OpenSwitch OPX Developers Guide

Release 3.0.0

DELLEMC

Contents

Getting started

OpenSwitch OPX is implemented using a standard Linux distribution—Debian Jessie. OpenSwitch OPX is binary-compatible with Debian Linux packages.

Linux kernel	Unmodified Linux kernel included with Debian distribution provides a robust base to support current state-of-the-art and future networking.
Linux IP stack	Rich feature set provided by the Linux standard IP stack without vendor-specific changes.
Linux tools	Standard Linux system administration tools are factory-installed in OpenSwitch OPX, 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. OpenSwitch OPX allows you to easily deploy the management and orchestration solutions that are typically available for Linux servers and storage systems.

Programmability

OpenSwitch OPX provides an object-centric API for application development—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, and OpenSwitch OPX APIs support Python and C/C++. A set of standard Debian software development packages is provided to allow you to develop applications for OpenSwitch OPX.

Open platform abstraction

OpenSwitch OPX implements a new, open object-centric application programming interface called the control plane service (CPS) application programming interface (API). The CPS API allows customer-developed applications to be independent of any underlying hardware or software technology. OpenSwitch OPX internally uses the switch abstraction interface (SAI) which Dell and partner companies contributed to the Open Compute Project. The SAI API allows OpenSwitch OPX to be independent of any network processor/switch hardware technology. See opencompute.org for more information about SAI.

System hardware integration with standard Linux APIs

OpenSwitch OPX integrates standard Linux networking APIs with the hardware functionality provided by networking devices—system and network processors. You can download and use open source software (such as Quagga and Nagios) on any OpenSwitch OPX platform.

Disaggregated hardware and software

OpenSwitch OPX provides an environment in which hardware and software are fully modular. You can select the software modules you want to install, and the hardware platforms you would like to use for your networking needs.
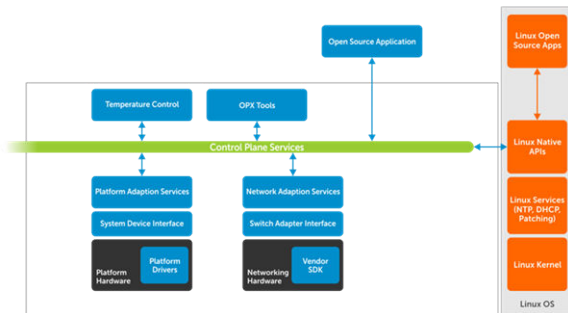
Topics:

Architecture

The main OpenSwitch OPX components include:

- Linux infrastructure
- Control plane services (CPS)
- Switch abstraction interface (SAI)
- Network adaptation service (NAS)
- System device interface (SDI)
- Platform adaptation service (PAS)
- Dell EMC applications and tools

Using well-defined APIs allow OpenSwitch OPX to provide full software modularity and abstraction of the hardware and software platforms.

Software partitioning	Software is partitioned into subcomponents that are organized as Linux packages—each 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 platform startup scripts are the only hardware-specific components in OpenSwitch OPX—all other modules are hardware-independent. Hardware-specific variations (such as number and names of physical ports, or number of power supplies) are defined using platform definition files. OpenSwitch OPX uses portable operating system interface (POSIX) APIs. Software platform implementation-specific details are abstracted using OpenSwitch OPX run-time libraries (common utilities and logging) as necessary.
Open API	User applications interact with OpenSwitch OPX modules using the CPS API, and OpenSwitch OPX provides an object-centric API in the control plane services (CPS) component.

Linux infrastructure

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

Control plane services

Control plane services (CPS) is at the core of the architecture, providing an object-centric framework that mediates interactions between OpenSwitch OPX applications and allows customer applications to interact with OpenSwitch OPX components.

There are two sets of application roles—clients and servers:

Client applications	Execute, create, set, get, and delete operations on individual objects or lists of objects.
Server applications	Execute operations requested by CPS client applications. Because client applications operate on objects, they are not aware of the location or name of the CPS server application that executes a requested operation.

The CPS framework supports a publisher/subscriber mode, and server applications can publish relevant events, while client applications can subscribe (register) for specific events and objects. CPS client applications can register for events when objects are created, modified,

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

The CPS API object model provides disaggregation between client and server applications. The client and server are unaware of one other —they communicate only through a CPS API object (see Client and server applications).

Custom-written applications use the CPS API to communicate with OpenSwitch OPX components. The OpenSwitch OPX provides both C/C++ and Python programming interfaces. The object model provided by the CPS layer is defined using YANG models which are used to generate C header files—providing a programmatic representation of objects and their attributes. The header files are shared between client and server applications—the C/C++ representation of objects and their attributes is designed to ensure compatibility between multiple versions of the object model.

Switch abstraction interface

The OpenSwitch OPX switch abstraction interface (SAI) implements an API for network processor units (NPUs) supported on Dell EMC platforms. 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. OpenSwitch OPX is NPU-independent and not locked into specific NPU hardware. If a new NPU is used in an OpenSwitch OPX platform, the only component that Dell EMC engineers replace is the SAI.

Network adaptation service

The network adaptation service (NAS) manages the high-level NPU abstraction and adaptation, 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 provides adaptation of low-level switch abstraction provided by the SAI to standard Linux networking APIs and interfaces, and to software CPS API functionality. The NAS manages the middleware that associates physical ports to Linux interfaces—also provides packet I/O services, using the Linux kernel IP stack.

System device interface

All hardware components except for NPUs are abstracted as system devices. The system device interface (SDI) API defines a low-level platform-independent abstraction for all types of system devices. Only system device drivers that implement the SDI API are hardware-specific—the API itself is hardware-independent.

Example system devices:

- Fan devices
- Power supplies
- Temperature sensors
- LEDs
- EEPROM
- Programmable devices
- Transceivers

Platform adaptation service

The platform adaptation service (PAS) provides a higher-level abstraction and aggregation of the functionality provided by the SDI component, and implements the CPS object models associated with system devices. The PAS monitors the status of system devices and reports changes or faults as CPS events. The PAS also allows applications to retrieve current status information and set control variables of system devices.

- 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 and get operations on transceivers

The PAS detects:

- Insertion and removal of common field replaceable units (FRUs), such as PSUs and fans
- Over-temperature based on pre-defined thresholds
- Media insertion on physical ports

PAS is responsible for:

- Monitoring the status of system devices
- Allowing applications to retrieve current status information
- Reporting status changes or faults as CPS events
- 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 PSUs. This component contains the platform-specific system startup configuration and scripts.

Dell EMC applications and tools

OpenSwitch OPX provides a set of tools and commands that allow system administrators to manage Dell EMC-specific software and hardware functionality (such as software upgrades, physical port, media information, and system inventory). OpenSwitch OPX provides a Dell EMC-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 components

OpenSwitch OPX services

- `opx-cps` — starts the CPS broker which mediates all operations and events
- `opx-pas` — PAS service which executes PAS functionality
- `opx-nas` — NAS service which executes NAS functionality
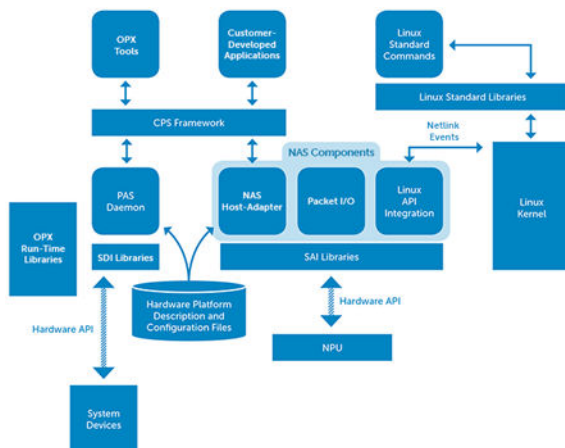
Applications which manage OpenSwitch OPX system components

- `opx-acl-init` — initializes default ACL configuration or control-plane protocol packets
- `opx-create-interface` — gets/sets interface parameters using the CPS API
- `opx-front-panel-ports` — manages physical port mapping to Linux interfaces
- `opx-ip` — gets/sets IP address parameters using the CPS API
- `opx-phy-media-config` — manages configuration of physical media
- `opx-platform-init` — initializes default platform-specific configuration
- `opx-qos-init` — initializes default QoS configuration
- `opx-monitor-phy-media` — monitors physical media (SFP) events generated by PAS when you insert a pluggable module (automatically configures port parameters)
- `opx-nas-init` — initializes default NAS configuration
- `opx-nas-shell` — runs NPU shell commands
- `opx-tmpctl` — manages the environment temperature control and executes the thermal control algorithm

> ⚠️ CAUTION: Do not disable the `opx-tmpctl` as hardware damage may result.

NAS Linux adaptation— integration with standard Linux APIs

The OpenSwitch OPX 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 enable/disable an interface, add/remove a route, or add/delete a VLAN).

Linux interfaces associated with physical ports	OpenSwitch OPX uses internal Linux *tap* devices to associate physical ports on the NPU with Linux interfaces. When a change in physical port status (up/down) occurs, the NAS daemon propagates the new port status to the associated tap device.
Packet I/O	Packet I/O describes control plane packet forwarding between physical ports and associated Linux interfaces— implemented as a standalone thread of the NAS daemon. Packets received by the NPU are forwarded to the CPU, and the packet I/O thread receives the packet through a SAI API callback. Each received packet contains the identity of the source physical port. The 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 standard sockets. Applications use tap devices to transmit packets, and the packet I/O receives the transmitted packet from the Linux IP stack. Based on the source tap device of the packet, the transmitted packet is forwarded to the associated physical port.

CPS services

The NAS daemon registers with the 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 OpenSwitch OPX. The PAS daemon also registers with CPS as a server application to provide CPS programmability for system devices.

File system organization

OpenSwitch OPX uses a standard Linux file system. The OpenSwitch OPX-specific system tools and configuration files are maintained under the following directory structure:

- */usr/bin* — contains binaries
- */usr/lib* — contains libraries
- */etc/opx* — contains platform description files and default configuration files

Platform description files	The platform files contain a description of hardware modules that apply to the current platform, such as number of physical ports, fans, and power supplies.
Default configuration files	The default configuration files contain initialization information applicable to the current platform, such as the initial SAI configuration and system ACL rules to be applied at initialization.

System startup

OpenSwitch OPX leverages the `systemd` framework for the startup of OpenSwitch OPX-specific processes. The `systemd` framework is enabled by default under Debian Jesse (see www.freedesktop.org/wiki/Software/systemd).

Boot sequence

After you install an OpenSwitch OPX image, the system automatically loads the image and boots.

1   After the switch powers up or reboots, the boot menu displays. After a short delay, the system auto boots by loading the image—in this case OPX-A partition. If required, during the delay you can interrupt the auto boot and select other options to select `OPX-B` to load another software image, or go back to `ONIE` for upgrades, system recovery, and so on.

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

2   Linux boots from the OPX-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 `systemd` daemon starts custom services during system initialization:

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

After OpenSwitch OPX custom services run successfully and the system boots up, the Linux prompt displays on the console for you to log in.

(i) | **NOTE:** If the service that creates internal Linux interfaces is unsuccessful, the system bootup waits 300 seconds before timing out and displays the Linux login prompt. Log in to OpenSwitch OPX and use the troubleshooting steps to determine the cause of the failure. You can use the `systemctl` command to determine if any services have failed, and the `journalctl` command to inspect the log contents.

Programmability

The control plane service (CPS) infrastructure is at the core of system programmability. The CPS supports the definition of a well-defined, data-centric application programming interface (CPS API) that allows customer applications to interact with system services. System applications also use the CPS API to communicate with one another.

The CPS infrastructure provides:

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

Topics:

Client and server applications

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

A temperature control (TC) application is a simple OpenSwitch OPX implementation that shows how the CPS API object model disaggregates client and server applications. The client is the TC application, and the server is the platform adaptation services (PAS) component.
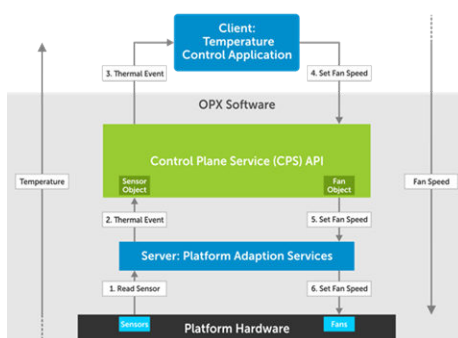
The TC application increases the speed of a system fan when the value reported by a temperature sensor exceeds a predefined threshold —it decreases the fan speed when the temperature falls below the threshold. The TC application needs to *subscribe* to temperature threshold crossing events published by the OpenSwitch OPX platform adaptation service (PAS), and invoke a set request to change the speed of the fan object. Neither the TC application nor the OpenSwitch OPX PAS are part of the CPS infrastructure—they function as user applications of the CPS infrastructure.

- Entities handled by the CPS infrastructure are called CPS objects (referred to as *objects*)—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—a set operation. The CPS service, which registers for the ownership of the specific object category, performs the operation.
- The CPS API provides database-like semantics—services that implement CPS operations (object owners) do not persistently store attribute values.

The PAS registers for ownership of both the temperature sensor and fan object types. The PAS application periodically reads the value of a temperature sensor. When the temperature is greater than a predefined threshold, the PAS creates a CPS overtemperature 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 determines the applications that have subscribed for the event and object type, and provides the applications (TC application) with the contents of the event.

The publisher of the event (PAS) does not have any knowledge of the applications that have subscribed for an event. 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 increases the fan speed—it creates a simple CPS transaction in which a set operation requests a change in the speed attribute value of the fan object. CPS API functions perform the transaction. When the transaction is committed, the CPS infrastructure finds the owner of the fan object category—the application that has registered to execute operations requested for fan objects (PAS).

The CPS infrastructure invokes the function registered by PAS to execute the set operation for fans. PAS simply invokes the set operation for low-level fan speed provided by the system device interface (SDI), which acts on the fan hardware and changes its speed.

CPS keys

All CPS objects require a key — the concept of a CPS key is central to the CPS infrastructure. An application uses a key to register for object events, perform get requests, and perform transactions. A CPS key has two components—a fixed key component represented as a sequence (array) of numeric values, and an optional (dynamic) key component.

The fixed part of a CPS key consists of:

- Qualifier — Target, observed, real-time
- Attribute identifiers that describe the object name and hierarchy — Object category and subcategory. An object category refers to a class of objects, such as interfaces, and object subcategories are defined relative to categories—LAG interface and VLAN interface which are subcategories of interfaces.

Qualifier	C enum Symbol	String Form	Applicability	Description
Target	`cps_api_qualifier_TARGET`	target	Configurable attributes	Current running-configuration
Observed	`cps_api_qualifier_OBSERVED`	observed	Configurable attributes and hardware status attributes	Configuration applied to hardware components or current status (provided by monitoring service — can be cached)

Qualifier	C enum Symbol	String Form	Applicability	Description
Real-Time	`cps_api_qualifier_RE ALTIME`	realtime	Hardware status attributes and hardware counters	Requests values immediately queried from hardware (no caching)
Registrations	`cps_api_qualifier_RE GISTRATION`	objreg	Internal to CPS infrastructure	Qualifier used when publishing events associated to object registrations
Proposed	`cps_api_qualifier_PR OPOSED`	proposed	Not applicable	Reserved

Represent the fixed component of a key in either binary (array of bytes) or string form. Express a string form either in numerical format (`1.34.2228241.2228246.2228236`) or using a name alias (`target/base-ip/ipv4/vrf-id/ifindex`).

Optional component

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

Build CPS key

Use the `cps_api_key_from_attr_with_qual` function to build a CPS key. Create a key by specifying the object name and CPS qualifier.

```
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 of the key, add attributes using the standard object attribute function.

Objects and attributes

CPS objects and object lists are used in the infrastructure. An object consists of a key which uniquely identifies the object, and zero or more object attributes.

A CPS object:

- Contains a variable number of attributes
- Can embed other objects
- Can be easily serialized—written to, read from, or a persistent or temporary storage device
- Supports attribute types:
  - `uint8_t`
  - `uint16_t`
  - `uint32_t`
  - `uint64_t`
  - `char []`
  - Attributes that contain other attributes

Object attributes

CPS object attributes are identified by a 64-bit ID tag (attribute identifier) and represented using a type, length, value (TLV) format. The attribute 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 to include the zero terminating octet.

ⓘ **NOTE:** Although the Python implementation automatically adds a zero octet to all string values, the C/C++ implementation does not. You must take into account the zero-terminating octet when you use a C/C++ application to set the length of an attribute.

CPS qualifiers

A CPS qualifier provides the type of object data to retrieve or act on. In the temperature control example (see Client and server applications), 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.

Target qualifier	Indicates that the server application (object owner) returns the values of attributes previously specified in a set operation.
Observed qualifier	Indicates that the server application (object owner) returns the values of the attributes already successfully applied to hardware entities (for user-configurable attributes), or the last retrieved hardware status information values.
Real-time qualifier	Indicates that the server application (object owner) reads the current status of the hardware entity to return the value of a requested attribute.

Observed vs. 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). 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. 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 vs. 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, the set operation to change the fan speed uses the *target qualifier*. Because it takes a few seconds for the 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.

Publish/subscribe

The CPS infrastructure provides a subscription mechanism for receiving and publishing object events. An application registers 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 that has a prefix match with the key associated to the published object.

If you publish a target IPv4 address object that has `base-ip/ipv4/address`:

- The object key is `{target, base-ip, ipv4, address}`
- The CPS qualifier (`target`) is a mandatory part of the key
- Any application that subscribes for objects that match any of the keys receives the event:
  - Key 1 `{Target, base-ip}` — Receives all events for objects under `base-ip`

- Key 2 `{Target, base-ip,ipv4}` — Receives all events for objects under IPv4
- Key 3 `{Target, base-ip,ipv4,address}` — Receives all IPv4 address objects
- Multiple applications can subscribe for events that match a specified key

The infrastructure generates events for CPS-specific conditions when new object owners (server applications) register with the CPS, or objects de-register from the CPS. The object key contains the registration qualifier `cps_api_qualifier_REGISTRATION` in which the registration or de-registration event refers to. The key also indicates if the event represents a new object registration or de-registration.

API operations

The CPS infrastructure supports database-like requests for creating, updating, deleting, and retrieving one or more objects in a single operation. The CPS API defines the get request 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 that specifies the keys of objects to retrieve. 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 if there is a failure.

Transactions

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

Although it is not necessary for 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 100,000 routes in a transaction, the result of the commit request is:

- All 100,000 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. 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 and object attribute handling
- CPS event handling and CPS operations

Object model representation

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

API language support

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

YANG modeling objects

YANG data models are used to define the content of CPS objects to configure and retrieve information from the software. A YANG model consists of types (typedef, groupings, and enums), containers (container, list, choice, and case), and properties (leaf and leaf-list).

Each property in the YANG container is a CPS object attribute. List containers nested in a YANG model are 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:

- 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`, and so on) found in the model
- A Python C extension library containing CPS object metadata and YANG name-to-CPS ID mapping information.

Include the generated C header to use the CPS object name and attribute identifiers to create or delete CPS objects and set values for each attribute in a C/C++ application.

Use the YANG module, and object and attribute names directly in a Python application. The CPS Python engine automatically uses the extension library to derive the corresponding identifiers for these names.

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 is 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 if an object is related to a set, delete, create or action transaction—used when events are received.

CPS API objects

Common commands used to manage CPS API objects are included.

get object

Retrieve and view the contents of a CPS API object.

```
cps_get_oid.py [-h] [-mod module] [-d]
                       [-qua {target,observed,proposed,realtime,registration,running,startup}]
                       [-attr ATTR] [-db]
                       module [additional [additional ...]]
```

- *module* — object's name and optional qualifier; for example, cps/node-group of if/interfaces/interface (a qualifier can optionally be placed at the beginning)
- *additional* — field can contain a series of object attributes in the form of attr=value combinations
- *-h, --help* — (Optional) displays this help message and exit
- *-mod module* — (Optional) alternate way to specify the module name
- *-d* — (Optional) print some additional details about the objects parsed and sent to the backend
- *-qua {target,observed,proposed,realtime,registration,running,startup}* — (Optional) object's qualifier
- *-attr ATTR* — (Optional) object attributes in the form of attr=value
- *-db* — (Optional) attempt to use the database directly to satisfy the request instead of the normally registered object

Retrieve entity object for slot 1 PSU

```
cps_get_oid.py observed/base-pas/entity entity-type=1 slot=1

===========base-pas/entity==========

base-pas/entity/platform-name =
base-pas/entity/hw-version = A00
base-pas/entity/present = 1
base-pas/entity/service-code = 226 457 410 55
base-pas/entity/insertion-timestamp = 1522692340
base-pas/entity/product-name = CN0T9FNW2829845L0006
base-pas/entity/oper-status = 1
base-pas/entity/vendor-name =
base-pas/entity/slot = 1
base-pas/entity/entity-type = 1
base-pas/entity/name = PSU Tray-1
base-pas/entity/service-tag = AEIOU##
base-pas/entity/part-number = 0T9FNW
base-pas/entity/slot = 1
base-pas/entity/fault-type = 1
base-pas/entity/entity-type = 1
base-pas/entity/ppid = CN0T9FNW2829845L0006
base-pas/entity/insertion-cnt = 1
base-pas/entity/admin-status = 1
------------------------------------------------
```

Alternate retrieve entity object for slot 1 PSU

```
cps_get_oid.py -qua observed base-pas/entity entity-type=1 slot=1
```

set object

Perform a CPS commit operation taking the object specified on the command line.

```
cps_set_oid.py [-h] [-mod module] [-d]
                       [-qua {target,observed,proposed,realtime,registration,running,startup}]
                       [-attr ATTR] [-db] -oper {delete,create,set,action}
                       [-commit-event]
                       module [additional [additional ...]]
```

- *module* — object's name and optional qualifier; for example, cps/node-group or if/interfaces/interface (a qualifier can optionally be placed at the beginning)
- *additional* — field can contain a series of object attributes in the form of attr=value combinations
- *-h, --help* — (Optional) displays this help message and exit
- *-mod module* — (Optional) alternate way to specify the module name
- *-d* — (Optional) print some additional details about the objects parsed and sent to the backend

- *-qua {target,observed,proposed,realtime,registration,running,startup}* — (Optional) object's qualifier
- *-attr ATTR* — (Optional) object attributes in the form of attr=value
- *-db* — (Optional) attempt to use the database directly to satisfy the request instead of the normally registered object
- *-oper {delete,create,set,action}* — (Optional) operations types; only used in CPS commit operations
- *-commit-event* — (Optional) flag will try to force the default state of the auto-commit event to true; only used in CPS commit operations

Turn on beacon LED

```
cps_set_oid.py -oper create base-pas/led entity-type=3 slot=1 name=Beacon on=1
Success
Key: 1.28.1835214.1835062.1835063.1835064.
base-pas/led/slot = 1
base-pas/led/on = 1
cps/object-group/return-code = 0
base-pas/led/entity-type = 3
base-pas/led/name = Beacon
```

Alternate turn on beacon LED

```
cps_set_oid.py -oper create target/base-pas/led entity-type=3 slot=1 name=Beacon on=1
```

Object dictionary support

The CPS object dictionary APIs access the metadata for each YANG model. The dictionary contains items for each YANG class or attribute:

- Static key of the element including the elements hierarchy
- Type of element
- Information associated with the element
- Model type—attribute, attribute-list, or object
- Unique attribute identifier

Access the CPS object dictionary in both C/C++ and Python (see CPS API reference).

YANG model C header

This example shows the C header file generated from the sFlow YANG model by the CPS YANG parser. The sFlow YANG model has two top-level containers:

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

See dell-base-sflow.yang for complete information.

The CPS YANG parser generates a C header for this model—the C header generation happens during the build process. The `opx-base-model` and `opx-cps` repos must be present in your workspace.

The header includes the C definitions for the YANG entities:

- Category for the YANG model is `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:
  - `BASE_SFLOW_ENTRY_IFINDEX`

– BASE_SFLOW_ENTRY_DIRECTION

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


/*
* Copyright (c) 2016 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
```

sFlow using YANG and Python

This Python example shows how to use a YANG model to configure a new sFlow entry. The Python utility `cps_utils` is used to create CPS objects and CPS transactions. See Application templates for information about how to write an application using the CPS API.

```
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 a GET API on a YANG container. The result displays the values configured in the software for all 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

OpenSwitch OPX provides YANG models to configure networking- and platform-related functions. These YANG models are defined by the network adaptation and platform adaptation services.

- `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-stp.yang` — spanning-tree protocols
- `dell-base-switch-element.yang` — global configuration parameters for NPU
- `dell-base-vlan.yang` — VLAN
- `dell-base-env-tempctl.yang` — temperature control (TC)
- `dell-base-pas.yang` — platform adaptation service (PAS)
- `dell-base-platform-common.yang` — common platform definitions

CPS API reference

To access the Python and C/C++ CPS API reference guides, see the README file in included with the software image. The YANG model files and C header files derived from the YANG models are included in the development packages and downloaded with the software image.

Application examples

This information contains links which describe how to write Python and C/C++ applications using the CPS API application examples. These links contain complete instructions, as well as links to directly view example application files.

VLAN	Python application examples
IP address	Python application examples
Route	Python application examples
ACL	Python application examples
MAC address table	Python and C/C++ application examples
Event	Python and C/C++ application examples

Topics:

VLAN application examples

(i) NOTE: VLAN refers to an NPU VLAN object which is modeled as a Linux bridge.

`cps_config_vlan.py` contains examples for adding and deleting VLANs, adding and deleting ports to/from VLANs, and so on. See cps_config_vlan.py for complete information.

Verify VLAN creation using CPS get

```
# cps_get_oid.py dell-base-if-cmn/if/interfaces/interface if/interfaces/interface/
type=ianaift:l2vlan

Key: 1.19.44.2883617.2883612.2883613.
dell-base-if-cmn/if/interfaces/interface/if-index = 57
dell-if/if/interfaces/interface/phys-address = 90:b1:1c:f4:b3:e3
base-if-vlan/if/interfaces/interface/id = 100
dell-if/if/interfaces/interface/untagged-ports = e101-001-0
if/interfaces/interface/type = ianaift:l2vlan
if/interfaces/interface/name = br100
dell-if/if/interfaces/interface/vlan-type = 1
dell-if/if/interfaces/interface/learning-mode = 1
dell-if/if/interfaces/interface/mtu = 1532
if/interfaces/interface/enabled = 0
```

> (i) **NOTE: OpenSwitch OPX allocates an `ifindex` for each VLAN created and further CPS set and get operations can use the `ifindex` as the key.**

Verify VLAN creation using Linux

```
$ brctl show
bridge name     bridge id               STP enabled     interfaces
br100           8000.000000000000       no
```

Verify VLAN port addition using CPS get

```
# cps_get_oid.py dell-base-if-cmn/if/interfaces/interface if/interfaces/interface/
type=ianaift:l2vlan

Key: 1.19.44.2883617.2883612.2883613.
dell-base-if-cmn/if/interfaces/interface/if-index = 43
dell-if/if/interfaces/interface/phys-address =
dell-if/if/interfaces/interface/untagged-ports = e101-003-0,e101-002-0,e101-001-0
base-if-vlan/if/interfaces/interface/id = 100
if/interfaces/interface/name = br100
dell-if/if/interfaces/interface/learning-mode = 1
if/interfaces/interface/enabled = 0

Key: 1.19.44.2883617.2883612.2883613.
dell-base-if-cmn/if/interfaces/interface/if-index = 3
dell-if/if/interfaces/interface/phys-address = 90:b1:1c:f4:ab:ed
base-if-vlan/if/interfaces/interface/id = 0
if/interfaces/interface/name = docker0
dell-if/if/interfaces/interface/learning-mode = 1
if/interfaces/interface/enabled = 0
```

Verify VLAN port addition using Linux

```
$ brctl show
bridge name     bridge id           STP     enabled interfaces
br100           8000.90b11cf4abee   no      e101-001-0
                                            e101-002-0
                                            e101-003-0
```

Verify VLAN port deletion using CPS get

```
# cps_get_oid.py dell-base-if-cmn/if/interfaces/interface if/interfaces/interface/
type=ianaift:l2vlan

Key: 1.19.44.2883617.2883612.2883613.
dell-base-if-cmn/if/interfaces/interface/if-index = 47
dell-if/if/interfaces/interface/phys-address =
dell-if/if/interfaces/interface/untagged-ports = e101-002-0
base-if-vlan/if/interfaces/interface/id = 100
if/interfaces/interface/name = br100
dell-if/if/interfaces/interface/learning-mode = 1
if/interfaces/interface/enabled = 0

Key: 1.19.44.2883617.2883612.2883613.
dell-base-if-cmn/if/interfaces/interface/if-index = 3
dell-if/if/interfaces/interface/phys-address = 90:b1:1c:f4:ab:ed
base-if-vlan/if/interfaces/interface/id = 0
if/interfaces/interface/name = docker0
dell-if/if/interfaces/interface/learning-mode = 1
if/interfaces/interface/enabled = 0
```

Verify VLAN port deletion using Linux

```
$ brctl show
bridge name bridge id STP enabled interfaces
br100 8000.000000000000 no          e101-002-0
```

Verify VLAN deletion using CPS get

```
# cps_get_oid.py dell-base-if-cmn/if/interfaces/interface if/interfaces/interface/
type=ianaift:l2vlan

Key: 1.19.44.2883617.2883612.2883613.
dell-base-if-cmn/if/interfaces/interface/if-index = 3
dell-if/if/interfaces/interface/phys-address = 90:b1:1c:f4:ab:ed
base-if-vlan/if/interfaces/interface/id = 0
if/interfaces/interface/name = docker0
dell-if/if/interfaces/interface/learning-mode = 1
if/interfaces/interface/enabled = 0
```

Verify VLAN deletion using Linux

```
$ brctl show br100
```

IP address application examples

See base_ip.py for complete information.

Verify IP address 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 using Linux

```
$ 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
```

Verify IP address Deletion using get (return indicates that e101-001-0 has no IP address)

```
$ cps_get_oid.py 'base-ip/ipv4/address'
```

Verify IP address deletion

```
$ 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
```

Route application examples

(i) NOTE: See the `dell-base-route.yang` model which defines the route object and attributes before you configure route settings.

See cps_config_route.py for complete information.

Verify route creation

```
$ 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
```

Verify route deletion

```
$ 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

$ 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.'}

$ ip route
1.1.1.0/24 dev e101-001-0 proto kernel scope link src 1.1.1.1
```

ACL application examples

ⓘ | **NOTE:** See the `dell-base-acl.yang` model which defines an ACL object and attributes before you configure ACL settings.

Create ACL table using Python

An ACL table groups 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.

ⓘ | **NOTE:** See the `dell-base-acl.yang` model which defines an ACL object and attributes before you configure ACL settings.

1   Import the CPS utility Python library.
```
import cps_utils
import nas_os_utils
```
2   Define the enum map.

NOTE: 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}
```
3   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)
```
4   Create the CPS object and populate the attributes.
```
cps_obj = cps_utils.CPSObject(module='base-acl/table')
```
5   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']])
```
6   Define the CPS object.
```
cps_update = ('create', cps_obj.get())
```
7   Define an add operation and object pair to the CPS transaction.

NOTE: Each CPS transaction can hold multiple CPS operation and object pairs.

```
cps_trans = cps_utils.CPSTransaction([cps_update])
```
8   Verify the return value.
```
ret = cps_trans.commit()
if not ret:
    raise RuntimeError ("Error creating ACL Table")
```

9   Retrieve the CPS object ID from the ACL table — this ID is used for all operations on the ACL table.

```
ret = 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)
```

create-acl.py

```python
#Python code block to create ACL /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])

#Verify return value
ret = cps_trans.commit()
if not ret:
    raise RuntimeError ("Error creating ACL Table")
ret = cps_utils.CPSObject (module='base-acl/table', obj=r[0]['change'])

#Retrieve CPS object ID
cps_get_val = 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 ACL table creation using CPS get

```
# 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
```

Create ACL entry using Python

An ACL entry is a rule that consists of a set of filters that define packets to be matched, and a set of actions to be performed on the matched packets.

1   Import the CPS utility Python library.

```
import cps_utils
import nas_os_utils
```

2   Define the enum map.

NOTE: 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}
```

3   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)
```

4   Create the CPS object based on the `dell-base-acl.yang` model, then define the leaf attributes.

```
cps_obj = cps_utils.CPSObject(module='base-acl/table)
```

5   Define the ACL table ID to indicate the group to which this ACL entry belongs to. The priority value determines the sequence of the ACL rule lookup in the ACL table group.

```
cps_obj.add_attr ('table-id', tbl_id) cps_obj.add_attr ('priority', 512)
```

6   Define the filters that the packets are matched to — 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 two attributes — match-type and match-value.

NOTE: Use the correct match-value attribute name depending on the value assigned to the match-type. Use the 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'))
```

7   Define actions to apply on matched packets — 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 two attributes — action-type and action-value.

NOTE: Use the correct action-value attribute name depending on the value assigned to the action-type. 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'])
```

8   Associate the CPS object an operation.

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

9   Add the CPS operation and object pair to a new transaction. Each CPS transaction holds multiple pairs of CPS operation and object updates.

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

10   Verify the return value.

```
ret = cps_trans.commit()
if not ret:
    raise RuntimeError ("Error creating MAC ACL Entry")
```

11   Retrieve the CPS object ID from the ACL table. This ID is used for all operations on the ACL table.

```
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)
```

create-acl-table.py

```
#Python code block to create ACL table entry /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}

#Inform 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])

#Verify return value
ret = cps_trans.commit()
if not ret:
    raise  RuntimeError ("Error creating ACL Table")

#Retrieve CPS Object ID
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 table 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 and verify return
ret = cps_trans.commit()
if not ret:
    raise RuntimeError ("Error creating MAC ACL Entry")

#Return CPS Object ID
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 ACL entry creation using CPS get

```
# 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 NPU ACL entry creation

```
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=0x00000000000000000000000000000000000000000000000002000000000000
    MASK=0x00000000000000000000000000000000000001ffffffffffffffffffffffffffff
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
```

Delete ACL entry using Python

1   Import the CPS utility Python library.

```
import cps_utils
```

2   Define the enum map.

NOTE: 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}
```

3   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)
```

4   Define the table and entry ID key values and create the CPS object.

```
cps_obj = cps_utils.CPSObject(module='base-acl/entry', data={'table-id': 'id': mac_eid})
```

5   Associate the object with a CPS operation.

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

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

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

7   Verify the return value.

```
ret = cps_trans.commit()
if not ret:
    raise RuntimeError ("Error deleting ACL Entry")
```

delete-acl.py

```
#Python code block to delete ACL entry
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])

#Verify return value
ret = cps_trans.commit()
if not ret:
    raise RuntimeError ("Error deleting ACL Entry")
```

MAC address table application examples

This information includes both Python and C/C++ application examples.

> (i) **NOTE: See the `dell-base-l2-mac.yang` model that defines the MAC object and attributes before creating a MAC address table entry.**

See cps_config_mac.py for examples of creating and deleting MAC entries, as well as manipulating MAC tables for VLANs. See nas_mac_unittest.cpp for examples of MAC operations in C++.

Event application examples

- See opx-alm-service for an example of a Python application that registers for events
- See cps_api_events_unittest.cpp for an example of a C++ application that registers for events
- See cps_send_event.py for an example of a publishing events in Python
- See cps_api_events_unittest.cpp for an example of a publishing events in C++

Application templates

Provided are templates to develop applications in Python and C/C++ for server and client applications, and event publishers and event subscribers. Client applications subscribe to events—server applications publish events, and applications can act as both clients and servers.

Event flow between CPS client and server applications
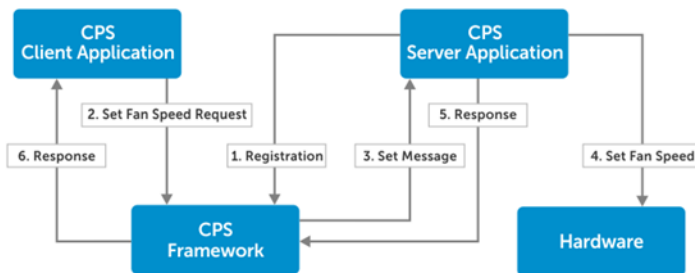
Event flow between a CPS event subscriber and publisher
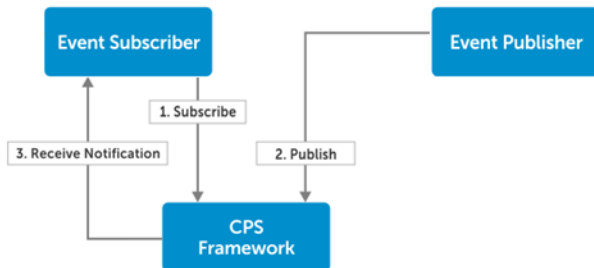
Python templates

Server application	Template for the structure of a server (object owner), including the transaction callback handler for set, create, delete, action operations and a separate handler for get operations.
Client application	Template for the structure of a client application including the execution of a get request.
Event publisher application	Template for the structure of an event publisher.
Event subscriber application	Template for the structure of an event subscriber. The application registers for events, and waits for events in a loop.

C/C++ templates

Server application	Template for the structure of a CPS server (object owner) including read and write functions for get operations (`xyz_read`)—set, create, delete, action operations (`xyz_write`), and rollback of failed transactions (`xyz_rollback`).
Client application	Template for the structure of a CPS client including read and write functions for get requests (building keys and object lists), and set requests (using a transaction).

Event publisher application	Template for the structure of a CPS event publisher including initialization of the event service, connection to the object library, and the event publish operation.
Event subscriber application	Template for the structure of a CPS event subscriber including initialization of the event service and event processing thread, and registration of the event handler function, and event processing callback. The key list specified in the registration is used to determine the events delivered to the application—list contains a single element.

Topics:

CPS server application templates

This information includes templates for the structure of a CPS server (object owner), including the transaction callback handler for set, create, and delete action operations, and a separate handler for get operations.

python-server-template.py

```
#Python code block for CPS server application
import   time import   cps
import   cps_utils

#Define the get callback handler function
def   get_callback(methods,   params):

#Append 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 attributes from 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 handle to CPS API service
    handle = cps.obj_init()

#Register above handlers to be run when a request is received for given key
cps.obj_register(handle, key,
                { get :  get_callback,
                  transaction :  transaction_callback
                }
              )

#Let the handlers run
while  True:
    time.sleep(1000)
```

c-template-server-application.c

```c
/*******************************************************************
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 */
        break;
    }

    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);
}
```

CPS client application templates

This information includes templates for the structure of a CPS client application, including the execution of a get request.

python-client-application-template.py

```
#Python code block for CPS client application
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-client.template.c

```
/****************************************************************
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 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 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);
}
```

CPS event publisher application templates

This information includes templates for the structure of a CPS event publisher.

python-event-publisher-application.py

```python
#Python code block for 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-publisher-application.c

```c
/****************************************************************
Template for event publish function
****************************************************************/

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);
}
```

CPS event subscriber application templates

This information includes templates for the structure of a CPS event subscriber. 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).

python-event-subscriber-application.py

```python
#Python block code for CPS event subscriber application
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'] == ...:
        ...
```

c-template-event-subscriber-application.c

```c
/********************************************************************
Template for event subscriber application
********************************************************************/

  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 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);
}
```