. I made the property private since it is not an actual trait of the center chip, it is only required to run the chip agnostic commands. chip\_type is a string representing the ASIC type of the chip. Currently only the XM and MQSS types are supported by the chip agnostic commands/YAMLS. If these commands are run with a chip type that isn’t XM or MQSS, the command will return NONE. chip\_instance is an integer representing a specific center chip on the FPC since there is often multiple center chips. This is also needed to run the chip agnostic commands/YAMLs since one of the arguments that must be passed to the YAML file for the command to be run is the chip instance:

def \_\_init\_\_(self, fpc\_num, chip\_type, chip\_instance):

self.\_\_fpc\_num = fpc\_num

self.chip\_type = chip\_type

self.chip\_instance = chip\_instance

Chip Agnostic Commands/YAMLs:

There are 10 chip agnostic command that can be run on the center chips through these 10 methods:

get\_CChipLiInterruptStats

get\_CChipWiStats

get\_CChipWoStats

get\_CChipFOStats

get\_CChipFiStats

get\_CChipFiErr (only the xm YAML works)

get\_CChipLoStats

get\_CChipHostDrop

get\_CChipDRDErr

get\_CChipPTStat

These commands are run by using YAML files. The YAML files use table/views to define the command to be run on the FPC and to capture the chip agnostic data. Inside the table, the command that is run in the FPC shell is defined by the “command” statement. The FPC and chip instance must be passed to the YAML as well. These are defined by the “target” statement and the “args”/”instance” statements, which define the FPC and chip instance to run the command on, respectively. The view and table can then be used to capture the chip agnostic data using various methods depending on the command’s CLI output.

Each chip type has a separate YAML file to run the command and capture data from the CLI. All of the YAML files can be found in the sub-package yamls.

Starting in PyEZ 2.3.1 all of the chip agnostic YAML files can be imported directly from this package. However, for this package, in order to import the YAML tables to run the commands, we had to make python modules that have the same name as the YAML files we want to import. All of these modules have either these four lines of code (1):

from jnpr.junos.factory import loadyaml

from os.path import splitext

\_YAML\_ = splitext(\_\_file\_\_)[0] + '.yml'

globals().update(loadyaml(\_YAML\_))

Or these lines (2):

import yaml

import yamlordereddictloader

from jnpr.junos.factory.factory\_loader import FactoryLoader

from jnpr.junos.factory import loadyaml

from os.path import splitext

\_YAML\_ = splitext(\_\_file\_\_)[0] + '.yml'

globals().update(FactoryLoader().load(yaml.load(open(\_YAML\_),\

Loader=yamlordereddictloader.Loader)))

Both of these variations are basically loading the corresponding YAML file and importing the definitions into the global namespace.

When we import all of the YAMLs in the beginning of the chassis\_class module like this:

from yamls.xm\_li\_error import CChipLiInterruptStatsTable as xm\_li

from yamls.xm\_wi\_statistics import CChipWiStatsTable as xm\_wi

from yamls.xm\_wo\_statistics import CChipWoStatsTable as xm\_wo

from yamls.xm\_fo\_statistics import CChipFOStatsTable as xm\_fo

[…]

The python module with the same name as the YAML file being imported runs one of the two variations of loading the YAML file as shown above. It is important that the import is aliased with a unique name because the chip agnostic YAMLs have the same table names for different chip types.

The methods in the center\_chip class that run the chip agnostic commands must first figure out which chip type the center chip is in order to use the correct YAML file. All of the chip agnostic command methods check this using the chip\_type attribute. For Example:

def get\_CChipFiErr(self):

with Device() as dev:

if self.chip\_type == "xm":

with open(path+"/xm\_fi\_error.yml", "r") as stream:

xm\_fi\_error = xm\_fi\_err(dev).get(target = self.\_\_fpc\_num , args = {'chip\_instance': self.chip\_instance})

xm\_fi\_error\_data\_dict = parse\_command\_output(xm\_fi\_error, stream)

return {self.\_\_table\_name(xm\_fi\_error):xm\_fi\_error\_data\_dict}

elif self.chip\_type == "mqss":

with open(path+"/ea\_fi\_error.yml", "r") as stream:

ea\_fi\_error = ea\_fi\_err(dev).get(target = self.\_\_fpc\_num , args = {'chip\_instance': self.chip\_instance})

ea\_fi\_error\_data\_dict = parse\_command\_output(ea\_fi\_error, stream)

return {self.\_\_table\_name(ea\_fi\_error):ea\_fi\_error\_data\_dict}

else:

print "This chip type does not have a YAML file for the command"

return None

Once the correct YAML file is selected for the chip, the YAML file is opened for reading, and then the get method is ran on the YAML table in order to run the command and fetch data from the view. This returns a dictionary of data that is then further parsed with the opened YAML file object (stream) by the parse\_command\_output function from the \_\_parse\_cchip\_command\_output module. This function then returns the relevant chip agnostic data from the command.

Once the data is returned from parse\_command\_output, the chip agnostic command methods return a dictionary with the key being the table name and the value being a dictionary returned from parse\_command\_output.

Note: Only EA and XM chip types are supported for the chip agnostic commands/YAMLS.

The \_\_parse\_cchip\_command\_output module:

This module is used to parse through the dictionary of data returned from the get method on the imported YAML tables. The Chassis class uses the function parse\_command\_output in order to parse the relevant data.

Example:

def get\_CChipLiInterruptStats(self):

with Device() as dev:

if self.chip\_type == "xm":

with open(path+"/xm\_li\_error.yml", "r") as stream:

xm\_li\_stats = xm\_li(dev).get(target = self.\_\_fpc\_num , args = {'chip\_instance': self.chip\_instance})

xm\_li\_stats\_data\_dict = parse\_command\_output(xm\_li\_stats, stream)

return {self.\_\_table\_name(xm\_li\_stats):xm\_li\_stats\_data\_dict}

[…]

The parse\_command\_output function takes the YAML data and the opened YAML file to parse the YAML data for chip agnostic data (any counter labeled with “cchip”) and to run the python code within “eval” statement, if present, within the YAML file, since that is also chip agnostic data.

Parsing the YAML data within the parse\_command\_output function can be broken up into two parts. The first step is to find all of the chip agnostic data within the dictionary that doesn’t need any more processing.

In order to accomplish this step, we must iterate through the dictionary returned by the get method on the YAML table. This is a little tricky, because this dictionary can contain nested dictionaries or just single key value pairs. Below is a list of the different ways each element in the dictionary of data returned from the YAML table is structured:

1. counter\_label: num
2. counter\_label: num, (tuple\_label1, tuple\_num1): {dict\_label1: dict\_val1, […] }, […]
3. Same structure as 2, but key for nested dictionaries can be an integer or string rather than a tuple.

Due to the various ways data is stored in these dictionaries, their structure must be tested before they are parsed through. Also, we only need the chip agnostic data to display for our command. In order to parse through the dictionary correctly, I first test for nested dictionaries. Then I parse through these dictionaries to find the chip agnostic data by only saving counter names that have “cchip” in them. Here is the part of parse\_command\_output that accomplishes this task:

[…]

for item in stats:

#checks for case 2 or 3

if isinstance(item[1],dict):

if "cchip" in str(item[0]):

[…]

else:

for ea\_fo\_key, ea\_fo\_val in item[1].items():

if "cchip" in str(ea\_fo\_key):

[…]

for key,val in item[1].items():

if "cchip" in str(item[0]):

[…]

#checks for case 1

else:

eval\_key\_list = \_\_find\_eval\_keys(d)

if eval\_key\_list:

if item[0] not in str(eval\_key\_list):

if "cchip" in str(item[0]):

[…]

else:

if "cchip" in str(item[0]):

[…]

[…]

The second step is to check for the “eval” statement in the YAML file.

Example:

In lines 10-11 in ea\_lo\_statistics.yml:

eval:

cchip-lookup-out-errors: >

sum([v['total'] for k,v in {{ data }}.items()])

If there is an eval statement, we must take the corresponding code in the eval and run it within the module in our global namespace to get the chip agnostic data. This requires some reformatting of the code string from the original and altering the data that is captured from the YAML, which is done with several helper functions.

The first helper function used by parse\_command\_output is \_\_get\_eval\_dict. This function takes the data captured from the YAML and returns a dictionary of the data that allows the eval code to run in our module. It does this by checking the data type of each value in the dictionary. If the value is an instance of dictionary, it adds the corresponding key-value pair to a dictionary. If the value isn’t a dictionary, it adds the key-value pair to a different dictionary. Once this is done, the function checks if the dictionary with values that are instances of dictionaries is nonempty. If this is true, it returns that dictionary. If this is false, it returns the dictionary with the entries that have values which are not instances of dictionary.

def \_\_get\_eval\_dict(stats):

evalDicts1 = {}

evalDicts2 = {}

for k,v in stats:

if isinstance(v, dict):

evalDicts1.update({k: v})

else:

evalDicts2.update({str(k): str(v)})

if evalDicts1:

return evalDicts1

else:

return evalDicts2

This sorting of that data captured by the YAML must be done due to the two different types of eval code. One type operates on the nested dictionaries in the values of the entries. For example, in ea\_lo\_statistics.yml:

eval:

cchip-lookup-out-errors: >

sum([v['total'] for k,v in {{ data }}.items()])

Since the code in eval expects each value to be a dictionary, we must get rid of all elements that don’t have dictionaries. The data returned can sometimes contain values that aren’t dictionaries, and this function guarantees each value the eval code operates on will be a dictionary.

The second type of eval statement uses Jinja formatting in order to use variables defined in the YAML within the eval code. For example, in xm\_pt\_statistics.yml:

CChipPTStatView:

regex:

pct\_wi\_1: 'PCT entries used by all WI-1 streams\s+:\s?(\d+)'

pct\_wi\_0: 'PCT entries used by all WI-0 streams\s+:\s?(\d+)'

pct\_fab: 'PCT entries used by all FI streams\s+:\s?(\d+)'

eval:

cchip-free-internal-packet-table-wan: '4096 - {{ pct\_wi\_1 }} + {{ pct\_wi\_0 }}'

cchip-free-internal-packet-table-fabric: '4096 - {{ pct\_fab }}'

The process of converting the Jinja formatting into variables defined in our namespace will be discussed shortly. In this case, the variables needed for the eval will be sorted into evalDicts2 in the \_\_get\_eval\_dicts function, which will be returned in this case since there aren’t values in these cases of evals which are instances of dictionary. We use the variables defined in this dictionary to replace the Jinja formatting in the eval code.

After the data returned from the YAML is parsed correctly, the eval code from the YAML file must be captured and altered to run in our namespace. This is done with the \_\_find\_eval function. This function takes the loaded YAML file and the parsed data from returned from \_\_get\_eval\_dict, and returns a list of strings containing the code that can be run in our namespace in order to compute the evals.

Once the eval statement is found within the YAML files, the code from the statement can be stored and altered in order to run in our namespace. In order for the code to work in our namespace, we must change the Jinja formatting to variables storing the same data that are defined in our namespace.

In our case, if the eval code contains “{{ data }}”, it is replaced with “eval\_counters”, which contains the same data that “{{ data }}” would capture. If the eval code doesn’t have “{{ data }}” in it, it replaces any Jinja formatting that is found within the code with corresponding variables that are captured from the YAML file, which is stored in the variable “eval\_counters”.

For example:

1. “sum([v['total'] for k,v in {{ data }}.items()])” becomes “sum([v['total'] for k,v in eval\_counters.items()])”
2. If “pct\_wi\_1”: 0 and “pct\_wi\_0”: 1 are entries within eval\_counters:

“4096 - {{ pct\_wi\_1 }} + {{ pct\_wi\_0 }}” becomes “4096 - 0 + 1”

Once the list of code is returned from the \_\_find\_eval function, the function parse\_command\_output iterates through the list and executes the code using the exec function, and adds the data to the dictionary that is returned by parse\_command\_output:

exec "num="+code in globals(), locals()

eval\_label = str(eval\_label)+":"

eval\_result = str(num)

table\_counters.update({eval\_label:eval\_result})

\_\_table\_name:

This method formats the table name by getting rid of all of the text after the colon and returns this string.

**The xml\_builder class:**

This class creates the XML output for the YANG model from a Chassis instance. It uses lxml to create the XML.

\_\_init\_\_:

To create an instance of xml\_builder, an instance of Chassis must be passed to the constructor.

An xml\_builder instance has two properties. The chassis\_isntance property is the Chassis instance that was used to create the instance of xml\_builder. The \_\_xml property is created by the \_\_create\_xml method. It stores the root node of the XML created from the chip agnostic command methods from the Chassis instance passed to the constructor.

def \_\_init\_\_(self, chassis\_instance):

if isinstance(chassis\_instance, Chassis):

self.chassis\_instance = chassis\_instance

self.\_\_create\_xml()

else:

raise ValueError("Must pass instance of Chassis to xml\_builder")

If the constructor isn’t passed an instance of Chassis, it raises a ValueError.

\_\_create\_xml:

This method is called in the constructor and sets the \_\_xml property. It builds the XML that is used by the YANG module using lxml. It does this by iterating through the online FPCs in the Chassis instance, and then iterating through each FPC’s center chips and running the chip agnostic commands on each center chip:

def \_\_create\_xml(self):

top = etree.Element('all-data')

self.\_\_xml = top

[…]

for fpc in self.chassis\_instance.get\_online\_fpcs():

[…]

center\_chip = etree.SubElement(fpc\_node, "center-chip")

[…]

#run all cchip commands

self.\_\_build\_chip\_data\_xml(tables\_container, chip.get\_CChipLiInterruptStats())

self.\_\_build\_chip\_data\_xml(tables\_container, chip.get\_CChipWiStats())

self.\_\_build\_chip\_data\_xml(tables\_container, chip.get\_CChipWoStats())

self.\_\_build\_chip\_data\_xml(tables\_container, chip.get\_CChipFOStats())

self.\_\_build\_chip\_data\_xml(tables\_container, chip.get\_CChipFiStats())

self.\_\_build\_chip\_data\_xml(tables\_container, chip.get\_CChipFiErr())

self.\_\_build\_chip\_data\_xml(tables\_container, chip.get\_CChipLoStats())

self.\_\_build\_chip\_data\_xml(tables\_container, chip.get\_CChipHostDrop())

self.\_\_build\_chip\_data\_xml(tables\_container, chip.get\_CChipDRDErr())

self.\_\_build\_chip\_data\_xml(tables\_container, chip.get\_CChipPTStat())

\_\_build\_chip\_data\_xml:

This method takes a SubElement object and the dictionary of data from the center chip and adds that data into more child elements under the SubElement it was passed.

It does this by first making a child element under the SubElement \_\_build\_chip\_data\_xml was passed. This child element holds all of the data for the particular YAML table it is currently parsing through. The child element’s name is the name of the YAML table. This is shown in line 69 of the XML builder module:

table\_name = etree.SubElement(tables\_container, table)

After this child node is created, the method iterates through the dictionary and adds all of the data as sub elements to table\_name:

if isinstance(counter\_data, dict):

plane = "plane\_"+str(counter\_label)

for packets\_label, packets\_num in counter\_data.items():

plane\_node = etree.SubElement(table\_name, plane)

packets\_node = etree.SubElement(plane\_node, packets\_label.replace(":",""))

packets\_node.text = packets\_label+str(packets\_num)

else:

str\_for\_counter\_name = counter\_label.split(":", 1)[0].replace(" ", "\_")

counter\_node = etree.SubElement(table\_name, str\_for\_counter\_name)

counter\_node.text = str\_for\_counter\_name + ":" + str(counter\_data)

The if statement is needed due to a special case in how data from ea\_fo\_statistics.yml is formatted. All of the other data captured is handled by the code inside the else statement.

print\_xml:

This method prints the full XML document created by xml\_builder. Ideally the method would convert the XML into a string with the tostring method and print, but the XML is usually too long and runs into a buffer limit in cscript. The workaround is to use the etree dump method, which should usually only be used for debugging, but works to print the XML for our usage:

def print\_xml(self):

etree.dump(self.\_\_xml)

update\_xml:

Updates the data in XML by running all chip agnostic commands again:

def update\_xml(self):

self.\_\_create\_xml()

**Acknowledgements**

Pasvorn Boonmark

Aditya Mahale

Devang Patel

Nitin Kumar