Dell Networking

OS10 Open Edition Administration and Programmability

The power to do more

TABLE OF CONTENTS

Front Matter

Copyright

2016-03

Rev. A01

Contributors

The Dell Networking team took guidance from the Book Sprints team (www.booksprints.net) and created the OS10 Open Edition User Guide in five days. On Day One, the team had no idea that they would have the capability to write and edit a comprehensive document by the end of Day Five. The team assembled here worked with amazing coordination and dedication to make this happen.

The contributors (in alphabetical order):

Devireddy, Srideep
Goodall, Darius
Kaludjerovic, Stevan
Lazar, Mike
Marisetty, Anitha
Mishra, Abhishek
Moopath Velayudhan, Mukesh
Mynam, Satish
Myneni, Bala
Narlajarla, Pratima
Persh, Howard
Planting, Laura
Raiyani, Viraj
Santhanam, Madhusudhanan
Schwartzberg, Alan
Sunny, Prince
Wichmann, Clifford

Book Sprints Team:

Illustrations and Cover Design: Henrik Van Leeuwen
HTML Book Production: Julien Taquet
Text Clean-Up: Raewyn Whyte
Tech Support: Juan Gutierrez
Book Sprint Facilitation: Barbara Rühling

Intended Audience

This document is intended for System Administrators and Developers with a technical background in:

- Advanced Linux
- Standard and Open Linux utilities
- Data Modelling using YANG
- Python and C/C++ Languages

The information in this document allows you to install, operate, administer, and develop applications for the Dell Networking Operating System 10 (OS10) Open Edition (OE).

How This Document is Organized

This document is organized as follows:

- **What is OS10?** provides a brief description of the functionality provided by OS10.
- **Architecture** describes the components and services included in OS10, including an overview of the run-time organization.
- **Installation** describes the installation procedures (manual and automatic).
- **System Administration** covers the tasks needed to properly maintain and upgrade a platform running OS10.
- **Networking Features** describes the operation and configuration of the various networking features the OS10 supports.
- **Programmability** provides an introduction to Control Plane Services (CPS) and YANG Models.
- **Troubleshooting** covers troubleshooting and debugging procedures.

The document provides use cases for:

- Orchestration Using Puppet (in **System Administration**)
- Monitoring Using Nagios (in **System Administration**)
- BGP Routing Using Quagga (in **Networking Features**)
- CAM Optimization using CPS APIs (in **Networking Features**)

All readers of this document should start by reading the **Product Overview**.

System administrators may want to continue with the sections **Installation**, **System Administration, Networking Features**, and the respective use cases.

DevOps engineers or software developers can go directly to the section **Programmability**.

All users should refer to the **Troubleshooting** section for troubleshooting and debugging procedures.

Note: In this document, OS10 refers to the Dell Networking OS10 Open Edition.

Dell Networking Support

The OS10 Open Edition software is supported only on a Dell ONIE-enabled switch. For a complete list of supported Dell switches, see http://www.dell.com/support. Dell does not provide support for third-party software/drivers, community projects, code development or implementation and development of security rules/policies.

OS10 Terminology

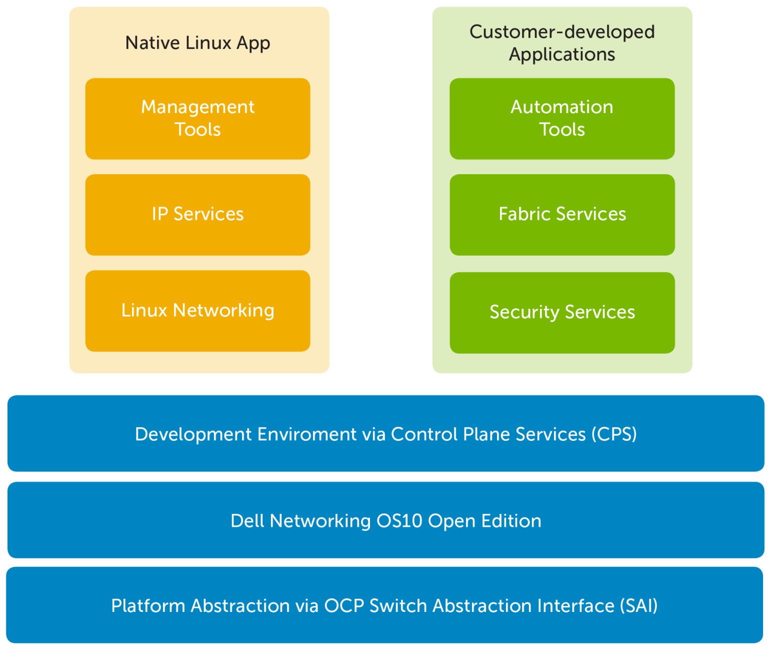
ACL	Access Control List
API	Application Programming Interface
CPS	Control Plane Service
EEPROM	Electrically Erasable Programmable Read-Only Memory
NAS	Network Adaptation Service
NPU	Network Processing Unit
NRPE	Nagios Remote Plugin Executor server
ONIE	Open Network Install Environment
PAS	Platform Adaptation Service
QSFP	Quad Small Form-factor Pluggable
QoS	Quality of Service
SDI	System Device Interface
SAI	Switch Abstract Interface
Physical port	System physical port (typically front-panel port)
Linux interface	Linux network device mapped to physical port
LAG port	Link Aggregation Group port in NPU
Bond interface	Link Aggregation Group interface in Linux

Product Overview

What is the Dell OS10 Open Edition?

The Dell Networking Operating System 10 (OS10) Open Edition (OE) is an innovative operating system for Dell Networking systems. This document describes how OS10 enables you to unleash new and creative ways to deploy, orchestrate, and manage your networking, servers and storage solutions in your data centers and enterprise environments. OS10 uses an unmodified Linux kernel and standard Linux distribution to take advantage of the rich Linux ecosystem and provide flexibility in customizing OS10 according to your networking needs.

Figure 1 – Dell OS10 Open Edition


Native Linux App
Management Tools
IP Services
Linux Networking
Customer-developed Applications
Automation Tools
Fabric Services
Security Services
Development Enviroment via Control Plane Services (CPS)
Dell Networking OS10 Open Edition
Platform Abstraction via OCP Switch Abstraction Interface (SAI)

Supported Hardware Platforms

OS10 version 10.1.0B is supported on the following Dell Networking systems:

- S3048-ON
- S4048-ON

- S6000-ON

For more information about a supported switch, refer to the hardware documentation at www.dell.com/support: display the Product Support page for a switch (View Products->Servers, Storage & Networking->Networking->Fixed Port Switches) and click the Manuals tab in the menu bar at the top of the page.

Standard Linux

OS10 is implemented using a standard Linux distribution (Debian Jessie). OS10 is binary-compatible with Debian Linux packages. Compared to deploying a vendor-built Linux distribution, the OS10 approach provides important advantages:

- You can install any Debian package from a standard Debian repository without having to rebuild the package.
- You can develop applications using the standard Debian development environment. You are not constrained by or locked into a hardware vendor-specific development environment.
- You can rapidly deploy updates to Debian software packages (such as security patches) as soon as they become available.
- You can securely upgrade OS10 using standard Linux tools (such as `apt-get` and `dpkg`) and repositories.

Linux Kernel

OS10 uses the unmodified Linux kernel Version 3.16. This kernel provides a robust base to support current state-of-the-art and future networking features.

Linux IP Stack

OS10 uses the standard Linux IP stack without vendor-specific changes. This implementation allows customers to take advantage of the rich feature set provided by the Linux standard IP stack.

Linux Tools

All standard Linux system administration tools are factory-installed in the OS10 or can be easily installed from standard Debian repositories.

Convergence of Networking, Servers, and Storage

The use of Linux as an operating system provides a solid foundation for the convergence of networking, server, and storage solutions. OS10 allows you to easily deploy the management and orchestration solutions that are typically available for Linux servers and storage systems.

OS10 Programmability

OS10 provides an object-centric API for application development. OS10 allows you to implement your own applications using a well-defined object model and set of programmatic APIs. The object model is defined using the YANG modeling language. OS10 APIs support C/C++ and Python programming languages.

A set of standard Debian software development packages is provided to allow you to develop applications for the OS10.

Open Platform Abstraction

OS10 implements a new, open object-centric application-programming interface called the Control Plane Services (CPS) API. The CPS API allows customer-developed applications to be completely independent of any underlying hardware or software technology.

In addition, OS10 internally uses the Switch Abstraction Interface (SAI), which Dell and partner companies contributed to the Open Compute Project. The SAI API allows the OS10 to be completely independent of any network processor/switch hardware technology. For more information about the SAI, see http://www.opencompute.org/wiki/Networking/SpecsAndDesigns#Switch_Abstraction_Interface.

System Hardware Integration with Standard Linux APIs

OS10 integrates standard Linux networking APIs with the hardware functionality provided by networking devices (systems and network processors). You can download and use open source software (such as Quagga) **off-the-shelf** on any OS10 platform.

Disaggregated Hardware and Software

OS10 provides an environment in which hardware and software are fully modular. You can select the software modules you want to install, as well as the hardware platforms you want to use for your networking needs.

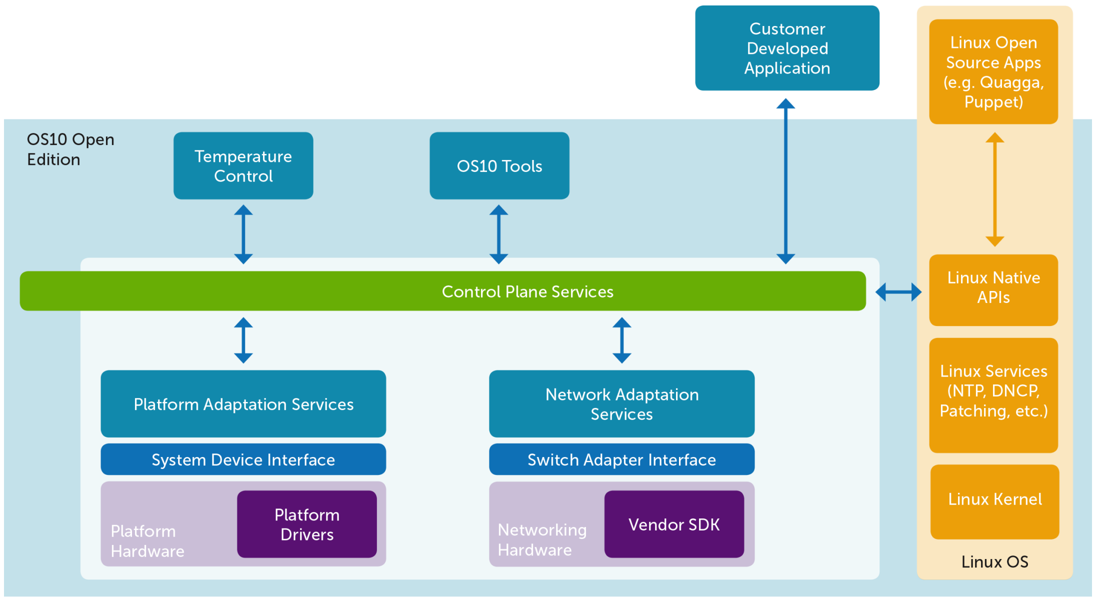
Architecture

Components and Services

The main OS10 components are:

- Linux infrastructure
- Control Plane Services (CPS)
- Switch Abstraction Interface (SAI)
- Network Adaptation Service (NAS)
- System Device Interface (SDI)
- Platform Adaptation Service (PAS)
- Dell applications and tools

Figure 2 – OS10 Architecture



Each OS10 component implements a set of well-defined APIs, resulting in full software modularity with hardware and software platform abstraction.

The OS10 architecture implements:

- **Software partitioning** – OS10 software is partitioned into sub-components that are organized as Linux packages. Each Linux package contains only related functionality.
- **Software layering** – System components depend only on the components that logically support them.
- **Hardware and software platform abstraction** – The SDI, SAI modules and the platform startup scripts are the only hardware-specific components in OS10. All the other modules are hardware-independent.
  Hardware-specific variations (such as the number and names of physical ports, and the number of power supplies) are defined using platform definition files.
  OS10 uses POSIX APIs. When necessary, implementation-specific details are abstracted using the OS10 run-time libraries (common utilities and logging).
- **Open Application Programming Interface** – User applications interact with the OS10 software modules using the CPS API. OS10 provides an object-centric API in the CPS component.

Linux Infrastructure

The Linux infrastructure consists of a collection of Linux services, libraries, and utilities pre-installed in an OS10 image. Together with the Linux kernel, these Linux components provide the foundation for the implementation of OS10-specific software components.

Control Plane Services

Control Plane Services (CPS) are at the core of the OS10 architecture. CPS provides an object-centric framework that mediates interactions between OS10 applications and allows customer applications to interact with OS10 software components.

CPS defines two types of application roles: clients and servers.

The CPS framework allows CPS **client applications** to execute create, set, get, and delete operations on individual objects or lists of objects. CPS **server applications** execute operations requested by CPS client applications. Because client applications operate on objects, they do not need to be aware of the location or name of the CPS server application that executes a requested operation.

In addition, the CPS framework supports a publisher/subscriber model. CPS server applications publish relevant events; client applications can subscribe (register) for specific events and objects. CPS client applications can register for events generated when objects are created, modified, or deleted.

The publisher/subscriber approach and object-centric operations allow for the completely independent operation of client and server applications.

Custom-written applications use the CPS API to communicate with the OS10 components.

CPS API and the OS10 Object Model

The object model provided by the CPS layer is defined using YANG files. The YANG models are used to generate C header files, which provide a programmatic representation of objects and their attributes. The header files are shared between client and server applications. The OS10 C/C++ representation of objects and their attributes is designed to ensure compatibility between multiple versions of the object model.

OS10 provides both C/C++ and Python programming interfaces for CPS.

Switch Abstraction Interface

The OS10 Switch Abstraction Interface (SAI) implements an API for NPUs supported on Dell systems. The SAI API is an open interface that abstracts vendor-specific NPU behavior. The SAI API is the result of a joint effort of multiple NPU vendors and user companies, who contributed the SAI to the Open Compute Platform.

OS10 is, therefore, NPU-independent and not locked to specific NPU hardware. If a new NPU is used in an OS10 platform, the only OS10 component that you need to replace is the SAI.

Network Adaptation Service

The Network Adaptation Service (NAS) manages the high-level network processor (NPU) abstraction and adaptation. The NAS abstracts and aggregates the core functionality required for networking access at Layer 1 (physical layer), Layer 2 (VLAN, link aggregation), Layer 3 (routing), ACL, QoS and network monitoring (mirroring and sFlow).

The NAS enables adaptation of the low-level switch abstraction provided by the Switch Abstraction Interface to:

- Standard Linux networking APIs and Linux Interfaces
- OS10-specific CPS API functionality.

The NAS manages the middleware that associates physical ports to Linux interfaces. In addition, the NAS is responsible for providing packet I/O services, using the Linux kernel IP stack.

System Device Interface

In OS10 a system device refers to a hardware component, such as:

- Fans/cooling devices
- Power supplies
- Temperature sensors
- LEDs
- EEPROM
- Programmable devices.
- Transceivers

All hardware components except for NPUs are abstracted as **system devices**.

The SDI API defines a low-level platform-independent abstraction for all types of system devices. Therefore, only system device drivers which implement the SDI API are hardware-specific, while the API itself is hardware-independent.

Platform Adaptation Service

The Platform Adaptation Service (PAS) provides a higher-level abstraction and aggregation of the functionality provided by the SDI component. In addition, the PAS implements the CPS object models associated with system devices. For example, the PAS CPS API allows user applications to:

- Read current temperature values reported by temperature sensors.
- Get and set fan speed values.
- Set a LED state.
- Read power levels reported by PSUs.

- Get system inventory and EEPROM information.
- Set transceiver module state (for example, Tx laser on/off) and get module information.

The PAS implements the following functionality:

- Detection of common equipment FRU (PSUs, fans) insertion and removal events
- Detection of **over-temperature** events for pre-defined temperature thresholds
- Detection of media insertion on physical ports.

In summary, PAS is responsible for:

- Monitoring the status of system devices.
- Reporting status changes or faults as CPS events.
- Allowing applications to retrieve current status information.
- Allowing applications to set the control variables of system devices.

Platform Description Infrastructure

The platform description infrastructure provides a means to describe specific per-platform configuration parameters, such as the number of ports per system, supported transceiver modules, mapping of Linux interfaces to physical ports, and number of fans and power supply units.

In addition, this component contains the platform-specific system startup configuration and scripts.

Dell Applications and Tools

OS10 provides a set of tools and commands that allow system administrators to access Dell-specific software and hardware functionality, such as software upgrades, physical port, media information, and system inventory. These OS10 tools are described in the following sections.

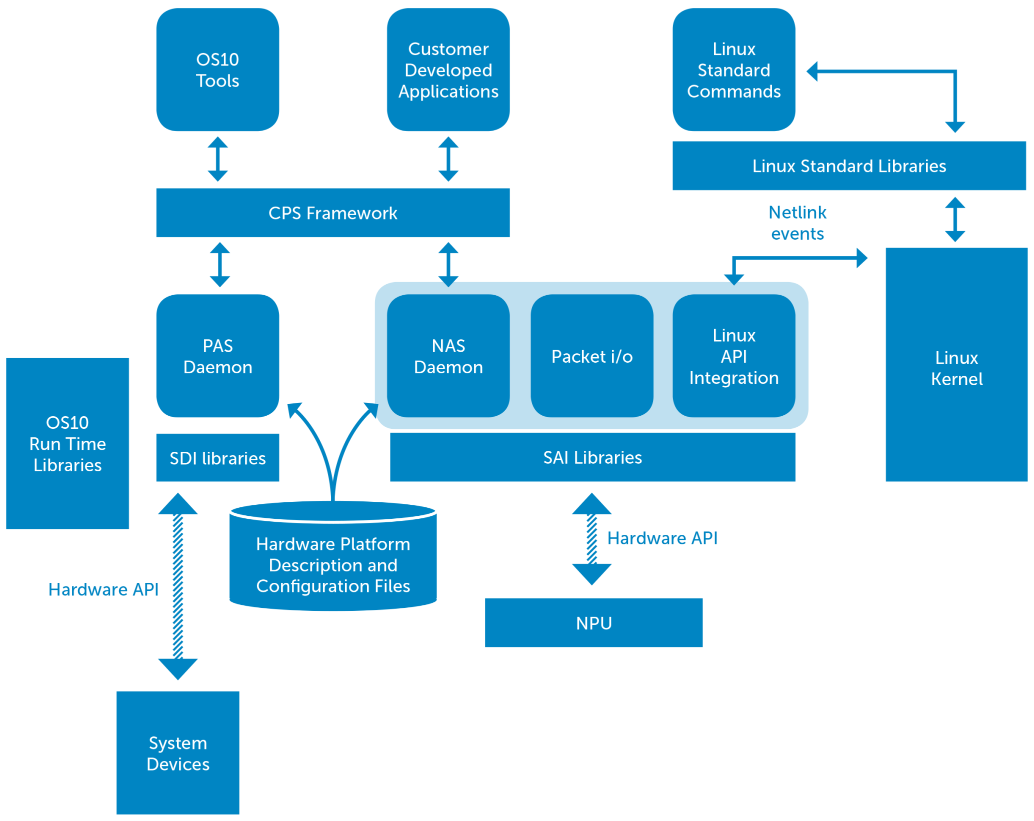
In addition, OS10 provides a Dell-implemented thermal control application which prevents damage of hardware components in case of overheating and/or fan failure.

⚠ CAUTION: DO NOT disable the thermal control application as hardware damage may result.

Run-Time Overview

A brief overview of the structure of OS10 software and run-time behavior is shown here:

Figure 3 – OS10 Run-Time



OS10 Processes

The OS10 platform uses the following processes:

`cps_api_svc`	Executes the CPS **broker** which mediates all CPS operations and events.
`base_pas_svc`	PAS daemon, which executes PAS functionality
`base_nas_svc`	NAS daemon, which executes NAS functionality

The following applications manage components of the OS10 system:

`base_env_tmpctl_svc`	Manages environment temperature control; executes the thermal control algorithm. ⚠ **CAUTION:** DO NOT disable this process; hardware damage may result.
`base_nas_front_panel_ports_svc`	Manages physical port mapping to Linux interfaces.
`base_nas_phy_media_config_svc`	Manages the configuration of physical media.
`base_nas_monitor_phy_media_svc`	Monitors physical media (SFP) events generated by PAS when you insert a pluggable module; automatically configures port parameters.
`base_ip_svc`	Gets/sets IP address parameters via the CPS API.
`base_interface_svc`	Gets/sets interface parameters via the CPS API.
`base_nas_shell_svc`	Executes NPU shell commands.

NAS Linux Adaptation Functionality

Integration with Standard Linux APIs

The OS10 NAS daemon seamlessly integrates standard Linux network APIs with NPU hardware functionality. The NAS daemon registers and listens to networking (netlink) events. When it receives an event, the NAS daemon processes the event contents and programs the NPU with relevant information, such as enabling/disabling an interface, adding/removing a route, or adding/deleting a VLAN.

Linux Interfaces Associated with Physical Ports

OS10 uses internal Linux **tap** devices to associate physical ports with Linux interfaces. When the NAS detects a change in physical port status (up/down), the daemon propagates the new port status to the associated tap device.

Packet I/O

In OS10, packet I/O describes control-plane packet forwarding between physical ports and associated Linux interfaces.

In the current OS10 version, packet I/O is implemented as a standalone thread of the NAS daemon. A packet received by the NPU is forwarded to the CPU. Packet I/O thread receives the packet via the SAI API callback. Each received packet contains the identity of the source physical port. Packet I/O module then injects the packet to the tap device associated with the source physical port. Applications receive packets from the Linux IP stack using the standard sockets.

Conversely, applications use tap devices to transmit packets. The packet I/O receives the transmitted packet from Linux IP stack. Based on source tap device of the packet, the transmitted packet is forwarded to the associated physical port.

CPS Services

The NAS daemon registers with CPS as a server application to provide CPS programmability of the packet NPU. The NAS performs create, delete, set, and get operations for objects which model the networking functionality defined by OS10.

Similarly, the PAS daemon also registers with CPS as a server application in order to provide CPS programmability for system devices.

File System Organization

The current version of OS10 uses a standard Linux ext4 file system. The Linux packages required for OS10 are pre-installed on the file system as per standard Linux package locations.

OS10 specific files are installed in the `/opt/dell/os10` and `/etc/opt/dell/os10` directories:

`/opt/dell/os10` contains binaries, libraries, and YANG models
`/etc/opt/dell/`os10 contains platform description files and default configuration files

Platform Description Files

The platform description files contain a description of the hardware modules that apply to the current platform; for example, the number of physical ports, fans, and power supplies.

Default Configuration Files

The default configuration files contain initialization information applicable to the current platform; for example, initial SAI configuration or system ACL rules to be applied at initialization

System Startup

OS10 uses the systemd framework to define the system startup sequence. OS10 unit definitions are stored in:

```
/etc/systemd/system/dn-base-group.target.wants
```

Installation

Overview

Install OS10 using an industry-standard Open Network Install Environment (ONIE) procedure. For detailed information about ONIE installation, see http://www.opencompute.org/wiki/Networking/ONIE.

ONIE supports two types of installation:

- Automatic (zero-touch) installation, in which the system automatically configures Ethernet interfaces, connects to an image server, and downloads and installs an OS10 image
- Manual installation, which requires manual configuration of an Ethernet port and manual specification of an OS10 image file.

The section describes step-by-step procedures to install OS10.

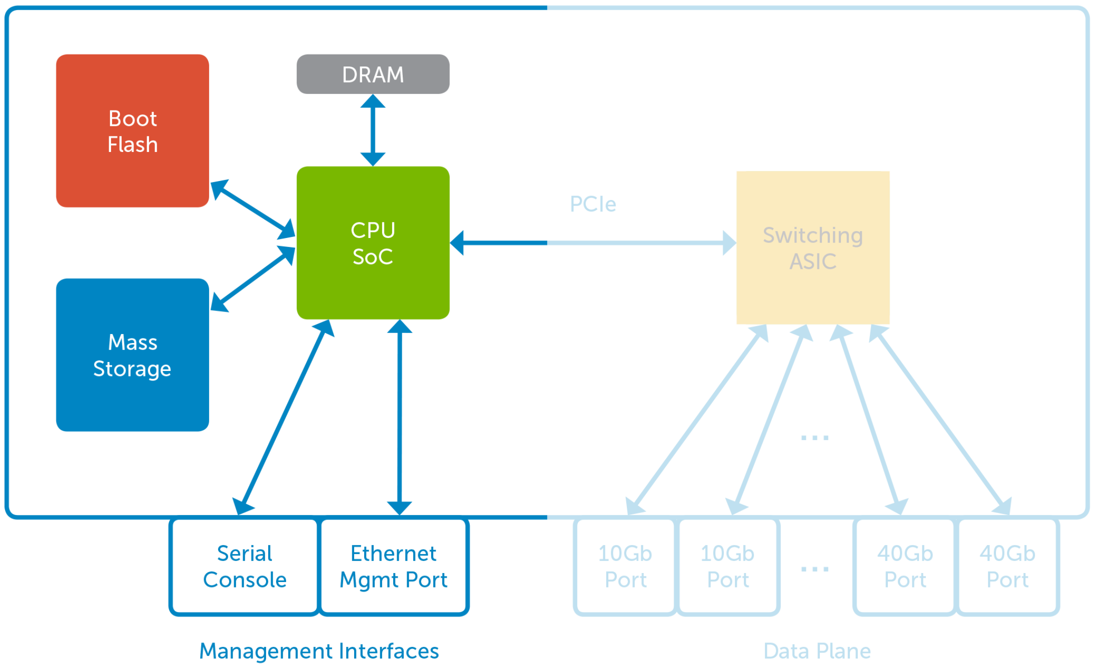
System Setup

Before you install OS10, ensure that the system is connected as follows:

- Connect a serial cable and terminal emulator to the console serial port. The required serial port settings are 115200, 8 data bits, and no parity.
- If you prefer downloading an image over an Ethernet network, connect the management Ethernet port to the network.

To locate the console port and management Ethernet port, refer to the Getting Started Guide shipped with your switch.

Figure 4 – Using the Serial and Management Port to Install OS10



Install OS10

When you power-up the switch, Dell Diagnostics (Diag) and ONIE software are pre-loaded. The boot menu screen is displayed on the console.

```
+--------------------------------------------------------------------------+
|*ONIE: Install OS                                                         |
| ONIE: Rescue                                                             |
| ONIE: Uninstall OS                                                       |
| ONIE: Update ONIE                                                        |
| ONIE: Embed ONIE                                                         |
| ONIE: Diag ONIE                                                          |
+--------------------------------------------------------------------------+
```

ONIE modes are listed on the screen:

- Install OS - Installs an OS10 image.
- Rescue - Reboots the system into ONIE for repair, debugging, and forensics.
- Uninstall OS - Deletes the contents of all disk partitions except ONIE.
- Update ONIE - Installs a new ONIE version.
- Embed ONIE - Formats an empty disk and install ONIE.
- Diag ONIE - Runs the system vendor's diagnostics.

You can install an OS10 image automatically using the ONIE image discovery process, or manually if no DHCP server is available. Download a platform-specific OS10 image from the Dell-recommended location.

CAUTION: During an automatic or manual OS10 installation, if an error condition occurs that results in an unsuccessful installation, DO NOT power cycle or reboot the system. Loss of data can occur.

Automatic Installation

You can automatically (zero-touch) install an OS10 image using the ONIE image discovery process by following the procedure in the ONIE specification at https://github.com/opencomputeproject/onie/wiki/Design-Spec-SW-Image-Discovery.

Manual Installation

If a DHCP server is not available, you can manually install an OS10 image.

1. Power up the system; it automatically boots up with the `ONIE: Install OS` option on the boot menu.

2. Stop the ONIE discovery process by entering the `onie-discovery-stop` command.

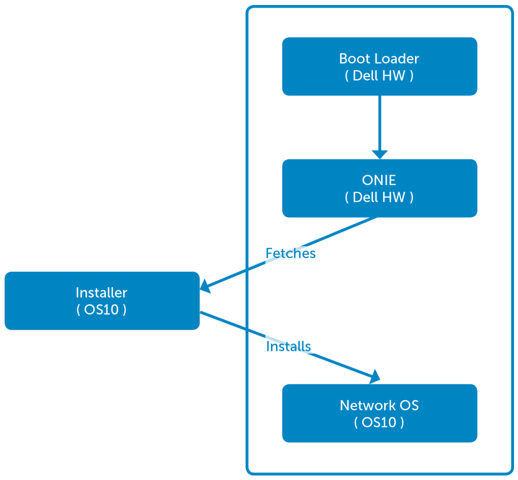
3. Assign a unique IP address to the management port by using the `ifconfig` command.

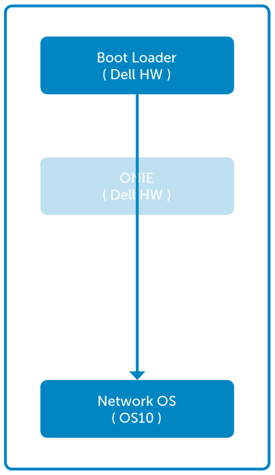
4. Locate the platform-specific Dell OS10 image that you want to download from a network server. Enter the `onie-nos-install` *image-url* command to download and install the image from an HTTP, FTP, or TFTP server; for example:

```
ONIE:/ # onie-nos-install http://192.168.0.1/os10_s6000.bin
```

The Dell OS10 image is installed on the system. The system automatically reboots and loads OS10.

Figure 5 – Installing OS10 over a Network – Figure 6 – Booting OS10





You can also install OS10 using USB media as follows:

1. Power up the system. It will automatically boot up with the `ONIE Install OS` option.
2. Stop the ONIE discovery process by entering `onie-discovery-stop`.
3. Create a USB mount location on the system by entering `mkdir /mnt/media`.
4. Enter the `mount` command to mount the USB media plugged in the USB port on the switch.
5. Enter the `onie-nos-install` command to install the OS10 image file from a specified USB path; for example:

```
ONIE:/ # onie-nos-install /mnt/media/os10_s6000.bin
```

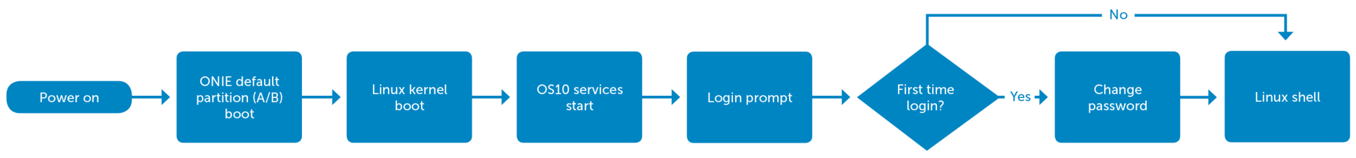
In the example, `/mnt/media` specifies where the USB partition is mounted.

NOTE: An OS10 Installer image creates two partitions on the system: OS10-A (active) and OS10-B (standby). After OS10 installation, the system boots up by default from OS10-A.

OS10 Boot Sequence

After you install an OS10 image, the system boots up in the following sequence:

Figure 7 – OS10 Boot Sequence



1. After the switch powers up or reboots, the boot menu is displayed. The system autoboots by loading the OS10 image in the OS10-A partition. To change the default partition, use the arrow keys to select another partition, such as OS10-B or ONIE, before the autoboot starts.

OS10 Boot Menu

```
+----------------------------------------------------------------------------+
|*OS10-A                                                                     |
| OS10-B                                                                     |
| ONIE                                                                       |
+----------------------------------------------------------------------------+
```

2. Linux boots from the OS10-A partition on the disk and starts the systemd daemon in the root file system as part of the initial setup before the Linux login displays.

The following OS10 custom services are started by the systemd daemon during system initialization:

- The PAS service initializes the platform and devices on the system.
- The NAS service initializes the NPU and system interfaces.
- Other OS10 services create Linux interfaces that map to physical, front-panel ports on the switch.

After OS10 custom services run successfully and the system boots up, the Linux prompt is displayed on the console. Log in using the default `linuxadmin` username and `linuxadmin` password.

During the first login, you are prompted to change the password. This password reset is mandatory. See the next section, Default Login and Password Management, for information on how to set a new password.

NOTE: If the OS10 service that creates internal Linux interfaces is unsuccessful, the system bootup waits 300 seconds before timing out and displaying the Linux login prompt.

3. At the OS10 Linux shell prompt, enter Linux administration commands, OS10 NPU or system-related scripts.

Default Login and Password Management

The first time OS10 boots up on a switch, use the default username `linuxadmin` and password `linuxadmin` to log in.

The default `linuxadmin` username is created for OS10 administration activities on the system. `linuxadmin` is part of the Linux sudo group which can execute privilege commands.

During the first OS10 login, you must change the default `linuxadmin` password for security reasons. The new password is saved on the system for future logins. To set a new OS10 user password, follow the command prompts below.

```
========================> linuxadmin/linuxadmin<=======================

Dell Networking Operating System (OS10)

OS10 login: linuxadmin
Password:linuxadmin    >> only for first-time login
You are required to change your password immediately (root enforced)
Changing password for linuxadmin.
(current) UNIX password:   >>> linuxadmin
Enter new UNIX password: >>> user preferred password
Retype new UNIX password:  >>> re-enter user preferred password
Linux OS10 3.16.7-ckt11 #1 SMP Wed Feb 3 17:43:04 PST 2016 x86_64
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
-* Dell Networking Operating System (OS10) *-
-* *-
-* Copyright (c) 1999-2016 by Dell Inc. All Rights Reserved. *-
-* *-
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-

This product is protected by U.S. and international copyright and
intellectual property laws. Dell and the Dell logo are trademarks
of Dell Inc. in the United States and/or other jurisdictions. All
other marks and names mentioned herein may be trademarks of their
respective companies.

linuxadmin@OS10:~$
```

System Administration

Overview

This section covers tasks needed to properly maintain and upgrade the system running OS10. This section includes system defaults, configuring interfaces, installing Linux packages, logging and upgrades.

System Defaults

When the system boots up, the following default system configuration is applied:

- All Linux interfaces are created and mapped to physical ports.
- All Linux interfaces are in Administratively Down state.
- The Management interface is `eth0`, and the management IP address is dynamically assigned using DHCP.
- OS10 processes are activated after system boot up. Refer to Run-Time Overview for the list of processes.
- ACL entries are installed to direct control-plane packets for protocols, such as LLDP and OSPF, to the Linux interfaces associated with physical ports. Refer to Default XML Configuration Files for the list of ACL entries.
- QoS initialization sets up the default scheduler hierarchy and map all packets to Queue 0.

Remote Access

Access the system remotely via SSH. You must configure the management IP address for remote access.

Using SSH

By default, SSH service is enabled. The username and password are both `linuxadmin`.

```
$ ssh linuxadmin@<managementipaddress>
```

System Utility Commands

`os10-ethtool` - Displays the interface statistics and the transceiver information (see Monitoring).

`os10-show-stats` - Displays the detailed statistics of a physical port (see Monitoring).

`os10-config-fanout` - Fans out native 40G ports to 4x10G interfaces (see Physical Ports in Networking Features).

`os10-switch-shell` - Executes NPU commands (see Debugging Interfaces in Troubleshooting).

`os10-config-switch` - Sets and gets values of different switching entities.

Example: `os10-config-switch` retrieves switch values

```
$ os10-config-switch show
Key: 1.30.1966121.
base-switch/switching-entities/switch-count = 1
Key: 1.30.1966121.1966082.1966085.
base-switch/switching-entities/switching-entity/bridge-table-size = 163840
base-switch/switching-entities/switching-entity/acl-table-max-priority = 11
base-switch/switching-entities/switching-entity/acl-entry-min-priority = 0
base-switch/switching-entities/switching-entity/acl-table-min-priority = 0
base-switch/switching-entities/switching-entity/ecmp-hash-fields  =  8,9,5,3,10,4,2,1,7,6
base-switch/switching-entities/switching-entity/npu-identifiers  =  0
base-switch/switching-entities/switching-entity/mac-age-timer = 1800
base-switch/switching-entities/switching-entity/lag-hash-algorithm = 2
base-switch/switching-entities/switching-entity/switch-id = 0
base-switch/switching-entities/switching-entity/temperature = 45
base-switch/switching-entities/switching-entity/switch-mode = 2
base-switch/switching-entities/switching-entity/lag-hash-fields  =  8,9,5,3,10,4,2,1,7,6
base-switch/switching-entities/switching-entity/max-ecmp-entry-per-group = 64
base-switch/switching-entities/switching-entity/ecmp-hash-algorithm = 2
base-switch/switching-entities/switching-entity/acl-entry-max-priority = 2147483647
base-switch/switching-entities/switching-entity/default-mac-address = 90:b1:1c:f4:aa:81
base-switch/switching-entities/switching-entity/max-mtu = 9216
```

Example: `os10-config-switch` reconfigures the mac-age-timer value

```
$ os10-config-switch set switch-id=0 mac-age-timer=1900
Success
$ os10-config-switch show
Key: 1.30.1966121.
base-switch/switching-entities/switch-count = 1
Key: 1.30.1966121.1966082.1966085.
base-switch/switching-entities/switching-entity/bridge-table-size = 163840
base-switch/switching-entities/switching-entity/acl-table-max-priority = 11
base-switch/switching-entities/switching-entity/acl-entry-min-priority = 0
base-switch/switching-entities/switching-entity/acl-table-min-priority = 0
base-switch/switching-entities/switching-entity/ecmp-hash-fields  =  8,9,5,3,10,4,2,1,7,6
base-switch/switching-entities/switching-entity/npu-identifiers  =  0
base-switch/switching-entities/switching-entity/mac-age-timer = 1900
base-switch/switching-entities/switching-entity/lag-hash-algorithm = 2
```

```
base-switch/switching-entities/switching-entity/switch-id = 0
base-switch/switching-entities/switching-entity/temperature = 54
base-switch/switching-entities/switching-entity/switch-mode = 2
base-switch/switching-entities/switching-entity/lag-hash-fields  =  8,9,5,3,10,4,2,1,7,6
base-switch/switching-entities/switching-entity/max-ecmp-entry-per-group = 64
base-switch/switching-entities/switching-entity/ecmp-hash-algorithm = 2
base-switch/switching-entities/switching-entity/acl-entry-max-priority = 2147483647
base-switch/switching-entities/switching-entity/default-mac-address = 90:b1:1c:f4:a5:23
base-switch/switching-entities/switching-entity/max-mtu = 9216
```

`os10-show-transceivers` Get the information about the transceiver type present.

Example : `os10-show-transceivers summary` displays the installed transceivers

```
$ os10-show-transceivers summary
Front Panel Port                Media Type          Part Number         Serial Number   DellQualified
1                      QSFP 40GBASE SR4          AFBR-79E4Z-D-FT1      7503832L005V              Yes
2                      QSFP 40GBASE SR4          AFBR-79EQDZ-FT1       482943B200GW              Yes
3                   QSFP 40GBASE CR4 3M          616750003             CN0FC6KV35D6864           Yes
4                           Not Present
5                           Not Present
.
.
.
.
32                           Not Present
```

Default XML Configuration Files

CAUTION: Modifying configuration files may affect the default system behaviour and can have adverse effects on how the system behaves.

The following XML configuration files are stored in the `/etc/opt/dell/os10` directory.

`base_qos_init.xml` - Specifies the default QoS entries which are applied to the NPU during system boot up as part of the systemd service.

`base_port_physical_mapping_table.xml` - Creates a mapping between physical ports to the Linux interfaces. All the interfaces are created during system boot up.

`core_rotate_config.xml` - Contains the core rotation configuration and is required by the script that performs the core rotation as part of the CRON Daemon service.

`dn_nas_default_init_config.xml` - Contains default configuration of objects such as Mirror, sFlow, VLAN created during system boot up as part of systemd service.

`dn_nas_fanout_init_config.xml` - Specifies the interfaces that are fanned out during system boot up.

`phy_media_default_npu_setting.xml` - Specifies the transceiver information such as transceiver type, speed etc.

`mgmt-intf.xml` - Specifies the management interface properties.

`env_tmpctl_config.xml` - Specifies parameters such as sensor names and temperature control algorithm to the temperature control module.

`init.xml` - Specifies the NPU-related settings during system boot up such as physical port properties, hashing alogorithms etc.

`nas_master_list.xml` - Lists all the ACL entries that are installed during boot up.

`nas_detail_list.xml` - Specifies all the fields for the ACL entries present in the `nas_master_list.xml` file.

Operations

Configure the Management Interface IP address

Configure the management interface IP address by editing the `/etc/network/interfaces` file.

```
$ cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d
auto eth0
    iface eth0 inet static
         address 10.11.133.40
         netmask 255.255.0.0
         gateway 10.11.133.254
    $ service networking restart
```

Secure the Management Interface

If you need to secure the management interface outside of `SSH`, use the `iptables` command or rate limiting to limit access to the management interface.

Configure a Physical Port

Refer to Configure an Interface using Linux Commands in Networking Features for information about how to configure physical port attributes.

Create User Accounts

Use standard Linux commands to manipulate user accounts. Examples of these commands are: `useradd`, `userdel`, `usermod` and `passwd`. Configure access privileges with the `usermod` command.

Configure Time and Date

Use the `date` command or NTP to configure the time and date.

```
$ date -s "16 FEB 2016 13:12:00"
Tue Feb 16 13:12:00 UTC 2016
```

OS10 User Commands

- `os10-show-version` — Displays software and system information.

```
$ os10-show-version
DELL_Networking_OS10_Software_Version=10.0.0B.1368
NAME="OS10-Base"
LONGNAME="Dell Networking OS10-Base"
VERSION="10.0.0B.1368"
PLATFORM="S6000"
ARCHITECTURE="x86_64"
BUILT_FROM="S6000 Base only Blue Print 0.1.0"
BUILD_DATE="2016-02-11T12:00:00.912-08:00"
COPYRIGHT="Copyright (c) 1999-2016 by Dell Inc. All Rights Reserved."
SYSTEM_UPTIME="4 days, 18 hours, 3 minutes"
```

- `os10-logging` — Enables logging. Log information appears in the syslog file var/log/syslog file.

```
$ os10-logging
 ============================================================================================================
=========
[show-id]                                                - displays ids of modules, log-levels and sub-
levels
[show] [all] | [module-id] {log-level} {log-sub-level}       - displays current logging status for all/given
                                                           sub-system/given module and log-level/given
                                                           module, log-level and sub-log-level
[enable] [all] | [module-id] {log-level } {log-sub-level}    - Enables logging status for all/given
                                                           sub-system/given module and log-level/given
                                                           module, log-level and sub-log-level
[disable] [all] | [module-id] {log-level } {log-sub-level}   - Disables logging status for all/given
                                                           sub-system/given module and log-level/given
                                                           module, log-level and sub-log-level

NOTE :  1. For enable and disable log-level and log-sub-level is optional when using module-id.
        If only module-id is given it will enable/disable all log-levels and log-sub-levels for that
        module-id, similarly if module-id and log-level is given, it will enable all log-sublevel for
        the module-id and log-level
        2. Instead of Module Ids now you can use the module name as a string as well, for eg.
        os10-logging enable L3_SERVICES
==============================================================================================
=========

$ os10-logging enable all
```

- `os10-show-env` - Displays system hardware information.

```
$ os10-show-env

Node
       Vendor name:            Dell
       Service tag:            69Y8VS1
       PPID:                           CN-08YWFG-28298-3AR-0087-A00
       Platform name:
       Product name:                   S6000
       Hardware version:
       Number of MAC addresses:        129
       Base MAC address:               90:b1:1c:f4:a8:30
       Operating status:               Fail
Power supplies
       Slot 1
               Present:                Yes
               Vendor name:
               Service tag:
               PPID:                   CN0T9FNW282983AR020
               Platform name:
               Product name:           CN0T9FNW282983AR020
               Hardware version:
               Operating status:       Up
               Input:                  AC
               Fan airflow:            Normal
       Slot 2
               Present:                Yes
               Vendor name:
               Service tag:
               PPID:
               Platform name:
               Product name:
               Hardware version:
               Operating status:       Up
               Input:                  Invalid
               Fan airflow:            Invalid
Fan trays
```

```
       Slot 1
               Present:                Yes
               Vendor name:
               Service tag: &310;                  PPID:                   CN0MGDH8282983AR028
               Platform name:
               Product name:           CN0MGDH8282983AR028
               Hardware version:
               Operating status:       Up
               Fan airflow:            Reverse
       Slot 2
               Present:                Yes
               Vendor name:
               Service tag:
               PPID:                   CN0MGDH8282983AR028
               Platform name:
               Product name:           CN0MGDH8282983AR028
               Hardware version:
               Operating status:       Up
               Fan airflow:            Reverse
       Slot 3
               Present:                Yes
               Vendor name:
               Service tag:
               PPID:                   CN0MGDH8282983AR028
               Platform name:
               Product name:           CN0MGDH8282983AR028
               Hardware version:
               Operating status:       Up
               Fan airflow:            Reverse
Fans
       Fan 1, PSU slot 1
               Operating status:       Up
               Speed (RPM):            6720
               Speed (%):              37
       Fan 1, Fan tray slot 1
               Operating status:       Up
               Speed (RPM):            6916
               Speed (%):              38
       Fan 2, Fan tray slot 1
               Operating status:       Up
               Speed (RPM):            6803
               Speed (%):              37
       Fan 1, Fan tray slot 2
               Operating status:       Up
               Speed (RPM):            7188
               Speed (%):              39
       Fan 2, Fan tray slot 2
               Operating status:       Up
               Speed (RPM):            7175
```

```
                Speed (%):              39
        Fan 1, Fan tray slot 3
                Operating status:       Up
                Speed (RPM):            7201
                Speed (%):              40
        Fan 2, Fan tray slot 3
                Operating status:       Up
                Speed (RPM):            6698
                Speed (%):              37
Temperature sensors
        Sensor T2 temp sensor, Card slot 1
                Operating status:               Up
                Temperature (degrees C):        33
        Sensor system-NIC temp sensor, Card slot 1
                Operating status:               Up
                Temperature (degrees C):        25
        Sensor Ambient temp sensor, Card slot 1
                Operating status:               Up
                Temperature (degrees C):        27
        Sensor NPU temp sensor, Card slot 1
                Operating status:               Up
                Temperature (degrees C):        46
```

LED Control

Use the `os10-env-set-led` utility to set the state of a logical LED, as represented by PAS.

```
o $ os10-env-set-led [ --entity-type [ psu | fantray | card ] ] [ --slot slot ] led-name [ on | off ]
```

Maintenance

Manage Linux Packages

Use the standard Linux utilities `apt-get` and `dpkg` to manage Linux packages. These utilities provide a simple way to retrieve and install packages from multiple sources using the Linux command line. Before installing a package, you must first configure the IP address of the management port.

NOTE: Ensure that the URL in the sources list (/etc/apt/sources.list) points to the linux.dell.com repositories before installing a Linux package.

Dell recommends using the `apt-get update` command before installing a package.

Use the `dpkg -s <packagename>` command to check the installation status of a particular package.

```
o $ dpkg -s <packagename>
```

Manage System Services

Check a Service Status

Use the `service <servicename> status` command to check the status of a service. The command output indicates whether the service is up and running, or inactive. You can also use the `systemctl` command to check the status of a service.

```
$ service snmpd status
  snmpd.service - LSB: SNMP agents
  Loaded: loaded (/etc/init.d/snmpd)
  Active: active (running) since Wed 2016-02-17 02:16:06 UTC; 2h 39min ago
  CGroup: /system.slice/snmpd.service
          └─930 /usr/sbin/snmpd -Lsd -Lf /dev/null -u snmp -g snmp -I -smux ...
```

Start, Stop, or Restart a Service

Use the `service <servicename> {start|stop|restart}` command to start, stop, or restart a service.

```
$ service snmpd stop
$ service snmpd status
  snmpd.service - LSB: SNMP agents
  Loaded: loaded (/etc/init.d/snmpd)
  Active: inactive (dead) since Wed 2016-02-17 05:00:27 UTC; 3s ago
 Process: 3370 ExecStop=/etc/init.d/snmpd stop (code=exited, status=0/SUCCESS)
```

```
$ service snmpd start
$ service snmpd status
  snmpd.service - LSB: SNMP agents
  Loaded: loaded (/etc/init.d/snmpd)
  Active: active (running) since Wed 2016-02-17 05:00:39 UTC; 1s ago
 Process: 3370 ExecStop=/etc/init.d/snmpd stop (code=exited, status=0/SUCCESS)
 Process: 3395 ExecStart=/etc/init.d/snmpd start (code=exited, status=0/SUCCESS)
  CGroup: /system.slice/snmpd.service
          └─3399 /usr/sbin/snmpd -Lsd -Lf /dev/null -u snmp -g snmp -I -smux...
```

```
$ service snmpd restart
$ service snmpd status
  snmpd.service - LSB: SNMP agents
  Loaded: loaded (/etc/init.d/snmpd)
  Active: active (running) since Wed 2016-02-17 05:00:46 UTC; 1s ago
 Process: 3407 ExecStop=/etc/init.d/snmpd stop (code=exited, status=0/SUCCESS)
 Process: 3412 ExecStart=/etc/init.d/snmpd start (code=exited, status=0/SUCCESS)
  CGroup: /system.slice/snmpd.service
          └─3416 /usr/sbin/snmpd -Lsd -Lf /dev/null -u snmp -g snmp -I -smux...
```

Upgrade OS10

Release images are ONIE installers that contain OS10. You can install an OS10 image by using the os10-image script. The script validates the image and extracts it into the standby partition.

OS10 supports active and standby partitions. You can switch between these partitions.

CAUTION: When you upgrade to a new OS10 version, existing files are erased or overwritten in the standby partition. You must manually move configuration files (such as interfaces and hostname) to a persistent location before you upgrade.

Upgrade the OS10 Image

1. Back up the active configuration.

- Copy active files to a persistent location.

```
$ cp /etc/snmp/snmpd.conf /config/
```

- Verify that all necessary files have been successfully copied.

```
$ /config# ls
etc  lost+found  snmpd.conf
```

2. Download the new OS10 image.

NOTE: You must configure the IP address of the management interface before you download an image. The image files are in .bin format. Ensure there is enough disk space before downloading the image. Refer to the OS10 Release Notes for information regarding disk space.

- Download the software image from the location provided by Dell using the `wget` command.

```
$ wget <http://URL/......../OS10.bin>
```

- Verify that the image has downloaded successfully.

```
$ ls
OS10.bin
```

3. Install the new OS10 image.

- Install the image in the standby partition using the `os10-image` script with the `-i` option, followed by the image path.

NOTE: The new image is always installed in the standby partition.

```
$ os10-image -i OS10.bin
```

- Once the installation is complete, check the new image version by using the `os10-image` script with the `-g` option. The system displays the software image version in the active and standby partitions, and indicates the partition to be used during the next reboot. In this example, the new image (10.0.0.1B) is installed in the standby partition.

```
$ os10-image -g


Active image version 10.0.0B
Standby image version 10.0.1B
Active image will be loaded for next reboot $
```

4. Configure the next-boot partition and reboot.

NOTE: You can change the next-boot partition from active to standby or from standby to active at any time. To cancel the change, use the **`os10-image`** script with the **`-c`** option.

- Configure the next-boot image to the image in the standby partition by using the `os10-image` script with the `-s` option and including the partition name. When the system reboots, the old standby partition becomes the new active partition.

```
$ os10-image -s standby
Success setting boot image to standby
```

- You must reboot the system for the new OS10 version to take effect.

```
$ reboot
```

5. Restore the saved configuration files.

- Copy the saved files from the persistent location into the active partition.

```
$ cp /config/snmpd.conf /etc/snmp/snmpd.conf
```

- Verify that all files have been copied successfully.

```
$ ls
snmp.conf  snmpd.conf
```

Boot to different ONIE Modes using os10-image

Use the `os10-image` script with the `-o` option to boot into ONIE install, uninstall or rescue modes.

- To change the next boot to ONIE install mode and verify the change:

```
$ os10-image -o install
WARNING: OS install requested
WARNING: This will erase all data
Are you sure (y/N)? y
Success setting boot mode to OS install
Reboot required to take effect

$ os10-image -g
Active image version 10.0.0B.1400
```

```
Standby image version 10.0.0B.1401
OS install mode will be loaded on next reboot
```

- To change the next boot to ONIE uninstall mode and verify the change:

```
$ os10-image -o uninstall
WARNING: OS uninstall requested
WARNING: This will erase all data
Are you sure (y/N)? y
Success setting boot mode to OS uninstall
Reboot required to take effect

$ os10-image -g
Active image version 10.0.0B.1400
Standby image version 1.0.0.1401
OS uninstall mode will be loaded on next reboot.
```

- To change the next boot to be ONIE rescue mode and verify the change:

```
$ os10-image -o rescue
WARNING: Rescue boot requested
Are you sure (y/N)? y
Success setting boot mode to Rescue boot
Reboot required to take effect

$ os10-image -g
Active image version 10.0.0B.1400
Standby image version 10.0.0B.1401
OS rescue mode will be loaded on next reboot
```

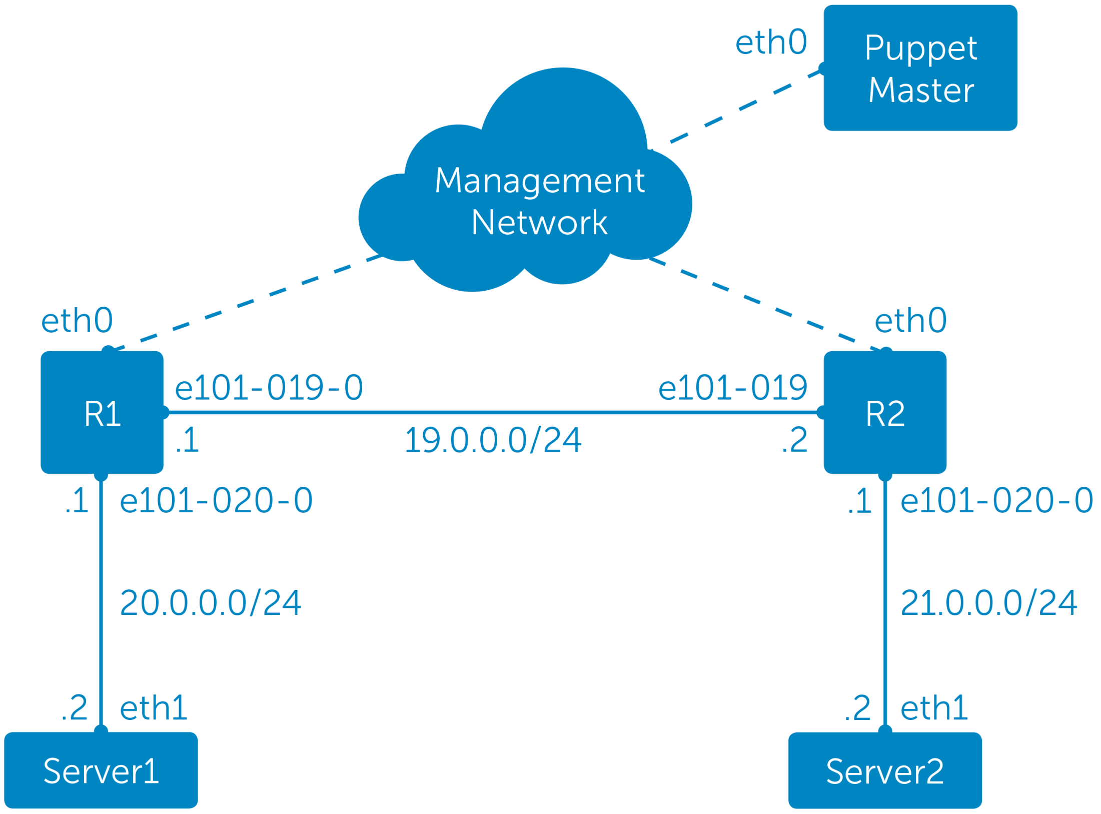
- To cancel the next boot to ONIE mode and verify the change:

```
$ os10-image -c


Cancelled pending uninstall mode at next reboot


$ os10-image -g
Active image version 10.0.0B.1400
Standby image version 10.0.0B.1401
Active image will be loaded on next reboot
```

Use Case: Orchestration Using Puppet

This use case describes how to use Puppet to configure OS10 systems. In this example, each system is connected to a server.

Figure 8 – Puppet Use Case



Prerequisites

1. Install the Puppet master on an external server and configure it to manage systems running OS10 by following the instructions at www.puppetlabs.com.

2. Install and configure the Puppet agent on both OS10 systems by following the instructions on www.puppetlabs.com.

3. Verify if the Puppet master can communicate with the Puppet agents through the management network.

Example: Puppet Configuration

The following code shows a sample Puppet manifest used to manage the two OS10 systems in the example.

```
node 'R1.dell.com' {
   $int_enabled = true
   $int_loopback = '2.2.2.2'
   $int_layer3 = {
      e101-019-0  => {'int'=>'e101-019-0', 'address' => '19.0.0.1', 'netmask' => '255.255.255.0',
'cidr_netmask' => 24},
      e101-020-0  => {'int'=>'e101-020-0', 'address' => '20.0.0.1', 'netmask' => '255.255.255.0',
'cidr_netmask' => 24},
   }

   $bgp = {
       myasn => 65000,
       peergroupv4 => [ { name => 'R2', asn => 65000, peers => [ '19.0.0.2','20.0.0.2' ] } ]
   }
   include ibgp::switch
}

node 'R2.dell.com' {
   $int_enabled = true
   $int_loopback = '3.3.3.3'
   $int_layer3 = {
       e101-019-0  => { 'int'=> 'e101-019-0', 'address' => '19.0.0.2', 'netmask' => '255.255.255.0',
'cidr_netmask' => 24 },
       e101-020-0  => { 'int'=> ' e101-020-0','address' => '21.0.0.1', 'netmask' => '255.255.255.0',
'cidr_netmask' => 24 },
   }

   $bgp = {
       myasn => 65000,
       peergroupv4 => [ { name => 'R1', asn => 65000, peers => [ '19.0.0.1','20.0.0.1' ] } ]
   }
   include ibgp::switch
}
```

The following code shows the `ibgp::switch` and `ibgp::quagga` class definitions.

```
class ibgp::switch {
   include ibgp::quagga
}

class ibgp::quagga {
   service { 'quagga':
       ensure    => running,
       hasstatus => false,
       enable    => true,
   }

   file { '/etc/quagga/daemons':
       owner  => quagga,
       group  => quagga,
       source => 'puppet:///modules/ibgp/quagga_daemons',
       notify => Service['quagga']
   }

   file { '/etc/quagga/Quagga.conf':
       owner   => root,
       group   => quaggavty,
       mode    => '0644',
       content => template('ibgp/Quagga.conf.erb'),
       notify  => Service['quagga']
   }
}
```

The `Quagga.conf.erb` file is shown here:

```
! This file is managed by Puppet

hostname zebra
log file /var/log/quagga/zebra.log
hostname ospfd
log file /var/log/quagga/ospfd.log
log timestamp precision 6
hostname bgpd
log file /var/log/quagga/bgpd.log
!
password cn321
enable password cn321
!
<%   @int_layer3.each_pair do |layer3, options| -%>
interface <%= options["int"] %>
ip address <%=options["address"]%>/<%=options["cidr_netmask"] %>
no shutdown
<%     end -%>

route-id <%= @int_loopback %>
<% if @bgp -%>
router bgp <%= @bgp["myasn"] %>
 maximum-paths ibgp 4
 bgp router-id <%= int_loopback %>
 bgp log-neighbor-changes
 network <%= @int_loopback %>/32
<%     @int_bridges.each_pair do |bridge, options| -%>    network <%= options["address"] %>/<%=
options["cidr_netmask"] %>
<%     end -%>
<%   @bgp["peergroupv4"].each do |peergroup| -%>
 neighbor <%= peergroup["name"] %> peer-group
 neighbor <%= peergroup["name"] %> remote-as <%= peergroup["asn"] %>
<%     if peergroup["name"]["routereflectorclient"] -%>
 neighbor <% peergroup["name"] %> route-reflector-client
<%     end -%>
<%     peergroup["peers"].each do |peer| -%>
 neighbor <%= peer %> peer-group <%= peergroup["name"] %>
<%     end -%>
<%   end -%>
<% end -%>
!
<% if @int_unnumbered -%>
<%   @int_unnumbbers.each do |interface| -%>
 no passive-interface <%= interface %>
<%   end -%>
 network <%= @int_loopback %>/32 area 0.0.0.0
```

```
<%   if @hostnetranges and @is_leaf -%>
<%     @hostnetranges.each do |hostnetrange| -%>
 network <%= hostnetrange %> area 0.0.0.0
<%     end -%>
<%   end
-%>  <% end -%>
```

The `quagga_daemons` file is shown here:

```
zebra=yes
bgpd=yes
ospfd=no
ospf6d=no
ripd=no
ripngd=no
isisd=no
babeld=no
```

Use Case: Monitoring Using Nagios

Nagios is an open source monitoring system which is used to monitor network services, applications. and processes. Nagios sends notifications about critical errors or failures on an OS10 system.

Nagios provides remote monitoring using the Nagios Remote Plugin Executor (NRPE), which communicates with the `check_nrpe` plugin in the Nagios server.

This use case describes how to set up a system running OS10 as a Nagios client.

NOTE: The use case assumes that you have already installed the Nagios server and it is running in an external host. For more information about Nagios, go to www.nagios.org.

Configure an OS10 System as a Nagios Client

To set up a system running OS10 as a Nagios client, you must install the Nagios NRPE server and Nagios plugins. The Nagios NRPE server is the agent which allows remote system monitoring.

1. Install the Nagios NRPE server on an OS10 system.

```
$ apt-get install nagios-nrpe-serve
```

2. Edit the allowed hosts to include the Nagios server IP address.

Once the Nagios NRPE server is successfully installed, edit the allowed hosts field in the `/etc/nagios/nrpe.cfg` file and include the Nagios server IP address.

```
$ cat nrpe.cfg
# ALLOWED HOST ADDRESSES
# This is an optional comma-delimited list of IP address or hostnames
# that are allowed to talk to the NRPE daemon. Network addresses with a bit mask
# (i.e. 192.168.1.0/24) are also supported. Hostname wildcards are not currently supported.
# Note: The daemon only does rudimentary checking of the client's IP
# address.  I would highly recommend adding entries in your /etc/hosts.allow
# file to allow only the specified host to connect to the port
# you are running this daemon on.
#
# NOTE: This option is ignored if NRPE is running under either inetd or xinetd
allowed_hosts=10.11.96.94
```

3. Restart the Nagios NRPE server on the OS10 system. You must restart the Nagios NRPE server on the system for the allowed host changes to take effect.

```
$ service nagios-nrpe-server restart
$ service nagios-nrpe-server status
  nagios-nrpe-server.service - LSB: Start/Stop the Nagios remote plugin execution daemon
  Loaded: loaded (/etc/init.d/nagios-nrpe-server)
  Active: active (running) since Wed 2016-02-17 22:27:57 UTC; 4s ago
 Process: 8340 ExecStop=/etc/init.d/nagios-nrpe-server stop (code=exited, status=0/SUCCESS)
 Process: 8345 ExecStart=/etc/init.d/nagios-nrpe-server start (code=exited, status=0/SUCCESS)
  CGroup: /system.slice/nagios-nrpe-server.service
          └─8348 /usr/sbin/nrpe -c /etc/nagios/nrpe.cfg -d

[...]
```

4. Install Nagios plugins.

   Nagios plugins are extensions to the Nagios Core (Nagios Core is the daemon running on the Nagios server). A plugin monitors the services and resources on an OS10 system and returns the results to the Nagios server. For more information about Nagios plugins, go to www.nagios.org. To install the required Nagios plugins:

```
$ apt-get install nagios-plugin
```

Configure a Nagios Server to Monitor an OS10 System

On a Nagios server, you must configure the OS10 system and the services which you want to be monitored.

1. Add the OS10 system for Nagios server monitoring.

Update the `clients.cfg` file on the Nagios server with the OS10 system IP address to enable monitoring:

```
define host{
        use                     linux-server
        host_name               Dell_OS10
        alias                   client
        address                 10.x.x.x
      }
```

2. Enter the commands to be used by Nagios services to monitor an OS10 system.

Enter check commands in the `commands.cfg` file on the Nagios server that you can reference in host, service, and contact definitions:

```
define command{
        command_name       check_nrpe
        command_line       $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
        }


define command{
        command_name       check_remote_disk
        command_line       $USER1$/check_disk -w $ARG1$ -c $ARG2$ -p $ARG3$
        }


define command{
        command_name       check_remote_procs
        command_line       $USER1$/check_procs -w $ARG1$ -c $ARG2$ -s $ARG3$
        }
```

3. Configure the services to be monitored on an OS10 system.

Edit the `clients.cfg` file on the Nagios server to configure the Nagios services to be monitored on an OS10 system with the commands specified in Step 2. The monitoring results of the Nagios services are returned to the Nagios core.

```
define service{
        use                             generic-service
           host_name                       Dell_OS10
        service_description             Current Processes
        check_command                   check_nrpe!check_total_procs
        }

define service{
        use                             generic-service
                host_name                       Dell_OS10
        service_description             Current Disk Space
        check_command                   check_nrpe!check_remote_disk
        }
```

Networking Features

Overview

OS10 supports the ability to model and configure various networking features in the Network Processing Unit (NPU) through two methods: Linux commands and CPS APIs. This section describes how to program networking features using Linux commands. Refer to Introduction to CPS for a description of the CPS framework.

Networking functionality in OS10 is handled by the Network Adaptation Service, which listens to netlink events for Layer 2 and Layer 3 configurations, and programs the NPU.

Supported Networking Features

Networking Feature	Configure with Linux Commands/Open Source Application	Configure with CPS API
Interfaces		
Physical	Yes	Yes
Link Aggregation (LAG)	Yes (Bond)	Yes
VLAN	Yes	Yes
Fanout (4x10G)	No	Yes (script)
Layer 2 Bridging		
LLDP	Yes	No
MAC address table	No	Yes
STP	Yes	Yes
VLAN	Yes	Yes
Layer 3 Routing		
ECMPv4	Yes	Yes
ECMPv6	Yes	Yes
IPv4	Yes	Yes
IPv6	Yes	Yes
Unicast routing	Yes	Yes
QoS	No	Yes
ACLs	No	Yes
Monitoring		
Mirroring	No	Yes
sFlow	No	Yes
Port and VLAN statistics	No	Yes

Interfaces

This section describes how to create and manage physical and virtual interfaces. Physical port interfaces refer to ports on the NPU and do not include the management port.

To manage OS10 interfaces, applications access physical and virtual ports using the Linux interfaces to which they are mapped. OS10 allocates an `ifindex` for each Linux interface. This is used in the CPS APIs to refer to the interface.

Mapping Physical Ports to Linux Interfaces

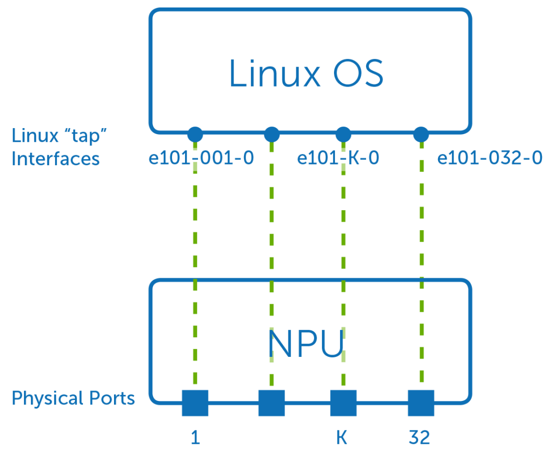
In OS10, each physical port is mapped to an internal Linux interface in the format **eNSS-PPP-F.vvvv**, where:

- **e** means that it is an Ethernet port.
- **N** is the node ID and is always set to 1.
- **SS** is the slot number and is always set to 01.
- **PPP** is the port number (1-999).
- **F** is the number of a 4x10G fanout port (0-9).
- **vvvv** is the VLAN ID number (0-4095).

For example, the e101-031-0 interface refers to physical port 31without a fanout; e101-005-4 identifies fanout port 4 on physical port 5; e101-001-2 identifies fanout port 2 on physical port 1.

Linux interfaces are created during OS10 bootup and represent the physical ports on the NPU in a one-to-one mapping. The mapping of Linux interfaces to physical port is shown here:

Figure 9 – Mapping of Linux Interfaces to Physical Ports



The internal Linux interfaces allow applications to configure physical port parameters, such as MTU, port state, and link state. Linux interfaces also provide packet I/O functionality and support applications in control plane transmission (sending and receiving).

Mapping the CPU Port to a Linux Interface

OS10 creates a dedicated Linux interface (`npu0`) that maps to the CPU port. You can use this to configure CoPP queue rates by specifying `npu0` as port in the QoS CPS API.

Physical Ports

By default, the status of a physical port is administratively down. Each interface has its own MAC address that is reserved and derived from the system MAC address.

Fanout (4x10G) Interfaces

Using a breakout cable, you can split a 40GbE physical port into four (quad) 10SFP+ ports (if supported by the NPU). Each 4x10G port is represented by a Linux interface with a fanout field in the interface name that identifies a 4x10G port.

In OS10, fanout interfaces are configured by using the `os10-config-fanout` script. This script allows you to fanout a 40GbE port or disable the fanned-out 4x10G configuration and return the physical port to 40G operation.

Script syntax: `os10-config-fanout` [*linux-interface*] [`true|false`]

- `true` enables 4x10G fanout on a 40GbE port.
- `false` disables 4x10G fanout on a 40GbE port.

```
$os10-config-fanout e101-005-0 true
Key: 1.20.1310766.1310754.1310755.1310756.1310757.
base-port/physical/unit-id = 0
base-port/physical/phy-media = 1
base-port/physical/front-panel-number = 0
base-port/physical/loopback = 0
base-port/physical/hardware-port-id = 45
base-port/physical/npu-id = 0
base-port/physical/fanout-mode = 4
base-port/physical/breakout-capabilities  =  4,2
base-port/physical/port-id = 45
base-port/physical/slot-id = 0
Deleting.. e101-005-0
Completed...

Creating interface e101-005-1
Creating interface e101-005-2
Creating interface e101-005-3
Creating interface e101-005-4
Successfully created interfaces...
```

Configure an Interface using Linux commands

Use standard Linux commands (eg: `ip route)` to configure physical interface parameters.

Set the MTU

```
$ip link set dev e101-002-0 mtu 1400

$ip link show e101-002-0

17: e101-002-0: <BROADCAST,MULTICAST> mtu 1400 qdisc noop state DOWN mode DEFAULT group default qlen 500
   link/ether 90:b1:1c:f4:ab:f2 brd ff:ff:ff:ff:ff:ff
   alias NAS## 0 29
```

Configure Layer 3 IPv4 and IPv6 addresses

```
$ip -6 addr add 2000::1/64 dev e101-001-0
$ip addr add 10.1.1.1/24 dev e101-001-0
$ip addr show e101-001-0

16: e101-001-0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 500
   link/ether 90:b1:1c:f4:ab:ee brd ff:ff:ff:ff:ff:ff
   inet 10.1.1.1/24 scope global e101-001-0
      valid_lft forever preferred_lft forever
   inet6 2000::1/64 scope global tentative
      valid_lft forever preferred_lft forever
```

Refer to CPS Application Examples for information about programming using the CPS API.

LAG Ports (Port Channel)/Bond Interfaces

A port channel/link aggregation group (LAGs - IEEE 802.3ad) corresponds to a Linux bond interface, which aggregates multiple physical interfaces into one virtual interface for load-balancing and link fail-overs.

A LAG and a bond interface both refer to a link aggregation group. The term `LAG` is used to refer to an NPU configuration; the term `bond interface` refers to a Linux configuration.

NOTE: A Linux bond interface must be up before you add member ports.

Create a Bond Interface (LAG)

```
$ip link add bond1 type bond mode balance-rr miimon 50
```

Bring up a Bond Interface

```
$ip link set dev bond1 up
$ip link show|grep bond1
50: bond1: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN mode DEFAULT group
default
```

Add a Port to a Bond Interface

```
$ip link set e101-010-0 master bond1
$ip link show | grep bond1
12: e101-010-0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond1 state UP mode DEFAULT
group default qlen 500
50: bond1: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
$
```

Configure an IP address on a bond interface

```
$ip addr add 20.1.1.1/24 dev bond1
$ifconfig bond1
bond1    Link encap:Ethernet  HWaddr 90:b1:1c:f4:9d:60
         inet addr:20.1.1.1  Bcast:0.0.0.0  Mask:255.255.255.0
         inet6 addr: fe80::8480:fcff:fe2f:d93b/64 Scope:Link
         UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
         RX packets:66 errors:0 dropped:1 overruns:0 frame:0
         TX packets:77 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:0
         RX bytes:4224 (4.1 KiB)  TX bytes:15648 (15.2 KiB)
```

Delete a port from a bond interface

```
$ip link set e101-010-0 nomaster
$ip link show | grep bond1
49: bond1: <NO-CARRIER,BROADCAST,MULTICAST,MASTER,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group
default
```

Delete a bond interface

```
$ip link delete bond1
$ip link show | grep bond1
```

For more information about how to use Linux bond interfaces, go to https://www.kernel.org/doc/Documentation/networking/bonding.txt.

Refer to CPS Application Examples for information about programming using the CPS API.

Virtual LAN (VLAN)/Bridge Interfaces

VLANs define broadcast domains in a Layer 2 network. In Linux, each VLAN is modeled as a separate Linux bridge instance. Refer to VLAN Bridging for information on creating the bridge instance and how to add/delete member ports.

Each Bridge instance is exposed as a Linux interface. A Linux bridge interface uses the MAC address from the first port added as its member port. The bridge interface is operationally up when at least one of its member interfaces is operationally up. You can assign IP addresses to multiple Bridge interfaces create an inter-VLAN routing domain.

Configure an IP address on a Bridge Interface

```
$brctl show

bridge name     bridge id               STP enabled     interfaces
br100           8000.90b11cf49d3c       no              e101-001-0.100

$ip addr add 100.1.1.1 dev br100
$ifconfig br100
br100     Link encap:Ethernet  HWaddr 90:b1:1c:f4:9d:3c
         inet addr:100.1.1.1  Bcast:0.0.0.0  Mask:255.255.255.255
         inet6 addr: fe80::92b1:1cff:fef4:9d3c/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:0
         RX bytes:0 (0.0 B)  TX bytes:738 (738.0 B)
```

Layer 2 Features

This section describes OS10 support for configuring Layer 2 features using Linux commands, including VLAN Bridging, Spanning Tree and LLDP. You can also write applications that access CPS APIs to configure Layer 2 features.

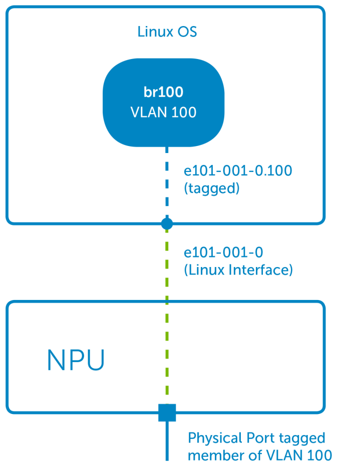
VLAN Bridging

OS10 supports Layer 2 VLAN Bridging by modeling each VLAN as a separate Linux Bridge instance. Each physical or LAG port that is a VLAN member is modeled by adding its corresponding Linux interface to the Bridge Instance.

This section describes how to configure Layer 2 bridging features using Linux bridging commands. Refer to the CPS Application Examples for examples of how to create a VLAN and assign members using the CPS API.

Create a VLAN and Add a Tagged Member

Figure 10 – Create VLAN and add Tagged Member



Creating a VLAN using the Linux Bridge command is a two-step process: create the bridge instance, then add a tagged member to the new bridge instance. OS10 determines the VLAN ID associated with each bridge instance using the VLAN ID of the first tagged member port assigned to the bridge instance. The VLAN is created only after you add the first tagged member to the bridge.

1. Create a Bridge Instance for the VLAN.

```
$brctl addbr br100
```

In the example, `br100` is the name of the Bridge Instance used to model a VLAN. Note that OS10 does not derive the VLAN ID from the name.

2. Add a tagged port to the VLAN.

Ensure that the Linux interface mapped to the port being added to the Vlan does not have an IP address.

```
$ifconfig e101-001-0
e101-001-0 Link encap:Ethernet  HWaddr 90:b1:1c:f4:9d:3c
         inet addr:1.1.1.1  Bcast:1.1.1.255  Mask:255.255.255.0
         BROADCAST MULTICAST  MTU:1500  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:2221 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:500
         RX bytes:0 (0.0 B)  TX bytes:523446 (511.1 KiB)
```

If the interface already has an IP address, remove the IP address before continuing to the next step.

```
$ip addr flush dev e101-001-0
```

3. Create a tagged virtual link
A Linux interface can only belong to a single bridge instance. To add the same interface to multiple VLAN domains, create a separate Linux virtual link for each VLAN in which the port is a member.
Enter the `ip link add` command to create a virtual tagged link for the `e101-001-0` Linux interface in VLAN 100. The `.100` suffix used in the Linux interface name indicates that the interface is VLAN tagged.

```
$ip link add link e101-001-0 name e101-001-0.100 type vlan id 100
```

4. Add the tagged virtual link to the VLAN.
Enter the `brctl addif` command to add the newly created virtual link to the Linux bridge instance created in Step 1. OS10 creates the VLAN and adds the physical port mapped to the e101-001-0 Linux interface as a tagged member of the VLAN.

```
$brctl addif br100 e101-001-0.100
```

5. Verify the VLAN configuration.

```
$brctl show
bridge name     bridge id               STP enabled     interfaces
br100           8000.90b11cf49d3c       no              e101-001-0.100
```

Add an Untagged Member to a VLAN

You can add a Linux interface directly to a Linux bridge without creating a separate VLAN-specific virtual link. Note that in this example the interface does not have the `.100` suffix, which means the interface is VLAN untagged.

```
$brctl addif br100 e101-002-0
```

A physical port can be an untagged member of only a single VLAN. However, a physical port can be added as a tagged member of multiple VLANs.

```
$brctl show
bridge name     bridge id               STP enabled     interfaces
br100           8000.90b11cf49d3c       no              e101-001-0.100
                                                        e101-002-0
```

Add a Port to Multiple VLAN Domains

To add the same port to a second VLAN, create another VLAN tagged virtual link on the e101-001-0 interface and add the new virtual link to the Linux bridge instance.

```
$brctl addbr br200

$ip link add link e101-001-0 name e101-001-0.200 type vlan id 200

$brctl addif br200 e101-001-0.200
```

NOTE: All the interfaces in a bridging instance must be either untagged or have the same tagged VLAN ID. For example, e101-001-0.100 and e101-002-0.200 should not be placed in the same Linux bridge instance.

```
$brctl show
bridge name     bridge id               STP enabled     interfaces
br100           8000.90b11cf49d3c       no              e101-001-0.100
                                                       e101-002-0
br200           8000.90b11cf49d3c       no              e101-001-0.200
```

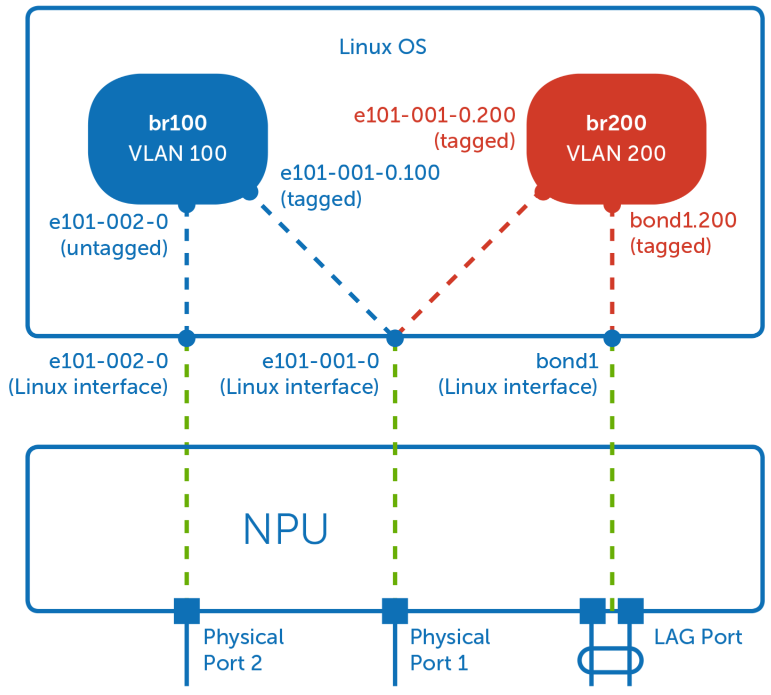
Add a Linux Bond Interface to a VLAN Domain

In the following command, bond1 is a Linux interface that maps to a LAG port in the NPU. Use this command to add the LAG port to the VLAN associated with the Bridge instance in the NPU.

```
$ip link add link bond1 name bond1.200 type vlan id 200

$brctl addif br200 bond1.200
```

The following illustration shows the result of these commands.

Figure 11 – Result of all VLAN Configurations



Remove a VLAN Member from a VLAN

To remove a port member from a VLAN, remove the member from the bridging instance.

```
$brctl delif br200 e101-001-0.100
```

Delete a VLAN

Delete a VLAN by deleting the bridging instance.

```
$brctl delbr br200
```

Spanning Tree

OS10 supports Spanning Tree provisioning using the CPS API, including:

- Create a new Spanning Tree group
- Add VLANs to a Spanning Tree group
- Remove VLANs from a Spanning Tree group
- Change the STP state of ports mapped to a Spanning Tree group
- Delete a Spanning Tree group

This enables OS10 users to run any of the Spanning Tree protocols such as STP, RSTP, PVST, and MSTP.

Refer to the `dell-base-stp.yang` for the STP parameters supported in OS10. Refer to Programmability for information on how to configure OS10 using YANG models and the CPS API.

Linux STP does not support the concept of Spanning Tree Group. In Linux, Spanning Tree can be enabled independently in each Bridge instance. OS10 internally treats this as a separate Spanning Tree Group for each VLAN.

Configure Spanning Tree in a Linux Bridge

To enable a Spanning Tree protocol in a Linux bridge:

1. Create a VLAN in the Linux bridge (see VLAN Bridging).
2. Enable STP on the bridge.

```
$brctl stp br100 on
$brctl show br100
bridge name     bridge id               STP enabled     interfaces
br100           8000.90b11cf4a918       yes             e101-001-0.100
```

```
$brctl showstp br100
br100
bridge id                8000.90b11cf4a918
designated root          8000.90b11cf4a918
root port                    0                   path cost                   0
```

```
max age                  20.00               bridge max age          20.00
hello time                2.00               bridge hello time        2.00
forward delay            15.00               bridge forward delay    15.00
ageing time             300.00
hello timer               0.00               tcn timer                0.00
topology change timer     0.00               gc timer                 0.00
flags


e101-001-0.100 (1)
port id                 8001                 state                disabled
designated root         8000.90b11cf4a918    path cost               100
designated bridge       8000.90b11cf4a918    message age timer        0.00
designated port         8001                 forward delay timer      0.00
designated cost            0                 hold timer               0.00
flags
```

Disable STP on a Linux Bridge

```
$brctl stp br100 off
$brctl show br100
bridge name     bridge id               STP enabled     interfaces
br100           8000.90b11cf4a918       no              e101-001-0.100
```

In the preceding example, when STP is enabled in a Linux bridge which has tagged VLAN Interfaces, OS10 creates a new Spanning Tree group and associates the VLAN ID of the bridge with the newly created Spanning Tree group. When you delete a bridge from the Linux kernel, the corresponding Spanning Tree group is deleted.

NOTE: If a Linux bridge contains only untagged ports, OS10 does not support the Spanning Tree Protocol on the bridge.

CAUTION: OS10 does not support RSTP, MSTP, and RPVST in a Linux bridge due to a Linux kernel limitation. STP is not supported on a bridge which has multiple member interfaces with different VLAN IDs.

LLDP

OS10 supports the operation of the Linux LLDP daemon on Linux interfaces. The following is a sample output from the Linux LLDP daemon.

```
$lldpcli show neighbors
-------------------------------------------------------------------------------
LLDP neighbors:
-------------------------------------------------------------------------------
Interface:    e101-003-0, via: LLDP, RID: 3, Time: 0 day, 01:17:18
  Chassis:
    ChassisID:    mac 90:b1:1c:f4:9d:3b
    SysName:      OS10
    Capability:   Repeater, on
    Capability:   Bridge, on
    Capability:   Router, on
    Port:
    PortID:       ifalias ethernet1/1/3
-------------------------------------------------------------------------------
```

MAC Address Forwarding Database (FDB)

OS10 provides a CPS data model for configuring and managing the MAC forwarding database using the CPS API. Refer to CPS Application Examples for an example of how to access/configure the MAC Address FDB.

OS10 does not support Linux commands to configure the MAC Address FDB.

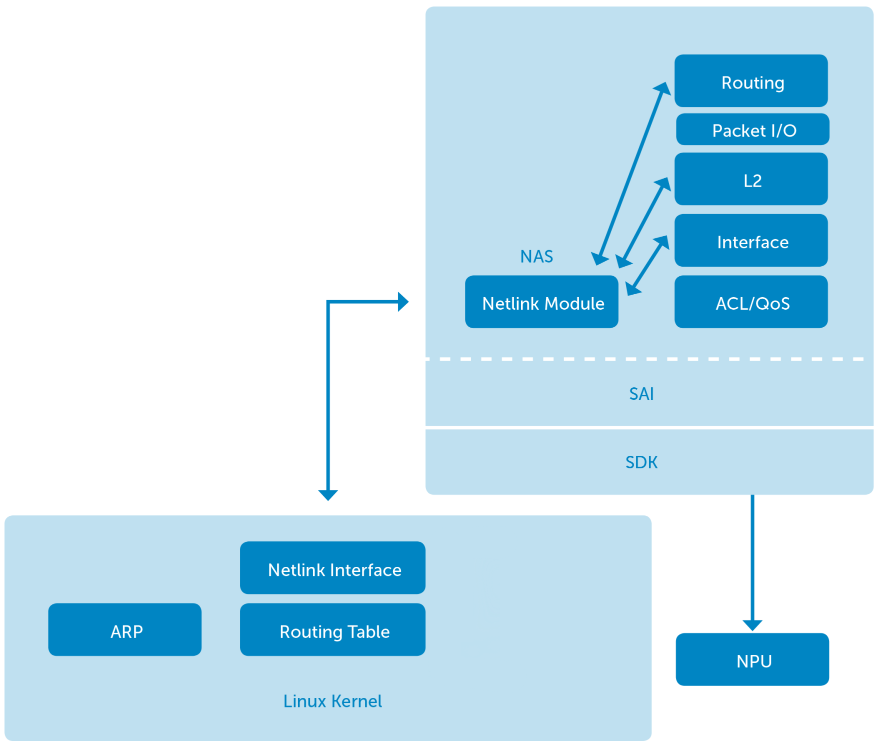
Layer 3 Features

Overview

This section describes the configuration of Layer 3 unicast routing to provision the NPU. OS10 supports unicast routing over Linux interfaces using routes configured in the Linux kernel routing table. Applications can also use the CPS API to configure routes.

The OS10 routing subsystem manages the forwarding information base. The routing subsystem programs routes with resolved next hops using ARP/Neighbor table entries received from the Linux kernel.

Figure 12 – OS10 Routing Subsystem



IPv4 Routing

Use the `ip route` command to create a route. A routing table entry consists of a destination IP address prefix and at least one next-hop address or a Linux interface.

Configure a Static Route

```
$ip route show
default dev eth0  scope link
3.3.3.0/24 dev e101-003-0  proto kernel  scope link  src 3.3.3.1
$ip route add 11.10.10.0/24  dev e101-003-0
$ip route show
default dev eth0  scope link
3.3.3.0/24 dev e101-003-0  proto kernel  scope link  src 3.3.3.1
11.10.10.0/24 dev e101-003-0  scope link
```

Configure a Static Route with a Next Hop

```
$ip route add 30.30.30.0/24 via 3.3.3.3
$ip route show
default dev eth0  scope link
3.3.3.0/24 dev e101-003-0   proto kernel  scope link  src 3.3.3.1
30.30.30.0/24 via 3.3.3.3 dev e101-003-0
```

Delete a Static Route

```
$ip route delete 11.10.10.0/24
$ip route show
default dev eth0  scope link
3.3.3.0/24 dev e101-003-0  proto kernel  scope link  src 3.3.3.1
$
```

To add a persistent static route that is saved after a reboot, configure the route in the /etc/network/interfaces configuration file.

For information about how to configure Linux routing, see http://linux-ip.net/html/tools-ip-route.html.

IPv6 Routing

Use the `ip -6` option to add, delete, or modify the IPv6 routes and nexthops in the IPv6 routing table.

```
$ip -6 route  add 5::5/64 via 3::3
$ip -6 route show
3::/64 dev e101-003-0  proto kernel  metric 256
5::/64 via 3::3 dev e101-003-0  metric 1024
```

Use the `ip monitor` command to debug and troubleshoot IPv6 routing.

```
$ip monitor
30.30.30.0/24 via 3.3.3.3 dev e00-3
3::/64 via 3::3 dev e101-003-0  metric 1024
5::/64 via 3::3 dev e101-003-0  metric 1024
```

ARP and Neighbor Tables

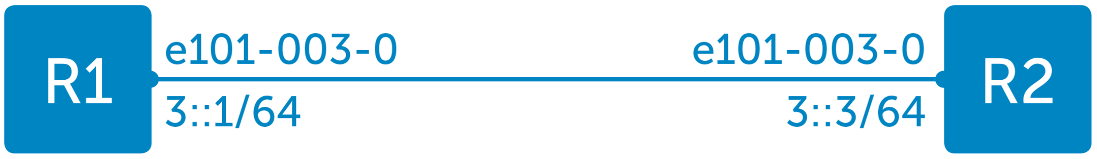
The ARP and Neighbor tables entries are used to resolve the adjacencies using the host IP address-to-MAC address binding. In Linux, the ARP table is used for IPv4 routing; the Neighbor table is used for IPv6 routing.

Use the `arp` or the `ip neighbor` command to configure and display kernel ARP table entries.

```
$arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
3.3.3.4                  ether   90:b1:1c:f4:9d:44   C                     e101-003-0
```

Use the `ip -6 neighbor` command to configure and display the IPv6 neighbor table.

Figure 13 – Simple IPv6 Routing Topology



Configure an IPv6 Address

```
$ifconfig e101-003-0  inet6 add 3::1/64
$ifconfig e101-003-0
e101-003-0 Link encap:Ethernet  HWaddr 90:b1:1c:f4:a8:ea
         inet addr:3.3.3.1  Bcast:3.3.3.255  Mask:255.255.255.0
         inet6 addr: 3::1/64 Scope:Global
         inet6 addr: fe80::92b1:1cff:fef4:a8ea/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:532 errors:0 dropped:0 overruns:0 frame:0
         TX packets:173 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:500
         RX bytes:46451 (45.3 KiB)  TX bytes:25650 (25.0 KiB)
```

Display the Neighbor Table

```
$ip -6 neighbor show
3::3 dev e101-003-0  lladdr 90:b1:1c:f4:9d:44 router REACHABLE
```

Ping an IPv6 Neighbor and Verify Using tcpdump

```
$ping6 3::3
PING 3::3(3::3) 56 data bytes
64 bytes from 3::3: icmp_seq=1 ttl=64 time=1.74 ms
$tcpdump -i e101-003-0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on e101-003-0, link-type EN10MB (Ethernet), capture size 262144 bytes
04:30:17.053115 IP6 3::1 > 3::3: ICMP6, echo request, seq 8, length 64
```

Equal Cost Multipath (ECMP)

The Linux networking stack supports Equal Cost Multipath by adding multiple nexthops to the route.

Configure an ECMP Route

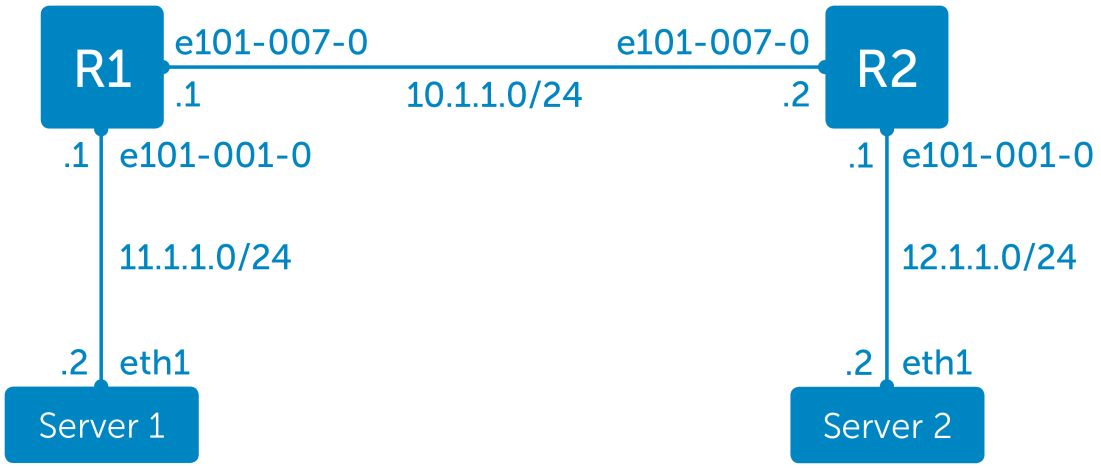
```
$ip route add 40.40.40.0/24 nexthop via 3.3.3.6 nexthop via 4.4.4.7
$ip route show
default dev eth0  scope link
3.3.3.0/24 dev e101-003-0  proto kernel  scope link  src 3.3.3.1
40.40.40.0/24
       nexthop via 3.3.3.6  dev e101-003-0 weight 1
       nexthop via 4.4.4.7  dev e101-004-0 weight 1
```

NOTE: In OS10, Linux kernel provides limited support for IPv6 multipath routing.

Example: L3 Routing Topology

IPV4 Topology

Figure 14 – Sample IPv4 Routing Topology



When you configure an IP address, you can use any Linux utility command, such as `ip addr add` or `ifconfig` to configure an interface.

Configure an IP Address on R1

```
$ip addr add 10.1.1.1/24 dev e101-007-0
$ip addr add 11.1.1.1/24 dev e101-001-0
```

Configure an IP Address on R2

```
$ip addr add 10.1.1.2/24 dev e101-007-0
$ip addr add 12.1.1.1/24 dev e101-001-0
```

Verify the IP Address Configuration on R1

```
$ip addr show e101-007-0
16: e101-007-0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 500
   link/ether 74:e6:e2:f6:af:87 brd ff:ff:ff:ff:ff:ff
   inet 10.1.1.1/24 scope global e101-007-0
      valid_lft forever preferred_lft forever
   inet6 fe80::76e6:e2ff:fef6:af87/64 scope link
      valid_lft forever preferred_lft forever
$ip addr show e101-001-0
10: e101-001-0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 500
   link/ether 74:e6:e2:f6:af:81 brd ff:ff:ff:ff:ff:ff
   inet 11.1.1.1/24 scope global e101-001-0
      valid_lft forever preferred_lft forever
   inet6 fe80::76e6:e2ff:fef6:af81/64 scope link
      valid_lft forever preferred_lft forever
```

Verify the IP Address Configuration on R2

```
$ip addr show e101-007-0
16: e101-007-0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 500
   link/ether 74:e6:e2:f6:ba:87 brd ff:ff:ff:ff:ff:ff
   inet 10.1.1.2/24 scope global e101-007-0
      valid_lft forever preferred_lft forever
   inet6 fe80::76e6:e2ff:fef6:ba87/64 scope link
      valid_lft forever preferred_lft forever
$ip addr show e101-001-0
```

```
10: e101-001-0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 500
   link/ether 74:e6:e2:f6:ba:81 brd ff:ff:ff:ff:ff:ff
   inet 12.1.1.1/24 scope global e101-001-0
      valid_lft forever preferred_lft forever
   inet6 fe80::76e6:e2ff:fef6:ba81/64 scope link
      valid_lft forever preferred_lft forever
```

Enable the Interfaces on R1 and R2

```
$ip link set dev e101-007-0 up  $ip link set dev e101-001-0 up
```

Configure Static Route to a Server

On R1:

```
$ip route add 12.1.1.0/24 via 10.1.1.2
```

On R2:

```
$ip route add 11.1.1.0/24 via 10.1.1.1
```

Ping a Neighbor Route and Server (Server 2) from R1

```
$ping 11.1.1.2
PING 11.1.1.2 (11.1.1.2) 56(84) bytes of data.
64 bytes from 11.1.1.2: icmp_seq=1 ttl=64 time=0.922 ms
^C
--- 11.1.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.922/0.922/0.922/0.000 ms
$ping 10.1.1.2
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=0.709 ms
^C
--- 10.1.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.709/0.709/0.709/0.000 ms
```

Display the ARP table on R1 and R2

On R1:

```
$arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
11.1.1.2                 ether   00:00:00:1d:9a:bd   C                     e101-001-0
10.1.1.2                 ether   74:e6:e2:f6:ba:87   C                     e101-007-0
```

On R2:

```
$arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.1.1.1                 ether   74:e6:e2:f6:af:87   C                     e101-007-0
12.1.1.2                 ether   00:00:00:1d:9a:be   C                     e101-001-0
```

CPS Support

Refer to CPS Application Examples for programming routes using the CPS API.

Dynamic Routing

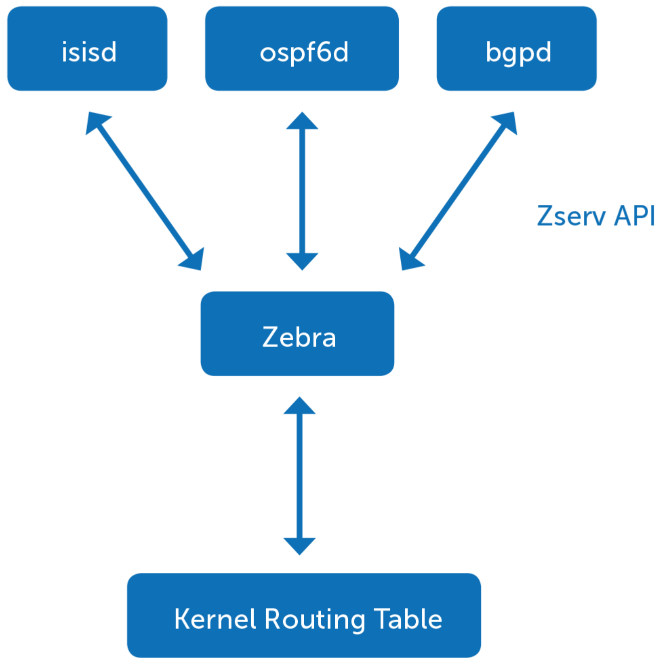
In OS10, configure BGP and OSPF using open source routing stacks such as Quagga, Bird, and other third-party applications.

Quagga Routing

Quagga is an open source routing application that provides OSPFv2,OSPFv3,RIPv1 and v2, RIPng and BGP-4 functionality.

The Quagga architecture consists of a core daemon zebra, which acts as an abstraction layer to the underlying Linux kernel and presents a Zserv API over a Unix or TCP stream to Quagga clients. The Zserv clients implement a routing protocol and communicate routing updates to the zebra daemon.

Figure 15 – Quagga Routing



Install Quagga

To install Quagga on OS10, configure an IP address on the management port (eth0) and verify that the switch can reach the package server using the `ping` command.

Install Quagga by entering the `apt-get` command.

```
$apt-get install -y quagga
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
 quagga
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 1217 kB of archives.
After this operation, 6323 kB of additional disk space will be used.
Get:1 http://10.11.56.31/debian/ jessie/main quagga amd64 0.99.23.1-1 [1217 kB]
Fetched 1217 kB in 0s (12.2 MB/s)
Preconfiguring packages ...
```

```
Selecting previously unselected package quagga.
(Reading database ... 46597 files and directories currently installed.)
Preparing to unpack .../quagga_0.99.23.1-1_amd64.deb ...
Unpacking quagga (0.99.23.1-1) ...
Processing triggers for systemd (215-17) ...
Processing triggers for man-db (2.7.0.2-5) ...
Setting up quagga (0.99.23.1-1) ...
Processing triggers for systemd (215-17) ...
Processing triggers for libc-bin (2.19-18) ...
```

When you install Quagga on OS10, it is stored in the /etc/quagga directory. By default, Quagga daemons and the `debian.conf` file are stored in the same directory.

Configure Quagga

By default, all routing protocol daemons installed with Quagga are disabled. You must enable the zebra daemon to install the routes in kernel routing table.

To configure Quagga:

1. Enter the `vim` command to open the `daemons` file for editing and change the daemon status to `yes.`

```
$vim /etc/quagga/daemons


zebra=yes
bgpd=yes
ospfd=no
ospf6d=no
ripd=no
ripngd=no
isisd=no
babeld=no
```

2. Create the `vtysh.conf` and `Quagga.conf` configuration files.

```
$cp /usr/share/doc/quagga/examples/vtysh.conf.sample /etc/quagga/vtysh.conf  $touch /etc/quagga/Quagga.conf
```

3. Restart the Quagga service using the `service quagga restart` Linux utility.

```
$/etc/quagga# service quagga restart
```

4. Enter the `service quagga status` command to display the status of Quagga protocol daemons.

```
$/etc/quagga# service quagga status
? quagga.service - LSB: start and stop the Quagga routing suite
  Loaded: loaded (/etc/init.d/quagga)
  Active: active (running) since Tue 2016-02-16 17:47:25 UTC; 4s ago
 Process: 5078 ExecStop=/etc/init.d/quagga stop (code=exited, status=0/SUCCESS)
 Process: 5097 ExecStart=/etc/init.d/quagga start (code=exited, status=0/SUCCESS)
  CGroup: /system.slice/quagga.service
          ??5111 /usr/lib/quagga/zebra --daemon -A 127.0.0.1
          ??5115 /usr/lib/quagga/bgpd --daemon -A 127.0.0.1
          ??5121 /usr/lib/quagga/watchquagga --daemon zebra bgpd

Feb 16 17:47:25 OS10 quagga[5097]: Loading capability module if not yet done.
Feb 16 17:47:25 OS10 quagga[5097]: Starting Quagga daemons (prio:10): zebra...d.
Feb 16 17:47:25 OS10 quagga[5097]: Starting Quagga monitor daemon: watchquagga.
Feb 16 17:47:25 OS10 watchquagga[5121]: watchquagga 0.99.23.1 watching [zebr...]
Feb 16 17:47:25 OS10 systemd[1]: Started LSB: start and stop the Quagga rou...e.
Feb 16 17:47:25 OS10 watchquagga[5121]: bgpd state -> up : connect succeeded
Feb 16 17:47:25 OS10 watchquagga[5121]: zebra state -> up : connect succeeded
Hint: Some lines were ellipsized, use -l to show in full
```

5. Enable the `service integrated-vtysh-config` option in the `vtysh.conf` file by deleting the !.

```
$/etc/quagga/vtysh.conf
!
! Sample
!
service integrated-vtysh-config
hostname quagga-router
```

```
username root nopassword
!
```

6. Add the`#echo VTYSH_PAGER=more` statement in the `/etc/environment` file to enable paging.

```
$echo VTYSH_PAGER=more > /etc/environment
```

7. Source the environment file before you access the Quagga shell.

```
$source /etc/environment
```

8. Access the Quagga shell.

```
$vtysh
Hello, this is Quagga (version 0.99.23.1)
Copyright 1996-2005 Kunihiro Ishiguro, et al.
OS10#
```

9. Save the configuration changes to the `Quagga.conf`file by entering the `write memory` command in the Quagga shell.

```
OS10# write memory
Building Configuration...
  Integrated configuration saved to /etc/quagga/Quagga.conf
  [OK]
```

10. Refer to Use Case: BGP Routing using Quagga for information about how to configure Quagga.

Monitoring

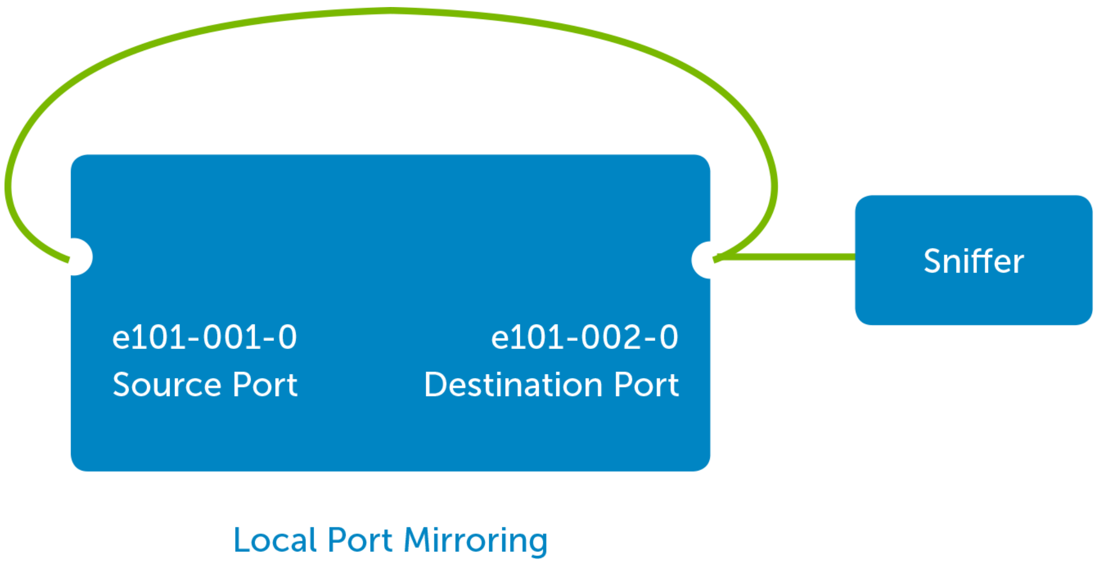
OS10 supports network monitoring features, such as sFlow and mirroring, to monitor and capture network traffic in the system. It also provides tools to collect port and VLAN statistics and port media information.

Mirroring

Mirroring enables the copying of packets from a mirroring source port to a mirroring destination port. OS10 supports the following types of mirroring:

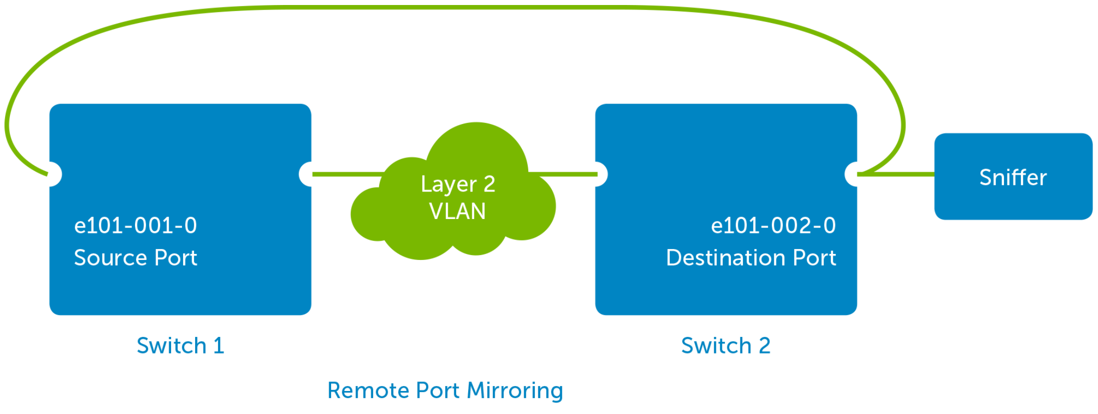
- Local port mirroring - Packets are forwarded from a source port to a destination port on the same system.

Figure 16 – Local Port Mirroring



- Remote port mirroring - mirrored packets are forwarded using a dedicated L2 VLAN.

Figure 17 – Remote Port Mirroring



OS10 provisions the following mirroring capabilities using the CPS API:

- Create a mirroring session
- Update a mirroring session
- Delete a mirroring session

See CPS Application Examples for programming using the CPS API.

NOTE: OS10 does not support Linux modeling of port mirroring.

sFlow

sFlow enables the sampling of incoming and outgoing packets on physical ports to monitor network traffic.

OS10 supports sFlow provisioning using the CPS API, including:

- Enable packet sampling on a physical port
- Disable packet sampling on a physical port
- Set the sampling interval rate
- Forward the sampled packet to an IP address/port.

See CPS Application Examples for programming using the CPS API.

NOTE: OS10 does not support Linux modeling of sFlow.

Statistics

os10-stat-show

OS10 provides the os10-stat-show script to display port statistics from Linux interfaces which map to the physical ports, and VLAN statistics.

os10-stat-show syntax

```
$os10-show-stats

os10-stat-show
          if_stat {iface_name} {filter_list}    - Get stats for all interfaces if no input provided
                                                - Get the statistics of given interface
                                                - filter_list is the filters if user wants
                                                  only specific statistics

          vlan_stat [vlan_ifname] {filter_list} - Get the statistics of given vlan interface name
                                                - filter_list is the filters if user wants
                                                  only specific statistics

           clear [iface_name]                    - Clear the interface statistics
```

Display Port Statistics

In the example, e101-001-0 is the Linux interface which maps to a physical port.

```
$os10-show-stats if_stat e101-001-0
Key:
base-stats/interface/ip/in-discards = 0
base-stats/interface/ether-octets = 13442942147
base-stats/interface/ether-out/pkts-64-octets = 0
base-stats/interface/ether-collisions = 0
base-stats/interface/ether-in/pkts-4096-to-9216-octets = 0
base-stats/interface/ether-in/pkts-1024-to-1518-octets = 0
base-stats/interface/ipv6-out-mcast-pkts = 0
base-stats/interface/if/in-octets = 0
base-stats/interface/ether-jabbers = 0
base-stats/interface/if-out-multicast-pkts = 36295
base-stats/interface/ether-out/pkts-256-to-511-octets = 0
```

```
base-stats/interface/if-out-errors = 0
base-stats/interface/ether-rx-no-errors = 0
base-stats/interface/ether/pkts-512-to-1023-octets = 0
base-stats/interface/ether/pkts-64-octets = 0
base-stats/interface/ether/pkts-1024-to-1518-octets = 13125220
base-stats/interface/ipv6/out-discards = 0
base-stats/interface/if/out-octets = 13442942147
base-stats/interface/ether-in/pkts-256-to-511-octets = 0
base-stats/interface/if/out-ucast-pkts = 13125220
base-stats/interface/ether-fragments = 0
base-stats/interface/ether-crc-align-errors = 0
base-stats/interface/ipv6-in-mcast-pkts = 0
base-stats/interface/ether-out/pkts-65-to-127-octets = 34014
base-stats/interface/if-out-broadcast-pkts = 0
base-stats/interface/ether-drop-events = 0
base-stats/interface/if/in-ucast-pkts = 0
base-stats/interface/ether-out/pkts-128-to-255-octets = 2281
base-stats/interface/ether-in/pkts-2048-to-4095-octets = 0
base-stats/interface/ether-tx-oversize-pkts = 0
base-stats/interface/ether/pkts-256-to-511-octets = 0
base-stats/interface/ether-multicast-pkts = 36295
base-stats/interface/ether-out/pkts-4096-to-9216-octets = 0
base-stats/interface/ether/pkts-128-to-255-octets = 2281
base-stats/interface/ether-in/pkts-128-to-255-octets = 0
base-stats/interface/time-stamp = 1455586392
base-stats/interface/ip-in-receives = 0
base-stats/interface/ether-rx-oversize-pkts = 0
base-stats/interface/ether-oversize-pkts = 0
base-stats/interface/ether-out/pkts-1024-to-1518-octets = 13125220
base-stats/interface/if-in-errors = 0
base-stats/interface/ether-out/pkts-512-to-1023-octets = 0
base-stats/interface/if/in-non-ucast-pkts = 0
base-stats/interface/ether-in/pkts-65-to-127-octets = 0
base-stats/interface/ether/pkts-65-to-127-octets = 34014
base-stats/interface/if/in-discards = 0
base-stats/interface/ipv6-in-receives = 0
base-stats/interface/if/out-non-ucast-pkts = 36295
base-stats/interface/ipv6/in-discards = 0
base-stats/interface/ether-tx-no-errors = 13161515
base-stats/interface/ether-broadcast-pkts = 0
base-stats/interface/if-in-broadcast-pkts = 0
base-stats/interface/if/out-discards = 0
base-stats/interface/ether-out/pkts-1519-to-2047-octets = 0
base-stats/interface/ether-out/pkts-2048-to-4095-octets = 0
base-stats/interface/ether-pkts = 13161515
base-stats/interface/ether-in/pkts-1519-to-2047-octets = 0
base-stats/interface/ether/pkts-4096-to-9216-octets = 0
base-stats/interface/ether-undersize-pkts = 0
base-stats/interface/if-in-unknown-protos = 0
```

```
base-stats/interface/if-out-qlen = 0
base-stats/interface/if-in-multicast-pkts = 0
base-stats/interface/ether-in/pkts-64-octets = 0
base-stats/interface/ether-in/pkts-512-to-1023-octets = 0
```

Display VLAN Statistics

In the example, br1 is the Linux bridge interface that maps to a VLAN.

```
$os10-show-stats vlan_stat br1
Key:
base-stats/vlan/time-stamp:  1455586573
base-stats/vlan/in-octets:  16381983058
base-stats/vlan/in-pkts:  16101053
base-stats/vlan/out-octets:  55146334258
base-stats/vlan/out-pkts:  67419926
```

Clear Interface Statistics

```
$ os10-show-stats clear e101-001-0
Success
```

NOTE: os10-show-stats does not support clearing VLAN statistics.

os10-ethtool

OS10 provides os10-ethtool to gather statistics and media information from a Linux interface which maps to a physical port.

os10-ethtool Syntax

```
$os10-ethtool -h
os10-ethtool -h|--help
os10-ethtool -v|--version
os10-ethtool -e|--eeprom-dump devname
os10-ethtool -s| --speed devname speed N [duplex half|full] [autoneg on|off]
os10-ethtool -S|--statistics devname
```

Display Port Statistics using os10-ethtool

In the example, os10-ethtool output is a subset of `os10-stats-show` for the same physical port interface.

```
$os10-ethtool -S e101-001-0
Statistics for interface e101-001-0
  Ether statistics:
    rx_bytes: 9185614848
    rx_no_errors: 0
    tx_no_errors: 9003181
    tx_total_collision: 0
    rx_undersize_packets: 0
    rx_jabbers: 0
    rx_fragments: 0
    rx_align_errors: 0
    rx_discards: 0
    rx_mcast_packets: 35445
    rx_bcast_packets: 0
    rx_oversize_packets: 0
    tx_oversize_packets: 0
    rx_64_byte_packets: 0
    rx_65_to_127_byte_packets: 0
    rx_128_to_255_byte_packets: 0
    rx_256_to_511_byte_packets: 0
    rx_512_to_1023_byte_packets: 0
    rx_1024_to_1518_byte_packets: 0
    rx_1519_to_2047_byte_packets: 0
    rx_2048_to_4095_byte_packets: 0
    rx_4096_to_9216_byte_packets: 0
    tx_64_byte_packets: 0
    tx_65_to_127_byte_packets: 33217
    tx_128_to_255_byte_packets: 2228
    tx_256_to_511_byte_packets: 0
    tx_512_to_1023_byte_packets: 0
    tx_1024_to_1518_byte_packets: 8967736
    tx_1519_to_2047_byte_packets: 0
    tx_2048_to_4095_byte_packets: 0
    tx_4096_to_9216_byte_packets: 0
```

Display Transceiver Information using os10-ethtool

```
$os10-ethtool -e e101-001-0
Show media info for e101-001-0
if_index is 17
Key: 2.19.1245389.1245248.1245249.1245250.
base-pas/media/rate-identifier = 0
```

```
base-pas/media/oper-status = 0
base-pas/media/category = 3
base-pas/media/voltage-state = 1
base-pas/media/bias-low-warning-threshold =
base-pas/media/vendor-pn = 568400002
base-pas/media/current-temperature = ??
base-pas/media/insertion-cnt = 0
base-pas/media/voltage-low-warning-threshold =
base-pas/media/cc_ext = 162
base-pas/media/length-om2 = 0
base-pas/media/length-om3 = 0
base-pas/media/rx-power-low-alarm-threshold =
base-pas/media/length-om1 = 0
base-pas/media/vendor-id = AP
base-pas/media/media-category/sfp-plus/br-max = 0
base-pas/media/connector = 33
base-pas/media/ext-transceiver = 0
base-pas/media/vendor-Specific = ffffffffffffff00000000000000000000000000000000000000000000000000
base-pas/media/media-category/sfp-plus/br-min = 0
base-pas/media/encoding = 0
base-pas/media/tx-power-high-warning-threshold =
base-pas/media/vendor-name = Amphenol
base-pas/media/rx-power-low-warning-threshold =
base-pas/media/slot = 1
base-pas/media/port = 1
base-pas/media/vendor-rev = 4700
base-pas/media/slot = 1
base-pas/media/port = 1
base-pas/media/tx-power-low-alarm-threshold =
base-pas/media/bias-low-alarm-threshold =
base-pas/media/capability = 6
base-pas/media/media-category/sfp-plus/sff-8472-compliance = 0
base-pas/media/diag-mon-type = 0
base-pas/media/temp-state = 1
base-pas/media/type = 43
base-pas/media/media-category/qsfp-plus/wavelength-tolerance = 0
base-pas/media/ext-identifier = 0
base-pas/media/temp-low-warning-threshold =
base-pas/media/voltage-high-warning-threshold =
base-pas/media/temp-high-alarm-threshold =
base-pas/media/length-sfm = 0
base-pas/media/rate-select-state = 0
base-pas/media/rx-power-measurement-type = 0
base-pas/media/wavelength = 0
base-pas/media/cc_base = 54
base-pas/media/temp-low-alarm-threshold =
base-pas/media/tx-power-low-warning-threshold =
base-pas/media/insertion-timestamp = 0
base-pas/media/current-voltage =
base-pas/media/bias-high-alarm-threshold =
base-pas/media/high-power-mode = 1
base-pas/media/br-nominal = 0
base-pas/media/options = 0
base-pas/media/rx-power-high-warning-threshold =
base-pas/media/date-code = 3131303632322000
base-pas/media/present = 1
base-pas/media/transceiver = 000000000000000205c
base-pas/media/length-cable = 2
base-pas/media/voltage-high-alarm-threshold =
base-pas/media/identifier = 12
base-pas/media/voltage-low-alarm-threshold =
base-pas/media/dell-qualified = 0
base-pas/media/length-sfm-km = 0
base-pas/media/rx-power-high-alarm-threshold =
base-pas/media/admin-status = 0
base-pas/media/serial-number = APF11240020140
base-pas/media/tx-power-high-alarm-threshold =
base-pas/media/temp-high-warning-threshold =
base-pas/media/bias-high-warning-threshold =
base-pas/media/enhanced-options = 0
base-pas/media/media-category/qsfp-plus/max-case-temp = 70
```

Access Control Lists (ACLs)

Access Control Lists (ACLs) are flexible, hardware-accelerated sets of rules used to match packets using packet header criteria and perform actions on the selected packets.

In OS10, you can configure an ACL on a physical port (NPU) only by using the CPS API; you cannot configure ACLs using Linux commands or an open source application.

Refer to the YANG model `dell-base-acl.yang` for the ACL parameters supported in OS10. Refer to CPS Application Examples for an example of how to configure ACLs using a YANG model and the CPS API.

OS10 supports following ACL features:

- Ingress and egress ACL rules
- Match packet header fields, including MAC address, Ethertype, IP address, IP protocol, TCP/ UDP port numbers, and In Port.
- Packet actions, including drop, trap/forward to CPU, redirect to port, change packet fields, and meter.
- Grouping ACL rules to enable multiple rule match for a single packet.

NOTE : OS10 does not support configuration of Access Control Lists using Linux commands.

Quality of Service (QoS)

OS10 supports QoS provisioning, including:

- Assigning packet to traffic classes using packet 802.1p, DSCP or more advanced ACL rules
- Marking
- Ingress rate policing using ACLs
- Mapping traffic classes to queues
- Egress Queue rate shaping
- WRED
- Hierarchical scheduling
- Egress Port-level shaping
- CoPP support for configuring CPU rate-limits

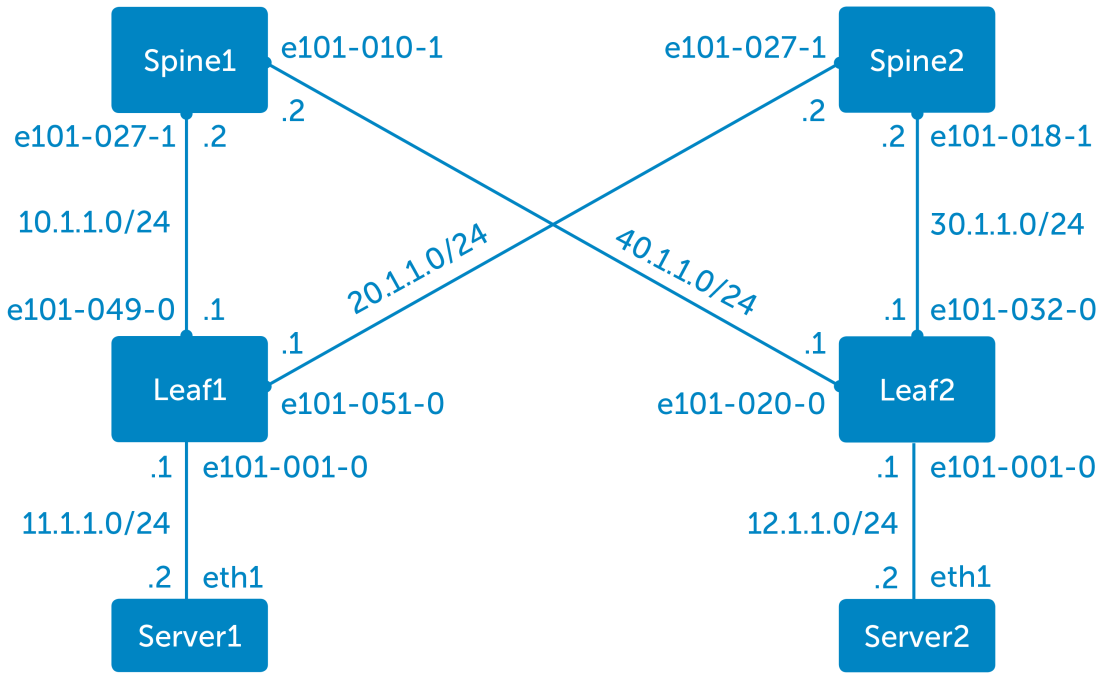
In OS10, you can configure QoS settings only by using the CPS API; you cannot configure QoS using Linux commands or an open source application. Refer to the YANG model `dell-base-qos.yang` for the QoS parameters supported in OS10. Refer to Programmability for information on how to configure OS10 using YANG models and the CPS API.

NOTE : OS10 does not support configuration of QoS parameters using Linux commands.

Use Case: BGP Routing Using Quagga

This use case describes how to configure BGP using Quagga in a CLOS (spine/leaf) network.

Figure 18 – Quagga BGP CLOS Network



The BGP topology used in this example (links, networks, nodes and their AS numbers) is described below.

Link	Network	Nodes in the Link	BGP AS number
Leaf1-to-Spine1	10.1.1.0/24	Leaf1	64501
		Spine1	64555
Leaf1-to-Spine2	20.1.1.0/24	Leaf1	64501
		Spine2	64555
Leaf2-to-Spine1	40.1.1.0/24	Leaf2	64502
		Spine1	64555
Leaf2-to-Spine2	30.1.1.0/24	Leaf2	64502
		Spine2	64555
Leaf1-to-Server1	11.1.1.0/24	Leaf1	64501
Leaf2-to-Server2	12.1.1.0/24	Leaf2	64502

Leaf1: Configure IP Addresses to Spine1, Spine2, and Server1

```
leaf1# conf t
leaf1(config)# int e101-049-0
leaf1(config-if)# ip address 10.1.1.1/24
leaf1(config-if)# no shutdown
leaf1(config-if)# exit

leaf1(config)# int e101-051-0
leaf1(config-if)# ip address 20.1.1.1/24
leaf1(config-if)# no shutdown
leaf1(config-if)# exit

leaf1(config)# int e101-001-0
leaf1(config-if)# ip address 11.1.1.1/24
leaf1(config-if)# no shut
```

Leaf2: Configure IP Addresses to Spine1, Spine2, and Server2

```
leaf2# configure t
leaf2(config)# int e101-032-0
leaf2(config-if)# ip address 30.1.1.1/24
leaf2(config-if)# no shut
leaf2(config-if)# exit

leaf2(config)# int e101-020-0
```

```
leaf2(config-if)# ip address 40.1.1.1/24
leaf2(config-if)# no shut
leaf2(config-if)# exit

leaf2(config)# int e101-001-0
leaf2(config-if)# ip address 12.1.1.1/24
leaf2(config-if)# no shut
```

Spine1: Configure IP Addresses to Leaf1 and Leaf2

```
spine1(config)# int e101-027-1
spine1(config-if)# ip address 10.1.1.2/24
spine1(config-if)# no shut
spine1(config-if)# exit

spine1(config)# int e101-010-1
spine1(config-if)# ip address 40.1.1.2/24
spine1(config-if)# no shut
```

Spine2: Configure IP Addresses to Leaf1 and Leaf2

```
spine2(config)# int e101-027-1
spine2(config-if)# ip address 20.1.1.2/24
spine2(config-if)# no shut
spine2(config-if)# exit

spine2(config)# int e101-018-1
spine2(config-if)# ip address 30.1.1.2/24
spine2(config-if)# no shutdown
spine2(config-if)# exit
```

Leaf1: Configure BGP to Spine1 and Spine2

```
leaf1(config)# router bgp 64501
leaf1(config-router)# neighbor 10.1.1.2 remote-as 64555
leaf1(config-router)# neighbor 20.1.1.2 remote-as 64555
leaf1(config-router)#network 10.1.1.0/24
leaf1(config-router)#network 20.1.1.0/24
leaf1(config-router)#network 11.1.1.0/24
```

Leaf2: Configure BGP to Spine1 and Spine2

```
leaf2(config)# router bgp 64502
leaf2(config-router)# neighbor 30.1.1.2 remote-as 64555
leaf2(config-router)# neighbor 40.1.1.2 remote-as 64555
leaf2(config-router)#network 12.1.1.0/24
leaf2(config-router)#network 30.1.1.0/24
leaf2(config-router)#network 40.1.1.0/24
```

Spine1: Configure BGP to Leaf1 and Leaf2

```
spine1# configure t
spine1(config)# router bgp 64555
spine1(config-router)# neighbor 10.1.1.1 remote-as 64501
spine1(config-router)# neighbor 40.1.1.1 remote-as 64502
spine1(config-router)# network 10.1.1.0/24
spine1(config-router)# network 40.1.1.0/24
```

Spine2: Configure BGP to Leaf1 and Leaf2

```
spine2# configure t
spine2(config)# router bgp 64555
spine2(config-router)# neighbor 30.1.1.1 remote-as 64502
spine2(config-router)# neighbor 20.1.1.1 remote-as 64501
spine2(config-router)# network 30.1.1.0/24
spine2(config-router)# network 20.1.1.0/24
```

Leaf1 and Leaf2: Configure ECMP

```
leaf1# configure t
leaf1(config)# router bgp 64501
leaf1(config)# maximum-paths 16

leaf2# configure t
leaf2(config)# router bgp 64502
leaf2(config)# maximum-paths 16
```

Leaf1 and Leaf2: Display BGP Neighbors

```
leaf1# show ip bgp sum
BGP router identifier 20.1.1.1, local AS number 64501
RIB entries 11, using 1232 bytes of memory
Peers 2, using 9136 bytes of memory

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.1.1.2        4 64555     196     201        0    0    0 02:39:02        4
20.1.1.2        4 64555     195     206        0    0    0 02:38:57        4

Total number of neighbors 2

leaf2# show ip bgp sum
BGP router identifier 40.1.1.1, local AS number 64502
RIB entries 11, using 1232 bytes of memory
Peers 2, using 9136 bytes of memory

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
30.1.1.2        4 64555     196     197        0    0    0 02:39:45        4
40.1.1.2        4 64555     192     204        0    0    0 02:39:42        4

Total number of neighbors 2
```

Spine1 and Spine2: Display BGP Neighbors

```
spine1# show ip bgp sum
BGP router identifier 40.1.1.2, local AS number 64555
RIB entries 11, using 1232 bytes of memory
Peers 2, using 9136 bytes of memory

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.1.1.1        4 64501     199     201        0    0    0 02:40:55        3
40.1.1.1        4 64502     202     198        0    0    0 02:41:01        3

Total number of neighbors 2

spine2# show ip bgp sum
BGP router identifier 30.1.1.2, local AS number 64555
RIB entries 11, using 1232 bytes of memory
Peers 2, using 9136 bytes of memory

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
20.1.1.1        4 64501     206     206        0    0    0 02:43:06        3
30.1.1.1        4 64502     197     203        0    0    0 02:43:20        3

Total number of neighbors 2
```

Leaf1 and Leaf2: Display Server Route as ECMP in Routing Table

```
leaf1# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

C>* 10.1.1.0/24 is directly connected, e101-049-0
C>* 11.1.1.0/24 is directly connected, e101-001-0
B>* 12.1.1.0/24 [20/0] via 10.1.1.2, e101-049-0, 02:44:45
  *                    via 20.1.1.2, e101-051-0, 02:44:45
C>* 13.1.1.0/24 is directly connected, e101-002-0
C>* 20.1.1.0/24 is directly connected, e101-051-0
B>* 30.1.1.0/24 [20/0] via 20.1.1.2, e101-051-0, 02:09:44
B>* 40.1.1.0/24 [20/0] via 10.1.1.2, e101-049-0, 02:11:50
C>* 127.0.0.0/8 is directly connected, lo


leaf2# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

B>* 10.1.1.0/24 [20/0] via 40.1.1.2, e101-020-0, 02:12:43
B>* 11.1.1.0/24 [20/0] via 30.1.1.2, e101-032-0, 02:45:14
  *                    via 40.1.1.2, e101-020-0, 02:45:14
C>* 12.1.1.0/24 is directly connected, e101-001-0
B>* 20.1.1.0/24 [20/0] via 30.1.1.2, e101-032-0, 02:10:16
C>* 30.1.1.0/24 is directly connected, e101-032-0
C>* 40.1.1.0/24 is directly connected, e101-020-0
C>* 127.0.0.0/8 is directly connected, lo
```

Leaf1 and Leaf2: Display Server Route as ECMP in Linux Routing Table

```
$ip route show
10.1.1.0/24 dev e101-049-0  proto kernel  scope link  src 10.1.1.1
11.1.1.0/24 dev e101-001-0  proto kernel  scope link  src 11.1.1.1
12.1.1.0/24  proto zebra
       nexthop via 10.1.1.2  dev e101-049-0 weight 1
       nexthop via 20.1.1.2  dev e101-051-0 weight 1
13.1.1.0/24 dev e101-002-0  proto kernel  scope link  src 13.1.1.1
20.1.1.0/24 dev e101-051-0  proto kernel  scope link  src 20.1.1.1
30.1.1.0/24 via 20.1.1.2 dev e101-051-0  proto zebra
40.1.1.0/24 via 10.1.1.2 dev e101-049-0  proto zebra

$ip route show
10.1.1.0/24 via 40.1.1.2 dev e101-020-0  proto zebra
11.1.1.0/24  proto zebra
       nexthop via 30.1.1.2  dev e101-032-0 weight 1
       nexthop via 40.1.1.2  dev e101-020-0 weight 1
12.1.1.0/24 dev e101-001-0  proto kernel  scope link  src 12.1.1.1
20.1.1.0/24 via 30.1.1.2 dev e101-032-0  proto zebra
30.1.1.0/24 dev e101-032-0  proto kernel  scope link  src 30.1.1.1
40.1.1.0/24 dev e101-020-0  proto kernel  scope link  src 40.1.1.1
```

Leaf1 and Leaf2: Display Server Route as ECMP in NPU Routing Table

```
$hshell -c 'l3 defip show'
Unit 0, Total Number of DEFIP entries: 16384
#     VRF     Net addr             Next Hop Mac        INTF MODID PORT PRIO CLASS HIT VLAN
4096  0        10.1.1.0/24          00:00:00:00:00:00 100002    0     0     0    0 y
4096  0        20.1.1.0/24          00:00:00:00:00:00 100002    0     0     0    0 y
4097  0        11.1.1.0/24          00:00:00:00:00:00 100002    0     0     0    0 y
4097  0        13.1.1.0/24          00:00:00:00:00:00 100002    0     0     0    0 n
4098  0        12.1.1.0/24          00:00:00:00:00:00 200000    0     0     0    0 n (ECMP)
4098  0        40.1.1.0/24          00:00:00:00:00:00 100004    0     0     0    0 n
4099  0        30.1.1.0/24          00:00:00:00:00:00 100005    0     0     0    0 n


$hshell -c 'l3 multipath show'

Multipath Egress Object 200000
Interfaces: 100004 100005
  Reference count: 1


$hshell -c 'l3 defip show'

Unit 0, Total Number of DEFIP entries: 16384
```

```
#     VRF     Net addr               Next Hop Mac        INTF MODID PORT PRIO CLASS HIT VLAN
4096  0        30.1.1.0/24          00:00:00:00:00:00 100002    0     0     0    0 y
4096  0        40.1.1.0/24          00:00:00:00:00:00 100002    0     0     0    0 y
4097  0        12.1.1.0/24          00:00:00:00:00:00 100002    0     0     0    0 y
4097  0        11.1.1.0/24          00:00:00:00:00:00 200000    0     0     0    0 n (ECMP)
4098  0        10.1.1.0/24          00:00:00:00:00:00 100004    0     0     0    0 n
4098  0        20.1.1.0/24          00:00:00:00:00:00 100005    0     0     0    0 n

$hshell -c 'l3 multipath show'

Multipath Egress Object 200000
Interfaces: 100004 100005
Reference count: 1
```

Use Case: CAM Optimization using CPS APIs

This use case describes how to optimize system resources, such as Content Addressable Memory (CAM), using the CPS APIs in OS10.

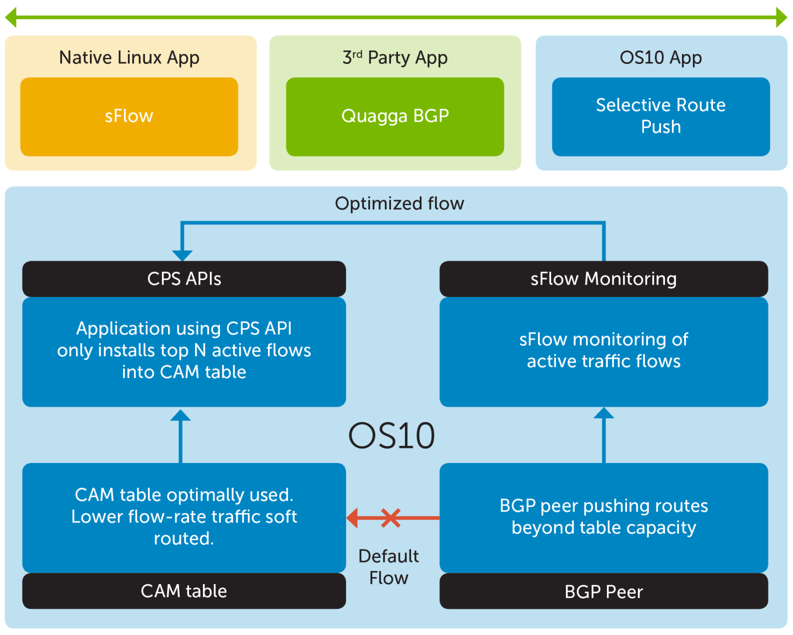
Top-of-the-rack (TOR) switches have a smaller CAM size and are usually installed in a data center network where the CAM resource requirement is not critical.

Without CAM optimization, all routes in the Forwarding Information Base (FIB) are sequentially installed into CAM, which may result in suboptimal utilization of CAM table space. Routes which are not active may be installed in the CAM table, whereas routes that are active and used by traffic flows may not be installed in the table and are software-routed instead.

Because only 2-3% of internet routes are active at a given time on a switch, by using CPS APIs, you can selectively add, delete, and update desired routes in CAM. This optimization allows you to use TOR switches at a WAN network edge.

The CAM optimization solution consists of an sFlow monitoring tool, a routing protocol such as BGP, CAM optimization application, and CPS APIs available in OS10.

Figure 19 – CAM Optimization Application


Native Linux App
sFlow
3rd Party App
Quagga BGP
OS10 App
Selective Route Push
Optimized flow
CPS APIs
Application using CPS API only installs top N active flows into CAM table
sFlow Monitoring
sFlow monitoring of active traffic flows
OS10
CAM table optimally used. Lower flow-rate traffic soft routed.
Default Flow
BGP peer pushing routes beyond table capacity
CAM table
BGP Peer

To optimize CAM using CPS APIs:

1. Use sFlow to determine the latest N active flows

Custom sFlow agent is configured using YANG model and CPS API to determine the latest N active routes on the system. See Monitoring for information about how to configure the sFlow agent on the system.

Use an Open Source sFlow collector, such as sflowRT, to capture the latest N flows. You can use a REST interface to communicate between the agent and the collector.

2. Configure a routing protocol

Existing routing stacks, such as Quagga and Bird, provide BGP, OSPF, and other routing protocols that can be configured on OS10. These stacks also provide the capability to block routes from getting installed in the CAM table. For CAM optimization, you must enable the feature that blocks routes.

You can also use CPS APIs to disable automatic route installation in the CAM table. See Programmability for programming using the CPS API.

Quagga BGP running on the switch provides routes with next hop information. Refer to Use Case: BGP Routing using Quagga for more information about how to configure Quagga. The Quagga routing stack peers with the neighboring router to obtain next-hop routing information.

3. Implement the route_collator

The route_collator is an application which you can write in any scripting language, such as Python. It combines the latest N flows (from Step 1) with routes with next-hop information (from Step 2) to determine the desired routes to be programmed on the switch. You can configure the route_collator to run at any frequency to determine the routes to include in CAM. A sample implementation of the route_collator is shown below:

```
if __name__ == "__main__" :
while True :

    getBgpRiBRoute = getBgpRouteFromDevice()                                                #RIB contents
    getTopFlowFromsFlow = top_Flows (topFlowCnt)                                            #Traffic Flow
contents
    getCollateRouteInfo = generatSflowOptimalPrefixes (getTopFlowFromsFlow, getBgpRiBRoute)     #TopN Prefix
Generator
    getCurrentCAMContent = getActualCamFromDevice()                                          #Current CAM
content using CPS
```

```
    routeDetail = routeFilter(getCollateRouteInfo, getCurrentCAMContent)                       #TopN Prefix
installation using CPS
    getNextCAMContent = getActualCamFromDevice( )
```

The `getBgpRouteFromDevice( )` function uses REST to get the next-hop routing information from the routing stack (such as Quagga and Bird).

The `top_Flows( )` function also uses REST to get the latest N flows from the sFlow collector.

The `generateSflowOptimalPrefixes( )` uses the latest N flows from top_Flows() to determine the best IP prefixes to serve hosts in the flows. A sample implementation of generateSflowOptimalPrefixes() using Python is shown below:

```
def generateSflowOptimalPrefixes(topSflowHosts, camPrefixiDict):
########## Test Data ############
  #topSflowHosts = ['1.1.1.1','2.2.2.2','3.3.3.3']
  #camPrefixiDict = {'1.1.1.0/24' : '10.1.1.1', '2.2.0.0/16' : '3.3.3.3'}
#################################
  sFlowOptimalPrefixes = { }

  for dest_ip in topSflowHosts :
      dest_ip = ipaddress.ip_address(dest_ip)
      for prefix, nhi in camPrefixiDict.items( ) :
          prefix = ipaddress.ip_network(prefix,strict=False)
          if dest_ip in prefix :
              sFlowOptimalPrefixes.update({str(prefix):nhi})

  return (sFlowOptimalPrefixes)
```

The getActualCamFromDevice() function uses a REST call to obtain the CAM contents with a CPS API. Refer to <Step 4. Install routes using CPS APIs on OS10 > for the CPS API to use.

The routeFilter() function requires two arguments:

- Desired CAM contents (output from the generateSflowOptimalPrefixes() call)
- Current CAM contents (output from the getActualCamFromDevice() call)

The routeFilter() function programs the switch with the desired routes using a REST call.

```
def routeFilter(desiredCAM, currentCAM):
    routeDelete={}
    routeUpdate={}
    routeAdd={}
    returnDictForDB ={}

    delete = set(currentCAM.keys()).difference(set(desiredCAM.keys()))    #Routes not Present in desiredCAM 
and in currentCAM
    update = set(desiredCAM.keys()).intersection(set(currentCAM.keys()))  #Route Present in desiredCAM and 
currentCAM
    add = set(desiredCAM.keys()).difference(set(currentCAM.keys()))       #Route Present in desiredCAM and not 
in currentCAM

    #---Delete---:
    for i in delete:
        routeDelete[i] = currentCAM[i]

    #Calling REST API, to update route into actual device
    deletePtr = requests.post(url, json.dumps(routeDelete), headers=headers)
    if deletePtr.ok :
        returnDeletePtrContent = ast.literal_eval(deletePtr.text)
        returnDictForDB.update({'routeDelete':returnDeletePtrContent})
    else :
        print ('Route Delete In CAM Failed')

    #---Update---:
    for i in update:
        routeUpdate[i] = desiredCAM[i]

    #Calling REST API, to update route into actual device
    updatePtr = requests.post(url, json.dumps(routeUpdate), headers=headers)
    if updatePtr.ok :
        returnUpdatePtrContent = ast.literal_eval(updatePtr.text)
       returnDictForDB.update({'routeUpdate':returnUpdatePtrContent})
    else :
        print ('Route Update In CAM Failed')

    #---Add---:
    for i in add:
        routeAdd[i] = desiredCAM[i]

    #Calling REST API, to update route into actual device
    addPtr = requests.post(url, json.dumps(routeAdd), headers=headers)
    if addPtr.ok :
        returnAddPtrContent = ast.literal_eval(addPtr.text)
        returnDictForDB.update({'routeAdd':returnAddPtrContent}
    else :
        print ('Route Add In CAM Failed')

    return returnDictForDB
```

4. Install routes using CPS APIs

The CPS API allows you to program desired routes in the CAM table. Refer to Example: Configuring a Route using Python in CPS Application Examples for configuring the routes using next hops or using the `cps-config-route` sample python script to get, add, delete, and update routes in CAM.

```
$ python cps-config-route.py
< Usage >
cps-config-route can be used to add/update/delete the route to the system

-h, --help: Show the option and usage
-a, --add : add new route
-d, --del : delete existing route
-i, --ip  : IP address that needs to be configured
   --ip6 : IP v6  address that needs to be configured
-n, --nh  : next-hop ip adress, needs ipv6 address if --ip6 used otherwise ipv4
           multiple ip address needs to supply as string separated by space
-u, --update: update existing route information
-s, --show: show existing route information
Example:
cps-config-route  --add --ip 192.168.10.2 --nh "127.0.0.1 127.0.0.2"
cps-config-route  --del --ip 192.168.10.2
cps-config-route  --update --ip 192.168.10.2 --nh "127.0.0.1 127.0.0.3"
cps-config-route  --show
cps-config-route  --show  --ip 192.168.10.2
$
```

Programmability

Overview

The section describes the programmability features of OS10 using the Control Plane Services (CPS) infrastructure. In addition to an introduction to basic CPS concepts, this section describes YANG data modeling used for OS10 programming, and the structure of CPS applications, through template applications and examples.

Introduction to Control Plane Services

The Control Plane Services (CPS) infrastructure is at the core of OS10 programmability. The CPS infrastructure supports the definition of a well-defined data-centric API (CPS API) that allows customer applications to interact with OS10 services. OS10 applications also use the CPS API infrastructure for communication with one another.

The CPS infrastructure provides the following features:

- A distributed framework for application interaction
- DB-like API semantics, for create, delete, set, commit, and get operations
- Publish/subscribe semantics
- Object addressability based on object keys
- Introspection

CPS Concepts

CPS server applications register for ownership of CPS objects and implement CPS operations, such as object create, set, get, and delete. CPS client applications operate upon objects and can subscribe to events published by CPS server applications.

Example: Temperature Control Application

To understand how the CPS infrastructure operates, consider the example of a Temperature Control (TC) application. This example illustrates a simple OS10 implementation.

The TC application increases the speed of a system fan when the value reported by a temperature sensor exceeds a pre-defined threshold; it decreases the fan speed when the temperature falls below the threshold. To perform these operations, the TC application needs to **subscribe** to temperature threshold crossing events published by the OS10 Platform Adaptation Service (PAS), and invoke a **set** request to change the speed of the fan object. Note that neither the TC application nor the OS10 PAS are part of the CPS infrastructure; they function as user applications of the CPS infrastructure.

This use case illustrates several important CPS concepts:

- Entities handled by the CPS infrastructure are called CPS objects (referred to as **objects** in this document); the temperature sensor and the fan are both modeled as objects.

- Objects have attributes; for example, temperature value and fan speed.
- CPS applications can publish events; when and how events are generated is controlled by the application implementation. For example, an application can publish an event when the value of an attribute changes or when the value of an attribute crosses a threshold.
- CPS applications can subscribe to (listen for) events.
- CPS applications can request operations to be performed on objects; for example, a set operation. The operation is performed by the CPS service that registers for the ownership of the specific object category.
- The CPS API provides database-like semantics. However, services that implement CPS operations (object owners) do not need to persistently store attribute values.

In the example, the PAS registers for ownership of both the temperature sensor and fan object types. During its execution, the PAS application periodically reads the value of a temperature sensor. When the temperature is greater than a pre-defined threshold, the PAS creates a CPS over-temperature event. The event contains the identity of the temperature sensor object and possibly the new temperature value. The PAS then publishes the event using one of the CPS API functions. The CPS infrastructure service determines the applications that have subscribed for the event and object type, and provides the applications (in this case, the TC application) with the contents of the event.

The publisher of the event (PAS) does not need to have any knowledge of the applications that have subscribed for an event. Conversely, the subscriber application (TC application) has no knowledge of the application that has published the event.

When the TC application receives the temperature event, it needs to increase the fan speed. To do so, it creates a simple CPS transaction in which a set operation requests a change in the speed attribute value of the fan object. The transaction is performed using the functions provided by the CPS API. When the transaction is committed, the CPS infrastructure finds the owner of the fan object category; namely, the application that has registered to execute operations requested for fan objects (in this case, PAS). The CPS infrastructure then invokes the function registered by PAS to execute the set operation for fans. In this case, PAS will simply invoke the set operation for low-level fan speed provided by the System Device Interface (SDI), which acts on the fan hardware device and changes its speed.

CPS Keys

The concept of a CPS key is central to the CPS infrastructure, as all CPS objects require a key. An application needs a key to:

- Register for object events.
- Perform get requests.
- Perform transactions.

A CPS key has two components:

- A fixed key component represented as a sequence (array) of numeric values
- An optional (dynamic) key component

Fixed Component of a CPS Key

The fixed part of a CPS key consists of:

- Qualifier (target, observed, real-time)
- Attribute identifiers that describe the object name and hierarchy

The qualifier can be any of the following values:

Qualifier	C enum Symbol	String Form	Applicability	Description
Target	cps_api_qualifier _TARGET	target	Configurable attributes	Currently running configuration
Observed	cps_api_qualifier _OBSERVED	observed	Configurable attributes Hardware status attributes	Configuration applied to hardware components or current hardware status (provided by a HW monitoring service; it can be cached)
Real-Time	cps_api_qualifier _REALTIME	realtime	hardware status attributes hardware counters	Requests values that are immediately queried from the hardware (no caching)
Registrations	cps_api_qualifier _REGISTRATION	objreg	Internal to CPS Infrastructure	Qualifier used when publishing events associated to object registrations
Proposed	cps_api_qualifier _PROPOSED	proposed	n/a	Reserved for future use

In addition, you can use another string form - a name alias - to represent a key in scripting and other high-level languages. For example, a numerical representation (entered as a string) of a key is `1.34.2228241.2228246.222823`6. The corresponding name alias is `target/base-ip/ipv4/vrf-id/ifindex`.

Optional Key Component

The optional (dynamic) component of a CPS key consists of a series of attribute values stored within the CPS object itself. This key component is created by adding the relevant attributes to the object.

Build a CPS Key

Build a CPS key using the `cps_api_key_from_attr_with_qual` function. Using this API, you creates a key by specifying the object name and CPS qualifier; for example:

```
cps_api_key_from_attr_with_qual(cps_api_object_key(the_object), object_name,cps_api_qualifier_TARGET);
```

The `cps_api_key_from_attr_with_qual` function looks up the dynamic portion of the key and copies it into the key using the qualifier target. To add the dynamic portion fo the key, you must add attributes using the standard object attribute function.

CPS Qualifiers

A CPS qualifier provides the type of object data to be retrieved or acted upon. As shown in the preceding temperature control example, a client application specifies the target qualifier in the CPS key of the fan object used to set the fan speed. This qualifier tells the PAS application to apply the specified speed to the fan (hardware) device. Applications can only use the target qualifier for create or set operations.

In a get operation, an application can use any supported qualifier: target, observed, or real-time.

- The target qualifier indicates that the server application (object owner) should return the values of attributes previously specified in a set operation.
- The observed qualifier indicates that the server application (object owner) should return the values of the attributes already successfully applied to hardware entities (for user configurable attributes), or the last retrieved hardware status information values.
- The real-time qualifier indicates that the server application (object owner) should read the current status of the hardware entity in order to return the value of a requested attribute.

Observed versus Real-Time Qualifiers

In the temperature control example, when an application uses the observed qualifier to perform a get operation on the temperature sensor object, the PAS returns the last value read from the sensor (cached value). However, when the real-time qualifier is used, the PAS reads the current value of the sensor and returns the current, instead of the cached, value. Note that using real-time instead of observed qualifiers only produces different results when the server application maintains cached values. If the application always reads the current hardware status when it performs a get operation, the results are identical.

Target versus Observed Qualifiers

When an application gets an attribute value after a set operation, target and observed qualifiers may produce different results. In the temperature control example (CPS Concepts section), the set operation to change fan speed should use the target qualifier. However, because it takes a few seconds for fan speed to reach the specified value, an immediate get operation using an observed qualifier may return a fan speed value different from a get operation that uses a target qualifier for the fan object key.

CPS Objects

CPS objects and CPS object lists are the primitives used in the CPS infrastructure. A CPS object consists of a key (which uniquely identifies the object) and zero or more object attributes.

A CPS object has the following properties:

- Contains a variable number of attributes
- Can embed other objects
- Can be easily serialized (can be written to, and read from, a persistent or temporary storage device)
- Support the following attribute types:
  - uint8_t
  - uint16_t
  - uint32_t
  - uint64_t
  - char []
  - attributes that contain other attributes.

CPS Object Attributes

CPS object attributes are identified by a 64-bit ID tag (attribute identifier) and represented using a tag, length, value (TLV) format. The value consists of a sequence of octets.

When an attribute contains a zero-terminated string, the terminating octet must be included in the length field. For example, the total length of the string "`Dell`" must be set to 5 in order to include the zero terminating octet. Note that while the Python implementation automatically adds a zero octet to all string values, the C/C++ implementation does not. Therefore, you must take into account the zero-terminating octet when you use a C/C++ application to set the length of an attribute.

CPS Publish/Subscribe Features

The CPS infrastructure provides a subscription mechanism for receiving and publishing object events. An application can register to receive objects using an object key. The CPS key identifies a set of objects in an object category.

An application can register for:

- All events for a given object category
- All events for an object name

When an application publishes an event, subscriber applications receive the event if they have registered a key which has a prefix match with the key associated to the published object. For example, if you publish a target IPv4 address object that has the following hierarchy:

```
base-ip/ipv4/address
```

The object key for the object is: `{target, base-ip, ipv4, address}`

Note that the CPS qualifier (in this case, `target`) is always a mandatory part of the key.

Any application that subscribes for objects that match any of the following keys receives the event:

- Key 1: `{Target, base-ip}` - Receive all events for objects under base-ip.
- Key 2: `{Target, base-ip,ipv4}` - Receive all events for objects under IPv4.

- Key 3: `{Target, base-ip,ipv4,adddress}` - Receive all IPv4, address objects.

Multiple applications can subscribe for events that match a specified key.

CPS Generated Events

The CPS infrastructure can generate events for CPS-specific conditions. Currently, the CPS infrastructure generates events when new object owners (server applications) are registered with CPS or objects are de-registered from CPS.

The object key contains the registration qualifier `cps_api_qualifier_REGISTRATION` and the key of the object to which the registration or de-registration event refers. The key also indicates whether the event represents a new object registration or de-registration.

CPS Operations

The CPS infrastructure supports database-like requests for creating, updating, deleting, and retrieving of one or more objects in a single operation. The CPS API defines get and transaction functionality. The CPS API also defines an **action** operation to allow applications to perform certain functions without affecting object states.

Get Requests

CPS get operations operate on lists of keys or an object filter, which is used to specify the keys of objects to be retrieved. An object filter can specify an instance or a range of objects.

Get requests are blocking. When a get operation completes, the client application receives the list of retrieved objects or an error code in case of failure.

Transactions

CPS transactions are required for create, delete, and set operations. The CPS API provides functions to start, commit, and abort transactions.

In order to perform a transaction, a CPS application must start a transaction, and then add operations (create, delete, set, or action) and their associated objects to the transaction. A transaction must be committed in order to complete.

Operations on the different objects added to a transaction are not necessarily executed in the order in which they are specified in the transaction code. However, the transactions requested by an application thread are always executed in order.

When a transaction is committed, the CPS infrastructure sends all transaction operations to their appropriate handlers, and the result of the request is returned. In order for a result to be returned, the transaction must be valid and all operations must be received by the registered applications. The semantics of a transaction allows any create/set/delete operation associated with the transaction to be completed in a way that allows future CPS calls to use the object data updated as a result of committing the transaction.

Note that although it is not necessary that all functions in the transaction are completed, it is necessary that all operations in the transaction are accepted by the registered applications and scheduled for processing. For example, if you add one hundred thousand routes in a transaction, the result of the commit request is:

- All one hundred thousand route objects are valid to be created.
- The application that creates the route objects completes the request.

Transaction Commit Result

The transaction commit function returns a success or a failure code. If a transaction commit fails, the entire transaction fails. In this case, the CPS infrastructure automatically calls the rollback functions provided by the CPS server applications. It is the responsibility of the server applications (object owner applications) to roll back any incremental changes that have already been performed as part of the transaction.

Blocking

The CPS infrastructure is middleware that facilitates communication between components. Therefore, the blocking nature of any CPS transaction or duration is not determined by the CPS infrastructure, but by the implementation of applications registered to perform the requested operations (object owners).

CPS API Functions

The CPS API provides functions for:

- Initialization of the CPS services in the context of the calling process
- Key management
- Object handling
- Object attribute handling
- CPS event handling
- CPS operations

CPS Object Model Representation

OS10 uses YANG to represent the OS10 object model. YANG object-model files are converted to C object-model header files, which you use to develop CPS applications.

CPS API Language Support

CPS provides C/C++ and Python application programming interfaces.

YANG Modeling of CPS Objects

OS10 uses YANG data models to define the content of CPS objects used to configure and retrieve information from the system using CPS APIs.

A YANG model consists of types (typedef, groupings, and enums), containers (container, list, choice, and case), and properties (leaf and leaf-list). For more information on YANG models, see the RFC: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF).

Each property in the YANG container is a CPS object attribute. List containers nested in a YANG model can be treated as multiple instances of embedded attributes. CPS also supports defining a separate CPS object from each nested container.

Using YANG-modeled data, the CPS YANG parser generates:

- A C/C++ header file containing:
    - YANG model name
    - typedefs extracted from the model
    - enumerations found in the model
    - enumeration of any YANG container or properties (leaf, leaf-list, container, list, etc) found in the model
- A Python C extension library containing CPS object meta-data and YANG name-to-CPS ID mapping information.

In a C/C++ application, include the generated C header to use the CPS object name and attribute identiers to create/delete CPS objects and set values for each attribute.

In Python applications, use the YANG module, object and attribute names directly. The CPS Python engine automatically uses the extension library to derive the corresponding identifiers for these names.

The following Python values are present in the top-level dictionary:

- **data** - A Python dictionary containing the actual values of an object. Each element in the dictionary is a key-value pair, where the key is an attribute name and the value can be a byte array or another dictionary. Depending on the object, the data dictionary may contain other dictionaries and a CPS key_data attribute that contains the instance keys for the object.

- **key** - A string that indicates the CPS key as a string or an alias
- **operation** - Indicates whether an object is related to a set, delete, create, or action transaction (used when events are received).

CPS Object Dictionary Support

The CPS object dictionary APIs can access the meta-data for each YANG model. The dictionary contains the following items for each YANG class or attribute:

- static key of the element (including the elements hierarchy)
- type of element
- documentation associated with the element
- model type (attribute, attribute-list, object)
- unique attribute identifier

You can access the CPS object dictionary in both C/C++ and Python. Refer to CPS API Reference for more information.

Example: C Header Generated from YANG Model

This section shows an example of C header file generation from a YANG model and an example of how to use the YANG attributes to configure OS10 using the CPS API in a Python script.

```
module dell-base-sflow {
   namespace "http://www.dell.com/networking/dell-ops/dell-base-sflow";
   prefix "base-sflow";

   import dell-base-common {
       prefix "base-cmn";
   }

   organization
       "Dell Inc";

   contact
   "http://www.dell.com/support/softwarecontacts";

   description
       "This module contains a collection of YANG definitions provided
```

```
         by platfrom to manage sflow objects";

    revision 2014-02-11 {
        description
            "Initial revision";
    }

    typedef traffic-path {
      type enumeration {
        enum "ingress" {
          value 1;
          description
            "Enable sampling on Ingress packets";
        }
        enum "egress" {
          value 2;
          description
            "Enable sampling of Egress packets";
        }
        enum "ingress-egress" {
          value 3;
          description
            "Enable sampling of Ingress and Egress packets";
        }
      }
      default "ingress-egress";
    }

    list entry {

        key "id";

        description
            "sflow session attributes";

        leaf id{
            type uint32;
            description
                "Session id to uniquely identify a sflow session";
        }

        leaf ifindex {
            type base-cmn:logical-ifindex;
            mandatory true;
            description
                "Interface index which uniquely identifies physical
                 interface in the switch where packet sampling needs to
                 to be enabled";
        }
        leaf direction {
            type base-sflow:traffic-path;
            mandatory true;
            description
                "Direction of packets in which sampling needs to be enabled";
        }

        leaf sampling-rate{
            type uint32;
            mandatory true;
            description "Rate at which packets sampling needs to be enabled";
        }
    }

    container socket-address {
        description
            "Address that sFlow Applications need to open UDP socket on
             to receive sampled packets. Sampled packets from all sFlow
             sessions are sent to a single UDP socket.";

        leaf ip {
            type base-cmn:ipv4-address;
            default 127.0.0.1;
        }
        leaf udp-port {
            type uint16;
            default 20001;
        }
    }
}
```

In this example, the YANG model has two top-level containers:

- a YANG list named `entry`
- a YANG container named `socket-address`

The CPS YANG parser generates the following C header for this YANG model. The header includes the following C definitions for the YANG entities:

- Category for the YANG model:
    - cps_api_obj_CAT_BASE_SFLOW
- Subcategory for each YANG container:
    - BASE_SFLOW_ENTRY_OBJ

    - BASE_SFLOW_SOCKET_ADDRESS_OBJ

- Attribute IDs for each property in each YANG container; for example:
    - BASE_SFLOW_ENTRY_IFINDEX
    - BASE_SFLOW_ENTRY_DIRECTION

```
/*
* source file : dell-base-sflow.h
*/



/*
* Copyright (c) 2015 Dell Inc.
*
* Licensed under the Apache License, Version 2.0 (the "License"); you may
* not use this file except in compliance with the License. You may obtain
* a copy of the License at http://www.apache.org/licenses/LICENSE-2.0
*
* THIS CODE IS PROVIDED ON AN  *AS IS* BASIS, WITHOUT WARRANTIES OR
* CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
* LIMITATION ANY IMPLIED WARRANTIES OR CONDITIONS OF TITLE, FITNESS
* FOR A PARTICULAR PURPOSE, MERCHANTABLITY OR NON-INFRINGEMENT.
*
* See the Apache Version 2.0 License for specific language governing
* permissions and limitations under the License.
*/
#ifndef DELL_BASE_SFLOW_H
#define DELL_BASE_SFLOW_H

#include "cps_api_operation.h"
#include "dell-base-common.h"
#include <stdint.h>
#include <stdbool.h>


#define cps_api_obj_CAT_BASE_SFLOW (27)

#define DELL_BASE_SFLOW_MODEL_STR "dell-base-sflow"


/*Enumeration base-sflow:traffic-path */
typedef enum {
  BASE_SFLOW_TRAFFIC_PATH_INGRESS = 1, /*Enable sampling on Ingress packets*/
  BASE_SFLOW_TRAFFIC_PATH_EGRESS = 2, /*Enable sampling of Egress packets*/
  BASE_SFLOW_TRAFFIC_PATH_INGRESS_EGRESS = 3, /*Enable sampling of Ingress and Egress packets*/
} BASE_SFLOW_TRAFFIC_PATH_t;
```

```
/*Object base-sflow/entry */
typedef enum {
/*type=uint32*/
/*Session id to uniquely identify a sflow session*/
  BASE_SFLOW_ENTRY_ID = 1769474,

/*type=base-cmn:logical-ifindex*/
/*Interface index which uniquely identifies physical
interface in the switch where packet sampling needs to
to be enabled*/
  BASE_SFLOW_ENTRY_IFINDEX = 1769475,

/*type=base-cmn:traffic-path*/
/*Direction of packets in which sampling needs to be enabled*/
  BASE_SFLOW_ENTRY_DIRECTION = 1769476,

/*type=uint32*/
/*Rate at which packets sampling needs to be enabled*/
  BASE_SFLOW_ENTRY_SAMPLING_RATE = 1769477,

} BASE_SFLOW_ENTRY_t;

/*Object base-sflow/socket-address */
typedef enum {
/*type=base-cmn:ipv4-address*/
  BASE_SFLOW_SOCKET_ADDRESS_IP = 1769479,

/*type=uint16*/
  BASE_SFLOW_SOCKET_ADDRESS_UDP_PORT = 1769480,

} BASE_SFLOW_SOCKET_ADDRESS_t;

/* Object subcategories */
typedef enum{
/*sflow session attributes*/
  BASE_SFLOW_ENTRY = 1769478,
  BASE_SFLOW_ENTRY_OBJ = 1769478,

/*Address that sFlow Applications need to open UDP socket on
to receive sampled packets. Sampled packets from all sFlow
sessions are sent to a single UDP socket.*/
  BASE_SFLOW_SOCKET_ADDRESS = 1769481,
  BASE_SFLOW_SOCKET_ADDRESS_OBJ = 1769481,

} BASE_SFLOW_OBJECTS_t;


#endif
```

Example: Configure sFlow using YANG and Python

The following Python example shows how to use a YANG model to configure a new sFlow entry in OS10. The process of writing an application to configure OS10 is explained in more detail in CPS Application Templates.

This example uses an OS10 Python utility named cps_utils to create CPS objects and CPS Transactions.

```
import cps_utils
import nas_os_utils


# Create a CPS object for the YANG container named 'entry'
cps_obj = cps_utils.CPSObject(module='base-sflow/entry')

# Add each property in the YANG container as an attribute to the CPS Object
cps_obj.add_attr ("ifindex", nas_os_utils.if_nametoindex('e101-003-0'))
cps_obj.add_attr ("direction", 1)
cps_obj.add_attr ("sampling-rate", 5000)

# Pair the CPS object with a CPS Operation - in this case it is a Create operation.
cps_update = ('create',cps_obj.get())

# Add the pair to the list of updates in a CPS transaction
cps_trans = cps_utils.CPSTransaction ([cps_update])

# Commit the transaction
r = cps_trans.commit()

if not r:
    print "Error"
else:
    print "Success"
```

The cps_get_oid is a Python utility that executes an OS10 CPS Get API on a YANG container. The result displays the values configured in OS10 for all the attributes in the YANG container.

```
$ cps_get_oid.py 'base-sflow/entry'    Key: 1.27.1769478.1769474.  base-sflow/entry/ifindex = 16  base-
sflow/entry/direction = 1  base-sflow/entry/sampling-rate = 5000  base-sflow/entry/id = 1
```

YANG Model Reference

OS10 provides the following YANG models to configure networking- and platform-related functions.These YANG models are defined by the Network Adaptation and Platform Adaptation Services.

YANG Models for Networking Features

YANG Model	Networking Feature
dell-base-acl.yang	Access control lists (ACLs)
dell-base-common.yang	Common definitions
dell-base-interface-common.yang	Interfaces
dell-base-l2-mac.yang	Layer 2 MAC address
dell-base-lag.yang	Port channels/ link aggregation groups (LAGs)
dell-base-mirror.yang	Port mirroring
dell-base-phy-interface.yang	Layer 1/physical layer (PHY) interfaces
dell-base-port-security.yang	Port security protocols
dell-base-qos.yang	Quality of Service (QoS)
dell-base-routing.yang	Routing protocols
dell-base-sflow.yang	sFlow
dell-base-statistics.yang	Diagnostic/statistical information
dell-base-stg.yang	STP protocols
dell-base-switch-element.yang	Global configuration parameters for NPU
dell-base-vlan.yang	VLAN

YANG Models for Platform Functionality

Platform Function	YANG Model
Platform Adaptation Service	dell-base-pas.yang
Temperature Control	dell-base-env-tempctl.yang
Common platform definitions	dell-base-platform-common.yang

CPS Application Templates

This section provides templates for developing applications in C/C++ and Python, including:

- Server applications
- Client applications
- Event publishers
- Event subscribers

Client applications subscribe to events; server applications publish events. However, applications can act as both servers and clients, and can publish and subscribe to events.

C Template: CPS Server Application

This section describes the structure of a CPS server (object owner) implemented in C. It illustrates the implementation of functions for:

- get operations (read function, xyz_read)
- set, create, delete, action operations (xyz_write)
- rollback of failed transactions (xyz_rollback)

```
/*******************************************************************

Template CPS API object server read handler function

This function is invoked by the CPS API service when a GET request

is placed for a registered CPS API object.  The binding of CPS
API object key to the read handler function is done below.

*******************************************************************/

cps_api_return_code_t xyz_read(
   void                 *context,
   cps_api_get_params_t *param,
   size_t               key_idx
   )
{
   /* Allocate a response object, and add to response */
   cps_api_object_t response_obj;

   response_obj = cps_api_object_list_create_obj_and_append(
                      param->list
                      );
   if (response_obj == CPS_API_OBJECT_NULL) {
       /* Failed to allocate response object
          => Indicate an error
       */

       return (cps_api_ret_code_ERR);
   }

   /* Fill in response object */

   cps_api_key_from_attr_with_qual(cps_api_object_key(response_obj),
                                  ...                                  );

   cps_api_set_key_data(response_obj, ...);
   ...      cps_api_set_key_data(response_obj, ...);
     cps_api_object_attr_add_...(response_obj, ...);
   ...      cps_api_object_attr_add_...(response_obj, ...);

   /* Indicate GET response successful */

   return (cps_api_ret_code_OK);}
```

```
/*******************************************************************

Template CPS API object server write handler function

This function is invoked by the CPS API service when a SET request
is placed for a registered CPS API object.  The binding of CPS
API object key to the write handler function is done below.

********************************************************************/

cps_api_return_code_t xyz_write(
   void                         *context,
   cps_api_transaction_params_t *param,
   size_t                       index_of_element_being_updated
   )
{
   /* Extract the object given in the request */

   cps_api_object_t request_obj;
```

```
request_obj = cps_api_object_list_get(
                  param->change_list,
                  index_of_element_being_updated
                  );
if (request_obj == CPS_API_OBJECT_NULL) {
    /* Failed to extract request object
       => Indicate error
    */

    return (cps_api_ret_code_ERR);
}

/* Assume error response */

cps_api_return_code_t result = cps_api_ret_code_ERR;

/* Determine the type of write operation */

switch (cps_api_object_type_operation(
            cps_api_object_key(request_obj)
            )
        ) {
case cps_api_oper_SET:
    /* SET operation requested */

    /* Create the rollback object, i.e. an object to return
       containing the old values for any attributes set, and
       add to transaction
    */

    cps_api_object_t rollback_obj;

    rollback_obj = cps_api_object_list_create_obj_and_append(
                       param->prev
                       );
    if (rollback_obj == CPS_API_OBJECT_NULL) {
        /* Failed to create rollback object */

        break;
    }

    /* Assume SET successful */

    result = cps_api_ret_code_OK;

    /* For each attribute given in the request, … */

    cps_api_object_it_t attr_iter;
    cps_api_object_it_begin(request_obj, &attr_iter);
    while (cps_api_object_it_valid(&attr_iter)) {
        /* Get the attribute id from the attribute iterator */

        cps_api_attr_id_t attr_id;

        attr_id = cps_api_object_attr_id(attr_iter.attr);

        /* Update the rollback object with the old value
           of the attribute
        */
        cps_api_object_attr_add_...(rollback_obj,
                                    attr_id,
                                    …
                                    );

        /* Extract the attribute from the request object */

        cps_api_object_attr_t attr;

        attr = cps_api_object_attr_get(request_obj, attr_id);
        if (attr == CPS_API_ATTR_NULL)) {
            /* Failed to extract attribute
               => Indicate error
            */
            result = cps_api_ret_code_ERR;

            continue;
        }

        /* Extract the value of the attribute in the request
           object
        */

        value = cps_api_object_attr_data_....(attr);

        /* Validate the requested attribute value, its
           consistency with other attributes and/or existing
           configuration, etc.
        */
    }

    /* If the whole request has been validated, do something with
       the extracted values – program hardware,
       take some action, etc.
    */

    break;
```

```
   case cps_api_oper_CREATE:
       /* CREATE operation requested */

       break;

   case cps_api_oper_DELETE:
       /* DELETE operation requested */

       break;

   case cps_api_oper_ACTION:
       /* ACTION operation requested */

       break;

   default:
       /* Invalid SET request type */
   }

   return (result);

}
```

```
/*******************************************************************

Template CPS API object server rollback handler function

*******************************************************************/

cps_api_return_code_t xyz_rollback(
   void                         *context,
   cps_api_transaction_params_t *param,
   size_t                       index_of_element_being_updated
   )
{
   /* Extract object to be rolled back */

   cps_api_object_t rollback_obj;

   rollback_obj = cps_api_object_list_get(
                     param->prev,
                     index_of_element_being_updated
                     );
   if (rollback_obj == CPS_API_OBJECT_NULL) {
       /* Failed to extract rollback object
          => Indicate failure
       */
```

```
        return (cps_api_ret_code_ERR);
    }

    /* For each attribute to be rolled back, … */

    cps_api_object_it_t attr_iter;

    cps_api_object_it_begin(rollback_obj, &attr_iter);
    while (cps_api_object_it_valid(&attr_iter)) {
        /* Get the attribute id from the attribute iterator */

        cps_api_attr_id_t attr_id;

        attr_id = cps_api_object_attr_id(attr_iter.attr);

        /* Extract the attribute from the rollback object */

        cps_api_object_attr_t attr;

        attr = cps_api_object_attr_get(rollback_obj, attr_id);
        if (attr == CPS_API_ATTR_NULL)) {
            /* Failed to extract attribute

               => Indicate error
            */

            result = cps_api_ret_code_ERR;

            continue;
        }

        /* Extract the value of the attribute in the rollback
           object
        */

        value = cps_api_object_attr_data_....(attr);

        /* Apply attribute value */
    }

    return (result);
}
```

```
/******************************************************************

Template mainline function for a CPS API object server

This function registers with the CPS API service, and registers handler
functions to be invoked by the CPS API service when CPS API requests
are made for certain CPS API objects.

*******************************************************************/

cps_api_return_code_t init(void)
{      /* Obtain a handle for the CPS API service */

   cps_api_operation_handle_t cps_hdl;

   if (cps_api_operation_subsystem_init(&cps_hdl, 1) !=
           cps_api_ret_code_OK
      ) {
       /* Failed to obtain handle for CPS API service
          => Indicate an error
       */

       return (cps_api_ret_code_ERR);
   }

   /* Allocate a CPS API object registration structure */

   cps_api_registration_functions_t reg;

   /* Assign the key of the CPS API object to be registered */

   cps_api_key_init(&reg.key, …);

   /* Assign the handler functions to be invoked for this object */

   reg._read_function     = xyz_read;
   reg._write_function    = xyz_write;
   reg._rollback_function = xyz_rollback;

   /* Use obtained handle for CPS API service */

   reg.handle = cps_hdl;

   /* Perform the object registration */

   if (cps_api_register(&reg) != cps_api_ret_code_OK) {
       /* Failed to register handler function with CPS API service
          => Indicate an error
       */
```

```
        return (cps_api_ret_code_ERR);
    }

    /* All done */

    return (cps_api_ret_code_OK);
    }
```

Python Template: CPS Server Application

This section describes the structure of a CPS server (object owner) implemented in Python. It illustrates the implementation of transaction callback handler for set, create, delete, action operations, as well a separate handler for get operations.

```
import time
import cps
import cps_utils

# Define the get callback handler function
   ddef get_callback(methods, params):
   # Append an object to the response, echoing back the key
   # from the request, and supplying some attributes

   params['list'].append({'key': params['filter']['key'],
                          'data': {'attr_1': 'value_1',
                                   …
                                   'attr_n': 'value_n'
                                  }
                         }
                        )

   return True

# Define the transaction callback handler function

def transaction_callback(methods, params):
   if params['operation'] == 'set':
       # Set operation requested

       # Extract the attributes from the request object

       attr_1 = params['change']['data']['attr_1']
       …          attr_n = params['change']['data']['attr_n']

       # Do something with them -- program hardware,
       # update the configuration, etc.

       return True

   if params[‘operation’] == ‘create’:

       return True

   if params[‘operation’] == ‘delete’:

       return True

   if params[‘operation’] == ‘action’:

       return True

   return False

# Obtain a handle to the CPS API service

handle = cps.obj_init()

# Register the above handlers to be run when a request is received
# for the given key

cps.obj_register(handle,
                key,
                { ’get’: get_callback,
                  ’transaction’: transaction_callback
                }
               )

# Let the handlers run

while True:
    time.sleep(1000)
```

C Template: CPS Client Application

This section describes the structure of a CPS client application implemented in C. It illustrates the implementation of:

- get requests, including building keys and object lists
- set requests, using a transaction

```
/*******************************************************************

Template to perform a CPS API GET request.

*******************************************************************/

cps_api_return_code_t do_get_request()
{
   /* Allocate and initialize the get request structure */

   cps_api_get_params_t get_req;

   if (cps_api_get_request_init(&get_req) != cps_api_ret_code_OK) {
       /* Failed to initialize get request
          => Indicate error
       */

       return (cps_api_ret_code_ERR);
   }

   /* Assume failure response */

   cps_api_return_code_t result = cps_api_ret_code_ERR;

   do {
       /* Allocate the request object and add it to the get
          request
       */

       cps_api_object_t request_obj;

       request_obj = cps_api_object_list_create_obj_and_append(
                         get_req.filters
                         );
       if (request_obj == CPS_API_OBJECT_NULL) {
           /* Failed to allocate response object and add it to
              get request
           */
```

```
    break;
}

/* Set the key and key attributes for the request object.
   (The actual object key and key attribute ids, types and
   values will of course depend on which object is being
   requested; such dependent values are indicated by ellipses
   below.  Consult the data model for the desired object.)
*/

cps_api_key_from_attr_with_qual(cps_api_object_key(
                                    request_obj
                                    ),
                                …
                                );

cps_api_set_key_data(request_obj, ...);
...          cps_api_set_key_data(request_obj, ...);

cps_api_object_attr_add_...(request_obj, ...);
...          cps_api_object_attr_add_...(request_obj, ...);

/* Do the GET request */

if (cps_api_get(&get_req) != cps_api_ret_code_OK) {
    /* GET request failed */

    break;
}

/* Extract the response object */

cps_api_object_t response_obj;

response_obj = cps_api_object_list_get(get_req.list, 0);

if (response_obj == CPS_API_OBJECT_NULL) {
    /* Failed to extract the response object */

    break;
}

/* Extract the desired object attributes from the
   response object.  (The actual object attributes
   will depend on the nature of the response object;
   such dependent values are indicated by ellipses
   below.  Consult the appropriate data model for
   details.)
*/
```

```
        cps_api_object_attr_t attr;

        attr = cps_api_object_attr_get(response_obj, …);
        if (attr == CPS_API_ATTR_NULL) {
            /* Failed to extract expected attribute */

            break;
        }

        /* Get the value for the attribute */

        … = cps_api_object_attr_data_...(attr);

        /* Do something with the extracted value */

        /* Indicate success */

        result = cps_api_ret_code_OK;
    } while (0);

    cps_api_get_request_close(&get_req);

    return (result);
}
```

```
/*******************************************************************

Template to perform a CPS API SET

*******************************************************************/

cps_api_return_code_t do_set_request()
{
    cps_api_transaction_params_t xact ;

    if (cps_api_transaction_init(&xact) != cps_api_ret_code_OK) {
        return (cps_api_ret_code_ERR);
    }

    cps_api_return_code_t result = cps_api_ret_code_ERR;

    do {
        cps_api_object_t request_obj;

        request_obj = cps_api_object_create() ;
        if (request_obj == CPS_API_OBJECT_NULL) {
            break;
        }

        /* Set key and attributes in request object */

        cps_api_key_from_attr_with_qual(cps_api_object_key(
                                            request_obj
                                            ),
                                        …
                                        );

        cps_api_set_key_data(request_obj, ...);
        ...          cps_api_set_key_data(request_obj, ...);

        cps_api_object_attr_add_...(request_obj, ...);
        ...          cps_api_object_attr_add_...(request_obj, ...);

        if (cps_api_set(&xact, request_obj) != cps_api_ret_code_OK) {
            cps_api_object_delete(request_obj);

            break;
        }

        result = cps_api_commit(&xact);
    } while (0);

    cps_api_transaction_close(&xact);

    return (result);
}
```

Python Template: CPS Client Application

This section describes the structure of a CPS client application implemented in Python. It illustrates the execution of a get request.

```
import cps
import cps_utils

# Example GET request

cps_get_response = []
cps.get([cps.key_from_name('observed','base-pas/chassis')],
   cps_get_response
   )
```

```
chassis_vendor_name = cps_attr_get(cps_get_response[0]['data'],
   'base-pas/chassis/vendor-name'
   )
```

C Template: CPS Event Publisher Application

This section describes the structure of a CPS event publisher implemented in C. It illustrates the initialization of the event service, connection to the CPS service, and event publish operation. Note that the object must be deallocated (deleted) after being published.

```
/*******************************************************************

Template to perform a CPS API SET

This function is a sample of how to compose a CPS API get request
object, and how to extract data from the GET response.

*******************************************************************/

cps_api_return_code_t event_publish(cps_api_object_t event_obj)
{
   static bool                           init_flag = false;
   static cps_api_event_service_handle_t handle;

   if (!init_flag) {
       /* Not initialized
          => Connect to CPS event subsystem
       */

       if (cps_api_event_service_init() != cps_api_ret_code_OK) {
           return (cps_api_ret_code_ERR);
       }

       if (cps_api_event_client_connect(&handle) !=
               cps_api_ret_code_OK
          ) {
           return (cps_api_ret_code_ERR);
       }

       /* Mark as initialized */

       init_flag = true;
   }

   cps_api_return_code_t result;
   /* Publish the given object */

   result = cps_api_event_publish(handle, event_obj);

   /* Consume the given object */

   cps_api_object_delete(event_obj);

   return (result);
}
```

Python Template: CPS Event Publisher Application

```
import cps
import cps_utils

handle = cps.event_connect()

obj = cps_utils.CPSObject('base-port/interface',qual='observed',
                    data= {"ifindex":23})

cps.event_send(handle, obj.get())
```

C Template: Event Subscriber Application

This section describes the structure of a CPS event subscriber implemented in C. It illustrates the initialization of the event service and event processing thread, registration of the event handler function and event processing callback. The key list specified in the registration is used to determine the events that are delivered to this application (in this case, the list contains a single element).

```
bool event_handler(cps_api_object_t object, void *context)
{      /* Extract key and attributes of received object */

   /* Do something with that information */
}
cps_api_return_code_t event_subscribe()
{      /* Connect to the CPS API event service */

   if (cps_api_event_service_init() != cps_api_ret_code_OK) {
       return (cps_api_ret_code_ERR);
   }
```

```
    if (cps_api_event_thread_init() != cps_api_ret_code_OK) {
        return (cps_api_ret_code_ERR);
    }

    /* Register the event handler function */

    cps_api_key_t key;

    cps_api_key_init(&key, …);

    cps_api_event_reg_t reg;

    reg.objects           = key;
    reg.number_of_objects = 1;

    if (cps_api_event_thread_reg(&reg, event_handler, 0)
        != cps_api_ret_code_OK
        ) {
        /* Failed to register handler */

        return (cps_api_ret_code_ERR);
    }

    /* Indicate success */

    return (cps_api_reg_code_OK);
}
```

Python Template: Event Subscriber Application

This section describes the structure of a CPS event subscriber implemented in Python. The application registers for events, and then waits for events in a loop.

```
import cps
import cps_utils

handle = cps.event_connect()

cps.event_register(handle, cps_api_object_key)

while True:
   ev = cps.event_wait(handle)

   if ev[‘key’] == ...:
       ...      elif ev['key'] == ...:
       ...
```

CPS Application Examples

This section provides examples of how to write C and Python applications that use the CPS API to:

- Configure interfaces.
- Set an IP address.
- Configure a route.
- Configure an ACL.
- Add/delete a static MAC address in the MAC address table.
- Register events.

Example: Create a VLAN using Python

1. Refer to the YANG model `dell-base-vlan.yang` that defines the VLAN object and its attributes.
2. Import the CPS utility Python library.

```
import cps_utils
```

3. Create the CPS object and populate the VLAN attributes

Create a CPS object based on the YANG container `entry` in the YANG module `base-vlan`.

```
cps_obj = cps_utils.CPSObject('base-vlan/entry')
```

Fill the 'id' attribute of this object with the VLAN ID.

```
cps_obj.add_attr ("id", 100)}
```

4. Associate a CPS operation with the CPS object. Use the CREATE operation to create a new VLAN

```
cps_update = ('create', cps_obj.get())
```

5. Add the CPS operation and object pair to a new CPS transaction

```
transaction = cps_utils.CPSTransaction([cps_update])
```

6. Commit the transaction

```
ret = transaction.commit()
```

7. Verify the return value.

```
if not ret:      raise RuntimeError ("Error creating Vlan")
```

Code block: Create a VLAN using Python

```
#import cps_utils

# Create CPS Object
cps_obj = cps_utils.CPSObject('base-vlan/entry')

# Populate the attributes for the CPS Object
cps_obj.add_attr ("id", 100)

# Associate a CPS Operation with the CPS Object
cps_update = ('create', cps_obj.get())

# Add the CPS Operation,Obj pair to a new CPS Transaction
transaction = cps_utils.CPSTransaction([cps_update])

# Commit the transaction
ret = transaction.commit()

# Check for failure
if not ret:
    raise RuntimeError ("Error creating Vlan")

print "Successfully created"
```

Verify the VLAN creation using CPS get

```
$ cps_get_oid.py 'base-vlan/entry'

Key: 1.31.2031630.2031618.
base-vlan/entry/id = 100
base-vlan/entry/learning-mode = 1
base-vlan/entry/admin-status = 0
base-vlan/entry/ifindex = 42
base-vlan/entry/name = br100
base-vlan/entry/mac-address = 000000000000
```

NOTE: OS10 allocates an ifindex for each VLAN created. Further CPS set, get operations can use this ifIndex as the key.

Verify the VLAN creation using Linux Commands

```
brctl show

bridge name     bridge id               STP enabled     interfaces

br100           8000.000000000000       no
```

Example: Add a VLAN Port using Python

1. Create a CPS Object and populate the VLAN attributes.

- Import nas_os_utils python library for converting interface name to index.

```
import nas_os_utils
```

- Create a CPS object based on the YANG container named 'entry' from the YANG module 'base-vlan'.

```
cps_obj = cps_utils.CPSObject('base-vlan/entry')
```

- Set the IfIndex that was allocated by OS10 when the VLAN is created.

```
vlan_ifindex=42
cps_obj.add_attr ("ifindex", vlan_ifindex)
```

- Add ports to the VLAN by setting the untagged-ports attribute. This is a leaf-list attribute that can be set using a Python list.

```
cps_obj.add_list ("untagged-ports", [nas_os_utils.if_nametoindex('e101-001-0')])
```

The e101-001-0 interface is added as an untagged member of the VLAN.

2. Associate an operation with the CPS object. Use a set operation to modify the property of an existing VLAN

```
cps_update = ('set', cps_obj.get()) &$10;
```

3. Add the CPS operation and object pair to a new CPS transaction.

```
transaction = cps_utils.CPSTransaction([cps_update])
```

4. Commit the transaction.

```
ret = transaction.commit()
```

5. Verify the return value.

```
if not ret:
```

Code block: Add a VLAN Port using Python

```
#Python code block to add port to vlan
import cps_utils
# Create CPS Object
cps_obj=cps_utils.CPSObject('base-vlan/entry')
# Populate the Vlan attributes
vlan_ifindex=42
cps_obj.add_attr ("ifindex", vlan_ifindex)
# Add one or more ports to the untagged-ports property of the VLAN
cps_obj.add_list ("untagged-ports", [16])
# Associate a CPS Set Operation with the CPS Object
cps_update = ('set', cps_obj.get())
# Add the CPS operation,obj pair to a new CPS Transaction
transaction = cps_utils.CPSTransaction([cps_update])
# Commit the transaction
ret = transaction.commit()
# Check for failure
if not ret:
    raise RuntimeError ("Error in delete port to Vlan")
```

Verify the VLAN port addition using CPS get

```
$ cps_get_oid.py 'base-vlan/entry'

Key: 1.31.2031630.2031618.
base-vlan/entry/id = 100
base-vlan/entry/learning-mode = 1
base-vlan/entry/ifindex = 42
base-vlan/entry/admin-status = 0
base-vlan/entry/untagged-ports  =  16
base-vlan/entry/ifindex = 42
base-vlan/entry/name = br100
base-vlan/entry/mac-address = 000000000000
```

Verify the VLAN port addition using Linux Commands

```
$ brctl show
bridge name     bridge id               STP enabled     interfaces
br100           8000.90b11cf4aab3       no              e101-001-0
```

Example: Delete a VLAN Port using Python

1. Create a CPS object and populate the VLAN attributes and the port to be deleted in the CPS object.

- Create a CPS object based on the YANG container `entry` in the YANG module `base-vlan`.

```
cps_utils.CPSObject('base-vlan/entry')
```

- Set the IfIndex that was allocated by OS10 when the VLAN is created.

```
vlan_ifindex=42      cps_obj.add_attr ("ifindex", vlan_ifindex)
```

- Set an empty port list for the untagged-ports attribute to remove the previous port configuration.

```
cps_obj.add_list ("untagged-ports", [])
```

2. Associate a CPS operation with the CPS object. Use a set operation to remove ports from the existing VLAN.

```
cps_update = ('set', cps_obj.get())
```

3. Add the CPS operation and object pair to a new CPS transaction.

```
transaction = cps_utils.CPSTransaction([cps_update])
```

4. Commit the transaction.

```
ret = transaction.commit()
```

5. Verify the return value.

```
if not ret:      raise RuntimeError ("Error adding port to Vlan")
```

Code block: Delete a VLAN Port using Python

```
#Python code block to add port to vlan
import cps_utils
# Create CPS Object
cps_obj=cps_utils.CPSObject('base-vlan/entry')
# Populate the Vlan attributes
vlan_ifindex=42
cps_obj.add_attr ("ifindex", vlan_ifindex)
# Add an empty port list to remove the untagged port
cps_obj.add_list ("untagged-ports", [])
# Associate a CPS Operation with the CPS Object
cps_update = ('set', cps_obj.get())
# Add the CPS operation,obj pair to a new CPS Transaction
transaction = cps_utils.CPSTransaction([cps_update])
# Commit the transaction
ret = transaction.commit()
# Check for failure
if not ret:
    raise RuntimeError ("Error in delete port to Vlan")
```

Verify the VLAN port deletion using CPS get

```
$ cps_get_oid.py 'base-vlan/entry'

Key: 1.31.2031630.2031618.
base-vlan/entry/id = 100
base-vlan/entry/learning-mode = 1
base-vlan/entry/ifindex = 42
base-vlan/entry/admin-status = 0
base-vlan/entry/ifindex = 42
base-vlan/entry/name = br100
base-vlan/entry/mac-address = 000000000000
```

Verify the VLAN port deletion using Linux Commands

```
$ brctl show

bridge name     bridge id               STP enabled     interfaces
br100           8000.000000000000       no
```

Example: Delete a VLAN using Python

1. Populate the VLAN attributes and delete the CPS object. Use the object `entry` in the Yang module `base-vlan`. Enter the VLAN ID in the `id` attribute in the object.

```
vlan_ifindex = 42
vlan_attributes = {"ifindex": vlan_ifindex}  cps_obj = cps_utils.CPSObject('base-vlan/entry',
data=vlan_attributes)
```

The CPS object is created and attributes are added in a single API call.

2. Associate a CPS operation with the CPS object. Use a delete operation to remove the existing VLAN.

```
cps_update = ('delete', cps_obj.get())
```

3. Add the CPS operation and object pair to a new CPS transaction

```
transaction = cps_utils.CPSTransaction([cps_update])
```

4. Commit the transaction

```
ret = transaction.commit()
```

5. Verify the return value.

```
if not ret:
    raise RuntimeError ("Error delete Vlan")
```

Code block: Delete a VLAN using Python

```
#Python code block to Delete vlan
import cps_utils
# Populate the attributes for the CPS Object
vlan_ifindex=42
vlan_attributes = {"ifindex": vlan_ifindex}
# Create CPS Object
cps_obj=cps_utils.CPSObject('base-vlan/entry',  data=vlan_attributes)
# Create the CPS Transaction and delete the CPS Object
cps_update = ('delete', cps_obj.get())
transaction = cps_utils.CPSTransaction([cps_update])
# Commit the transaction
ret = transaction.commit()
# Check for failure
if not ret:
    raise RuntimeError ("Error Deleting Vlan")
```

Verify the VLAN deletion using CPS get

```
cps_get_oid.py 'base-vlan/entry' &#10,
```

The CPS object is empty and not listed in the get response.

Verify the VLAN deletion using Linux Commands

```
brctl show
```

brctl displays vlan 100 as deleted.

Example: Configure an IP Address using Python

1. Refer to the YANG model `dell-base-ip.yang` which defines the IP address object and its attributes.

2. Import the CPS utility Python library.

```
import cps_utils
```

3. Populate base-ip attributes and create a CPS object. Enter the Ifindex and prefix length of the interface to set the IP address.

```
ifindex=16
ip_addr="10.0.0.1"
pfix_len=16
ip_attributes = {"base-ip/ipv4/ifindex": ifindex,"ip":ip_addr,"prefix-length":pfix_len}
```

4. Enter `ipv4` as the IP attribute type to convert between the string and byte-array format.

```
cps_utils.add_attr_type('base-ip/ipv4/address/ip',"ipv4")
cps_obj=cps_utils.CPSObject('base-ip/ipv4/address',data=ip_attributes)
```

5. Create the CPS transaction and add the CPS object.

```
cps_update = ('create', cps_obj.get())
```

6. Add the CPS operation and object pair to a new CPS transaction.

```
transaction = cps_utils.CPSTransaction([cps_update])
```

7. Commit the transaction.

```
ret = transaction.commit()
```

8. Verify the return value.

```
if not ret:
    raise RuntimeError ("Error creating Vlan")
```

Code block: Configure an IP Address using Python

```
#Python code block to set ip address
import cps_utils
# Populate the attributes for the CPS Object
ifindex=16
ip_addr="10.0.0.1"
pfix_len=16
ip_attributes = {"base-ip/ipv4/ifindex": ifindex,"ip":ip_addr,"prefix-length":pfix_len}
# Create CPS Object
cps_utils.add_attr_type('base-ip/ipv4/address/ip',"ipv4")
cps_obj=cps_utils.CPSObject('base-ip/ipv4/address',data=ip_attributes)
# Create the CPS Transaction and delete the CPS Object
cps_update = ('create', cps_obj.get())
transaction = cps_utils.CPSTransaction([cps_update])
# Commit the transaction
# Check for failure
if not ret:
    raise RuntimeError ("Error ")
```

Verify IP address configuration using CPS get

```
$ cps_get_oid.py 'base-ip/ipv4/address'

Key: 1.34.2228241.2228246.2228236.2228240.2228228.
base-ip/ipv4/address/prefix-length = 16
base-ip/ipv4/vrf-id = 0
base-ip/ipv4/name = e101-001-0
base-ip/ipv4/ifindex = 16
base-ip/ipv4/address/ip = 0a000001
```

Verify IP address configuration using Linux Commands

```
$ ip addr show e101-001-0
16: e101-001-0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 500
   link/ether 90:b1:1c:f4:aa:b3 brd ff:ff:ff:ff:ff:ff
   inet 10.0.0.1/16 scope global e101-001-0
      valid_lft forever preferred_lft forever
```

Example: Delete an IP Address using Python

1. Import the CPS utility Python library.

```
import cps_utils
```

2. Populate base-ip attributes and delete the CPS object. Enter the Ifindex, IP address, and rpefix length of the interface.

```
idx=16
ip_addr="10.0.0.1"
pfix_len=16
ip_attributes = {"base-ip/ipv4/ifindex": idx,"ip":ip_addr,"prefix-length":pfix_len}
```

3. Add `ipv4 as` the IP attribute type to convert the IP address between string and byte-array format.

```
cps_utils.add_attr_type('base-ip/ipv4/address/ip',"ipv4")
cps_obj=cps_utils.CPSObject('base-ip/ipv4/address',data=ip_attributes)
```

4. Create the CPS transaction and add the CPS object.

```
cps_update = ('create', cps_obj.get())
```

5. Add the CPS operation and object pair to a new CPS transaction.

```
transaction = cps_utils.CPSTransaction([cps_update])
```

6. Commit the transaction.

```
ret = transaction.commit()
```

7. Verify the return value.

```
if not ret:
    raise RuntimeError ("Error creating Vlan")
```

Code block: Delete an IP Address using Python

```
#Python code block to Delete ip address
import cps_utils
# Populate the attributes for the CPS Object
idx=16
ip_addr="10.0.0.1"
pfix_len=16
ip_attributes = {"base-ip/ipv4/ifindex": idx,"ip":ip_addr,"prefix-length":pfix_len}
# Create CPS Object
cps_utils.add_attr_type('base-ip/ipv4/address/ip',"ipv4")
cps_obj=cps_utils.CPSObject('base-ip/ipv4/address',data=ip_attributes)
# Create the CPS Transaction and delete the CPS Object
cps_update = ('delete', cps_obj.get())
transaction = cps_utils.CPSTransaction([cps_update])
# Commit the transaction
# Check for failure
if not ret:
    raise RuntimeError ("Error ")
```

Verify the IP address deletion using CPS get

```
$ cps_get_oid.py 'base-ip/ipv4/address'
```

The return indicates that e101-001-0 has no IP address.

Verify the IP address deletion using Linux commands

```
$ ip addr show e101-001-0
16: e101-001-0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 500
   link/ether 90:b1:1c:f4:aa:b3 brd ff:ff:ff:ff:ff:ff
```

Example: Configure a Route using Python

1. Refer to the YANG model `dell-base-route.yang` which defines a route object and its attributes.

2. Import the CPS utility and netaddr Python library.

```
import cps_utils
import socket
import netaddr as net
```

3. Populate the route attributes and create the CPS object. Use the object `entry` in the YANG module `base-route`. Enter the version, route prefix and prefix length.

```
version = 'ipv4'
route_ip = '70.5.5.0'

obj = cps_utils.CPSObject('base-route/obj/entry')

obj.add_attr("vrf-id", 0)

if version == 'ipv4':
   obj.add_attr("af", socket.AF_INET)
elif version == 'ipv6':
   obj.add_attr("af", socket.AF_INET6)

ip = net.IPNetwork(route_ip)

obj.add_attr_type("route-prefix", version)
obj.add_attr("route-prefix", str(ip.network))
obj.add_attr("prefix-len", int(ip.prefixlen))
```

4. Populate the next-hop attributes and create the CPS object. The next-hop attributes are a list in the YANG model. Add multiple next hops to create ECMP routes. This example uses one next hop.

```
nh_addr = '1.1.1.2'

l = ["nh-list", "0", "nh-addr"]
obj.add_embed_attr(l, nh_addr)
obj.add_attr("nh-count", 1)
```

5. Create the CPS transaction and add the CPS object.

```
cps_update = ('create', obj.get())
transaction = cps_utils.CPSTransaction([cps_update])

ret = transaction.commit()
```

6. Verify the return code.

```
if not ret:
    raise RuntimeError ("Error creating Route")
```

Code block: Configure a Route using Python

```
#Python block to create a route
#File name - route_create

import cps_utils
import socket
import netaddr as net

#Populate the attributes


ersion = 'ipv4'
route_ip = '70.5.5.0'

obj = cps_utils.CPSObject('base-route/obj/entry')

obj.add_attr("vrf-id", 0)

if version == 'ipv4':
   obj.add_attr("af", socket.AF_INET)
elif version == 'ipv6':
   obj.add_attr("af", socket.AF_INET6)

ip = net.IPNetwork(route_ip)

obj.add_attr_type("route-prefix", version)
obj.add_attr("route-prefix", str(ip.network))
obj.add_attr("prefix-len", int(ip.prefixlen))

nh_addr = '1.1.1.2'
```

```
l = ["nh-list", "0", "nh-addr"]
obj.add_embed_attr(l, nh_addr)
obj.add_attr("nh-count", 1)

print obj.get()

cps_update = ('create', obj.get())
transaction = cps_utils.CPSTransaction([cps_update])

ret = transaction.commit()

if not ret:
    raise RuntimeError ("Error creating Route")
```

Verify the route creation using Linux Commands

```
$ ip route
1.1.1.0/24 dev e101-001-0  proto kernel  scope link  src 1.1.1.1
70.5.5.0 via 1.1.1.2 dev e101-001-0  proto none
```

Example: Delete a Route using Python

1. Import the CPS utility and netaddr Python library.

```
import cps_utils
import socket
import netaddr as net
```

2. Populate the route attributes and create the CPS object. Use the object `entry` in the YANG module `base-route`. Enter the version, route prefix, and prefix length.

```
version = 'ipv4'
route_ip = '70.5.5.0'

obj = cps_utils.CPSObject('base-route/obj/entry')

obj.add_attr("vrf-id", 0)

if version == 'ipv4':
    obj.add_attr("af", socket.AF_INET)
elif version == 'ipv6':
    obj.add_attr("af", socket.AF_INET6)

ip = net.IPNetwork(route_ip)

obj.add_attr_type("route-prefix", version)
obj.add_attr("route-prefix", str(ip.network))
obj.add_attr("prefix-len", int(ip.prefixlen))
```

3. Create the CPS transaction and add the CPS object.

```
cps_update = ('delete', obj.get())
transaction = cps_utils.CPSTransaction([cps_update])

ret = transaction.commit()
```

4. Verify the return code.

```
if not ret:      raise RuntimeError ("Error creating Route")
```

Code block: Delete a Route using Python

```
#Python code block to delete a route
#File name - route_delete

import cps_utils
import socket
import netaddr as net

version = 'ipv4'
route_ip = '70.5.5.0'

obj = cps_utils.CPSObject('base-route/obj/entry')

obj.add_attr("vrf-id", 0)

if version == 'ipv4':
   obj.add_attr("af", socket.AF_INET)
elif version == 'ipv6':
   obj.add_attr("af", socket.AF_INET6)

ip = net.IPNetwork(route_ip)
```

```
obj.add_attr_type("route-prefix", version)
obj.add_attr("route-prefix", str(ip.network))
obj.add_attr("prefix-len", int(ip.prefixlen))

print obj.get()
cps_update = ('delete', obj.get())
transaction = cps_utils.CPSTransaction([cps_update])

ret = transaction.commit()

if not ret:
    raise RuntimeError ("Error deleting Route")
```

Verify the route deletion using Linux commands

```
OS10:~# ip route
1.1.1.0/24 dev e101-001-0  proto kernel  scope link  src 1.1.1.1
70.5.5.0 via 1.1.1.2 dev e101-001-0  proto none

OS10:~# python route_delete
{'data': {'base-route/obj/entry/prefix-len': bytearray(b' \x00\x00\x00'), 'base-route/obj/entry/vrf-id':
bytearray(b'\x00\x00\x00\x00'), 'base-route/obj/entry/af': bytearray(b'\x02\x00\x00\x00'), 'base-
route/obj/entry/route-prefix': 'F\x05\x05\x00'}, 'key':
'1.26.1704016.1703992.1703995.1703980.1703978.1703979.'}

OS10:~# ip route
1.1.1.0/24 dev e101-001-0  proto kernel  scope link  src 1.1.1.1
```

Configure an ACL using Python - Prerequisite Steps

1. Refer to the YANG model `dell-base-acl.yang` which defines an ACL object and attributes.

2. Import the CPS utility Python library.

```
import cps_utils
```

3. Create a YANG enum map. A CPS python application does not automatically map the YANG model enum name to a number.

```
e_stg = {'INGRESS': 1, 'EGRESS': 2}

e_ftype = {'SRC_MAC': 3, 'DST_MAC': 4, 'SRC_IP': 5, 'DST_IP': 6,
          'IN_PORT': 9, 'DSCP': 21}
e_atype = {'PACKET_ACTION': 3, 'SET_TC': 10}
e_ptype = {'DROP': 1}
```

4. Register the attribute type with the CPS utility for attributes with non-integer values.

```
type_map = {
   'base-acl/entry/SRC_MAC_VALUE/addr': 'mac',
   'base-acl/entry/SRC_MAC_VALUE/mask': 'mac',
}
for key,val in type_map.items():
cps_utils.cps_attr_types_map.add_type(key, val)
```

Example: Create an ACL table using Python

An ACL table groups ACL entries and allows a packet to match one of the entries in the group. A packet can simultaneously match ACL entries in different tables. The table priority determines the order in which match criteria are applied.

1. Follow the procedure in Configure an ACL using Python - Prerequisite Steps.

2. Create the CPS Object and populate the attributes. Create a CPS object based on the YANG container `table` in the YANG model `base-acl.`

```
cps_obj = cps_utils.CPSObject(module='base-acl/table')
```

3. Set the stage and priority.

```
cps_obj.add_attr ('stage', e_stg['INGRESS'])
cps_obj.add_attr ('priority', 99)
```

The allowed-match-list attribute is a YANG leaf list, which takes multiple values provided with a Python list.

```
cps_obj.add_list ('allowed-match-fields', [e_ftype['SRC_MAC'],
                                           e_ftype['DST_IP'],
                                           e_ftype['DSCP'],
                                           e_ftype['IN_PORT']])
```

4. Associate the CPS object with a CPS operation.

```
cps_update = ('create', cps_obj.get())
```

5. Add the CPS operation and object pair to a new CPS transaction. Each CPS transaction can hold multiple CPS operation and object pairs.

```
cps_trans = cps_utils.CPSTransaction([cps_update])
```

6. Commit the transaction.

```
r = cps_trans.commit()
if not r:
    raise RuntimeError ("Error creating ACL Table")
```

7. OS10 allocates an ID for each new ACL table. Retrieve this ID and use it as the key for future get, set, and delete operations on the ACL table.

```
cps_get_val = cps_utils.CPSObject (module='base-acl/table', obj=r[0]['change'])
tbl_id = cps_get_val.get_attr_data ('id')
print "Successfully created ACL Table " + str(tbl_id)
```

Code Block: Create an ACL Table using Python

```
#!/usr/bin/python
"""
Simple Base ACL CPS config using the generic CPS Python module and utilities.
```

```
Create ACL Table
Create ACL Entry to Drop all packets received on specific port from specific Src MAC
"""

import cps_utils
import nas_os_utils

# Yang Enum name to number map
e_stg = {'INGRESS': 1, 'EGRESS': 2}
e_ftype = {'SRC_MAC': 3, 'DST_MAC': 4, 'SRC_IP': 5, 'DST_IP': 6,
          'IN_PORT': 9, 'DSCP': 21}
e_atype = {'PACKET_ACTION': 3, 'SET_TC': 10}
e_ptype = {'DROP': 1}

# Teach CPS utility about the type of each attribute
type_map = {
   'base-acl/entry/SRC_MAC_VALUE/addr': 'mac',
   'base-acl/entry/SRC_MAC_VALUE/mask': 'mac',
}
for key,val in type_map.items():
   cps_utils.cps_attr_types_map.add_type(key, val)


# Create ACL Table
#
# Create CPS Object and fill leaf attributes
cps_obj = cps_utils.CPSObject(module='base-acl/table')
cps_obj.add_attr ('stage', e_stg['INGRESS'])
cps_obj.add_attr ('priority', 99)
# Populate the leaf-list attribute
cps_obj.add_list ('allowed-match-fields', [e_ftype['SRC_MAC'],
                                           e_ftype['DST_IP'],
                                           e_ftype['DSCP'],
                                           e_ftype['IN_PORT']])

# Associate the CPS Object with a CPS operation
cps_update = ('create', cps_obj.get())
# Add the CPS object to a new CPS Transaction
cps_trans = cps_utils.CPSTransaction([cps_update])

# Commit the CPS transaction
r = cps_trans.commit()
if not r:
    raise RuntimeError ("Error creating ACL Table")
ret = cps_utils.CPSObject (module='base-acl/table', obj=r[0]['change'])
tbl_id = ret.get_attr_data ('id')
print "Successfully created ACL Table " + str(tbl_id)
```

Verify the ACL table creation in OS10

```
# cps_get_oid.py 'base-acl/table'
Key: 1.25.1638504.1638499.
base-acl/table/npu-id-list  =  0
base-acl/table/stage = 1
base-acl/table/priority = 99
base-acl/table/allowed-match-fields  =  3,6,9,21
base-acl/table/id = 2
```

Example: Create an ACL Entry using Python

An ACL entry is a rule that consists of:

- Set of filters that define packets to be matched.
- Set of actions to be performed on the matched packets.

1. Follow the procedure in Configure an ACL using Python - Prerequisite Steps.

Import nas_os_utils python library for converting interface name to index.

```
import nas_os_utils
```

2. Create the CPS object and enter the leaf attributes. Create a CPS object based on the YANG container `entry` in the YANG model `base-acl.`

```
cps_obj = cps_utils.CPSObject(module='base-acl/entry')
```

3. Enter the ID of the ACL table that you created in the previous example to indicate the group to which this ACL entry belongs. The priority value determines the sequence of ACL rule lookup in the ACL table group.

```
cps_obj.add_attr ('table-id', tbl_id)  cps_obj.add_attr ('priority', 512)
```

4. Enter the filters that define the packets to be matched. The filter attribute is a YANG nested list. The add_embed_attr() function is used to create multiple instances for nested lists. Each filter instance is made up of 2 attributes - match-type and match-value.

NOTE: Use the correct match-value attribute name depending on the value assigned to the match-type (eg: Use attribute name **`SRC_MAC_VALUE`** when match-type is **`SRC_MAC)`**.

- Filter 1 - Match packets with a specific source MAC address.

```
cps_obj.add_embed_attr (['match','0','type'], e_ftype['SRC_MAC'])
cps_obj.add_embed_attr (['match','0','SRC_MAC_VALUE','addr'],
                   '50:10:6e:00:00:00', 2)
```

- Filter 2 - Match packets received on a specific port.

```
cps_obj.add_embed_attr (['match','1','type'], e_ftype['IN_PORT'])
cps_obj.add_embed_attr (['match','1','IN_PORT_VALUE'], nas_os_utils.if_nametoindex('e101-001-0'))
```

5. Specify actions to be applied on matched packets. The action attribute is a YANG nested list. The add_embed_attr() function is used to create multiple instances for nested lists. Each action instance is made up of 2 attributes - action-type and action-value.

NOTE: Use the correct action-value attribute name depending on the value assigned to the action-type (eg: Use attribute name **`PACKET_ACTION_VALUE`** when action-type is **`PACKET_ACTION)`**.

- Action - Drop

```
cps_obj.add_embed_attr (['action','0','type'], e_atype['PACKET_ACTION'])
cps_obj.add_embed_attr (['action','0','PACKET_ACTION_VALUE'], e_ptype['DROP'])
```

6. Associate the CPS Object with a CPS operation.

```
cps_update = ('create', cps_obj.get())
```

7. Add the CPS operation and object pair to a new CPS transaction. Each CPS transaction holds multiple pairs of CPS operation and object updates.

```
cps_trans = cps_utils.CPSTransaction([cps_update])
```

8. Commit the CPS transaction.

```
r = cps_trans.commit()
if not r:
    raise RuntimeError ("Error creating MAC ACL Entry")
```

9. OS10 allocates an ID for each new ACL entry. Retrieve this ID and use it as the key for future get, set, and delete operations on this ACL entry.

```
cps_get_val = cps_utils.CPSObject (module='base-acl/entry', obj=r[0]['change'])
mac_eid = cps_get_val.get_attr_data ('id')
print "Successfully created MAC ACL Entry " + str(mac_eid)
```

Code Block: Create an ACL Table and then an ACL Entry using Python

```
#!/usr/bin/python
"""
Simple Base ACL CPS config using the generic CPS Python module and utilities.

Create ACL Table
Create ACL Entry to Drop all packets received on specific port from specific Src MAC
"""

import cps_utils
import nas_os_utils

# Yang Enum name to number map
e_stg = {'INGRESS': 1, 'EGRESS': 2}
e_ftype = {'SRC_MAC': 3, 'DST_MAC': 4, 'SRC_IP': 5, 'DST_IP': 6,
          'IN_PORT': 9, 'DSCP': 21}
e_atype = {'PACKET_ACTION': 3, 'SET_TC': 10}
e_ptype = {'DROP': 1}

# Teach CPS utility about the type of each attribute
type_map = {
   'base-acl/entry/SRC_MAC_VALUE/addr': 'mac',
   'base-acl/entry/SRC_MAC_VALUE/mask': 'mac',
}
for key,val in type_map.items():
   cps_utils.cps_attr_types_map.add_type(key, val)


# Create ACL Table
#
# Create CPS Object and fill leaf attributes
cps_obj = cps_utils.CPSObject(module='base-acl/table')
cps_obj.add_attr ('stage', e_stg['INGRESS'])
cps_obj.add_attr ('priority', 99)
# Populate the leaf-list attribute
cps_obj.add_list ('allowed-match-fields', [e_ftype['SRC_MAC'],
                                           e_ftype['DST_IP'],
                                           e_ftype['DSCP'],
                                           e_ftype['IN_PORT']])

# Associate the CPS Object with a CPS operation
cps_update = ('create', cps_obj.get())
# Add the CPS object to a new CPS Transaction
cps_trans = cps_utils.CPSTransaction([cps_update])

# Commit the CPS transaction
r = cps_trans.commit()
if not r:
    raise RuntimeError ("Error creating ACL Table")
ret = cps_utils.CPSObject (module='base-acl/table', obj=r[0]['change'])
tbl_id = ret.get_attr_data ('id')
print "Successfully created ACL Table " + str(tbl_id)


# Create ACL entry
# Drop all packets received on specific port from specific range of MACs
#
# Create CPS Object and fill leaf attributes
cps_obj = cps_utils.CPSObject(module='base-acl/entry')
cps_obj.add_attr ('table-id', tbl_id)
cps_obj.add_attr ('priority', 512)

# Filters  # Match Filter 1 - Src MAC
cps_obj.add_embed_attr (['match','0','type'], e_ftype['SRC_MAC'])
# The 2 at the end indicates that the type should be deducted from the last 2 attrs (SRC_MAC_VALUE,addr)
cps_obj.add_embed_attr (['match','0','SRC_MAC_VALUE','addr'],
                   '50:10:6e:00:00:00', 2)
# Match Filter 2 - Rx Port
cps_obj.add_embed_attr (['match','1','type'], e_ftype['IN_PORT'])
cps_obj.add_embed_attr (['match','1','IN_PORT_VALUE'], nas_os_utils.if_nametoindex('e101-001-0'))
```

```
# Action - Drop
cps_obj.add_embed_attr (['action','0','type'], e_atype['PACKET_ACTION'])
cps_obj.add_embed_attr (['action','0','PACKET_ACTION_VALUE'], e_ptype['DROP'])

# Associate the CPS Object with a CPS operation
cps_update = ('create', cps_obj.get())
# Add the CPS object to a new CPS Transaction
cps_trans = cps_utils.CPSTransaction([cps_update])

# Commit the CPS transaction
r = cps_trans.commit()
if not r:
    raise RuntimeError ("Error creating MAC ACL Entry")

ret = cps_utils.CPSObject (module='base-acl/entry', obj=r[0]['change'])
mac_eid = ret.get_attr_data ('id')
print "Successfully created MAC ACL Entry " + str(mac_eid)
```

Verify the ACL entry creation in OS10

```
# cps_get_oid.py 'base-acl/entry'
Key: 1.25.1638505.1638428.1638429.
base-acl/entry/table-id = 2
base-acl/entry/id = 1
base-acl/entry/match/IN_PORT_VALUE = 23
base-acl/entry/match/type = 9
base-acl/entry/match/SRC_MAC_VALUE/mask = ffffff000000
base-acl/entry/match/SRC_MAC_VALUE/addr = 50106e000000
base-acl/entry/match/type = 3
base-acl/entry/action/PACKET_ACTION_VALUE = 1
base-acl/entry/action/type = 3
base-acl/entry/npu-id-list  =  0
base-acl/entry/priority = 512
```

Verify the ACL entry creation in the NPU

```
EID 0x00000018: gid=0x2,
    slice=1, slice_idx=0, part =0 prio=0x200, flags=0x10202, Installed, Enabled
        tcam: color_indep=0,
Stage
StageIngress
InPort
    DATA=0x0000000000000000000000000000000000000000000000000002000000000000
    MASK=0x00000000000000000000000000000000000001ffffffffffffffffffffffffff
```

```
SrcMac
    Offset0: 241 Width0: 48
    DATA=0x00005010 6e000000
    MASK=0x0000ffff ff000000
        action={act=Drop, param0=0(0), param1=0(0), param2=0(0), param3=0(0)}
        policer=
        statistics=NULL
```

Example: Delete an ACL Entry using Python

1. Follow the procedure in Configure an ACL using Python - Prerequisite Steps.
2. Create the CPS Object and enter the table-id and entry-id key values.

```
cps_obj = cps_utils.CPSObject(module='base-acl/entry',                               data={'table-id':
tbl_id,                                       'id': mac_eid})
```

3. Associate the CPS object with a CPS operation.

```
cps_update = ('delete', cps_obj.get())
```

4. Add the CPS operation and object pair to a new CPS transaction.

```
cps_trans = cps_utils.CPSTransaction([cps_update])
```

5. Commit the CPS transaction.

```
r = cps_trans.commit()
if not r:
    raise RuntimeError ("Error deleting ACL Entry")
```

Code block: Delete an ACL Entry using Python

```
import cps_utils

#Create the CPS Object and fill the table-id and entry-id key values
cps_obj = cps_utils.CPSObject(module='base-acl/entry',
                              data={'table-id': 2,
                                    'id': 1})

#Associate the CPS Object with a CPS operation
cps_update = ('delete', cps_obj.get())

#Add the CPS object to a new CPS Transaction
cps_trans = cps_utils.CPSTransaction([cps_update])

#Commit the CPS transaction
r = cps_trans.commit()
if not r:
    raise RuntimeError ("Error deleting ACL Entry")
```

Example: Configure a MAC Table entry using Python

1. Refer to the YANG model `dell-base-l2-mac.yang` which defines the MAC object and attributes for OS10.
2. Import the CPS Utility Python library.

```
import cps_utils
```

3. Register the mac-address attribute type as mac-address to convert between the string and byte-array format.

```
cps_utils.add_attr_type("base-mac/table/mac-address", "mac")
```

4. Create new MAC entry by entering the MAC address, interface index and VLAN attributes.

```
d =  {"mac-address": "00:0a:0b:cc:0d:0e","ifindex": 18,"vlan": "100"}
```

5. Create a CPS object.

```
obj = cps_utils.CPSObject('base-mac/table',data= d)
```

6. Add the operation to the object.

```
tr_obj = ('create', obj.get())
```

7. Create a transaction object.

```
transaction = cps_utils.CPSTransaction([tr_obj])
```

8. Commit the transaction

```
ret = transaction.commit()
```

9. Verify the response and handle errors

```
if not ret:
    raise RuntimeError ("Error creating MAC Table Entry")
```

Code block: Configure a MAC Table Entry using Python

```
import cps_utils

cps_utils.add_attr_type("base-mac/table/mac-address", "mac")

d =  {"mac-address": "00:0a:0b:cc:0d:0e",
      "ifindex": 18,
      "vlan": "100"}
obj = cps_utils.CPSObject('base-mac/table',data= d)

tr_obj = ('create', obj.get())

transaction = cps_utils.CPSTransaction([tr_obj])
ret = transaction.commit()

if not ret:
    raise RuntimeError ("Error creating MAC Table Entry")
```

NOTE: When you do not enter a qualifier for a key, **`target`** is used by default when creating the object. To use a different qualifier, use the following syntax:

```
obj = cps_utils.CPSObject('base-mac/table',qual='observed',data= d)
```

Example: Configure a MAC Table Entry using C/C++

```
#include "cps_api_object.h"
#include "dell-base-l2-mac.h"
#include "cps_class_map.h"
#include "cps_api_object_key.h"

#include <stdint.h>
#include <net/if.h>

bool cps_create_mac(){

   // Create and initialize the transaction object
   cps_api_transaction_params_t tran;

   if (cps_api_transaction_init(&tran) != cps_api_ret_code_OK ){
       return false;
   }

   // Create and initialize the key
   cps_api_key_t key;
   cps_api_key_from_attr_with_qual(&key, BASE_MAC_TABLE_OBJ, cps_api_qualifier_TARGET);

   // Create the object
   cps_api_object_t obj = cps_api_object_create();

   if(obj == NULL ){
       cps_api_transaction_close(&tran);
       return false;
   }

   // Set the key for the obejct
   cps_api_object_set_key(obj,&key);

   // Add attributes mandatory to create MAC address entry
   uint8_t mac_addr[6] = {0x0,0xa,0xb,0xc,0xd,0xe};
   uint16_t vlan_id = 100;

   cps_api_object_attr_add(obj,BASE_MAC_TABLE_MAC_ADDRESS, mac_addr, sizeof(hal_mac_addr_t));
   cps_api_object_attr_add_u32(obj,BASE_MAC_TABLE_IFINDEX,if_nametoindex("e101-001-0") );
```

```
    cps_api_object_attr_add_u16(obj,BASE_MAC_TABLE_VLAN,vlan_id);

    // Add the object along with the operation to transaction
    if(cps_api_create(&tran,obj) != cps_api_ret_code_OK ){
        cps_api_object_delete(obj);
        return false;
    }

    // Commit the transaction
    if(cps_api_commit(&tran) != cps_api_ret_code_OK ) {
        cps_api_transaction_close(&tran);
        return false;
    }

    // Cleanup the Transaction
    cps_api_transaction_close(&tran);

    return true;
}
```

Verify MAC table entry is created in OS10.

```
cps_get_oid.py base-mac/query
Key: 1.16.1048594.
base-mac/query/ifindex = 18
base-mac/query/actions = 2
base-mac/query/static = 1
base-mac/query/mac-address = 000a0bcc0d0e
base-mac/query/vlan = 100
```

Example: Get a MAC Table Entry using Python

1. Import the cps_utils and cps libraries.

```
import cps_utils
import cps
```

2. Register the mac-address attribute type as `mac` to convert between the string and byte-array format.

```
cps_utils.add_attr_type("base-mac/query/mac-address", "mac")
```

3. Enter the MAC Address to be queried and request-type 2 to request the object based on the MAC address.

```
d =  {"mac-address": "00:0a:0b:cc:0d:0e","request-type":"2"}
```

4. Create a CPS object for a get request.

```
obj = cps_utils.CPSObject('base-mac/query',data= d)
```

5. Create a list which contains the objects with filters for a get request.

```
filter_list = []
```

6. Add the filter object to the Get list.

```
filter_list.append(obj.get())
```

7. Create a list for the Get response.

```
l = []
```

8. Perform the get operation.

```
if cps.get(filter_list,l):
   #Check if get returned objects in the response list
   if l:
       for ret_obj in l:
           # Print the returned Objects
           cps_utils.print_obj(ret_obj)
   else:
           print "No objects found"
else:
    raise RuntimeError ("Error Getting MAC Table Entries")
```

Code block: Get a MAC Table Entry using Python

```
import cps_utils
import cps

cps_utils.add_attr_type("base-mac/query/mac-address", "mac")

d =  {"mac-address": "00:0a:0b:cc:0d:0e","request-type":"2"}
obj = cps_utils.CPSObject('base-mac/query',
                    data= d)

filter_list = []
filter_list.append(obj.get())
l = []

if cps.get(filter_list,l):
   if l:
       for ret_obj in l:
           cps_utils.print_obj(ret_obj)
   else:
           print "No objects found"
else:
    raise RuntimeError ("Error Getting MAC Table Entries")
```

Example: Get a MAC Table Entry using C/C++

```
#include "cps_api_object.h"
#include "dell-base-l2-mac.h"
#include "cps_class_map.h"
#include "cps_api_object_key.h"

#include <stdio.h>

bool cps_get_mac(){

   // Create and initialize the Get object
   cps_api_get_params_t gp;
   cps_api_get_request_init(&gp);

   // Create a new object and append it to get request's filter object list
   cps_api_object_t obj = cps_api_object_list_create_obj_and_append(gp.filters);
   if(obj == NULL){
       cps_api_get_request_close(&gp);
       return false;
   }
```

```
    // Create, initialize and set the key for object
    cps_api_key_t key;
    cps_api_key_from_attr_with_qual(&key, BASE_MAC_QUERY_OBJ, cps_api_qualifier_TARGET);
    cps_api_object_set_key(obj,&key);

    //Perform a get request
    bool rc=false;
    if (cps_api_get(&gp)==cps_api_ret_code_OK) {
        rc = true;
        size_t mx = cps_api_object_list_size(gp.list);
        for (size_t ix = 0 ; ix < mx ; ++ix ) {
            cps_api_object_t obj = cps_api_object_list_get(gp.list,ix);
            cps_api_object_attr_t vlan_id = cps_api_object_attr_get(obj,BASE_MAC_QUERY_VLAN);
            cps_api_object_attr_t ifindex = cps_api_object_attr_get(obj,BASE_MAC_QUERY_IFINDEX);
            cps_api_object_attr_t mac_addr = cps_api_object_attr_get(obj,BASE_MAC_QUERY_MAC_ADDRESS);

            printf("VLAN ID %d\n",cps_api_object_attr_data_u16(vlan_id));
            printf("Ifindex %d\n",cps_api_object_attr_data_u32(ifindex));
            char mt[6];
            char mac_string[20];
            memcpy(mt, cps_api_object_attr_data_bin(mac_addr), 6);
            sprintf(mac_string, "%x:%x:%x:%x:%x:%x", mt[0], mt[1], mt[2], mt[3], mt[4], mt[5]);
            printf("MAC Address %s\n",mac_string);
        }
    }

    // Close the get the request
    cps_api_get_request_close(&gp);
    return rc;
}
```

Example: Delete a MAC Table Entry using Python

1. Import CPS Utility Python Library

```
import cps_utils
```

2. Register the attribute type as mac-address to convert between the string and byte-array format.

```
cps_utils.add_attr_type("base-mac/table/mac-address", "mac")
```

3. Delete the static MAC address entry; specify a MAC address, interface index and VLAN.

```
d =  {"mac-address": "00:0a:0b:cc:0d:0e",
         "ifindex": 18,
         "vlan": "100"}
```

4. Create a CPS object.

```
obj = cps_utils.CPSObject('base-mac/table',data= d)
```

5. Add the operation to the object.

```
tr_obj = ('delete', obj.get())
```

6. Create a transaction object.

```
transaction = cps_utils.CPSTransaction([tr_obj])
```

7. Commit the transaction.

```
ret = transaction.commit()
```

8. Check the response and print appropriate message.

```
if not ret:
    raise RuntimeError("Error deleting entry from MAC Table")
```

Code Block: Delete a MAC Table Entry using Python

```
import cps_utils

cps_utils.add_attr_type("base-mac/table/mac-address", "mac")

d =  {"mac-address": "00:0a:0b:cc:0d:0e",
     "ifindex": 18,
     "vlan": "100"}

obj = cps_utils.CPSObject('base-mac/table',data= d)
tr_obj = ('delete', obj.get())

transaction = cps_utils.CPSTransaction([tr_obj])

ret = transaction.commit()

if not ret:
    raise RuntimeError("Error deleting entry from MAC Table")
```

Example: Delete a MAC Table Entry using C/C++

```
#include "cps_api_object.h"
#include "dell-base-l2-mac.h"
#include "cps_class_map.h"
#include "cps_api_object_key.h"

#include <stdint.h>
#include <net/if.h>

bool cps_delete_mac(){
   // Create and initialize the transaction object
   cps_api_transaction_params_t tran;
   if (cps_api_transaction_init(&tran) != cps_api_ret_code_OK ){
       return false;
   }

   // Create and initialize the key
   cps_api_key_t key;
   cps_api_key_from_attr_with_qual(&key, BASE_MAC_TABLE_OBJ, cps_api_qualifier_TARGET);

   // Create the object
   cps_api_object_t obj = cps_api_object_create();

   if(obj == NULL ){
       cps_api_transaction_close(&tran);
       return false;
   }

   // Set the key for the obejct
   cps_api_object_set_key(obj,&key);

   // Add attributes mandatory to create MAC address entry
   uint8_t mac_addr[6] = {0x0,0xa,0xb,0xc,0xd,0xe};
```

```
    uint16_t vlan_id = 131;

    cps_api_object_attr_add(obj,BASE_MAC_TABLE_MAC_ADDRESS, mac_addr, sizeof(hal_mac_addr_t));
    cps_api_object_attr_add_u32(obj,BASE_MAC_TABLE_IFINDEX,if_nametoindex("e101-001-0") );
    cps_api_object_attr_add_u16(obj,BASE_MAC_TABLE_VLAN,vlan_id);

    // Add the object along with the operation to transaction
    if(cps_api_delete(&tran,obj) != cps_api_ret_code_OK ){
        cps_api_delete_object(obj);
        return false;
    }

    // Commit the transaction
    if(cps_api_commit(&tran) != cps_api_ret_code_OK ) {
        cps_api_transaction_close(&tran);
        return false;
    }

    // Cleanup the Transaction
    cps_api_transaction_close(&tran);

    return true;
}
```

NOTE: To delete the MAC address from all VLANs, specify the **`vlan`** attribute and its value in the object. To delete all MAC entries from an interface, specify the **`ifindex`** attribute and its value in the object. To delete MAC entries from both a VLAN and member interface, specify the **`vlan`** and **`ifindex`** attributes and their values in the object.

NOTE: Deletion of **static** entries based only on VLAN, interface or a VLAN/Interface combination is not supported. To delete a static entry, you must add the **`mac-address,vlan`** and **`ifindex`** attributes and their values to the object.

Example: Remove MAC Table Entries from Multiple VLANs using Python

To remove MAC entries from multiple VLANs (and/or interfaces) in a single CPS call:

1. Import the cps_utils python library.

```
import cps_utils
```

2. Enter the VLANs from which you want to remove MAC entries.

```
vlan_list =[1,2,3,4,5]
```

3. Create a CPS object.

```
obj = cps_utils.CPSObject('base-mac/flush')
```

4. Add the list of VLANs to the object.

```
count = 0
el = ["input/filter","0","vlan"]
for vlan in vlan_list:
       obj.add_embed_attr(el, vlan)
       count = count + 1
       el[1] = str(count)
```

5. Add the operation to the object.

```
tr_obj = ('rpc', obj.get())
```

6. Create a transaction object.

```
transaction = cps_utils.CPSTransaction([tr_obj])
```

7. Commit the transaction.

```
ret = transaction.commit()
```

8. Verify the return code and print appropriate message.

```
if not ret:
    raise RuntimeError("Error Flushing entries from MAC Table")
```

Code Block: Remove MAC Table Entries from Multiple VLANs using Python

```
import cps_utils

vlan_list =[1,2,3,4,5]
obj = cps_utils.CPSObject('base-mac/flush')

count = 0
el = ["input/filter","0","vlan"]
for vlan in vlan_list:
       obj.add_embed_attr(el, vlan)
       count = count + 1
       el[1] = str(count)

tr_obj = ('rpc', obj.get())
transaction = cps_utils.CPSTransaction([tr_obj])
ret = transaction.commit()

if not ret:
    raise RuntimeError("Error Flushing entries from MAC Table")
```

Example: Remove MAC Table Entries from Multiple VLANs using C/C++

```
#include "cps_api_object.h"
#include "dell-base-l2-mac.h"
#include "cps_class_map.h"
#include "cps_api_object_key.h"

#include <stdint.h>
#include <net/if.h>

bool cps_flush_mac(){
   // Create and initialize the transaction object
   cps_api_transaction_params_t tran;
   if (cps_api_transaction_init(&tran) != cps_api_ret_code_OK ){
       return false;
   }

   // Create and initialize the key
   cps_api_key_t key;
   cps_api_key_from_attr_with_qual(&key, BASE_MAC_FLUSH_OBJ, cps_api_qualifier_TARGET);

   // Create the object
   cps_api_object_t obj = cps_api_object_create();
```

```
    if(obj == NULL ){
        cps_api_transaction_close(&tran);
        return false;
    }

    // Set the key for the obejct
    cps_api_object_set_key(obj,&key);

    // Add attributes to Flush MAC entries
    cps_api_attr_id_t ids[3] = {BASE_MAC_FLUSH_INPUT_FILTER,0, BASE_MAC_FLUSH_INPUT_FILTER_VLAN };
    const int ids_len = sizeof(ids)/sizeof(ids[0]);

    uint16_t vlan_list[3]={1,2,3};
    for(unsigned int ix=0; ix<sizeof(vlan_list)/sizeof(vlan_list[0]); ++ix){
        ids[1]=ix;
        cps_api_object_e_add(obj,ids,ids_len,cps_api_object_ATTR_T_U16,&(vlan_list[ix]),sizeof(vlan_list[ix]));

    }

    unsigned int ifindex_list[] = { if_nametoindex("e101-001-0"),if_nametoindex("e101-002-0"),
                                    if_nametoindex("e101-003-0")};
    ids[2]=BASE_MAC_FLUSH_INPUT_FILTER_IFINDEX;
    for(unsigned int ix=0; ix<sizeof(ifindex_list)/sizeof(ifindex_list[0]); ++ix){
        ids[1]=ix;

cps_api_object_e_add(obj,ids,ids_len,cps_api_object_ATTR_T_U16,&ifindex_list[ix],sizeof(ifindex_list[ix]));
    }

    // Add the object along with the operation to transaction
    if(cps_api_action(&tran,obj) != cps_api_ret_code_OK ){
        cps_api_object_delete(obj);
        return false;
    }

    // Commit the transaction
    if(cps_api_commit(&tran) != cps_api_ret_code_OK ) {
        cps_api_transaction_close(&tran);
        return false;
    }

    // Cleanup the Transaction
    cps_api_transaction_close(&tran);

    return true;
}
```

Example: Register for Events using Python

1. Import the CPS Python library.

```
import cps
```

2. Create a handle to connect to event service.

```
handle = cps.event_connect()
```

3. Register a key with the event service to receive notification when an event for this key is published.

```
cps.event_register(handle, cps.key_from_name('observed','base-port/interface'))
while True:
       # wait for the event
       o = cps.event_wait(handle)
       print o
```

Code Block: Register for Events using Python

```
import cps

handle = cps.event_connect()

cps.event_register(handle, cps.key_from_name('observed','base-port/interface'))
while True:
       o = cps.event_wait(handle)
       print o
```

Example: Register for Events using C/C++

```
#include "cps_api_events.h"
#include "cps_api_object.h"
#include "dell-base-phy-interface.h"
#include "cps_class_map.h"
#include "cps_api_object_key.h"

#include <stdlib.h>
```

```
#include <stdio.h>
#include <unistd.h>

//Callback for the interface event handling
static bool cps_if_event_cb(cps_api_object_t obj, void *param){

   char buf[1024];
   cps_api_object_to_string(obj,buf,sizeof(buf));
   printf("Object Received %s \n",buf);
   return true;
}

bool cps_reg_intf_events(){

   // Initialize the event service
   if (cps_api_event_service_init() != cps_api_ret_code_OK) {
         return false;
    }
   // Initialize the event handling thread
   if (cps_api_event_thread_init() != cps_api_ret_code_OK) {
       return false;
   }

   //Create and initialize the key
   cps_api_key_t key;
   cps_api_key_from_attr_with_qual(&key, BASE_PORT_INTERFACE_OBJ,
                                    cps_api_qualifier_OBSERVED);

   //Create the registration object
   cps_api_event_reg_t reg;
   memset(&reg,0,sizeof(reg));

   reg.number_of_objects = 1;
   reg.objects = &key;

   // Register to receive events for key created above
   if (cps_api_event_thread_reg(&reg, cps_if_event_cb,NULL)!=cps_api_ret_code_OK) {
      return false;
   }

   //Wait for the events
   while(1){
       sleep(1);
   }

   return true;

}
```

NOTE: CPS Python does not support a C/C++ API to to register a callback when events are published.

Example: Publish Events using Python

1. Import cps and cps_utils libraries.

```
import cps
import cps_utils
```

2. Create a handle to connect to the event service.

```
handle = cps.event_connect()
```

3. Create an object.

```
obj = cps_utils.CPSObject('base-port/interface',qual='observed',
                           data= {"ifindex":23})
```

4. Publish an object.

```
cps.event_send(handle, obj.get())
```

Code Block: Publish Events using Python

```
import cps
import cps_utils

handle = cps.event_connect()

obj = cps_utils.CPSObject('base-port/interface',qual='observed',
                           data= {"ifindex":23})

cps.event_send(handle, obj.get())
```

Example: Publish Events using C/C++

```
#include "cps_api_events.h"
#include "cps_api_object.h"
#include "dell-base-phy-interface.h"
#include "cps_class_map.h"
#include "cps_api_object_key.h"

#include <stdio.h>
#include <net/if.h>

bool cps_pub_intf_event() {

    static cps_api_event_service_handle_t handle;
    if (cps_api_event_service_init() != cps_api_ret_code_OK) {
        return false;
    }
    if (cps_api_event_client_connect(&handle) != cps_api_ret_code_OK) {
        return false;
    }

    //Create and intialize the key
    cps_api_key_t key;
    cps_api_key_from_attr_with_qual(&key, BASE_PORT_INTERFACE_OBJ,
                                cps_api_qualifier_OBSERVED);

   // Create the object
   cps_api_object_t obj = cps_api_object_create();

   if(obj == NULL){
       return false;
   }

   // Add attributes to the object
   cps_api_object_attr_add_u32(obj,BASE_PORT_INTERFACE_IFINDEX,
                            if_nametoindex("e101-001-0"));

   //Set the Key to the object
   cps_api_object_set_key(obj,&key);

   //Publish the object
   if(cps_api_event_publish(handle,obj)!= cps_api_ret_code_OK){
       cps_api_object_delete(obj);
       return false;
   }
   // Delete the object
   cps_api_object_delete(obj);
   return true;

}
```

CPS API Reference

The OS10 website provides reference documentation for:

- CPS C/C++ API
- CPS Python API
- OS10 YANG models

The YANG model files and C header files derived from the YANG models are included in the development packages provided for OS10. You can download the development packages from the OS10 website.

Troubleshooting

Overview

This section describes the methods and tools available for gathering information and debugging OS10.

OS10 Linux and Network Debugging

Standard utilities, such as user and file management commands, are included with the OS10 image. Use these utilities for debugging.

Packet Analysis

`tcpdump` is a Linux packet analyzer tool used to capture packets on any network interface. Only a privileged user can execute the command. For more options and support, refer to the Linux manual page for tcpdump http://linux.die.net/man/8/tcpdump .

For example, to capture all the packets on physical ports of the switch, enter:

```
$ tcpdump -i e101-003-0

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on e101-003-0, link-type EN10MB (Ethernet), capture size 262144 bytes
01:39:22.457185 IP 3.3.3.1 > 3.3.3.4: ICMP echo request, id 5320, seq 26, length 64
01:39:22.457281 IP 3.3.3.1 > 3.3.3.4: ICMP echo reply, id 5320, seq 26, length 64
```

Coredumps

Core image files are saved in the /var/coredumps directory.

Get Port Statistics

Use the following command to dump interface statistics:

```
$ os10-show-stats

os10-show-stats

          if_stat {iface_name} {filter_list}    - Get stats for all interfaces if no input provided
                                                - Get the statistics of given interface
                                                - filter_list is the filters if user want
                                                  only specific statistics

          vlan_stat [vlan_ifindex] {filter_list} - Get the statistics of given vlan ifindex
                                                 - filter_list is the filters if user want
                                                   only specific statistics
          clear [iface_name]                    - Clear the interface statistics
```

For example: `os10-show-stats if_stat e101-001-0`

For more information on this script and its use, refer to Monitoring (os10-show-stats).

Firmware Versions

To display version information about installed firmware, open the /var/log/firmware_versions file.

Transceivers

To display information about installed transceivers, enter the following command:

```
$ os10-show-transceivers summary

Front Panel Port                Media Type          Part Number        Serial Number   DellQualified
  1                         QSFP 40GBASE SR4        FTL410QE1C            MLJ004C         No
```

Debugging Interfaces

During system startup, physical ports are mapped to Linux network interfaces. Refer to the Physical Interfaces section in Networking Features for more details regarding the naming convention of Linux network interfaces.

- Use the `os10-switch-shell ps` command to list all ports created in the NPU.
- Use the `hshell` command to verify the administrative and operating status of an interface in the NPU.

```
 os10-switch-shell "ps xe4"   >> indicated port is enabled in NPU, but no admin or oper up

          ena/    speed/ link auto    STP                  lrn  inter   max  loop
     port  link    duplex scan neg?   state   pause  discrd ops   face frame  back
      xe4  !ena   40G  FD   SW  No   Forward          None   FA  XGMII  1528

 os10-switch-shell "ps xe4"   >> indicated port is enabled in NPU, admin and oper up.

         ena/    speed/ link auto    STP                  lrn  inter   max  loop
```

```
    port  link    duplex scan neg?   state   pause  discrd ops   face frame  back
     xe4  up  40G  FD   SW  No   Forward          None   FA  XGMII  1528

os10-switch-shell "ps xe4"   >> indicated port is enabled in NPU, admin up but oper up.

         ena/    speed/ link auto    STP                  lrn  inter   max  loop
    port  link    duplex scan neg?   state   pause  discrd ops   face frame  back
     xe4  down 40G  FD   SW  No   Forward          None   FA  XGMII  1528
```

Troubleshooting Tips

Problem: Linux network interfaces are not created

Resolution:

- Check that the NAS process is running. Check /var/log/syslog for errors.
- If NAS is not running, check that services on which NAS depends are running.

Problem: Interfaces not enabled in the NPU

Resolution:

- Check that SAI/NPU SDK have initialized correctly. Check /var/log/syslog for errors.

Layer 3 Troubleshooting

Use the following Linux commands to verify routing-related tables:

- `ip route show`
- `arp -a`
- IPv6 debugging:
  - `ip -6 neighbor show`
  - `ip -6 route show`

To troubleshoot routing issues in the NPU:

- `os10-switch-shell "l3 defip show"`
- `os10-switch-shell "l3 l3table show"`
- `os10-switch-shell "l3 egress show"`
- For IPv6 routes:
  - `os10-switch-shell "l3 ip6route show"`
  - `os10-switch-shell "l3 ip6host show"`

- Multipath:
  - `os10-switch-shell "l3 multipath show"`
  - `os10-switch-shell "l3 egress show"`
- Traffic:
  - `os10-switch-shell "show c"`

To enable NAS and SAI Layer 3 logging:

- `os10-logging enable ROUTE`
- `os10-logging enable INTERFACE`
- `os10-logging enable NETLINK`
- `os10-logging enable SAI_NEXT_HOP`
- `os10-logging enable SAI_ROUTE`

NOTE: The NPU debug commands mentioned above are for illustration purposes only. Refer to the respective NPU SDK for further details.

Layer 2 Troubleshooting

To verify that all Linux network interfaces are created:

- `ip link show`

To verify VLANs and STP:

- `brtcl show`
- `brctl showstp <br_name>`

To enable NAS and SAI Layer 2 logging:

- `os10-logging enable NAS_L2`
- `os10-logging enable INTERFACE`
- `os10-logging enable L2MAC`
- `os10-logging enable SAI_FDB`
- `os10-logging enable SAI_STP`

To display layer status in the NPU:

- `os10-switch-shell "l2 show"`
- `os10-switch-shell "stg show"`
- `os10-switch-shell "vlan show"`

Log Management

Syslog is a utility for tracking and logging many types of system messages, including informational and extremely critical messages. Logging for all OS10 modules is routed to /var/log/syslog. Examples of PAS andNAS logging are shown in this section.

Application Logging Format

```
date <timestamp> <hostname> <processname> <filename> <functionname>
<line #> string
```

Example: PAS Logging

```
Feb 16 18:16:52 OS10 pas_svc: [PAS:PAS]:pas_entity.c:dn_entity_poll:366, PSU 1 is present
Feb 16 18:16:52 OS10 pas_svc: [PAS:PAS]:pas_entity.c:dn_entity_poll:366, Fan Tray 1 is present
```

Example: NAS Logging

```
Jan 24 18:49:18 OS10 nas_svc: [INTERFACE:INT-CREATE]:port/nas_int_port.cpp:nas_int_port_create:347, Interface
created 0:29:e101-021-0 - 22
Jan 24 18:49:18 OS10 nas_svc: [INTERFACE:NAS-INT-CREATE], Interface e101-021-0 initial link state is 2
Jan 24 18:49:18 OS10 nas_svc: [INTERFACE:INT-STATE]:port/nas_int_port.cpp:nas_int_port_link_change:312,
Interface state change 0:29 to 2
```

OS10 application-specific logging is controlled by the os10-logging script on the target.

```
os10-logging

====================================================================================================
[show-id]                                                  - displays ids of modules, log-levels and sub-
levels
[show] [all] | [module-id] {log-level} {log-sub-level}     - displays current logging status for all/given
                                                            sub-sytem/given module and log-level/given
                                                            module, log-level and sub-log-level
[enable] [all] | [module-id] {log-level } {log-sub-level}  - Enables logging status for all/given
                                                            sub-sytem/given module and log-level/given
                                                            module, log-level and sub-log-level
[disable] [all] | [module-id] {log-level } {log-sub-level} - Disables logging status for all/given
```

```
                                                            sub-sytem/given module and log-level/given
                                                            module, log-level and sub-log-level
 NOTE :1. For enable and disable log-level and log-sub-level is optional when using module-id.
        If only module-id is given it will enable/disable all log-levels and log-sub-levels for that
        module-id, similarly if module-id and log-level is given, it will enable All log-sublevel for
        the module-id and log-level
       2. Instead of Module Ids now you can use the module name as string as well, for eg.
        os10-logging enable L3_SERVICES
====================================================================================================


Module IDs
========================================
NULL                             0
NPU                              1
BOARD                            2
SYSTEM                           3
QOS                              4
COM                              5
INTERFACE                        6
NETLINK                          7
ROUTE                            8
ACL                              9
SFP                              10
MGMT                             11
DSAPI                            12
DB_SQL                           13
NDI                              14
SAI_FDB                          15
SAI_VLAN                         16
SAI_PORT                         17
SAI_SWITCH                       18
SAI_ROUTER                       19
SAI_ROUTER_INTF                  20
SAI_NEIGHBOR                     21
SAI_NEXT_HOP                     22
SAI_NEXT_HOP_GROUP               23
SAI_ROUTE                        24
SAI_MIRROR                       25
NAS_L2                           26
NAS_COM                          27
SAI_LAG                          31
L2MAC                            36
L2CMN                            37
NAS_OS                           38
SAI_STP                          39
BGP                              40
RTM                              43
L3CMN                            44
```

```
PAS                                 47
PAS_FUSE                            48
PAS_TIMER                           49
SAI_SAMPLEPACKET                    50
ENV_TMPCTL                          51
SAI_HOSTIF                          59
SAI_QUEUE                           60
SAI_MAPS                            61
SAI_POLICER                         62
SAI_WRED                            63
SAI_SCHEDULER                       64
SAI_SCHEDULER_GRP                   65
SAI_QOS                             66
SERV_HW                             67
DATASTORE                           68
L3_SERVICES                         69
IPM                                 70
MGMT_INTF                           71
FWD_SVCS_SFLOW                      73
SAI_UDF                             78
SAI_BUFFERS                         79
SWITCH_RES_MGR                      80
============================


Log Level IDs
========================================
ERR                                 3
INFO                                6
DEBUG                               7


Log Sub Level IDs
========================================
CRITICAL                            0
MAJOR                               1
MINOR                               2
WARNING                             3
```

The SAI module has its own specific logging. To turn on SAI-specific logs, follow these steps:

```
os10-switch-log set [module_name] [level_name]
- Set the given module name's logging level to
 given level_name
```

```
eg. os10-switch-log set ALL debug

module_name

{'WRED': 18, 'FDB': 3, 'ROUTE': 6, 'VLAN': 4, 'HOST_INTERFACE': 12, 'ACL': 11, 'MIRROR': 13, 'QOS_QUEUE': 20,
'SCHEDULER_GROUP': 22, 'PORT': 2, 'VIRTUAL_ROUTER': 5, 'NEXT_HOP_GROUP': 8, 'SWITCH': 1, 'POLICER': 17,
'NEIGHBOR': 10, 'UNSPECIFIED': 0, 'SAMPLEPACKET': 14, 'QOS_MAPS': 19, 'STP': 15, 'ALL': 23, 'LAG': 16,
'ROUTER_INTERFACE': 9, 'NEXT_HOP': 7, 'SCHEDULER': 21}

level_name

{'info': 2, 'notice': 3, 'warning': 4, 'critical': 6, 'error': 5, 'debug': 1}
```

sosreport

`sosreport` is a standard Linux tool for collecting system information. For a description of `sosreport`, refer to: http://sos.readthedocs.org/en/latest.

sosreport plugins

Plugins are a means to provide extensibility to the reporting mechanism and enable you to specify the information, logs and configuration data you want to collect for your modules.

Information about writing a plugin is provided at:
https://github.com/sosreport/sos/wiki/How-to-Write-a-Plugin

Several modules of OS10 provide plugins to `sosreport` so that it can collect status information about various components of OS10 software and hardware. In order to execute sosreport, use the following command:

```
#sosreport --batch -a[...]
Creating compressed archive...
Your sosreport has been generated and saved in:
/tmp/sosreport-OS10-20160219191403.tar.gz
The checksum is: a6d3508bde238e46c8cc7923f10c861c
```

This command executes all OS10 plugins and generates a compressed tar file (.tar.gz) file as output. The name of the file is provided in the output of the command. For example, the name of the tar file generated for the previous example is:

sosreport-OS10-20160219191403.tar.gz

The following information is contained in the generated tar file:

- OS10 version
- ACL counters and statistics
- ACL table contents
- Port mirroring setup
- Physical port setup
- sFlow setup

- STP setup
- Transceiver status and setup
- Linux interface setup and statistics
- Hardware platform information
- NPU low level setup information

In addition, sosreport collects log files and coredumps.

CPS API Object Management

Refer to the Programmability chapter for more information on CPS API objects.

get Object

Use the `cps_get_oid.py` command to retrieve and display the contents of a CPS API object.

```
cps_get_oid.py category/subcategory [ key=value ] ...
```

Where:

category	Category of the requested CPS API object
subcategory	Subcategory of the requested CPS API object
key	Name of object key attribute
value	Value for given key attribute

Example

The following command retrieves the entity object for the PSU in slot 1.

```
cps_get_oid.py base-pas/entity entity-type=1 slot=1
```

set Object

Use the `cps_set_oid.py` command to set one or more attributes of a CPS API object,

```
cps_set_oid.py category/subcategory [ key=value ] ... [ attr=value ] ...
```

Where:

category	Category of the requested CPS API object
subcategory	Subcategory of the requested CPS API object
key	Name of object key attribute
attr	Name of object attribute to set
value	Value for given attribute

Example

The following command sets the beacon LED on.

```
cps_set_oid.py base-pas/led entity-type=3 slot=1 name=Beacon on=1
```

Event Trace

Use the `cps_trace_events.py` command to display CPS API events as they occur.

```
cps_trace_events.py qualifier.category
```

`Where:`

qualifier	`Qualifier of the requested CPS API object to trace`
category	`Category of the requested CPS API object to trace`

Note: you must enter the command as a CPS API key in dotted-decimal format. Consult the section on CPS API keys for how to form a key, and the appropriate header files for actual key values.

Example

The following command prints all CPS API events generated by the PAS service ("observed" qualifier)**:**

```
cps_trace_events.py 2.19
```

Password Recovery

Follow these steps to recover your password.

1. Connect to the serial console port. The serial settings are: 115200 baud, 8 data bits, no parity.

2. Reboot or power up the system.

3. Press **ESC** at the GNU GRUB prompt.

OS10 GNU GRUB Boot Menu

```
+----------------------------------------------------------------------------+
|*OS10-A                                                                     |
| OS10-B                                                                     |
| ONIE                                                                       |
+----------------------------------------------------------------------------+
```

4. Press **e** for edit on OS10-A which places you in the OS10 GRUB editor.

5. Use the arrow keys to highlight the line that begins with *linux* and at the very end of the line, add `init=/bin/bash`.

OS10 GRUB editor

```
+----------------------------------------------------------------------------+
|setparams 'OS10-A'                                                          |
|                                                                            |
|    set root='(hd0,gpt7)'                                                   |
|    echo    'Loading OS10 ...'                                              |
|    linux   (hd0,gpt7)/boot/os10.linux console=ttyS0,115200 root=/dev/sda7 \|
|rw init=/bin/bash                                                           |
|    initrd  (hd0,gpt7)/boot/os10.initrd                                     |
+----------------------------------------------------------------------------+
```

6. Press **ctrl+x** to boot your system. Your system will boot up to a password-less root shell.

7. At the prompt, type in: `passwd yourusername.`

```
root@OS10:/# passwd linuxadmin &10;
```

8. Executing Step 7 prompts you to enter new password. Enter the new password.

```
root@OS10:/# passwd linuxadmin
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

9. Reboot the system by entering the `reboot -f` command to boot with OS10.

```
root@OS10:/# reboot -f
Rebooting.[  822.327073] sd 0:0:0:0: [sda] Synchronizing SCSI cache

[  822.340656] reboot: Restarting system

[  822.344339] reboot: machine restart
BIOS (Dell Inc) Boot Selector
S6000-ON 3.20.0.0 (32-port TE/FG)
```

10. Enter the new password created to log in to the system.