PFE FCI API Reference

Rev. 2.4.1 — 18 September 2023 User manual

Revision History

Revision history

Revision	Change Description
1.0.0	Initial version. Contains description of FCI API and following features: • Interface Management • IPv4/IPv6 Router (TCP/UDP) • L2 Bridge (Switch) • Flexible Parser • Flexible Router
1.1.0	Added description of simple bridge (without VLAN awareness). Disabled part describing async messaging as it is currently not used. Description of fci_cmd(), fci_query(), and fci_write() simplified. Various minor improvements.
1.2.0	Improved description of Router and Bridge configuration steps. Added missing byte order information to various command argument values. Following values unified with rest of structure members to be in network byte order: • `fpp_rt_cmd_t::id` • `fpp_rt_cmd_t::flags` • `fpp_ct_cmd_t::route_id` • `fpp_ct_cmd_t::route_id_reply` • `fpp_ct_cmd_t::flags` • `fpp_fp_table_cmd_t::position`
1.2.1	Added FPP_IF_MIRROR to fpp_if_flags_t. Added name of interface to mirror the traffic to fpp_phy_if_cmd_t.
1.3.0	Description of various elements re-phrased to better explain their purpose. Created summary lists of functions, commands, and events and added links to them to improve document navigation. Added usage examples for FPP_CMD_PHY_IF, FPP_CMD_LOG_IF, and FPP_CMD_IP_ROUTE commands. Described relevant fpp_rt_cmd_t structure members.
1.4.0	Added usage examples for FPP_CMD_IPV4_CONNTRACK and FPP_CMD_IPV6_CONNTRACK. Related argument structures documentation updated. Removed unwanted and unsupported symbol descriptions.
1.5.0	Added statistics for physical `fpp_phy_if_stats_t` and logical `fpp_algo_stats_t` interfaces. Statistics are in network byte order.
1.6.0	Added API for data passing: `FPP_CMD_DATA_BUF_PUT` and `FPP_CMD_DATA_BUF_AVAIL` with related `fpp_buf_cmd_t`.
1.7.0	Described the IPsec offload configuration and related `FPP_CMD_SPD` command.

Revision history *...continued*

Revision	Change Description
1.8.0	Added QoS configuration commands: FPP_CMD_QOS_QUEUE, FPP_CMD_QOS_SCHEDULER and FPP_CMD_QOS_SHAPER with related argument structures.
1.8.1	Licensing notice within headers of examples updated.
1.9.0	Added L2L3 Bridge, Feature management and Static Entries (L2 Bridge) API. Synced with recent changes. Various improvements.
1.9.1	Copyright notice and document classification updated. Removed the "Index" chapters.
1.9.2	The fpp_rt_cmd_t updated to include src_mac member description. Various minor improvements.
1.10.0	New demo examples. Also, document thoroughly checked and modified.
1.11.0	Added API for management of physical interface MAC addresses. Added API for SPAN mirroring. Minor corrections in FPP_CMD_L2_BD and FPP_CMD_FP_RULE chapters. Improved formatting of struct chapters.
1.11.1	Updated fpp_if_flags_t flags.
1.12.0	Added API for Ingress QoS. Added chapter about limitations. Fixed minor issue in physical interface demo codes.
1.13.0	Removed simple (non-VLAN aware) L2 Bridge. Use VLAN-aware L2 Bridge instead.
1.14.0	Updated description of Egress QoS feature with information about queue slot pools. Updated Egress QoS demo code.
1.15.0	Updated FPP_CMD_IPV4_CONNTRACK and FPP_CMD_IPV6_CONNTRACK with info about conntrack statistics.
1.16.0	Added FCI ownership description and FCI negotiation commands FPP_CMD_FCI_OWNERSHIP_LOCK, FPP_CMD_FCI_OWNERSHIP_UNLOCK and their usage examples. Updated fpp_phy_if_cmd_t with new data member (name of PTP management interface).
1.16.1	Added limitations of Egress QoS Scheduler configuration.
2.0.0	Document template changed. Modified layout and texts in Egress QoS feature chapter. Factual content is unchanged, but the chapter should be more comprehensible now.
2.1.0	Added Health Monitor FCI API event (FPP_CMD_HEALTH_MONITOR_EVENT). Added API for management of FW feature elements (FPP_CMD_FW_FEATURE_ELEMENT). Added chapter about FW features and FW feature elements. Added missing error code in description of FPP_CMD_FW_FEATURE. Fixed a few minor errors in the document.
2.2.0	Added routing table FCI API events (FPP_CMD_IPV4_CONNTRACK_CHANGE and FPP_CMD_IPV6_CONNTRACK_CHANGE).
2.3.0	Added gPTP timer owner FCI API(FPP_CMD_TIMER_LOCK and FPP_CMD_TIMER_UNLOCK).
2.3.1	Added Physical Interface Configuration Flag (FPP_IF_FF_ALL_TCP).
2.3.2	Added information about mutual exclusivity of match rules FPP_IF_MATCH_SIP6 / FPP_IF_MATCH_DIP6 / FPP_IF_MATCH_SIP / FPP_IF_MATCH_DIP. See fpp_if_m_rules_t.
2.3.3	Minor formatting updates in chapter FPP_ERR.

Revision history...*continued*

Revision	Change Description
2.3.4	Added note about FCI command usage when PFE interface database is locked.
2.4.0	Added chapter about FCI events. Clarified description of `fci_catch()`, `fci_register_cb()` and `fci_cb_retval_t`. Added description of `fci_catch()` MCAL deviation.
2.4.1	Updated/clarified content of IP Router chapter. Added structured information about 5-tuple/3-tuple conntrack matching modes. Clarified that 3-tuple matching mode is not available for TCP/UDP conntracks.

1 Introduction

This is Fast Control Interface available for host applications to communicate with the networking engine.

The FCI is intended to provide a generic configuration and monitoring interface for the networking acceleration HW. Provided API shall remain the same within all HW/ OS-specific implementations to keep dependent applications portable across various systems.

The LibFCI is not directly touching the HW. Instead, it only passes commands to a dedicated software component (OS/HW-specific endpoint) and receives return values. The endpoint is then responsible for HW configuration. This approach supports a kernel-user space deployment where the user space contains only API and the logic is implemented in kernel.

Implementation uses appropriate transport mechanism to pass data between LibFCI user and the endpoint. For reference: in Linux a netlink socket is used; in QNX a message is used.

2 How to use the FCI API

2.1 Sending FCI commands

1. Call `fci_open()` to get an `FCI_CLIENT` instance, using `FCI_GROUP_NONE` as a multicast group mask. This opens a connection to an FCI endpoint.
2. Call `fci_write()` or `fci_query()` to send a command to the endpoint. See Commands Summary.
   - Endpoint receives the command and executes requested actions.
   - Endpoint generates a response and sends it back to the client.
3. [optional] Repeat the previous step to send all requested FCI commands.
4. Call `fci_close()` to finalize the `FCI_CLIENT` instance.

2.2 Capturing FCI events

FCI endpoint can send asynchronous messages (FCI events) to FCI clients. This can be utilized to receive immediate notifications when certain events occur at the endpoint. For list of supported events, see Events Summary.

How to configure FCI client to capture FCI events:

1. Call `fci_open()` to get `FCI_CLIENT` instance, using `FCI_GROUP_CATCH` as a multicast group mask. This opens a connection to FCI endpoint and it creates FCI client which can send FCI commands and also capture FCI events.
2. Call `fci_register_cb()` to register a user-defined callback function for processing of FCI events.
   Note: *Without this step, `fci_catch()` will not capture any FCI events.*
3. Call `fci_catch()` to start capturing FCI events. See Events Summary for list of capturable events.
4. When `fci_catch()` captures an FCI event, it calls the user-defined callback function to process the event.

Note: The user-defined callback function must return `FCI_CB_CONTINUE`, otherwise `fci_catch()` terminates.

5. [optional] To stop capturing FCI events, call `fci_register_cb()` with NULL parameter.
6. To finish FCI session and close the FCI client, call `fci_close()`.

2.3 FCI ownership in Master-Slave setup

The FCI ownership applies only to PFE driver Master-Slave setup. This feature ensures that at any given time there is only one driver instance which can successfully issue FCI commands. Configuration of FCI Ownership is stored in the Master instance. The configuration specifies which driver instances are allowed to acquire FCI ownership.

There can be only one FCI owner at a time. Only the FCI commands issued by the FCI owner can be executed successfully. FCI commands issued by a driver instance which does not have FCI ownership are never executed and return an error code instead. Driver instance can acquire or release the FCI ownership via the following means:

- **Manually**, by requesting the FCI ownership via dedicated FCI commands `FPP_CMD_FCI_OWNERSHIP_LOCK` and `FPP_CMD_FCI_OWNERSHIP_UNLOCK`. If the ownership is acquired manually, it has to be manually released as well (responsibility of the requesting driver).
- **Automatically**, by acquiring floating FCI ownership if it is not held at the moment by other client. The floating FCI ownership is granted temporarily for each issued FCI command. Once the execution of the respective FCI command is finished, the FCI ownership is automatically released, so other sender can take FCI ownership.

3 Acronyms and Definitions

- **PFE:**
  Packet Forwarding Engine. A dedicated HW component (networking accelerator) which is configured by this FCI API.
- **NBO:**
  Network Byte Order. When working with values or properties which are stored in [NBO], consider using appropriate endianess conversion functions.
- **L2/L3/L4:**
  Layers of the OSI model.
- **Physical Interface:**
  See Physical Interface.
- **Logical Interface:**
  See Logical Interface.
- **Classification Algorithm:**
  Method how ingress traffic is processed by the PFE firmware.
- **Route:**
  In the context of PFE, a route represents a way to reach an external network node. It is utilized for IP Router packet forwarding. It holds the following information:
  – Egress physical interface (PFE interface to reach the external network node).
  – MAC address of the external network node.
- **Conntrack:**
  "Tracked connection", a data structure with information about a connection. In the context of PFE, it always refers to an IP connection (TCP, UDP, other). The term is

equal to a 'routing table entry'. Each conntrack is linked with some **route**. The route is used for forwarding of traffic that matches properties of a conntrack.

- **RSPAN:**
  Remote Switch port Analyzer. A way of monitoring traffic via traffic mirroring between ports. In the context of PFE, this refers to traffic mirroring between physical interfaces. See chapter Physical Interface and its subchapter Mirroring rules management.

4 Commands Summary

- `FPP_CMD_PHY_IF`
  Management of physical interfaces.
- `FPP_CMD_LOG_IF`
  Management of logical interfaces.
- `FPP_CMD_IF_LOCK_SESSION`
  Get exclusive access to interface database.
- `FPP_CMD_IF_UNLOCK_SESSION`
  Cancel exclusive access to interface database.
- `FPP_CMD_IF_MAC`
  Management of interface MAC addresses.
- `FPP_CMD_MIRROR`
  Management of interface mirroring rules.
- `FPP_CMD_L2_BD`
  Management of L2 bridge domains.
- `FPP_CMD_L2_STATIC_ENT`
  Management of L2 static entries.
- `FPP_CMD_L2_FLUSH_LEARNED`
  Remove all dynamically learned MAC table entries.
- `FPP_CMD_L2_FLUSH_STATIC`
  Remove all static MAC table entries.
- `FPP_CMD_L2_FLUSH_ALL`
  Remove all MAC table entries.
- `FPP_CMD_FP_TABLE`
  Management of Flexible Parser tables.
- `FPP_CMD_FP_RULE`
  Management of Flexible Parser rules.
- `FPP_CMD_IPV4_RESET`
  Remove all IPv4 routes and conntracks.
- `FPP_CMD_IPV6_RESET`
  Remove all IPv6 routes and conntracks.
- `FPP_CMD_IP_ROUTE`
  Management of IP routes.
- `FPP_CMD_IPV4_CONNTRACK`
  Management of IPv4 conntracks.
- `FPP_CMD_IPV6_CONNTRACK`
  Management of IPv6 conntracks.
- `FPP_CMD_IPV4_SET_TIMEOUT`
  Configuration of conntrack timeouts.
- `FPP_CMD_DATA_BUF_PUT`
  Send arbitrary data to the accelerator.
- `FPP_CMD_SPD`

Management of the IPsec offload.
- FPP_CMD_QOS_QUEUE
  Management of Egress QoS queues.
- FPP_CMD_QOS_SCHEDULER
  Management of Egress QoS schedulers.
- FPP_CMD_QOS_SHAPER
  Management of Egress QoS shapers.
- FPP_CMD_QOS_POLICER
  Ingress QoS policer enable/disable.
- FPP_CMD_QOS_POLICER_FLOW
  Management of Ingress QoS packet flows.
- FPP_CMD_QOS_POLICER_WRED
  Management of Ingress QoS WRED queues.
- FPP_CMD_QOS_POLICER_SHP
  Management of Ingress QoS shapers.
- FPP_CMD_FW_FEATURE
  Management of FW features.
- FPP_CMD_FW_FEATURE_ELEMENT
  Management of FW feature elements.
- FPP_CMD_FCI_OWNERSHIP_LOCK
  Management of FCI ownership in Master-Slave setup.
- FPP_CMD_FCI_OWNERSHIP_UNLOCK
  Management of FCI ownership in Master-Slave setup.
- FPP_CMD_TIMER_LOCK
  Management of IEEE 1588 timer ownership.
- FPP_CMD_TIMER_UNLOCK
  Management of IEEE 1588 timer ownership.

5 Events Summary

- FPP_CMD_DATA_BUF_AVAIL
  Driver sends custom data from accelerator to host.
- FPP_CMD_ENDPOINT_SHUTDOWN
  Driver reports that FCI endpoint is shutting down.
- FPP_CMD_HEALTH_MONITOR_EVENT
  Driver reports some Health Monitor event.
- FPP_CMD_HEALTH_MONITOR_EVENT
  Driver reports some Health Monitor event.
- FPP_CMD_IPV4_CONNTRACK_CHANGE
  Driver reports status change of some IPv4 conntrack.
- FPP_CMD_IPV6_CONNTRACK_CHANGE
  Driver reports status change of some IPv6 conntrack.

6 Functions Summary

- fci_open()
  Connect to endpoint and create a client instance.
- fci_close()
  Close a connection to endpoint and destroy the client instance.

- `fci_write()`
  Execute FCI command without data response.
- `fci_cmd()`
  Execute FCI command with data response.
- `fci_query()`
  Alternative to `fci_cmd()`.
- `fci_catch()`
  Poll for and process received asynchronous messages.
- `fci_register_cb()`
  Register a callback to be called in case of a received message.

7 Interface Management

7.1 Physical Interface

Physical interfaces are static objects (defined at startup), which represent hardware interfaces of PFE. They are used by PFE for ingress/egress of network traffic.

Physical interfaces have several configurable properties. See `FPP_CMD_PHY_IF` and `fpp_phy_if_cmd_t`. Among all these properties, a `.mode` property is especially important. Mode of a physical interface specifies which classification algorithm shall be applied on ingress traffic of the interface.

Every physical interface can have a list of logical interfaces. By default, all physical interfaces are in a default mode (`FPP_IF_OP_DEFAULT`). In the default mode, ingress traffic of a given physical interface is processed using only the associated **default** Logical Interface.

FCI operations related to physical interfaces:

- To **list** available physical interfaces:
  1. Lock the interface database.
     (`FPP_CMD_IF_LOCK_SESSION`)
  2. Read out properties of physical interface(s).
     (`FPP_CMD_PHY_IF` + `FPP_ACTION_QUERY` and `FPP_ACTION_QUERY_CONT`)
  3. Unlock the interface database.
     (`FPP_CMD_IF_UNLOCK_SESSION`)
- To **modify** properties of a physical interface (read-modify-write):
  1. Lock the interface database.
     (`FPP_CMD_IF_LOCK_SESSION`)
  2. Read out properties of the target physical interface.
     (`FPP_CMD_PHY_IF` + `FPP_ACTION_QUERY` and `FPP_ACTION_QUERY_CONT`)
  3. Locally modify the properties.
     (see `fpp_phy_if_cmd_t`)
  4. Write the modified properties back to PFE.
     (`FPP_CMD_PHY_IF` + `FPP_ACTION_UPDATE`)
  5. Unlock the interface database.
     (`FPP_CMD_IF_UNLOCK_SESSION`)

Note: *When the interface database is locked via [FPP_CMD_IF_LOCK_SESSION](), only the FCI commands which require the locked database should be used. Failure to adhere to this note may lead to unexpected FCI command failures. FCI commands which require the locked database can be identified in this reference manual by always having extra lock/unlock steps listed in their operation description.*

Table 1. Hardcoded physical interface names and physical interface IDs

name	ID	comment
emac0	0	Representation of real physical ports connected to PFE.
emac1	1	
emac2	2	
---	-	---reserved---
util	5	Special internal port for communication with the util firmware. (fully functional only with the PREMIUM firmware)
hif0	6	Host Interfaces. Used for traffic forwarding between PFE and a host.
hif1	7	
hif2	8	
hif3	9	

7.1.1 MAC address management

Emac physical interfaces can have multiple MAC addresses. This can be used for MAC address filtering - emac physical interfaces can be configured to accept traffic intended for several different recipients (several different destination MAC addresses).

FCI operations related to MAC address management:

- To **add** a new MAC address to emac physical interface:
  1. Lock the interface database.
     (`FPP_CMD_IF_LOCK_SESSION`)
  2. Add a new MAC address to emac physical interface.
     (`FPP_CMD_IF_MAC` + `FPP_ACTION_REGISTER`)
  3. Unlock the interface database.
     (`FPP_CMD_IF_UNLOCK_SESSION`)
- To **remove** a MAC address from emac physical interface:
  1. Lock the interface database.
     (`FPP_CMD_IF_LOCK_SESSION`)
  2. Remove the MAC address from emac physical interface.
     (`FPP_CMD_IF_MAC` + `FPP_ACTION_DEREGISTER`)
  3. Unlock the interface database.
     (`FPP_CMD_IF_UNLOCK_SESSION`)
- To **list** MAC addresses of emac physical interface:
  1. Lock the interface database.
     (`FPP_CMD_IF_LOCK_SESSION`)
  2. Read out MAC address(es) of the target emac physical interface.
     (`FPP_CMD_IF_MAC` + `FPP_ACTION_QUERY` and `FPP_ACTION_QUERY_CONT`)

3. Unlock the interface database.
   (FPP_CMD_IF_UNLOCK_SESSION)

***Note:** When the interface database is locked via FPP_CMD_IF_LOCK_SESSION, only the FCI commands which require the locked database should be used. Failure to adhere to this note may lead to unexpected FCI command failures. FCI commands which require the locked database can be identified in this reference manual by always having extra lock/unlock steps listed in their operation description.*

7.1.2 Mirroring rules management

Physical interfaces can be configured to mirror their ingress or egress traffic. Configuration data for mirroring are managed as separate entities - mirroring rules.

FCI operations related to mirroring rules management:

- To **create** a new mirroring rule:
  (FPP_CMD_MIRROR + FPP_ACTION_REGISTER)
- To **assign** a mirroring rule to a physical interface:
  Write name of the desired mirror rule in `.rx_mirrors[i]` or `.tx_mirrors[i]` property of the physical interface.
  (use steps described in Physical Interface, section **modify**)
- To **update** a mirroring rule:
  (FPP_CMD_MIRROR + FPP_ACTION_UPDATE)
- To **list** available mirroring rules:
  (FPP_CMD_MIRROR + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)

7.1.3 Examples

demo_feature_physical_interface.c

7.2 Logical Interface

Logical interfaces are dynamic objects (definable at runtime) which represent traffic endpoints. They are associated with their respective parent physical interfaces. Logical interfaces can be used for the following purposes:

- To forward traffic from PFE to a host.
- To forward traffic or its replicas between physical interfaces (1:N distribution).
- To serve as classification & forwarding rules for Flexible Router.

Logical interfaces have several configurable properties. See `FPP_CMD_LOG_IF` and `fpp_log_if_cmd_t`.

Logical interfaces can be created and destroyed at runtime. Every *physical* interface can have a list of associated *logical* interfaces. The very first logical interface in the list (tail position) is considered the **default** logical interface of the given physical interface. New logical interfaces are always added to the top of the list (head position), creating a sequence which is ordered from the head (the newest one) back to the tail (the default one). This forms a classification sequence, which is important if the parent physical interface operates in the Flexible Router mode.

Similar to physical interfaces, the logical interfaces can be set to a **promiscuous** mode. For logical interfaces, a promiscuous mode means a logical interface will accept all ingress traffic it is asked to classify, regardless of the interface's active match rules.

FCI operations related to logical interfaces:

- To **create** a new logical interface in PFE:
  1. Lock the interface database.
     (FPP_CMD_IF_LOCK_SESSION)
  2. Create a new logical interface.
     (FPP_CMD_LOG_IF + FPP_ACTION_REGISTER)
  3. Unlock the interface database.
     (FPP_CMD_IF_UNLOCK_SESSION)
- To **remove** a logical interface from PFE:
  1. Lock the interface database.
     (FPP_CMD_IF_LOCK_SESSION)
  2. Remove the logical interface.
     (FPP_CMD_LOG_IF + FPP_ACTION_DEREGISTER)
  3. Unlock the interface database.
     (FPP_CMD_IF_UNLOCK_SESSION)
- To **list** available logical interfaces:
  1. Lock the interface database.
     (FPP_CMD_IF_LOCK_SESSION)
  2. Read out properties of logical interface(s).
     (FPP_CMD_LOG_IF + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)
  3. Unlock the interface database.
     (FPP_CMD_IF_UNLOCK_SESSION)
- To **modify** properties of a logical interface (read-modify-write):
  1. Lock the interface database.
     (FPP_CMD_IF_LOCK_SESSION)
  2. Read out properties of the target logical interface.
     (FPP_CMD_LOG_IF + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)
  3. Locally modify the properties.
     (see fpp_log_if_cmd_t)
  4. Write the modified properties back to PFE.
     (FPP_CMD_LOG_IF + FPP_ACTION_UPDATE)
  5. Unlock the interface database.
     (FPP_CMD_IF_UNLOCK_SESSION)

Note: *When the interface database is locked via FPP_CMD_IF_LOCK_SESSION, only the FCI commands which require the locked database should be used. Failure to adhere to this note may lead to unexpected FCI command failures. FCI commands which require the locked database can be identified in this reference manual by always having extra lock/unlock steps listed in their operation description.*

7.2.1 Examples

demo_feature_flexible_router.c

8 PFE Features

8.1 IPv4/IPv6 Router

Introduction

IPv4/IPv6 Router is a dedicated PFE feature which can forward ingress IP packets to egress physical interfaces based on a configured set of rules. This accelerated forwarding ("fast path" forwarding) is fully executed by PFE without any host involvement (only the initial configuration is needed). The feature can be utilized to offload a host from routine forwarding tasks.

When routing is needed and the feature is inactive, then all IP packets are passed to the host for routing/forwarding decision[1].

When routing is needed and the feature is active, then only the IP packets which don't match any "fast path" criteria are passed to the host for routing/forwarding decision.

8.1.1 Configuration

1. [optional] Reset the Router.
   This clears all existing IPv4/IPv6 routes and conntracks in PFE.
   For IPv4: (`FPP_CMD_IPV4_RESET`)

   For IPv6: (`FPP_CMD_IPV6_RESET`)
2. Create one or more IP routes.
   For both IPv4/IPv6: (`FPP_CMD_IP_ROUTE` + `FPP_ACTION_REGISTER`)
3. Create one or more IPv4/IPv6 conntracks.
   When filling conntrack data, use only valid (existing) IP route IDs.
   For IPv4: (`FPP_CMD_IPV4_CONNTRACK` + `FPP_ACTION_REGISTER`)

   For IPv6: (`FPP_CMD_IPV6_CONNTRACK` + `FPP_ACTION_REGISTER`)
4. Configure the physical interfaces which shall classify their ingress traffic by the Router classification algorithm. Use steps described in Physical Interface (section **modify**) and do the following for each desired physical interface:
   - Set mode of the interface to `FPP_IF_OP_ROUTER`.
   - Enable the interface by setting the flag `FPP_IF_ENABLED`.

Once the Router is operational, all ingress IP packets of the Router-configured physical interfaces are matched against existing conntracks. If a packet matches some existing conntrack, it is processed and modified according to conntrack properties (destination MAC, NAT, PAT, etc.) and then gets fast-forwarded via egress physical interface as specified by route of the conntrack.

8.1.2 Additional operations

- Conntracks are subjected to aging. If no matching packets are detected on a conntrack for a specified time period, the conntrack is automatically removed from PFE.
  To **set** the **timeout** period, use the following command (shared for both IPv4 and IPv6 conntracks):

1 Forwarding based on a host decision is considered a "slow path" forwarding.

(FPP_CMD_IPV4_SET_TIMEOUT)

- To **remove** a route or a conntrack:[2]
  For route: (FPP_CMD_IP_ROUTE + FPP_ACTION_DEREGISTER)

  For IPv4: (FPP_CMD_IPV4_CONNTRACK + FPP_ACTION_DEREGISTER)

  For IPv6: (FPP_CMD_IPV6_CONNTRACK + FPP_ACTION_DEREGISTER)
- To **list** available routes or conntracks:
  For route: (FPP_CMD_IP_ROUTE + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)

  For IPv4: (FPP_CMD_IPV4_CONNTRACK + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)

  For IPv6: (FPP_CMD_IPV6_CONNTRACK + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)
- By default, PFE conntracks decrement TTL of processed IP packets. This behavior can be set/unset for individual conntracks by their flag CTCMD_FLAGS_TTL_DECREMENT.
  To **modify** an already existing conntrack:
  For IPv4: (FPP_CMD_IPV4_CONNTRACK + FPP_ACTION_UPDATE)

  For IPv6: (FPP_CMD_IPV6_CONNTRACK + FPP_ACTION_UPDATE)
- By default, all TCP packets which match some existing PFE TCP conntrack are forwarded via "fast path". However, PFE can be configured to treat differently the matching TCP packets which contain SYN, FIN or RST flag. This special treament is configurable per each Physical Interface.
  See FPP_IF_FF_ALL_TCP for more information.

8.1.3 Examples

demo_feature_router_simple.c

demo_feature_router_nat.c

8.2 L2 Bridge (Switch)

Introduction

L2 Bridge is a dedicated feature to offload a host from tasks related to MAC address-based forwarding of Ethernet frames. PFE can be configured to act as a network switch, implementing the following functionality:

- **MAC table:** L2 Bridge uses its own MAC table to keep track of encountered MAC addresses. Each MAC table entry consists of a MAC address and a physical interface which should be used to reach the given MAC address. MAC table entries can be dynamic (learned) or static.
- **MAC address learning:** L2 Bridge is capable of automatically adding (learning) new MAC table entries from ingress frames with new (not yet encountered) source MAC addresses.
- **Aging:** MAC table entries are subjected to aging. If a MAC table entry is not used for a certain (hardcoded) time period, it is automatically removed from the MAC table. Static entries are not affected by aging.
- **Static entries:** It is possible to manually add static (non-aging) entries to the MAC table. Static entries can be used as a part of L2 Bridge forward-only configuration (with

2 Note: Removing a route which is used by some conntracks causes the associated connntracks to be removed as well.

MAC learning disabled). With such a setup, only a predetermined traffic (matching the static entries) will be forwarded.
- **Blocking states of physical interfaces:** Each physical interface which is configured to be a part of the L2 Bridge can be finetuned to allow/deny MAC learning or frame forwarding of its ingress traffic. See `fpp_phy_if_block_state_t`.
- **Port migration:** If there is already a learned MAC table entry (a MAC address + a target physical interface) and the MAC address is detected on another interface, then the entry is automatically updated (new target physical interface is set).
- **VLAN Awareness:** The L2 Bridge uses its own VLAN table to support VLAN-based policies like Ingress or Egress port membership. It also supports configuration of bridge domain ports (represented by physical interfaces) to provide VLAN tagging and untagging services, effectively allowing creation of access / trunk ports.

The L2 Bridge utilizes PFE HW accelerators to perform highly optimized MAC and VLAN table lookups. Host is responsible only for the initial bridge configuration via the FCI API.

L2 Bridge VLAN Awareness and Domains

The VLAN awareness is based on entities called Bridge Domains (BD), which are visible to both the classifier firmware and the driver. BDs are used to abstract particular VLANs. Every BD has a configurable set of properties (see `fpp_l2_bd_cmd_t`):

- Associated VLAN ID.
- Set of physical interfaces which represent ports of the BD.
- Information about which ports are tagged or untagged.
  - Tagged port adds a VLAN tag to egressed frames if they are not VLAN tagged, or keeps the tag of the frames intact if they are already VLAN tagged.
  - Untagged port removes the VLAN tag from egressed frames if the frames are VLAN tagged.
- Instruction how to process matching uni-cast frames.
- Instruction how to process matching multi-cast frames.

The L2 Bridge recognizes several BD types:

- **Default BD:**
  Factory default VLAN ID of this bridge domain is **1**. This domain is used to process ingress frames which either have a VLAN tag equal to the Default BD's VLAN ID, or don't have a VLAN tag at all (untagged Ethernet frames).
- **Fall-back BD:**
  This domain is used to process ingress frames which have an unknown VLAN tag. Unknown VLAN tag means that the VLAN tag does not match any existing standard BD nor the default BD.
- **Standard BD:**
  Standard user-defined bridge domains. Used by a VLAN-aware Bridge. These BDs process ingress frames which have a VLAN tag that matches the BD's VLAN ID.

8.2.1 Configuration

1. Create a bridge domain (VLAN domain).
   (`FPP_CMD_L2_BD` + `FPP_ACTION_REGISTER`)
2. Configure hit/miss actions of the bridge domain.
   (`FPP_CMD_L2_BD` + `FPP_ACTION_UPDATE`)
3. Configure which physical interfaces are considered members (ports) of the bridge domain. Also specify which ports are VLAN tagged and which ports are not.

(FPP_CMD_L2_BD + FPP_ACTION_UPDATE)

4. Repeat previous steps to create all required bridge domains (VLAN domains).
   (note that physical interfaces can be members of multiple bridge domains)
5. Configure the physical interfaces which shall classify their ingress traffic by the VLAN-aware Bridge classification algorithm. Use steps described in Physical Interface (section **modify**) and do the following for each desired physical interface:
   - Set mode of the interface to `FPP_IF_OP_VLAN_BRIDGE`.
   - Enable the promiscuous mode by setting the flag `FPP_IF_PROMISC`.
   - Enable the interface by setting the flag `FPP_IF_ENABLED`.

Once the L2 Bridge is operational, ingress Ethernet frames of the Bridge-configured physical interfaces are processed according to setup of bridge domains. VLAN tag of every ingress frame is inspected and the frame is then processed by an appropriate bridge domain.

8.2.2 Additional operations

- To **remove** a bridge domain:[3]
  (FPP_CMD_L2_BD + FPP_ACTION_DEREGISTER)
- To **list** available bridge domains:
  (FPP_CMD_L2_BD + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)
- To **modify** properties of the target bridge domain (read-modify-write):
  1. Read properties of the target bridge domain.
     (FPP_CMD_L2_BD + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)
  2. Locally modify the properties.
     (see fpp_l2_bd_cmd_t)
  3. Write the modified properties back to PFE.
     FPP_CMD_L2_BD + FPP_ACTION_UPDATE)

8.2.3 Static MAC table entries

It is possible to manually add static (non-aging) entries to the MAC table. Static entries can be used as a part of L2 Bridge forward-only configuration (with MAC learning disabled). With such a setup, only a predetermined traffic (matching the static entries) will be forwarded.

FCI operations related to MAC table static entries:

- To **create** a new static entry:
  (FPP_CMD_L2_STATIC_ENT + FPP_ACTION_REGISTER)
- To **remove** a static entry:
  (FPP_CMD_L2_STATIC_ENT + FPP_ACTION_DEREGISTER)
- To **list** available static entries:
  (FPP_CMD_L2_STATIC_ENT + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)
- To **modify** properties of a static entry (read-modify-write):
  1. Read properties of the target static entry.
     (FPP_CMD_L2_STATIC_ENT + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)
  2. Locally modify the properties.

3 Note: Default BD and Fall-back BD cannot be removed.

(see fpp_l2_static_ent_cmd_t)

3. Write the modified properties back to PFE.
(FPP_CMD_L2_STATIC_ENT + FPP_ACTION_UPDATE)

- To **flush** (remove) all static entries in PFE:
(FPP_CMD_L2_FLUSH_STATIC)

8.2.4 Examples

demo_feature_L2_bridge_vlan.c

8.3 L2L3 Bridge

Introduction

L2L3 Bridge is an extension of the L2 Bridge and IP Router features. It allows both features to be simultaneously available on a physical interface. Traffic with specific destination MAC addresses is passed to the IP Router. The rest is handled by the L2 Bridge.

8.3.1 Configuration

1. Configure IPv4/IPv6 Router.
2. Configure L2 Bridge.
3. Create at least one MAC table static entry with the 'local' flag. Note that if a static entry is configured as local, then its egress list is ignored. Also note that 'local' static entries must have a correct VLAN (and MAC address) in order to properly match the ingress traffic.
First: (FPP_CMD_L2_STATIC_ENT + FPP_ACTION_REGISTER)

   Then: (FPP_CMD_L2_STATIC_ENT + FPP_ACTION_UPDATE)
4. Configure the physical interfaces which shall classify their ingress traffic by the L2L3 Bridge classification algorithm. Use steps described in Physical Interface (section **modify**) and do the following for each desired physical interface:
   - Set mode of the interface to FPP_IF_OP_L2L3_VLAN_BRIDGE.
   - Enable the promiscuous mode by setting the flag FPP_IF_PROMISC.
   - Enable the interface by setting the flag FPP_IF_ENABLED.

Once the L2L3 Bridge is operational, it checks the ingress traffic of L2L3 Bridge-configured physical interfaces against 'local' static entries in the L2 Bridge MAC table. If traffic's destination MAC matches a MAC address of some 'local' static entry, then the traffic is passed to the IP Router. Otherwise the traffic is passed to the L2 Bridge.

8.3.2 Examples

demo_feature_L2L3_bridge_vlan.c

8.4 Flexible Parser

Introduction

Flexible Parser is a PFE firmware-based feature which can classify ingress traffic according to a set of custom classification rules. The feature is intended to be used as an extension of other PFE features/classification algorithms. Flexible Parser consists of the following elements:

- **FP rule:** classification rule. See FPP_CMD_FP_RULE. FP rules inspect content of Ethernet frames. Based on the inspection result (whether the condition of a rule is satisfied or not), a next step of the Flexible Parser classification process is taken.
- **FP table:** An ordered set of FP rules. See FPP_CMD_FP_TABLE. These tables can be assigned as extensions of other PFE features/classification algorithms. Namely, they can be used as an argument for:
  - Flexible Filter of a physical interface. See fpp_phy_if_cmd_t.ftable. Flexible Filter acts as a traffic filter, pre-emptively discarding ingress traffic which is rejected by the associated FP table. Accepted traffic is then processed according to mode of the physical interface.
  - FPP_IF_MATCH_FP0 / FPP_IF_MATCH_FP1 match rules of a logical interface. See Flexible Router.

Flexible Parser classification introduces a performance penalty which is proportional to a count of rules and complexity of a used table. Always consider whether the use of this feature is really necessary. If it is necessary, then try to use FP tables with as few rules as possible.

8.4.1 Configuration

1. Create one or multiple FP rules.
   (FPP_CMD_FP_RULE + FPP_ACTION_REGISTER)
2. Create one or multiple FP tables.
   (FPP_CMD_FP_TABLE + FPP_ACTION_REGISTER)
3. Assign rules to tables. Each rule can be assigned only to one table.
   (FPP_CMD_FP_TABLE + FPP_ACTION_USE_RULE)
4. [optional] If required, an FP rule can be removed from an FP table. The rule can be then assigned to a different table.
   (FPP_CMD_FP_TABLE + FPP_ACTION_UNUSE_RULE)
5. Use FP tables wherever they are required. See FP table.

Once an FP table is configured and put to use, it will start classifying the ingress traffic in whatever role it was assigned to (See FP table). Classification always starts from the very first rule of the table (index 0). Normally, rules of the table are evaluated sequentially till the traffic is either accepted, rejected, or the end of the table is reached. If the end of the table is reached and the traffic is still not accepted nor rejected, then Flexible Parser automatically rejects it.

Based on the action of an FP rule, it is possible to make a jump from the currently evaluated rule to any other rule in the same table. This can be used in some complex scenarios.

Warning: *Do not modify FP tables which are already in use! Always first remove the FP table from use, then modify it (add/delete/rearrange rules), then put it back to its use. Failure to adhere to this warning will result in an undefined behavior of Flexible Parser.*

Warning: *It is prohibited to use jumps to create loops. Failure to adhere to this warning will result in an undefined behavior of Flexible Parser.*

8.4.2 Additional operations

- It is advised to always remove rules and tables which are not needed, because these unused objects would needlessly occupy limited internal memory of PFE.
  To **remove** an FP rule or an FP table:
  For FP rules: (FPP_CMD_FP_RULE + FPP_ACTION_DEREGISTER)

  For FP tables: (FPP_CMD_FP_TABLE + FPP_ACTION_DEREGISTER)
- To **list** FP rules or FP tables:
  For FP rules: (FPP_CMD_FP_RULE + FPP_ACTION_QUERY and FPP_ACTION_QUERY)

  For FP tables: (FPP_CMD_FP_TABLE + FPP_ACTION_QUERY and FPP_ACTION_QUERY)

8.4.3 FP table example

This is an example of how a Flexible Parser table can look like.

- Every row is one FP rule.
- The classification process starts from the rule 0.
- ACCEPT/REJECT means the classification is terminated with the given result.
- CONTINUE means that the next rule in a sequence (next row) shall be evaluated.
- NEXT_RULE <name> means that the next rule to evaluate shall be the rule <name>.
- FrameData is an inspected value froman ingress Ethernet frame. Each rule can inspect a different value from the frame. See FPP_CMD_FP_RULE and fpp_fp_rule_props_t, fields .offset and .offset_from.
- RuleData is a template value inside the FP rule. It is compared with the inspected value from the ingress Ethernet frame.
- Mask is a bitmask specifying which bits of the RuleData and FrameData shall be compared (the rest of the bits is ignored).

Table 2. FP table example

i	Rule	Flags	Mask	Condition of the rule + actions
0	MyR_01	.invert=true FP_REJECT	!=0	if ((FrameData & Mask) **!=** (RuleData & Mask)) then REJECT else CONTINUE
1	MyR_02	FP_ACCEPT	!=0	if ((FrameData & Mask) == (RuleData & Mask)) then ACCEPT else CONTINUE
2	MyR_03	FP_NEXT_RULE	!=0	if ((FrameData & Mask) == (RuleData & Mask)) then NEXT_RULE MyR_11 else CONTINUE
3	MyR_04r	FP_REJECT	==0	REJECT
4	MyR_11	.invert=true FP_NEXT_RULE	!=0	if ((FrameData & Mask) **!=** (RuleData & Mask)) then NEXT_RULE MyR_21 else CONTINUE
5	MyR_12a	FP_ACCEPT	==0	ACCEPT
6	MyR_21	.invert=true FP_ACCEPT	!=0	if ((FrameData & Mask) **!=** (RuleData & Mask)) then ACCEPT else CONTINUE
7	MyR_22r	FP_REJECT	==0	REJECT

8.4.4 Examples

demo_feature_flexible_filter.c

8.5 Flexible Router

Introduction

Flexible Router is a PFE firmware-based feature which uses logical interfaces (and their match rules) to classify ingress traffic. Replicas of the accepted traffic can be forwarded to one or multiple physical interfaces.

Flexible Router classification introduces a performance penalty which is proportional to a count of used logical interfaces (and their match rules). Always consider whether the use of this feature is really necessary. If it is necessary, then try to use as few logical interfaces as possible.

8.5.1 Configuration

1. Lock the interface database.
   (`FPP_CMD_IF_LOCK_SESSION`)
2. Create one or multiple logical interfaces. See Logical Interface for more info. For Flexible Router purposes, pay attention to the order of logical interfaces.
   (`FPP_CMD_LOG_IF` + `FPP_ACTION_REGISTER`)
3. Configure the logical interfaces. Use steps described in Logical Interface (section **modify**) and do the following for each desired logical interface:
   - [optional] Set interface properties such as egress, match rules and match rule arguments.
   - [optional] If multiple match rules are used, then set or clear the flag `FPP_IF_MATCH_OR` in order to specify a logical relation between the rules.
   - Enable the interface by setting the flag `FPP_IF_ENABLED`.
4. Configure the physical interfaces which shall classify their ingress traffic by the Flexible Router classification algorithm. Use steps described in Physical Interface (section **modify**) and do the following for each desired physical interface:
   - Set mode of the interface to `FPP_IF_OP_FLEXIBLE_ROUTER`.
   - Enable the interface by setting the flag `FPP_IF_ENABLED`.
5. Unlock the interface database.
   (`FPP_CMD_IF_UNLOCK_SESSION`)

Note: *When the interface database is locked via `FPP_CMD_IF_LOCK_SESSION`, only the FCI commands which require the locked database should be used. Failure to adhere to this note may lead to unexpected FCI command failures. FCI commands which require the locked database can be identified in this reference manual by always having extra lock/unlock steps listed in their operation description.*

Once the Flexible Router is operational, it classifies the ingress traffic of Flexible Router configured physical interfaces. The process is based on the classification sequence of logical interfaces (see Logical Interface). Classifier walks through the sequence from the head position back to tail, matching the ingress traffic against match rules of logical interfaces which are in the sequence. If a match is found (traffic conformswith match rules of the given logical interface), then the traffic is processed according to the interface's configuration (forwarded, dropped, sent to a host, etc.).

8.5.2 Configuration example

The following example shows a scenario where emac1 physical interface is configured in the `FPP_IF_OP_FLEXIBLE_ROUTER` mode. Goal is to classify ingress traffic on emac1 interface. If the traffic matches classification criteria, a replica of the traffic is egressed through both emac2 and hif0 interfaces.

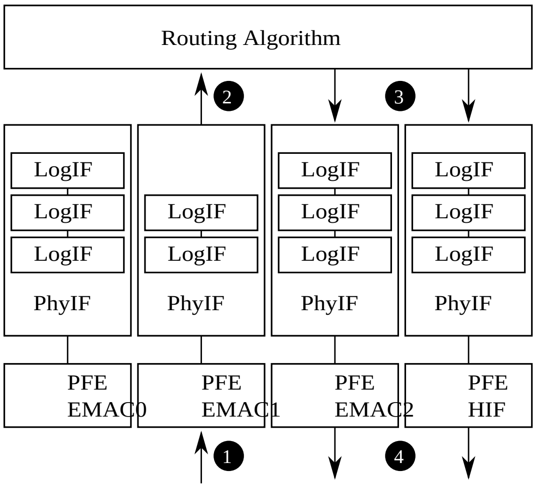

Figure 1. Flexible Router Configuration Example

1. Traffic is ingressed (received) through emac1 port of PFE.
2. If some logical interface accepts the traffic, then information about the matching logical interface (and its parent physical interface) is passed to the Routing Algorithm. Algorithm reads the logical interface and retrieves forwarding properties.
3. Traffic is forwarded by the Routing Algorithm based on the provided information. In this example, the logical interface specified that a replica of the traffic shall be forwarded to both emac2 and hif0 interfaces.
4. Traffic is transmitted via physical interfaces.

8.5.3 Examples

demo_feature_flexible_router.c

8.6 IPsec Offload

Introduction

The IPsec offload feature is a premium one and requires a special premium firmware version to be available for use. It allows the chosen IP frames to be transparently encoded by the IPsec and IPsec frames to be transparently decoded without the CPU intervention using just the PFE and HSE engines.

The SPD database needs to be established on an interface which contains entries describing frame match criteria together with the SA ID reference to the SA established within the HSE describing the IPsec processing criteria. Frames matching the criteria are then processed by the HSE according to the chosen SA and returned for the classification via physical interface of UTIL PE. Normal classification follows the IPsec processing thus the decrypted packets can be e.g. routed.

Warning: *The IPsec offload feature is available only for some Premium versions of PFE firmware. The feature should not be used with a firmware which does not support it. Failure to adhere to this warning will result in an undefined behavior of PFE.*

FCI operations related to IPsec offload:

- To **create** a new SPD entry in the SPD table of a physical interface:
  (`FPP_CMD_SPD` + `FPP_ACTION_REGISTER`)
- To **remove** an SPD entry from the SPD table of a physical interface:
  (`FPP_CMD_SPD` + `FPP_ACTION_DEREGISTER`)
- To **list** existing SPD entries from the SPD table of a physical interface:
  (`FPP_CMD_SPD` + `FPP_ACTION_QUERY` and `FPP_ACTION_QUERY_CONT`)

The HSE also requires the configuration via interfaces of the HSE firmware which is out of the scope of this document. The SAs referenced within the SPD entries must exist prior to creation of the respective SPD entry.

8.6.1 Examples

demo_feature_spd.c

8.7 Egress QoS

Introduction

The Egress QoS allows user to prioritize, aggregate and shape traffic intended to leave the accelerator through some Physical Interface. Egress QoS is implemented as follows:

- Each **emac** physical interface has its own QoS block.
- All **hif** physical interfaces share one common QoS block.

LIMITATIONS

Egress QoS Scheduler:

- Modification of **emac** Egress QoS Scheduler is permitted only if the given emac interface was **not** yet enabled (`FPP_IF_ENABLED` flag was **not** yet set) since the PFE reset. After the given emac interface gets enabled for the first time, do not attempt any further modifications of its Egress QoS scheduler till the PFE gets reset.
- Modification of **hif** Egress QoS Scheduler is prohibited.

8.7.1 QoS block parameters

Every QoS block has a platform-specific number of queues, schedulers and shapers. The following applies for each S32G/PFE QoS block:

- **Queues:**
  - For each **emac**:
    - Number of queues: 8
      (see Traffic queueing algorithm)
    - Size of a queue slot pool: 255
      (see Queue slot pools)
    - Probability zones per queue: 8
  - For each **hif**:
    - Number of queues: 2
      (see Traffic queueing algorithm)
    - Size of a queue slot pool: 32
      (see Queue slot pools)
    - Probability zones per queue: 8
- **Schedulers:**
  - For each **emac**:
    - Number of schedulers: 2
    - Number of scheduler inputs: 8
    - Traffic sources which can be connected to scheduler inputs:
      (see Table 3 and `fpp_qos_scheduler_cmd_t.input_src`)
  - For each **hif**:
    Modification of **hif** Egress QoS Scheduler is prohibited.
    (see LIMITATIONS)

Table 3. Egress QoS Scheduler: input sources

Source	Description
0 - 7	Queue 0 - 7
8	Output of Scheduler 0
255	Invalid (nothing connected)

- **Shapers:**
  - For each **emac**:
    - Number of shapers: 4
    - Shaper positions:
      (see Table 4 and `fpp_qos_shaper_cmd_t.position`)
  - Shared by **all hifs**:
    - Number of shapers shared by all hifs: 4
    - Shaper positions:
      (see Table 4 and `fpp_qos_shaper_cmd_t.position`)

Table 4. Egress QoS Shaper: positions

Position	Description
0	Output of Scheduler 1 (QoS master output)
1 - 8	Input 0 - 7 of Scheduler **1**
9 - 16	Input 0 - 7 of Scheduler **0**
255	Invalid (shaper disconnected)

Note: *Only shapers connected to common scheduler inputs are aware of each other and share the 'conflicting transmission' signal.*

Queue slot pools

Every QoS block has it own pool of queue slots. These slots can be assigned to particular queues. Length of a queue is equal to number of assigned slots. It is possible to configure queue lengths via FCI API. Setting a queue length (`fpp_qos_queue_cmd_t.max`) means assigning given number of slots to the given queue. Sum of all queue lengths of the particular physical interface cannot be bigger than size of its queue slot pool.

See section Queues for info about queue slot pool sizes for various physical interfaces.

```
Examples of queue slots distribution:
=====================================
case 1:
    Example of asymmetrical queue slots distribution for emac0.
    Only 241 queue slots from the emac0 queue slot pool are utilized.
    The remaining 14 free queue slots are simply unused.
    They could be used to lenghten some emac0 queues, if needed.
    Emac0 queue slot distribution in this example:
        [*] queue 0      : 150 slots
        [*] queue 1 .. 7 : 13 slots per each queue.

case 2:
    Example of symmetrical queue slots distribution for hif0.
    All 32 queue slots of the hif0 are utilized.
    There are no free queue slots left on hif0.
    Hif0 queue slot distributuion in this example:
        [*] queue 0 : 16 slots
        [*] queue 1 : 16 slots
```

Traffic queueing algorithm

An algorithm that sorts the egressing traffic into particular Egress QoS queues. Produces the following results:

- For **emac**:
  ID of a queue (0 .. 7) where to store the traffic.

- For **hif**:
  ID of a queue (0 or 1) where to store the traffic.
  Hifs have only two queues:
  - 0 : Low priority queue (L)
  - 1 : High priority queue (H)

Pseudocode which represents the traffic queueing algorithm:

```
get_queue_for_packet(pkt)
{
    queue = 0;
    if (pkt.hasVlanTag)
    {
        queue = pkt.VlanHdr.PCP;
    }
    else
    {
        if (pkt.isIPv4)
        {
            queue = (pkt.IPv4Hdr.DSCP) / 8;
        }
        if (pkt.isIPv6)
        {
            queue = (pkt.IPv6Hdr.TrafficClass.DS) / 8;
        }
    }

    if (is_hif)
    {
        queue = (queue < 4) ? 0 : 1;
    }

    return queue;
}
```

8.7.2 Configuration

By default, the egress QoS topology looks like this:

```
                SCH1
                (RR)
             +---------+
      Q0--->| 0        |
      Q1--->| 1        |
      Q2--->| 2        |
      Q3--->| 3        +--->
      Q4--->| 4        |
      ...   | ...      |
      Q7--->| 7        |
             +---------+
```

All queues are connected to Scheduler 1 and the scheduler discipline is set to Round Robin. Rate mode is set to Data Rate (bps). Queues are in Tail Drop mode.

FCI operations related to Egress QoS:

- To **list QoS queue** properties:
  (FPP_CMD_QOS_QUEUE + FPP_ACTION_QUERY)
- To **list QoS scheduler** properties:
  (FPP_CMD_QOS_SCHEDULER + FPP_ACTION_QUERY)
- To **list QoS shaper** properties:
  (FPP_CMD_QOS_SHAPER + FPP_ACTION_QUERY)

- To **modify QoS queue** properties (read-modify-write):
  1. Read QoS queue properties.
     (FPP_CMD_QOS_QUEUE + FPP_ACTION_QUERY)
  2. Locally modify the properties.

(see fpp_qos_queue_cmd_t)

3. Write the modified properties back to PFE.
(FPP_CMD_QOS_QUEUE + FPP_ACTION_UPDATE)

- To **modify QoS scheduler** properties (read-modify-write): [4]
  1. Read QoS scheduler properties.
  (FPP_CMD_QOS_SCHEDULER + FPP_ACTION_QUERY)
  2. Locally modify the properties.
  (see fpp_qos_scheduler_cmd_t)
  3. Write the modified properties back to PFE.
  (FPP_CMD_QOS_SCHEDULER + FPP_ACTION_UPDATE)
- To **modify QoS shaper** properties (read-modify-write):
  1. Read QoS shaper properties.
  (FPP_CMD_QOS_SHAPER + FPP_ACTION_QUERY)
  2. Locally modify the properties.
  (see fpp_qos_shaper_cmd_t)
  3. Write the modified properties back to PFE.
  (FPP_CMD_QOS_SHAPER + FPP_ACTION_UPDATE)

8.7.3 Examples

demo_feature_qos.c

8.8 Ingress QoS

Introduction

The Ingress QoS allows user to prioritize, aggregate and shape traffic as it comes into the accelerator through an **emac** Physical Interface, before it is further processed by the accelerator.

Each **emac** physical interface has its own Ingress QoS Policer block. Every Policer block has its dedicated flow classification table, WRED queues and Ingress QoS shapers. Exact size of flow classification table and exact numbers/limits of WRED queues and Ingress QoS shapers are platform-specific.

8.8.1 Policer block parameters

The following applies for each S32G/PFE Ingress QoS block ("policer"):

- **Flow** classification table:
  - Maximum number of flows: 64
- **WRED** queues:
  - Number of queues: 3 (DMEM, LMEM, RXF)
  - Maximum queue depth: 8192 for DMEM ; 512 for LMEM and RXF
  - Probability zones per queue: 4

4 Note: Modification of QoS Scheduler properties is constrained by certain limitations. See LIMITATIONS.

- **Ingress QoS shapers:**
  - Number of shapers: 2

8.8.2 Configuration

By default, the Ingress QoS block ("policer") is organized as follows:

```
                              policer
         +-------------------------------------------------+
         |                                                 |
         |  +------------+   +--------+   +-------------+  |
Ingress  |  |            |   |  WRED  |   | Ingress QoS |  |   Further PFE
traffic--+->| flow table +-->| queues +-->|   shapers   +--+-->processing
         |  |            |   |        |   |             |  |
         |  +-----+------+   +---+----+   +-------------+  |
         |        |              |                         |
         +--------+--------------+-------------------------+
                  |              |
                 possible packet drop
```

- `policer` (FPP_CMD_QOS_POLICER)
  - The Ingress QoS block ("policer") itself.
  - The whole block can be enabled/disabled. If the block is disabled, it is bypassed and does not affect performance.

- `flow table` which contains flows (FPP_CMD_QOS_POLICER_FLOW)
  - Flow classification table. Contains user-defined flows.
  - Each flow represents a certain criteria, such as traffic type to match (VLAN, ARP, IPv4, etc.) or some data within the traffic to match (match VLAN ID, match IP address, etc).
  - Ingressing traffic is compared with flows and their criteria. If traffic matches some flow, then (based on flow action), the traffic gets either dropped or marked as Managed or Reserved. Traffic which does not match any flow from the table is marked as Unmanaged.

- `WRED queues` (FPP_CMD_QOS_POLICER_WRED)
  - Ingress QoS WRED queues. These queues (by HW design) always use WRED algorithm.
  - Individual queues can be disabled. If all queues are disabled, then the WRED queueing module is bypassed.
  - Traffic is queued (or possibly dropped) based on the momentary queue fill and also based on the marking of the traffic (Unmanaged/Managed/Reserved). See description of `fpp_iqos_wred_thr_t` enum members.

- `Ingress QoS shapers` (FPP_CMD_QOS_POLICER_SHP)
  - Ingress QoS shapers. These shapers can be used to shape ingress traffic to ensure optimal data flow.
  - Individual shapers can be disabled. If all shapers are disabled, then the Ingress QoS shaper module is bypassed.

– Shapers can be assigned to shape one of several predefined traffic types. See description of fpp_iqos_shp_type_t enum members.

FCI operations related to Ingress QoS:

- To **get** Ingress QoS **policer** status:
  (FPP_CMD_QOS_POLICER + FPP_ACTION_QUERY)
- To **list** Ingress QoS **flow** properties:
  (FPP_CMD_QOS_POLICER_FLOW + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)
- To **list** Ingress QoS **WRED** queue properties:
  (FPP_CMD_QOS_POLICER_WRED + FPP_ACTION_QUERY)
- To **list** Ingress QoS **shaper** properties:
  (FPP_CMD_QOS_POLICER_SHP + FPP_ACTION_QUERY)

- To **enable/disable** Ingress QoS **policer**:
  (FPP_CMD_QOS_POLICER + FPP_ACTION_UPDATE)
- To **add** Ingress QoS **flow** to flow classification table:
  (FPP_CMD_QOS_POLICER_FLOW + FPP_ACTION_REGISTER)
- To **remove** Ingress QoS **flow** from flow classification table:
  (FPP_CMD_QOS_POLICER_FLOW + FPP_ACTION_DEREGISTER)
- To **modify** Ingress QoS **WRED** queue properties (read-modify-write):
  1. Read Ingress QoS WRED queue properties.
     (FPP_CMD_QOS_POLICER_WRED + FPP_ACTION_QUERY)
  2. Locally modify the properties.
     (see fpp_qos_policer_wred_cmd_t)
  3. Write the modified properties back to PFE.
     (FPP_CMD_QOS_POLICER_WRED + FPP_ACTION_UPDATE)
- To **modify** Ingress QoS **shaper** properties (read-modify-write):
  1. Read Ingress QoS shaper properties.
     (FPP_CMD_QOS_POLICER_SHP + FPP_ACTION_QUERY)
  2. Locally modify the properties.
     (see fpp_qos_policer_shp_cmd_t)
  3. Write the modified properties back to PFE.
     (FPP_CMD_QOS_POLICER_SHP + FPP_ACTION_UPDATE)

8.8.3 Examples

demo_feature_qos_policer.c

8.9 FW features

Introduction

PFE Firmware offers several features which are configurable via FCI API. Some FW features are simple switches which toggle a specific PFE Firmware functionality.

Other FW features are more complex and have additional extension data (FW feature elements).

Note: *For details about Firmware features, see appropriate documentation of FW features and their elements.*

FCI operations related to FW features:

- To **list** all FW features:
  (FPP_CMD_FW_FEATURE + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)
- To **enable/disable** a FW feature:
  (FPP_CMD_FW_FEATURE + FPP_ACTION_UPDATE)

8.9.1 FW feature elements

Extension data for FW features. Highly feature-specific. Each FW feature element contains some data. This data may be one value or an array of values. Element data may be editable, or it may be read-only. This all depends on the particular element and its purpose. For details, see appropriate documentation of FW feature elements.

FCI operations related to FW feature elements:

- To **list** elements of a target FW feature:
  (FPP_CMD_FW_FEATURE_ELEMENT + FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT)
- To **query** a particular element of a target FW feature:
  (FPP_CMD_FW_FEATURE_ELEMENT + FPP_ACTION_QUERY)
- To **update** a FW feature element:
  (FPP_CMD_FW_FEATURE_ELEMENT + FPP_ACTION_UPDATE)

8.9.2 Examples

demo_fwfeat.c

9 Commands

9.1 FPP_CMD_PHY_IF

Related topics:	Physical Interface, FPP_CMD_IF_LOCK_SESSION
Related data types:	fpp_phy_if_cmd_t

```
/**
 * @def         FPP_CMD_PHY_IF
 *
 * @brief       FCI command for management of physical interfaces.
 *
 * @details     Related data types: fpp_phy_if_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of a physical interface.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) a physical interface query session and get properties
 *                   of the first physical interface from the internal list of physical interfaces.
```

```
 *               - FPP_ACTION_QUERY_CONT
 *                    Continue the query session and get properties of the next physical interface
 *                    from the list. Intended to be called in a loop (to iterate through the list).
 *
 * @note         All operations with physical interfaces require exclusive
 *               lock of the interface database. See FPP_CMD_IF_LOCK_SESSION.
 */
#define FPP_CMD_PHY_IF
```

9.1.1 Actions

FPP_ACTION_UPDATE

Modify properties of a physical interface. It is recommended to use the read-modify write approach (see Physical Interface). Some properties cannot be modified (see `fpp_phy_if_cmd_t`).

```
fpp_phy_if_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UPDATE,  // Action
  .name   = "...",              // Interface name (see chapter Physical Interface)

  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
             // Some properties cannot be modified (see fpp_phy_if_cmd_t).
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_PHY_IF, sizeof(fpp_phy_if_cmd_t),
                                       (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of a physical interface.

```
fpp_phy_if_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
};

fpp_phy_if_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_PHY_IF,
                sizeof(fpp_phy_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first physical interface from
//  the internal list of physical interfaces.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_PHY_IF,
                sizeof(fpp_phy_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next physical interface from
//  the internal list of physical interfaces.
```

9.1.2 Return values

- FPP_ERR_OK

Success
- `FPP_ERR_IF_ENTRY_NOT_FOUND`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the physical interface query session (no more interfaces).
  - For other ACTIONs:
    Unknown (nonexistent) physical interface was requested.
- `FPP_ERR_IF_WRONG_SESSION_ID`
  Some other client has the interface database locked for exclusive access.
- `FPP_ERR_MIRROR_NOT_FOUND`
  Unknown (nonexistent) mirroring rule in the `.rx_mirrors` or `.tx_mirrors` property.
- `FPP_ERR_FW_FEATURE_NOT_AVAILABLE`
  Attempted to modify properties which are not available (not enabled in FW).
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.1.3 Examples

demo_feature_physical_interface.c

demo_phy_if.c

9.2 FPP_CMD_LOG_IF

Related topics: Logical Interface, `FPP_CMD_IF_LOCK_SESSION`
Related data types: `fpp_log_if_cmd_t`

```
/**
 * @def         FPP_CMD_LOG_IF
 *
 * @brief       FCI command for management of logical interfaces.
 *
 * @details     Related data types: fpp_log_if_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Create a new logical interface.
 *              - FPP_ACTION_DEREGISTER
 *                   Remove (destroy) an existing logical interface.
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of a logical interface.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) a logical interface query session and get properties
 *                   of the first logical interface from the internal collective list of all
 *                   logical interfaces (regardless of physical interface affiliation).
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next logical interface
 *                   from the list. Intended to be called in a loop (to iterate through the list).
 *
 * @note        All operations with logical interfaces require exclusive
 *              lock of the interface database. See FPP_CMD_IF_LOCK_SESSION.
 */
#define FPP_CMD_LOG_IF
```

9.2.1 Actions

FPP_ACTION_REGISTER

Create a new logical interface. The newly created interface is by default disabled and without any configuration. For configuration, see FPP_ACTION_UPDATE.

Warning: *Do not create multiple logical interfaces with the same name.*

```
fpp_log_if_cmd_t cmd_to_fci =
{
  .action      = FPP_ACTION_REGISTER,  // Action
  .name        = "...",                // Interface name (user-defined)
  .parent_name = "..."                 // Parent physical interface name
                                       // (see chapter Physical Interface)
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_LOG_IF, sizeof(fpp_log_if_cmd_t),
                                       (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_DEREGISTER

Remove (destroy) an existing logical interface.

```
fpp_log_if_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_DEREGISTER,  // Action
  .name   = "..."                   // Name of an existing logical interface.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_LOG_IF, sizeof(fpp_log_if_cmd_t),
                                       (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_UPDATE

Modify properties of a logical interface. It is recommended to use the read-modify-write approach (see Logical Interface). Some properties cannot be modified (see fpp_log_if_cmd_t).

```
fpp_log_if_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UPDATE,  // Action
  .name   = "...",              // Name of an existing logical interface.

  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
             // Some properties cannot be modified (see fpp_log_if_cmd_t).
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_LOG_IF, sizeof(fpp_log_if_cmd_t),
                                       (unsigned short*)(&cmd_to_fci));
```

`FPP_ACTION_QUERY` and `FPP_ACTION_QUERY_CONT`

Get properties of a logical interface.

```
fpp_log_if_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
};

fpp_log_if_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_LOG_IF,
                sizeof(fpp_log_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first logical interface from
//  the internal collective list of all logical interfaces.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_LOG_IF,
                sizeof(fpp_log_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next logical interface from
//  the internal collective list of all logical interfaces.
```

9.2.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_IF_ENTRY_NOT_FOUND`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the logical interface query session (no more interfaces).
  - For other ACTIONs:
    Unknown (nonexistent) logical interface was requested.
- `FPP_ERR_IF_ENTRY_ALREADY_REGISTERED`
  Requested logical interface already exists (is already registered).
- `FPP_ERR_IF_WRONG_SESSION_ID`
  Some other client has the interface database locked for exclusive access.
- `FPP_ERR_IF_RESOURCE_ALREADY_LOCKED`
  Same as `FPP_ERR_IF_WRONG_SESSION_ID`.
- `FPP_ERR_IF_MATCH_UPDATE_FAILED`
  Update of match flags has failed.
- `FPP_ERR_IF_EGRESS_UPDATE_FAILED`
  Update of the `.egress` bitset has failed.
- `FPP_ERR_IF_EGRESS_DOESNT_EXIST`
  Invalid (nonexistent) egress physical interface in the `.egress` bitset.
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.2.3 Examples

demo_feature_flexible_router.c

demo_log_if.c

9.3 FPP_CMD_IF_LOCK_SESSION

Related topics: Physical Interface, Logical Interface, Flexible Router, FPP_CMD_IF_UNLOCK_SESSION
Related data types: ---

```
/**
 * @def         FPP_CMD_IF_LOCK_SESSION
 *
 * @brief       FCI command to get exclusive access to interface database.
 *
 * @details     Related data types: ---
 *
 * @details     Supported `.action` values: ---
 */
#define FPP_CMD_IF_LOCK_SESSION
```

Note: *When the interface database is locked via FPP_CMD_IF_LOCK_SESSION, only the FCI commands which require the locked database should be used. Failure to adhere to this note may lead to unexpected FCI command failures. FCI commands which require the locked database can be identified in this reference manual by always having extra lock/unlock steps listed in their operation description.*

9.3.1 Actions

This command is used "as is", without any specific ACTION.

```
int rtn = 0;
rtn = fci_write(client, FPP_CMD_IF_LOCK_SESSION, 0, NULL);
```

9.3.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_IF_RESOURCE_ALREADY_LOCKED
  Some other client has the interface database locked for exclusive access.
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.3.3 Examples

demo_feature_physical_interface.c

demo_feature_flexible_router.c

demo_common.c

9.4 FPP_CMD_IF_UNLOCK_SESSION

Related topics: Physical Interface, Logical Interface, Flexible Router, FPP_CMD_IF_LOCK_SESSION
Related data types: ---

```
 * @def         FPP_CMD_IF_UNLOCK_SESSION
 *
 * @brief       FCI command to cancel exclusive access to interface database.
 *
 * @details     Related data types: ---
 *
 * @details     Supported `.action` values: ---
 */
#define FPP_CMD_IF_UNLOCK_SESSION
```

9.4.1 Actions

This command is used "as is", without any specific ACTION.

```
int rtn = 0;
rtn = fci_write(client, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL);
```

9.4.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_IF_WRONG_SESSION_ID
  Either the database is not locked, or it is currently locked by some other client.
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.4.3 Examples

demo_feature_physical_interface.c

demo_feature_flexible_router.c

demo_common.c

9.5 FPP_CMD_IF_MAC

Related topics: MAC address management, FPP_CMD_IF_LOCK_SESSION
Related data types: fpp_if_mac_cmd_t

```
/**
 * @def         FPP_CMD_IF_MAC
 *
 * @brief       FCI command for management of interface MAC addresses.
 *
 * @details     Related data types: fpp_if_mac_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Add a new MAC address to an interface.
 *              - FPP_ACTION_DEREGISTER
 *                   Remove an existing MAC address from an interface.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) a MAC address query session and get
 *                   the first MAC address of the requested interface.
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get the next MAC address
 *                   of the requested interface. Intended to be called in a loop
 *                   (to iterate through the list).
 *
 * @note        All operations with interface MAC addresses require exclusive
 *              lock of the interface database. See FPP_CMD_IF_LOCK_SESSION.
 *
 * @note        MAC address management is available only for emac physical interfaces.
 */
#define FPP_CMD_IF_MAC
```

9.5.1 Actions

FPP_ACTION_REGISTER

Add a new MAC address to emac physical interface.

```
fpp_if_mac_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_REGISTER,  // Action
  .name   = "...",                // Physical interface name
  .mac    = {...}                 // Physical interface MAC
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_IF_MAC, sizeof(fpp_if_mac_cmd_t),
                                        (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_DEREGISTER

Remove an existing MAC address from emac physical interface.

```
fpp_if_mac_cmd_t cmd_to_fci =
```

```
{
  .action = FPP_ACTION_DEREGISTER,  // Action
  .name   = "...",                  // Physical interface name
  .mac    = {...}                   // Physical interface MAC
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_IF_MAC, sizeof(fpp_if_mac_cmd_t),
                                       (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get MAC addresses of a requested emac physical interface.

```
fpp_if_mac_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
  .name   = "...",            // Physical interface name
};
 *
fpp_if_mac_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;
 *
int rtn = 0;
rtn = fci_query(client, FPP_CMD_IF_MAC,
                sizeof(fpp_if_mac_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds the first MAC address of the requested physical interface.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_IF_MAC,
                sizeof(fpp_if_mac_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds the next MAC address of the requested physical interface.
```

9.5.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_IF_MAC_NOT_FOUND`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the MAC address query session (no more MAC addresses).
  - For other ACTIONs:
    Unknown (nonexistent)MAC address was requested.
- `FPP_ERR_IF_MAC_ALREADY_REGISTERED`
  Requested MAC address already exists (is already registered).
- `FPP_ERR_IF_ENTRY_NOT_FOUND`
  Unknown (nonexistent) physical interface was requested.
- `FPP_ERR_IF_NOT_SUPPORTED`
  Requested physical interface does not support MAC address management.
- `FPP_ERR_IF_WRONG_SESSION_ID`
  Some other client has the interface database locked for exclusive access.
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.

- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.5.3 Examples

demo_feature_physical_interface.c

demo_if_mac.c

9.6 FPP_CMD_MIRROR

Related topics: Mirroring rules management
Related data types: fpp_mirror_cmd_t

```
/**
 * @def         FPP_CMD_MIRROR
 *
 * @brief       FCI command for management of interface mirroring rules.
 *
 * @details     Related data types: fpp_mirror_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Create a new mirroring rule.
 *              - FPP_ACTION_DEREGISTER
 *                   Remove (destroy) an existing mirroring rule.
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of a mirroring rule.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) a mirroring rule query session and get properties
 *                   of the first mirroring rule from the internal list of mirroring rules.
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next mirroring rule
 *                   from the list. Intended to be called in a loop (to iterate through the list).
 */
#define FPP_CMD_MIRROR
```

9.6.1 Actions

FPP_ACTION_REGISTER

Create a new mirroring rule. When creating a new mirroring rule, it is also possible to simultaneously set its properties (using the same rules which apply to FPP_ACTION_UPDATE).

```
fpp_mirror_cmd_t cmd_to_fci =
{
  .action        = FPP_ACTION_REGISTER, // Action
  .name          = "...",               // Name of the mirroring rule.
  .egress_phy_if = "...",               // Name of the physical interface where to mirror.

  // optional
  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_MIRROR, sizeof(fpp_mirror_cmd_t),
                                       (unsigned short*)(&cmd_to_fci));
```


FPP_ACTION_DEREGISTER

Remove (destroy) an existing mirroring rule.

```
fpp_mirror_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_DEREGISTER,  // Action
  .name   = "...",                  // Name of the mirroring rule.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_MIRROR, sizeof(fpp_mirror_cmd_t),
                                       (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_UPDATE

Modify properties of a mirroring rule. It is recommended to use the read-modify-write approach. Some properties cannot be modified (see `fpp_mirror_cmd_t`).

```
fpp_mirror_cmd_t cmd_to_fci =
{
  .action        = FPP_ACTION_UPDATE,   // Action
  .name          = "...",               // Name of the mirroring rule.

  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
             // Some properties cannot be modified (see fpp_mirror_cmd_t).
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_MIRROR, sizeof(fpp_mirror_cmd_t),
                                       (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of a mirroring rule.

```
fpp_mirror_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
};

fpp_mirror_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_MIRROR,
                sizeof(fpp_mirror_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first mirroring rule from
//  the internal list of mirroring rules.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_MIRROR,
                sizeof(fpp_mirror_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next mirroring rule from
//  the internal list of mirroring rules.
```

9.6.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_MIRROR_NOT_FOUND`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the mirroring rule query session (no more mirroring rules).
  - For other ACTIONs:
    Unknown (nonexistent) mirroring rule was requested.
- `FPP_ERR_MIRROR_ALREADY_REGISTERED`
  Requested mirroring rule already exists (is already registered).
- `FPP_ERR_WRONG_COMMAND_PARAM`
  Unexpected value of some property.
- `FPP_ERR_IF_ENTRY_NOT_FOUND`
  Unknown (nonexistent) physical interface in the `.egress_phy_if` property.
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.6.3 Examples

demo_feature_physical_interface.c

demo_mirror.c

9.7 FPP_CMD_L2_BD

Related topics:	L2 Bridge L2L3 Bridge
Related data types:	`fpp_l2_bd_cmd_t`

```
/**
 * @def         FPP_CMD_L2_BD
 *
 * @brief       FCI command for management of L2 bridge domains.
 *
 * @details     Related data types: fpp_l2_bd_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Create a new bridge domain.
 *              - FPP_ACTION_DEREGISTER
 *                   Remove (destroy) an existing bridge domain.
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of a bridge domain.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) a bridge domain query session and get properties
 *                   of the first bridge domain from the internal list of bridge domains.
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next bridge domain
 *                   from the list. Intended to be called in a loop (to iterate through the list).
 */
#define FPP_CMD_L2_BD
```

9.7.1 Actions

FPP_ACTION_REGISTER

Create a new bridge domain. When creating a new bridge domain, it is also possible to simultaneously set its properties (using the same rules which apply to FPP_ACTION_UPDATE).

```
fpp_l2_bd_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_REGISTER,  // Action
  .vlan   = ...,                  // VLAN ID of a new bridge domain. [NBO] (user-defined)

  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
             // Some properties cannot be modified (see fpp_l2_bd_cmd_t).
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_L2_BD, sizeof(fpp_l2_bd_cmd_t),
                                      (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_DEREGISTER

Remove (destroy) an existing bridge domain.

```
fpp_l2_bd_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_DEREGISTER,  // Action
  .vlan   = ...,                    // VLAN ID of an existing bridge domain. [NBO]
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_L2_BD, sizeof(fpp_l2_bd_cmd_t),
                                      (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_UPDATE

Modify properties of a bridge domain. It is recommended to use the read-modify-write approach. Some properties cannot be modified (see fpp_l2_bd_cmd_t).

```
fpp_l2_bd_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UPDATE,  // Action
  .vlan   = ...,                // VLAN ID of an existing bridge domain. [NBO]

  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
             // Some properties cannot be modified (see fpp_l2_bd_cmd_t).
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_L2_BD, sizeof(fpp_l2_bd_cmd_t),
                                      (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of a bridge domain.

```
fpp_l2_bd_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
};

fpp_l2_bd_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_L2_BD,
                sizeof(fpp_l2_bd_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first bridge domain from
//  the internal list of bridge domains.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_L2_BD,
                sizeof(fpp_l2_bd_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next bridge domain from
//  the internal list of bridge domains.
```

9.7.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_L2_BD_NOT_FOUND
  - For FPP_ACTION_QUERY or FPP_ACTION_QUERY_CONT:
    The end of the bridge domain query session (no more bridge domains).
  - For other ACTIONs:
    Unknown (nonexistent) bridge domain was requested.
- FPP_ERR_L2_BD_ALREADY_REGISTERED
  Requested bridge domain already exists (is already registered).
- FPP_ERR_WRONG_COMMAND_PARAM
  Unexpected value of some property.
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.7.3 Examples

demo_feature_L2_bridge_vlan.c

demo_feature_L2L3_bridge_vlan.c

demo_l2_bd.c

9.8 FPP_CMD_L2_STATIC_ENT

Related topics: Static MAC table entries L2 Bridge L2L3 Bridge
Related data types: `fpp_l2_static_ent_cmd_t`

```
/**
 * @def         FPP_CMD_L2_STATIC_ENT
 *
 * @brief       FCI command for management of L2 static entries.
 *
 * @details     Related data types: fpp_l2_static_ent_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Create a new static entry.
 *              - FPP_ACTION_DEREGISTER
 *                   Remove (destroy) an existing static entry.
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of a static entry.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) static entry query session and get properties
 *                   of the first static entry from the internal collective list of all
 *                   L2 static entries (regardless of bridge domain affiliation).
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next static entry
 *                   from the list. Intended to be called in a loop (to iterate through the list).
 *
 * @note        When using this command, it is recommended to disable dynamic learning
 *              of MAC addresses on all physical interfaces which are configured to be
 *              a part of L2 Bridge or L2L3 Bridge.
 *              See FPP_CMD_PHY_IF and fpp_phy_if_block_state_t.
 */
#define FPP_CMD_L2_STATIC_ENT
```

9.8.1 Actions

FPP_ACTION_REGISTER

Create a new L2 static entry.

```
fpp_l2_static_ent_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_REGISTER,  // Action
  .vlan   = ...,                  // VLAN ID of an associated bridge domain. [NBO]
  .mac    = ...,                  // Static entry MAC address.
  .forward_list = ...             // Egress physical interfaces. [NBO]
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_L2_STATIC_ENT, sizeof(fpp_l2_static_ent_cmd_t),
                                               (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_DEREGISTER

Remove (destroy) an existing L2 static entry.

```
fpp_l2_static_ent_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_DEREGISTER,  // Action
  .vlan   = ...,                    // VLAN ID of an associated bridge domain. [NBO]
```

```
  .mac    = ...                     // Static entry MAC address.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_L2_STATIC_ENT, sizeof(fpp_l2_static_ent_cmd_t),
                                              (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_UPDATE

Modify properties of L2 static entry. It is recommended to use the read-modify-write approach. Some properties cannot be modified (see fpp_l2_static_ent_cmd_t).

```
fpp_l2_static_ent_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UPDATE,  // Action
  .vlan   = ...,                // VLAN ID of an associated bridge domain. [NBO]
  .mac    = ...,                // Static entry MAC address.

  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
             // Some properties cannot be modified (see fpp_l2_static_ent_cmd_t).
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_L2_STATIC_ENT, sizeof(fpp_l2_static_ent_cmd_t),
                                              (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of L2 static entry.

```
fpp_l2_static_ent_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
};

fpp_l2_static_ent_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_L2_STATIC_ENT,
                sizeof(fpp_l2_static_ent_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first static entry from
//  the internal collective list of all static entries.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_L2_STATIC_ENT,
                sizeof(fpp_l2_static_ent_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next static entry from
//  the internal collective list of all static entries.
```

9.8.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_L2_STATIC_EN_NOT_FOUND
  – For FPP_ACTION_QUERY or FPP_ACTION_QUERY_CONT:

The end of the L2 static entry query session (no more L2 static entries).
- For other ACTIONs:
  Unknown (nonexistent) L2 static entry was requested.

- `FPP_ERR_L2_STATIC_ENT_ALREADY_REGISTERED`
  Requested L2 static entry already exists (is already registered).
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.8.3 Examples

demo_feature_L2_bridge_vlan.c

demo_feature_L2L3_bridge_vlan.c

demo_l2_bd.c

9.9 FPP_CMD_L2_FLUSH_LEARNED

Related topics: L2 Bridge, L2L3 Bridge
Related data types: ---

```
/**
 * @def         FPP_CMD_L2_FLUSH_LEARNED
 *
 * @brief       FCI command to remove all dynamically learned MAC table entries.
 *
 * @details     Supported `.action` values: ---
 */
#define FPP_CMD_L2_FLUSH_LEARNED
```

9.9.1 Actions

This command is used "as is", without any specific ACTION.

```
int rtn = 0;
rtn = fci_write(client, FPP_CMD_L2_FLUSH_LEARNED, 0, NULL);
```

9.9.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.

- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.9.3 Examples

demo_l2_bd.c

9.10 FPP_CMD_L2_FLUSH_STATIC

Related topics: L2 Bridge, L2L3 Bridge
Related data types: ---

```
/**
 * @def         FPP_CMD_L2_FLUSH_STATIC
 *
 * @brief       FCI command to remove all static MAC table entries.
 *
 * @details     Supported `.action` values: ---
 */
#define FPP_CMD_L2_FLUSH_STATIC
```

9.10.1 Actions

This command is used "as is", without any specific ACTION.

```
int rtn = 0;
rtn = fci_write(client, FPP_CMD_L2_FLUSH_STATIC, 0, NULL);
```

9.10.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.10.3 Examples

demo_l2_bd.c

9.11 FPP_CMD_L2_FLUSH_ALL

Related topics: L2 Bridge, L2L3 Bridge
Related data types: ---

```
/**
 * @def         FPP_CMD_L2_FLUSH_ALL
 *
 * @brief       FCI command to remove all MAC table entries (clear the whole MAC table).
 *
 * @details     Supported `.action` values: ---
 */
#define FPP_CMD_L2_FLUSH_ALL
```

9.11.1 Actions

This command is used "as is", without any specific ACTION.

```
int rtn = 0;
rtn = fci_write(client, FPP_CMD_L2_FLUSH_ALL, 0, NULL);
```

9.11.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.11.3 Examples

demo_l2_bd.c

9.12 FPP_CMD_FP_TABLE

Related topics: Flexible Parser, FPP_CMD_FP_RULE
Related data types: fpp_fp_table_cmd_t, fpp_fp_rule_props_t

```
/**
 * @def         FPP_CMD_FP_TABLE
 *
 * @brief       FCI command for management of Flexible Parser tables.
 *
 * @details     Related data types: fpp_fp_table_cmd_t, fpp_fp_rule_props_t
```

```
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Create a new FP table.
 *              - FPP_ACTION_DEREGISTER
 *                   Remove (destroy) an existing FP table.
 *              - FPP_ACTION_USE_RULE
 *                   Insert an FP rule into an FP table at the specified position.
 *              - FPP_ACTION_UNUSE_RULE
 *                   Remove an FP rule from an FP table.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) an FP table query session and get properties
 *                   of the first FP rule from the requested FP table.
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next FP rule
 *                   from the requested FP table. Intended to be called in a loop
 *                   (to iterate through the requested FP table).
 */
#define FPP_CMD_FP_TABLE
```

9.12.1 Actions

`FPP_ACTION_REGISTER`

Create a new FP table.

```
fpp_fp_table_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_REGISTER,    // Action
  .table_info.t.table_name = "..."  // Name of a new FP table.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_FP_TABLE, sizeof(fpp_fp_table_cmd_t),
                                                (unsigned short*)(&cmd_to_fci));
```

`FPP_ACTION_DEREGISTER`

Remove (destroy) an existing FP table.

Note: *FP table cannot be destroyed if it is in use by some PFE feature. First remove the table from use, then destroy it.*

```
fpp_fp_table_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_DEREGISTER,  // Action
  .table_info.t.table_name = "..."  // Name of an existing FP table.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_FP_TABLE, sizeof(fpp_fp_table_cmd_t),
                                                (unsigned short*)(&cmd_to_fci));
```

`FPP_ACTION_USE_RULE`

Insert an FP rule at the specified position in an FP table.

- If there are already some rules in the table, they are shifted accordingly to make room for the newly inserted rule.

- If the desired position is greater than the count of all rules in the table, the newly inserted rule is placed as the last rule of the table.

Note: *Each FP rule can be assigned only to one FP table (cannot be simultaneously a member of multiple FP tables).*

```
fpp_fp_table_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_USE_RULE,     // Action
  .table_info.t.table_name = "...",  // Name of an existing FP table.
  .table_info.t.rule_name  = "...",  // Name of an existing FP rule.
  .table_info.t.position   = ...     // Desired position of the rule in the table.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_FP_TABLE, sizeof(fpp_fp_table_cmd_t),
                                                (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_UNUSE_RULE

Remove an FP rule from an FP table.

```
fpp_fp_table_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UNUSE_RULE,   // Action
  .table_info.t.table_name = "...",  // Name of an existing FP table.
  .table_info.t.rule_name  = "...",  // Name of an FP rule which is a member of the table.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_FP_TABLE, sizeof(fpp_fp_table_cmd_t),
                                                (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of an FP **rule** from the requested FP table. Query result (properties of the **rule**) is stored in the member `.table_info.r`.

Note: *There is currently no way to read a list of existing FP tables from PFE.*

```
fpp_fp_table_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY         // Action
  .table_info.t.table_name = "...",  // Name of an existing FP table.
};

fpp_fp_table_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_FP_TABLE,
                sizeof(fpp_fp_table_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci.table_info.r' now holds properties of the first FP rule from
//  the requested FP table.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_FP_TABLE,
                sizeof(fpp_fp_table_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci.table_info.r' now holds properties of the next FP rule from
//  the requested FP table.
```

9.12.2 Return values

- `FPP_ERR_OK`
  Success
- `ENOENT (-2)`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the FP table query session (no more FP **rules** in the requested table).
  - For other ACTIONs:
    Unknown (nonexistent) FP table was requested.
- `EEXIST (-17)`
  Requested FP table already exists (is already registered).
- `EACCES (-13)`
  Requested FP table cannot be destroyed (is probably in use by some PFE feature).
- `FPP_ERR_WRONG_COMMAND_PARAM`
  Unexpected value of some property.
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.12.3 Examples

demo_feature_flexible_filter.c

demo_fp.c

9.13 FPP_CMD_FP_RULE

Related topics: Flexible Parser, FPP_CMD_FP_TABLE
Related data types: `fpp_fp_rule_cmd_t`, `fpp_fp_rule_props_t`

```
/**
 * @def         FPP_CMD_FP_RULE
 *
 * @brief       FCI command for management of Flexible Parser rules.
 *
 * @details     Related data types: fpp_fp_rule_cmd_t, fpp_fp_rule_props_t
 *
 * @details     Each FP rule consists of a condition specified by the following properties:
 *              `.data`, `.mask` and `.offset` + `.offset_from`. FP rule then works as follows:
 *              32-bit data value from the inspected Ethernet frame (at given offset_from +
 *              offset position, masked by the mask) is compared with the data value
 *              (masked by the same mask). If the values are equal, then condition of
 *              the FP rule is true. An invert flag may be set to invert the condition result.
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Create a new FP rule.
 *              - FPP_ACTION_DEREGISTER
 *                   Remove (destroy) an existing FP rule.
 *              - FPP_ACTION_QUERY
```

```
 *                      Initiate (or reinitiate) an FP rule query session and get properties
 *                      of the first FP rule from the internal collective list of all
 *                      FP rules (regardless of FP table affiliation).
 *                 - FPP_ACTION_QUERY_CONT
 *                      Continue the query session and get properties of the next FP rule
 *                      from the list. Intended to be called in a loop (to iterate through the list).
 */
#define FPP_CMD_FP_RULE
```

9.13.1 Actions

FPP_ACTION_REGISTER

Create a new FP rule. For detailed info about FP rule properties, see fpp_fp_rule_cmd_t.

```
fpp_fp_rule_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_REGISTER,  // Action
  .r.rule_name = "...",           // Rule name. A string of up to 15 characters + '\0'.
  .r.data      = ...,             // Expected data. [NBO]
  .r.mask      = ...,             // Bitmask. [NBO]
  .r.offset    = ...,             // Offset (in bytes). [NBO]
  .r.invert    = ...,             // Invert the match result.

  .r.next_rule_name = "...",      // Name of the FP rule to jump to if '.match_action' ==
                                  // FP_NEXT_RULE. Set all-zero if unused.

  .r.match_action = ...,          // Action to do if the inspected frame matches
                                  // the FP rule criteria.

  .r.offset_from = ...            // Header for offset calculation.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_FP_RULE, sizeof(fpp_fp_rule_cmd_t),
                                         (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_DEREGISTER

Remove (destroy) an existing FP rule.

Note: *FP rule cannot be destroyed if it is a member of some FP table. First remove the rule from the table, then destroy the rule.*

```
fpp_fp_rule_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_DEREGISTER,  // Action
  .r.rule_name = "...",             // Name of an existing FP rule.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_FP_RULE, sizeof(fpp_fp_rule_cmd_t),
                                         (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of an FP rule. Query result is stored in the member `.r`.

```
fpp_fp_rule_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
};

fpp_fp_rule_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_FP_RULE,
                sizeof(fpp_fp_rule_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci.r' now holds properties of the first FP rule from
//  the internal collective list of all FP rules.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_FP_RULE,
                sizeof(fpp_fp_rule_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci.r' now holds properties of the next FP rule from
//  the internal collective list of all FP rules.
```

9.13.2 Return values

- `FPP_ERR_OK`
  Success
- `ENOENT (-2)`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the FP rule query session (no more FP rules).
  - For other ACTIONs:
    Unknown (nonexistent) FP rule was requested.
- `EEXIST (-17)`
  Requested FP rule already exists (is already registered).
- `EACCES (-13)`
  Requested FP rule cannot be destroyed (is probably a member of some FP table).
- `FPP_ERR_WRONG_COMMAND_PARAM`
  Unexpected value of some property.
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.13.3 Examples

demo_feature_flexible_filter.c

demo_fp.c

9.14 FPP_CMD_DATA_BUF_PUT

Related topics: `FPP_CMD_DATA_BUF_AVAIL`

Related data types: fpp_buf_cmd_t

```
/**
 * @def          FPP_CMD_DATA_BUF_PUT
 *
 * @brief        FCI command to send an arbitrary data to the accelerator.
 *
 * @details      Related data types: fpp_buf_cmd_t
 *
 * @details      Command is intended to be used to send custom data to the accelerator.
 *               Format of the command argument is given by the fpp_buf_cmd_t structure.
 *               Size of the structure also defines the maximum payload length.
 *               Subsequent commands are not successful until the accelerator
 *               reads and acknowledges the current request.
 *
 * @details      Supported `.action` values: ---
 *
 * @note         This feature works only with custom PFE Firmware.
 */
#define FPP_CMD_DATA_BUF_PUT
```

9.14.1 Actions

This command is used "as is", without any specific ACTION.

```
fpp_buf_cmd_t cmd_to_fci =
{
  .payload = ...,  // Fill the buffer with custom payload.
  .len     = ...   // Set payload length in number of bytes.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_DATA_BUF_PUT, sizeof(fpp_buf_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));
```

9.14.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_AGAIN
  Previous command has not been finished yet.
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.14.3 Examples

9.15 FPP_CMD_SPD

Related topics: IPsec Offload
Related data types: fpp_spd_cmd_t

```
/**
 * @def         FPP_CMD_SPD
 *
 * @brief       FCI command for management of the IPsec offload (SPD entries).
 *
 * @details     Related data types: fpp_spd_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Create a new SPD entry.
 *              - FPP_ACTION_DEREGISTER
 *                   Remove (destroy) an existing SPD entry.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) an SPD entry query session and get properties
 *                   of the first SPD entry from the SPD database of a target physical interface.
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next SPD entry
 *                   from the SPD database of the target physical interface.
 *                   Intended to be called in a loop (to iterate through the database).
 *
 * @warning     The IPsec offload feature is available only for some Premium versions
 *              of PFE firmware. The feature should not be used with a firmware which
 *              does not support it. Failure to adhere to this warning will result
 *              in an undefined behavior of PFE.
 */
#define FPP_CMD_SPD
```

9.15.1 Actions

FPP_ACTION_REGISTER

Create a new SPD entry in the SPD database of a target physical interface.

```
fpp_spd_cmd_t cmd_to_fci =
{
  .action   = FPP_ACTION_REGISTER,  // Action

  .name     = "...",  // Physical interface name.
  .flags    =  ...,   // SPD entry flags. A bitset.
  .position =  ...,   // Entry position. [NBO]
  .saddr    = {...},  // Source IP address. [NBO]
  .daddr    = {...},  // Destination IP address. [NBO]

  .sport    =  ...,   // Source port. [NBO]
                      // Optional (does not have to be set). See '.flags'.

  .dport    =  ...,   // Destination port. [NBO]
                      // Optional (does not have to be set). See '.flags'.

  .protocol =  ...,   // IANA IP Protocol Number (protocol ID).

  .sa_id    =  ...,   // SAD entry identifier for HSE. [NBO]
                      // Used only when '.spd_action' == SPD_ACT_PROCESS_ENCODE).

  .spi      =  ...    // SPI to match in the ingress traffic. [NBO]
                      // Used only when '.spd_action' == SPD_ACT_PROCESS_DECODE).
};

int rtn = 0;
rtn = fci_write(cliet, FPP_CMD_SPD, sizeof(fpp_spd_cmd_t),
                                   (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_DEREGISTER

Remove (destroy) an existing SPD entry.

```
fpp_spd_cmd_t cmd_to_fci =
{
  .action   = FPP_ACTION_DEREGISTER,  // Action
  .name     = "...",  // Physical interface name (see chapter Physical Interface).
  .position =  ...,   // Entry position. [NBO]
};

int rtn = 0;
rtn = fci_write(cliet, FPP_CMD_SPD, sizeof(fpp_spd_cmd_t),
                                   (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of an SPD entry.

```
fpp_spd_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
  .name   = "...",  // Physical interface name (see chapter Physical Interface).
};

fpp_spd_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_SPD,
                sizeof(fpp_spd_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first SPD entry from
//  the SPD database of the target physical interface..

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_SPD,
                sizeof(fpp_spd_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next SPD entry from
//  the SPD database of the target physical interface.
```

9.15.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_IF_ENTRY_NOT_FOUND`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the SPD entry query session (no more SPD entries).
  - For other ACTIONs:
    Unknown (nonexistent) SPD entry was requested.
- `FPP_ERR_FW_FEATURE_NOT_AVAILABLE`
  The feature is not available (not enabled in FW).
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.

- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.15.3 Examples

demo_feature_spd.c

demo_spd.c

9.16 FPP_CMD_QOS_QUEUE

Related topics: Egress QoS
Related data types: `fpp_qos_queue_cmd_t`

```
/**
 * @def         FPP_CMD_QOS_QUEUE
 *
 * @brief       FCI command for management of Egress QoS queues.
 *
 * @details     Related data types: fpp_qos_queue_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of Egress QoS queue.
 *              - FPP_ACTION_QUERY
 *                   Get properties of a target Egress QoS queue.
 */
#define FPP_CMD_QOS_QUEUE
```

9.16.1 Actions

FPP_ACTION_UPDATE

Modify properties of an Egress QoS queue.

```
fpp_qos_queue_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_UPDATE,  // Action
  .if_name = "...",              // Physical interface name.
  .id      =  ...,               // Queue ID.

  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
             // Some properties cannot be modified (see fpp_qos_queue_cmd_t).
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_QOS_QUEUE, sizeof(fpp_qos_queue_cmd_t),
                                           (unsigned short*)(&cmd_to_fci));
```

`FPP_ACTION_QUERY`

Get properties of a target Egress QoS queue.

```
fpp_qos_queue_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_QUERY  // Action
  .if_name = "...",            // Physical interface name.
  .id      =  ...              // Queue ID.
};

fpp_qos_queue_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_QOS_QUEUE,
                sizeof(fpp_qos_queue_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the target Egress QoS queue.
```

9.16.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_QOS_QUEUE_NOT_FOUND`
  Unknown (nonexistent) Egress QoS queue was requested.
- `FPP_ERR_QOS_QUEUE_SUM_OF_LENGTHS_EXCEEDED`
  Sum of all Egress QoS queue lengths for a given physical interface would exceed limits of the interface. First shorten some other queues of the interface, then lengthen the queue of interest.
- `FPP_ERR_WRONG_COMMAND_PARAM`
  Unexpected value of some property.
- `FPP_ERR_IF_NOT_SUPPORTED`
  Requested interface does not support Egress QoS queue management.
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.16.3 Examples

demo_feature_qos.c

demo_qos.c

9.17 FPP_CMD_QOS_SCHEDULER

Related topics: Egress QoS
Related data types: `fpp_qos_scheduler_cmd_t`

```
/**
 * @def         FPP_CMD_QOS_SCHEDULER
 *
 * @brief       FCI command for management of Egress QoS schedulers.
 *
 * @details     Related data types: fpp_qos_scheduler_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of Egress QoS scheduler.
 *              - FPP_ACTION_QUERY
 *                   Get properties of a target Egress QoS scheduler.
 *
 * @note        Modification of QoS Scheduler properties is constrained by certain limitations.
 *              See FCI API Reference, Egress QoS limitations.
 */
#define FPP_CMD_QOS_SCHEDULER
```

9.17.1 Actions

FPP_ACTION_UPDATE

Modify properties of an Egress QoS scheduler.

```
fpp_qos_scheduler_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_UPDATE,  // Action
  .if_name = "...",              // Physical interface name.
  .id      =  ...,               // Scheduler ID.

  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
             // Some properties cannot be modified (see fpp_qos_scheduler_cmd_t).
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_QOS_SCHEDULER, sizeof(fpp_qos_scheduler_cmd_t),
                                              (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY

Get properties of a target Egress QoS scheduler.

```
fpp_qos_scheduler_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_QUERY  // Action
  .if_name = "...",            // Physical interface name.
  .id      =  ...              // Scheduler ID.
};

fpp_qos_scheduler_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_QOS_SCHEDULER,
                sizeof(fpp_qos_scheduler_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the target Egress QoS scheduler.
```

9.17.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_QOS_SCHEDULER_NOT_FOUND`
  Unknown (nonexistent) Egress QoS scheduler was requested.
- `FPP_ERR_WRONG_COMMAND_PARAM`
  Unexpected value of some property.
- `FPP_ERR_IF_NOT_SUPPORTED`
  Requested interface does not support Egress QoS queue management.
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.17.3 Examples

demo_feature_qos.c

demo_qos.c

9.18 FPP_CMD_QOS_SHAPER

Related topics: Egress QoS
Related data types: `fpp_qos_shaper_cmd_t`

```
/**
 * @def         FPP_CMD_QOS_SHAPER
 *
 * @brief       FCI command for management of Egress QoS shapers.
 *
 * @details     Related data types: fpp_qos_shaper_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of Egress QoS shaper.
 *              - FPP_ACTION_QUERY
 *                   Get properties of a target Egress QoS shaper.
 */
#define FPP_CMD_QOS_SHAPER
```

9.18.1 Actions

FPP_ACTION_UPDATE

Modify properties of an Egress QoS shaper.

```
fpp_qos_shaper_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_UPDATE,  // Action
```

```
  .if_name = "...",              // Physical interface name.
  .id      =  ...,               // Shaper ID.

  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
             // Some properties cannot be modified (see fpp_qos_shaper_cmd_t).
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_QOS_SHAPER, sizeof(fpp_qos_shaper_cmd_t),
                                           (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY

Get properties of a target Egress QoS shaper.

```
fpp_qos_shaper_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_QUERY  // Action
  .if_name = "...",            // Physical interface name.
  .id      =  ...,             // Shaper ID.
};

fpp_qos_shaper_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_QOS_SHAPER,
                sizeof(fpp_qos_shaper_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the target Egress QoS shaper.
```

9.18.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_QOS_SHAPER_NOT_FOUND
  Unknown (nonexistent) Egress QoS shaper was requested.
- FPP_ERR_WRONG_COMMAND_PARAM
  Unexpected value of some property.
- FPP_ERR_IF_NOT_SUPPORTED
  Requested interface does not support Egress QoS queue management.
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.18.3 Examples

demo_feature_qos.c

demo_qos.c

9.19 FPP_CMD_QOS_POLICER

Related topics: Ingress QoS
Related data types: fpp_qos_policer_cmd_t

```
/**
 * @def         FPP_CMD_QOS_POLICER
 *
 * @brief       FCI command for Ingress QoS policer enable/disable.
 *
 * @details     Related data types: fpp_qos_policer_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                   Enable/disable Ingress QoS policer of a physical interface.
 *              - FPP_ACTION_QUERY
 *                   Get status of a target Ingress QoS policer.
 *
 * @note        Management of Ingress QoS policer is available only for emac physical interfaces.
 *
 * @note        Effects of enable/disable:
 *              - If an Ingress QoS policer gets disabled, then its associated
 *                flow table, WRED module and shaper module get disabled as well.
 *              - If an Ingress QoS policer gets enabled, then it starts with
 *                default configuration. This means:
 *                  - clear flow table
 *                  - default WRED configuration
 *                  - default shaper configuration
 */
#define FPP_CMD_QOS_POLICER
```

9.19.1 Actions

FPP_ACTION_UPDATE

Enable/disable Ingress QoS policer of an **emac** physical interface.

```
fpp_qos_policer_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_UPDATE,
  .if_name = "...",     // Physical interface name ('emac' interfaces only).
  .enable  =  ...       // 0 == disabled ; 1 == enabled
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_QOS_POLICER, sizeof(fpp_qos_policer_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY

Get status (enabled/disabled) of an Ingress QoS policer.

```
fpp_qos_policer_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_QUERY,
  .if_name = "...",     // Physical interface name ('emac' interfaces only).
};

fpp_qos_policer_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;
```

```
int rtn = 0;
rtn = fci_query(client, FPP_CMD_QOS_POLICER,
                sizeof(fpp_qos_policer_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds the '.enable' field set accordingly.
```

9.19.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_WRONG_COMMAND_PARAM
  Wrong physical interface provided (i.e. non-'emac'), or unexpected value of some property.
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.19.3 Examples

demo_feature_qos_policer.c

demo_qos_pol.c

9.20 FPP_CMD_QOS_POLICER_FLOW

Related topics: Ingress QoS
Related data types: fpp_qos_policer_flow_cmd_t, fpp_iqos_flow_spec_t

```
/**
 * @def         FPP_CMD_QOS_POLICER_FLOW
 *
 * @brief       FCI command for management of Ingress QoS packet flows.
 *
 * @details     Related data types: fpp_qos_policer_flow_cmd_t, fpp_iqos_flow_spec_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                  Add a flow to an Ingress QoS flow classification table.
 *              - FPP_ACTION_DEREGISTER
 *                  Remove a flow from an Ingress QoS flow classification table.
 *              - FPP_ACTION_QUERY
 *                  Initiate (or reinitiate) a flow query session and get properties
 *                  of the first flow from an Ingress QoS flow clasification table.
 *              - FPP_ACTION_QUERY_CONT
 *                  Continue the query session and get properties of the next
 *                  flow from the table. Intended to be called in a loop
 *                  (to iterate through the table).
 *
 * @note        Management of Ingress QoS packet flows is available only
 *              for emac physical interfaces.
 *
 * @note        Management of Ingress QoS packet flows is possible only
```

```
 *              if the associated FPP_CMD_QOS_POLICER is enabled.
 */
#define FPP_CMD_QOS_POLICER_FLOW
```

9.20.1 Actions

FPP_ACTION_REGISTER

Add a packet flow to an Ingress QoS flow classification table. Specify flow parameters and the action to be done for packets which conform to the given flow.

```
fpp_qos_policer_flow_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_REGISTER,
  .if_name = "...",  // Physical interface name ('emac' interfaces only).
  .id      =  ...,   // Position in the classification table. 0xFF == automatic placement.

  .flow    = {...}   // Flow specification structure.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_QOS_POLICER_FLOW, sizeof(fpp_qos_policer_flow_cmd_t),
                                                 (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_DEREGISTER

Remove a flow from an Ingress QoS flow classification table.

```
fpp_qos_policer_flow_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_DEREGISTER,
  .if_name = "...",  // Physical interface name ('emac' interfaces only).
  .id      =  ...,   // Position in the classification table.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_QOS_POLICER_FLOW, sizeof(fpp_qos_policer_flow_cmd_t),
                                                 (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of the Ingress QoS flow.

```
fpp_qos_policer_flow_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_QUERY,
  .if_name = "...",  // Physical interface name ('emac' interfaces only).
  .id      =  ...,   // Entry position in the table, from 0 to "table size - 1".
};

fpp_qos_policer_flow_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_QOS_POLICER_FLOW,
                sizeof(fpp_qos_policer_flow_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds the content of the first available (i.e. active) flow
//  from the Ingress QoS flow classification table of the target physical interface.
```

```
cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_QOS_POLICER_FLOW,
                sizeof(fpp_qos_policer_flow_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next available flow from the table.
```

9.20.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_QOS_POLICER_FLOW_NOT_FOUND`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the Ingress QoS flow query session (no more flows).
  - For other ACTIONs:
    Unknown (nonexistent) Ingress QoS flow was requested.
- `FPP_ERR_QOS_POLICER_FLOW_TABLE_FULL`
  Attempting to register flow with `.id >= FPP_IQOS_FLOW_TABLE_SIZE` or flow table full.
- `FPP_ERR_WRONG_COMMAND_PARAM`
  Wrong physical interface provided (i.e. non-'emac'), or unexpected value of some property.
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.20.3 Examples

demo_feature_qos_policer.c

demo_qos_pol.c

9.21 FPP_CMD_QOS_POLICER_WRED

Related topics: Ingress QoS
Related data types: `fpp_qos_policer_wred_cmd_t`

```
/**
 * @def         FPP_CMD_QOS_POLICER_WRED
 *
 * @brief       FCI command for management of Ingress QoS WRED queues.
 *
 * @details     Related data types: fpp_qos_policer_wred_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of Ingress QoS WRED queue.
 *              - FPP_ACTION_QUERY
 *                   Get properties of a target Ingress QoS WRED queue.
```

```
 *
 * @note        Management of Ingress QoS packet flows is available only
 *              for emac physical interfaces.
 *
 * @note        Management of Ingress QoS packet flows is possible only
 *              if the associated FPP_CMD_QOS_POLICER is enabled.
 */
#define FPP_CMD_QOS_POLICER_WRED
```

9.21.1 Actions

FPP_ACTION_UPDATE

Update Ingress QoS WRED queue of a target physical interface.

```
fpp_qos_policer_wred_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_UPDATE,
  .if_name = "...",      // Physical interface name ('emac' interfaces only).
  .queue   =  ...,       // Target Ingress QoS WRED queue (DMEM, LMEM, RXF).
  .enable  =  ...,       // Enable/disable switch (0 == disabled, 1 == enabled).

  .thr[]   = {...},      // Min/max/full WRED thresholds.
                         // 0xFFFF == let HW keep its currently configured thld value.

  .zprob[] = {...}       // WRED drop probabilities for all zones in 1/16 increments.
                         // 0xFF == let HW keep its currently configured zone value.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_QOS_POLICER_WRED, sizeof(fpp_qos_policer_wred_cmd_t),
                                                 (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY

Get properties of a target Ingress QoS WRED queue.

```
fpp_qos_policer_wred_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_QUERY,
  .if_name = "...",      // Physical interface name ('emac' interfaces only).
  .queue   =  ...,       // Target Ingress QoS WRED queue (DMEM, LMEM, RXF).
};

fpp_qos_policer_wred_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_QOS_POLICER_WRED,
                sizeof(fpp_qos_policer_wred_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the target Ingress QoS WRED queue.
```

9.21.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_WRONG_COMMAND_PARAM`

Wrong physical interface provided (i.e. non-'emac'), or unexpected value of some property.

- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.21.3 Examples

demo_feature_qos_policer.c

demo_qos_pol.c

9.22 FPP_CMD_QOS_POLICER_SHP

Related topics: Ingress QoS
Related data types: `fpp_qos_policer_shp_cmd_t`

```
/**
 * @def         FPP_CMD_QOS_POLICER_SHP
 *
 * @brief       FCI command for management of Ingress QoS shapers.
 *
 * @details     Related data types: fpp_qos_policer_shp_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of Ingress QoS shaper.
 *              - FPP_ACTION_QUERY
 *                   Get properties of a target Ingress QoS shaper.
 *
 * @note        Management of Ingress QoS packet flows is available only
 *              for emac physical interfaces.
 *
 * @note        Management of Ingress QoS packet flows is possible only
 *              if the associated FPP_CMD_QOS_POLICER is enabled.
 */
#define FPP_CMD_QOS_POLICER_SHP
```

9.22.1 Actions

`FPP_ACTION_UPDATE`

Configure Ingress QoS credit based shaper (IEEE 802.1Q) of a target physical interface.

```
fpp_qos_policer_shp_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_UPDATE,
  .if_name = "...",  // Physical interface name ('emac' interfaces only).
  .id      =  ...,   // ID of the target Ingress QoS shaper.

  ... = ...  // Properties (data fields) to be updated, and their new (modified) values.
             // Some properties cannot be modified (see fpp_qos_policer_shp_cmd_t).
};
```



```
int rtn = 0;
rtn = fci_write(client, FPP_CMD_QOS_POLICER_SHP, sizeof(fpp_qos_policer_shp_cmd_t),
                                                (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY

Get properties of a target Ingress QoS shaper.

```
fpp_qos_policer_shp_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_QUERY,
  .if_name = "...",  // Physical interface name ('emac' interfaces only).
  .id      =  ...    // ID of the target Ingress QoS shaper.
};

fpp_qos_policer_shp_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_QOS_POLICER_SHP,
                sizeof(fpp_qos_policer_shp_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the target Ingress QoS shaper.
```

9.22.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_WRONG_COMMAND_PARAM
  Wrong physical interface provided (i.e. non-'emac'), or unexpected value of some property.
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.22.3 Examples

demo_feature_qos_policer.c

demo_qos_pol.c

9.23 FPP_CMD_FW_FEATURE

Related topics: FW features
Related data types: fpp_fw_features_cmd_t

```
/**
 * @def          FPP_CMD_FW_FEATURE
```

```
 *
 * @brief       FCI command for management of configurable FW features.
 *
 * @details     Related data types: fpp_fw_features_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                   Enable/disable a FW feature.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) a FW feature query session and get properties
 *                   of the first FW feature from the internal list of FW features.
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next FW feature
 *                   from the list. Intended to be called in a loop (to iterate through the list).
 */
#define FPP_CMD_FW_FEATURE
```

9.23.1 Actions

`FPP_ACTION_UPDATE`

Enable/disable a FW feature.

```
fpp_fw_features_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UPDATE,  // Action
  .val    = ...                 // 0 == disabled ; 1 == enabled
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_FW_FEATURE, sizeof(fpp_fw_features_cmd_t),
                                            (unsigned short*)(&cmd_to_fci));
```

`FPP_ACTION_QUERY` and `FPP_ACTION_QUERY_CONT`

Get properties of a FW feature.

```
fpp_fw_features_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
};

fpp_fw_features_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_FW_FEATURE,
                sizeof(fpp_fw_features_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first FW feature from
//  the internal list of FW features.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_FW_FEATURE,
                sizeof(fpp_fw_features_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next FW feature from
//  the internal list of FW features.
```

9.23.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_FW_FEATURE_NOT_FOUND
  - For FPP_ACTION_QUERY or FPP_ACTION_QUERY_CONT:
    The end of the FW feature query session (no more FW features).
  - For other ACTIONs:
    Unknown (nonexistent) FW feature was requested.
- FPP_ERR_FW_FEATURE_NOT_AVAILABLE
  Requested FW feature exists, but is not available.
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.23.3 Examples

demo_fwfeat.c

9.24 FPP_CMD_FW_FEATURE_ELEMENT

Related topics: FW features
Related data types: fpp_fw_features_element_cmd_t

```
/**
 * @def         FPP_CMD_FW_FEATURE_ELEMENT
 *
 * @brief       FCI command for management of particular FW feature elements.
 *
 * @details     Related data types: fpp_fw_features_element_cmd_t
 *
 * @details     Extension data for FW features. Highly feature-specific.
 *              Each FW feature element contains some data.
 *              This data may be one value or an array of values.
 *              Element data may be editable, or it may be read-only.
 *              This all depends on the particular element and its purpose.
 *              For details, see appropriate documentation of FW feature elements.
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                   Set new data for a FW feature element.
 *              - FPP_ACTION_QUERY
 *                   - If .element_name[] empty:
 *                     Initiate (or reinitiate) a FW feature query session and get properties
 *                     of the first FW feature element from the internal list of FW features.
 *                   - If .element_name[] NOT empty:
 *                     Get properties of a target FW feature element.
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next FW feature element
 *                   from the list. Intended to be called in a loop (to iterate through the list).
 */
#define FPP_CMD_FW_FEATURE_ELEMENT
```

9.24.1 Actions

FPP_ACTION_UPDATE

Set new data for a FW feature element.

```
fpp_fw_features_element_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UPDATE,  // Action

  .fw_feature_name = ...,        // Name of a parent fpp_fw_features_cmd_t.
  .element_name    = ...,        // Name of the target FW feature element.
  .group           = ...,        // Fill correct group of the target FW feature element.

  .unit_size = 1/2/4,    // Fill correct unit size of target element's data items.
  .index     = ...,      // Index where to start in element's data array (min=1).
  .count     = ...,      // How many items in element's data array to update.
  .payload   = ...       // Data items, stored as bytestream.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_FW_FEATURE_ELEMENT, sizeof(fpp_fw_features_element_cmd_t),
                                           (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of a FW feature element.

Use this query type to list all available elements of a particular FW feature.

```
fpp_fw_features_element_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY,  // Action
  .fw_feature_name = ...,      // Name of a parent fpp_fw_features_cmd_t.
  .element_name    = "",       // Leave empty.
  .group  = 0/1/2              // Select element group to query or '0' to query all groups
};

fpp_fw_features_element_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_FW_FEATURE_ELEMENT,
                sizeof(fpp_fw_features_element_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first FW feature element from
//  the internal list of target FW feature's elements.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_FW_FEATURE_ELEMENT,
                sizeof(fpp_fw_features_element_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next FW feature element from
//  the internal list of target FW feature's elements.
```

FPP_ACTION_QUERY

Get properties of a FW feature element.

Use this query type to get directly the target element of a particular FW feature.

```
fpp_fw_features_element_cmd_t cmd_to_fci =
{
```

```
  .action = FPP_ACTION_QUERY,  // Action
  .fw_feature_name = ...,      // Name of a parent fpp_fw_features_cmd_t.
  .element_name    = ...,      // Name of the target FW feature element.

  .group  = 0/1/2              // Select element group to query or '0' to query all groups
                               // The first matching element is returned.
};

fpp_fw_features_element_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_FW_FEATURE_ELEMENT,
                sizeof(fpp_fw_features_element_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the target FW feature element.
```

9.24.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_FW_FEATURE_ELEMENT_NOT_FOUND`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the FW feature element query session (no more FW feature elements).
  - For other ACTIONs:
    Unknown (nonexistent) FW feature element was requested.
- `FPP_ERR_FW_FEATURE_ELEMENT_READ_ONLY`
  Update of read-only FW feature element was requested.
- `FPP_ERR_FW_FEATURE_NOT_FOUND`
  Unknown (nonexistent) parent FW feature was requested.
- `FPP_ERR_FW_FEATURE_NOT_AVAILABLE`
  Requested parent FW feature exists, but is not available.
- `FPP_ERR_WRONG_COMMAND_PARAM`
  Unexpected value of some property.
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.24.3 Examples

demo_fwfeat.c

9.25 FPP_CMD_FCI_OWNERSHIP_LOCK

Related topics: FCI ownership in Master-Slave setup, `FPP_CMD_FCI_OWNERSHIP_UNLOCK`
Related data types: ---

```
/**
```

```
 * @def         FPP_CMD_FCI_OWNERSHIP_LOCK
 *
 * @brief       FCI command to get FCI ownership.
 *
 * @details     Related data types: ---
 *
 * @details     Supported `.action` values: ---
 */
#define FPP_CMD_FCI_OWNERSHIP_LOCK
```

9.25.1 Actions

This command is used "as is", without any specific ACTION.

```
int rtn = 0;
rtn = fci_write(client, FPP_CMD_FCI_OWNERSHIP_LOCK, 0, NULL);
```

9.25.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized get FCI ownership.
- FPP_ERR_FCI_OWNERSHIP_ALREADY_LOCKED
  The FCI ownership is already held by other client.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.25.3 Examples

demo_fci_owner.c

9.26 FPP_CMD_FCI_OWNERSHIP_UNLOCK

Related topics: FCI ownership in Master-Slave setup, FPP_CMD_FCI_OWNERSHIP_LOCK
Related data types: ---

```
/**
 * @def         FPP_CMD_FCI_OWNERSHIP_UNLOCK
 *
 * @brief       FCI command to release FCI ownership.
 *
 * @details     Related data types: ---
 *
 * @details     Supported `.action` values: ---
 */
#define FPP_CMD_FCI_OWNERSHIP_UNLOCK
```

9.26.1 Actions

This command is used "as is", without any specific ACTION.

```
int rtn = 0;
rtn = fci_write(client, FPP_CMD_FCI_OWNERSHIP_UNLOCK, 0, NULL);
```

9.26.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.26.3 Examples

demo_fci_owner.c

9.27 FPP_CMD_IPV4_CONNTRACK

Related topics: IPv4/IPv6 Router, L2L3 Bridge, FPP_CTCMD
Related data types: `fpp_ct_cmd_t`

```
/**
 * @def         FPP_CMD_IPV4_CONNTRACK
 *
 * @brief       FCI command for management of IPv4 conntracks.
 *
 * @details     Related data types: fpp_ct_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Create a new IPv4 conntrack and bind it to previously created route(s).
 *              - FPP_ACTION_DEREGISTER
 *                   Remove (destroy) an existing IPv4 conntrack.
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of IPv4 conntrack.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) IPv4 conntrack query session and get properties
 *                   of the first IPv4 conntrack from the internal list of IPv4 conntracks.
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next IPv4 conntrack
 *                   from the list. Intended to be called in a loop (to iterate through the list).
 */
#define FPP_CMD_IPV4_CONNTRACK
```

9.27.1 Conntrack matching modes (5-tuple, 3-tuple)

By default, PFE uses **5-tuple** matching mode when searching for a suitable conntrack. This means 5 significant data elements must match between a conntrack and an ingress IP packet:

- `.protocol` protocol ID
- `.saddr` source IP address
- `.daddr` destination IP address
- `.sport` source port
- `.dport` destination port

For individual non-TCP/non-UDP conntracks[5], it is possible to configure **3-tuple** matching mode.

To configure a conntrack in 3-tuple matching mode, leave `.sport` and `.dport` as zero (0) when creating the conntrack. This allows matching based only on protocol ID + source IP address + destination IP address. This matching mode is intended for protocols that disregard ports (e.g. ICMP).

9.27.2 Relationship between orig and reply direction

By default, the connection is created as bi-directional. It means that for a normal `FPP_CMD_IPV4_CONNTRACK` command, two routing table entries are created in PFE:

- One entry for a standard flow ('**orig**' direction).
  Parameters of this entry are defined by the following `fpp_ct_cmd_t` members:
  `.protocol`, `.saddr`, `.daddr`, `.sport` and `.dport`
- One entry for a reverse flow ('**reply**' direction).
  Parameters of this entry are defined by the following `fpp_ct_cmd_t` members:
  `.protocol`, `.saddr_reply`, `.daddr_reply`, `.sport_reply` and `.dport_reply`

To create an uni-directional connection (only one routing table entry for given `FPP_CMD_IPV4_CONNTRACK` command), set one[6] of these flags when configuring a conntrack:

- To create '**orig**' direction only: Set `.flags |= CTCMD_FLAGS_REP_DISABLED` and don't set `.route_id_reply` (leave it zero).
- To create '**reply**' direction only: Set `.flags |= CTCMD_FLAGS_ORIG_DISABLED` and don't set `.route_id` (leave it zero).

9.27.3 NAT, PAT and NAPT

To configure NAT, PAT or NAPT connection, set 'reply' IP addresses and ports to different values than 'orig' IP addresses and ports.

1. `.daddr_reply != .saddr`
   Source IP address of packets in the '**orig**' direction will be changed from its original `.saddr` value to `.daddr_reply` value. In case of a bi-directional connection, destination IP address of packets in the '**reply**' direction will be conversely changed from `.daddr_reply` to `.saddr`.

5 3-tuple matching mode is **NOT** available for TCP/UDP conntracks due to speed optimizations.
6 Do **NOT** set both flags in one `FPP_CMD_IPV4_CONNTRACK` command. Doing so results in undefined behavior of PFE.

2. `.saddr_reply != .daddr`
   Destination IP address of packets in the '**orig**' direction will be changed from its original `.daddr` value to `.saddr_reply` value. In case of a bi-directional connection, source IP address of packets in the '**reply**' direction will be conversely changed from `.saddr_reply` to `.daddr`.

3. `.dport_reply != .sport`
   Destination port of packets in the '**orig**' direction will be changed from its original `.sport` value to `.dport_reply` value. In case of a bi-directional connection, source port of packets in the '**reply**' direction will be conversely changed from `.dport_reply` to `.sport`.

4. `.sport_reply != .dport`
   Destination port of packets in the '**orig**' direction will be changed from its original `.dport` value to `.sport_reply` value. In case of a bi-directional connection, source port of packets in the '**reply**' direction will be conversely changed from `.sport_reply` to `.dport`.

9.27.4 Actions

FPP_ACTION_REGISTER

Create a new IPv4 conntrack.

See Conntrack matching modes (5-tuple, 3-tuple).

See Relationship between orig and reply direction.

See NAT, PAT and NAPT.

```
fpp_ct_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_REGISTER,  // Action

  .saddr = ...,         // 'orig' direction: Source IP address. [NBO]
  .daddr = ...,         // 'orig' direction: Destination IP address. [NBO]
  .sport = ...,         // 'orig' direction: Source port. [NBO]
  .dport = ...,         // 'orig' direction: Destination port. [NBO]

  .saddr_reply = ...,   // 'reply' direction: Source IP address. [NBO]
                        // Used for NAT, otherwise equals '.daddr'.

  .daddr_reply = ...,   // 'reply' direction: Destination IP address.
                        // Used for NAT, otherwise equals '.saddr'.

  .sport_reply = ...,   // 'reply' direction: Source port. [NBO]
                        // Used for NAT, otherwise equals '.dport'.

  .dport_reply = ...,   // 'reply' direction: Destination port. [NBO]
                        // Used for NAT, otherwise equals '.sport'.

  .protocol = ...,      // IANA IP Protocol Number (protocol ID). [NBO]

  .flags = ...,         // Flags. A bitset. [NBO]

  .route_id = ...,      // 'orig' direction: ID of an associated route. [NBO]
                        // See FPP_CMD_IP_ROUTE.

  .route_id_reply = ...,   // 'reply' direction: ID of an associated route. [NBO]
                           // See FPP_CMD_IP_ROUTE.

  .vlan = ...,          // 'orig' direction: VLAN tag. [NBO]
                        // If non-zero, then this VLAN tag is added to the routed packet.
                        // If the packet already has a VLAN tag, then its tag is replaced.

  .vlan_reply = ...     // 'reply' direction: VLAN tag. [NBO]
                        // If non-zero, then this VLAN tag is added to the routed packet.
```

```
                        // If the packet already has a VLAN tag, then its tag is replaced.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_IPV4_CONNTRACK, sizeof(fpp_ct_cmd_t),
                                               (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_DEREGISTER

Remove (destroy) an existing IPv4 conntrack.

'Orig' properties are mandatory for this action. 'Reply' properties are optional.

```
fpp_ct_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_DEREGISTER,  // Action

  // Identification of the target conntrack.
  .saddr = ...,         // 'orig' direction: Source IP address. [NBO]
  .daddr = ...,         // 'orig' direction: Destination IP address. [NBO]
  .sport = ...,         // 'orig' direction: Source port. [NBO]
  .dport = ...          // 'orig' direction: Destination port. [NBO]
  .protocol = ...,      // IANA IP Protocol Number (protocol ID). [NBO]

  .saddr_reply = ...,   // 'reply' direction: Source IP address. [NBO]
                        // Used for NAT, otherwise equals '.daddr'.

  .daddr_reply = ...,   // 'reply' direction: Destination IP address.
                        // Used for NAT, otherwise equals '.saddr'.

  .sport_reply = ...,   // 'reply' direction: Source port. [NBO]
                        // Used for NAT, otherwise equals '.dport'.

  .dport_reply = ...,   // 'reply' direction: Destination port. [NBO]
                        // Used for NAT, otherwise equals '.sport'.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_IPV4_CONNTRACK, sizeof(fpp_ct_cmd_t),
                                               (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_UPDATE

Modify properties of an IPv4 conntrack.

```
fpp_ct_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UPDATE,  // Action

  // Identification of the target conntrack.
  .saddr = ...,      // 'orig' direction: Source IP address. [NBO]
  .daddr = ...,      // 'orig' direction: Destination IP address. [NBO]
  .sport = ...,      // 'orig' direction: Source port. [NBO]
  .dport = ...,      // 'orig' direction: Destination port. [NBO]
  .protocol = ...,   // IANA IP Protocol Number (protocol ID). [NBO]

  // Modification of the target conntrack.
  .flags |= ntohs(CTCMD_FLAGS_TTL_DECREMENT)  // The only modification available:
                                              // set/unset TTL decrement flag.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_IPV4_CONNTRACK, sizeof(fpp_ct_cmd_t),
                                               (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of an IPv4 conntrack.

```
fpp_ct_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
};

fpp_ct_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_IPV4_CONNTRACK,
                sizeof(fpp_ct_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first IPv4 conntrack from
//  the internal list of IPv4 conntracks.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_IPV4_CONNTRACK,
                sizeof(fpp_ct_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next IPv4 conntrack from
//  the internal list of IPv4 conntracks.
```

9.27.5 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_CT_ENTRY_NOT_FOUND
  - For FPP_ACTION_QUERY or FPP_ACTION_QUERY_CONT:
    The end of the IPv4 conntrack query session (no more IPv4 conntracks).
  - For other ACTIONs:
    Unknown (nonexistent) IPv4 conntrack was requested.
- FPP_ERR_CT_ENTRY_ALREADY_REGISTERED
  Requested IPv4 conntrack already exists (is already registered).
- FPP_ERR_WRONG_COMMAND_PARAM
  Unexpected value of some property (probably nonexistent route).
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.27.6 Examples

demo_feature_router_simple.c

demo_feature_router_nat.c

demo_feature_L2L3_bridge_vlan.c

demo_rt_ct.c

9.28 FPP_CMD_IPV6_CONNTRACK

Related topics: IPv4/IPv6 Router, L2L3 Bridge, FPP_CTCMD
Related data types: `fpp_ct6_cmd_t`

```
/**
 * @def         FPP_CMD_IPV6_CONNTRACK
 *
 * @brief       FCI command for management of IPv6 conntracks.
 *
 * @details     Related data types: fpp_ct6_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Create a new IPv6 conntrack and bind it to previously created route(s).
 *              - FPP_ACTION_DEREGISTER
 *                   Remove (destroy) an existing IPv6 conntrack.
 *              - FPP_ACTION_UPDATE
 *                   Modify properties of IPv6 conntrack.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) IPv6 conntrack query session and get properties
 *                   of the first IPv6 conntrack from the internal list of IPv6 conntracks.
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next IPv6 conntrack
 *                   from the list. Intended to be called in a loop (to iterate through the list).
 */
#define FPP_CMD_IPV6_CONNTRACK
```

9.28.1 Conntrack matching modes (5-tuple, 3-tuple)

By default, PFE uses **5-tuple** matching mode when searching for a suitable conntrack. This means 5 significant data elements must match between a conntrack and an ingress IP packet:

- `.protocol` protocol ID
- `.saddr` source IP address
- `.daddr` destination IP address
- `.sport` source port
- `.dport` destination port

For individual non-TCP/non-UDP conntracks[7], it is possible to configure **3-tuple** matching mode.

To configure a conntrack in 3-tuple matching mode, leave `.sport` and `.dport` as zero (0) when creating the conntrack. This allows matching based only on protocol ID + source IP address + destination IP address. This matching mode is intended for protocols that disregard ports (e.g. ICMP).

9.28.2 Relationship between orig and reply direction

By default, the connection is created as bi-directional. It means that for a normal `FPP_CMD_IPV6_CONNTRACK` command, two routing table entries are created in PFE:

- One entry for a standard flow ('**orig**' direction).
  Parameters of this entry are defined by the following `fpp_ct6_cmd_t` members:

7 3-tuple matching mode is **NOT** available for TCP/UDP conntracks due to speed optimizations.

`.protocol`, `.saddr`, `.daddr`, `.sport` and `.dport`

- One entry for a reverse flow ('**reply**' direction).
  Parameters of this entry are defined by the following [fpp_ct6_cmd_t](#) members:
  `.protocol`, `.saddr_reply`, `.daddr_reply`, `.sport_reply` and `.dport_reply`

To create an uni-directional connection (only one routing table entry for given `FPP_CMD_IPV6_CONNTRACK` command), set one[8] of these flags when configuring a conntrack:

- To create '**orig**' direction only: Set `.flags |= CTCMD_FLAGS_REP_DISABLED` and don't set `.route_id_reply` (leave it zero).
- To create '**reply**' direction only: Set `.flags |= CTCMD_FLAGS_ORIG_DISABLED` and don't set `.route_id` (leave it zero).

9.28.3 NAT, PAT and NAPT

To configure NAT, PAT or NAPT connection, set 'reply' IP addresses and ports to different values than 'orig' IP addresses and ports.

1. `.daddr_reply != .saddr`
   Source IP address of packets in the '**orig**' direction will be changed from its original `.saddr` value to `.daddr_reply` value. In case of a bi-directional connection, destination IP address of packets in the '**reply**' direction will be conversely changed from `.daddr_reply` to `.saddr`.

2. `.saddr_reply != .daddr`
   Destination IP address of packets in the '**orig**' direction will be changed from its original `.daddr` value to `.saddr_reply` value. In case of a bi-directional connection, source IP address of packets in the '**reply**' direction will be conversely changed from `.saddr_reply` to `.daddr`.

3. `.dport_reply != .sport`
   Destination port of packets in the '**orig**' direction will be changed from its original `.sport` value to `.dport_reply` value. In case of a bi-directional connection, source port of packets in the '**reply**' direction will be conversely changed from `.dport_reply` to `.sport`.

4. `.sport_reply != .dport`
   Destination port of packets in the '**orig**' direction will be changed from its original `.dport` value to `.sport_reply` value. In case of a bi-directional connection, source port of packets in the '**reply**' direction will be conversely changed from `.sport_reply` to `.dport`.

9.28.4 Actions

`FPP_ACTION_REGISTER`

Create a new IPv6 conntrack.

See [Conntrack matching modes (5-tuple, 3-tuple)](#).

8 Do **NOT** set both flags in one `FPP_CMD_IPV6_CONNTRACK` command. Doing so results in undefined behavior of PFE.

See Relationship between orig and reply direction.

See NAT, PAT and NAPT.

```
fpp_ct6_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_REGISTER,  // Action

  .saddr = {...},         // 'orig' direction: Source IP address. [NBO]
  .daddr = {...},         // 'orig' direction: Destination IP address. [NBO]
  .sport =  ...,          // 'orig' direction: Source port. [NBO]
  .dport =  ...,          // 'orig' direction: Destination port. [NBO]

  .saddr_reply = {...},   // 'reply' direction: Source IP address. [NBO]
                          // Used for NAT, otherwise equals '.daddr'.

  .daddr_reply = {...},   // 'reply' direction: Destination IP address.
                          // Used for NAT, otherwise equals '.saddr'.

  .sport_reply = ...,     // 'reply' direction: Source port. [NBO]
                          // Used for NAT, otherwise equals '.dport'.

  .dport_reply = ...,     // 'reply' direction: Destination port. [NBO]
                          // Used for NAT, otherwise equals '.sport'.

  .protocol = ...,        // IANA IP Protocol Number (protocol ID). [NBO]

  .flags = ...,           // Flags. A bitset. [NBO]

  .route_id = ...,        // 'orig' direction: ID of an associated route. [NBO]
                          // See FPP_CMD_IP_ROUTE.

  .route_id_reply = ...,  // 'reply' direction: ID of an associated route. [NBO]
                          // See FPP_CMD_IP_ROUTE.

  .vlan = ...,         // 'orig' direction: VLAN tag. [NBO]
                       // If non-zero, then this VLAN tag is added to the routed packet.
                       // If the packet already has a VLAN tag, then its tag is replaced.

  .vlan_reply = ...    // 'reply' direction: VLAN tag. [NBO]
                       // If non-zero, then this VLAN tag is added to the routed packet.
                       // If the packet already has a VLAN tag, then its tag is replaced.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_IPV6_CONNTRACK, sizeof(fpp_ct6_cmd_t),
                                               (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_DEREGISTER

Remove (destroy) an existing IPv6 conntrack.

'Orig' properties are mandatory for this action. 'Reply' properties are optional.

```
fpp_ct6_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_DEREGISTER,  // Action

  // Identification of the target conntrack.
  .saddr = {...},        // 'orig' direction: Source IP address. [NBO]
  .daddr = {...},        // 'orig' direction: Destination IP address. [NBO]
  .sport =  ...,         // 'orig' direction: Source port. [NBO]
  .dport =  ...          // 'orig' direction: Destination port. [NBO]
  .protocol = ...,       // IANA IP Protocol Number (protocol ID). [NBO]

  .saddr_reply = {...},  // 'reply' direction: Source IP address. [NBO]
                         // Used for NAT, otherwise equals '.daddr'.

  .daddr_reply = {...},  // 'reply' direction: Destination IP address.
                         // Used for NAT, otherwise equals '.saddr'.

  .sport_reply = ...,    // 'reply' direction: Source port. [NBO]
                         // Used for NAT, otherwise equals '.dport'.

  .dport_reply = ...,    // 'reply' direction: Destination port. [NBO]
```

```
                         // Used for NAT, otherwise equals '.sport'.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_IPV6_CONNTRACK, sizeof(fpp_ct6_cmd_t),
                                               (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_UPDATE

Modify properties of an IPv6 conntrack.

```
fpp_ct6_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UPDATE,  // Action

  // Identification of the target conntrack.
  .saddr = {...},   // 'orig' direction: Source IP address. [NBO]
  .daddr = {...},   // 'orig' direction: Destination IP address. [NBO]
  .sport =  ...,    // 'orig' direction: Source port. [NBO]
  .dport =  ...,    // 'orig' direction: Destination port. [NBO]
  .protocol = ...,  // IANA IP Protocol Number (protocol ID). [NBO]

  // Modification of the target conntrack.
  .flags |= ntohs(CTCMD_FLAGS_TTL_DECREMENT)  // The only modification available:
                                              // set/unset TTL decrement flag.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_IPV6_CONNTRACK, sizeof(fpp_ct6_cmd_t),
                                               (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of an IPv6 conntrack.

```
fpp_ct6_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
};

fpp_ct6_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_IPV6_CONNTRACK,
                sizeof(fpp_ct6_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first IPv6 conntrack from
//  the internal list of IPv6 conntracks.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_IPV6_CONNTRACK,
                sizeof(fpp_ct6_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next IPv6 conntrack from
//  the internal list of IPv6 conntracks.
```

9.28.5 Return values

- FPP_ERR_OK

Success

- `FPP_ERR_CT_ENTRY_NOT_FOUND`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the IPv6 conntrack query session (no more IPv6 conntracks).
  - For other ACTIONs:
    Unknown (nonexistent) IPv6 conntrack was requested.
- `FPP_ERR_CT_ENTRY_ALREADY_REGISTERED`
  Requested IPv6 conntrack already exists (is already registered).
- `FPP_ERR_WRONG_COMMAND_PARAM`
  Unexpected value of some property (probably nonexistent route).
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.28.6 Examples

demo_feature_router_simple.c

demo_feature_router_nat.c

demo_feature_L2L3_bridge_vlan.c

demo_rt_ct.c

9.29 FPP_CMD_IP_ROUTE

Related topics: IPv4/IPv6 Router, L2L3 Bridge
Related data types: `fpp_rt_cmd_t`

```
/**
 * @def         FPP_CMD_IP_ROUTE
 *
 * @brief       FCI command for management of IP routes.
 *
 * @details     Related data types: fpp_rt_cmd_t
 *
 * @details     In the context of PFE, a route represents a way to reach an external network node.
 *              It is utilized for IP Router packet forwarding. It holds the following information:
 *              - Egress physical interface (PFE interface to reach the external network node).
 *              - MAC address of the external network node.
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_REGISTER
 *                   Create a new route.
 *              - FPP_ACTION_DEREGISTER
 *                   Remove (destroy) an existing route.
 *              - FPP_ACTION_QUERY
 *                   Initiate (or reinitiate) a route query session and get properties
 *                   of the first route from the internal collective list of all routes
 *                   (regardless of IP type nor conntrack affiliation).
 *              - FPP_ACTION_QUERY_CONT
 *                   Continue the query session and get properties of the next route
 *                   from the list. Intended to be called in a loop (to iterate through the list).
 */
```

```
#define FPP_CMD_IP_ROUTE
```

9.29.1 Actions

FPP_ACTION_REGISTER

Create a new route. For detailed info about route properties, see fpp_rt_cmd_t.

```
fpp_rt_cmd_t cmd_to_fci =
{
  .action  = FPP_ACTION_REGISTER,  // Action
  .src_mac = ...,                  // Source MAC address.
  .dst_mac = ...,                  // Destination MAC address.
  .output_device = ...,            // Name of the egress physical interface.
  .id    = ...,                    // Route ID. [NBO]. User-defined.
  .flags = ...                     // Flags. [NBO]. 1 for IPv4 routes, 2 for IPv6 routes.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_IP_ROUTE, sizeof(fpp_rt_cmd_t),
                                         (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_DEREGISTER

Remove (destroy) an existing route.

```
fpp_rt_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_DEREGISTER,  // Action
  .id     = ...                     // Route ID. [NBO]. User-defined.
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_IP_ROUTE, sizeof(fpp_rt_cmd_t),
                                         (unsigned short*)(&cmd_to_fci));
```

FPP_ACTION_QUERY and FPP_ACTION_QUERY_CONT

Get properties of a route.

```
fpp_rt_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_QUERY  // Action
};

fpp_rt_cmd_t reply_from_fci = {0};
unsigned short reply_length = 0u;

int rtn = 0;
rtn = fci_query(client, FPP_CMD_IP_ROUTE,
                sizeof(fpp_rt_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the first route from
//  the internal collective list of all routes.

cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
rtn = fci_query(client, FPP_CMD_IP_ROUTE,
                sizeof(fpp_rt_cmd_t), (unsigned short*)(&cmd_to_fci),
                &reply_length, (unsigned short*)(&reply_from_fci));

// 'reply_from_fci' now holds properties of the next route from
```

```
//  the internal collective list of all routes.
```

9.29.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_RT_ENTRY_NOT_FOUND`
  - For `FPP_ACTION_QUERY` or `FPP_ACTION_QUERY_CONT`:
    The end of the route query session (no more routes).
  - For other ACTIONs:
    Unknown (nonexistent) route was requested.
- `FPP_ERR_RT_ENTRY_ALREADY_REGISTERED`
  Requested route already exists (is already registered).
- `FPP_ERR_WRONG_COMMAND_PARAM`
  Unexpected value of some property.
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.29.3 Examples

demo_feature_router_simple.c

demo_feature_router_nat.c

demo_feature_L2L3_bridge_vlan.c

demo_rt_ct.c

9.30 FPP_CMD_IPV4_RESET

Related topics:	IPv4/IPv6 Router, L2L3 Bridge
Related data types:	---

```
/**
 * @def          FPP_CMD_IPV4_RESET
 *
 * @brief        FCI command to remove all IPv4 routes and conntracks.
 *
 * @details      Related data types: ---
 *
 * @details      Supported `.action` values: ---
 */
#define FPP_CMD_IPV4_RESET
```

9.30.1 Actions

This command is used "as is", without any specific ACTION.

```
int rtn = 0;
rtn = fci_write(client, FPP_CMD_IPV4_RESET, 0, NULL);
```

9.30.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.30.3 Examples

demo_feature_router_simple.c

demo_feature_router_nat.c

demo_feature_L2L3_bridge_vlan.c

demo_rt_ct.c

9.31 FPP_CMD_IPV6_RESET

Related topics: IPv4/IPv6 Router, L2L3 Bridge
Related data types: ---

```
/**
 * @def         FPP_CMD_IPV6_RESET
 *
 * @brief       FCI command to remove all IPv6 routes and conntracks.
 *
 * @details     Related data types: ---
 *
 * @details     Supported `.action` values: ---
 */
#define FPP_CMD_IPV6_RESET
```

9.31.1 Actions

This command is used "as is", without any specific ACTION.

```
int rtn = 0;
```

```
rtn = fci_write(client, FPP_CMD_IPV6_RESET, 0, NULL);
```

9.31.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
  The client is not FCI owner.
- FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
  The client is not authorized to get FCI ownership.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.31.3 Examples

demo_feature_router_simple.c

demo_feature_router_nat.c

demo_feature_L2L3_bridge_vlan.c

demo_rt_ct.c

9.32 FPP_CMD_IPV4_SET_TIMEOUT

Related topics: IPv4/IPv6 Router, L2L3 Bridge
Related data types: fpp_timeout_cmd_t

```
/**
 * @def         FPP_CMD_IPV4_SET_TIMEOUT
 *
 * @brief       FCI command for configuration of conntrack timeouts.
 *
 * @details     Related data types: fpp_timeout_cmd_t
 *
 * @details     Supported `.action` values: ---
 *
 * @note        This FCI command sets timeouts for both IPv4 and IPv6 conntrack types.
 */
#define FPP_CMD_IPV4_SET_TIMEOUT
```

9.32.1 Actions

This command is used "as is", without any specific ACTION.

```
fpp_timeout_cmd_t cmd_to_fci =
{
  .protocol = ...,       // IP Protocol Number (protocol ID). [NBO]
                         // The only accepted values are 6 (TCP), 17 (UDP) or 0 (others).

  .timeout_value1 = ...  // Timeout value in seconds. [NBO]
};
```

```
int rtn = 0;
rtn = fci_write(client, FPP_CMD_IPV4_SET_TIMEOUT, sizeof(fpp_timeout_cmd_t),
                                                  (unsigned short*)(&cmd_to_fci));
```

This command allows for configuration of conntrack default timeout periods. Three protocol groups are distinguished: TCP (6), UDP (17) and others (all other protocols; usually represented by 0). Timeout can be set independently for each of these groups.

Factory-default timeout values are:

- 5 days for TCP
- 300 seconds for UDP
- 240 seconds for `others`

If these timeouts are updated (changed), then all newly created conntracks are created with updated timeout values. Conntracks which were created before the change have their timeout updated with the first received packet after the change.

9.32.2 Return values

- `FPP_ERR_OK`
  Success
- `FPP_ERR_FCI_OWNERSHIP_NOT_OWNER`
  The client is not FCI owner.
- `FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED`
  The client is not authorized to get FCI ownership.
- `FPP_ERR_INTERNAL_FAILURE`
  Internal FCI failure.

9.32.3 Examples

demo_feature_router_simple.c

demo_feature_router_nat.c

demo_feature_L2L3_bridge_vlan.c

demo_rt_ct.c

9.33 FPP_CMD_TIMER_LOCK

Related data types: `fpp_timer_cmd_t`

```
/**
 * @def         FPP_CMD_TIMER_LOCK
 *
 * @brief       FCI command to acquire ownership of IEEE 1588 timer.
 *
 * @details     Related data types: fpp_timer_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                  Acquire ownership of IEEE 1588 timer.
```

```
 */
#define FPP_CMD_TIMER_LOCK
```

9.33.1 Actions

FPP_ACTION_UPDATE

Acquire ownership of IEEE 1588 timer.

```
fpp_timer_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UPDATE,  // Action
  .name   = "...",              // Physical interface name
};

int rtn = 0;
rtn = fci_write(client, FPP_CMD_TIMER_LOCK, sizeof(fpp_timer_cmd_t),
                                            (unsigned short*)(&cmd_to_fci));
```

9.33.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_IF_ENTRY_NOT_FOUND
  Unknown (nonexistent) physical interface was requested.
- FPP_ERR_IF_NOT_SUPPORTED
  Requested physical interface does not support setting of IEEE 1588 timer ownership.
- FPP_ERR_TIMER_ALREADY_LOCKED
  The IEEE 1588 timer is already locked (owned) by other PFE driver instance.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

9.34 FPP_CMD_TIMER_UNLOCK

Related data types: fpp_timer_cmd_t

```
/**
 * @def         FPP_CMD_TIMER_UNLOCK
 *
 * @brief       FCI command to release ownership of IEEE 1588 timer.
 *
 * @details     Related data types: fpp_timer_cmd_t
 *
 * @details     Supported `.action` values:
 *              - FPP_ACTION_UPDATE
 *                   Release ownership of IEEE 1588 timer.
 */
#define FPP_CMD_TIMER_UNLOCK
```

9.34.1 Actions

FPP_ACTION_UPDATE

Release ownership of IEEE 1588 timer.

```
fpp_timer_cmd_t cmd_to_fci =
{
  .action = FPP_ACTION_UPDATE,  // Action
  .name   = "...",              // Physical interface name
};
int rtn = 0;

rtn = fci_write(client, FPP_CMD_TIMER_UNLOCK, sizeof(fpp_timer_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));
```

9.34.2 Return values

- FPP_ERR_OK
  Success
- FPP_ERR_IF_ENTRY_NOT_FOUND
  Unknown (nonexistent) physical interface was requested.
- FPP_ERR_IF_NOT_SUPPORTED
  Requested physical interface does not support setting of IEEE 1588 timer ownership.
- FPP_ERR_TIMER_NOT_OWNER
  The PFE driver instance is not owner of the target IEEE 1588 timer.
- FPP_ERR_INTERNAL_FAILURE
  Internal FCI failure.

10 Events

10.1 FPP_CMD_DATA_BUF_AVAIL

Related topics: FPP_CMD_DATA_BUF_PUT
Related data types: ---

```
/**
 * @def         FPP_CMD_DATA_BUF_AVAIL
 *
 * @brief       FCI event: driver sends custom data from accelerator to host.
 *
 * @details     Related data types: custom payload data
 *
 * @details     Indication of this event also carries the buffer payload and payload
 *              length. Both are available via the event callback arguments.
 *              See callback type and callback arguments in fci_register_cb().
 */
#define FPP_CMD_DATA_BUF_AVAIL
```

10.1.1 Actions

This FCI event returns custom, user-defined data. It does not directly use the concept of ACTIONs.

10.2 FPP_CMD_ENDPOINT_SHUTDOWN

Related topics: ---
Related data types: ---

```
/**
 * @def         FPP_CMD_ENDPOINT_SHUTDOWN
 *
 * @brief       FCI event: driver reports that FCI endpoint is shutting down.
 *
 * @details     Related data types: ---
 */
#define FPP_CMD_ENDPOINT_SHUTDOWN
```

10.2.1 Actions

This FCI event has no associated ACTIONs.

10.3 FPP_CMD_HEALTH_MONITOR_EVENT

Related topics: ---
Related data types: fpp_health_monitor_cmd_t

```
/**
 * @def         FPP_CMD_HEALTH_MONITOR_EVENT
 *
 * @brief       FCI event: driver reports some Health Monitor event.
 *
 * @details     Related data types: fpp_health_monitor_cmd_t
 *
 * @details     For details about Health Monitor events (exact meaning of ID, type and src),
 *              see Health Monitor documentation.
 */
#define FPP_CMD_HEALTH_MONITOR_EVENT
```

10.3.1 Actions

This FCI event has no associated ACTIONs.

10.4 FPP_CMD_IPV4_CONNTRACK_CHANGE

Related topics: IPv4/IPv6 Router

Related data types: fpp_ct_cmd_t

```
/**
 * @def         FPP_CMD_IPV4_CONNTRACK_CHANGE
 *
 * @brief       FCI event: driver reports status change of some IPv4 conntrack.
 *
 * @details     Related data types: fpp_ct_cmd_t
 */
#define FPP_CMD_IPV4_CONNTRACK_CHANGE
```

10.4.1 Actions

FPP_ACTION_REMOVED

Conntrack was removed from PFE IP Router. Conntrack can be removed by the following means:

- timeout (aging)
  (see FPP_CMD_IPV4_SET_TIMEOUT)
- Removed by FCI command for conntrack removal.
  (FPP_CMD_IPV4_CONNTRACK + FPP_ACTION_DEREGISTER)
- Removed by FCI command for general PFE IPv4 Router reset.
  (FPP_CMD_IPV4_RESET)

10.5 FPP_CMD_IPV6_CONNTRACK_CHANGE

Related topics: IPv4/IPv6 Router
Related data types: fpp_ct6_cmd_t

```
/**
 * @def         FPP_CMD_IPV6_CONNTRACK_CHANGE
 *
 * @brief       FCI event: driver reports status change of some IPv6 conntrack.
 *
 * @details     Related data types: fpp_ct6_cmd_t
 */
#define FPP_CMD_IPV6_CONNTRACK_CHANGE
```

10.5.1 Actions

FPP_ACTION_REMOVED

Conntrack was removed from PFE IP Router. Conntrack can be removed by the following means:

- timeout (aging)
  (see FPP_CMD_IPV4_SET_TIMEOUT)
- Removed by FCI command for conntrack removal.

(FPP_CMD_IPV6_CONNTRACK + FPP_ACTION_DEREGISTER)

- Removed by FCI command for general PFE IPv6 Router reset.
(FPP_CMD_IPV6_RESET)

11 Functions

11.1 fci_open()

Related topics: fci_close()
Related data types: fci_client_type_t, fci_mcast_groups_t

```
/**
 * @brief       Creates new FCI client and opens a connection to FCI endpoint.
 *
 * @details     Binds the FCI client with FCI endpoint. This enables sending/receiving data
 *              to/from the endpoint. Refer to the remaining API for possible communication
 *              options.
 *
 * @param[in]   client_type
 *                  Client type. Default value is FCI_CLIENT_DEFAULT.
 *                  See fci_client_type_t.
 * @param[in]   group
 *                  32-bit multicast group mask. Each bit represents single multicast address.
 *                  FCI instance will listen to specified multicast addresses as well it will
 *                  send data to all specified multicast groups.
 *                  See fci_mcast_groups_t.
 *
 * @return      The FCI client instance or NULL if failed
 */
FCI_CLIENT * fci_open(
        fci_client_type_t client_type,
        fci_mcast_groups_t group );
```

11.2 fci_close()

Related topics: fci_open()
Related data types: ---

```
/**
 * @brief       Disconnects from FCI endpoint and destroys FCI client instance.
 *
 * @details     Terminates the FCI client and releases all allocated resources.
 *
 * @param[in]   client
 *                  The FCI client instance
 *
 * @return      0 if success, error code otherwise
 */
int fci_close(
        FCI_CLIENT *client );
```

11.3 fci_catch()

Related topics: Events Summary, fci_register_cb()
Related data types: ---

```
/**
 * @brief       Catch and process FCI events delivered to the FCI client.
 *
 * @details     FCI endpoint (PFE driver) can send asynchronous messages (FCI events)
 *              to FCI clients. This function captures and processes FCI events which
 *              are delivered to FCI client.
 *
 * @details     GENERAL
 *              This is a blocking function. It should run in a dedicated parallel thread.
 *              This function has internal infinite loop. While running, it waits for
 *              incoming FCI event. When FCI event arrives, this function calls
 *              a user-defined callback to process the event. Based on return value
 *              of the callback, this function either terminates, or keeps running
 *              and waiting for next incoming FCI event.
 *
 * @details     DEVIATION for FCI API of MCAL PFE Driver
 *              In FCI API of MCAL PFE Driver, this function is not blocking.
 *              It should be periodically called (polled) from main loop of environment.
 *              The function works as follows:
 *              - When called, the function sequentially processes all FCI events which
 *                are at that moment queued in MCAL PFE Driver.
 *              - For every queued FCI event, the function calls a user-defined callback
 *                to process the event. Based on return value of the callback, the function
 *                either terminates, or moves to the next queued FCI event.
 *              - After all queued FCI events are processed, the function terminates.
 *
 * @note        To register a user-defined callback, use fci_register_cb().
 *
 * @note        Multicast group FCI_GROUP_CATCH shall be used when opening FCI client
 *              for catching messages. See fci_open().
 *
 * @see         fci_register_cb()
 *
 * @param[in]   client
 *                  FCI client instance
 *
 * @return      0 if success, error code otherwise
 *              Note that non-zero error code may be a return value of a user-defined callback.
 */
int fci_catch(
        FCI_CLIENT *client );
```

11.4 fci_cmd()

Related topics: Commands Summary
Related data types: ---

```
/**
 * @brief       Run an FCI command with optional data response.
 *
 * @details     This routine can be used when one need to perform any command either with or
 *              without data response. If the command responded with some data structure the
```

```
 *              structure is written into the rep_buf. The length of the returned
 *              data structure(number of bytes) is written into rep_len.
 *
 * @note        The rep_buf buffer must be aligned to 4.
 *
 * @param[in]      client
 *                     The FCI client instance
 * @param[in]      fcode
 *                     Command to be executed.
 * @param[in]      cmd_buf
 *                     Pointer to structure holding command arguments.
 * @param[in]      cmd_len
 *                     Length of the command arguments structure in bytes.
 * @param[out]     rep_buf
 *                     Pointer to memory where the data response shall be written.
 *                     Can be NULL.
 * @param[in,out]  rep_len
 *                     Pointer to variable where number of response bytes shall be written.
 *
 * @return       <0 Failed to execute the command.
 *              >=0 Command was executed with given return value (FPP_ERR_OK for success).
 */
int fci_cmd(
        FCI_CLIENT *client,
        unsigned short fcode,
        unsigned short *cmd_buf,
        unsigned short cmd_len,
        unsigned short *rep_buf,
        unsigned short *rep_len );
```

11.5 fci_query()

Related topics: Commands Summary

Related data types: ---

```
/**
 * @brief       Execute FCI command with data response.
 *
 * @details     This routine can be used when one need to perform a command which is resulting
 *              in a data response. It is suitable for various 'query' commands like reading
 *              of whole tables or structured entries from the endpoint.
 *
 * @note        If either rsp_data or rsplen is NULL pointer, the response data is discarded.
 *
 * @param[in]   client
 *                  The FCI client instance
 * @param[in]   fcode
 *                  Command to be executed.
 * @param[in]   cmd_len
 *                  Length of the command arguments structure in bytes.
 * @param[in]   cmd_buf
 *                  Pointer to structure holding command arguments.
 * @param[out]  rep_len
 *                  Pointer to memory where length of the data response will be provided.
 * @param[out]  rep_buf
 *                  Pointer to memory where the data response shall be written.
 *
 * @return       <0 Failed to execute the command.
 *              >=0 Command was executed with given return value (FPP_ERR_OK for success).
 */
int fci_query(
        FCI_CLIENT *client,
        unsigned short fcode,
        unsigned short cmd_len,
        unsigned short *cmd_buf,
```



```
        unsigned short *rep_len,
        unsigned short *rep_buf );
```

11.6 fci_write()

Related topics: Commands Summary
Related data types: ---

```
/**
 * @brief       Execute FCI command.
 *
 * @details     Similar as the fci_query() but without data response. The endpoint receiving
 *              the command is still responsible for generating a response but the response
 *              is not delivered to the caller.
 *
 * @param[in]   client
 *                  The FCI client instance
 * @param[in]   fcode
 *                  Command to be executed.
 * @param[in]   cmd_len
 *                  Length of the command arguments structure in bytes.
 * @param[in]   cmd_buf
 *                  Pointer to structure holding command arguments.
 *
 * @return       <0 Failed to execute the command.
 *              >=0 Command was executed with given return value (FPP_ERR_OK for success).
 */
int fci_write(
        FCI_CLIENT *client,
        unsigned short fcode,
        unsigned short cmd_len,
        unsigned short *cmd_buf );
```

11.7 fci_register_cb()

Related topics: Events Summary, fci_catch()
Related data types: fci_cb_retval_t

```
/**
 * @brief       Register a user-defined callback function for processing of FCI events.
 *
 * @details     The registered callback function is utilized by fci_catch() to process
 *              incoming FCI events. Content of the callback function is user-defined.
 *
 * @attention   In order to keep the parent fci_catch() up and running, the registered
 *              callback function must return FCI_CB_CONTINUE. When the callback function
 *              returns any other value, the parent fci_catch() terminates.
 *
 * @see         fci_catch()
 *
 * @param[in]   client
 *                  FCI client instance
 * @param[in]   event_cb
 *                  User-defined callback function.
 *
```



```
 *                   Parameters of the callback prototype:
 *                   (arguments are provided by FCI API when the callback is utilized)
 *                       [in] fcode
 *                               ID of the captured FCI event.
 *                               Equivalent to name of a #define which defines an FCI event.
 *                               (e.g. FPP_CMD_ENDPOINT_SHUTDOWN)
 *                               See FCI API Reference, chapter Events.
 *                       [in] len
 *                               Number of bytes in the payload buffer.
 *                       [in] payload
 *                               Pointer to the payload buffer. Holds data of the captured FCI event.
 *                               Some FCI events have no associated data.
 *                               See FCI API Reference, chapter Events.
 *
 * @return      0 if success, error code otherwise
 */
int fci_register_cb(
        FCI_CLIENT *client,
        fci_cb_retval_t (*event_cb)(
                            unsigned short fcode,
                            unsigned short len,
                            unsigned short *payload) );
```

12 Structs

12.1 fpp_if_m_args_t

Related topics: FPP_CMD_LOG_IF
Related data types: fpp_log_if_cmd_t, fpp_if_m_rules_t,

```
/**
 * @brief       Match rules arguments.
 *
 * @details     Related data types: fpp_log_if_cmd_t, fpp_if_m_rules_t
 *
 * @details     Each value is an argument for some match rule.
 *
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* VLAN ID. [NBO]. See FPP_IF_MATCH_VLAN. */
    uint16_t vlan;

    /* EtherType. [NBO]. See FPP_IF_MATCH_ETHTYPE. */
    uint16_t ethtype;

    /* L4 source port. [NBO]. See FPP_IF_MATCH_SPORT. */
    uint16_t sport;

    /* L4 destination port [NBO]. See FPP_IF_MATCH_DPORT. */
    uint16_t dport;

    /* Source and destination IP addresses */
    struct
    {
        struct
        {
            /* IPv4 source address. [NBO]. See FPP_IF_MATCH_SIP. */
            uint32_t sip;

            /* IPv4 destination address. [NBO]. See FPP_IF_MATCH_DIP. */
```

```
            uint32_t dip;
        } v4;

        struct
        {
            /* IPv6 source address. [NBO]. See FPP_IF_MATCH_SIP6. */
            uint32_t sip[4];

            /* IPv6 destination address. [NBO]. See FPP_IF_MATCH_DIP6. */
            uint32_t dip[4];
        } v6;
    } ipv;

    /* IP Protocol Number (protocol ID). See FPP_IF_MATCH_PROTO. */
    uint8_t proto;

    /* Source MAC Address. See FPP_IF_MATCH_SMAC. */
    uint8_t smac[6];

    /* Destination MAC Address. See FPP_IF_MATCH_DMAC. */
    uint8_t dmac[6];

    /* Flexible Parser table 0 (name). See FPP_IF_MATCH_FP0. */
    char fp_table0[16];

    /* Flexible Parser table 1 (name). See FPP_IF_MATCH_FP1. */
    char fp_table1[16];

    /* HIF header cookie. [NBO]. See FPP_IF_MATCH_HIF_COOKIE. */
    uint32_t hif_cookie;

} fpp_if_m_args_t;
```

12.2 fpp_phy_if_stats_t

Related topics: FPP_CMD_PHY_IF
Related data types: fpp_phy_if_cmd_t

```
/**
 * @brief       Physical interface statistics.
 *
 * @details     Related data types: fpp_phy_if_cmd_t
 *
 * @note        All values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Count of ingress frames for the given interface. */
    uint32_t ingress;

    /* Count of egress frames for the given interface. */
    uint32_t egress;

    /* Count of ingress frames with detected error (e.g. checksum). */
    uint32_t malformed;

    /* Count of ingress frames which were discarded. */
    uint32_t discarded;

} fpp_phy_if_stats_t;
```

12.3 fpp_algo_stats_t

Related topics: FPP_CMD_LOG_IF
Related data types: fpp_log_if_cmd_t

```
/**
 * @brief       Logical interface statistics.
 *
 * @details     Related data types: fpp_log_if_cmd_t
 *
 * @note        All values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Count of frames processed (regardless of the result). */
    uint32_t processed;

    /* Count of frames matching the selection criteria. */
    uint32_t accepted;

    /* Count of frames not matching the selection criteria. */
    uint32_t rejected;

    /* Count of frames marked to be dropped. */
    uint32_t discarded;

} fpp_algo_stats_t;
```

12.4 fpp_phy_if_cmd_t

Related topics: FPP_CMD_PHY_IF
Related data types: fpp_if_flags_t, fpp_phy_if_op_mode_t, fpp_phy_if_block_state_t, fpp_phy_if_stats_t

```
/**
 * @brief       Data structure for a physical interface.
 *
 * @details     Related FCI commands: FPP_CMD_PHY_IF
 *
 * @note        - Some values are in a network byte order [NBO].
 * @note        - Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Interface name. [ro] */
    char name[IFNAMSIZ];

    /* Interface ID. [NBO,ro] */
    uint32_t id;

    /* Interface flags. [NBO]. A bitset. */
    fpp_if_flags_t flags;

    /* Interface mode. */
    fpp_phy_if_op_mode_t mode;

    /* Interface blocking state. */
    fpp_phy_if_block_state_t block_state;

    /* Physical interface statistics. [ro] */
```

```
    fpp_phy_if_stats_t stats;

    /* Names of associated mirroring rules for ingress traffic. See FPP_CMD_MIRROR.
       Empty string at given position == position is disabled. */
    char rx_mirrors[FPP_MIRRORS_CNT][MIRROR_NAME_SIZE];

    /* Names of associated mirroring rules for egress traffic. See FPP_CMD_MIRROR.
       Empty string at given position == position is disabled. */
    char tx_mirrors[FPP_MIRRORS_CNT][MIRROR_NAME_SIZE];

    /* Name of a Flexible Parser table which shall be used
       as a Flexible Filter of this physical interface.
       Empty string == Flexible filter is disabled.
       See Flexible Parser for more info. */
    char ftable[16];

    /* Name of a physical interface which serves as
       a PTP management interface.
       Empty string == disabled */
    char ptp_mgmt_if[IFNAMSIZ];

} fpp_phy_if_cmd_t;
```

12.5 fpp_log_if_cmd_t

Related topics: FPP_CMD_LOG_IF
Related data types: fpp_if_flags_t, fpp_if_m_rules_t, fpp_if_m_args_t, fpp_algo_stats_t

```
/**
 * @brief       Data structure for a logical interface.
 *
 * @details     Related FCI commands: FPP_CMD_LOG_IF
 *
 * @note        - Some values are in a network byte order [NBO].
 * @note        - Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* RESERVED. Do not use. */
    uint8_t res[2];

    /* Interface name. [ro] */
    char name[IFNAMSIZ];

    /* Interface ID. [NBO,ro] */
    uint32_t id;

    /* Parent physical interface name. [ro] */
    char parent_name[IFNAMSIZ];

    /* Parent physical interface ID. [NBO,ro] */
    uint32_t parent_id;

    /* Egress physical interfaces. [NBO]. A bitset.
       Each physical interface is represented by a bitflag.
       Conversion between a physical interface ID and a corresponding
       bitflag is (1uL << "physical interface ID"). */
    uint32_t egress;

    /* Interface flags. [NBO]. A bitset. */
    fpp_if_flags_t flags;

    /* Match rules. [NBO]. A bitset. */
    fpp_if_m_rules_t match;
```



```
    /* Match rules arguments. */
    fpp_if_m_args_t CAL_PACKED_ALIGNED(4) arguments;

    /* Logical interface statistics [ro] */
    fpp_algo_stats_t CAL_PACKED_ALIGNED(4) stats;

} fpp_log_if_cmd_t;
```

12.6 fpp_if_mac_cmd_t

Related topics: FPP_CMD_IF_MAC
Related data types: ---

```
/**
 * @brief       Data structure for interface MAC address.
 *
 * @details     Related FCI commands: FPP_CMD_IF_MAC
 */
typedef struct CAL_PACKED_ALIGNED(2)
{
    /* Action */
    uint16_t action;

    /* Physical interface name. */
    char name[IFNAMSIZ];

    /* Physical interface MAC. */
    uint8_t mac[6];

} fpp_if_mac_cmd_t;
```

12.7 fpp_modify_args_t

Related topics: FPP_CMD_MIRROR
Related data types: fpp_mirror_cmd_t, fpp_modify_actions_t

```
/**
 * @brief       Arguments for mirroring rule modification actions.
 *
 * @details     Related data types: fpp_mirror_cmd_t, fpp_modify_actions_t
 *
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(2)
{
    /* VLAN ID to be used by MODIFY_ACT_ADD_VLAN_HDR. [NBO] */
    uint16_t vlan;

} fpp_modify_args_t;
```

12.8 fpp_mirror_cmd_t

Related topics: FPP_CMD_MIRROR

Related data types: fpp_modify_actions_t, fpp_modify_args_t

```
/**
 * @brief       Data structure for interface mirroring rule.
 *
 * @details     Related FCI commands: FPP_CMD_MIRROR
 *
 * @note        - Some values are in a network byte order [NBO].
 * @note        - Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Name of the mirroring rule. [ro] */
    char name[MIRROR_NAME_SIZE];

    /* Name of the physical interface where to mirror. */
    char egress_phy_if[IFNAMSIZ];

    /* Name of a Flexible Parser table that can be used
       to filter which frames to mirror.
       Empty string == disabled (no filtering).
       See Flexible Parser for more info. */
    char filter_table_name[16];

    /* Modifications to be done on mirrored frame. [NBO] */
    fpp_modify_actions_t m_actions;

    /* Configuration values (arguments) for m_actions. */
    fpp_modify_args_t m_args;

} fpp_mirror_cmd_t;
```

12.9 fpp_l2_bd_stats_t

Related topics: FPP_CMD_L2_BD
Related data types: fpp_l2_bd_cmd_t

```
/**
 * @brief       Domain statistics.
 *
 * @details     Related data types: fpp_l2_bd_cmd_t
 *
 * @note        All values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Count of ingress frames for the given domain. */
    uint32_t ingress;

    /* Count of egress frames for the given domain. */
    uint32_t egress;

    /* Count of ingress bytes. */
    uint32_t ingress_bytes;

    /* Count of egress bytes. */
    uint32_t egress_bytes;

} fpp_l2_bd_stats_t;
```

12.10 fpp_l2_bd_cmd_t

Related topics: FPP_CMD_L2_BD
Related data types: fpp_l2_bd_flags_t, fpp_l2_bd_stats_t

```
/**
 * @brief       Data structure for L2 bridge domain.
 *
 * @details     Related FCI commands: FPP_CMD_L2_BD
 *
 * @details     Bridge domain actions (what to do with a frame):
 *              | value | meaning |
 *              |-------|---------|
 *              | 0     | Forward |
 *              | 1     | Flood   |
 *              | 2     | Punt    |
 *              | 3     | Discard |
 *
 * @note        - Some values are in a network byte order [NBO].
 * @note        - Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Bridge domain VLAN ID. [NBO,ro] */
    uint16_t vlan;

    /* Bridge domain action when the destination MAC of an inspected
       frame is an unicast MAC and it matches some entry in the
       Bridge MAC table. */
    uint8_t ucast_hit;

    /* Bridge domain action when the destination MAC of an inspected
       frame is an unicast MAC and it does NOT match any entry in the
       Bridge MAC table. */
    uint8_t ucast_miss;

    /* Similar to ucast_hit, but for frames which have a multicast
       destination MAC address. */
    uint8_t mcast_hit;

    /* Similar to ucast_miss, but for frames which have a multicast
       destination MAC address. */
    uint8_t mcast_miss;

    /* Bridge domain ports. [NBO]. A bitset.
       Ports are represented by physical interface bitflags.
       If a bitflag of some physical interface is set here, the interface
       is then considered a port of the given bridge domain.
       Conversion between a physical interface ID and a corresponding
       bitflag is (1uL << "physical interface ID"). */
    uint32_t if_list;

    /* A bitset [NBO], denoting which bridge domain ports from
       '.if_list' are considered untagged (their egress frames
       have the VLAN tag removed).
       Ports which are present in both the '.if_list' bitset and
       this bitset are considered untagged.
       Ports which are present only in the '.if_list' bitset are
       considered tagged. */
    uint32_t untag_if_list;

    /* Bridge domain flags [NBO,ro] */
    fpp_l2_bd_flags_t flags;

    /* Domain traffic statistics. [ro] */
    fpp_l2_bd_stats_t CAL_PACKED_ALIGNED(4) stats;

} fpp_l2_bd_cmd_t;
```

12.11 fpp_l2_static_ent_cmd_t

Related topics: FPP_CMD_L2_STATIC_ENT
Related data types: ---

```
/**
 * @brief       Data structure for L2 static entry.
 *
 * @details     Related FCI commands: FPP_CMD_L2_STATIC_ENT
 *
 * @note        - Some values are in a network byte order [NBO].
 * @note        - Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* VLAN ID of an associated bridge domain. [NBO,ro]
       VLAN-aware static entries are applied only on frames
       which have a matching VLAN tag.
       For non-VLAN aware static entries, use VLAN ID of
       the Default BD (Default Bridge Domain). */
    uint16_t vlan;

    /* Static entry MAC address. [ro] */
    uint8_t mac[6];

    /* Egress physical interfaces. [NBO]. A bitset.
       Frames with matching destination MAC address (and VLAN tag)
       are forwarded through all physical interfaces which are a part
       of this bitset. Physical interfaces are represented by
       bitflags. Conversion between a physical interface ID and
       a corresponding bitflag is (1uL << "physical interface ID"). */
    uint32_t forward_list;

    /* Local MAC address. (0 == false, 1 == true)
       A part of L2L3 Bridge feature. If true, then the forward list
       of such a static entry is ignored and frames with
       a corresponding destination MAC address are passed to
       the IP router algorithm. See chapter about L2L3 Bridge. */
    uint8_t local;

    /* Frames with matching destination MAC address (and VLAN tag)
       shall be discarded. (0 == disabled, 1 == enabled) */
    uint8_t dst_discard;

    /* Frames with matching source MAC address (and VLAN tag)
       shall be discarded. (0 == disabled, 1 == enabled) */
    uint8_t src_discard;

} fpp_l2_static_ent_cmd_t;
```

12.12 fpp_fp_rule_props_t

Related topics: FPP_CMD_FP_TABLE, FPP_CMD_FP_RULE
Related data types: fpp_fp_table_cmd_t, fpp_fp_rule_cmd_t, fpp_fp_rule_match_action_t, fpp_fp_offset_from_t

```
/**
 * @brief       Properties of an FP rule (Flexible Parser rule).
 *
 * @details     Related data types: fpp_fp_table_cmd_t, fpp_fp_rule_cmd_t
 *
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED
{
    /* Rule name. A string of up to 15 characters + '\0'. */
    uint8_t rule_name[16];

    /* Expected data. [NBO]. This value is expected to be found
       at the specified offset in the inspected Ethernet frame. */
    uint32_t data;

    /* Bitmask [NBO], selecting which bits of a 32bit value shall
       be used for data comparison. This bitmask is applied on both
       '.data' value and the inspected value for the frame. */
    uint32_t mask;

    /* Offset (in bytes) of the inspected value in the frame. [NBO]
       This offset is calculated from the '.offset_from' header. */
    uint16_t offset;

    /* Invert the match result before match action is selected. */
    uint8_t invert;

    /* Name of the FP rule to jump to if '.match_action' ==
       FP_NEXT_RULE. Set all-zero if unused. This next rule must
       be in the same FP table (cannot jump across tables). */
    uint8_t next_rule_name[16];

    /* Action to do if the inspected frame matches the FP rule criteria. */
    fpp_fp_rule_match_action_t match_action;

    /* Header for offset calculation. */
    fpp_fp_offset_from_t offset_from;

} fpp_fp_rule_props_t;
```

12.13 fpp_fp_rule_cmd_t

Related topics: FPP_CMD_FP_TABLE, FPP_CMD_FP_RULE
Related data types: fpp_fp_rule_props_t

```
/**
 * @brief       Data structure for an FP rule.
 *
 * @details     Related FCI commands: FPP_CMD_FP_RULE
 */
typedef struct CAL_PACKED_ALIGNED(2)
{
    /* Action */
    uint16_t action;

    /* Properties of the rule. */
    fpp_fp_rule_props_t r;

} fpp_fp_rule_cmd_t;
```

12.14 fpp_fp_table_cmd_t

Related topics: FPP_CMD_FP_TABLE, FPP_CMD_FP_RULE
Related data types: fpp_fp_rule_props_t

```
/**
 * @brief       Data structure for an FP table.
 *
 * @details     Related FCI commands: FPP_CMD_FP_TABLE
 *
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(2)
{
    /* Action */
    uint16_t action;

    union
    {
        struct
        {
            /* Name of the FP table to be administered. */
            uint8_t table_name[16];

            /* Name of the FP rule to be added/removed. */
            uint8_t rule_name[16];

            /* Position in the table where to add the rule. [NBO] */
            uint16_t position;
        } t;

        /* Query result - properties of a rule from the table */
        fpp_fp_rule_props_t r;
    } table_info;

} fpp_fp_table_cmd_t;
```

12.15 fpp_buf_cmd_t

Related topics: FPP_CMD_DATA_BUF_PUT
Related data types: ---

```
/**
 * @brief       Data structure for sending custom data to accelerator.
 *
 * @details     Related FCI commands: FPP_CMD_DATA_BUF_PUT
 */
typedef struct CAL_PACKED
{
    /* The payload. */
    uint8_t payload[64];

    /* Payload length in number of bytes. */
    uint8_t len;

    /* RESERVED. Do not use. */
    uint8_t reserved1;

    /* RESERVED. Do not use. */
    uint16_t reserved2;

} fpp_buf_cmd_t;
```


12.16 fpp_spd_cmd_t

Related topics: FPP_CMD_SPD

Related data types: fpp_spd_flags_t, fpp_spd_action_t,

```
/**
 * @brief       Data structure for an SPD entry.
 *
 * @details     Related FCI commands: FPP_CMD_SPD
 *
 * @note        Some values are in a network byte order [NBO].
 * @note        HSE is a Hardware Security Engine, a separate HW accelerator.
 *              Its configuration is outside the scope of this document.
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Physical interface name. */
    char name[IFNAMSIZ];

    /* SPD entry flags. A bitset. */
    fpp_spd_flags_t flags;

    /* Entry position. [NBO]
       0 : insert as the first entry of the SPD table.
       N : insert as the Nth entry of the SPD table, starting from 0.
       Entries are inserted (not overwritten). Already existing entries are shifted
       to make room for the newly inserted one.
       If (N > current count of SPD entries) then the new entry gets inserted
       as the last entry of the SPD table. */
    uint16_t position;

    /* Source IP address. [NBO]
       IPv4 uses only element [0]. Address type is set in '.flags' */
    uint32_t saddr[4];

    /* Destination IP address. [NBO]
       IPv4 uses only element [0]. Address type is set in '.flags' */
    uint32_t daddr[4];

    /* Source port. [NBO]
       Optional (does not have to be set). See '.flags' */
    uint16_t sport;

    /* Destination port. [NBO]
       Optional (does not have to be set). See '.flags' */
    uint16_t dport;

    /* IANA IP Protocol Number (protocol ID). */
    uint8_t protocol;

    /* SAD entry identifier for HSE. [NBO]
       Used only when '.spd_action' == SPD_ACT_PROCESS_ENCODE).
       Corresponding SAD entry must exist in HSE. */
    uint32_t sa_id;

    /* SPI to match in the ingress traffic. [NBO]
       Used only when '.spd_action' == SPD_ACT_PROCESS_DECODE). */
    uint32_t spi;

    /* Action to be done on the frame. */
    fpp_spd_action_t spd_action;

} fpp_spd_cmd_t;
```

12.17 fpp_qos_queue_cmd_t

Related topics: FPP_CMD_QOS_QUEUE
Related data types: ---

```
/**
 * @brief       Data structure for QoS queue.
 *
 * @details     Related FCI commands: FPP_CMD_QOS_QUEUE
 *
 * @note        - Some values are in a network byte order [NBO].
 * @note        - Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Physical interface name. [ro] */
    char if_name[IFNAMSIZ];

    /* Queue ID. [ro]
       minimal ID == 0
       maximal ID is implementation defined. See Egress QoS. */
    uint8_t id;

    /* Queue mode:
       0 == Disabled. Queue will drop all packets.
       1 == Default. HW implementation-specific. Normally not used.
       2 == Tail drop
       3 == WRED */
    uint8_t mode;

    /* Minimum threshold. [NBO].
       Value is `.mode`-specific:
       - Disabled, Default:
           n/a
       - Tail drop:
           n/a
       - WRED:
           Threshold in number of packets in the queue at which
           the WRED lowest drop probability zone starts.
           While the queue fill level is below this threshold,
           the drop probability is 0%. */
    uint32_t min;

    /* Maximum threshold. [NBO].
       Value is `.mode`-specific:
       - Disabled, Default:
           n/a
       - Tail drop:
           The queue length in number of packets.
           Queue length is the number of packets
           the queue can accommodate before drops will occur.
       - WRED:
           Threshold in number of packets in the queue at which
           the WRED highest drop probability zone ends.
           While the queue fill level is above this threshold,
           the drop probability is 100%. */
    uint32_t max;

    /* WRED drop probabilities for all probability zones in [%].
       The lowest probability zone is `.zprob[0]`.
       Only valid for `.mode = WRED`.
       Value 255 means 'invalid'.
       Number of zones per queue is implementation-specific. See Egress QoS. */
    uint8_t zprob[32];
```

```
} fpp_qos_queue_cmd_t;
```

12.18 fpp_qos_scheduler_cmd_t

Related topics: FPP_CMD_QOS_SCHEDULER
Related data types: ---

```
/**
 * @brief       Data structure for QoS scheduler.
 *
 * @details     Related FCI commands: FPP_CMD_QOS_SCHEDULER
 *
 * @note        - Some values are in a network byte order [NBO].
 * @note        - Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Physial interface name. [ro] */
    char if_name[IFNAMSIZ];

    /* Scheduler ID. [ro]
       minimal ID == 0
       maximal ID is implementation defined. See Egress QoS. */
    uint8_t id;

    /* Scheduler mode:
       0 == Scheduler disabled
       1 == Data rate (payload length)
       2 == Packet rate (number of packets) */
    uint8_t mode;

    /* Scheduler algorithm:
        0 == PQ (Priority Queue).
               Input with the highest priority is serviced first.
               Input 0 has the lowest priority.
        1 == DWRR (Deficit Weighted Round Robin).
        2 == RR (Round Robin).
        3 == WRR (Weighted Round Robin). */
    uint8_t algo;

    /* Input enable bitfield. [NBO]
       When a bit `n` is set it means that scheduler input `n`
       is enabled and connected to traffic source defined by `.source[n]`.
       Number of inputs is implementation-specific. See Egress QoS. */
    uint32_t input_en;

    /* Input weight. [NBO].
       Scheduler algorithm-specific:
       - PQ, RR:
           n/a
       - WRR, DWRR:
           Weight in units given by `.mode` */
    uint32_t input_w[32];

    /* Traffic source for each scheduler input.
       Traffic sources are implementation-specific. See Egress QoS. */
    uint8_t input_src[32];

} fpp_qos_scheduler_cmd_t;
```

12.19 fpp_qos_shaper_cmd_t

Related topics: FPP_CMD_QOS_SHAPER
Related data types: ---

```
/**
 * @brief       Data structure for QoS shaper.
 *
 * @details     Related FCI commands: FPP_CMD_QOS_SHAPER
 *
 * @note        - Some values are in a network byte order [NBO].
 * @note        - Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Physial interface name. [ro] */
    char if_name[IFNAMSIZ];

    /* Shaper ID. [ro]
       minimal ID == 0
       maximal ID is implementation defined. See Egress QoS. */
    uint8_t id;

    /* Position of the shaper.
       Positions are implementation defined. See Egress QoS. */
    uint8_t position;

    /* Idle slope in units per second (see `.mode`). [NBO] */
    uint32_t isl;

    /* Max credit. [NBO] */
    int32_t max_credit;

    /* Min credit. [NBO] */
    int32_t min_credit;

    /* Shaper mode:
       0 == Shaper disabled
       1 == Data rate.
            `.isl` is in bits-per-second.
            `.max_credit` and `.min_credit` are in number of bytes.
       2 == Packet rate.
            `isl` is in packets-per-second.
            `.max_credit` and `.min_credit` are in number of packets. */
    uint8_t mode;

} fpp_qos_shaper_cmd_t;
```

12.20 fpp_qos_policer_cmd_t

Related topics: FPP_CMD_QOS_POLICER
Related data types: ---

```
/**
 * @brief       Data structure for Ingress QoS policer enable/disable.
 *
 * @details     Related FCI commands: FPP_CMD_QOS_POLICER
 *
 * @note        Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
```



```
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Physical interface name ('emac' interfaces only). [ro] */
    char if_name[IFNAMSIZ];

    /* Enable/disable switch of the Ingress QoS Policer HW module.
       0 == disabled, 1 == enabled. */
    uint8_t enable;

} fpp_qos_policer_cmd_t;
```

12.21 fpp_iqos_flow_args_t

Related topics: FPP_CMD_QOS_POLICER_FLOW, FPP_IQOS
Related data types: fpp_iqos_flow_spec_t, fpp_iqos_flow_arg_type_t

```
/**
 * @brief       Arguments for argumentful flow types.
 *
 * @details     Related data types: fpp_iqos_flow_spec_t, fpp_iqos_flow_arg_type_t
 *
 * @details     Bitmasking works as follows:
 *              if ((PacketData & Mask) == (ArgData & Mask)), then packet matches the flow.
 *                - `PacketData` is the inspected value from an ingress packet.
 *                - `ArgData`    is the argument value of an argumentful flow type.
 *                - `Mask`       is the bitmask of an argumentful flow type (_m).
 *
 *              Example:
 *              If `.l4proto_m = 0x07` , then only the lowest 3 bits of the L4 protocol field
 *              are compared. Any protocol with the matching lowest 3 bits is accepted.
 *
 *              For IP addresses, the network prefix (e.g. /24) is internally converted
 *              to valid subnet mask (/24 == 0xFFFFFFF0).
 *
 *              It is advised to use the precomputed bitmask symbols when comparing
 *              whole values (all bits). Do not use custom bitmasks unless
 *              some specific scenario needs such refinement.
 *
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* FPP_IQOS_ARG_VLAN: VLAN ID (max 4095). [NBO] */
    uint16_t vlan;

    /* FPP_IQOS_ARG_VLAN: VLAN ID comparison bitmask (12b). [NBO]
       Use FPP_IQOS_VLAN_ID_MASK to compare whole value (all bits). */
    uint16_t vlan_m;

    /* FPP_IQOS_ARG_TOS: TOS field for IPv4, TCLASS for IPv6. */
    uint8_t tos;

    /* FPP_IQOS_ARG_TOS: TOS comparison bitmask.
       Use FPP_IQOS_TOS_MASK to compare whole value (all bits). */
    uint8_t tos_m;

    /* FPP_IQOS_ARG_L4PROTO: L4 protocol field for IPv4 and IPv6. */
    uint8_t l4proto;

    /* FPP_IQOS_ARG_L4PROTO: L4 protocol comparison bitmask.
       Use FPP_IQOS_L4PROTO_MASK to compare whole value (all bits). */
    uint8_t l4proto_m;

    /* FPP_IQOS_ARG_SIP: Source IP address for IPv4/IPv6. [NBO] */
    uint32_t sip;
```

```
    /* FPP_IQOS_ARG_DIP: Destination IP address for IPv4/IPv6. [NBO] */
    uint32_t dip;

    /* FPP_IQOS_ARG_SIP: Source IP address - network prefix.
       Use FPP_IQOS_SDIP_MASK to compare whole address (all bits). */
    uint8_t sip_m;

    /* FPP_IQOS_ARG_DIP: Destination IP address - network prefix.
       Use FPP_IQOS_SDIP_MASK to compare whole address (all bits). */
    uint8_t dip_m;

    /* FPP_IQOS_ARG_SPORT: Max L4 source port. [NBO] */
    uint16_t sport_max;

    /* FPP_IQOS_ARG_SPORT: Min L4 source port. [NBO] */
    uint16_t sport_min;

    /* FPP_IQOS_ARG_DPORT: Max L4 destination port. [NBO] */
    uint16_t dport_max;

    /* FPP_IQOS_ARG_DPORT: Min L4 destination port. [NBO] */
    uint16_t dport_min;

} fpp_iqos_flow_args_t;
```

12.22 fpp_iqos_flow_spec_t

Related topics: FPP_CMD_QOS_POLICER_FLOW
Related data types: fpp_qos_policer_flow_cmd_t, fpp_iqos_flow_type_t, fpp_iqos_flow_arg_type_t, fpp_iqos_flow_args_t, fpp_iqos_flow_action_t

```
/**
 * @brief       Specification of Ingress QoS packet flow.
 *
 * @details     Related FCI commands: FPP_CMD_QOS_POLICER_FLOW
 *
 * @details     Related data types: fpp_qos_policer_flow_cmd_t
 *
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Argumentless flow types to match. Bitset mask. [NBO] */
    fpp_iqos_flow_type_t type_mask;

    /* Argumentful flow types to match. Bitset mask. [NBO] */
    fpp_iqos_flow_arg_type_t arg_type_mask;

    /* Arguments for argumentful flow types. Related to 'arg_type_mask'. */
    fpp_iqos_flow_args_t CAL_PACKED_ALIGNED(4) args;

    /* Action to be done for matching packets. */
    fpp_iqos_flow_action_t action;

} fpp_iqos_flow_spec_t;
```

12.23 fpp_qos_policer_flow_cmd_t

Related topics: FPP_CMD_QOS_POLICER_FLOW

Related data types: fpp_iqos_flow_spec_t

```
/**
 * @brief       Data structure for Ingress QoS packet flow.
 *
 * @details     Related FCI commands: FPP_CMD_QOS_POLICER_FLOW
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Physical interface name ('emac' interfaces only). */
    char if_name[IFNAMSIZ];

    /* Position in the classification table.
       minimal ID == 0
       maximal ID is implementation defined. See Ingress QoS.
       For FPP_ACTION_REGISTER, value 0xFF means "don't care".
       If 0xFF is set as registration id, driver will automatically
       choose the first available free position. */
    uint8_t id;

    /* Flow specification. */
    fpp_iqos_flow_spec_t CAL_PACKED_ALIGNED(4) flow;

} fpp_qos_policer_flow_cmd_t;
```

12.24 fpp_qos_policer_wred_cmd_t

Related topics: FPP_CMD_QOS_POLICER_WRED
Related data types: fpp_iqos_queue_t, fpp_iqos_wred_zone_t, fpp_iqos_wred_thr_t

```
/**
 * @brief       Data structure for Ingress QoS WRED queue.
 *
 * @details     Related FCI commands: FPP_CMD_QOS_POLICER_WRED
 *
 * @note        - Some values are in a network byte order [NBO].
 * @note        - Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Physical interface name ('emac' interfaces only). [ro] */
    char if_name[IFNAMSIZ];

    /* Target Ingress QoS WRED queue. [ro] */
    fpp_iqos_queue_t queue;

    /* Enable/disable switch of the target WRED queue HW module.
       0 == disabled, 1 == enabled. */
    uint8_t enable;

    /* WRED queue thresholds. [NBO]
       See fpp_iqos_wred_thr_t.
       Unit is "number of packets".
       Min value == 0
       Max value is implementation defined. See Ingress QoS.
       If set as `0xFFFF`, then HW keeps its currently configured value. */
    uint16_t thr[FPP_IQOS_WRED_THR_COUNT];

    /* WRED drop probabilities for all probability zones.
```

```
        See fpp_iqos_wred_zone_t.
        One unit (1) represents probability of 1/16.
        Min value == 0   ( 0/16 = 0%)
        Max value == 15  (15/16 = 93,75%)
        If set as 0xFF, then HW keeps its currently configured value. */
    uint8_t zprob[FPP_IQOS_WRED_ZONES_COUNT];

} fpp_qos_policer_wred_cmd_t;
```

12.25 fpp_qos_policer_shp_cmd_t

Related topics: FPP_CMD_QOS_POLICER_SHP
Related data types: fpp_iqos_shp_type_t, fpp_iqos_shp_rate_mode_t

```
/**
 * @brief       Data structure for Ingress QoS shaper.
 *
 * @details     Related FCI commands: @ref FPP_CMD_QOS_POLICER_SHP
 *
 * @note        - Some values are in a network byte order [NBO].
 * @note        - Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Physical interface name ('emac' interfaces only). [ro] */
    char if_name[IFNAMSIZ];

    /* ID of the target Ingress QoS shaper. [ro]
       Min ID == 0
       Max ID is implementation defined. See Ingress QoS. */
    uint8_t id;

    /* Enable/disable switch of the target Ingress QoS shaper
       HW module. 0 == disabled, 1 == enabled. */
    uint8_t enable;

    /* Shaper type. Port level, bcast or mcast. */
    fpp_iqos_shp_type_t type;

    /* Shaper mode. Bits or packets. */
    fpp_iqos_shp_rate_mode_t mode;

    /* Idle slope. Units are '.mode' dependent. [NBO] */
    uint32_t isl;

    /* Max credit. Units are '.mode' dependent. [NBO] */
    int32_t max_credit;

    /* Min credit. Units are '.mode' dependent. [NBO]
       Must be negative. */
    int32_t min_credit;

} fpp_qos_policer_shp_cmd_t;
```

12.26 fpp_fw_features_cmd_t

Related topics: FPP_CMD_FW_FEATURE

Related data types: fpp_fw_feature_flags_t

```
/**
 * @brief       Data structure for FW feature setting.
 *
 * @details     Related FCI commands: FPP_CMD_FW_FEATURE
 *
 * @note        Some values cannot be modified by FPP_ACTION_UPDATE [ro].
 */
typedef struct CAL_PACKED_ALIGNED(2)
{
    /* Action */
    uint16_t action;

    /* Feature name. [ro] */
    char name[FPP_FEATURE_NAME_SIZE + 1];

    /* Feature description. [ro] */
    char desc[FPP_FEATURE_DESC_SIZE + 1];

    /* Feature current state.
       0 == disabled ; 1 == enabled */
    uint8_t val;

    /* Feature configuration variant. [ro] */
    fpp_fw_feature_flags_t flags;

    /* Factory default value of the '.val' property. [ro] */
    uint8_t def_val;

    /* RESERVED. Do not use. */
    uint8_t reserved;

} fpp_fw_features_cmd_t;
```

12.27 fpp_fw_features_element_cmd_t

Related topics: FPP_CMD_FW_FEATURE_ELEMENT

Related data types: ---

```
/**
 * @brief       Data structure for FW feature element.
 *
 * @details     Related FCI commands: FPP_CMD_FW_FEATURE_ELEMENT
 *
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Name of a fw feature. (see fpp_fw_features_cmd_t) */
    char fw_feature_name[FPP_FEATURE_NAME_SIZE + 1];

    /* Name of the fw feature's target element */
    char element_name[FPP_FEATURE_NAME_SIZE + 1];

    /* Element group
       0 : ANY (no group specified)
           Special value, intended only for FPP_ACTION_QUERY.
           FPP_ACTION_QUERY command with EMPTY element_name[] and with this group
           starts a QUERY/QUERY_CONT session that will successively report all
           elements of the parent fw feature (regardless of their element group).
       1 : CFG (configuration group)
```

```
            Command with this group can target only some configuration element.
            FPP_ACTION_QUERY command with EMPTY element_name[] and with this group
            starts a QUERY/QUERY_CONT process that will successively report all
            configuration elements of the parent fw feature.
        2 : STATS (statistics group)
            Command with this group can target only some statistics element.
            FPP_ACTION_QUERY command with EMPTY element_name[] and with this group
            starts a QUERY/QUERY_CONT session that will successively report all
            statistics elements of the parent fw feature. */
    uint8_t group;

    /* Byte size of element's data unit.
       Data unit exact size (and underlying data type) is feature and element specific.
       See appropriate documentation of fw feature elements. */
    uint8_t unit_size;

    /* Index into element data (as laid out in PFE firmware).
       The first data unit in .payload is related to this index. */
    uint8_t index;

    /* Count of consecutive data units in .payload */
    uint8_t count;

    /* Data (composed of one or more data units).
       IMPORTANT:
       Endianess of multibyte data units is element-specific.
       See appropriate documentation of FW feature elements. */
    uint8_t payload[128];

} fpp_fw_features_element_cmd_t;
```

12.28 fpp_health_monitor_cmd_t

Related topics: FPP_CMD_HEALTH_MONITOR_EVENT
Related data types: ---

```
/**
 * @brief       Data structure for Health Monitor event.
 *
 * @details     Related FCI event: FPP_CMD_HEALTH_MONITOR_EVENT
 *
 * @details     For details about Health Monitor events (exact meaning of ID, type and src),
 *              see Health Monitor documentation.
 */
typedef struct CAL_PACKED_ALIGNED(2)
{
    /* Action */
    uint16_t action;

    /* Event ID as reported by Health Monitor. */
    uint16_t id;

    /* 0 == information ; 1 == warning ; 2 == error */
    uint8_t type;

    /* Source (which part of PFE asserted the event) */
    uint8_t src;

    /* Event description */
    char desc[FPP_HEALTH_MONITOR_DESC_SIZE];

} fpp_health_monitor_cmd_t;
```

12.29 fpp_conntrack_stats_t

Related topics: FPP_CMD_IPV4_CONNTRACK, FPP_CMD_IPV6_CONNTRACK
Related data types: fpp_ct_cmd_t, fpp_ct6_cmd_t

```
/**
 * @brief       Conntrack statistics.
 *
 * @details     Related data types: fpp_ct_cmd_t, fpp_ct6_cmd_t
 *
 * @note        All values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Number of frames that hit the conntrack. */
    uint32_t hit;

    /* Sum of bytesizes of all frames that hit the conntrack. */
    uint32_t hit_bytes;

} fpp_conntrack_stats_t;
```

12.30 fpp_ct_cmd_t

Related topics: FPP_CMD_IPV4_CONNTRACK
Related data types: fpp_conntrack_stats_t

```
/**
 * @brief       Data structure for IPv4 conntrack.
 *
 * @details     Related FCI commands: FPP_CMD_IPV4_CONNTRACK, FPP_CMD_IP_ROUTE
 *
 * @details     For detailed explanation how to create conntracks, see IP Router.
 *
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* RESERVED. Do not use. */
    uint16_t rsvd0;

    /* 'orig' direction: Source IP address. [NBO] */
    uint32_t saddr;

    /* 'orig' direction: Destination IP address. [NBO] */
    uint32_t daddr;

    /* 'orig' direction: Source port. [NBO] */
    uint16_t sport;

    /* 'orig' direction: Destination port. [NBO] */
    uint16_t dport;

    /* 'reply' direction: Source IP address. [NBO]
        Used for NAT, otherwise equals '.daddr'. */
    uint32_t saddr_reply;

    /* 'reply' direction: Destination IP address. [NBO]
        Used for NAT, otherwise equals '.saddr'. */
    uint32_t daddr_reply;
```

```
    /* 'reply' direction: Source port. [NBO]
        Used for NAT, otherwise equals '.dport'. */
    uint16_t sport_reply;

    /* 'reply' direction: Destination port. [NBO]
        Used for NAT, otherwise equals '.sport'. */
    uint16_t dport_reply;

    /* IANA IP Protocol Number (protocol ID). [NBO] */
    uint16_t protocol;

    /* Flags. A bitset. [NBO]. See FPP_CMD_IPV4_CONNTRACK. */
    uint16_t flags;

    /* RESERVED. Do not use. */
    uint32_t fwmark;

    /* 'orig' direction: ID of an associated route. [NBO]
        See FPP_CMD_IP_ROUTE. */
    uint32_t route_id;

    /* 'reply' direction: ID of an associated route. [NBO]
        See FPP_CMD_IP_ROUTE. */
    uint32_t route_id_reply;

    /* 'orig' direction: VLAN tag. [NBO]
        If non-zero, then this VLAN tag is added to the routed packet.
        If the packet already has a VLAN tag, then its tag is replaced. */
    uint16_t vlan;

    /* 'reply' direction: VLAN tag. [NBO]
        If non-zero, then this VLAN tag is added to the routed packet.
        If the packet already has a VLAN tag, then its tag is replaced. */
    uint16_t vlan_reply;

    /* 'orig'  statistics [ro] */
    fpp_conntrack_stats_t CAL_PACKED_ALIGNED(4) stats;

    /* 'reply' statistics [ro] */
    fpp_conntrack_stats_t CAL_PACKED_ALIGNED(4) stats_reply;

} fpp_ct_cmd_t;
```

12.31 fpp_ct6_cmd_t

Related topics: FPP_CMD_IPV6_CONNTRACK
Related data types: fpp_conntrack_stats_t

```
/**
 * @brief       Data structure for IPv6 conntrack.
 *
 * @details     Related FCI commands: FPP_CMD_IPV6_CONNTRACK, FPP_CMD_IP_ROUTE
 *
 * @details     For detailed explanation how to create conntracks, see IP Router.
 *
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* RESERVED. Do not use. */
    uint16_t rsvd1;

    /* 'orig' direction: Source IP address. [NBO] */
    uint32_t saddr[4];
```

```
    /* 'orig' direction: Destination IP address. [NBO] */
    uint32_t daddr[4];

    /* 'orig' direction: Source port. [NBO] */
    uint16_t sport;

    /* 'orig' direction: Destination port. [NBO] */
    uint16_t dport;

    /* 'reply' direction: Source IP address. [NBO]
        Used for NAT, otherwise equals '.daddr'. */
    uint32_t saddr_reply[4];

    /* 'reply' direction: Destination IP address. [NBO]
        Used for NAT, otherwise equals '.saddr'. */
    uint32_t daddr_reply[4];

    /* 'reply' direction: Source port. [NBO]
        Used for NAT, otherwise equals '.dport'. */
    uint16_t sport_reply;

    /* 'reply' direction: Destination port. [NBO]
        Used for NAT, otherwise equals '.sport'. */
    uint16_t dport_reply;

    /* IANA IP Protocol Number (protocol ID). [NBO] */
    uint16_t protocol;

    /* Flags. A bitset. [NBO]. See FPP_CMD_IPV4_CONNTRACK. */
    uint16_t flags;

    /* RESERVED. Do not use. */
    uint32_t fwmark;

    /* 'orig' direction: ID of an associated route. [NBO]
        See FPP_CMD_IP_ROUTE. */
    uint32_t route_id;

    /* 'reply' direction: ID of an associated route. [NBO]
        See FPP_CMD_IP_ROUTE. */
    uint32_t route_id_reply;

    /* 'orig' direction: VLAN tag. [NBO]
        If non-zero, then this VLAN tag is added to the routed packet.
        If the packet already has a VLAN tag, then its tag is replaced. */
    uint16_t vlan;

    /* 'reply' direction: VLAN tag. [NBO]
        If non-zero, then this VLAN tag is added to the routed packet.
        If the packet already has a VLAN tag, then its tag is replaced. */
    uint16_t vlan_reply;

    /* 'orig'  statistics [ro] */
    fpp_conntrack_stats_t CAL_PACKED_ALIGNED(4) stats;

    /* 'reply' statistics [ro] */
    fpp_conntrack_stats_t CAL_PACKED_ALIGNED(4) stats_reply;

} fpp_ct6_cmd_t;
```

12.32 fpp_rt_cmd_t

Related topics: FPP_CMD_IP_ROUTE
Related data types: ---

```
/**
 * @brief       Data structure for a route.
```

```
 *
 * @details     Related FCI commands: FPP_CMD_IP_ROUTE
 *
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* RESERVED. Do not use. */
    uint16_t mtu;

    /* Source MAC address. When a packet is routed, this address
       is set as the source MAC address of the packet. If left
       unset (all-zero), then PFE automatically uses MAC address
       of the associated physical interface (.output_device). */
    uint8_t src_mac[6];

    /* Destination MAC address. When a packet is routed, this address
       is set as the destination MAC address of the packet. */
    uint8_t dst_mac[6];

    /* RESERVED. Do not use. */
    uint16_t pad;

    /* Name of the egress physical interface.
       When a packet is routed, it is egressed through this physical interface. */
    char output_device[IFNAMSIZ];

    /* RESERVED. Do not use. */
    char input_device[IFNAMSIZ];

    /* RESERVED Do not use. */
    char underlying_input_device[IFNAMSIZ];

    /* Route ID. [NBO]. Unique route identifier. */
    uint32_t id;

    /* Flags. [NBO].
       1 for IPv4 route, 2 for IPv6 route. */
    uint32_t flags;

    /* RESERVED. Do not use. */
    uint32_t dst_addr[4];

} fpp_rt_cmd_t;
```

12.33 fpp_timeout_cmd_t

Related topics: FPP_CMD_IPV4_SET_TIMEOUT
Related data types: ---

```
/**
 * @brief       Data structure for conntrack timeout setting.
 *
 * @details     Related FCI commands: FPP_CMD_IPV4_SET_TIMEOUT
 *
 * @note        This FCI command sets timeouts for both IPv4 and IPv6 conntrack types.
 * @note        Some values are in a network byte order [NBO].
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* IP Protocol Number (protocol ID). [NBO]
       The only accepted values are:
        6 (timeout for TCP traffic)
       17 (timeout for UDP traffic)
        0 (timeout for all other traffic) */
```

```
    uint16_t protocol;

    /* RESERVED. Do not use. */
    uint16_t sam_4o6_timeout;

    /* New timeout value in seconds. [NBO] */
    uint32_t timeout_value1;

    /* RESERVED. Do not use. */
    uint32_t timeout_value2;

} fpp_timeout_cmd_t;
```

12.34 fpp_timer_cmd_t

Related topics: FPP_CMD_TIMER_LOCK, FPP_CMD_TIMER_UNLOCK
Related data types: ---

```
/**
 * @brief       Data structure for IEEE 1588 timer ownership.
 *
 * @details     Related FCI commands: FPP_CMD_TIMER_LOCK, FPP_CMD_TIMER_UNLOCK
 */
typedef struct CAL_PACKED_ALIGNED(4)
{
    /* Action */
    uint16_t action;

    /* Physical interface name. */
    char if_name[IFNAMSIZ];

    /* RESERVED (do not use) */
    char reserved;
} fpp_timer_cmd_t;
```

13 Enums

13.1 fci_mcast_groups_t

Related topics: fci_open()
Related data types: ---

```
/**
 * @typedef     fci_mcast_groups_t
 *
 * @brief       List of supported multicast groups.
 *
 * @details     An FCI client instance can be member of a multicast group.
 *              It means it can send and receive multicast messages to/from another
 *              group members (another FCI instances or FCI endpoints). This can be
 *              in most cases used by FCI endpoint to notify all associated FCI instances
 *              about some event has occurred.
 *
 * @note        Each group is intended to be represented by a single bit flag
 *              (max 32-bit, so it is possible to have max 32 multicast groups).
 *              Then, groups can be combined using bitwise OR operation.
 */
```

```
typedef enum
{
    /* Default MCAST group value (no group). Intended for sending of FCI commands. */
    FCI_GROUP_NONE

    /* MCAST group for catching events. */
    FCI_GROUP_CATCH

} fci_mcast_groups_t;
```

13.2 fci_client_type_t

Related topics: fci_open()
Related data types: ---

```
/**
 * @typedef     fci_client_type_t
 *
 * @brief       List of supported FCI client types.
 *
 * @details     FCI client can specify using this type to which FCI endpoint shall be connected.
 */
typedef enum
{
    /* Default type (equivalent of legacy FCILIB_FF_TYPE macro) */
    FCI_CLIENT_DEFAULT

    /* Due to compatibility purposes */
    FCILIB_FF_TYPE

} fci_client_type_t;
```

13.3 fci_cb_retval_t

Related topics: fci_register_cb()
Related data types: ---

```
/**
 * @typedef     fci_cb_retval_t
 *
 * @brief       Return values of user-defined callback function.
 *
 * @details     These return values shall be returned by user-defined callback function.
 *              They signal to fci_catch() function whether it should continue or terminate.
 *
 * @see         fci_register_cb()
 */
typedef enum
{
    /* Stop waiting for FCI events and terminate fci_catch() function. */
    FCI_CB_STOP

    /* Continue waiting for next FCI events. */
    FCI_CB_CONTINUE

} fci_cb_retval_t;
```

13.4 fpp_if_flags_t

Related topics: FPP_CMD_PHY_IF, FPP_CMD_LOG_IF
Related data types: fpp_phy_if_cmd_t, fpp_log_if_cmd_t

```
/**
 * @brief       Interface flags
 *
 * @details     Related data types: fpp_phy_if_cmd_t, fpp_log_if_cmd_t
 *
 * @details     Some of these flags are applicable only for physical interfaces [phyif],
 *              some are applicable only for logical interfaces [logif] and some are applicable
 *              for both [phyif,logif].
 */
typedef enum CAL_PACKED
{
    /* [phyif,logif]
        If set, the interface is enabled. */
    FPP_IF_ENABLED

    /* [phyif,logif]
        If set, the interface is configured as promiscuous.
        promiscuous phyif:
          All ingress traffic is accepted, regardless of destination MAC.
        promiscuous logif:
          All inspected traffic is accepted, regardless of active match rules. */
    FPP_IF_PROMISC

    /* [phyif]
        Special handling of ingress TCP SYN|FIN|RST packets in routing process.
        Applicable only when the interface uses mode which involves traffic routing
        (e.g. FPP_IF_OP_ROUTER).
        If set:
          Ingress TCP SYN|FIN|RST packets which match some conntrack are fast-forwarded
          as usual (according to the matching conntrack). This is default behavior.
        If not set:
          Ingress TCP SYN|FIN|RST packets which match some conntrack are not fast-forwarded.
          Instead, they are passed to the default logical interface. */
    FPP_IF_FF_ALL_TCP

    /* [logif]
        If multiple match rules are active and this flag is set,
        then the final result of a match process is logical OR of the rules.
        If this flag is not set, then the final result is logical AND of the rules. */
    FPP_IF_MATCH_OR

    /* [logif]
        If set, discard matching frames. */
    FPP_IF_DISCARD



    /* [phyif]
        If set, the interface enforces a strict VLAN conformance check. */
    FPP_IF_VLAN_CONF_CHECK

    /* [phyif]
        If set, the interface enforces a strict PTP conformance check. */
    FPP_IF_PTP_CONF_CHECK

    /* [phyif]
        If set, then PTP traffic is accepted even if the FPP_IF_VLAN_CONF_CHECK is set. */
    FPP_IF_PTP_PROMISC

    /* [logif]
        If set, a loopback mode is enabled. */
    FPP_IF_LOOPBACK

    /* [phyif]
        If set, the interface accepts QinQ-tagged traffic. */
    FPP_IF_ALLOW_Q_IN_Q
```

```
    /* [phyif]
        If set, then packets with TTL<2 are automatically discarded.
        If not set, then packets with TTL<2 are passed to the default logical interface. */
    FPP_IF_DISCARD_TTL


} fpp_if_flags_t;
```

13.5 fpp_if_m_rules_t

Related topics: FPP_CMD_LOG_IF
Related data types: fpp_log_if_cmd_t, fpp_if_m_args_t,

```
/**
 * @brief       Match rules.
 *
 * @details     Related data types: fpp_log_if_cmd_t, fpp_if_m_args_t
 *
 * @note        L2/L3/L4 are layers of the OSI model.
 */
typedef enum CAL_PACKED
{
    /* Match ETH packets */
    FPP_IF_MATCH_TYPE_ETH

    /* Match VLAN tagged packets */
    FPP_IF_MATCH_TYPE_VLAN

    /* Match PPPoE packets */
    FPP_IF_MATCH_TYPE_PPPOE

    /* Match ARP packets */
    FPP_IF_MATCH_TYPE_ARP

    /* Match multicast (L2) packets */
    FPP_IF_MATCH_TYPE_MCAST

    /* Match IPv4 packets */
    FPP_IF_MATCH_TYPE_IPV4

    /* Match IPv6 packets */
    FPP_IF_MATCH_TYPE_IPV6

    /* Reserved */
    FPP_IF_MATCH_RESERVED7

    /* Reserved */
    FPP_IF_MATCH_RESERVED8

    /* Match IPX packets */
    FPP_IF_MATCH_TYPE_IPX

    /* Match L2 broadcast packets */
    FPP_IF_MATCH_TYPE_BCAST

    /* Match UDP packets */
    FPP_IF_MATCH_TYPE_UDP

    /* Match TCP packets */
    FPP_IF_MATCH_TYPE_TCP

    /* Match ICMP packets */
    FPP_IF_MATCH_TYPE_ICMP

    /* Match IGMP packets */
    FPP_IF_MATCH_TYPE_IGMP
```

```
    /* Match VLAN ID (see fpp_if_m_args_t) */
    FPP_IF_MATCH_VLAN

    /* Match IP Protocol Number (protocol ID) See fpp_if_m_args_t. */
    FPP_IF_MATCH_PROTO

    /* Match L4 source port (see fpp_if_m_args_t) */
    FPP_IF_MATCH_SPORT

    /* Match L4 destination port (see fpp_if_m_args_t) */
    FPP_IF_MATCH_DPORT

    /* Match source IPv6 address (see fpp_if_m_args_t)
       This rule is mutually exclusive with the following match rules:
       FPP_IF_MATCH_SIP, FPP_IF_MATCH_DIP. */
    FPP_IF_MATCH_SIP6

    /* Match destination IPv6 address (see fpp_if_m_args_t)
       This rule is mutually exclusive with the following match rules:
       FPP_IF_MATCH_SIP, FPP_IF_MATCH_DIP. */
    FPP_IF_MATCH_DIP6

    /* Match source IPv4 address (see fpp_if_m_args_t)
       This rule is mutually exclusive with the following match rules:
       FPP_IF_MATCH_SIP6, FPP_IF_MATCH_DIP6. */
    FPP_IF_MATCH_SIP

    /* Match destination IPv4 address (see fpp_if_m_args_t)
       This rule is mutually exclusive with the following match rules:
       FPP_IF_MATCH_SIP6, FPP_IF_MATCH_DIP6. */
    FPP_IF_MATCH_DIP

    /* Match EtherType (see fpp_if_m_args_t) */
    FPP_IF_MATCH_ETHTYPE

    /* Match Ethernet frames accepted by Flexible Parser 0 (see fpp_if_m_args_t) */
    FPP_IF_MATCH_FP0

    /* Match Ethernet frames accepted by Flexible Parser 1 (see fpp_if_m_args_t) */
    FPP_IF_MATCH_FP1

    /* Match source MAC address (see fpp_if_m_args_t) */
    FPP_IF_MATCH_SMAC

    /* Match destination MAC address (see fpp_if_m_args_t) */
    FPP_IF_MATCH_DMAC

    /* Match HIF header cookie. HIF header cookie is a part of internal overhead data.
       It is attached to traffic data by a host's PFE driver. */
    FPP_IF_MATCH_HIF_COOKIE


} fpp_if_m_rules_t;
```

13.6 fpp_phy_if_op_mode_t

Related topics: FPP_CMD_PHY_IF
Related data types: fpp_phy_if_cmd_t

```
/**
 * @brief       Physical interface operation mode.
 *
 * @details     Related data types: fpp_phy_if_cmd_t
 */
typedef enum CAL_PACKED
{
    /* Default operation mode */
    FPP_IF_OP_DEFAULT
```

```
    /* L2 Bridge */
    FPP_IF_OP_VLAN_BRIDGE

    /* IPv4/IPv6 Router */
    FPP_IF_OP_ROUTER

    /* Flexible Router */
    FPP_IF_OP_FLEXIBLE_ROUTER

    /* L2L3 Bridge */
    FPP_IF_OP_L2L3_VLAN_BRIDGE

} fpp_phy_if_op_mode_t;
```

13.7 fpp_phy_if_block_state_t

Related topics: FPP_CMD_PHY_IF
Related data types: fpp_phy_if_cmd_t

```
/**
 * @brief       Physical interface blocking state.
 *
 * @details     Related data types: fpp_phy_if_cmd_t
 *
 * @details     Used when a physical interface is configured in a Bridge-like mode.
 *              See L2 Bridge and L2L3 Bridge. Affects the following Bridge-related
 *              capabilities of a physical interface:
 *                - Learning of MAC addresses from the interface's ingress traffic.
 *                - Forwarding of the interface's ingress traffic.
 */
typedef enum CAL_PACKED
{
    /* Learning and forwarding enabled. */
    BS_NORMAL

    /* Learning and forwarding disabled. */
    BS_BLOCKED

    /* Learning enabled, forwarding disabled. */
    BS_LEARN_ONLY

    /* Learning disabled, forwarding enabled.
       Traffic is forwarded only if its both source and destination MAC addresses
       are known to the bridge. */
    BS_FORWARD_ONLY

} fpp_phy_if_block_state_t;
```

13.8 fpp_modify_actions_t

Related topics: FPP_CMD_MIRROR
Related data types: fpp_mirror_cmd_t, fpp_modify_args_t

```
/**
 * @brief       Mirroring rule modification actions.
 *
 * @details     Related data types: fpp_mirror_cmd_t, fpp_modify_args_t
 */
typedef enum CAL_PACKED
```

```
{
    /* No action to be done. */
    MODIFY_ACT_NONE

    /* Construct/Update outer VLAN Header. */
    MODIFY_ACT_ADD_VLAN_HDR


} fpp_modify_actions_t;
```

13.9 fpp_l2_bd_flags_t

Related topics: FPP_CMD_L2_BD
Related data types: fpp_l2_bd_cmd_t

```
/**
 * @brief       L2 bridge domain flags
 *
 * @details     Related data types: fpp_l2_bd_cmd_t
 */
typedef enum CAL_PACKED
{
    /* Domain type is default */
    FPP_L2_BD_DEFAULT

    /* Domain type is fallback */
    FPP_L2_BD_FALLBACK

} fpp_l2_bd_flags_t;
```

13.10 fpp_fp_rule_match_action_t

Related topics: FPP_CMD_FP_RULE
Related data types: fpp_fp_rule_props_t

```
/**
 * @brief       Action to do with an inspected Ethernet frame
 *              if the frame matches FP rule criteria.
 *
 * @details     Related data types: fpp_fp_rule_props_t
 *
 * @details     Exact meaning of FP_ACCEPT and FP_REJECT (what happens with the inspected frame)
 *              depends on the context in which the parent FP table is used. See Flexible Parser.
 *              Generally (without any further logic inversions), FP_ACCEPT means the frame is
 *              accepted and processed by PFE, while FP_REJECT means the frame is discarded.
 */
typedef enum CAL_PACKED
{
    /* Flexible Parser accepts the frame. */
    FP_ACCEPT

    /* Flexible Parser rejects the frame. */
    FP_REJECT

    /* Flexible Parser continues with the matching process,
       but jumps to a specific FP rule in the FP table. */
    FP_NEXT_RULE

} fpp_fp_rule_match_action_t;
```

13.11 fpp_fp_offset_from_t

Related topics: FPP_CMD_FP_RULE
Related data types: fpp_fp_rule_props_t

```
/**
 * @brief       Header for offset calculation.
 *
 * @details     Related data types: fpp_fp_rule_props_t
 *
 * @details     Offset can be calculated either from the L2, L3 or L4 header beginning.
 *              The L2 header is also the beginning of an Ethernet frame.
 *
 * @details     L2 header is always a valid header for offset calculation.
 *              Other headers may be missing in some Ethernet frames.
 *              If an FP rule expects L3/L4 header (for offset calculation) but the given
 *              header is missing in the inspected Ethernet frame, then the result
 *              of the matching process is "frame does not match FP rule criteria".
 */
typedef enum CAL_PACKED
{
    /* Calculate offset from the L2 header (frame beginning). */
    FP_OFFSET_FROM_L2_HEADER

    /* Calculate offset from the L3 header. */
    FP_OFFSET_FROM_L3_HEADER

    /* Calculate offset from the L4 header. */
    FP_OFFSET_FROM_L4_HEADER

} fpp_fp_offset_from_t;
```

13.12 fpp_spd_action_t

Related topics: FPP_CMD_SPD
Related data types: fpp_spd_cmd_t

```
/**
 * @brief       Action to be done for frames matching the SPD entry criteria.
 *
 * @details     Related data types: fpp_spd_cmd_t
 */
typedef enum CAL_PACKED
{

    /* Discard the frame. */
    FPP_SPD_ACTION_DISCARD

    /* Bypass IPsec and forward normally. */
    FPP_SPD_ACTION_BYPASS

    /* Send to HSE for encoding. */
    FPP_SPD_ACTION_PROCESS_ENCODE

    /* Send to HSE for decoding. */
    FPP_SPD_ACTION_PROCESS_DECODE

} fpp_spd_action_t;
```

13.13 fpp_spd_flags_t

Related topics: FPP_CMD_SPD
Related data types: fpp_spd_cmd_t

```
/**
 * @brief       Flags for SPD entry.
 *
 * @details     Related data types: fpp_spd_cmd_t
 */
typedef enum CAL_PACKED
{
    /* IPv4 if this flag NOT set. IPv6 if set. */
    FPP_SPD_FLAG_IPv6

    /* If set: do NOT match fpp_spd_cmd_t.sport. */
    FPP_SPD_FLAG_SPORT_OPAQUE

    /* If set: do NOT match fpp_spd_cmd_t.dport. */
    FPP_SPD_FLAG_DPORT_OPAQUE

} fpp_spd_flags_t;
```

13.14 fpp_iqos_flow_type_t

Related topics: FPP_CMD_QOS_POLICER_FLOW
Related data types: fpp_iqos_flow_spec_t

```
/**
 * @brief       Argumentless flow types (match flags).
 *
 * @details     Related data types: fpp_iqos_flow_spec_t
 */
typedef enum CAL_PACKED
{
    /* Match ETH packets. */
    FPP_IQOS_FLOW_TYPE_ETH

    /* Match PPPoE packets. */
    FPP_IQOS_FLOW_TYPE_PPPOE

    /* Match ARP packets. */
    FPP_IQOS_FLOW_TYPE_ARP

    /* Match IPv4 packets. */
    FPP_IQOS_FLOW_TYPE_IPV4

    /* Match IPv6 packets. */
    FPP_IQOS_FLOW_TYPE_IPV6

    /* Match IPX packets. */
    FPP_IQOS_FLOW_TYPE_IPX

    /* Match L2 multicast packets. */
    FPP_IQOS_FLOW_TYPE_MCAST

    /* Match L2 broadcast packets. */
    FPP_IQOS_FLOW_TYPE_BCAST

    /* Match VLAN tagged packets. */
```

```
    FPP_IQOS_FLOW_TYPE_VLAN


} fpp_iqos_flow_type_t;
```

13.15 fpp_iqos_flow_arg_type_t

Related topics: FPP_CMD_QOS_POLICER_FLOW
Related data types: fpp_iqos_flow_spec_t, fpp_iqos_flow_args_t

```
/**
 * @brief       Argumentful flow types (match flags).
 *
 * @details     Related data types: fpp_iqos_flow_spec_t, fpp_iqos_flow_args_t
 */
typedef enum CAL_PACKED
{
    /* Match bitmasked VLAN value. */
    FPP_IQOS_ARG_VLAN

    /* Match bitmasked TOS value. */
    FPP_IQOS_ARG_TOS

    /* Match bitmasked L4 protocol value. */
    FPP_IQOS_ARG_L4PROTO

    /* Match prefixed source IPv4/IPv6 address. */
    FPP_IQOS_ARG_SIP

    /* Match prefixed destination IPv4/IPv6 address. */
    FPP_IQOS_ARG_DIP

    /* Match L4 source port range. */
    FPP_IQOS_ARG_SPORT

    /* Match L4 destination port range. */
    FPP_IQOS_ARG_DPORT


} fpp_iqos_flow_arg_type_t;
```

13.16 fpp_iqos_flow_action_t

Related topics: FPP_CMD_QOS_POLICER_FLOW
Related data types: fpp_iqos_flow_spec_t

```
/**
 * @brief       Action to be done for matching packets.
 *
 * @details     Related data types: fpp_iqos_flow_spec_t
 */
typedef enum CAL_PACKED
{
    /* Classify the matching packet as Managed traffic. Default action. */
    FPP_IQOS_FLOW_MANAGED

    /* Drop the packet. */
    FPP_IQOS_FLOW_DROP
```

```
    /* Classify the matching packet as Reserved traffic. */
    FPP_IQOS_FLOW_RESERVED

} fpp_iqos_flow_action_t;
```

13.17 fpp_iqos_queue_t

Related topics: FPP_CMD_QOS_POLICER_WRED
Related data types: fpp_qos_policer_wred_cmd_t

```
/**
 * @brief       Supported target queues of Ingress QoS WRED.
 *
 * @details     Related data types: fpp_qos_policer_wred_cmd_t
 */
typedef enum CAL_PACKED
{
    /* Queue which is in DMEM (Data Memory). Standard storage. */
    FPP_IQOS_Q_DMEM

    /* Queue which is in LMEM (Local Memory). Faster execution but smaller capacity. */
    FPP_IQOS_Q_LMEM

    /* Queue which is in FIFO of the associated physical interface. */
    FPP_IQOS_Q_RXF

} fpp_iqos_queue_t;
```

13.18 fpp_iqos_wred_zone_t

Related topics: FPP_CMD_QOS_POLICER_WRED
Related data types: fpp_qos_policer_wred_cmd_t

```
/**
 * @brief       Supported probability zones of Ingress QoS WRED queue.
 *
 * @details     Related data types: fpp_qos_policer_wred_cmd_t
 *
 * @note        This enum represents valid array indexes into
 *              `fpp_qos_policer_wred_cmd_t.zprob[]`.
 */
typedef enum CAL_PACKED
{
    /* WRED probability zone 1 (lowest). */
    FPP_IQOS_WRED_ZONE1

    /* WRED probability zone 2. */
    FPP_IQOS_WRED_ZONE2

    /* WRED probability zone 3. */
    FPP_IQOS_WRED_ZONE3

    /* WRED probability zone 4 (highest). */
    FPP_IQOS_WRED_ZONE4

} fpp_iqos_wred_zone_t;
```

13.19 fpp_iqos_wred_thr_t

Related topics: FPP_CMD_QOS_POLICER_WRED
Related data types: fpp_qos_policer_wred_cmd_t

```
/**
 * @brief       Thresholds of Ingress QoS WRED queue.
 *
 * @details     Related data types: fpp_qos_policer_wred_cmd_t
 *
 * @note        This enum represents valid array indexes into
 *              `fpp_qos_policer_wred_cmd_t.thr[]`
 */
typedef enum CAL_PACKED
{
    /* WRED queue min threshold.
        If queue fill below `.thr[FPP_IQOS_WRED_MIN_THR]`, the following applies:
        - Drop Unmanaged traffic by probability zones.
        - Keep Managed and Reserved traffic. */
    FPP_IQOS_WRED_MIN_THR

    /* WRED queue max threshold.
       If queue fill over `.thr[FPP_IQOS_WRED_MIN_THR]` but below `.thr[FPP_IQOS_WRED_MAX_THR]`,
       the following applies:
       - Drop all Unmanaged and Managed traffic.
       - Keep Reserved traffic. */
    FPP_IQOS_WRED_MAX_THR

    /* WRED queue full threshold.
       If queue fill over `.thr[FPP_IQOS_WRED_FULL_THR]`, then drop all traffic. */
    FPP_IQOS_WRED_FULL_THR


} fpp_iqos_wred_thr_t;
```

13.20 fpp_iqos_shp_type_t

Related topics: FPP_CMD_QOS_POLICER_SHP
Related data types: fpp_qos_policer_shp_cmd_t

```
/**
 * @brief       Types of Ingress QoS shaper.
 *
 * @details     Related data types: fpp_qos_policer_shp_cmd_t
 */
typedef enum CAL_PACKED
{
    /* Port level data rate shaper. */
    FPP_IQOS_SHP_PORT_LEVEL

    /* Shaper for broadcast packets. */
    FPP_IQOS_SHP_BCAST

    /* Shaper for multicast packets. */
    FPP_IQOS_SHP_MCAST


} fpp_iqos_shp_type_t;
```

13.21 fpp_iqos_shp_rate_mode_t

Related topics: FPP_CMD_QOS_POLICER_SHP
Related data types: fpp_qos_policer_shp_cmd_t

```
/**
 * @brief       Modes of Ingress QoS shaper.
 *
 * @details     Related data types: fpp_qos_policer_shp_cmd_t
 */
typedef enum CAL_PACKED
{
    /* `.isl` is in bits-per-second.
       `.max_credit` and `.min_credit` are in number of bytes. */
    FPP_IQOS_SHP_BPS

    /* `.isl` is in packets-per-second.
       `.max_credit` and `.min_credit` are in number of packets. */
    FPP_IQOS_SHP_PPS


} fpp_iqos_shp_rate_mode_t;
```

13.22 fpp_fw_feature_flags_t

Related topics: FPP_CMD_FW_FEATURE
Related data types: fpp_fw_features_cmd_t

```
/**
 * @brief       Feature flags
 *
 * @details     Flags combinations:
 *              - FEAT_PRESENT is missing:
 *                   The feature is not available.
 *              - FEAT_PRESENT is set, but FEAT_RUNTIME is missing:
 *                   The feature is always enabled (cannot be disabled).
 *              - FEAT_PRESENT is set and FEAT_RUNTIME is set:
 *                   The feature can be enabled/disable at runtime.
 *                   Enable state must be read out of DMEM.
 */
typedef enum CAL_PACKED
{
    /* RESERVED */
    FEAT_NONE

    /* Feature not available if this not set. */
    FEAT_PRESENT

    /* Feature can be enabled/disabled at runtime. */
    FEAT_RUNTIME

} fpp_fw_feature_flags_t;
```

13.23 FW feature element groups

Related topics: FPP_CMD_FW_FEATURE_ELEMENT

Related data types: ---

```
enum CAL_PACKED
{
    /* ANY (no group specified; command will target all available groups) */
    FW_FEATURE_ELEMENT_DEFAUT

    /* CFG (command will target only configuration group) */
    FW_FEATURE_ELEMENT_CONFIG

    /* STATS (command will target only statistics group) */
    FW_FEATURE_ELEMENT_STATS

};
```

14 Miscellaneous symbols

14.1 FCI_CLIENT

Related topics: ---
Related data types: ---

```
/**
 * @struct      FCI_CLIENT
 *
 * @brief       The FCI client representation type
 *
 * @details     This is the FCI instance representation. It is used by the rest of the API
 *              to communicate with associated endpoint. The endpoint can be a standalone
 *              application/driver taking care of HW configuration tasks and shall be able
 *              to interpret commands sent via the LibFCI API.
 */
typedef struct __fci_client_tag FCI_CLIENT;
```

14.2 FPP_IQOS

Related topics: Ingress QoS
Related data types: fpp_iqos_flow_args_t

```
/**
 * @brief       Pre-computed bitmask for comparison
 *              of the whole VLAN ID (all bits compared).
 *
 * @details     Related data types: fpp_iqos_flow_args_t
 */
#define FPP_IQOS_VLAN_ID_MASK

/**
 * @brief       Pre-computed bitmask for comparison
 *              of the whole TOS/TCLASS field (all bits compared).
 *
 * @details     Related data types: fpp_iqos_flow_args_t
 */
```

```
#define FPP_IQOS_TOS_MASK


/**
 * @brief       Pre-computed bitmask for comparison
 *              of the whole L4 protocol field (all bits compared).
 *
 * @details     Related data types: fpp_iqos_flow_args_t
 */
#define FPP_IQOS_L4PROTO_MASK


/**
 * @brief       Pre-computed bitmask for comparison
 *              of the whole IP address (all bits compared).
 *
 * @details     Related data types: fpp_iqos_flow_args_t
 */
#define FPP_IQOS_SDIP_MASK
```

14.3 FPP_CTCMD

Related topics: IPv4/IPv6 Router, FPP_CMD_IPV4_CONNTRACK, FPP_CMD_IPV6_CONNTRACK
Related data types: fpp_ct_cmd_t, fpp_ct6_cmd_t

```
/**
 * @def         CTCMD_FLAGS_ORIG_DISABLED
 *
 * @brief       Disable connection originator.
 *
 * @details     Related data types: fpp_ct_cmd_t, fpp_ct6_cmd_t
 */
#define CTCMD_FLAGS_ORIG_DISABLED


/**
 * @def         CTCMD_FLAGS_REP_DISABLED
 *
 * @brief       Disable connection replier. Can be used to create uni-directional connections.
 *
 * @details     Related data types: fpp_ct_cmd_t, fpp_ct6_cmd_t
 */
#define CTCMD_FLAGS_REP_DISABLED


/**
 * @def         CTCMD_FLAGS_TTL_DECREMENT
 *
 * @brief       Enable TTL decrement. Conntrack with this flag decrements TTL of routed packets.
 *
 * @details     Related data types: fpp_ct_cmd_t, fpp_ct6_cmd_t
 */
#define CTCMD_FLAGS_TTL_DECREMENT
```

14.4 FPP_ACTION

Related topics: ---
Related data types: ---

```
#define FPP_ACTION_REGISTER
```

```
#define FPP_ACTION_DEREGISTER
#define FPP_ACTION_KEEP_ALIVE
#define FPP_ACTION_REMOVED
#define FPP_ACTION_UPDATE
#define FPP_ACTION_QUERY
#define FPP_ACTION_QUERY_CONT
#define FPP_ACTION_USE_RULE
#define FPP_ACTION_UNUSE_RULE
```

14.5 FPP_ERR

Related topics: ---
Related data types: ---

```
#define FPP_ERR_OK
#define FPP_ERR_UNKNOWN_COMMAND
#define FPP_ERR_WRONG_COMMAND_SIZE
#define FPP_ERR_WRONG_COMMAND_PARAM
#define FPP_ERR_UNKNOWN_ACTION
#define FPP_ERR_ENTRY_NOT_FOUND
#define FPP_ERR_INTERNAL_FAILURE

#define FPP_ERR_IF_ENTRY_ALREADY_REGISTERED
#define FPP_ERR_IF_ENTRY_NOT_FOUND
#define FPP_ERR_IF_EGRESS_DOESNT_EXIST
#define FPP_ERR_IF_EGRESS_UPDATE_FAILED
#define FPP_ERR_IF_MATCH_UPDATE_FAILED
#define FPP_ERR_IF_OP_UPDATE_FAILED
#define FPP_ERR_IF_OP_CANNOT_CREATE
#define FPP_ERR_IF_NOT_SUPPORTED
#define FPP_ERR_IF_RESOURCE_ALREADY_LOCKED
#define FPP_ERR_IF_WRONG_SESSION_ID

#define FPP_ERR_IF_MAC_ALREADY_REGISTERED
#define FPP_ERR_IF_MAC_NOT_FOUND

#define FPP_ERR_MIRROR_ALREADY_REGISTERED
```

```
#define FPP_ERR_MIRROR_NOT_FOUND

#define FPP_ERR_L2_BD_ALREADY_REGISTERED
#define FPP_ERR_L2_BD_NOT_FOUND

#define FPP_ERR_L2_STATIC_ENT_ALREADY_REGISTERED
#define FPP_ERR_L2_STATIC_EN_NOT_FOUND

#define FPP_ERR_FP_RULE_NOT_FOUND

#define FPP_ERR_AGAIN

#define FPP_ERR_QOS_QUEUE_NOT_FOUND
#define FPP_ERR_QOS_QUEUE_SUM_OF_LENGTHS_EXCEEDED
#define FPP_ERR_QOS_SCHEDULER_NOT_FOUND
#define FPP_ERR_QOS_SHAPER_NOT_FOUND

#define FPP_ERR_QOS_POLICER_FLOW_TABLE_FULL
#define FPP_ERR_QOS_POLICER_FLOW_NOT_FOUND

#define FPP_ERR_FW_FEATURE_NOT_FOUND
#define FPP_ERR_FW_FEATURE_NOT_AVAILABLE

#define FPP_ERR_FW_FEATURE_ELEMENT_NOT_FOUND
#define FPP_ERR_FW_FEATURE_ELEMENT_READ_ONLY

#define FPP_ERR_FCI_OWNERSHIP_NOT_AUTHORIZED
#define FPP_ERR_FCI_OWNERSHIP_ALREADY_LOCKED
#define FPP_ERR_FCI_OWNERSHIP_NOT_OWNER
#define FPP_ERR_FCI_OWNERSHIP_NOT_ENABLED

#define FPP_ERR_TIMER_ALREADY_LOCKED
#define FPP_ERR_TIMER_NOT_OWNER

#define FPP_ERR_CT_ENTRY_ALREADY_REGISTERED
#define FPP_ERR_CT_ENTRY_NOT_FOUND
```



```
#define FPP_ERR_RT_ENTRY_ALREADY_REGISTERED

#define FPP_ERR_RT_ENTRY_NOT_FOUND
```

14.6 Misc

Related topics: ---
Related data types: ---

```
#ifndef CAL_PACKED
#define CAL_PACKED    __attribute__((packed))
#endif /* CAL_PACKED */


#ifndef CAL_PACKED_ALIGNED
#define CAL_PACKED_ALIGNED(n) __attribute__((packed, aligned(n)))
#endif /* CAL_PACKED_ALIGNED */


#define IFNAMSIZ 16 /* Maximum length of interface name */


/* Size limit for the strings specifying mirror name. */
#define MIRROR_NAME_SIZE 16


/* Number of mirrors which can be configured per rx/tx on a physical interface.
   The value is equal to the number supported by the firmware. */
#define FPP_MIRRORS_CNT 2U


#define FPP_FEATURE_NAME_SIZE 32


#define FPP_FEATURE_DESC_SIZE 128

                #define FPP_HEALTH_MONITOR_DESC_SIZE
```

15 Examples

15.1 demo_feature_physical_interface.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
```

```
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"
#include "demo_if_mac.h"
#include "demo_mirror.h"

extern int demo_feature_L2_bridge_vlan(FCI_CLIENT* p_cl);


/*
 * @brief      Use libFCI to configure advanced properties of physical interfaces.
 * @details    Scenario description:
 *               [*] Let there be two computers (PCs), both in the same network subnet.
 *                   Both PCs are connected to PFE, each to one PFE emac physical interface.
 *                   PFE acts as a simple bridge.
 *               [*] MAC address filtering:
 *                   Selected emac physical interfaces should not work in a promiscuous mode,
 *                   but should accept only traffic from a selected range of destination MAC
 *                   addresses. Use libFCI to configure this MAC address filtering.
 *               [*] Mirroring:
 *                   Use libFCI to create and assign mirroring rules. Task is to mirror
 *                   a copy of all PC0<->PC1 communication to emac2 physical interface.
 *             PC description:
 *               PC0:
 *                 --> IP address:  10.3.0.2/24
 *                 --> MAC address: 0A:01:23:45:67:89
 *                     (this is just a demo MAC; real MAC of the real PC0 should be used)
 *                 --> Accessible via PFE's emac0 physical interface.
 *               PC1:
 *                 --> IP address:  10.3.0.5/24
 *                 --> MAC address: 0A:FE:DC:BA:98:76
 *                     (this is just a demo MAC; real MAC of the real PC1 should be used)
 *                 --> Accessible via PFE's emac1 physical interface.
 *
 * @note       This code uses a suite of "demo_" functions. The "demo_" functions encapsulate
 *             manipulation of libFCI data structs and calls of libFCI functions.
 *             It is advised to inspect content of these "demo_" functions.
 *
 * @param[in]  p_cl         FCI client
 *                          To create a client, use libFCI function fci_open().
 * @return     FPP_ERR_OK : All FCI commands were successfully executed.
 *                          Physical interfaces should be configured now.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_feature_physical_interface(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = FPP_ERR_OK;


    /* setup PFE to classify traffic (not needed, but done for demo purposes)*/
    /* ==================================================================== */
    rtn = demo_feature_L2_bridge_vlan(p_cl);


    /* create a mirroring rule */
    /* ====================== */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_mirror_add(p_cl, NULL, "MirroringRule0", "emac2");
    }


    /* configure physical interfaces */
    /* ============================ */
    if (FPP_ERR_OK == rtn)
    {
        /* lock the interface database of PFE */
        rtn = demo_if_session_lock(p_cl);
```

```
        if (FPP_ERR_OK == rtn)
        {
            fpp_phy_if_cmd_t phyif = {0};

            /* configure physical interface "emac0" */
            /* ----------------------------------- */
            if (FPP_ERR_OK == rtn)
            {
                /* add MAC address filter: accept traffic with dest. MAC == MAC of PC1 */
                if (FPP_ERR_OK == rtn)
                {
                    rtn = demo_if_mac_add(p_cl, (uint8_t[6]){0x0A,0xFE,0xDC,0xBA,0x98,0x76},
                                                "emac0");
                }


                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac0");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_set_rx_mirror(&phyif, 0, "MirroringRule0");  /* mirror */
                    demo_phy_if_ld_set_promisc(&phyif, false);  /* disable promiscuous mode
                                                                   (MAC filters are used) */

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }

            /* configure physical interface "emac1" */
            /* ----------------------------------- */
            if (FPP_ERR_OK == rtn)
            {
                /* add MAC address filter: accept traffic with dest. MAC == MAC of PC0 */
                if (FPP_ERR_OK == rtn)
                {
                    rtn = demo_if_mac_add(p_cl, (uint8_t[6]){0x0A,0x01,0x23,0x45,0x67,0x89},
                                                "emac1");
                }


                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac1");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_set_rx_mirror(&phyif, 0, "MirroringRule0");  /* mirror */
                    demo_phy_if_ld_set_promisc(&phyif, false);  /* disable promiscuous mode
                                                                   (MAC filters are used) */

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }

            /* configure physical interface "emac2" */
            /* ----------------------------------- */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac2");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_DEFAULT);
                    demo_phy_if_ld_set_block_state(&phyif, BS_NORMAL);

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }
        }

        /* unlock the interface database of PFE */
        rtn = demo_if_session_unlock(p_cl, rtn);
    }


    return (rtn);
}


/* ========================================================================= */
```


15.2 demo_feature_L2_bridge_vlan.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"
#include "demo_l2_bd.h"


/*
 * @brief      Use libFCI to configure PFE as a VLAN-aware L2 bridge.
 * @details    Scenario description:
 *               [*] Let there be six computers (PCs):
 *                     --> Three PCs (PC0_NOVLAN, PC0_100 and PC0_200) are accessible via
 *                         PFE's emac0 physical interface.
 *                     --> Three PCs (PC1_NOVLAN, PC1_100 and PC1_200) are accessible via
 *                         PFE's emac1 physical interface.
 *               [*] Use libFCI to configure PFE as a VLAN-aware L2 bridge, allowing the PCs
 *                   to communicate as follows:
 *                     --> PC0_NOVLAN and PC1_NOVLAN  (untagged traffic)
 *                     --> PC0_100 and PC1_100        (VLAN 100 tagged traffic)
 *                     --> PC0_200 and PC1_200        (VLAN 200 tagged traffic)
 *               [*] Additional requirements:
 *                     --> Dynamic learning of MAC addresses shall be disabled on
 *                         emac0 and emac1 interfaces.
 *                     --> In VLAN 200 domain, a replica of all passing traffic shall be sent
 *                         to a host.
 *             PC description:
 *               PC0_NOVLAN
 *                 --> IP  address: 10.3.0.2/24
 *                 --> MAC address: 0A:01:23:45:67:89
 *                 --> Accessible via PFE's emac0 physical interface.
 *                 --> Sends untagged traffic
 *               PC1_NOVLAN
 *                 --> IP  address: 10.3.0.5/24
 *                 --> MAC address: 0A:FE:DC:BA:98:76
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Sends untagged traffic
 *               PC0_100:
 *                 --> IP  address: 10.100.0.2/24
 *                 --> MAC address: 02:11:22:33:44:55
 *                 --> Accessible via PFE's emac0 physical interface.
 *                 --> Belongs to VLAN 100 domain.
 *               PC1_100:
```

```
 *                 --> IP  address: 10.100.0.5/24
 *                 --> MAC address: 02:66:77:88:99:AA
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Belongs to VLAN 100 domain.
 *               PC0_200:
 *                 --> IP  address: 10.200.0.2/24
 *                 --> MAC address: 06:CC:BB:AA:99:88
 *                 --> Accessible via PFE's emac0 physical interface.
 *                 --> Belongs to VLAN 200 domain.
 *               PC1_200:
 *                 --> IP  address: 10.200.0.5/24
 *                 --> MAC address: 06:77:66:55:44:33
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Belongs to VLAN 200 domain.
 *
 * @note       This code uses a suite of "demo_" functions. The "demo_" functions encapsulate
 *             manipulation of libFCI data structs and calls of libFCI functions.
 *             It is advised to inspect content of these "demo_" functions.
 *
 * @param[in]  p_cl         FCI client
 *                          To create a client, use libFCI function fci_open().
 * @return     FPP_ERR_OK : All FCI commands were successfully executed.
 *                          VLAN-aware L2 bridge should be up and running.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_feature_L2_bridge_vlan(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = FPP_ERR_OK;


    /* clear L2 bridge MAC table (not required; done for demo purposes) */
    /* ============================================================== */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_l2_flush_all(p_cl);
    }


    /* create and configure bridge domains */
    /* =================================== */
    if (FPP_ERR_OK == rtn)
    {
        fpp_l2_bd_cmd_t bd = {0};

        /* Default BD (VLAN == 1) */
        /* ---------------------- */
        /* This bridge domain already exists (automatically created at driver startup). */
        /* It is used by PFE to process untagged traffic. */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "bd" */
            rtn = demo_l2_bd_get_by_vlan(p_cl, &bd, 1u);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_l2_bd_ld_insert_phyif(&bd, 0u, false);  /* 0u == ID of emac0 */
                demo_l2_bd_ld_insert_phyif(&bd, 1u, false);  /* 1u == ID of emac1 */
                demo_l2_bd_ld_set_ucast_hit(&bd, 0u);   /* 0u == bridge action "FORWARD" */
                demo_l2_bd_ld_set_ucast_miss(&bd, 1u);  /* 1u == bridge action "FLOOD" */
                demo_l2_bd_ld_set_mcast_hit(&bd, 0u);   /* 0u == bridge action "FORWARD" */
                demo_l2_bd_ld_set_mcast_miss(&bd, 1u);  /* 1u == bridge action "FLOOD" */

                /* update data in PFE */
                rtn = demo_l2_bd_update(p_cl, &bd);
            }
        }

        /* bridge domain 100 */
        /* ----------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new bridge domain in PFE */
            rtn = demo_l2_bd_add(p_cl, &bd, 100u);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new domain */
                demo_l2_bd_ld_insert_phyif(&bd, 0u, true);  /* 0u == ID of emac0 */
                demo_l2_bd_ld_insert_phyif(&bd, 1u, true);  /* 1u == ID of emac1 */
                demo_l2_bd_ld_set_ucast_hit(&bd, 0u);   /* 0u == bridge action "FORWARD" */
                demo_l2_bd_ld_set_ucast_miss(&bd, 1u);  /* 1u == bridge action "FLOOD" */
                demo_l2_bd_ld_set_mcast_hit(&bd, 0u);   /* 0u == bridge action "FORWARD" */
                demo_l2_bd_ld_set_mcast_miss(&bd, 1u);  /* 1u == bridge action "FLOOD" */

                /* update the new bridge domain in PFE */
                rtn = demo_l2_bd_update(p_cl, &bd);
```

```
        }
    }

    /* bridge domain 200 */
    /* ----------------- */
    if (FPP_ERR_OK == rtn)
    {
        /* create a new bridge domain in PFE */
        rtn = demo_l2_bd_add(p_cl, &bd, 200u);
        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data of the new domain */
            demo_l2_bd_ld_insert_phyif(&bd, 0u, true);  /* 0u == ID of emac0 */
            demo_l2_bd_ld_insert_phyif(&bd, 1u, true);  /* 1u == ID of emac1 */
            demo_l2_bd_ld_set_ucast_hit(&bd, 0u);   /* 0u == bridge action "FORWARD" */
            demo_l2_bd_ld_set_ucast_miss(&bd, 1u);  /* 1u == bridge action "FLOOD" */
            demo_l2_bd_ld_set_mcast_hit(&bd, 0u);   /* 0u == bridge action "FORWARD" */
            demo_l2_bd_ld_set_mcast_miss(&bd, 1u);  /* 1u == bridge action "FLOOD" */

            /* update the new bridge domain in PFE */
            rtn = demo_l2_bd_update(p_cl, &bd);
        }
    }
}


/* create and configure static MAC table entries */
/* ============================================= */
if (FPP_ERR_OK == rtn)
{
    fpp_l2_static_ent_cmd_t stent = {0};

    /* static entry for bridge domain 1 (MAC of PC0_NOVLAN) */
    /* ---------------------------------------------------- */
    if (FPP_ERR_OK == rtn)
    {
        /* create a new static entry in PFE */
        rtn = demo_l2_stent_add(p_cl, &stent, 1u,
                                (uint8_t[6]){0x0A,0x01,0x23,0x45,0x67,0x89});

        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data of the new static entry */
            /* 0u == ID of emac0 */
            demo_l2_stent_ld_set_fwlist(&stent, (1uL << 0u));

            /* update the new static entry in PFE */
            rtn = demo_l2_stent_update(p_cl, &stent);
        }
    }

    /* static entry for bridge domain 1 (MAC of PC1_NOVLAN) */
    /* ---------------------------------------------------- */
    if (FPP_ERR_OK == rtn)
    {
        /* create a new static entry in PFE */
        rtn = demo_l2_stent_add(p_cl, &stent, 1u,
                                (uint8_t[6]){0x0A,0xFE,0xDC,0xBA,0x98,0x76});

        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data of the new static entry */
            /* 1u == ID of emac1 */
            demo_l2_stent_ld_set_fwlist(&stent, (1uL << 1u));

            /* update the new static entry in PFE */
            rtn = demo_l2_stent_update(p_cl, &stent);
        }
    }


    /* static entry for bridge domain 100 (MAC of PC0_100) */
    /* --------------------------------------------------- */
    if (FPP_ERR_OK == rtn)
    {
        /* create a new static entry in PFE */
        rtn = demo_l2_stent_add(p_cl, &stent, 100u,
                                (uint8_t[6]){0x02,0x11,0x22,0x33,0x44,0x55});

        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data of the new static entry */
            /* 0u == ID of emac0 */
            demo_l2_stent_ld_set_fwlist(&stent, (1uL << 0u));

            /* update the new static entry in PFE */
```

```
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }

        /* static entry for bridge domain 100 (MAC of PC1_100) */
        /* ------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 100u,
                                    (uint8_t[6]){0x02,0x66,0x77,0x88,0x99,0xAA});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                /* 1u == ID of emac1 */
                demo_l2_stent_ld_set_fwlist(&stent, (1uL << 1u));

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }

        /* static entry for bridge domain 200 (MAC of PC0_200) */
        /* ------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 200u,
                                    (uint8_t[6]){0x06,0xCC,0xBB,0xAA,0x99,0x88});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                /* 0u == ID of emac0 ; 7u == hif1 */
                demo_l2_stent_ld_set_fwlist(&stent, ((1uL << 0u) | (1uL << 7u)));

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }

        /* static entry for bridge domain 200 (MAC of PC1_200) */
        /* ------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 200u,
                                    (uint8_t[6]){0x06,0x77,0x66,0x55,0x44,0x33});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                /* 1u == ID of emac1 ; 7u == hif1 */
                demo_l2_stent_ld_set_fwlist(&stent, ((1uL << 1u) | (1uL << 7u)));

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }
    }


    /* configure physical interfaces */
    /* ============================ */
    if (FPP_ERR_OK == rtn)
    {
        /* lock the interface database of PFE */
        rtn = demo_if_session_lock(p_cl);
        if (FPP_ERR_OK == rtn)
        {
            fpp_phy_if_cmd_t phyif = {0};

            /* configure physical interface "emac0" */
            /* ------------------------------------ */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac0");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_promisc(&phyif, true);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_VLAN_BRIDGE);
                    demo_phy_if_ld_set_block_state(&phyif, BS_FORWARD_ONLY);
```



```
                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }

            /* configure physical interface "emac1" */
            /* ----------------------------------- */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac1");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_promisc(&phyif, true);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_VLAN_BRIDGE);
                    demo_phy_if_ld_set_block_state(&phyif, BS_FORWARD_ONLY);

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }
        }

        /* unlock the interface database of PFE */
        rtn = demo_if_session_unlock(p_cl, rtn);
    }


    /* clear dynamic (learned) entries from L2 bridge MAC table */
    /* ======================================================== */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_l2_flush_learned(p_cl);
    }


    return (rtn);
}

/* ========================================================================= */
```

15.3 demo_feature_router_simple.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <stdbool.h>
```

```
#include <stdio.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"
#include "demo_rt_ct.h"


/*
 * @brief      Use libFCI to configure PFE as a simple router.
 * @details    Scenario description:
 *               [*] Let there be two computers (PCs): PC0_7 and PC1_11.
 *                   Each PC is in a different network subnet.
 *               [*] Use libFCI to configure PFE as a simple router, allowing ICMP (ping)
 *                   communication between PC0_7 and PC1_11.
 *             PC description:
 *               PC0_7:
 *                 --> IP  address: 10.7.0.2/24
 *                 --> MAC address: 0A:01:23:45:67:89
 *                     (this is just a demo MAC; real MAC of the real PC0_7 should be used)
 *                 --> Accessible via PFE's emac0 physical interface.
 *                 --> Configured to send 10.11.0.0 traffic to PFE's emac0.
 *               PC1_11:
 *                 --> IP  address: 10.11.0.5/24
 *                 --> MAC address: 0A:FE:DC:BA:98:76
 *                     (this is just a demo MAC; real MAC of the real PC1_11 should be used)
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Configured to send 10.7.0.0 traffic to PFE's emac1.
 *
 * @note       This code uses a suite of "demo_" functions. The "demo_" functions encapsulate
 *             manipulation of libFCI data structs and calls of libFCI functions.
 *             It is advised to inspect content of these "demo_" functions.
 *
 * @param[in]  p_cl         FCI client
 *                          To create a client, use libFCI function fci_open().
 * @return     FPP_ERR_OK : All FCI commands were successfully executed.
 *                          Router should be up and running.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_feature_router_simple(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = FPP_ERR_OK;


    /* clear all IPv4 routes and conntracks in PFE (not necessary, done for demo purposes) */
    /* ================================================================================== */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_rtct_reset_ip4(p_cl);
    }


    /* create routes */
    /* ============= */
    if (FPP_ERR_OK == rtn)
    {
        fpp_rt_cmd_t rt = {0};

        /* route 7 (route to PC0_7) */
        /* ------------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new route */
            demo_rt_ld_set_as_ip4(&rt);
            demo_rt_ld_set_dst_mac(&rt, (const uint8_t[6]){0x0A,0x01,0x23,0x45,0x67,0x89});
            demo_rt_ld_set_egress_phyif(&rt, "emac0");

            /* create a new route in PFE */
            rtn = demo_rt_add(p_cl, 7uL, &rt);
        }

        /* route 11 (route to PC1_11) */
        /* -------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new route */
            demo_rt_ld_set_as_ip4(&rt);
            demo_rt_ld_set_dst_mac(&rt, (const uint8_t[6]){0x0A,0xFE,0xDC,0xBA,0x98,0x76});
            demo_rt_ld_set_egress_phyif(&rt, "emac1");

            /* create a new route in PFE */
            rtn = demo_rt_add(p_cl, 11uL, &rt);
        }
```

```
    }


    /* set timeout for conntracks (not necessary; done for demo purposes) */
    /* ================================================================== */
    if (FPP_ERR_OK == rtn)
    {
        demo_ct_timeout_others(p_cl, 0xFFFFFFFFuL);  /* ping is ICMP, that is 'others' */
    }


    /* create conntracks */
    /* ================= */
    if (FPP_ERR_OK == rtn)
    {
        fpp_ct_cmd_t ct = {0};

        /* conntrack from PC0_7 to PC1_11 (and back) */
        /* ----------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new conntrack */
            /*  This conntrack is configured as a bi-directional conntrack.
                FCI command to create this conntrack results in two connections being
                created in PFE:
                    --> one for the "orig" direction
                    --> one for the "reply" direction
            */
            demo_ct_ld_set_protocol(&ct, 1u);  /* 1 == ICMP */
            demo_ct_ld_set_orig_dir(&ct,  0x0A070002u,0x0A0B0005u,0u,0u, 0u,11uL, false);
            demo_ct_ld_set_reply_dir(&ct, 0x0A0B0005u,0x0A070002u,0u,0u, 0u, 7uL, false);

            /* create a new conntrack in PFE */
            rtn = demo_ct_add(p_cl, &ct);
        }
    }


    /* configure physical interfaces */
    /* ============================= */
    if (FPP_ERR_OK == rtn)
    {
        /* lock the interface database of PFE */
        rtn = demo_if_session_lock(p_cl);
        if (FPP_ERR_OK == rtn)
        {
            fpp_phy_if_cmd_t phyif = {0};

            /* configure physical interface "emac0" */
            /* ------------------------------------ */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac0");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_promisc(&phyif, false);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_ROUTER);

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }

            /* configure physical interface "emac1" */
            /* ------------------------------------ */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac1");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_promisc(&phyif, false);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_ROUTER);

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }
        }

        /* unlock the interface database of PFE */
```

```
        rtn = demo_if_session_unlock(p_cl, rtn);
    }


    return (rtn);
}


/* ========================================================================= */
```

15.4 demo_feature_router_nat.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"
#include "demo_rt_ct.h"


/*
 * @brief      Use libFCI to configure PFE as a router (with one-to-many NAT).
 * @details    Scenario description:
 *               [*] Let there be three computers (PCs):
 *                     --> PC0_20, which acts as a server
 *                     --> PC1_2, which acts as a client
 *                     --> PC1_5, which acts as a client
 *               [*] Use libFCI to configure PFE as a router (with one-to-many NAT), allowing
 *                   TCP communication between the server PC and client PCs.
 *               [*] Client PCs can communicate with the server PC via TCP port 4000.
 *                   This scenario requires both source and destination port to be 4000.
 *                   (no use of ephemeral ports)
 *               [*] PC0_20 (server) has a public IP address (200.201.202.20/16).
 *               [*] PC1_2 and PC1_5 (clients) have private IP addresses from 10.x.x.x range.
 *                   They both share one public IP address (100.101.102.10/16) to communicate
 *                   with the outside world (NAT+PAT "one-to-many" mapping).
 *             PC description:
 *               PC0_20 (server):
 *                 --> IP address:  200.201.202.20/16
 *                 --> MAC address: 0A:BB:CC:DD:EE:FF
 *                     (this is just a demo MAC; real MAC of the real PC0 should be used)
 *                 --> Accessible via PFE's emac0 physical interface.
 *                 --> Configured to send 100.101.0.0 traffic to PFE's emac0.
 *                 --> Listens on TCP port 4000.
 *               PC1_2 (client_2):
```

```
 *                 --> IP address:  10.11.0.2/24
 *                 --> MAC address: 0A:11:33:55:77:99
 *                     (this is just a demo MAC; real MAC of the real PC1_2 should be used)
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Configured to send 200.201.0.0 traffic to PFE's emac1.
 *                 --> Hidden behind NAT.
 *               PC1_5 (client_5):
 *                 --> IP address: 10.11.0.5/24
 *                 --> MAC address: 0A:22:44:66:88:AA
 *                     (this is just a demo MAC; real MAC of the real PC1_5 should be used)
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Configured to send 200.201.0.0 traffic to PFE's emac1.
 *                 --> Hidden behind NAT.
 *             Additional info:
 *               [+] Conntrack struct has data members for an "orig" direction and for
 *                   a "reply" direction. See FPP_CMD_IPV4_CONNTRACK.
 *                   The "reply" direction data can be used for two purposes:
 *                     - To automatically create a reply direction conntrack together with
 *                       the orig direction conntrack in one FCI command.
 *                     - To modify parts of the "orig" direction packet (IPs/ports),
 *                       effectively creating NAT/PAT behavior.
 *
 * @note       This code uses a suite of "demo_" functions. The "demo_" functions encapsulate
 *             manipulation of libFCI data structs and calls of libFCI functions.
 *             It is advised to inspect content of these "demo_" functions.
 *
 * @param[in]  p_cl         FCI client
 *                          To create a client, use libFCI function fci_open().
 * @return     FPP_ERR_OK : All FCI commands were successfully executed.
 *                          Router should be up and running.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_feature_router_nat(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = FPP_ERR_OK;


    /* clear all IPv4 routes and conntracks in PFE (not necessary, done for demo purposes) */
    /* =================================================================================== */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_rtct_reset_ip4(p_cl);
    }


    /* create routes */
    /* ============= */
    if (FPP_ERR_OK == rtn)
    {
        fpp_rt_cmd_t rt = {0};

        /* route 20 (route to PC0_20) */
        /* -------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new route */
            demo_rt_ld_set_as_ip4(&rt);
            demo_rt_ld_set_dst_mac(&rt, (const uint8_t[6]){0x0A,0xBB,0xCC,0xDD,0xEE,0xFF});
            demo_rt_ld_set_egress_phyif(&rt, "emac0");

            /* create a new route in PFE */
            rtn = demo_rt_add(p_cl, 20uL, &rt);
        }

        /* route 2 (route to PC1_2) */
        /* ------------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new route */
            demo_rt_ld_set_as_ip4(&rt);
            demo_rt_ld_set_dst_mac(&rt, (const uint8_t[6]){0x0A,0x11,0x33,0x55,0x77,0x99});
            demo_rt_ld_set_egress_phyif(&rt, "emac1");

            /* create a new route in PFE */
            rtn = demo_rt_add(p_cl, 2uL, &rt);
        }

        /* route 5 (route to PC1_5) */
        /* ------------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new route */
            demo_rt_ld_set_as_ip4(&rt);
            demo_rt_ld_set_dst_mac(&rt, (const uint8_t[6]){0x0A,0x22,0x44,0x66,0x88,0xAA});
            demo_rt_ld_set_egress_phyif(&rt, "emac1");
```

```
            /* create a new route in PFE */
            rtn = demo_rt_add(p_cl, 5uL, &rt);
        }
    }


    /* set timeout for conntracks (not necessary; done for demo purposes) */
    /* ================================================================== */
    if (FPP_ERR_OK == rtn)
    {
        demo_ct_timeout_tcp(p_cl, 0xFFFFFFFFuL);
    }


    /* create conntracks between PC1_2 (client_2) and PC0_20 (server) */
    /* ============================================================== */
    if (FPP_ERR_OK == rtn)
    {
        fpp_ct_cmd_t ct = {0};

        /* from PC1_2 (client_2) to PC0_20 (server) */
        /* ---------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new conntrack */
            /*  This conntrack is configured as an unidirectional NAT/PAT conntrack.
                FCI command to create this conntrack results in one connection being
                created in PFE - a connection from PC1_2 to PC0_20 ("orig" direction only).
                Packets routed by this conntrack are modified by PFE as follows:
                    --> Source IP of the routed packet is replaced with the conntrack's
                        "reply" dir destination IP address (NAT behavior).
                    --> Source port of the routed packet is replaced with the conntrack's
                        "reply" dir destination port (PAT behavior).
            */
            demo_ct_ld_set_protocol(&ct, 6u);  /* 6 == TCP */
            demo_ct_ld_set_orig_dir(&ct, 0x0A0B0003u,0xC8C9CA14u,4000u,4000u, 0u,20uL, true);
            demo_ct_ld_set_reply_dir(&ct,0xC8C9CA14u,0x6465660Au,4000u,40003u,0u, 0uL, false);

            /* create a new conntrack in PFE */
            rtn = demo_ct_add(p_cl, &ct);
        }

        /* from PC0_20 (server) back to PC1_2 (client_2) */
        /* --------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new conntrack */
            /*  This conntrack is a complement to the previous one - it represents
                connection from PC0_20 back to PC1_2.
                Notice that this conntrack translates source IP / source port of
                the routed packet back to the values expected by the PC1_2.
            */
            demo_ct_ld_set_protocol(&ct, 6u);  /* 6 == TCP */
            demo_ct_ld_set_orig_dir(&ct, 0xC8C9CA14u,0x6465660Au,4000u,40003u,0u,2uL, true);
            demo_ct_ld_set_reply_dir(&ct,0x0A0B0003u,0xC8C9CA14u,4000u,4000u, 0u,0uL, false);

            /* create a new conntrack in PFE */
            rtn = demo_ct_add(p_cl, &ct);
        }
    }


    /* create conntracks between PC1_5 (client_5) and PC0_20 (server) */
    /* ============================================================== */
    if (FPP_ERR_OK == rtn)
    {
        fpp_ct_cmd_t ct = {0};

        /* from PC1_5 (client_5) to PC0_20 (server) */
        /* ---------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new conntrack */
            demo_ct_ld_set_protocol(&ct, 6u);  /* 6 == TCP */
            demo_ct_ld_set_orig_dir(&ct, 0x0A0B0005u,0xC8C9CA14u,4000u,4000u, 0u,20uL, true);
            demo_ct_ld_set_reply_dir(&ct,0xC8C9CA14u,0x6465660Au,4000u,40005u,0u, 0uL, false);

            /* create a new conntrack in PFE */
            rtn = demo_ct_add(p_cl, &ct);
        }

        /* from PC0_20 (server) back to PC1_5 (client_5) */
        /* --------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
```

```
            /* locally prepare data for a new conntrack */
            demo_ct_ld_set_protocol(&ct, 6u);  /* 6 == TCP */
            demo_ct_ld_set_orig_dir(&ct, 0xC8C9CA14u,0x6465660Au,4000u,40005u,0u,5uL, true);
            demo_ct_ld_set_reply_dir(&ct,0x0A0B0005u,0xC8C9CA14u,4000u,4000u, 0u,0uL, false);

            /* create a new conntrack in PFE */
            rtn = demo_ct_add(p_cl, &ct);
        }
    }


    /* configure physical interfaces */
    /* ============================ */
    if (FPP_ERR_OK == rtn)
    {
        /* lock the interface database of PFE */
        rtn = demo_if_session_lock(p_cl);
        if (FPP_ERR_OK == rtn)
        {
            fpp_phy_if_cmd_t phyif = {0};

            /* configure physical interface "emac0" */
            /* ------------------------------------ */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac0");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_promisc(&phyif, false);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_ROUTER);

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }

            /* configure physical interface "emac1" */
            /* ------------------------------------ */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac1");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_promisc(&phyif, false);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_ROUTER);

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }
        }

        /* unlock the interface database of PFE */
        rtn = demo_if_session_unlock(p_cl, rtn);
    }


    return (rtn);
}


/* ========================================================================= */
```

15.5 demo_feature_L2L3_bridge_vlan.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
```

```
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"
#include "demo_l2_bd.h"
#include "demo_rt_ct.h"


/*
 * @brief      Use libFCI to configure PFE as VLAN-aware L2L3 bridge.
 * @details    Scenario description:
 *               [*] Let there be four computers (PCs):
 *                     --> Two PCs (PC0_100 and PC0_200) are accessible via
 *                         PFE's emac0 physical interface.
 *                     --> Two PCs (PC1_100 and PC1_200) are accessible via
 *                         PFE's emac1 physical interface.
 *               [*] Use libFCI to configure PFE as VLAN-aware L2L3 bridge, allowing
 *                   communication between the PCs as follows:
 *                     --> PC0_100 and PC1_100 are both in the VLAN domain 100.
 *                         PFE shall operate as a VLAN-aware L2 bridge, allowing communication
 *                         between these two PCs.
 *                     --> PC0_200 and PC1_200 are both in the VLAN domain 200.
 *                         PFE shall operate as a VLAN-aware L2 bridge, allowing communication
 *                         between these two PCs.
 *                     --> PC0_100 and PC1_200 are in different VLAN domains.
 *                         PFE shall operate as a router, allowing ICMP (ping) and
 *                         TCP (port 4000) communication between these two PCs.
 *               [*] Additional requirements:
 *                     --> Dynamic learning of MAC addresses shall be disabled on
 *                         emac0 and emac1 interfaces.
 *             PFE emac description:
 *               emac0:
 *                 --> MAC address: 00:01:BE:BE:EF:11
 *               emac1:
 *                 --> MAC address: 00:01:BE:BE:EF:22
 *             PC description:
 *               PC0_100:
 *                 --> IP  address: 10.100.0.2/24
 *                 --> MAC address: 02:11:22:33:44:55
 *                 --> Accessible via PFE's emac0 physical interface.
 *                 --> Configured to send 10.200.0.0 traffic to PFE's emac0.
 *                 --> Belongs to VLAN 100 domain.
 *               PC1_100:
 *                 --> IP  address: 10.100.0.5/24
 *                 --> MAC address: 02:66:77:88:99:AA
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Belongs to VLAN 100 domain.
 *               PC0_200:
 *                 --> IP  address: 10.200.0.2/24
 *                 --> MAC address: 06:CC:BB:AA:99:88
 *                 --> Accessible via PFE's emac0 physical interface.
 *                 --> Belongs to VLAN 200 domain.
 *               PC1_200:
 *                 --> IP  address: 10.200.0.5/24
 *                 --> MAC address: 06:77:66:55:44:33
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Configured to send 10.100.0.0 traffic to PFE's emac1.
 *                 --> Belongs to VLAN 200 domain.
 *
```

```
 * @note       This code uses a suite of "demo_" functions. The "demo_" functions encapsulate
 *             manipulation of libFCI data structs and calls of libFCI functions.
 *             It is advised to inspect content of these "demo_" functions.
 *
 * @param[in]  p_cl         FCI client
 *                          To create a client, use libFCI function fci_open().
 * @return     FPP_ERR_OK : All FCI commands were successfully executed.
 *                          L2L3 bridge should be up and running.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_feature_L2L3_bridge_vlan(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = FPP_ERR_OK;


    /*
        configure VLAN-aware L2 bridge
    */


    /* clear L2 bridge MAC table (not required; done for demo purposes) */
    /* ============================================================== */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_l2_flush_all(p_cl);
    }


    /* create and configure bridge domains */
    /* ================================= */
    if (FPP_ERR_OK == rtn)
    {
        fpp_l2_bd_cmd_t bd = {0};

        /* bridge domain 100 */
        /* ----------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new bridge domain in PFE */
            rtn = demo_l2_bd_add(p_cl, &bd, 100u);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new domain */
                demo_l2_bd_ld_insert_phyif(&bd, 0u, true);  /* 0u == ID of emac0 */
                demo_l2_bd_ld_insert_phyif(&bd, 1u, true);  /* 1u == ID of emac1 */
                demo_l2_bd_ld_set_ucast_hit(&bd, 0u);   /* 0u == bridge action "FORWARD" */
                demo_l2_bd_ld_set_ucast_miss(&bd, 1u);  /* 1u == bridge action "FLOOD" */
                demo_l2_bd_ld_set_mcast_hit(&bd, 0u);   /* 0u == bridge action "FORWARD" */
                demo_l2_bd_ld_set_mcast_miss(&bd, 1u);  /* 1u == bridge action "FLOOD" */

                /* update the new bridge domain in PFE */
                rtn = demo_l2_bd_update(p_cl, &bd);
            }
        }

        /* bridge domain 200 */
        /* ----------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new bridge domain in PFE */
            rtn = demo_l2_bd_add(p_cl, &bd, 200u);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new domain */
                demo_l2_bd_ld_insert_phyif(&bd, 0u, true);  /* 0u == ID of emac0 */
                demo_l2_bd_ld_insert_phyif(&bd, 1u, true);  /* 1u == ID of emac1 */
                demo_l2_bd_ld_set_ucast_hit(&bd, 0u);   /* 0u == bridge action "FORWARD" */
                demo_l2_bd_ld_set_ucast_miss(&bd, 1u);  /* 1u == bridge action "FLOOD" */
                demo_l2_bd_ld_set_mcast_hit(&bd, 0u);   /* 0u == bridge action "FORWARD" */
                demo_l2_bd_ld_set_mcast_miss(&bd, 1u);  /* 1u == bridge action "FLOOD" */

                /* update the new bridge domain in PFE */
                rtn = demo_l2_bd_update(p_cl, &bd);
            }
        }
    }


    /* create and configure static MAC table entries */
    /* =========================================== */
    if (FPP_ERR_OK == rtn)
    {
        fpp_l2_static_ent_cmd_t stent = {0};

        /* static entry for bridge domain 100 (MAC of PC0_100) */
```

```
        /* -------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 100u,
                                    (uint8_t[6]){0x02,0x11,0x22,0x33,0x44,0x55});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                /* 0u == ID of emac0 */
                demo_l2_stent_ld_set_fwlist(&stent, (1uL << 0u));

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }

        /* static entry for bridge domain 100 (MAC of PC1_100) */
        /* -------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 100u,
                                    (uint8_t[6]){0x02,0x66,0x77,0x88,0x99,0xAA});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                /* 1u == ID of emac1 */
                demo_l2_stent_ld_set_fwlist(&stent, (1uL << 1u));

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }

        /* static entry for bridge domain 200 (MAC of PC0_200) */
        /* -------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 200u,
                                    (uint8_t[6]){0x06,0xCC,0xBB,0xAA,0x99,0x88});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                /* 0u == ID of emac0 */
                demo_l2_stent_ld_set_fwlist(&stent, (1uL << 0u));

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }

        /* static entry for bridge domain 200 (MAC of PC1_200) */
        /* -------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 200u,
                                    (uint8_t[6]){0x06,0x77,0x66,0x55,0x44,0x33});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                /* 1u == ID of emac1 */
                demo_l2_stent_ld_set_fwlist(&stent, (1uL << 1u));

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }
    }


    /* create special 'local' static MAC table entries (required for L2L3 bridge) */
    /* ========================================================================== */
    /* 'local' static MAC table entries are used to select the traffic which should be
       classified by the Router. The rest of the traffic is classified by the L2 bridge. */
    if (FPP_ERR_OK == rtn)
    {
        fpp_l2_static_ent_cmd_t stent = {0};

        /* [vlan 100] ; if traffic destination MAC == MAC of emac0, then pass it to Router */
```

```
        /* ------------------------------------------------------------------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 100u,
                                    (uint8_t[6]){0x00,0x01,0xBE,0xBE,0xEF,0x11});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                demo_l2_stent_ld_set_local(&stent, true);

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }

        /* [vlan 100] ; if traffic destination MAC == MAC of emac1, then pass it to Router */
        /* ------------------------------------------------------------------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 100u,
                                    (uint8_t[6]){0x00,0x01,0xBE,0xBE,0xEF,0x22});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                demo_l2_stent_ld_set_local(&stent, true);

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }

        /* [vlan 200] ; if traffic destination MAC == MAC of emac0, then pass it to Router */
        /* ------------------------------------------------------------------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 200u,
                                    (uint8_t[6]){0x00,0x01,0xBE,0xBE,0xEF,0x11});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                demo_l2_stent_ld_set_local(&stent, true);

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }

        /* [vlan 200] ; if traffic destination MAC == MAC of emac1, then pass it to Router */
        /* ------------------------------------------------------------------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* create a new static entry in PFE */
            rtn = demo_l2_stent_add(p_cl, &stent, 200u,
                                    (uint8_t[6]){0x00,0x01,0xBE,0xBE,0xEF,0x22});

            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data of the new static entry */
                demo_l2_stent_ld_set_local(&stent, true);

                /* update the new static entry in PFE */
                rtn = demo_l2_stent_update(p_cl, &stent);
            }
        }
    }


    /*
        configure router
    */


    /* clear all IPv4 routes and conntracks in PFE (not necessary, done for demo purposes) */
    /* ================================================================================== */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_rtct_reset_ip4(p_cl);
    }
```

```
    /* create routes */
    /* ============= */
    if (FPP_ERR_OK == rtn)
    {
        fpp_rt_cmd_t rt = {0};

        /* route 10 (route to PC0_100) */
        /* --------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new route */
            demo_rt_ld_set_as_ip4(&rt);
            demo_rt_ld_set_dst_mac(&rt, (const uint8_t[6]){0x02,0x11,0x22,0x33,0x44,0x55});
            demo_rt_ld_set_egress_phyif(&rt, "emac0");

            /* create a new route in PFE */
            rtn = demo_rt_add(p_cl, 10uL, &rt);
        }

        /* route 20 (route to PC1_200) */
        /* --------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new route */
            demo_rt_ld_set_as_ip4(&rt);
            demo_rt_ld_set_dst_mac(&rt, (const uint8_t[6]){0x06,0x77,0x66,0x55,0x44,0x33});
            demo_rt_ld_set_egress_phyif(&rt, "emac1");

            /* create a new route in PFE */
            rtn = demo_rt_add(p_cl, 20uL, &rt);
        }
    }


    /* set timeout for conntracks (not necessary; done for demo purposes) */
    /* ================================================================== */
    if (FPP_ERR_OK == rtn)
    {
        demo_ct_timeout_others(p_cl, 0xFFFFFFFFuL);  /* ping is ICMP, that is 'others' */
        demo_ct_timeout_tcp(p_cl, 0xFFFFFFFFuL);
    }


    /* create conntracks */
    /* ================= */
    if (FPP_ERR_OK == rtn)
    {
        fpp_ct_cmd_t ct = {0};

        /* ICMP conntrack from PC0_100 to PC1_200 (and back) */
        /* ------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new conntrack */
            /*      This conntrack is configured as a bi-directional conntrack.
                    One FCI command results in two connections being created in PFE -
                    one for the "orig" direction and one for the "reply" direction.
                    This conntrack also modifies VLAN tag of the routed packet.
            */
            demo_ct_ld_set_protocol(&ct, 1u);  /* 1 == ICMP */
            demo_ct_ld_set_orig_dir(&ct, 0x0A640002u,0x0AC80005u,0u,0u,200u,20uL,false);
            demo_ct_ld_set_reply_dir(&ct,0x0AC80005u,0x0A640002u,0u,0u,100u,10uL,false);

            /* create a new conntrack in PFE */
            rtn = demo_ct_add(p_cl, &ct);
        }

        /* TCP conntrack from PC0_100 to PC1_200 (and back) */
        /* ------------------------------------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new conntrack */
            /*      This conntrack is configured as a bi-directional conntrack.
                    One FCI command results in two connections being created in PFE -
                    one for the "orig" direction and one for the "reply" direction.
                    This conntrack also modifies VLAN tag of the routed packet.
            */
            demo_ct_ld_set_protocol(&ct, 6u);  /* 6 == TCP */
            demo_ct_ld_set_orig_dir(&ct, 0x0A640002u,0x0AC80005u,4000u,4000u,200u,20uL,false);
            demo_ct_ld_set_reply_dir(&ct,0x0AC80005u,0x0A640002u,4000u,4000u,100u,10uL,false);

            /* create a new conntrack in PFE */
            rtn = demo_ct_add(p_cl, &ct);
        }
    }
```

```
    /*
        configure physical interfaces
    */


    /* configure physical interfaces */
    /* ============================ */
    if (FPP_ERR_OK == rtn)
    {
        /* lock the interface database of PFE */
        rtn = demo_if_session_lock(p_cl);
        if (FPP_ERR_OK == rtn)
        {
            fpp_phy_if_cmd_t phyif = {0};

            /* configure physical interface "emac0" */
            /* ------------------------------------ */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac0");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_promisc(&phyif, true);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_L2L3_VLAN_BRIDGE);
                    demo_phy_if_ld_set_block_state(&phyif, BS_FORWARD_ONLY);

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }

            /* configure physical interface "emac1" */
            /* ------------------------------------ */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac1");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_promisc(&phyif, true);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_L2L3_VLAN_BRIDGE);
                    demo_phy_if_ld_set_block_state(&phyif, BS_FORWARD_ONLY);

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }
        }

        /* unlock the interface database of PFE */
        rtn = demo_if_session_unlock(p_cl, rtn);
    }


    return (rtn);
}


/* ========================================================================= */
```

15.6 demo_feature_flexible_filter.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
```

```
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"
#include "demo_fp.h"


extern int demo_feature_L2_bridge_vlan(FCI_CLIENT* p_cl);


/*
 * @brief      Use libFCI to configure a Flexible Filter in PFE.
 * @details    Scenario description:
 *               [*] Let there be two computers (PCs), both in the same network subnet.
 *                   Both PCs are connected through PFE. PFE acts as a simple bridge.
 *               [*] Use libFCI to configure a Flexible Filter on PFE's emac0 physical
 *                   interface, allowing only a specific type of ingress traffic to pass
 *                   for further classification. Non-compliant traffic is discarded.
 *               [*] Criteria for the allowed ingress traffic on PFE's emac0:
 *                     --> Type of the traffic is either ARP or ICMP.
 *                     --> Source IP address is always the IP address of the PC0.
 *                     --> Destination IP address is always the IP address of the PC1.
 *             PC description:
 *               PC0:
 *                 --> IP address: 10.3.0.2/24
 *                 --> Accessible via PFE's emac0 physical interface.
 *                 --> Has static ARP entry for PC1.
 *               PC1:
 *                 --> IP address: 10.3.0.5/24
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Has static ARP entry for PC0.
 *             Additional info:
 *               Pseudocode of the comparison process done by this demo's FP table:
 *               [0] r_arp_ethtype : (ethtype != ARP)  ? (GOTO r_icmp_ethtype) : (next_line)
 *               [1] r_arp_sip     : (sip != 10.3.0.2) ? (REJECT)              : (next_line)
 *               [2] r_arp_dip     : (dip == 10.3.0.5) ? (ACCEPT)              : (next_line)
 *               [3] r_arp_discard : (true)            ? (REJECT)              : (REJECT)
 *               [4] r_icmp_ethtype: (ethtype != IPv4) ? (REJECT)              : (next_line)
 *               [5] r_icmp_proto  : (proto != ICMP)   ? (REJECT)              : (next_line)
 *               [6] r_icmp_sip    : (sip != 10.3.0.2) ? (REJECT)              : (next_line)
 *               [7] r_icmp_dip    : (sip == 10.3.0.5) ? (ACCEPT)              : (next_line)
 *               [8] r_icmp_discard: (true)            ? (REJECT)              : (REJECT)
 *
 * @note       This code uses a suite of "demo_" functions. The "demo_" functions encapsulate
 *             manipulation of libFCI data structs and calls of libFCI functions.
 *             It is advised to inspect content of these "demo_" functions.
 *
 * @param[in]  p_cl         FCI client
 *                          To create a client, use libFCI function fci_open().
 * @return     FPP_ERR_OK : All FCI commands were successfully executed.
 *                          Flexible Parser table should be set in PFE.
 *                          Flexible Filter on PFE's emac0 should be up and running.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_feature_flexible_filter(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = FPP_ERR_OK;


    /* setup PFE to classify traffic (not needed by Flexible Filter, done for demo purposes)*/
    /* ==================================================================================== */
```



```
    rtn = demo_feature_L2_bridge_vlan(p_cl);


    /* create FP rules */
    /* =============== */
    if (FPP_ERR_OK == rtn)
    {
        fpp_fp_rule_cmd_t rule = {0};

        /* rule [0] */
        /* -------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new rule */
            demo_fp_rule_ld_set_data(&rule, 0x08060000);  /* 0x0806 == EtherType for ARP */
            demo_fp_rule_ld_set_mask(&rule, 0xFFFF0000);
            demo_fp_rule_ld_set_offset(&rule, 12u, FP_OFFSET_FROM_L2_HEADER);
            demo_fp_rule_ld_set_invert(&rule, true);
            demo_fp_rule_ld_set_match_action(&rule, FP_NEXT_RULE, "r_icmp_ethtype");

            /* create a new rule in PFE */
            rtn = demo_fp_rule_add(p_cl, "r_arp_ethtype", &rule);
        }

        /* rule [1] */
        /* -------- */
        if (FPP_ERR_OK == rtn)
        {
            demo_fp_rule_ld_set_data(&rule, 0x0A030002);  /* ARP protocol: sender IP */
            demo_fp_rule_ld_set_mask(&rule, 0xFFFFFFFF);
            demo_fp_rule_ld_set_offset(&rule, 28u, FP_OFFSET_FROM_L2_HEADER);
            demo_fp_rule_ld_set_invert(&rule, true);
            demo_fp_rule_ld_set_match_action(&rule, FP_REJECT, NULL);

            rtn = demo_fp_rule_add(p_cl, "r_arp_sip", &rule);
        }

        /* rule [2] */
        /* -------- */
        if (FPP_ERR_OK == rtn)
        {
            demo_fp_rule_ld_set_data(&rule, 0x0A030005);  /* ARP protocol: target IP */
            demo_fp_rule_ld_set_mask(&rule, 0xFFFFFFFF);
            demo_fp_rule_ld_set_offset(&rule, 38u, FP_OFFSET_FROM_L2_HEADER);
            demo_fp_rule_ld_set_invert(&rule, false);
            demo_fp_rule_ld_set_match_action(&rule, FP_ACCEPT, NULL);

            rtn = demo_fp_rule_add(p_cl, "r_arp_dip", &rule);
        }

        /* rule [3] */
        /* -------- */
        if (FPP_ERR_OK == rtn)
        {
            demo_fp_rule_ld_set_data(&rule, 0x00);
            demo_fp_rule_ld_set_mask(&rule, 0x00);
            demo_fp_rule_ld_set_offset(&rule, 0u, FP_OFFSET_FROM_L2_HEADER);
            demo_fp_rule_ld_set_invert(&rule, false);
            demo_fp_rule_ld_set_match_action(&rule, FP_REJECT, NULL);

            rtn = demo_fp_rule_add(p_cl, "r_arp_discard", &rule);
        }

        /* rule [4] */
        /* -------- */
        if (FPP_ERR_OK == rtn)
        {
            demo_fp_rule_ld_set_data(&rule, 0x08000000);  /* 0x0800 == EtherType for IPv4 */
            demo_fp_rule_ld_set_mask(&rule, 0xFFFF0000);
            demo_fp_rule_ld_set_offset(&rule, 12u, FP_OFFSET_FROM_L2_HEADER);
            demo_fp_rule_ld_set_invert(&rule, true);
            demo_fp_rule_ld_set_match_action(&rule, FP_REJECT, NULL);

            rtn = demo_fp_rule_add(p_cl, "r_icmp_ethtype", &rule);
        }

        /* rule [5] */
        /* -------- */
        if (FPP_ERR_OK == rtn)
        {
            demo_fp_rule_ld_set_data(&rule, 0x01000000);  /* 0x01 == ICMP protocol type */
            demo_fp_rule_ld_set_mask(&rule, 0xFF000000);
            demo_fp_rule_ld_set_offset(&rule, 9u, FP_OFFSET_FROM_L3_HEADER);  /* from L3 */
            demo_fp_rule_ld_set_invert(&rule, true);
            demo_fp_rule_ld_set_match_action(&rule, FP_REJECT, NULL);
```

```
            rtn = demo_fp_rule_add(p_cl, "r_icmp_proto", &rule);
        }

        /* rule [6] */
        /* -------- */
        if (FPP_ERR_OK == rtn)
        {
            demo_fp_rule_ld_set_data(&rule, 0x0A030002);  /* IP protocol: source IP */
            demo_fp_rule_ld_set_mask(&rule, 0xFFFFFFFF);
            demo_fp_rule_ld_set_offset(&rule, 12u, FP_OFFSET_FROM_L3_HEADER);  /* from L3 */
            demo_fp_rule_ld_set_invert(&rule, true);
            demo_fp_rule_ld_set_match_action(&rule, FP_REJECT, NULL);

            rtn = demo_fp_rule_add(p_cl, "r_icmp_sip", &rule);
        }

        /* rule [7] */
        /* -------- */
        if (FPP_ERR_OK == rtn)
        {
            demo_fp_rule_ld_set_data(&rule, 0x0A030005);  /* IP protocol: destination IP */
            demo_fp_rule_ld_set_mask(&rule, 0xFFFFFFFF);
            demo_fp_rule_ld_set_offset(&rule, 16u, FP_OFFSET_FROM_L3_HEADER);  /* from L3 */
            demo_fp_rule_ld_set_invert(&rule, false);
            demo_fp_rule_ld_set_match_action(&rule, FP_ACCEPT, NULL);

            rtn = demo_fp_rule_add(p_cl, "r_icmp_dip", &rule);
        }

        /* rule [8] */
        /* -------- */
        if (FPP_ERR_OK == rtn)
        {
            demo_fp_rule_ld_set_data(&rule, 0x00);
            demo_fp_rule_ld_set_mask(&rule, 0x00);
            demo_fp_rule_ld_set_offset(&rule, 0u, FP_OFFSET_FROM_L3_HEADER);
            demo_fp_rule_ld_set_invert(&rule, false);
            demo_fp_rule_ld_set_match_action(&rule, FP_REJECT, NULL);

            rtn = demo_fp_rule_add(p_cl, "r_icmp_discard", &rule);
        }
    }


    /* create (and fill) FP table */
    /* ========================== */
    if (FPP_ERR_OK == rtn)
    {
        /* create FP table */
        /* --------------- */
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_fp_table_add(p_cl, "my_filter_table");
        }

        /* fill the table with rules */
        /* ------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_fp_table_insert_rule(p_cl, "my_filter_table", "r_arp_ethtype",  0u);
        }
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_fp_table_insert_rule(p_cl, "my_filter_table", "r_arp_sip", 1u);
        }
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_fp_table_insert_rule(p_cl, "my_filter_table", "r_arp_dip", 2u);
        }
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_fp_table_insert_rule(p_cl, "my_filter_table", "r_arp_discard", 3u);
        }
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_fp_table_insert_rule(p_cl, "my_filter_table", "r_icmp_ethtype", 4u);
        }
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_fp_table_insert_rule(p_cl, "my_filter_table", "r_icmp_proto", 5u);
        }
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_fp_table_insert_rule(p_cl, "my_filter_table", "r_icmp_sip", 6u);
        }
        if (FPP_ERR_OK == rtn)
```

```
        {
            rtn = demo_fp_table_insert_rule(p_cl, "my_filter_table", "r_icmp_dip", 7u);
        }
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_fp_table_insert_rule(p_cl, "my_filter_table", "r_icmp_discard", 8u);
        }
    }


    /* assign the created FP table as a Flexible Filter for emac0 */
    /* ========================================================== */
    if (FPP_ERR_OK == rtn)
    {
        /* lock the interface database of PFE */
        rtn = demo_if_session_lock(p_cl);
        if (FPP_ERR_OK == rtn)
        {
            fpp_phy_if_cmd_t phyif = {0};

            /* get data from PFE and store them in the local variable "phyif" */
            rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac0");
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_phy_if_ld_set_flexifilter(&phyif, "my_filter_table");

                /* update data in PFE */
                rtn = demo_phy_if_update(p_cl, &phyif);
            }
        }

        /* unlock the interface database of PFE */
        rtn = demo_if_session_unlock(p_cl, rtn);
    }


    return (rtn);
}


/* ========================================================================= */
```

15.7 demo_feature_flexible_router.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include "fpp.h"
```

```
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"
#include "demo_log_if.h"


/*
 * @brief      Use libFCI to configure PFE as a Flexible Router.
 * @details    Scenario description:
 *               [*] Let there be two computers (PCs).
 *                   Each PC is in a different network subnet.
 *               [*] Use libFCI to configure PFE as a Flexible Router, allowing the PCs
 *                   to communicate with each other.
 *               [*] Only a specific traffic is allowed through PFE (the rest is discarded).
 *                   Criteria for the allowed traffic:
 *                     --> Only ARP and ICMP traffic is allowed through PFE.
 *                     --> No further limitations for ARP traffic.
 *                     --> For ICMP traffic, only IPs of PC0 and PC1 are allowed
 *                         to communicate with each other. ICMP traffic from
 *                         any other IP must be blocked.
 *                     --> EXTRA: All traffic which passes through PFE must also be mirrored
 *                                to the emac2 physical interface.
 *               [*] NOTE:
 *                   Flexible Router is best used for special, non-standard requirements.
 *                   Scanning of traffic data and chaining of logical interfaces presents
 *                   an additional overhead.
 *                   PFE features such as L2 bridge or L3 router offer a better performance
 *                   and are recommended over the Flexible Router in all cases where
 *                   they can be used to satisfy the given requirements.
 *             PC description:
 *               PC0:
 *                 --> IP address: 10.7.0.2/24
 *                 --> Accessible via PFE's emac0 physical interface.
 *                 --> Configured to send 10.11.0.0 traffic to PFE's emac0.
 *               PC1:
 *                 --> IP address: 10.11.0.5/24
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Configured to send 10.7.0.0 traffic to PFE's emac1.
 *
 * @note       This code uses a suite of "demo_" functions. The "demo_" functions encapsulate
 *             manipulation of libFCI data structs and calls of libFCI functions.
 *             It is advised to inspect content of these "demo_" functions.
 *
 * @param[in]  p_cl         FCI client
 *                          To create a client, use libFCI function fci_open().
 * @return     FPP_ERR_OK : All FCI commands were successfully executed.
 *                          Flexible Router should be up and running.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_feature_flexible_router(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = FPP_ERR_OK;


    /* lock the interface database of PFE */
    rtn = demo_if_session_lock(p_cl);


    /* create and configure logical interfaces on emac0 */
    /* ============================================== */
    /* NOTE: creation order of logical interfaces is IMPORTANT */
    if (FPP_ERR_OK == rtn)
    {
        fpp_log_if_cmd_t logif = {0};

        /* create a "sinkhole" logical interface for unsuitable ingress traffic */
        /* -------------------------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create new logical interface in PFE and store a copy of its data in "logif" */
            rtn = demo_log_if_add(p_cl, &logif, "MyLogif0_sink", "emac0");
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_log_if_ld_set_promisc(&logif, true);  /* promisc == accept everything */
                demo_log_if_ld_set_discard_on_m(&logif, true);
                demo_log_if_ld_enable(&logif);

                /* update data in PFE */
                rtn = demo_log_if_update(p_cl, &logif);
            }
        }
```

```
        /* create and configure a logical interface for ARP ingress traffic */
        /* ---------------------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_log_if_add(p_cl, &logif, "MyLogif0_arp", "emac0");
            if (FPP_ERR_OK == rtn)
            {
                /* NOTE: 1u == ID of emac1 ; 2u == ID of emac2 */
                demo_log_if_ld_set_promisc(&logif, false);
                demo_log_if_ld_set_egress_phyifs(&logif, ((1uL << 1u) | (1uL << 2u)));
                demo_log_if_ld_set_match_mode_or(&logif, false);
                demo_log_if_ld_clear_all_mr(&logif);
                demo_log_if_ld_set_mr_type_arp(&logif, true);
                demo_log_if_ld_enable(&logif);

                rtn = demo_log_if_update(p_cl, &logif);
            }
        }

        /* create and configure a logical interface for ICMP ingress traffic */
        /* ----------------------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_log_if_add(p_cl, &logif, "MyLogif0_icmp", "emac0");
            if (FPP_ERR_OK == rtn)
            {
                /* NOTE: 1u == ID of emac1 ; 2u == ID of emac2 */
                demo_log_if_ld_set_promisc(&logif, false);
                demo_log_if_ld_set_egress_phyifs(&logif, ((1uL << 1u) | (1uL << 2u)));
                demo_log_if_ld_set_match_mode_or(&logif, false);
                demo_log_if_ld_clear_all_mr(&logif);
                demo_log_if_ld_set_mr_type_icmp(&logif, true);
                demo_log_if_ld_set_mr_sip(&logif, true, 0x0A070002);
                demo_log_if_ld_set_mr_dip(&logif, true, 0x0A0B0005);
                demo_log_if_ld_enable(&logif);

                rtn = demo_log_if_update(p_cl, &logif);
            }
        }
    }


    /* create and configure logical interfaces on emac1 */
    /* ================================================ */
    /* NOTE: creation order of logical interfaces is IMPORTANT */
    if (FPP_ERR_OK == rtn)
    {
        fpp_log_if_cmd_t logif = {0};

        /* create a "sinkhole" logical interface for unsuitable ingress traffic */
        /* -------------------------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* create new logical interface in PFE and store a copy of its data in "logif" */
            rtn = demo_log_if_add(p_cl, &logif, "MyLogif1_sink", "emac1");
            if (FPP_ERR_OK == rtn)
            {
                demo_log_if_ld_set_promisc(&logif, true);  /* promisc == accept everything */
                demo_log_if_ld_set_discard_on_m(&logif, true);
                demo_log_if_ld_enable(&logif);

                rtn = demo_log_if_update(p_cl, &logif);
            }
        }


        /* create and configure a logical interface for ARP ingress traffic */
        /* ---------------------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_log_if_add(p_cl, &logif, "MyLogif1_arp", "emac1");
            if (FPP_ERR_OK == rtn)
            {
                /* NOTE: 0u == ID of emac0 ; 2u == ID of emac2 */
                demo_log_if_ld_set_promisc(&logif, false);
                demo_log_if_ld_set_egress_phyifs(&logif, ((1uL << 0u) | (1uL << 2u)));
                demo_log_if_ld_set_match_mode_or(&logif, false);
                demo_log_if_ld_clear_all_mr(&logif);
                demo_log_if_ld_set_mr_type_arp(&logif, true);
                demo_log_if_ld_enable(&logif);

                rtn = demo_log_if_update(p_cl, &logif);
            }
        }

        /* create and configure a logical interface for ICMP ingress traffic */
```

```
        /* ----------------------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            rtn = demo_log_if_add(p_cl, &logif, "MyLogif1_icmp", "emac1");
            if (FPP_ERR_OK == rtn)
            {
                /* NOTE: 0u == ID of emac0 ; 2u == ID of emac2 */
                demo_log_if_ld_set_promisc(&logif, false);
                demo_log_if_ld_set_egress_phyifs(&logif, ((1uL << 0u) | (1uL << 2u)));
                demo_log_if_ld_set_match_mode_or(&logif, false);
                demo_log_if_ld_clear_all_mr(&logif);
                demo_log_if_ld_set_mr_type_icmp(&logif, true);
                demo_log_if_ld_set_mr_sip(&logif, true, 0x0A0B0005);
                demo_log_if_ld_set_mr_dip(&logif, true, 0x0A070002);
                demo_log_if_ld_enable(&logif);

                rtn = demo_log_if_update(p_cl, &logif);
            }
        }
    }


    /* configure physical interfaces */
    /* ============================ */
    if (FPP_ERR_OK == rtn)
    {
        fpp_phy_if_cmd_t phyif = {0};

        /* configure physical interface "emac0" */
        /* ----------------------------------  */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "phyif" */
            rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac0");
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_phy_if_ld_enable(&phyif);
                demo_phy_if_ld_set_promisc(&phyif, true);
                demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_FLEXIBLE_ROUTER);

                /* update data in PFE */
                rtn = demo_phy_if_update(p_cl, &phyif);
            }
        }

        /* configure physical interface "emac1" */
        /* ----------------------------------  */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "phyif" */
            rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac1");
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_phy_if_ld_enable(&phyif);
                demo_phy_if_ld_set_promisc(&phyif, true);
                demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_FLEXIBLE_ROUTER);

                /* update data in PFE */
                rtn = demo_phy_if_update(p_cl, &phyif);
            }
        }
    }


    /* unlock the interface database of PFE */
    rtn = demo_if_session_unlock(p_cl, rtn);


    return (rtn);
}


/* ========================================================================= */
```

15.8 demo_feature_spd.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
```



```
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"
#include "demo_log_if.h"
#include "demo_spd.h"


/*
 * @brief      Use libFCI to configure PFE IPsec support.
 * @details    Scenario description:
 *               [*] Let there be two computers (PCs):
 *                     --> PC0, which uses encrypted communication.
 *                     --> PC1, which uses unencrypted communication.
 *               [*] Use libFCI to configure PFE IPsec support, allowing ICMP (ping) and
 *                   TCP (port 4000) communication between PC0 and PC1.
 *                     --> Traffic from PC0 should be decrypted by PFE, then sent to PC1.
 *                     --> Traffic from PC1 should be encrypted by PFE, then sent to PC0.
 *               [*] NOTE:
 *                   To fully enable PFE IPsec support, it is required to configure
 *                   the underlying HSE (Hardware Security Engine). HSE configuration
 *                   is not done by the FCI API and is outside the scope of this demo.
 *             PC description:
 *               PC0:
 *                 --> IP address: 10.7.0.2/24
 *                 --> Accessible via PFE's emac0 physical interface.
 *                 --> Configured to send 10.11.0.0 traffic to PFE's emac0.
 *                 --> Requires IPsec-encrypted communication.
 *               PC1:
 *                 --> IP address: 10.11.0.5/24
 *                 --> Accessible via PFE's emac1 physical interface.
 *                 --> Configured to send 10.7.0.0 traffic to PFE's emac1.
 *
 * @note       This code uses a suite of "demo_" functions. The "demo_" functions encapsulate
 *             manipulation of libFCI data structs and calls of libFCI functions.
 *             It is advised to inspect content of these "demo_" functions.
 *
 * @param[in]  p_cl         FCI client
 *                          To create a client, use libFCI function fci_open().
 * @return     FPP_ERR_OK : All FCI commands were successfully executed.
 *                          IPsec support should be up and running.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_feature_spd(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = FPP_ERR_OK;


    /* configure SPD database entries on emac0 */
    /* ====================================== */
    if (FPP_ERR_OK == rtn)
```

```
    {
        fpp_spd_cmd_t spd = {0};
        uint32_t src_ip[4] = {0};
        uint32_t dst_ip[4] = {0};

        /* create SPD entry for ICMP traffic (ping) from PC0 to PC1 */
        /* ----------------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new SPD entry */
            /* SPI passed in the demo_spd_ld_set_action() should be known by HSE */
            src_ip[0] = 0x0A070002;
            dst_ip[0] = 0x0A0B0005;
            demo_spd_ld_set_protocol(&spd, 1u);  /* 1 == ICMP */
            demo_spd_ld_set_ip(&spd, src_ip, dst_ip, false);
            demo_spd_ld_set_port(&spd, false, 0u, false, 0u);
            demo_spd_ld_set_action(&spd, FPP_SPD_ACTION_PROCESS_DECODE, 0u, 0x11335577);

            /* create a new SPD entry in PFE */
            rtn = demo_spd_add(p_cl, "emac0", 0u, &spd);
        }

        /* create SPD entry for TCP traffic from PC0 to PC1 */
        /* ------------------------------------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new SPD entry */
            /* SPI passed in the demo_spd_ld_set_action() should be known by HSE */
            src_ip[0] = 0x0A070002;
            dst_ip[0] = 0x0A0B0005;
            demo_spd_ld_set_protocol(&spd, 6u);  /* 6 == TCP */
            demo_spd_ld_set_ip(&spd, src_ip, dst_ip, false);
            demo_spd_ld_set_port(&spd, true, 4000u, true, 4000u);
            demo_spd_ld_set_action(&spd, FPP_SPD_ACTION_PROCESS_DECODE, 0u, 0x22446688);

            /* create a new SPD entry in PFE */
            rtn = demo_spd_add(p_cl, "emac0", 1u, &spd);
        }
    }


    /* configure SPD database entries on emac1 */
    /* ======================================= */
    if (FPP_ERR_OK == rtn)
    {
        fpp_spd_cmd_t spd = {0};
        uint32_t src_ip[4] = {0};
        uint32_t dst_ip[4] = {0};

        /* create SPD entry for ICMP traffic (ping) from PC1 to PC0 */
        /* ----------------------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new SPD entry */
            /* SA_ID passed in the demo_spd_ld_set_action() should be
               a valid index to some SAD entry in HSE */
            src_ip[0] = 0x0A0B0005;
            dst_ip[0] = 0x0A070002;
            demo_spd_ld_set_protocol(&spd, 1u);  /* 1 == ICMP */
            demo_spd_ld_set_ip(&spd, src_ip, dst_ip, false);
            demo_spd_ld_set_port(&spd, false, 0u, false, 0u);
            demo_spd_ld_set_action(&spd, FPP_SPD_ACTION_PROCESS_ENCODE, 1u, 0u);

            /* create a new SPD entry in PFE */
            rtn = demo_spd_add(p_cl, "emac1", 0u, &spd);
        }

        /* create SPD entry for TCP traffic from PC1 to PC0 */
        /* ------------------------------------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new SPD entry */
            /* SA_ID passed in the demo_spd_ld_set_action() should be
               a valid index to some SAD entry in HSE */
            src_ip[0] = 0x0A0B0005;
            dst_ip[0] = 0x0A070002;
            demo_spd_ld_set_protocol(&spd, 6u);  /* 6 == TCP */
            demo_spd_ld_set_ip(&spd, src_ip, dst_ip, false);
            demo_spd_ld_set_port(&spd, true, 4000u, true, 4000u);
            demo_spd_ld_set_action(&spd, FPP_SPD_ACTION_PROCESS_ENCODE, 2u, 0);

            /* create a new SPD entry in PFE */
            rtn = demo_spd_add(p_cl, "emac1", 1u, &spd);
        }
    }
```

```
    /* configure physical interfaces */
    /* ============================ */
    if (FPP_ERR_OK == rtn)
    {
        /* lock the interface database of PFE */
        rtn = demo_if_session_lock(p_cl);
        if (FPP_ERR_OK == rtn)
        {
            fpp_phy_if_cmd_t phyif = {0};

            /* configure physical interface "emac0" */
            /* ------------------------------------ */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac0");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_promisc(&phyif, false);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_DEFAULT);

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }

            /* configure physical interface "emac1" */
            /* ------------------------------------ */
            if (FPP_ERR_OK == rtn)
            {
                /* get data from PFE and store them in the local variable "phyif" */
                rtn = demo_phy_if_get_by_name(p_cl, &phyif, "emac1");
                if (FPP_ERR_OK == rtn)
                {
                    /* modify locally stored data */
                    demo_phy_if_ld_enable(&phyif);
                    demo_phy_if_ld_set_promisc(&phyif, false);
                    demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_DEFAULT);

                    /* update data in PFE */
                    rtn = demo_phy_if_update(p_cl, &phyif);
                }
            }

            /* configure physical interface "util" */
            /* ----------------------------------- */
            /* This interface represents interaction between PFE and HSE.
               This example configures util in Flexible Router mode to allow for distribution
               of the traffic which arrives from HSE. */
            if (FPP_ERR_OK == rtn)
            {
                fpp_log_if_cmd_t logif = {0};
                fpp_phy_if_cmd_t phyif = {0};

                /* create and configure a logical interface for traffic from PC0 to PC1 */
                /* -------------------------------------------------------------------- */
                if (FPP_ERR_OK == rtn)
                {
                    rtn = demo_log_if_add(p_cl, &logif, "From-PC0_to-PC1", "util");
                    if (FPP_ERR_OK == rtn)
                    {
                        /* NOTE: 1u == ID of emac1 */
                        demo_log_if_ld_set_promisc(&logif, false);
                        demo_log_if_ld_set_egress_phyifs(&logif, (1uL << 1u));
                        demo_log_if_ld_set_match_mode_or(&logif, false);
                        demo_log_if_ld_clear_all_mr(&logif);
                        demo_log_if_ld_set_mr_sip(&logif, true, 0x0A070002);
                        demo_log_if_ld_set_mr_dip(&logif, true, 0x0A0B0005);
                        demo_log_if_ld_enable(&logif);

                        rtn = demo_log_if_update(p_cl, &logif);
                    }
                }

                /* create and configure a logical interface for traffic from PC1 to PC0 */
                /* -------------------------------------------------------------------- */
                if (FPP_ERR_OK == rtn)
                {
                    rtn = demo_log_if_add(p_cl, &logif, "From-PC1_to-PC0", "util");
                    if (FPP_ERR_OK == rtn)
                    {
                        /* NOTE: 0u == ID of emac0 */
                        demo_log_if_ld_set_promisc(&logif, false);
```

```
                        demo_log_if_ld_set_egress_phyifs(&logif, (1uL << 0u));
                        demo_log_if_ld_set_match_mode_or(&logif, false);
                        demo_log_if_ld_clear_all_mr(&logif);
                        demo_log_if_ld_set_mr_sip(&logif, true, 0x0A0B0005);
                        demo_log_if_ld_set_mr_dip(&logif, true, 0x0A070002);
                        demo_log_if_ld_enable(&logif);

                        rtn = demo_log_if_update(p_cl, &logif);
                    }
                }

                /* configure physical interface "util" */
                /* ---------------------------------- */
                if (FPP_ERR_OK == rtn)
                {
                    /* get data from PFE and store them in the local variable "phyif" */
                    rtn = demo_phy_if_get_by_name(p_cl, &phyif, "util");
                    if (FPP_ERR_OK == rtn)
                    {
                        /* modify locally stored data */
                        demo_phy_if_ld_enable(&phyif);
                        demo_phy_if_ld_set_promisc(&phyif, false);
                        demo_phy_if_ld_set_mode(&phyif, FPP_IF_OP_FLEXIBLE_ROUTER);

                        /* update data in PFE */
                        rtn = demo_phy_if_update(p_cl, &phyif);
                    }
                }
            }
        }

        /* unlock the interface database of PFE */
        rtn = demo_if_session_unlock(p_cl, rtn);
    }


    return (rtn);
}


/* ========================================================================= */
```

15.9 demo_feature_qos.c

```
/* =========================================================================
 *  Copyright 2020-2022 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include "fpp.h"
#include "fpp_ext.h"
```



```
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"
#include "demo_qos.h"


extern int demo_feature_L2_bridge_vlan(FCI_CLIENT* p_cl);


/*
 * @brief      Use libFCI to configure PFE egress QoS feature.
 * @details    Scenario description:
 *               [*] Let there be two computers (PCs), both in the same network subnet.
 *                   Both PCs are connected through PFE. PFE acts as a simple bridge.
 *               [*] Use libFCI to configure PFE egress QoS feature on PFE's emac0 physical
 *                   interface, to prioritize and shape egress communication on emac0.
 *               [*] NOTE:
 *                   Be aware that all Egress QoS queues of a physical interface share
 *                   a single pool of available slots. This means that sum of all Egress QoS
 *                   queue lengths for every interface must fit within some limit.
 *                   See FCI API Reference (chapter Egress QoS) for interface limits.
 *             PC description:
 *               PC0:
 *                 --> IP address: 10.3.0.2/24
 *                 --> Accessible via PFE's emac0 physical interface.
 *               PC1:
 *                 --> IP address: 10.3.0.5/24
 *                 --> Accessible via PFE's emac1 physical interface.
 *             Additional info:
 *               QoS topology of this example:
 * @verbatim
                           SCH0
                           (WRR)
                        +--------+               SCH1
                  Q0--->| 0      |               (PQ)
                  Q1--->| 1      |            +--------+
                        | ...    +--->SHP0--->| 0      |
                        | 6      |            | 1      |
                        | 7      |            | ...    |
                        +--------+            | 4      +--->SHP2--->
                                              | 5      |
                                 Q6---SHP1--->| 6      |
                                 Q7---------->| 7      |
                                              +--------+
   @endverbatim
 * @note       This code uses a suite of "demo_" functions. The "demo_" functions encapsulate
 *             manipulation of libFCI data structs and calls of libFCI functions.
 *             It is advised to inspect content of these "demo_" functions.
 *
 * @param[in]  p_cl         FCI client
 *                          To create a client, use libFCI function fci_open().
 * @return     FPP_ERR_OK : All FCI commands were successfully executed.
 *                          Egress QoS should be up and running.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_feature_qos(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = FPP_ERR_OK;


    /* setup PFE to classify traffic (not needed by Egress QoS, done for demo purposes)*/
    /* ============================================================================== */
    rtn = demo_feature_L2_bridge_vlan(p_cl);


    /* configure Egress QoS queues for emac0 */
    /* ==================================== */
    if (FPP_ERR_OK == rtn)
    {
        fpp_qos_queue_cmd_t que = {0};

        /* first shorten and disable unused queues to free some slots in the shared pool */

        /* queue 2 (disabled) */
        /* ------------------ */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "que" */
            rtn = demo_qos_que_get_by_id(p_cl, &que, "emac0", 2u);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_qos_que_ld_set_mode(&que, 0u);  /* 0 == DISABLED */
                demo_qos_que_ld_set_max(&que, 0u);
```

```
            /* update data in PFE */
            rtn = demo_qos_que_update(p_cl, &que);
        }
    }

    /* queue 3 (disabled) */
    /* ------------------ */
    if (FPP_ERR_OK == rtn)
    {
        /* get data from PFE and store them in the local variable "que" */
        rtn = demo_qos_que_get_by_id(p_cl, &que, "emac0", 3u);
        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data */
            demo_qos_que_ld_set_mode(&que, 0u);  /* 0 == DISABLED */
            demo_qos_que_ld_set_max(&que, 0u);

            /* update data in PFE */
            rtn = demo_qos_que_update(p_cl, &que);
        }
    }

    /* queue 4 (disabled) */
    /* ------------------ */
    if (FPP_ERR_OK == rtn)
    {
        /* get data from PFE and store them in the local variable "que" */
        rtn = demo_qos_que_get_by_id(p_cl, &que, "emac0", 4u);
        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data */
            demo_qos_que_ld_set_mode(&que, 0u);  /* 0 == DISABLED */
            demo_qos_que_ld_set_max(&que, 0u);

            /* update data in PFE */
            rtn = demo_qos_que_update(p_cl, &que);
        }
    }

    /* queue 5 (disabled) */
    /* ------------------ */
    if (FPP_ERR_OK == rtn)
    {
        /* get data from PFE and store them in the local variable "que" */
        rtn = demo_qos_que_get_by_id(p_cl, &que, "emac0", 5u);
        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data */
            demo_qos_que_ld_set_mode(&que, 0u);  /* 0 == DISABLED */
            demo_qos_que_ld_set_max(&que, 0u);

            /* update data in PFE */
            rtn = demo_qos_que_update(p_cl, &que);
        }
    }

    /* now configure used queues ; keep in mind that sum of max lengths must be <255 */

    /* queue 0 */
    /* ------- */
    if (FPP_ERR_OK == rtn)
    {
        /* get data from PFE and store them in the local variable "que" */
        rtn = demo_qos_que_get_by_id(p_cl, &que, "emac0", 0u);
        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data */
            demo_qos_que_ld_set_mode(&que, 3u);  /* 3 == WRED */
            demo_qos_que_ld_set_min(&que, 25u);
            demo_qos_que_ld_set_max(&que, 100u);
            demo_qos_que_ld_set_zprob(&que, 0u, 10u);
            demo_qos_que_ld_set_zprob(&que, 1u, 20u);
            demo_qos_que_ld_set_zprob(&que, 2u, 30u);
            demo_qos_que_ld_set_zprob(&que, 3u, 40u);
            demo_qos_que_ld_set_zprob(&que, 4u, 50u);
            demo_qos_que_ld_set_zprob(&que, 5u, 60u);
            demo_qos_que_ld_set_zprob(&que, 6u, 70u);
            demo_qos_que_ld_set_zprob(&que, 7u, 80u);

            /* update data in PFE */
            rtn = demo_qos_que_update(p_cl, &que);
        }
    }

    /* queue 1 */
```

```
        /* ------- */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "que" */
            rtn = demo_qos_que_get_by_id(p_cl, &que, "emac0", 1u);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_qos_que_ld_set_mode(&que, 2u);  /* 2 == TAIL DROP */
                demo_qos_que_ld_set_max(&que, 50u);

                /* update data in PFE */
                rtn = demo_qos_que_update(p_cl, &que);
            }
        }

        /* queue 6 */
        /* ------- */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "que" */
            rtn = demo_qos_que_get_by_id(p_cl, &que, "emac0", 6u);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_qos_que_ld_set_mode(&que, 3u);  /* 3 == WRED */
                demo_qos_que_ld_set_min(&que, 10u);
                demo_qos_que_ld_set_max(&que, 50u);
                demo_qos_que_ld_set_zprob(&que, 0u, 20u);
                demo_qos_que_ld_set_zprob(&que, 1u, 20u);
                demo_qos_que_ld_set_zprob(&que, 2u, 40u);
                demo_qos_que_ld_set_zprob(&que, 3u, 40u);
                demo_qos_que_ld_set_zprob(&que, 4u, 60u);
                demo_qos_que_ld_set_zprob(&que, 5u, 60u);
                demo_qos_que_ld_set_zprob(&que, 6u, 80u);
                demo_qos_que_ld_set_zprob(&que, 7u, 80u);

                /* update data in PFE */
                rtn = demo_qos_que_update(p_cl, &que);
            }
        }

        /* queue 7 */
        /* ------- */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "que" */
            rtn = demo_qos_que_get_by_id(p_cl, &que, "emac0", 7u);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_qos_que_ld_set_mode(&que, 2u);  /* 2 == TAIL DROP */
                demo_qos_que_ld_set_max(&que, 50u);

                /* update data in PFE */
                rtn = demo_qos_que_update(p_cl, &que);
            }
        }
    }


    /* configure Egress QoS schedulers for emac0 */
    /* ========================================= */
    if (FPP_ERR_OK == rtn)
    {
        fpp_qos_scheduler_cmd_t sch = {0};

        /* scheduler 0 */
        /* ----------- */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "sch" */
            rtn = demo_qos_sch_get_by_id(p_cl, &sch, "emac0", 0u);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_qos_sch_ld_set_mode(&sch, 2u);  /* 2 == packet rate */
                demo_qos_sch_ld_set_algo(&sch, 3u);  /* 3 == WRR */
                demo_qos_sch_ld_set_input(&sch, 0u, true,    0u, 10000u);
                demo_qos_sch_ld_set_input(&sch, 1u, true,    1u, 20000u);
                demo_qos_sch_ld_set_input(&sch, 2u, false, 255u, 0u);
                demo_qos_sch_ld_set_input(&sch, 3u, false, 255u, 0u);
                demo_qos_sch_ld_set_input(&sch, 4u, false, 255u, 0u);
                demo_qos_sch_ld_set_input(&sch, 5u, false, 255u, 0u);
                demo_qos_sch_ld_set_input(&sch, 6u, false, 255u, 0u);
                demo_qos_sch_ld_set_input(&sch, 7u, false, 255u, 0u);
```

```
                /* update data in PFE */
                rtn = demo_qos_sch_update(p_cl, &sch);
            }
        }

        /* scheduler 1 */
        /* ----------- */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "sch" */
            rtn = demo_qos_sch_get_by_id(p_cl, &sch, "emac0", 1u);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_qos_sch_ld_set_mode(&sch, 1u);  /* 1 == data rate */
                demo_qos_sch_ld_set_algo(&sch, 0u);  /* 0 == PQ */
                demo_qos_sch_ld_set_input(&sch, 0u, true,    8u, 0u);
                demo_qos_sch_ld_set_input(&sch, 1u, false, 255u, 0u);
                demo_qos_sch_ld_set_input(&sch, 2u, false, 255u, 0u);
                demo_qos_sch_ld_set_input(&sch, 3u, false, 255u, 0u);
                demo_qos_sch_ld_set_input(&sch, 4u, false, 255u, 0u);
                demo_qos_sch_ld_set_input(&sch, 5u, false, 255u, 0u);
                demo_qos_sch_ld_set_input(&sch, 6u, true,    6u, 0u);
                demo_qos_sch_ld_set_input(&sch, 7u, true,    7u, 0u);

                /* update data in PFE */
                rtn = demo_qos_sch_update(p_cl, &sch);
            }
        }
    }


    /* configure Egress QoS shapers for emac0 */
    /* ====================================== */
    if (FPP_ERR_OK == rtn)
    {
        fpp_qos_shaper_cmd_t shp = {0};

        /* shaper 0 */
        /* -------- */
        rtn = demo_qos_shp_get_by_id(p_cl, &shp, "emac0", 0u);
        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data */
            demo_qos_shp_ld_set_mode(&shp, 2u);     /* 2 == packet rate */
            demo_qos_shp_ld_set_position(&shp, 1u); /* 1 == input #0 of scheduler 1 */
            demo_qos_shp_ld_set_isl(&shp, 1000u);   /* packets per sec */
            demo_qos_shp_ld_set_min_credit(&shp, -5000);
            demo_qos_shp_ld_set_max_credit(&shp, 10000);

            /* update data in PFE */
            rtn = demo_qos_shp_update(p_cl, &shp);
        }

        /* shaper 1 */
        /* -------- */
        rtn = demo_qos_shp_get_by_id(p_cl, &shp, "emac0", 1u);
        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data */
            demo_qos_shp_ld_set_mode(&shp, 2u);     /* 2 == packet rate */
            demo_qos_shp_ld_set_position(&shp, 7u); /* 7 == input #6 of scheduler 1 */
            demo_qos_shp_ld_set_isl(&shp, 2000u);   /* packets per sec */
            demo_qos_shp_ld_set_min_credit(&shp, -4000);
            demo_qos_shp_ld_set_max_credit(&shp,  8000);

            /* update data in PFE */
            rtn = demo_qos_shp_update(p_cl, &shp);
        }

        /* shaper 2 */
        /* -------- */
        rtn = demo_qos_shp_get_by_id(p_cl, &shp, "emac0", 2u);
        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data */
            demo_qos_shp_ld_set_mode(&shp, 1u);      /* 1 == data rate */
            demo_qos_shp_ld_set_position(&shp, 0u);  /* 0 == output of scheduler 1 */
            demo_qos_shp_ld_set_isl(&shp, 30000u);    /* bits per sec */
            demo_qos_shp_ld_set_min_credit(&shp, -60000);
            demo_qos_shp_ld_set_max_credit(&shp,  90000);

            /* update data in PFE */
            rtn = demo_qos_shp_update(p_cl, &shp);
        }
```

```
    }

    return (rtn);
}

/* ========================================================================= */
```

15.10 demo_feature_qos_policer.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"
#include "demo_qos_pol.h"


extern int demo_feature_L2_bridge_vlan(FCI_CLIENT* p_cl);


/*
 * @brief      Use libFCI to configure PFE ingress QoS feature.
 * @details    Scenario description:
 *               [*] Let there be two computers (PCs), both in the same network subnet.
 *                   Both PCs are connected through PFE. PFE acts as a simple bridge.
 *               [*] Use libFCI to configure PFE ingress QoS feature on PFE's emac0 physical
 *                   interface, to prioritize and shape ingress communication on emac0.
 *             PC description:
 *               PC0:
 *                 --> IP address: 10.3.0.2/24
 *                 --> Accessible via PFE's emac0 physical interface.
 *               PC1:
 *                 --> IP address: 10.3.0.5/24
 *                 --> Accessible via PFE's emac1 physical interface.
 *             Additional info (parameters of emac0 ingress QoS policing):
 *               [+] Ingressing ARP traffic shall be classified as Managed.
 *               [+] Ingressing IPv4 TCP traffic from PC0 IP shall be classified as Reserved.
 *               [+] Ingressing IPv4 UDP traffic (from any source) shall be dropped.
 *               [+] One WRED queue is required, with maximal depth of 255 and with linear
 *                   rise of drop probability for Unmanaged traffic.
 *               [+] One port-level shaper is required.
 *
```

```
 * @note       This code uses a suite of "demo_" functions. The "demo_" functions encapsulate
 *             manipulation of libFCI data structs and calls of libFCI functions.
 *             It is advised to inspect content of these "demo_" functions.
 *
 * @param[in]  p_cl         FCI client
 *                          To create a client, use libFCI function fci_open().
 * @return     FPP_ERR_OK : All FCI commands were successfully executed.
 *                          Ingress QoS policer should be up and running.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_feature_qos_policer(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = FPP_ERR_OK;


    /* setup PFE to classify traffic (not needed by Egress QoS, done for demo purposes)*/
    /* ============================================================================== */
    rtn = demo_feature_L2_bridge_vlan(p_cl);

    /* enable Ingress QoS policer on emac0 */
    /* =================================== */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_pol_enable(p_cl, "emac0", true);
    }

    /* configure Ingress QoS flows for emac0 */
    /* ===================================== */
    if (FPP_ERR_OK == rtn)
    {
        fpp_qos_policer_flow_cmd_t polflow = {0};

        /* flow 0 - ARP traffic shall be Managed */
        /* ------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new flow */
            memset(&polflow, 0, sizeof(fpp_qos_policer_flow_cmd_t));
            demo_polflow_ld_set_m_type_arp(&polflow, true);
            demo_polflow_ld_set_action(&polflow, FPP_IQOS_FLOW_MANAGED);

            /* create a new flow in PFE */
            rtn = demo_polflow_add(p_cl, "emac0", 0u, &polflow);
        }

        /* flow 1 - specific TCP traffic shall be Reserved */
        /* ----------------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new flow */
            memset(&polflow, 0, sizeof(fpp_qos_policer_flow_cmd_t));
            demo_polflow_ld_set_m_type_ip4(&polflow, true);
            demo_polflow_ld_set_am_proto(&polflow, true, 6u, FPP_IQOS_L4PROTO_MASK);
            demo_polflow_ld_set_am_sip(&polflow, true, 0x0A030002, FPP_IQOS_SDIP_MASK);
            demo_polflow_ld_set_action(&polflow, FPP_IQOS_FLOW_RESERVED);

            /* create a new flow in PFE */
            rtn = demo_polflow_add(p_cl, "emac0", 1u, &polflow);
        }

        /* flow 2 - UDP traffic shall be dropped */
        /* ------------------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* locally prepare data for a new flow */
            memset(&polflow, 0, sizeof(fpp_qos_policer_flow_cmd_t));
            demo_polflow_ld_set_am_proto(&polflow, true, 17u, FPP_IQOS_L4PROTO_MASK);
            demo_polflow_ld_set_action(&polflow, FPP_IQOS_FLOW_DROP);

            /* create a new flow in PFE */
            rtn = demo_polflow_add(p_cl, "emac0", 2u, &polflow);
        }
    }


    /* configure Ingress QoS WRED queues for emac0 */
    /* =========================================== */
    if (FPP_ERR_OK == rtn)
    {
        fpp_qos_policer_wred_cmd_t polwred = {0};

        /* WRED queue LMEM */
        /* --------------- */
        if (FPP_ERR_OK == rtn)
        {
```

```
            /* get data from PFE and store them in the local variable "polwred" */
            rtn = demo_polwred_get_by_que(p_cl, &polwred, "emac0", FPP_IQOS_Q_LMEM);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_polwred_ld_enable(&polwred, true);
                demo_polwred_ld_set_min(&polwred, 0u);
                demo_polwred_ld_set_max(&polwred, 200u);  /* over 200 == drop all Unmanaged */
                demo_polwred_ld_set_full(&polwred, 255u); /* over 255 == drop everything */
                demo_polwred_ld_set_zprob(&polwred, FPP_IQOS_WRED_ZONE1,  0u);
                demo_polwred_ld_set_zprob(&polwred, FPP_IQOS_WRED_ZONE2, 30u);
                demo_polwred_ld_set_zprob(&polwred, FPP_IQOS_WRED_ZONE3, 60u);
                demo_polwred_ld_set_zprob(&polwred, FPP_IQOS_WRED_ZONE4, 90u);

                /* update data in PFE */
                rtn = demo_polwred_update(p_cl, &polwred);
            }
        }

        /* WRED queue DMEM (disabled) */
        /* -------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "polwred" */
            rtn = demo_polwred_get_by_que(p_cl, &polwred, "emac0", FPP_IQOS_Q_DMEM);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_polwred_ld_enable(&polwred, false);

                /* update data in PFE */
                rtn = demo_polwred_update(p_cl, &polwred);
            }
        }

        /* WRED queue RXF (disabled) */
        /* ------------------------- */
        if (FPP_ERR_OK == rtn)
        {
            /* get data from PFE and store them in the local variable "polwred" */
            rtn = demo_polwred_get_by_que(p_cl, &polwred, "emac0", FPP_IQOS_Q_RXF);
            if (FPP_ERR_OK == rtn)
            {
                /* modify locally stored data */
                demo_polwred_ld_enable(&polwred, false);

                /* update data in PFE */
                rtn = demo_polwred_update(p_cl, &polwred);
            }
        }
    }


    /* configure Ingress QoS shapers for emac0 */
    /* ======================================= */
    if (FPP_ERR_OK == rtn)
    {
        fpp_qos_policer_shp_cmd_t polshp = {0};

        /* Ingress QoS shaper 0 */
        /* -------------------- */
        rtn = demo_polshp_get_by_id(p_cl, &polshp, "emac0", 0u);
        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data */
            demo_polshp_ld_enable(&polshp, true);
            demo_polshp_ld_set_type(&polshp, FPP_IQOS_SHP_PORT_LEVEL);
            demo_polshp_ld_set_mode(&polshp, FPP_IQOS_SHP_PPS);
            demo_polshp_ld_set_isl(&polshp, 1000u);
            demo_polshp_ld_set_min_credit(&polshp, -5000u);
            demo_polshp_ld_set_max_credit(&polshp, 10000u);

            /* update data in PFE */
            rtn = demo_polshp_update(p_cl, &polshp);
        }

        /* Ingress QoS shaper 1 (disabled) */
        /* ------------------------------- */
        rtn = demo_polshp_get_by_id(p_cl, &polshp, "emac0", 1u);
        if (FPP_ERR_OK == rtn)
        {
            /* modify locally stored data */
            demo_polshp_ld_enable(&polshp, false);

            /* update data in PFE */
            rtn = demo_polshp_update(p_cl, &polshp);
```

```
        }
    }

    return (rtn);
}

/* ========================================================================= */
```

15.11 demo_common.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>
#include <stdio.h>

#include <stdint.h>
#include <stddef.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"


/* ==== PUBLIC FUNCTIONS =================================================== */


/*
 * @brief      Check rtn value and print error text if (FPP_ERR_OK != rtn).
 * @param[in]  rtn          Current return value of a caller function.
 * @param[in]  p_txt_error  Text to be printed if (FPP_ERR_OK != rtn).
 */
void print_if_error(int rtn, const char* p_txt_error)
{
    assert(NULL != p_txt_error);

    if (FPP_ERR_OK != rtn)
    {
        printf("ERROR (%d): %s\n", rtn, p_txt_error);
    }
}


/*
 * @brief          Network-to-host (ntoh) function for enum datatypes.
 * @param[in,out]  p_rtn  Value which is to be converted to a host byte order.
 * @param[in]      size   Byte size of the value.
```



```
 */
void ntoh_enum(void* p_rtn, size_t size)
{
    assert(NULL != p_rtn);

    switch (size)
    {
        case (sizeof(uint16_t)):
            *((uint16_t*)p_rtn) = ntohs(*((uint16_t*)p_rtn));
        break;

        case (sizeof(uint32_t)):
            *((uint32_t*)p_rtn) = ntohl(*((uint32_t*)p_rtn));
        break;

        default:
            /* do nothing ; 'uint8_t' falls into this category as well */
        break;
    }
}


/*
 * @brief          Host-to-network (hton) function for enum datatypes.
 * @param[in,out]  p_rtn  Value which is to be converted to a network byte order.
 * @param[in]      size   Byte size of the value.
 */
void hton_enum(void* p_rtn, size_t size)
{
    assert(NULL != p_rtn);

    switch (size)
    {
        case (sizeof(uint16_t)):
            *((uint16_t*)p_rtn) = htons(*((uint16_t*)p_rtn));
        break;

        case (sizeof(uint32_t)):
            *((uint32_t*)p_rtn) = htonl(*((uint32_t*)p_rtn));
        break;

        default:
            /* do nothing ; 'uint8_t' falls into this category as well */
        break;
    }
}


/*
 * @brief       Check and set text.
 * @param[out]  p_dst   Destination text array (to be modified).
 * @param[in]   p_src   Source text array.
 *                      Can be NULL or empty (""). If NULL or empty, then
 *                      the destination text array is zeroed.
 * @param[in]   dst_ln  Size of the destination text array.
 * @return      FPP_ERR_OK : Function executed successfully.
 *              other      : Some error occured (represented by the respective error code).
 */
int set_text(char* p_dst, const char* p_src, const uint16_t dst_ln)
{
    assert(NULL != p_dst);
    assert(0u != dst_ln);
    /* 'p_src' is allowed to be NULL */

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    if ((NULL == p_src) || ('\0' == p_src[0]))
    {
        /* zeroify dst */
        memset(p_dst, 0, dst_ln);
        rtn = FPP_ERR_OK;
    }
    else if ((strlen(p_src) + 1u) > dst_ln)
    {
        rtn = FPP_ERR_INTERNAL_FAILURE;  /* src is too long */
    }
    else
    {
        /* set dst */
        strncpy(p_dst, p_src, dst_ln);
        rtn = FPP_ERR_OK;
    }

    return (rtn);
}
```

```
/*
 * @brief      Lock the interface database of PFE for exclusive access by this FCI client.
 * @details    The interface database is stored in PFE.
 * @param[in]  p_cl  FCI client
 * @return     FPP_ERR_OK : Lock successful
 *             other      : Lock not successful
 */
int demo_if_session_lock(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    return fci_write(p_cl, FPP_CMD_IF_LOCK_SESSION, 0u, NULL);
}


/*
 * @brief      Unlock exclusive access lock of the PFE's interface database.
 * @details    The exclusive access lock can be unlocked only by a FCI client which
 *             currently holds exclusive access to the interface database.
 * @param[in]  p_cl  FCI client
 * @param[in]  rtn   Current return value of a caller function.
 * @return     If a caller function provides NON-ZERO rtn, then that rtn value is returned.
 *             If a caller function provides ZERO rtn, then return values are:
 *             FPP_ERR_OK : Unlock successful
 *             other      : Unlock not successful
 */
int demo_if_session_unlock(FCI_CLIENT* p_cl, int rtn)
{
    assert(NULL != p_cl);

    int rtn_unlock = fci_write(p_cl, FPP_CMD_IF_UNLOCK_SESSION, 0u, NULL);
    rtn = ((FPP_ERR_OK == rtn) ? (rtn_unlock) : (rtn));

    return (rtn);
}


/*
 * @brief       Open connection to an FCI endpoint as a command-mode FCI client.
 * @details     Command-mode client can configure PFE via the FCI endpoint by
 *              issuing FCI commands.
 * @param[out]  pp_rtn_cl  Pointer to a newly created FCI client.
 * @return      FPP_ERR_OK : New FCI client was successfully created.
 *              other      : Failed to create a FCI client.
 */
int demo_client_open_in_cmd_mode(FCI_CLIENT** pp_rtn_cl)
{
    assert(NULL != pp_rtn_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    FCI_CLIENT* p_cl = fci_open(FCI_CLIENT_DEFAULT, FCI_GROUP_NONE);
    if (NULL != p_cl)
    {
        *pp_rtn_cl = p_cl;
        rtn = FPP_ERR_OK;
    }

    return (rtn);
}


/*
 * @brief      Close connection to a FCI endpoint and destroy the associated FCI client.
 * @param[in]  p_cl  The FCI client to be destroyed.
 * @return     FPP_ERR_OK : The FCI client was successfully destroyed.
 *             other      : Failed to destroy the FCI client instance.
 */
int demo_client_close(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    return fci_close(p_cl);
}


/* ========================================================================= */
```

15.12 demo_phy_if.c

```
/* =========================================================================
```



```
 *  Copyright 2020-2023 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_phy_if.h"


/* ==== PRIVATE FUNCTIONS ================================================== */


/*
 * @brief       Set/unset a flag in a physical interface struct.
 * @param[out]  p_rtn_phyif  Struct to be modified.
 * @param[in]   enable       New state of a flag.
 * @param[in]   flag         The flag.
 */
static void set_phyif_flag(fpp_phy_if_cmd_t* p_rtn_phyif, bool enable, fpp_if_flags_t flag)
{
    assert(NULL != p_rtn_phyif);

    hton_enum(&flag, sizeof(fpp_if_flags_t));

    if (enable)
    {
        p_rtn_phyif->flags |= flag;
    }
    else
    {
        p_rtn_phyif->flags &= (fpp_if_flags_t)(~flag);
    }
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from PFE ============= */


/*
 * @brief       Use FCI calls to get configuration data of a requested physical interface
 *              from PFE. Identify the interface by its name.
 * @details     To use this function properly, the interface database of PFE must be
 *              locked for exclusive access. See demo_phy_if_get_by_name_sa() for
 *              an example of a database lock procedure.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_phyif  Space for data from PFE.
 * @param[in]   p_name       Name of the requested physical interface.
 *                           Names of physical interfaces are hardcoded.
 *                           See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : The requested physical interface was found.
 *                           A copy of its configuration data was stored into p_rtn_phyif.
```



```
 *                            REMINDER: data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                            No data copied.
 */
int demo_phy_if_get_by_name(FCI_CLIENT* p_cl, fpp_phy_if_cmd_t* p_rtn_phyif,
                            const char* p_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_phyif);
    assert(NULL != p_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_phy_if_cmd_t cmd_to_fci = {0};
    fpp_phy_if_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_PHY_IF,
                        sizeof(fpp_phy_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop (with a search condition) */
    while ((FPP_ERR_OK == rtn) && (0 != strcmp((reply_from_fci.name), p_name)))
    {
        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_PHY_IF,
                        sizeof(fpp_phy_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_phyif = reply_from_fci;
    }

    print_if_error(rtn, "demo_phy_if_get_by_name() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get configuration data of a requested physical interface
 *              from PFE. Identify the interface by its name.
 * @details     This is a standalone (_sa) function.
 *              It shows how to properly access a physical interface. Namely:
 *              1. Lock the interface database of PFE for exclusive access by this FCI client.
 *              2. Execute one or more FCI calls which access physical or logical interfaces.
 *              3. Unlock the exclusive access lock.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_phyif  Space for data from PFE.
 * @param[in]   p_name       Name of the requested physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : The requested physical interface was found.
 *                            A copy of its configuration data was stored into p_rtn_phyif.
 *                            REMINDER: data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                            No data copied.
 */
inline int demo_phy_if_get_by_name_sa(FCI_CLIENT* p_cl, fpp_phy_if_cmd_t* p_rtn_phyif,
                                      const char* p_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_phyif);
    assert(NULL != p_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    /* lock the interface database of PFE for exclusive access by this FCI client */
    rtn = fci_write(p_cl, FPP_CMD_IF_LOCK_SESSION, 0, NULL);

    print_if_error(rtn, "demo_phy_if_get_by_name_sa() --> "
                        "fci_write(FPP_CMD_IF_LOCK_SESSION) failed!");

    /* execute "payload" - FCI calls which access physical or logical interfaces */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_phy_if_get_by_name(p_cl, p_rtn_phyif, p_name);
    }

    /* unlock the exclusive access lock */
    /* result of the unlock action is returned only if previous "payload" actions were OK */
```

```
    const int rtn_unlock = fci_write(p_cl, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL);
    rtn = ((FPP_ERR_OK == rtn) ? (rtn_unlock) : (rtn));

    print_if_error(rtn_unlock, "demo_phy_if_get_by_name_sa() --> "
                               "fci_write(FPP_CMD_IF_UNLOCK_SESSION) failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to update data in PFE ============= */


/*
 * @brief          Use FCI calls to update configuration of a target physical interface
 *                 in PFE.
 * @details        To use this function properly, the interface database of PFE must be
 *                 locked for exclusive access. See demo_phy_if_get_by_name_sa() for
 *                 an example of a database lock procedure.
 * @param[in]      p_cl     FCI client
 * @param[in,out]  p_phyif  Local data struct which represents a new configuration of
 *                          the target physical interface.
 *                          It is assumed that the struct contains a valid data of some
 *                          physical interface.
 * @return        FPP_ERR_OK : Configuration of the target physical interface was
 *                             successfully updated in PFE.
 *                             The local data struct was automatically updated with
 *                             readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                             The local data struct was not updated.
 */
int demo_phy_if_update(FCI_CLIENT* p_cl, fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_cl);
    assert(NULL != p_phyif);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_phy_if_cmd_t cmd_to_fci = (*p_phyif);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_PHY_IF, sizeof(fpp_phy_if_cmd_t),
                                          (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_phy_if_get_by_name(p_cl, p_phyif, (p_phyif->name));
    }

    print_if_error(rtn, "demo_phy_if_update() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : modify local data (no FCI calls) ================ */
/*
 * @defgroup    localdata_phyif  [localdata_phyif]
 * @brief:      Functions marked as [localdata_phyif] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_phyif (a struct with configuration data).
 *              Initial data for p_phyif can be obtained via demo_phy_if_get_by_name().
 *              If some modifications are made to local data, then after all modifications
 *              are done and finished, call demo_phy_if_update() to update
 *              the configuration of a real physical interface in PFE.
 */


/*
 * @brief          Enable ("up") a physical interface.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif  Local data to be modified.
 */
void demo_phy_if_ld_enable(fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    set_phyif_flag(p_phyif, true, FPP_IF_ENABLED);
}


/*
 * @brief          Disable ("down") a physical interface.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif  Local data to be modified.
 */
```

```
void demo_phy_if_ld_disable(fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    set_phyif_flag(p_phyif, false, FPP_IF_ENABLED);
}


/*
 * @brief          Set/unset a promiscuous mode of a physical interface.
 * @details        [localdata_phyif]
 *                 Promiscuous mode of a physical interface means the interface
 *                 will accept and process all incoming traffic, regardless of
 *                 the traffic's destination MAC.
 * @param[in,out]  p_phyif  Local data to be modified.
 * @param[in]      enable   Request to set/unset the promiscuous mode.
 */
void demo_phy_if_ld_set_promisc(fpp_phy_if_cmd_t* p_phyif, bool enable)
{
    assert(NULL != p_phyif);
    set_phyif_flag(p_phyif, enable, FPP_IF_PROMISC);
}


/*
 * @brief          Set/unset Fast Forwarding of all TCP traffic on a physical interface.
 * @details        [localdata_phyif]
 *                 When this flag is set, all TCP traffic is fast-forwarded. This is
 *                 default behavior.
 *                 When this flag is NOT set, TCP SYN|FIN|RST packets are not fast-forwarded.
 *                 Instead, they are passed to default logical interface (to host).
 * @param[in,out]  p_phyif  Local data to be modified.
 * @param[in]      enable   Request to set/unset the Fast Forwarding of all TCP traffic.
 */
void demo_phy_if_ld_set_ff_all_tcp(fpp_phy_if_cmd_t* p_phyif, bool enable)
{
    assert(NULL != p_phyif);
    set_phyif_flag(p_phyif, enable, FPP_IF_FF_ALL_TCP);
}


/*
 * @brief          Set/unset a VLAN conformance check on a physical interface.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif  Local data to be modified.
 * @param[in]      enable   Request to set/unset the VLAN conformance check.
 */
void demo_phy_if_ld_set_vlan_conf(fpp_phy_if_cmd_t* p_phyif, bool enable)
{
    assert(NULL != p_phyif);
    set_phyif_flag(p_phyif, enable, FPP_IF_VLAN_CONF_CHECK);
}


/*
 * @brief          Set/unset a PTP conformance check on a physical interface.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif  Local data to be modified.
 * @param[in]      enable   Request to set/unset the PTP conformance check.
 */
void demo_phy_if_ld_set_ptp_conf(fpp_phy_if_cmd_t* p_phyif, bool enable)
{
    assert(NULL != p_phyif);
    set_phyif_flag(p_phyif, enable, FPP_IF_PTP_CONF_CHECK);
}


/*
 * @brief          Set/unset a PTP promiscuous mode on a physical interface.
 * @details        [localdata_phyif]
 *                 This flag allows a PTP traffic to pass entry checks even if
 *                 the strict VLAN conformance check is active.
 * @param[in,out]  p_phyif  Local data to be modified.
 * @param[in]      enable   Request to set/unset the PTP promiscuous mode.
 */
void demo_phy_if_ld_set_ptp_promisc(fpp_phy_if_cmd_t* p_phyif, bool enable)
{
    assert(NULL != p_phyif);
    set_phyif_flag(p_phyif, enable, FPP_IF_PTP_PROMISC);
}


/*
 * @brief          Set/unset acceptance of a Q-in-Q traffic on a physical interface.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif  Local data to be modified.
 * @param[in]      enable   Request to set/unset the Q-in-Q acceptance.
```

```
 */
void demo_phy_if_ld_set_qinq(fpp_phy_if_cmd_t* p_phyif, bool enable)
{
    assert(NULL != p_phyif);
    set_phyif_flag(p_phyif, enable, FPP_IF_ALLOW_Q_IN_Q);
}


/*
 * @brief          Set/unset discarding of packets which have TTL<2.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif  Local data to be modified.
 * @param[in]      enable   Request to set/unset discarding of packets which have TTL<2.
 */
void demo_phy_if_ld_set_discard_ttl(fpp_phy_if_cmd_t* p_phyif, bool enable)
{
    assert(NULL != p_phyif);
    set_phyif_flag(p_phyif, enable, FPP_IF_DISCARD_TTL);
}


/*
 * @brief          Set an operation mode of a physical interface.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif  Local data to be modified.
 * @param[in]      mode     New operation mode.
 *                          For details about physical interface operation modes,
 *                          see description of the fpp_phy_if_op_mode_t type in
 *                          FCI API Reference.
 */
void demo_phy_if_ld_set_mode(fpp_phy_if_cmd_t* p_phyif, fpp_phy_if_op_mode_t mode)
{
    assert(NULL != p_phyif);
    hton_enum(&mode, sizeof(fpp_phy_if_op_mode_t));
    p_phyif->mode = mode;
}


/*
 * @brief          Set a blocking state of a physical interface.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif      Local data to be modified.
 * @param[in]      block_state  New blocking state
 *                              For details about physical interface blocking states,
 *                              see description of the fpp_phy_if_block_state_t type in
 *                              FCI API Reference.
 */
void demo_phy_if_ld_set_block_state(fpp_phy_if_cmd_t* p_phyif,
                                    fpp_phy_if_block_state_t block_state)
{
    assert(NULL != p_phyif);
    hton_enum(&block_state, sizeof(fpp_phy_if_block_state_t));
    p_phyif->block_state = block_state;
}


/*
 * @brief          Set rx mirroring rule of a physical interface.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif        Local data to be modified.
 * @param[in]      idx            Index into the array of interface's rx mirroring rules.
 * @param[in]      p_mirror_name  Name of a mirroring rule.
 *                                Can be NULL. If NULL or "" (empty string), then
 *                                this mirroring rule slot is unused (disabled).
 */
void demo_phy_if_ld_set_rx_mirror(fpp_phy_if_cmd_t* p_phyif, uint8_t idx,
                                  const char* p_mirror_name)
{
    assert(NULL != p_phyif);
    /* 'p_mirror_name' is allowed to be NULL */

    if (FPP_MIRRORS_CNT > idx)
    {
        set_text(p_phyif->rx_mirrors[idx], p_mirror_name, MIRROR_NAME_SIZE);
    }
}


/*
 * @brief          Set tx mirroring rule of a physical interface.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif        Local data to be modified.
 * @param[in]      idx            Index into the array of interface's tx mirroring rules.
 * @param[in]      p_mirror_name  Name of a mirroring rule.
 *                                Can be NULL. If NULL or "" (empty string), then
 *                                this mirroring rule slot is unused (disabled).
```

```
 */
void demo_phy_if_ld_set_tx_mirror(fpp_phy_if_cmd_t* p_phyif, uint8_t idx,
                                  const char* p_mirror_name)
{
    assert(NULL != p_phyif);
    /* 'p_mirror_name' is allowed to be NULL */

    if (FPP_MIRRORS_CNT > idx)
    {
        set_text(p_phyif->tx_mirrors[idx], p_mirror_name, MIRROR_NAME_SIZE);
    }
}


/*
 * @brief          Set FlexibleParser table to act as a FlexibleFilter for
 *                 a physical interface.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif       Local data to be modified.
 * @param[in]      p_table_name  Name of a FlexibleParser table.
 *                               Can be NULL. If NULL or "" (empty string), then
 *                               FlexibleFilter of this physical interface is disabled.
 */
void demo_phy_if_ld_set_flexifilter(fpp_phy_if_cmd_t* p_phyif, const char* p_table_name)
{
    assert(NULL != p_phyif);
    /* 'p_table_name' is allowed to be NULL */

    set_text(p_phyif->ftable, p_table_name, IFNAMSIZ);
}

/*
 * @brief          Set physical interface which shall be used as an egress for PTP traffic.
 * @details        [localdata_phyif]
 * @param[in,out]  p_phyif  Local data to be modified.
 * @param[in]      p_name   Name of a physical interface.
 *                          Can be NULL. If NULL or "" (empty string), then
 *                          this feature is disabled and PTP traffic is processed the same
 *                          way as any other traffic.
 */
void demo_phy_if_ld_set_ptp_mgmt_if(fpp_phy_if_cmd_t* p_phyif, const char* p_name)
{
    assert(NULL != p_phyif);
    /* 'p_name' is allowed to be NULL */

    set_text(p_phyif->ptp_mgmt_if, p_name, IFNAMSIZ);
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */


/*
 * @brief      Query the status of the "enable" flag.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the physical interface:
 *             true  : was enabled  ("up")
 *             false : was disabled ("down")
 */
bool demo_phy_if_ld_is_enabled(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);

    fpp_if_flags_t tmp_flags = (p_phyif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_ENABLED);
}


/*
 * @brief      Query the status of the "enable" flag (inverted logic).
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the physical interface:
 *             true  : was disabled ("down")
 *             false : was enabled  ("up")
 */
bool demo_phy_if_ld_is_disabled(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    return !demo_phy_if_ld_is_enabled(p_phyif);
}
```



```
/*
 * @brief      Query the status of the "promiscuous mode" flag.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the physical interface:
 *             true  : was in a promiscuous mode
 *             false : was NOT in a promiscuous mode
 */
bool demo_phy_if_ld_is_promisc(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);

    fpp_if_flags_t tmp_flags = (p_phyif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_PROMISC);
}


/*
 * @brief      Query the status of the "Fast Forwarding of all TCP traffic" flag.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the physical interface:
 *             true  : was fast-forwarding all TCP traffic
 *             false : was passing TCP SYN/FIN/RST packets to the default logical interface
 */
bool demo_phy_if_ld_is_ff_all_tcp(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);

    fpp_if_flags_t tmp_flags = (p_phyif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_FF_ALL_TCP);
}


/*
 * @brief      Query the status of the "VLAN conformance check" flag.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the physical interface:
 *             true  : was checking VLAN conformance of an incoming traffic
 *             false : was NOT checking VLAN conformance of an incoming traffic
 */
bool demo_phy_if_ld_is_vlan_conf(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);

    fpp_if_flags_t tmp_flags = (p_phyif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_VLAN_CONF_CHECK);
}


/*
 * @brief      Query the status of the "PTP conformance check" flag.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the physical interface:
 *             true  : was checking PTP conformance of an incoming traffic
 *             false : was NOT checking PTP conformance of an incoming traffic
 */
bool demo_phy_if_ld_is_ptp_conf(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);

    fpp_if_flags_t tmp_flags = (p_phyif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_PTP_CONF_CHECK);
}


/*
 * @brief      Query the status of the "PTP promisc" flag.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the physical interface:
 *             true  : was using PTP promiscuous mode
 *             false : was NOT using PTP promiscuous mode
 */
bool demo_phy_if_ld_is_ptp_promisc(const fpp_phy_if_cmd_t* p_phyif)
{
```



```
    assert(NULL != p_phyif);

    fpp_if_flags_t tmp_flags = (p_phyif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_PTP_PROMISC);
}


/*
 * @brief      Query the status of the "Q-in-Q" flag.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the physical interface:
 *             true  : was accepting Q-in-Q traffic
 *             false : was NOT accepting Q-in-Q traffic
 */
bool demo_phy_if_ld_is_qinq(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);

    fpp_if_flags_t tmp_flags = (p_phyif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_ALLOW_Q_IN_Q);
}


/*
 * @brief      Query the status of the "discard if TTL<2" flag.
 * @details    [localdata_phyif]
 *             This feature applies only if the physical interface is in a mode
 *             which decrements TTL of packets (e.g. L3 Router).
 * @param[in]  p_phyif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the physical interface:
 *             true  : was discarding packets which have TTL<2 (only for some modes)
 *             false : was sending packets which have TTL<2 to a host (only for some modes)
 */
bool demo_phy_if_ld_is_discard_ttl(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);

    fpp_if_flags_t tmp_flags = (p_phyif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_DISCARD_TTL);
}


/*
 * @brief      Query the name of a physical interface.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     Name of the physical interface.
 */
const char* demo_phy_if_ld_get_name(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    return (p_phyif->name);
}


/*
 * @brief      Query the ID of a physical interface.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     ID of the physical interface.
 */
uint32_t demo_phy_if_ld_get_id(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    return ntohl(p_phyif->id);
}


/*
 * @brief      Query the flags of a physical interface (the whole bitset).
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     Flags bitset at time when the data was obtained from PFE.
 */
fpp_if_flags_t demo_phy_if_ld_get_flags(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);

    fpp_if_flags_t tmp_flags = (p_phyif->flags);
```

```
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (tmp_flags);
}


/*
 * @brief      Query the operation mode of a physical interface.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     Operation mode of the physical interface at time when
 *             the data was obtained from PFE.
 */
fpp_phy_if_op_mode_t demo_phy_if_ld_get_mode(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);

    fpp_phy_if_op_mode_t tmp_mode = (p_phyif->mode);
    ntoh_enum(&tmp_mode, sizeof(fpp_phy_if_op_mode_t));

    return (tmp_mode);
}


/*
 * @brief      Query the blocking state of a physical interface.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     Blocking state of the physical interface at time when
 *             the data was obtained from PFE.
 */
fpp_phy_if_block_state_t demo_phy_if_ld_get_block_state(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);

    fpp_phy_if_block_state_t tmp_block_state = (p_phyif->block_state);
    ntoh_enum(&tmp_block_state, sizeof(fpp_phy_if_op_mode_t));

    return (tmp_block_state);
}


/*
 * @brief      Query the name of rx mirroring rule.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @param[in]  idx      Index into the array of interface's tx mirroring rules.
 * @return     Name of the mirroring rule which was assigned to the given slot at time when
 *             the data was obtained from PFE.
 */
const char* demo_phy_if_ld_get_rx_mirror(const fpp_phy_if_cmd_t* p_phyif, uint8_t idx)
{
    assert(NULL != p_phyif);
    return ((FPP_MIRRORS_CNT > idx) ? (p_phyif->rx_mirrors[idx]) : (""));
}


/*
 * @brief      Query the name of tx mirroring rule.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @param[in]  idx      Index into the array of interface's tx mirroring rules.
 * @return     Name of the mirroring rule which was assigned to the given slot at time when
 *             the data was obtained from PFE.
 */
const char* demo_phy_if_ld_get_tx_mirror(const fpp_phy_if_cmd_t* p_phyif, uint8_t idx)
{
    assert(NULL != p_phyif);
    return ((FPP_MIRRORS_CNT > idx) ? (p_phyif->tx_mirrors[idx]) : (""));
}


/*
 * @brief      Query the name of a FlexibleParser table which is being used as
 *             a FlexibleFilter for a physical interface.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     Name of the FlexibleParser table which was being used as a FlexibleFilter
 *             of the physical interface at time when the data was obtained from PFE.
 */
const char* demo_phy_if_ld_get_flexifilter(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    return (p_phyif->ftable);
}
```

```
/*
 * @brief      Query the physical interface which is being used as a PTP management interface.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     Name of the physical interface which is being used as the PTP management
 *             interface.
 */
const char* demo_phy_if_ld_get_ptp_mgmt_if(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    return (p_phyif->ptp_mgmt_if);
}


/*
 * @brief      Query the statistics of a physical interface - ingress.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     Count of ingress packets at the time when the data was obtained form PFE.
 */
uint32_t demo_phy_if_ld_get_stt_ingress(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    return ntohl(p_phyif->stats.ingress);
}


/*
 * @brief      Query the statistics of a physical interface - egress.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     Count of egressed packets at the time when the data was obtained form PFE.
 */
uint32_t demo_phy_if_ld_get_stt_egress(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    return ntohl(p_phyif->stats.egress);
}


/*
 * @brief      Query the statistics of a physical interface - malformed.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     Count of malformed packets at the time when the data was obtained form PFE.
 */
uint32_t demo_phy_if_ld_get_stt_malformed(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    return ntohl(p_phyif->stats.malformed);
}


/*
 * @brief      Query the statistics of a physical interface - discarded.
 * @details    [localdata_phyif]
 * @param[in]  p_phyif  Local data to be queried.
 * @return     Count of discarded packets at the time when the data was obtained form PFE.
 */
uint32_t demo_phy_if_ld_get_stt_discarded(const fpp_phy_if_cmd_t* p_phyif)
{
    assert(NULL != p_phyif);
    return ntohl(p_phyif->stats.discarded);
}


/* ==== PUBLIC FUNCTIONS : misc ============================================ */


/*
 * @brief      Use FCI calls to iterate through all available physical interfaces in PFE and
 *             execute a callback print function for each reported physical interface.
 * @details    To use this function properly, the interface database of PFE must be
 *             locked for exclusive access. See demo_phy_if_get_by_name_sa() for
 *             an example of a database lock procedure.
 * @param[in]  p_cl        FCI client
 * @param[in]  p_cb_print  Callback print function.
 *                         --> If the callback returns ZERO, then all is OK and
 *                             a next physical interface is picked for a print process.
 *                         --> If the callback returns NON-ZERO, then some problem is
 *                             assumed and this function terminates prematurely.
 * @return     FPP_ERR_OK : Successfully iterated through all available physical interfaces.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_phy_if_print_all(FCI_CLIENT* p_cl, demo_phy_if_cb_print_t p_cb_print)
```

```
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);


    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_phy_if_cmd_t cmd_to_fci = {0};
    fpp_phy_if_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_PHY_IF,
                    sizeof(fpp_phy_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        rtn = p_cb_print(&reply_from_fci);

        print_if_error(rtn, "demo_phy_if_print_all() --> "
                            "non-zero return from callback print function!");

        if (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_PHY_IF,
                            sizeof(fpp_phy_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more physical interfaces to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_IF_ENTRY_NOT_FOUND == rtn)
    {
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_phy_if_print_all() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available physical interfaces in PFE.
 * @details     To use this function properly, the interface database of PFE must be
 *              locked for exclusive access. See demo_phy_if_get_by_name_sa() for
 *              an example of a database lock procedure.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_count  Space to store the count of physical interfaces.
 * @return      FPP_ERR_OK : Successfully counted all available physical interfaces.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_phy_if_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);


    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_phy_if_cmd_t cmd_to_fci = {0};
    fpp_phy_if_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint32_t count = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_PHY_IF,
                    sizeof(fpp_phy_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        count++;

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_PHY_IF,
                        sizeof(fpp_phy_if_cmd_t), (unsigned short*)(&cmd_to_fci),
```

```
                         &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* query loop runs till there are no more physical interfaces to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_IF_ENTRY_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_phy_if_get_count() failed!");

    return (rtn);
}


/* ========================================================================= */
```

15.13 demo_log_if.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_log_if.h"


/* ==== PRIVATE FUNCTIONS ================================================== */


/*
 * @brief       Set/unset a flag in a logical interface struct.
 * @param[out]  p_rtn_phyif  Struct to be modified.
 * @param[in]   enable       New state of a flag.
 * @param[in]   flag         The flag.
 */
static void set_logif_flag(fpp_log_if_cmd_t* p_rtn_logif, bool enable, fpp_if_flags_t flag)
{
    assert(NULL != p_rtn_logif);

    hton_enum(&flag, sizeof(fpp_if_flags_t));
```

```
    if (enable)
    {
        p_rtn_logif->flags |= flag;
    }
    else
    {
        p_rtn_logif->flags &= (fpp_if_flags_t)(~flag);
    }
}


/*
 * @brief       Set/unset a match rule flag in a logical interface struct.
 * @param[out]  p_rtn_logif  Struct to be modified.
 * @param[in]   enable       New state of a match rule flag.
 * @param[in]   match_rule   The match rule flag.
 */
static void set_logif_mr_flag(fpp_log_if_cmd_t* p_rtn_logif, bool enable,
                              fpp_if_m_rules_t match_rule)
{
    assert(NULL != p_rtn_logif);

    hton_enum(&match_rule, sizeof(fpp_if_m_rules_t));

    if (enable)
    {
        p_rtn_logif->match |= match_rule;
    }
    else
    {
        p_rtn_logif->match &= (fpp_if_m_rules_t)(~match_rule);
    }
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from PFE ============== */


/*
 * @brief       Use FCI calls to get configuration data of a requested logical interface
 *              from PFE. Identify the interface by its name.
 * @details     To use this function properly, the interface database of PFE must be
 *              locked for exclusive access. See demo_log_if_get_by_name_sa() for
 *              an example of a database lock procedure.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_logif  Space for data from PFE.
 * @param[in]   p_name       Name of the requested logical interface.
 *                           Names of logical interfaces are user-defined.
 *                           See demo_log_if_add().
 * @return      FPP_ERR_OK : The requested logical interface was found.
 *                           A copy of its configuration data was stored into p_rtn_logif.
 *                           REMINDER: data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_log_if_get_by_name(FCI_CLIENT* p_cl, fpp_log_if_cmd_t* p_rtn_logif,
                            const char* p_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_logif);
    assert(NULL != p_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_log_if_cmd_t cmd_to_fci = {0};
    fpp_log_if_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_LOG_IF,
                    sizeof(fpp_log_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop (with a search condition) */
    while ((FPP_ERR_OK == rtn) && (0 != strcmp((reply_from_fci.name), p_name)))
    {
        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_LOG_IF,
                        sizeof(fpp_log_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
```

```
    {
        *p_rtn_logif = reply_from_fci;
    }

    print_if_error(rtn, "demo_log_if_get_by_name() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get configuration data of a requested logical interface
 *              from PFE. Identify the interface by its name.
 * @details     This is a standalone (_sa) function.
 *              It shows how to properly access a logical interface. Namely:
 *              1. Lock the interface database of PFE for exclusive access by this FCI client.
 *              2. Execute one or more FCI calls which access physical or logical interfaces.
 *              3. Unlock the exclusive access lock.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_logif  Space for data from PFE.
 * @param[in]   p_name       Name of the requested logical interface.
 *                           Names of logical interfaces are user-defined.
 *                           See demo_log_if_add().
 * @return      FPP_ERR_OK : The requested logical interface was found.
 *                           A copy of its configuration data was stored into p_rtn_logif.
 *                           REMINDER: data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
inline int demo_log_if_get_by_name_sa(FCI_CLIENT* p_cl, fpp_log_if_cmd_t* p_rtn_logif,
                                      const char* p_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_logif);
    assert(NULL != p_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    /* lock the interface database of PFE for exclusive access by this FCI client */
    rtn = fci_write(p_cl, FPP_CMD_IF_LOCK_SESSION, 0, NULL);

    print_if_error(rtn, "demo_log_if_get_by_name_sa() --> "
                        "fci_write(FPP_CMD_IF_LOCK_SESSION) failed!");

    /* execute "payload" - FCI calls which access physical or logical interfaces */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_log_if_get_by_name(p_cl, p_rtn_logif, p_name);
    }

    /* unlock the interface database's exclusive access lock */
    /* result of the unlock action is returned only if previous "payload" actions were OK */
    const int rtn_unlock = fci_write(p_cl, FPP_CMD_IF_UNLOCK_SESSION, 0, NULL);
    rtn = ((FPP_ERR_OK == rtn) ? (rtn_unlock) : (rtn));

    print_if_error(rtn_unlock, "demo_log_if_get_by_name_sa() --> "
                               "fci_write(FPP_CMD_IF_UNLOCK_SESSION) failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to update data in PFE ============= */


/*
 * @brief          Use FCI calls to update configuration of a target logical interface
 *                 in PFE.
 * @details        To use this function properly, the interface database of PFE must be
 *                 locked for exclusive access. See demo_log_if_get_by_name_sa() for
 *                 an example of a database lock procedure.
 * @param[in]      p_cl     FCI client
 * @param[in,out]  p_phyif  Local data struct which represents a new configuration of
 *                          the target logical interface.
 *                          It is assumed that the struct contains a valid data of some
 *                          logical interface.
 * @return        FPP_ERR_OK : Configuration of the target logical interface was
 *                             successfully updated in PFE.
 *                             The local data struct was automatically updated with
 *                             readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                              The local data struct was not updated.
 */
int demo_log_if_update(FCI_CLIENT* p_cl, fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_cl);
```

```
    assert(NULL != p_logif);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_log_if_cmd_t cmd_to_fci = (*p_logif);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_LOG_IF, sizeof(fpp_log_if_cmd_t),
                                         (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_log_if_get_by_name(p_cl, p_logif, (p_logif->name));
    }

    print_if_error(rtn, "demo_log_if_update() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to add/del items in PFE =========== */


/*
 * @brief       Use FCI calls to create a new logical interface in PFE.
 * @details     To use this function properly, the interface database of PFE must be
 *              locked for exclusive access. See demo_log_if_get_by_name_sa() for
 *              an example of a database lock procedure.
 * @param[in]   p_cl           FCI client
 * @param[out]  p_rtn_logif    Space for data from PFE.
 *                             This will contain a copy of configuration data of
 *                             the newly created logical interface.
 *                             Can be NULL. If NULL, then there is no local data to fill.
 * @param[in]   p_name         Name of the new logical interface.
 *                             The name is user-defined.
 * @param[in]   p_parent_name  Name of a parent physical interface.
 *                             Names of physical interfaces are hardcoded.
 *                             See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : New logical interface was created.
 *                           If applicable, then its configuration data were
 *                           copied into p_rtn_logif.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_log_if_add(FCI_CLIENT* p_cl, fpp_log_if_cmd_t* p_rtn_logif, const char* p_name,
                   const char* p_parent_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_name);
    assert(NULL != p_parent_name);
    /* 'p_rtn_logif' is allowed to be NULL */

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_log_if_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_name, IFNAMSIZ);
    if (FPP_ERR_OK == rtn)
    {
        rtn = set_text((cmd_to_fci.parent_name), p_parent_name, IFNAMSIZ);
    }

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_REGISTER;
        rtn = fci_write(p_cl, FPP_CMD_LOG_IF, sizeof(fpp_log_if_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));
    }

    /* read back and update caller data (if applicable) */
    if ((FPP_ERR_OK == rtn) && (NULL != p_rtn_logif))
    {
        rtn = demo_log_if_get_by_name(p_cl, p_rtn_logif, p_name);
    }

    print_if_error(rtn, "demo_log_if_add() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to destroy the target logical interface in PFE.
```



```
 * @details    To use this function properly, the interface database of PFE must be
 *             locked for exclusive access. See demo_log_if_get_by_name_sa() for
 *             an example of a database lock procedure.
 * @param[in]  p_cl    FCI client
 * @param[in]  p_name  Name of the logical interface to destroy.
 * @return     FPP_ERR_OK : The logical interface was destroyed.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_log_if_del(FCI_CLIENT* p_cl, const char* p_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_log_if_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_name, IFNAMSIZ);

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_DEREGISTER;
        rtn = fci_write(p_cl, FPP_CMD_LOG_IF, sizeof(fpp_log_if_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_log_if_del() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : modify local data (no FCI calls) ================ */
/*
 * @defgroup    localdata_logif  [localdata_logif]
 * @brief:      Functions marked as [localdata_logif] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_logif (a struct with configuration data).
 *              Initial data for p_logif can be obtained via demo_log_if_get_by_name().
 *              If some modifications are made to local data, then after all modifications
 *              are done and finished, call demo_log_if_update() to update
 *              the configuration of a real logical interface in PFE.
 */


/*
 * @brief          Enable ("up") a logical interface.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 */
void demo_log_if_ld_enable(fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    set_logif_flag(p_logif, true, FPP_IF_ENABLED);
}


/*
 * @brief          Disable ("down") a logical interface.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 */
void demo_log_if_ld_disable(fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    set_logif_flag(p_logif, false, FPP_IF_ENABLED);
}


/*
 * @brief          Set/unset a promiscuous mode of a logical interface.
 * @details        [localdata_logif]
 *                 Promiscuous mode of a logical interface means the interface
 *                 will accept all incoming traffic, regardless of active match rules.
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      enable   Request to set/unset the promiscuous mode.
 */
void demo_log_if_ld_set_promisc(fpp_log_if_cmd_t* p_logif, bool enable)
{
    assert(NULL != p_logif);
    set_logif_flag(p_logif, enable, FPP_IF_PROMISC);
}


/*
```

```
 * @brief          Set/unset a loopback mode of a logical interface.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      enable   Request to set/unset the loopback mode.
 */
void demo_log_if_ld_set_loopback(fpp_log_if_cmd_t* p_logif, bool enable)
{
    assert(NULL != p_logif);
    set_logif_flag(p_logif, enable, FPP_IF_LOOPBACK);
}


/*
 * @brief          Set match mode (chaining mode of match rules).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      match_mode_is_or  Request to set match mode.
 *                                   For details about logical interface match modes,
 *                                   see description of the fpp_if_flags_t type
 *                                   in FCI API Reference.
 */
void demo_log_if_ld_set_match_mode_or(fpp_log_if_cmd_t* p_logif, bool match_mode_is_or)
{
    assert(NULL != p_logif);
    set_logif_flag(p_logif, match_mode_is_or, FPP_IF_MATCH_OR);
}


/*
 * @brief          Set/unset inverted mode of traffic acceptance.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      enable   Request to set/unset inverted mode.
 *                          For details about logical interface inverted mode,
 *                          see description of the fpp_if_flags_t type
 *                          in FCI API Reference.
 */
void demo_log_if_ld_set_discard_on_m(fpp_log_if_cmd_t* p_logif, bool enable)
{
    assert(NULL != p_logif);
    set_logif_flag(p_logif, enable, FPP_IF_DISCARD);
}


/*
 * @brief          Set target physical interfaces (egress vector) which
 *                 shall receive a copy of the accepted traffic.
 * @details        [localdata_logif]
 *                 New egress vector fully replaces the old one.
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      egress   Target physical interfaces (egress vector). A bitset.
 *                          Each physical interface is represented by one bit.
 *                          Conversion between physical interface ID and a corresponding
 *                          egress vector bit is (1uL << "ID of a target physical interface").
 */
void demo_log_if_ld_set_egress_phyifs(fpp_log_if_cmd_t* p_logif, uint32_t egress)
{
    assert(NULL != p_logif);
    p_logif->egress = htonl(egress);
}


/*
 * @brief      Query the flags of a logical interface (the whole bitset).
 * @details    [localdata_phyif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Flags bitset at time when the data was obtained from PFE.
 */
fpp_if_flags_t demo_log_if_ld_get_flags(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);

    fpp_if_flags_t tmp_flags = (p_logif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (tmp_flags);
}




/*
 * @brief          Clear all match rules of a logical interface.
 *                 (also zeroify all match rule arguments of the logical interface)
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
```

```
 */
void demo_log_if_ld_clear_all_mr(fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    p_logif->match = 0u;
    memset(&(p_logif->arguments), 0, sizeof(fpp_if_m_args_t));
}


/*
 * @brief          Set/unset the given match rule (TYPE_ETH).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_eth(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_ETH);
}


/*
 * @brief          Set/unset the given match rule (TYPE_VLAN).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_vlan(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_VLAN);
}


/*
 * @brief          Set/unset the given match rule (TYPE_PPPOE).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_pppoe(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_PPPOE);
}


/*
 * @brief          Set/unset the given match rule (TYPE_ARP).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_arp(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_ARP);
}


/*
 * @brief          Set/unset the given match rule (TYPE_MCAST).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_mcast(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_MCAST);
}


/*
 * @brief          Set/unset the given match rule (TYPE_IPV4).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_ip4(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_IPV4);
}
```

```
/*
 * @brief          Set/unset the given match rule (TYPE_IPV6).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_ip6(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_IPV6);
}


/*
 * @brief          Set/unset the given match rule (TYPE_IPX).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_ipx(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_IPX);
}


/*
 * @brief          Set/unset the given match rule (TYPE_BCAST).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_bcast(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_BCAST);
}


/*
 * @brief          Set/unset the given match rule (TYPE_UDP).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_udp(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_UDP);
}


/*
 * @brief          Set/unset the given match rule (TYPE_TCP).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_tcp(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_TCP);
}


/*
 * @brief          Set/unset the given match rule (TYPE_ICMP).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_icmp(fpp_log_if_cmd_t* p_logif, bool set)
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_ICMP);
}


/*
 * @brief          Set/unset the given match rule (TYPE_IGMP).
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 */
void demo_log_if_ld_set_mr_type_igmp(fpp_log_if_cmd_t* p_logif, bool set)
```

```
{
    assert(NULL != p_logif);
    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_TYPE_IGMP);
}


/*
 * @brief          Set/unset the given match rule (VLAN) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      vlan     New VLAN ID for this match rule.
 *                          When this match rule is active, it compares value of its
 *                          'vlan' argument with the value of traffic's 'VID' field.
 */
void demo_log_if_ld_set_mr_vlan(fpp_log_if_cmd_t* p_logif, bool set, uint16_t vlan)
{
    assert(NULL != p_logif);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_VLAN);
    p_logif->arguments.vlan = htons(vlan);
}


/*
 * @brief          Set/unset the given match rule (PROTO) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      proto    New IP Protocol Number for this match rule.
 *                          When this match rule is active, it compares value of its
 *                          'proto' argument with the value of traffic's 'Protocol' field.
 *                          See "IANA Assigned Internet Protocol Number":
 *               https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
 */
void demo_log_if_ld_set_mr_proto(fpp_log_if_cmd_t* p_logif, bool set, uint8_t proto)
{
    assert(NULL != p_logif);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_PROTO);
    p_logif->arguments.proto = proto;
}


/*
 * @brief          Set/unset the given match rule (SPORT) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      sport    New source port value for this match rule.
 *                          When this match rule is active, it compares value of its
 *                          'sport' argument with the value of traffic's 'source port' field.
 */
void demo_log_if_ld_set_mr_sport(fpp_log_if_cmd_t* p_logif, bool set, uint16_t sport)
{
    assert(NULL != p_logif);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_SPORT);
    p_logif->arguments.sport = htons(sport);
}


/*
 * @brief          Set/unset the given match rule (DPORT) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      dport    New destination port value for this match rule.
 *                          When this match rule is active, it compares value of its
 *                          'dport' argument with the value of traffic's
 *                          'destination port' field.
 */
void demo_log_if_ld_set_mr_dport(fpp_log_if_cmd_t* p_logif, bool set, uint16_t dport)
{
    assert(NULL != p_logif);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_DPORT);
    p_logif->arguments.dport = htons(dport);
}


/*
 * @brief          Set/unset the given match rule (SIP6) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
```

```
 * @param[in]      p_sip6   New source IPv6 address for this match rule.
 *                          When this match rule is active, it compares value of its
 *                          'sip' argument with the value of traffic's
 *                          'source address' (applicable on IPv6 traffic only).
 */
void demo_log_if_ld_set_mr_sip6(fpp_log_if_cmd_t* p_logif, bool set,
                               const uint32_t p_sip6[4])
{
    assert(NULL != p_logif);
    assert(NULL != p_sip6);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_SIP6);

    p_logif->arguments.ipv.v6.sip[0] = htonl(p_sip6[0]);
    p_logif->arguments.ipv.v6.sip[1] = htonl(p_sip6[1]);
    p_logif->arguments.ipv.v6.sip[2] = htonl(p_sip6[2]);
    p_logif->arguments.ipv.v6.sip[3] = htonl(p_sip6[3]);
}


/*
 * @brief          Set/unset the given match rule (SIP6) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      p_dip6   New destination IPv6 address for this match rule.
 *                          When this match rule is active, it compares value of its
 *                          'dip' argument with the value of traffic's
 *                          'destination address' (applicable on IPv6 traffic only).
 */
void demo_log_if_ld_set_mr_dip6(fpp_log_if_cmd_t* p_logif, bool set,
                               const uint32_t p_dip6[4])
{
    assert(NULL != p_logif);
    assert(NULL != p_dip6);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_DIP6);

    p_logif->arguments.ipv.v6.dip[0] = htonl(p_dip6[0]);
    p_logif->arguments.ipv.v6.dip[1] = htonl(p_dip6[1]);
    p_logif->arguments.ipv.v6.dip[2] = htonl(p_dip6[2]);
    p_logif->arguments.ipv.v6.dip[3] = htonl(p_dip6[3]);
}


/*
 * @brief          Set/unset the given match rule (SIP) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      sip      New source IPv4 address for this match rule.
 *                          When this match rule is active, it compares value of its
 *                          'sip' argument with the value of traffic's
 *                          'source address' (applicable on IPv4 traffic only).
 */
void demo_log_if_ld_set_mr_sip(fpp_log_if_cmd_t* p_logif, bool set, uint32_t sip)
{
    assert(NULL != p_logif);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_SIP);
    p_logif->arguments.ipv.v4.sip = htonl(sip);
}


/*
 * @brief          Set/unset the given match rule (DIP) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      dip      New destination IPv4 address for this match rule.
 *                          When this match rule is active, it compares value of its
 *                          'dip' argument with the value of traffic's
 *                          'destination address' (applicable on IPv4 traffic only).
 */
void demo_log_if_ld_set_mr_dip(fpp_log_if_cmd_t* p_logif, bool set, uint32_t dip)
{
    assert(NULL != p_logif);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_DIP);
    p_logif->arguments.ipv.v4.dip = htonl(dip);
}


/*
 * @brief          Set/unset the given match rule (ETHTYPE) and its argument.
 * @details        [localdata_logif]
```



```
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      ethtype  New EtherType number for this match rule.
 *                          When this match rule is active, it compares value of its
 *                          'ethtype' argument with the value of traffic's 'EtherType' field.
 *                          See "IANA EtherType number (IEEE 802)":
 *               https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml
 */
void demo_log_if_ld_set_mr_ethtype(fpp_log_if_cmd_t* p_logif, bool set, uint16_t ethtype)
{
    assert(NULL != p_logif);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_ETHTYPE);
    p_logif->arguments.ethtype = htons(ethtype);
}


/*
 * @brief          Set/unset the given match rule (FP0) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      fp_table0_name  Name of a FlexibleParser table for this match rule.
 *                                 Requested FlexibleParser table must already exist in PFE.
 *                                 When this match rule is active, it inspects traffic
 *                                 according to rules listed in the referenced
 *                                 FlexibleParser table.
 */
void demo_log_if_ld_set_mr_fp0(fpp_log_if_cmd_t* p_logif, bool set,
                               const char* fp_table0_name)
{
    assert(NULL != p_logif);
    /* 'fp_table0_name' is allowed to be NULL */

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_FP0);
    set_text((p_logif->arguments.fp_table0), fp_table0_name, IFNAMSIZ);
}


/*
 * @brief          Set/unset the given match rule (FP1) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      fp_table1_name  Name of a FlexibleParser table for this match rule.
 *                                 Requested FlexibleParser table must already exist in PFE.
 *                                 When this match rule is active, it inspects traffic
 *                                 according to rules listed in the referenced
 *                                 FlexibleParser table.
 */
void demo_log_if_ld_set_mr_fp1(fpp_log_if_cmd_t* p_logif, bool set,
                               const char* fp_table1_name)
{
    assert(NULL != p_logif);
    /* 'fp_table1_name' is allowed to be NULL */

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_FP1);
    set_text((p_logif->arguments.fp_table1), fp_table1_name, IFNAMSIZ);
}


/*
 * @brief          Set/unset the given match rule (SMAC) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      p_smac   New source MAC address for this match rule.
 *                          When this match rule is active, it compares value of its
 *                          'smac' argument with the value of traffic's 'source MAC' field.
 */
void demo_log_if_ld_set_mr_smac(fpp_log_if_cmd_t* p_logif, bool set, const uint8_t p_smac[6])
{
    assert(NULL != p_logif);
    assert(NULL != p_smac);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_SMAC);
    memcpy((p_logif->arguments.smac), p_smac, (6 * sizeof(uint8_t)));
}


/*
 * @brief          Set/unset the given match rule (DMAC) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      p_dmac   New destination MAC address for this match rule.
```

```
 *                           When this match rule is active, it compares value of its
 *                           'dmac' argument with the value of traffic's
 *                           'destination MAC' field.
 */
void demo_log_if_ld_set_mr_dmac(fpp_log_if_cmd_t* p_logif, bool set, const uint8_t p_dmac[6])
{
    assert(NULL != p_logif);
    assert(NULL != p_dmac);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_DMAC);
    memcpy((p_logif->arguments.dmac), p_dmac, (6 * sizeof(uint8_t)));
}


/*
 * @brief          Set/unset the given match rule (HIF_COOKIE) and its argument.
 * @details        [localdata_logif]
 * @param[in,out]  p_logif  Local data to be modified.
 * @param[in]      set      Request to set/unset the given match rule.
 * @param[in]      hif_cookie  New hif cookie value for this match rule.
 *                              When this match rule is active, it compares value of its
 *                             'hif_cookie' argument with the value of a hif_cookie tag.
 *                              Hif_cookie tag is a part of internal overhead data, attached
 *                              to traffic by a host's PFE driver.
 */
void demo_log_if_ld_set_mr_hif_cookie(fpp_log_if_cmd_t* p_logif, bool set,
                                      uint32_t hif_cookie)
{
    assert(NULL != p_logif);

    set_logif_mr_flag(p_logif, set, FPP_IF_MATCH_HIF_COOKIE);
    p_logif->arguments.hif_cookie = htonl(hif_cookie);
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */


/*
 * @brief      Query the status of the "enable" flag.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the logical interface:
 *             true  : was enabled  ("up")
 *             false : was disabled ("down")
 */
bool demo_log_if_ld_is_enabled(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);

    fpp_if_flags_t tmp_flags = (p_logif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_ENABLED);
}


/*
 * @brief      Query the status of the "enable" flag (inverted logic).
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the logical interface:
 *             true  : was disabled ("down")
 *             false : was enabled  ("up")
 */
bool demo_log_if_ld_is_disabled(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return !demo_log_if_ld_is_enabled(p_logif);
}


/*
 * @brief      Query the status of the "promiscuous mode" flag.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the logical interface:
 *             true  : was in a promiscuous mode
 *             false : was NOT in a promiscuous mode
 */
bool demo_log_if_ld_is_promisc(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);

    fpp_if_flags_t tmp_flags = (p_logif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));
```

```
    return (bool)(tmp_flags & FPP_IF_PROMISC);
}


/*
 * @brief      Query the status of the "loopback" flag.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the logical interface:
 *             true  : was in a loopback mode
 *             false : was NOT in a loopback mode
 */
bool demo_log_if_ld_is_loopback(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);

    fpp_if_flags_t tmp_flags = (p_logif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_LOOPBACK);
}


/*
 * @brief      Query the status of the "match mode" flag (chaining mode of match rules).
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the logical interface:
 *             true  : was using OR match mode
 *             false : was using AND match mode
 */
bool demo_log_if_ld_is_match_mode_or(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);

    fpp_if_flags_t tmp_flags = (p_logif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_MATCH_OR);
}


/*
 * @brief      Query the status of the "discard on match" flag.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the logical interface:
 *             true  : was discarding traffic that passed its matching process
 *             false : was NOT discarding traffic that passed its matching process
 */
bool demo_log_if_ld_is_discard_on_m(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);

    fpp_if_flags_t tmp_flags = (p_logif->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_if_flags_t));

    return (bool)(tmp_flags & FPP_IF_DISCARD);
}


/*
 * @brief      Query whether a physical interface is a member of
 *             a logical interface's egress vector.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @param[in]  egress_bitflag  Queried physical interface. A bitflag.
 *                             Each physical interface is represented by one bit.
 *                             Conversion between physical interface ID and a corresponding
 *                             egress vector bit is
 *                             (1uL << "ID of a target physical interface").
 * @return     At time when the data was obtained from PFE, the logical interface:
 *             true  : had at least one queried egress bitflag set
 *             false : had none of the queried egress bitflags set
 */
bool demo_log_if_ld_is_egress_phyifs(const fpp_log_if_cmd_t* p_logif, uint32_t egress_bitflag)
{
    assert(NULL != p_logif);
    return (bool)(ntohl(p_logif->match) & egress_bitflag);
}


/*
 * @brief      Query whether a match rule is active or not.
 * @details    [localdata_logif]
```



```
 * @param[in]  p_logif  Local data to be queried.
 * @param[in]  match_rule  Queried match rule.
 * @return     At time when the data was obtained from PFE, the logical interface:
 *             true  : had at least one queried match rule set
 *             false : had none of the queried match rules set
 */
bool demo_log_if_ld_is_match_rule(const fpp_log_if_cmd_t* p_logif,
                                  fpp_if_m_rules_t match_rule)
{
    assert(NULL != p_logif);

    fpp_if_m_rules_t tmp_match = (p_logif->match);
    ntoh_enum(&tmp_match, sizeof(fpp_if_m_rules_t));

    return (bool)(tmp_match & match_rule);
}




/*
 * @brief      Query the name of a logical interface.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Name of the logical interface.
 */
const char* demo_log_if_ld_get_name(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return (p_logif->name);
}


/*
 * @brief      Query the ID of a logical interface.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     ID of the logical interface.
 */
uint32_t demo_log_if_ld_get_id(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohl(p_logif->id);
}


/*
 * @brief      Query the name of a logical interface's parent.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Name of the parent physical interface.
 */
const char* demo_log_if_ld_get_parent_name(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return (p_logif->parent_name);
}


/*
 * @brief      Query the ID of a logical interface's parent.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     ID of the parent physical interface.
 */
uint32_t demo_log_if_ld_get_parent_id(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohl(p_logif->parent_id);
}


/*
 * @brief      Query the target physical interfaces (egress vector) of a logical interface.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Egress vector.
 */
uint32_t demo_log_if_ld_get_egress(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohl(p_logif->egress);
}
```

```
/*
 * @brief      Query the match rule bitset of a logical interface.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Match rule bitset.
 */
fpp_if_m_rules_t demo_log_if_ld_get_mr_bitset(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);

    fpp_if_m_rules_t tmp_match = (p_logif->match);
    ntoh_enum(&tmp_match, sizeof(fpp_if_m_rules_t));

    return (tmp_match);
}


/*
 * @brief      Query the argument of the match rule VLAN.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (VLAN ID) of the given match rule.
 */
uint16_t demo_log_if_ld_get_mr_arg_vlan(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohs(p_logif->arguments.vlan);
}


/*
 * @brief      Query the argument of the match rule PROTO.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (Protocol ID) of the given match rule.
 */
uint8_t demo_log_if_ld_get_mr_arg_proto(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return (p_logif->arguments.proto);
}


/*
 * @brief      Query the argument of the match rule SPORT.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (source port ID) of the given match rule.
 */
uint16_t demo_log_if_ld_get_mr_arg_sport(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohs(p_logif->arguments.sport);
}


/*
 * @brief      Query the argument of the match rule DPORT.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (destination port ID) of the given match rule.
 */
uint16_t demo_log_if_ld_get_mr_arg_dport(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohs(p_logif->arguments.dport);
}


/*
 * @brief      Query the argument of the match rule SIP6.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (source IPv6) of the given match rule.
 */
const uint32_t* demo_log_if_ld_get_mr_arg_sip6(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    static uint32_t rtn_sip6[4] = {0u};

    rtn_sip6[0] = ntohl(p_logif->arguments.ipv.v6.sip[0]);
    rtn_sip6[1] = ntohl(p_logif->arguments.ipv.v6.sip[1]);
    rtn_sip6[2] = ntohl(p_logif->arguments.ipv.v6.sip[2]);
    rtn_sip6[3] = ntohl(p_logif->arguments.ipv.v6.sip[3]);
```

```
    return (rtn_sip6);
}


/*
 * @brief      Query the argument of the match rule DIP6.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (destination IPv6) of the given match rule.
 */
const uint32_t* demo_log_if_ld_get_mr_arg_dip6(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    static uint32_t rtn_dip6[4] = {0u};

    rtn_dip6[0] = ntohl(p_logif->arguments.ipv.v6.dip[0]);
    rtn_dip6[1] = ntohl(p_logif->arguments.ipv.v6.dip[1]);
    rtn_dip6[2] = ntohl(p_logif->arguments.ipv.v6.dip[2]);
    rtn_dip6[3] = ntohl(p_logif->arguments.ipv.v6.dip[3]);

    return (rtn_dip6);
}


/*
 * @brief      Query the argument of the match rule SIP.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (source IPv4) of the given match rule.
 */
uint32_t demo_log_if_ld_get_mr_arg_sip(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohl(p_logif->arguments.ipv.v4.sip);
}


/*
 * @brief      Query the argument of the match rule DIP.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (destination IPv4) of the given match rule.
 */
uint32_t demo_log_if_ld_get_mr_arg_dip(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohl(p_logif->arguments.ipv.v4.dip);
}


/*
 * @brief      Query the argument of the match rule ETHTYPE.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (EtherType ID) of the given match rule.
 */
uint16_t demo_log_if_ld_get_mr_arg_ethtype(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohs(p_logif->arguments.ethtype);
}


/*
 * @brief      Query the argument of the match rule FP0.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (name of a FlexibleParser table) of the given match rule.
 */
const char* demo_log_if_ld_get_mr_arg_fp0(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return (p_logif->arguments.fp_table0);
}


/*
 * @brief      Query the argument of the match rule FP1.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (name of a FlexibleParser table) of the given match rule.
 */
const char* demo_log_if_ld_get_mr_arg_fp1(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return (p_logif->arguments.fp_table1);
```

```
}


/*
 * @brief      Query the argument of the match rule SMAC.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (source MAC address) of the given match rule.
 */
const uint8_t* demo_log_if_ld_get_mr_arg_smac(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return (p_logif->arguments.smac);
}


/*
 * @brief      Query the argument of the match rule DMAC.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (destination MAC address) of the given match rule.
 */
const uint8_t* demo_log_if_ld_get_mr_arg_dmac(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return (p_logif->arguments.dmac);
}


/*
 * @brief      Query the argument of the match rule HIF_COOKIE.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Argument (hif cookie value) of the given match rule.
 */
uint32_t demo_log_if_ld_get_mr_arg_hif_cookie(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohl(p_logif->arguments.hif_cookie);
}




/*
 * @brief      Query the statistics of a logical interface - processed.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Count of processed packets at the time when the data was obtained form PFE.
 */
uint32_t demo_log_if_ld_get_stt_processed(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohl(p_logif->stats.processed);
}


/*
 * @brief      Query the statistics of a logical interface - accepted.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Count of accepted packets at the time when the data was obtained form PFE.
 */
uint32_t demo_log_if_ld_get_stt_accepted(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohl(p_logif->stats.accepted);
}


/*
 * @brief      Query the statistics of a logical interface - rejected.
 * @details    [localdata_logif]
 * @param[in]  p_logif  Local data to be queried.
 * @return     Count of rejected packets at the time when the data was obtained form PFE.
 */
uint32_t demo_log_if_ld_get_stt_rejected(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohl(p_logif->stats.rejected);
}


/*
 * @brief      Query the statistics of a logical interface - discarded.
 * @details    [localdata_logif]
```



```
 * @param[in]  p_logif  Local data to be queried.
 * @return     Count of discarded packets at the time when the data was obtained form PFE.
 */
uint32_t demo_log_if_ld_get_stt_discarded(const fpp_log_if_cmd_t* p_logif)
{
    assert(NULL != p_logif);
    return ntohl(p_logif->stats.discarded);
}


/* ==== PUBLIC FUNCTIONS : misc ============================================ */


/*
 * @brief      Use FCI calls to iterate through all available logical interfaces in PFE and
 *             execute a callback print function for each applicable logical interface.
 * @details    To use this function properly, the interface database of PFE must be
 *             locked for exclusive access. See demo_log_if_get_by_name_sa() for
 *             an example of a database lock procedure.
 * @param[in]  p_cl        FCI client
 * @param[in]  p_cb_print  Callback print function.
 *                         --> If the callback returns ZERO, then all is OK and
 *                             a next logical interface is picked for a print process.
 *                         --> If the callback returns NON-ZERO, then some problem is
 *                             assumed and this function terminates prematurely.
 * @param[in]  p_parent_name   [optional parameter] Name of a parent physical interface.
 *                             Names of physical interfaces are hardcoded.
 *                             See FCI API Reference, chapter Interface Management.
 *                             Can be NULL.
 *                             If NULL, then all available logical interfaces are printed.
 *                             If non-NULL, then only those logical interfaces which are
 *                             children of the given physical interface are printed.
 * @return     FPP_ERR_OK : Successfully iterated through all available logical interfaces.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_log_if_print_all(FCI_CLIENT* p_cl, demo_log_if_cb_print_t p_cb_print,
                         const char* p_parent_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);
    /* 'p_parent_name' is allowed to be NULL */

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_log_if_cmd_t cmd_to_fci = {0};
    fpp_log_if_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_LOG_IF,
                    sizeof(fpp_log_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        if ((NULL == p_parent_name) ||
            (0 == strcmp((reply_from_fci.parent_name), p_parent_name)))
        {
            rtn = p_cb_print(&reply_from_fci);
        }

        if (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_LOG_IF,
                            sizeof(fpp_log_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more logical interfaces to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_IF_ENTRY_NOT_FOUND == rtn)
    {
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_log_if_print_all() failed!");

    return (rtn);
}


/*
```

```
 * @brief       Use FCI calls to get a count of all available logical interfaces in PFE.
 * @details     To use this function properly, the interface database of PFE must be
 *              locked for exclusive access. See demo_log_if_get_by_name_sa() for
 *              an example of a database lock procedure.
 * @param[in]   p_cl           FCI client
 * @param[out]  p_rtn_count    Space to store the count of logical interfaces.
 * @param[in]   p_parent_name  [optional parameter] Name of a parent physical interface.
 *                             Names of physical interfaces are hardcoded.
 *                             See FCI API Reference, chapter Interface Management.
 *                             Can be NULL.
 *                             If NULL, then all available logical interfaces are counted.
 *                             If non-NULL, then only those logical interfaces which are
 *                             children of the given physical interface are counted.
 * @return      FPP_ERR_OK : Successfully counted all applicable logical interfaces.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_log_if_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                          const char* p_parent_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);
    /* 'p_parent_name' is allowed to be NULL */

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_log_if_cmd_t cmd_to_fci = {0};
    fpp_log_if_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint32_t count = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_LOG_IF,
                    sizeof(fpp_log_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        if ((NULL == p_parent_name) ||
            (0 == strcmp((reply_from_fci.parent_name), p_parent_name)))
        {
            count++;
        }

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_LOG_IF,
                        sizeof(fpp_log_if_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* query loop runs till there are no more logical interfaces to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_IF_ENTRY_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_log_if_get_count() failed!");

    return (rtn);
}


/* ========================================================================= */
```

15.14 demo_if_mac.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
```

```
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_if_mac.h"


/* ==== PUBLIC FUNCTIONS : use FCI calls to add/del items in PFE =========== */

/*
 * @brief       Use FCI calls to add a new MAC address to an interface.
 * @details     To use this function properly, the interface database of PFE must be
 *              locked for exclusive access. See demo_phy_if_get_by_name_sa() for
 *              an example of a database lock procedure.
 * @param[in]   p_cl       FCI client
 * @param[out]  p_mac      New MAC address.
 * @param[in]   p_name     Name of a target physical interface.
 *                         Names of physical interfaces are hardcoded.
 *                         See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : New MAC address was added to the target physical interface.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_if_mac_add(FCI_CLIENT* p_cl, const uint8_t p_mac[6], const char* p_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_mac);
    assert(NULL != p_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_if_mac_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_name, IFNAMSIZ);
    if (FPP_ERR_OK == rtn)
    {
        memcpy(cmd_to_fci.mac, p_mac, (6 * sizeof(uint8_t)));
    }

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_REGISTER;
        rtn = fci_write(p_cl, FPP_CMD_IF_MAC, sizeof(fpp_if_mac_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_if_mac_add() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to remove the target MAC address from an interface.
 * @details     To use this function properly, the interface database of PFE must be
 *              locked for exclusive access. See demo_phy_if_get_by_name_sa() for
```



```
 *              an example of a database lock procedure.
 * @param[in]   p_cl       FCI client
 * @param[out]  p_mac      MAC address to be remove.
 * @param[in]   p_name     Name of a target physical interface.
 *                         Names of physical interfaces are hardcoded.
 *                         See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : The MAC address was removed from the target physical interface.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_if_mac_del(FCI_CLIENT* p_cl, const uint8_t p_mac[6], const char* p_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_mac);
    assert(NULL != p_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_if_mac_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_name, IFNAMSIZ);
    if (FPP_ERR_OK == rtn)
    {
        memcpy(cmd_to_fci.mac, p_mac, (6 * sizeof(uint8_t)));
    }

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_DEREGISTER;
        rtn = fci_write(p_cl, FPP_CMD_IF_MAC, sizeof(fpp_if_mac_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_if_mac_del() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */
/*
 * @defgroup    localdata_if_mac  [localdata_if_mac]
 * @brief:      Functions marked as [localdata_if_mac] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_if_mac (a struct with MAC data).
 */


/*
 * @brief      Query the name of a target interface.
 * @details    [localdata_if_mac]
 * @param[in]  p_if_mac  Local data to be queried.
 * @return     Name of the target interface.
 */
const char* demo_if_mac_ld_get_name(const fpp_if_mac_cmd_t* p_if_mac)
{
    assert(NULL != p_if_mac);
    return (p_if_mac->name);
}


/*
 * @brief      Query the MAC address of a target interface.
 * @details    [localdata_if_mac]
 * @param[in]  p_if_mac  Local data to be queried.
 * @return     MAC address of the target interface.
 */
const uint8_t* demo_if_mac_ld_get_mac(const fpp_if_mac_cmd_t* p_if_mac)
{
    assert(NULL != p_if_mac);
    return (p_if_mac->mac);
}


/* ==== PUBLIC FUNCTIONS : misc ============================================ */


/*
 * @brief      Use FCI calls to iterate through all MAC addresses of a target interface
 *             in PFE. Execute a callback print function for each MAC address.
 * @details    To use this function properly, the interface database of PFE must be
 *             locked for exclusive access. See demo_phy_if_get_by_name_sa() for
 *             an example of a database lock procedure.
 * @param[in]  p_cl        FCI client
 * @param[in]  p_cb_print  Callback print function.
```

```
 *                          --> If the callback returns ZERO, then all is OK and
 *                              a next physical interface is picked for a print process.
 *                          --> If the callback returns NON-ZERO, then some problem is
 *                              assumed and this function terminates prematurely.
 * @param[in]   p_name     Name of a target physical interface.
 *                          Names of physical interfaces are hardcoded.
 *                          See FCI API Reference, chapter Interface Management.
 * @return     FPP_ERR_OK : Successfully iterated through all MAC addresses.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_if_mac_print_by_name(FCI_CLIENT* p_cl, demo_if_mac_cb_print_t p_cb_print,
                              const char* p_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);
    assert(NULL != p_name);


    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_if_mac_cmd_t cmd_to_fci = {0};
    fpp_if_mac_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_IF_MAC,
                        sizeof(fpp_if_mac_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

        /* query loop */
        while (FPP_ERR_OK == rtn)
        {
            rtn = p_cb_print(&reply_from_fci);

            print_if_error(rtn, "demo_if_mac_print_by_name() --> "
                                "non-zero return from callback print function!");

            if (FPP_ERR_OK == rtn)
            {
                cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
                rtn = fci_query(p_cl, FPP_CMD_IF_MAC,
                                sizeof(fpp_if_mac_cmd_t), (unsigned short*)(&cmd_to_fci),
                                &reply_length, (unsigned short*)(&reply_from_fci));
            }
        }

        /* query loop runs till there are no more MAC addresses to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_IF_MAC_NOT_FOUND == rtn)
        {
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_if_mac_print_by_name() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all MAC addresses of a target interface
 *              in PFE.
 * @details     To use this function properly, the interface database of PFE must be
 *              locked for exclusive access. See demo_phy_if_get_by_name_sa() for
 *              an example of a database lock procedure.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_count  Space to store the count of MAC addresses.
 * @param[in]   p_name       Name of a target physical interface.
 *                           Names of physical interfaces are hardcoded.
 *                           See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : Successfully counted all MAC addresses of the target interface.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_if_mac_get_count_by_name(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                                  const char* p_name)
{
```

```
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);
    assert(NULL != p_name);


    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_if_mac_cmd_t cmd_to_fci = {0};
    fpp_if_mac_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint32_t count = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_IF_MAC,
                        sizeof(fpp_if_mac_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

        /* query loop */
        while (FPP_ERR_OK == rtn)
        {
            count++;

            if (FPP_ERR_OK == rtn)
            {
                cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
                rtn = fci_query(p_cl, FPP_CMD_IF_MAC,
                                sizeof(fpp_if_mac_cmd_t), (unsigned short*)(&cmd_to_fci),
                                &reply_length, (unsigned short*)(&reply_from_fci));
            }
        }

        /* query loop runs till there are no more MAC addresses to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_IF_MAC_NOT_FOUND == rtn)
        {
            *p_rtn_count = count;
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_if_mac_get_count_by_name() failed!");

    return (rtn);
}


/* ========================================================================= */
```

15.15 demo_mirror.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
```

```
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_mirror.h"


/* ==== PRIVATE FUNCTIONS ================================================== */


/*
 * @brief       Set/unset a modification action flag in a mirroring rule struct.
 * @param[out]  p_rtn_mirror  Struct to be modified.
 * @param[in]   enable        New state of a modification action flag.
 * @param[in]   action        The 'modify action' flag.
 */
static void set_mirror_ma_flag(fpp_mirror_cmd_t* p_rtn_mirror, bool enable,
                               fpp_modify_actions_t action)
{
    assert(NULL != p_rtn_mirror);

    hton_enum(&action, sizeof(fpp_modify_actions_t));

    if (enable)
    {
        p_rtn_mirror->m_actions |= action;
    }
    else
    {
        p_rtn_mirror->m_actions &= (fpp_modify_actions_t)(~action);
    }
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from PFE ============== */


/*
 * @brief       Use FCI calls to get configuration data of a requested mirroring rule
 *              from PFE. Identify the rule by its name.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_mirror Space for data from PFE.
 * @param[in]   p_name       Name of the requested mirroring rule.
 *                           Names of mirroring rules are user-defined.
 *                           See demo_mirror_add().
 * @return      FPP_ERR_OK : The requested mirrroring rule was found.
 *                           A copy of its configuration data was stored into p_rtn_mirror.
 *                           REMINDER: data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_mirror_get_by_name(FCI_CLIENT* p_cl, fpp_mirror_cmd_t* p_rtn_mirror,
                            const char* p_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_mirror);
    assert(NULL != p_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_mirror_cmd_t cmd_to_fci = {0};
    fpp_mirror_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_MIRROR,
                        sizeof(fpp_mirror_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop (with a search condition) */
```

```
    while ((FPP_ERR_OK == rtn) && (0 != strcmp((reply_from_fci.name), p_name)))
    {
        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_MIRROR,
                        sizeof(fpp_mirror_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_mirror = reply_from_fci;
    }

    print_if_error(rtn, "demo_mirror_get_by_name() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to update data in PFE ============= */


/*
 * @brief          Use FCI calls to update configuration of a target mirroring rule
 *                 in PFE.
 * @param[in]      p_cl     FCI client
 * @param[in,out]  p_mirror Local data struct which represents a new configuration of
 *                          the target mirroring rule.
 *                          It is assumed that the struct contains a valid data of some
 *                          mirroring rule.
 * @return        FPP_ERR_OK : Configuration of the target mirroring rule was
 *                             successfully updated in PFE.
 *                             The local data struct was automatically updated with
 *                             readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                             The local data struct was not updated.
 */
int demo_mirror_update(FCI_CLIENT* p_cl, fpp_mirror_cmd_t* p_mirror)
{
    assert(NULL != p_cl);
    assert(NULL != p_mirror);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_mirror_cmd_t cmd_to_fci = (*p_mirror);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_MIRROR, sizeof(fpp_mirror_cmd_t),
                                          (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_mirror_get_by_name(p_cl, p_mirror, (p_mirror->name));
    }

    print_if_error(rtn, "demo_mirror_update() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to add/del items in PFE =========== */


/*
 * @brief       Use FCI calls to create a new mirroring rule in PFE.
 * @param[in]   p_cl           FCI client
 * @param[out]  p_rtn_logif    Space for data from PFE.
 *                             This will contain a copy of configuration data of
 *                             the newly created mirroring rule.
 *                             Can be NULL. If NULL, then there is no local data to fill.
 * @param[in]   p_name         Name of the new mirroring rule.
 *                             The name is user-defined.
 * @param[in]   p_phyif_name   Name of an egress physical interface.
 *                             Names of physical interfaces are hardcoded.
 *                             See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : New mirroring rule was created.
 *                           If applicable, then its configuration data were
 *                           copied into p_rtn_mirror.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_mirror_add(FCI_CLIENT* p_cl, fpp_mirror_cmd_t* p_rtn_mirror, const char* p_name,
                    const char* p_phyif_name)
```

```
{
    assert(NULL != p_cl);
    assert(NULL != p_name);
    assert(NULL != p_phyif_name);
    /* 'p_rtn_mirror' is allowed to be NULL */

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_mirror_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_name, MIRROR_NAME_SIZE);
    if (FPP_ERR_OK == rtn)
    {
        rtn = set_text((cmd_to_fci.egress_phy_if), p_phyif_name, IFNAMSIZ);
    }

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_REGISTER;
        rtn = fci_write(p_cl, FPP_CMD_MIRROR, sizeof(fpp_mirror_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));
    }

    /* read back and update caller data (if applicable) */
    if ((FPP_ERR_OK == rtn) && (NULL != p_rtn_mirror))
    {
        rtn = demo_mirror_get_by_name(p_cl, p_rtn_mirror, p_name);
    }

    print_if_error(rtn, "demo_mirror_add() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to destroy the target mirroring rule in PFE.
 * @param[in]  p_cl    FCI client
 * @param[in]  p_name  Name of the mirroring rule to destroy.
 * @return     FPP_ERR_OK : The mirroring rule was destroyed.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_mirror_del(FCI_CLIENT* p_cl, const char* p_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_mirror_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_name, MIRROR_NAME_SIZE);

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_DEREGISTER;
        rtn = fci_write(p_cl, FPP_CMD_MIRROR, sizeof(fpp_mirror_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_mirror_del() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : modify local data (no FCI calls) ================ */
/*
 * @defgroup    localdata_mirror  [localdata_mirror]
 * @brief:      Functions marked as [localdata_mirror] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_mirror (a struct with configuration data).
 *              Initial data for p_mirror can be obtained via demo_mirror_get_by_name().
 *              If some local data modifications are made, then after all local data changes
 *              are done and finished, call demo_mirror_update() to update
 *              the configuration of a real mirroring rule in PFE.
 */


/*
 * @brief          Set an egress physical interface of a mirroring rule.
 * @details        [localdata_mirror]
 * @param[in,out]  p_mirror       Local data to be modified.
 * @param[in]      p_mirror_name  Name of a physical interface which shall be used as egress.
```



```
 *                                Names of physical interfaces are hardcoded.
 *                                See the FCI API Reference, chapter Interface Management.
 */
void demo_mirror_ld_set_egress_phyif(fpp_mirror_cmd_t* p_mirror, const char* p_phyif_name)
{
    assert(NULL != p_mirror);
    assert(NULL != p_phyif_name);
    set_text((p_mirror->egress_phy_if), p_phyif_name, IFNAMSIZ);
}


/*
 * @brief          Set FlexibleParser table to act as a filter for a mirroring rule.
 * @details        [localdata_mirror]
 * @param[in,out]  p_mirror      Local data to be modified.
 * @param[in]      p_table_name  Name of a FlexibleParser table.
 *                               Can be NULL. If NULL or "" (empty string), then
 *                               filter of this mirroring rule is disabled.
 */
void demo_mirror_ld_set_filter(fpp_mirror_cmd_t* p_mirror, const char* p_table_name)
{
    assert(NULL != p_mirror);
    /* 'p_table_name' is allowed to be NULL */

    set_text(p_mirror->filter_table_name, p_table_name, IFNAMSIZ);
}




/*
 * @brief          Clear all modification actions of a mirroring rule.
 *                 (also zeroify all modification action arguments of the mirroring rule)
 * @details        [localdata_mirror]
 * @param[in,out]  p_mirror  Local data to be modified.
 */
void demo_mirror_ld_clear_all_ma(fpp_mirror_cmd_t* p_mirror)
{
    assert(NULL != p_mirror);
    p_mirror->m_actions = 0u;
    memset(&(p_mirror->m_args), 0, sizeof(fpp_modify_args_t));
}


/*
 * @brief          Set/unset the given modification action (ADD_VLAN_HDR) and its argument.
 * @details        [localdata_mirror]
 * @param[in,out]  p_mirror  Local data to be modified.
 * @param[in]      set       Request to set/unset the given match rule.
 * @param[in]      vlan      New VLAN ID for this match rule.
 *                           When this match rule is active, it compares value of its
 *                           'vlan' argument with the value of traffic's 'VID' field.
 */
void demo_mirror_ld_set_ma_vlan(fpp_mirror_cmd_t* p_mirror, bool set, uint16_t vlan)
{
    assert(NULL != p_mirror);

    set_mirror_ma_flag(p_mirror, set, MODIFY_ACT_ADD_VLAN_HDR);
    p_mirror->m_args.vlan = htons(vlan);
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */


/*
 * @brief      Query whether a modification action is active or not.
 * @details    [localdata_mirror]
 * @param[in]  p_mirror  Local data to be queried.
 * @param[in]  action    Queried 'modify action'.
 * @return     At time when the data was obtained from PFE, the mirroring rule:
 *             true  : had at least one queried 'modify action' bitflags set
 *             false : had none of the queried 'modify action' bitflags set
 */
bool demo_mirror_ld_is_ma(const fpp_mirror_cmd_t* p_mirror,
                          fpp_modify_actions_t action)
{
    assert(NULL != p_mirror);

    fpp_modify_actions_t tmp_actions = (p_mirror->m_actions);
    ntoh_enum(&tmp_actions, sizeof(fpp_modify_actions_t));

    return (bool)(tmp_actions & action);
}
```

```
/*
 * @brief      Query the name of a mirroring rule.
 * @details    [localdata_mirror]
 * @param[in]  p_mirror  Local data to be queried.
 * @return     Name of the mirroring rule.
 */
const char* demo_mirror_ld_get_name(const fpp_mirror_cmd_t* p_mirror)
{
    assert(NULL != p_mirror);
    return (p_mirror->name);
}


/*
 * @brief      Query the egress interface of a mirroring rule.
 * @details    [localdata_mirror]
 * @param[in]  p_mirror  Local data to be queried.
 * @return     Name of a physical interface which is used as an egress interface
 *             of the mirroring rule.
 */
const char* demo_mirror_ld_get_egress_phyif(const fpp_mirror_cmd_t* p_mirror)
{
    assert(NULL != p_mirror);
    return (p_mirror->egress_phy_if);
}


/*
 * @brief      Query the name of a FlexibleParser table which is being used as
 *             a filter for a mirroring rule.
 * @details    [localdata_mirror]
 * @param[in]  p_mirror  Local data to be queried.
 * @return     Name of the FlexibleParser table which is used as a filter
 *             of the mirroring rule.
 */
const char* demo_mirror_ld_get_filter(const fpp_mirror_cmd_t* p_mirror)
{
    assert(NULL != p_mirror);
    return (p_mirror->filter_table_name);
}




/*
 * @brief      Query the modification action bitset of a mirroring rule.
 * @details    [localdata_mirror]
 * @param[in]  p_mirror  Local data to be queried.
 * @return     'Modify action' bitset.
 */
fpp_modify_actions_t demo_mirror_ld_get_ma_bitset(const fpp_mirror_cmd_t* p_mirror)
{
    assert(NULL != p_mirror);

    fpp_modify_actions_t tmp_actions = (p_mirror->m_actions);
    ntoh_enum(&tmp_actions, sizeof(fpp_modify_actions_t));

    return (tmp_actions);
}


/*
 * @brief      Query the argument of the modification action match rule ADD_VLAN_HDR.
 * @details    [localdata_mirror]
 * @param[in]  p_mirror  Local data to be queried.
 * @return     Argument (VLAN ID) of the given modification action.
 */
uint16_t demo_mirror_ld_get_ma_vlan(const fpp_mirror_cmd_t* p_mirror)
{
    assert(NULL != p_mirror);
    return ntohs(p_mirror->m_args.vlan);
}


/* ==== PUBLIC FUNCTIONS : misc =========================================== */


/*
 * @brief      Use FCI calls to iterate through all available mirroring rules in PFE and
 *             execute a callback print function for each mirroring rule.
 * @param[in]  p_cl        FCI client
 * @param[in]  p_cb_print  Callback print function.
 *                         --> If the callback returns ZERO, then all is OK and
 *                             a next mirroring rule is picked for a print process.
```

```
 *                          --> If the callback returns NON-ZERO, then some problem is
 *                              assumed and this function terminates prematurely.
 * @return     FPP_ERR_OK : Successfully iterated through all available mirroring rules.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_mirror_print_all(FCI_CLIENT* p_cl, demo_mirror_cb_print_t p_cb_print)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_mirror_cmd_t cmd_to_fci = {0};
    fpp_mirror_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_MIRROR,
                    sizeof(fpp_mirror_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        rtn = p_cb_print(&reply_from_fci);

        if (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_MIRROR,
                            sizeof(fpp_mirror_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more mirroring rules to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_MIRROR_NOT_FOUND == rtn)
    {
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_mirror_print_all() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available mirroring rules in PFE.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_count  Space to store the count of mirroring rules.
 * @return      FPP_ERR_OK : Successfully counted all available mirroring rules.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No value copied.
 */
int demo_mirror_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_mirror_cmd_t cmd_to_fci = {0};
    fpp_mirror_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint16_t count = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_MIRROR,
                    sizeof(fpp_mirror_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        count++;

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_MIRROR,
                        sizeof(fpp_mirror_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }
```

```
    /* query loop runs till there are no more mirroring rules to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_MIRROR_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_mirror_get_count() failed!");

    return (rtn);
}


/* ========================================================================= */
```

15.16 demo_l2_bd.c

```
/* =========================================================================
 *  Copyright 2020-2022 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_l2_bd.h"


/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from PFE ============== */


/*
 * @brief       Use FCI calls to get configuration data of a requested bridge domain
 *              from PFE. Identify the domain by its VLAN ID.
 * @param[in]   p_cl      FCI client
 * @param[out]  p_rtn_bd  Space for data from PFE.
 * @param[in]   vlan      VLAN ID of the requested bridge domain.
 * @return      FPP_ERR_OK : The requested bridge domain was found.
 *                           A copy of its configuration data was stored into p_rtn_bd.
 *                           REMINDER: data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_l2_bd_get_by_vlan(FCI_CLIENT* p_cl, fpp_l2_bd_cmd_t* p_rtn_bd, uint16_t vlan)
```



```
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_bd);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_l2_bd_cmd_t cmd_to_fci = {0};
    fpp_l2_bd_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_L2_BD,
                    sizeof(fpp_l2_bd_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop (with a search condition) */
    while ((FPP_ERR_OK == rtn) && (ntohs(reply_from_fci.vlan) != vlan))
    {
        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_L2_BD,
                        sizeof(fpp_l2_bd_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_bd = reply_from_fci;
    }

    print_if_error(rtn, "demo_l2_bd_get_by_vlan() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get configuration data of a requested static entry
 *              from PFE. Identify the entry by VLAN ID of the parent bridge domain and
 *              by MAC address of the entry.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_stent  Space for data from PFE.
 * @param[in]   vlan         VLAN ID of the parent bridge domain.
 * @param[in]   p_mac        MAC address of the requested static entry.
 * @return      FPP_ERR_OK : The requested static entry was found.
 *                           A copy of its configuration data was stored into p_rtn_stent.
 *                           REMINDER: data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_l2_stent_get_by_vlanmac(FCI_CLIENT* p_cl, fpp_l2_static_ent_cmd_t* p_rtn_stent,
                                 uint16_t vlan, const uint8_t p_mac[6])
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_stent);
    assert(NULL != p_mac);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_l2_static_ent_cmd_t cmd_to_fci = {0};
    fpp_l2_static_ent_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_L2_STATIC_ENT,
                    sizeof(fpp_l2_static_ent_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop (with a search condition) */
    while ((FPP_ERR_OK == rtn) &&
            !(
                (ntohs(reply_from_fci.vlan) == vlan) &&
                (0 == memcmp((reply_from_fci.mac), p_mac, 6))
             )
          )
    {
        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_L2_STATIC_ENT,
                        sizeof(fpp_l2_static_ent_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
```

```
    {
        *p_rtn_stent = reply_from_fci;
    }

    print_if_error(rtn, "demo_l2_stent_get_by_vlanmac() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to update data in PFE ============= */


/*
 * @brief          Use FCI calls to update configuration of a target bridge domain
 *                 in PFE.
 * @param[in]      p_cl  FCI client
 * @param[in,out]  p_bd  Local data struct which represents a new configuration of
 *                       the target bridge domain.
 *                       It is assumed that the struct contains a valid data of some
 *                       bridge domain.
 * @return        FPP_ERR_OK : Configuration of the target bridge domain was
 *                             successfully updated in PFE.
 *                             The local data struct was automatically updated with
 *                             readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                             The local data struct was not updated.
 */
int demo_l2_bd_update(FCI_CLIENT* p_cl, fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_cl);
    assert(NULL != p_bd);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_l2_bd_cmd_t cmd_to_fci = (*p_bd);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_L2_BD, sizeof(fpp_l2_bd_cmd_t),
                                        (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_l2_bd_get_by_vlan(p_cl, p_bd, ntohs(p_bd->vlan));
    }

    print_if_error(rtn, "demo_l2_bd_update() failed!");

    return (rtn);
}


/*
 * @brief          Use FCI calls to update configuration of a target static entry
 *                 in PFE.
 * @param[in]      p_cl     FCI client
 * @param[in,out]  p_stent  Local data struct which represents a new configuration of
 *                          the target static entry.
 *                          It is assumed that the struct contains a valid data of some
 *                          static entry.
 * @return        FPP_ERR_OK : Configuration of the target static entry was
 *                             successfully updated in PFE.
 *                             The local data struct was automatically updated with
 *                             readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                             Local data struct not updated.
 */
int demo_l2_stent_update(FCI_CLIENT* p_cl, fpp_l2_static_ent_cmd_t* p_stent)
{
    assert(NULL != p_cl);
    assert(NULL != p_stent);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_l2_static_ent_cmd_t cmd_to_fci = (*p_stent);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_L2_STATIC_ENT, sizeof(fpp_l2_static_ent_cmd_t),
                                                (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_l2_stent_get_by_vlanmac(p_cl, p_stent,
                                           ntohs(p_stent->vlan), (p_stent->mac));
```

```
    }

    print_if_error(rtn, "demo_l2_stent_update() failed!");

    return (rtn);
}


/*
 * @brief         Use FCI calls to flush static entries from MAC tables of
 *                all bridge domains in PFE.
 * @param[in]     p_cl  FCI client
 * @return        FPP_ERR_OK : Static MAC table entries of all bridge domains were
 *                             successfully flushed in PFE.
 *                other      : Some error occurred (represented by the respective error code).
 */
int demo_l2_flush_static(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = fci_write(p_cl, FPP_CMD_L2_FLUSH_STATIC, 0u, NULL);
    print_if_error(rtn, "demo_l2_flush_static() failed!");
    return (rtn);
}


/*
 * @brief         Use FCI calls to flush dynamically learned entries from MAC tables of
 *                all bridge domains in PFE.
 * @param[in]     p_cl  FCI client
 * @return        FPP_ERR_OK : Learned MAC table entries of all bridge domains were
 *                             successfully flushed in the PFE.
 *                other      : Some error occurred (represented by the respective error code).
 */
int demo_l2_flush_learned(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = fci_write(p_cl, FPP_CMD_L2_FLUSH_LEARNED, 0u, NULL);
    print_if_error(rtn, "demo_l2_flush_learned() failed!");
    return (rtn);
}


/*
 * @brief         Use FCI calls to flush all entries from MAC tables of
 *                all bridge domains in PFE.
 * @param[in]     p_cl  FCI client
 * @return        FPP_ERR_OK : All MAC table entries of all bridge domains were
 *                             successfully flushed in the PFE.
 *                other      : Some error occurred (represented by the respective error code).
 */
int demo_l2_flush_all(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = fci_write(p_cl, FPP_CMD_L2_FLUSH_ALL, 0u, NULL);
    print_if_error(rtn, "demo_l2_flush_all() failed!");
    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to add/del items in PFE =========== */


/*
 * @brief       Use FCI calls to create a new bridge domain in PFE.
 * @param[in]   p_cl      FCI client
 * @param[out]  p_rtn_if  Space for data from PFE.
 *                        This will contain a copy of configuration data of
 *                        the newly created bridge domain.
 *                        Can be NULL. If NULL, then there is no local data to fill.
 * @param[in]   vlan      VLAN ID of the new bridge domain.
 * @return      FPP_ERR_OK : New bridge domain was created.
 *                           If applicable, then its configuration data were
 *                           copied into p_rtn_bd.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_l2_bd_add(FCI_CLIENT* p_cl, fpp_l2_bd_cmd_t* p_rtn_bd, uint16_t vlan)
{
    assert(NULL != p_cl);
    /* 'p_rtn_bd' is allowed to be NULL */

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_l2_bd_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci.vlan = htons(vlan);
```



```
    cmd_to_fci.ucast_hit  = 3u;  /* 3 == discard */
    cmd_to_fci.ucast_miss = 3u;  /* 3 == discard */
    cmd_to_fci.mcast_hit  = 3u;  /* 3 == discard */
    cmd_to_fci.mcast_miss = 3u;  /* 3 == discard */

    /* send data */
    cmd_to_fci.action = FPP_ACTION_REGISTER;
    rtn = fci_write(p_cl, FPP_CMD_L2_BD, sizeof(fpp_l2_bd_cmd_t),
                                    (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data (if applicable) */
    if ((FPP_ERR_OK == rtn) && (NULL != p_rtn_bd))
    {
        rtn = demo_l2_bd_get_by_vlan(p_cl, p_rtn_bd, vlan);
    }

    print_if_error(rtn, "demo_l2_bd_add() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to destroy the target bridge domain in PFE.
 * @param[in]  p_cl    FCI client
 * @param[in]  vlan    VLAN ID of the bridge domain to destroy.
 *                     NOTE: Bridge domains marked as "default" or "fallback"
 *                           cannot be destroyed.
 * @return     FPP_ERR_OK : The bridge domain was destroyed.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_l2_bd_del(FCI_CLIENT* p_cl, uint16_t vlan)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_l2_bd_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci.vlan = htons(vlan);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_DEREGISTER;
    rtn = fci_write(p_cl, FPP_CMD_L2_BD, sizeof(fpp_l2_bd_cmd_t),
                                    (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_l2_bd_del() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to create a new static entry in PFE.
 *              The new entry is associated with a provided parent bridge domain.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_stent  Space for data from PFE.
 *                           This will contain a copy of configuration data of
 *                           the newly created static entry.
 *                           Can be NULL. If NULL, then there is no local data to fill.
 * @param[in]   vlan         VLAN ID of the parent bridge domain.
 * @param[in]   p_mac        MAC address of the new static entry.
 * @return      FPP_ERR_OK : New static entry was created.
 *                           If applicable, then its configuration data were
 *                           copied into p_rtn_stent.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_l2_stent_add(FCI_CLIENT* p_cl, fpp_l2_static_ent_cmd_t* p_rtn_stent,
                      uint16_t vlan, const uint8_t p_mac[6])
{
    assert(NULL != p_cl);
    assert(NULL != p_mac);
    /* 'p_rtn_stent' is allowed to be NULL */

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_l2_static_ent_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci.vlan = htons(vlan);
    memcpy(cmd_to_fci.mac, p_mac, 6);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_REGISTER;
    rtn = fci_write(p_cl, FPP_CMD_L2_STATIC_ENT, sizeof(fpp_l2_static_ent_cmd_t),
                                            (unsigned short*)(&cmd_to_fci));
```

```
    /* read back and update caller data (if applicable) */
    if ((FPP_ERR_OK == rtn) && (NULL != p_rtn_stent))
    {
        rtn = demo_l2_stent_get_by_vlanmac(p_cl, p_rtn_stent, vlan, p_mac);
    }

    print_if_error(rtn, "demo_l2_stent_add() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to destroy the target static entry in PFE.
 * @param[in]  p_cl    FCI client
 * @param[in]  vlan    VLAN ID of the parent bridge domain.
 * @param[in]  p_mac   MAC address of the static entry to destroy.
 * @return     FPP_ERR_OK : The static entry was destroyed.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_l2_stent_del(FCI_CLIENT* p_cl, uint16_t vlan, const uint8_t p_mac[6])
{
    assert(NULL != p_cl);
    assert(NULL != p_mac);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_l2_static_ent_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci.vlan = htons(vlan);
    memcpy(cmd_to_fci.mac, p_mac, 6);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_DEREGISTER;
    rtn = fci_write(p_cl, FPP_CMD_L2_STATIC_ENT, sizeof(fpp_l2_static_ent_cmd_t),
                                                 (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_l2_stent_del() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : modify local data (no FCI calls) ================ */
/*
 * @defgroup    localdata_bd  [localdata_bd]
 * @brief:      Functions marked as [localdata_bd] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_bd (a struct with configuration data).
 *              Initial data for p_bd can be obtained via demo_l2_bd_get_by_vlan().
 *              If some local data modifications are made, then after all local data changes
 *              are done and finished, call demo_l2_bd_update() to update
 *              the configuration of a real bridge domain in PFE.
 */


/*
 * @brief          Set action to be done if unicast packet's destination MAC is
 *                 found (hit) in a bridge domain's MAC table.
 * @details        [localdata_bd]
 * @param[in,out]  p_bd         Local data to be modified.
 * @param[in]      hit_action   New action.
 *                              For details about bridge domain hit/miss actions,
 *                              see a description of the ucast_hit in FCI API Reference.
 */
void demo_l2_bd_ld_set_ucast_hit(fpp_l2_bd_cmd_t* p_bd, uint8_t hit_action)
{
    assert(NULL != p_bd);
    p_bd->ucast_hit = hit_action;
}


/*
 * @brief          Set action to be done if unicast packet's destination MAC is NOT
 *                 found (miss) in a bridge domain's MAC table.
 * @details        [localdata_bd]
 * @param[in,out]  p_bd         Local data to be modified.
 * @param[in]      miss_action  New action.
 *                              For details about bridge domain hit/miss actions,
 *                              see a description of the ucast_hit in FCI API Reference.
 */
void demo_l2_bd_ld_set_ucast_miss(fpp_l2_bd_cmd_t* p_bd, uint8_t miss_action)
{
    assert(NULL != p_bd);
    p_bd->ucast_miss = miss_action;
```

```
}


/*
 * @brief          Set action to be done if multicast packet's destination MAC is
 *                 found (hit) in a bridge domain's MAC table.
 * @details        [localdata_bd]
 * @param[in,out]  p_bd         Local data to be modified.
 * @param[in]      hit_action   New action.
 *                              For details about bridge domain hit/miss actions,
 *                              see a description of the ucast_hit in FCI API Reference.
 */
void demo_l2_bd_ld_set_mcast_hit(fpp_l2_bd_cmd_t* p_bd, uint8_t hit_action)
{
    assert(NULL != p_bd);
    p_bd->mcast_hit = hit_action;
}


/*
 * @brief          Set action to be done if multicast packet's destination MAC is NOT
 *                 found (miss) in a bridge domain's MAC table.
 * @details        [localdata_bd]
 * @param[in,out]  p_bd         Local data to be modified.
 * @param[in]      hit_action   New action.
 *                              For details about bridge domain hit/miss actions,
 *                              see a description of the ucast_hit in FCI API Reference.
 */
void demo_l2_bd_ld_set_mcast_miss(fpp_l2_bd_cmd_t* p_bd, uint8_t miss_action)
{
    assert(NULL != p_bd);
    p_bd->mcast_miss = miss_action;
}


/*
 * @brief          Insert a physical interface into a bridge domain.
 * @details        [localdata_bd]
 * @param[in,out]  p_bd      Local data to be modified.
 * @param[in]      phyif_id  ID of the physical interface.
 *                           IDs of physical interfaces are hardcoded.
 *                           See FCI API Reference, chapter Interface Management.
 * @param[in]      vlan_tag  Request to add/keep a VLAN tag (true) or to remove
 *                           the VLAN tag (false) of a traffic egressed through
 *                           the given physical interface.
 */
void demo_l2_bd_ld_insert_phyif(fpp_l2_bd_cmd_t* p_bd, uint32_t phyif_id, bool vlan_tag)
{
    assert(NULL != p_bd);

    if (32uL > phyif_id)  /* a check to prevent an undefined behavior */
    {
        const uint32_t phyif_bitmask = (1uL << phyif_id);

        p_bd->if_list |= htonl(phyif_bitmask);

        if (vlan_tag)
        {
            /* VLAN TAG is desired == physical interface must NOT be on the untag list. */
            p_bd->untag_if_list &= htonl((uint32_t)(~phyif_bitmask));
        }
        else
        {
            /* VLAN TAG is NOT desired == physical interface must BE on the untag list. */
            p_bd->untag_if_list |= htonl(phyif_bitmask);
        }
    }
}


/*
 * @brief          Remove a physical interface from a bridge domain.
 * @details        [localdata_bd]
 * @param[in,out]  p_bd      Local data to be modified.
 * @param[in]      phyif_id  ID of the physical interface.
 *                           IDs of physical interfaces are hardcoded.
 *                           See FCI API Reference, chapter Interface Management.
 */
void demo_l2_bd_ld_remove_phyif(fpp_l2_bd_cmd_t* p_bd, uint32_t phyif_id)
{
    assert(NULL != p_bd);

    if (32uL > phyif_id)  /* a check to prevent an undefined behavior */
    {
        const uint32_t phyif_bitmask = (1uL << phyif_id);
        p_bd->if_list &= htonl((uint32_t)(~phyif_bitmask));
```

```
    }
}




/*
 * @defgroup    localdata_stent  [localdata_stent]
 * @brief:      Functions marked as [localdata_stent] acess only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_stent (a struct with configuration data).
 *              Initial data for p_stent can be obtained via demo_l2_stent_get_by_vlanmac().
 *              If some local data modifications are made, then after all local data changes
 *              are done and finished, call demo_l2_stent_update() to update
 *              the configuration of a real static entry in PFE.
 */


/*
 * @brief          Set target physical interfaces (forwarding list) which
 *                 shall receive a copy of the accepted traffic.
 * @details        [localdata_stent]
 *                 New forwarding list fully replaces the old one.
 * @param[in,out]  p_stent  Local data to be modified.
 * @param[in]      fwlist   Target physical interfaces (forwarding list). A bitset.
 *                          Each physical interface is represented by one bit.
 *                          Conversion between physical interface ID and a corresponding
 *                          fwlist bit is (1uL << "ID of a target physical interface").
 */
void demo_l2_stent_ld_set_fwlist(fpp_l2_static_ent_cmd_t* p_stent, uint32_t fwlist)
{
    assert(NULL != p_stent);
    p_stent->forward_list = htonl(fwlist);
}


/*
 * @brief          Set/unset 'local' flag of a static entry.
 * @details        [localdata_stent]
 *                 Related to L2L3 Bridge feature (see FCI API Reference).
 * @param[in,out]  p_stent  Local data to be modified.
 * @param[in]      set      Request to set/unset the flag.
 *                          See description of the fpp_l2_static_ent_cmd_t type
 *                          in FCI API reference.
 */
void demo_l2_stent_ld_set_local(fpp_l2_static_ent_cmd_t* p_stent, bool set)
{
    assert(NULL != p_stent);
    p_stent->local = set;  /* NOTE: implicit cast from bool to uint8_t */
}


/*
 * @brief          Set/unset a flag for a frame discarding feature tied with a static entry.
 * @details        [localdata_stent]
 * @param[in,out]  p_stent  Local data to be modified.
 * @param[in]      set      Request to set/unset the flag.
 *                          See description of fpp_l2_static_ent_cmd_t type
 *                          in FCI API reference.
 */
void demo_l2_stent_ld_set_src_discard(fpp_l2_static_ent_cmd_t* p_stent, bool set)
{
    assert(NULL != p_stent);
    p_stent->src_discard = set;  /* NOTE: implicit cast from bool to uint8_t */
}


/*
 * @brief          Set/unset a flag for a frame discarding feature tied with a static entry.
 * @details        [localdata_stent]
 * @param[in,out]  p_stent  Local data to be modified.
 * @param[in]      set      Request to set/unset the flag.
 *                          See description of fpp_l2_static_ent_cmd_t type
 *                          in FCI API reference.
 */
void demo_l2_stent_ld_set_dst_discard(fpp_l2_static_ent_cmd_t* p_stent, bool set)
{
    assert(NULL != p_stent);
    p_stent->dst_discard = set;  /* NOTE: implicit cast from bool to uint8_t */
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */


/*
```

```
 * @brief      Query status of a "default" flag.
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the bridge domain:
 *             true  : was set as a default domain.
 *             false : was NOT set as a default domain.
 */
bool demo_l2_bd_ld_is_default(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);

    fpp_l2_bd_flags_t tmp_flags = (p_bd->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_l2_bd_flags_t));

    return (bool)(tmp_flags & FPP_L2_BD_DEFAULT);
}


/*
 * @brief      Query status of a "fallback" flag.
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the bridge domain:
 *             true  : was set as a fallback domain.
 *             false : was NOT set as a fallback domain.
 */
bool demo_l2_bd_ld_is_fallback(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);

    fpp_l2_bd_flags_t tmp_flags = (p_bd->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_l2_bd_flags_t));

    return (bool)(tmp_flags & FPP_L2_BD_FALLBACK);
}


/*
 * @brief      Query whether a physical interface is a member of a bridge domain.
 * @details    [localdata_bd]
 * @param[in]  p_bd      Local data to be queried.
 * @param[in]  phyif_id  ID of the physical interface.
 *                       IDs of physical interfaces are hardcoded.
 *                       See FCI API Reference, chapter Interface Management.
 * @return     At time when the data was obtained from PFE, the requested physical interface:
 *             true  : was a member of the given bridge domain.
 *             false : was NOT a member of the given bridge domain.
 */
bool demo_l2_bd_ld_has_phyif(const fpp_l2_bd_cmd_t* p_bd, uint32_t phyif_id)
{
    assert(NULL != p_bd);

    bool rtn = false;

    if (32uL > phyif_id)
    {
        const uint32_t phyif_bitmask = (1uL << phyif_id);
        rtn = (bool)(ntohl(p_bd->if_list) & phyif_bitmask);
    }

    return (rtn);
}


/*
 * @brief      Query whether traffic from a physical interface is tagged by a bridge domain.
 *             This function returns meaningful results only if
 *             the target physical interface is a member of the bridge domain.
 *             See demo_l2_bd_ld_has_phyif().
 * @details    [localdata_bd]
 * @param[in]  p_bd      Local data to be queried.
 * @param[in]  phyif_id  ID of the physical interface.
 *                       IDs of physical interfaces are hardcoded.
 *                       See FCI API Reference, chapter Interface Management.
 * @return     At time when the data was obtained from PFE, traffic from
 *             the requested physical interface:
 *             true  : was being VLAN tagged by the given bridge domain.
 *             false : was NOT being VLAN tagged by the given bridge domain.
 */
bool demo_l2_bd_ld_is_phyif_tagged(const fpp_l2_bd_cmd_t* p_bd, uint32_t phyif_id)
{
    assert(NULL != p_bd);

    bool rtn = false;
    if (32uL > phyif_id)
    {
```

```
        /* untag_list uses inverted logic - if interface IS on the list, it is UNTAGGED */
        const uint32_t phyif_bitmask = (1uL << phyif_id);
        rtn = !(ntohl(p_bd->untag_if_list) & phyif_bitmask);
    }
    return (rtn);
}



/*
 * @brief      Query the VLAN ID of a bridge domain.
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     VLAN ID of the bridge domain.
 */
uint16_t demo_l2_bd_ld_get_vlan(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return ntohs(p_bd->vlan);
}


/*
 * @brief      Query the bridge action which is executed on unicast hit.
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Bridge action (see a description of the ucast_hit in FCI API Reference).
 */
uint8_t demo_l2_bd_ld_get_ucast_hit(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return (p_bd->ucast_hit);
}


/*
 * @brief      Query the bridge action which is executed on unicast miss.
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Bridge action (see a description of the ucast_hit in FCI API Reference).
 */
uint8_t demo_l2_bd_ld_get_ucast_miss(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return (p_bd->ucast_miss);
}


/*
 * @brief      Query the bridge action which is executed on multicast hit.
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Bridge action (see a description of the ucast_hit in FCI API Reference).
 */
uint8_t demo_l2_bd_ld_get_mcast_hit(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return (p_bd->mcast_hit);
}


/*
 * @brief      Query the bridge action which is executed on multicast miss.
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Bridge action (see a description of the ucast_hit in FCI API Reference).
 */
uint8_t demo_l2_bd_ld_get_mcast_miss(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return (p_bd->mcast_miss);
}


/*
 * @brief      Query the list of member physical interfaces of a bridge domain.
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Bitset with physical interfaces being represented as bits.
 */
uint32_t demo_l2_bd_ld_get_if_list(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return ntohl(p_bd->if_list);
}
```

```
/*
 * @brief      Query the untag list of a bridge domain.
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Bitset with physical interfaces being represented as bits.
 */
uint32_t demo_l2_bd_ld_get_untag_if_list(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return ntohl(p_bd->untag_if_list);
}


/*
 * @brief      Query the flags of a bridge domain (the whole bitset).
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Flags bitset.
 */
fpp_l2_bd_flags_t demo_l2_bd_ld_get_flags(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);

    fpp_l2_bd_flags_t tmp_flags = (p_bd->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_l2_bd_flags_t));

    return (tmp_flags);
}


/*
 * @brief      Query the domain traffic statistics - ingress
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Count of ingress packets at the time when the data was obtained from PFE.
 */
uint32_t demo_l2_bd_ld_get_stt_ingress(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return ntohl(p_bd->stats.ingress);
}


/*
 * @brief      Query the domain traffic statistics - ingress in bytes
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Number of ingress bytes at the time when the data was obtained from PFE.
 */
uint32_t demo_l2_bd_ld_get_stt_ingress_bytes(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return ntohl(p_bd->stats.ingress_bytes);
}


/*
 * @brief      Query the domain traffic statistics - egress
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Count of egress packets at the time when the data was obtained from PFE.
 */
uint32_t demo_l2_bd_ld_get_stt_egress(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return ntohl(p_bd->stats.egress);
}


/*
 * @brief      Query the domain traffic statistics - egress in bytes
 * @details    [localdata_bd]
 * @param[in]  p_bd  Local data to be queried.
 * @return     Number of egress bytes at the time when the data was obtained from PFE.
 */
uint32_t demo_l2_bd_ld_get_stt_egress_bytes(const fpp_l2_bd_cmd_t* p_bd)
{
    assert(NULL != p_bd);
    return ntohl(p_bd->stats.egress_bytes);
}
```

```
/*
 * @brief      Query whether a physical interface is a member of
 *             a static entry's forwarding list.
 * @details    [localdata_stent]
 * @param[in]  p_stent  Local data to be queried.
 * @param[in]  fwlist_bitflag  Queried physical interface. A bitflag.
 *                             Each physical interface is represented by one bit.
 *                             Conversion between physical interface ID and a corresponding
 *                             fwlist bit is (1uL << "ID of a target physical interface").
 *                             Hint: It is recommended to always query only a single bitflag.
 * @return     At time when the data was obtained from PFE, the static entry:
 *             true  : had at least one queried forward list bitflag set
 *             false : had none of the queried forward list bitflags set
 */
bool demo_l2_stent_ld_is_fwlist_phyifs(const fpp_l2_static_ent_cmd_t* p_stent,
                                       uint32_t fwlist_bitflag)
{
    assert(NULL != p_stent);
    return (bool)(ntohl(p_stent->forward_list) & fwlist_bitflag);
}


/*
 * @brief      Query status of the "local" flag of a static entry.
 * @details    [localdata_stent]
 * @param[in]  p_stent  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the static entry:
 *             true  : was set as local.
 *             false : was NOT set as local.
 */
bool demo_l2_stent_ld_is_local(const fpp_l2_static_ent_cmd_t* p_stent)
{
    assert(NULL != p_stent);
    return (bool)(p_stent->local);
}


/*
 * @brief      Query status of the "src_discard" flag of a static entry.
 * @details    [localdata_stent]
 * @param[in]  p_stent  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the static entry:
 *             true  : was set to discard ETH frames with a matching source MAC.
 *             false : was NOT set to discard ETH frames with a matching source MAC.
 */
bool demo_l2_stent_ld_is_src_discard(const fpp_l2_static_ent_cmd_t* p_stent)
{
    assert(NULL != p_stent);
    return (bool)(p_stent->src_discard);
}


/*
 * @brief      Query status of the "dst_discard" flag of a static entry.
 * @details    [localdata_stent]
 * @param[in]  p_stent  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the static entry:
 *             true  : was set to discard ETH frames with a matching destination MAC.
 *             false : was NOT set to discard ETH frames with a matching destination MAC.
 */
bool demo_l2_stent_ld_is_dst_discard(const fpp_l2_static_ent_cmd_t* p_stent)
{
    assert(NULL != p_stent);
    return (bool)(p_stent->dst_discard);
}




/*
 * @brief      Query the VLAN ID of a static entry.
 * @details    [localdata_stent]
 * @param[in]  p_stent  Local data to be queried.
 * @return     VLAN ID of the static entry.
 */
uint16_t demo_l2_stent_ld_get_vlan(const fpp_l2_static_ent_cmd_t* p_stent)
{
    assert(NULL != p_stent);
    return ntohs(p_stent->vlan);
}


/*
 * @brief      Query the MAC address of a static entry.
 * @details    [localdata_stent]
 * @param[in]  p_stent  Local data to be queried.
```

```
 * @return     MAC address of the static entry.
 */
const uint8_t* demo_l2_stent_ld_get_mac(const fpp_l2_static_ent_cmd_t* p_stent)
{
    assert(NULL != p_stent);
    return (p_stent->mac);
}


/*
 * @brief      Query the forwarding list (a bitset) of a static entry.
 * @details    [localdata_stent]
 * @param[in]  p_stent  Local data to be queried.
 * @return     Bitset with physical interfaces being represented as bits.
 */
uint32_t demo_l2_stent_ld_get_fwlist(const fpp_l2_static_ent_cmd_t* p_stent)
{
    assert(NULL != p_stent);
    return ntohl(p_stent->forward_list);
}


/* ==== PUBLIC FUNCTIONS : misc ============================================ */


/*
 * @brief      Use FCI calls to iterate through all available bridge domains in PFE and
 *             execute a callback print function for each bridge domain.
 * @param[in]  p_cl        FCI client
 * @param[in]  p_cb_print  Callback print function.
 *                         --> If the callback returns ZERO, then all is OK and
 *                             a next bridge domain is picked for a print process.
 *                         --> If the callback returns NON-ZERO, then some problem is
 *                             assumed and this function terminates prematurely.
 * @return     FPP_ERR_OK : Successfully iterated through all available bridge domains.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_l2_bd_print_all(FCI_CLIENT* p_cl, demo_l2_bd_cb_print_t p_cb_print)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_l2_bd_cmd_t cmd_to_fci = {0};
    fpp_l2_bd_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_L2_BD,
                    sizeof(fpp_l2_bd_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        rtn = p_cb_print(&reply_from_fci);

        if (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_L2_BD,
                            sizeof(fpp_l2_bd_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more bridge domains to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_L2_BD_NOT_FOUND == rtn)
    {
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_l2_bd_print_all() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available bridge domains in PFE.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_count  Space to store the count of bridge domains.
 * @return      FPP_ERR_OK : Successfully counted all available bridge domains.
```

```
 *                            Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                            No value copied.
 */
int demo_l2_bd_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_l2_bd_cmd_t cmd_to_fci = {0};
    fpp_l2_bd_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint16_t count = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_L2_BD,
                    sizeof(fpp_l2_bd_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        count++;

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_L2_BD,
                        sizeof(fpp_l2_bd_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* query loop runs till there are no more bridge domains to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_L2_BD_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_l2_bd_get_count() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to iterate through all available static entries in PFE and
 *             execute a callback print function for each applicable static entry.
 * @param[in]  p_cl        FCI client instance
 * @param[in]  p_cb_print  Callback print function.
 *                         --> If the callback returns ZERO, then all is OK and
 *                             a next static entry is picked for a print process.
 *                         --> If the callback returns NON-ZERO, then some problem is
 *                             assumed and this function terminates prematurely.
 * @param[in]  by_vlan     [optional parameter]
 *                          Request to print only those static entries
 *                          which are associated with a particular bridge domain.
 * @param[in]  vlan        [optional parameter]
 *                          VLAN ID of a bridge domain.
 *                          Applicable only if (true == by_vlan), otherwise ignored.
 * @return     FPP_ERR_OK : Successfully iterated through all available static entries.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_l2_stent_print_all(FCI_CLIENT* p_cl, demo_l2_stent_cb_print_t p_cb_print,
                            bool by_vlan, uint16_t vlan)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_l2_static_ent_cmd_t cmd_to_fci = {0};
    fpp_l2_static_ent_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_L2_STATIC_ENT,
                    sizeof(fpp_l2_static_ent_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /*  query loop  */
    while (FPP_ERR_OK == rtn)
    {
```

```
        if ((false == by_vlan) ||
            ((true == by_vlan) && (ntohs(reply_from_fci.vlan) == vlan)))
        {
            rtn = p_cb_print(&reply_from_fci);
        }

        if (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_L2_STATIC_ENT,
                    sizeof(fpp_l2_static_ent_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more static entries to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_L2_STATIC_EN_NOT_FOUND == rtn)
    {
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_l2_stent_print_all() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all applicable static entries in PFE.
 * @param[in]   p_cl         FCI client instance
 * @param[out]  p_rtn_count  Space to store the count of static entries.
 * @param[in]   by_vlan     [optional parameter]
 *                           Request to count only those static entries
 *                           which are associated with a particular bridge domain.
 * @param[in]   vlan        [optional parameter]
 *                           VLAN ID of a bridge domain.
 *                           Applicable only if (true == by_vlan), otherwise ignored.
 * @return      FPP_ERR_OK : Successfully counted all applicable static entries.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No value copied.
 */
int demo_l2_stent_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                            bool by_vlan, uint16_t vlan)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_l2_static_ent_cmd_t cmd_to_fci = {0};
    fpp_l2_static_ent_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint16_t count = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_L2_STATIC_ENT,
                    sizeof(fpp_l2_static_ent_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        if ((false == by_vlan) ||
            ((true == by_vlan) && (ntohs(reply_from_fci.vlan) == vlan)))
        {
            count++;
        }

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_L2_STATIC_ENT,
                    sizeof(fpp_l2_static_ent_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* query loop runs till there are no more static entries to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_L2_STATIC_EN_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_l2_stent_get_count() failed!");
```

```
    return (rtn);
}


/* ========================================================================= */
```

15.17 demo_fp.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_fp.h"


/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from PFE ============== */


/*
 * @brief       Use FCI calls to get configuration data of a requested FP rule
 *              from PFE. Identify the rule by its name.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_rule   Space for data from PFE.
 * @param[out]  p_rtn_idx    Space for index of the requested FP rule.
 *                           This is a generic index of the given rule in a common pool of
 *                           FP rules within PFE. It has no ties to any particular FP table.
 *                           Can be NULL. If NULL, then no index is stored.
 * @param[in]   p_rule_name  Name of the requested FP rule.
 *                           Names of FP rules are user-defined.
 *                           See demo_fp_rule_add().
 * @return      FPP_ERR_OK : The requested FP rule was found.
 *                           A copy of its configuration data was stored into p_rtn_rule.
 *                           Its common pool index was stored into p_rtn_idx.
 *                           REMINDER: data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_fp_rule_get_by_name(FCI_CLIENT* p_cl, fpp_fp_rule_cmd_t* p_rtn_rule,
                             uint16_t* p_rtn_idx, const char* p_rule_name)
{
    assert(NULL != p_cl);
```

```
    assert(NULL != p_rtn_rule);
    assert(NULL != p_rule_name);
    /* 'p_rtn_index' is allowed to be NULL */

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_fp_rule_cmd_t cmd_to_fci = {0};
    fpp_fp_rule_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint16_t idx = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_FP_RULE,
                    sizeof(fpp_fp_rule_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop (with the search condition) */
    while ((FPP_ERR_OK == rtn) &&
           (0 != strcmp((char*)(reply_from_fci.r.rule_name), p_rule_name)))
    {
        idx++;

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_FP_RULE,
                        sizeof(fpp_fp_rule_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_rule = reply_from_fci;
        if (NULL != p_rtn_idx)
        {
            *p_rtn_idx = idx;
        }
    }

    print_if_error(rtn, "demo_fp_rule_get_by_name() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to add/del items in PFE =========== */


/*
 * @brief       Use FCI calls to create a new FP rule in PFE.
 * @param[in]   p_cl         FCI client
 * @param[in]   p_rule_name  Name of the new FP rule.
 *                           The name is user-defined.
 * @param[in]   p_rule_data  Configuration data of the new FP rule.
 *                           To create a new FP rule, a local data struct must be created,
 *                           configured and then passed to this function.
 *                           See [localdata_fprule] to learn more.
 * @return      FPP_ERR_OK : New FP rule was created.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_fp_rule_add(FCI_CLIENT* p_cl, const char* p_rule_name,
                     const fpp_fp_rule_cmd_t* p_rule_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_rule_name);
    assert(NULL != p_rule_data);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_fp_rule_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci = *p_rule_data;
    rtn = set_text((char*)(cmd_to_fci.r.rule_name), p_rule_name, IFNAMSIZ);

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_REGISTER;
        rtn = fci_write(p_cl, FPP_CMD_FP_RULE, sizeof(fpp_fp_rule_cmd_t),
                                               (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_fp_rule_add() failed!");

    return (rtn);
}
```

```
/*
 * @brief      Use FCI calls to destroy the target FP rule in PFE.
 * @param[in]  p_cl         FCI client
 * @param[in]  p_rule_name  Name of the FP rule to destroy.
 * @return     FPP_ERR_OK : The FP rule was destroyed.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_fp_rule_del(FCI_CLIENT* p_cl, const char* p_rule_name)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_fp_rule_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((char*)(cmd_to_fci.r.rule_name), p_rule_name, IFNAMSIZ);

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_DEREGISTER;
        rtn = fci_write(p_cl, FPP_CMD_FP_RULE, sizeof(fpp_fp_rule_cmd_t),
                                              (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_fp_rule_del() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to create a new FP table in PFE.
 * @param[in]   p_cl          FCI client
 * @param[in]   p_table_name  Name of the new FP table.
 *                            The name is user-defined.
 * @return      FPP_ERR_OK : New FP table was created.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_fp_table_add(FCI_CLIENT* p_cl, const char* p_table_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_table_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_fp_table_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((char*)(cmd_to_fci.table_info.t.table_name), p_table_name, IFNAMSIZ);

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_REGISTER;
        rtn = fci_write(p_cl, FPP_CMD_FP_TABLE, sizeof(fpp_fp_table_cmd_t),
                                               (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_fp_table_add() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to destroy the target FP table in PFE.
 * @param[in]  p_cl          FCI client
 * @param[in]  p_table_name  Name of the FP table to destroy.
 * @return     FPP_ERR_OK : The FP table was destroyed.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_fp_table_del(FCI_CLIENT* p_cl, const char* p_table_name)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_fp_table_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((char*)(cmd_to_fci.table_info.t.table_name), p_table_name, IFNAMSIZ);

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
```

```
        cmd_to_fci.action = FPP_ACTION_DEREGISTER;
        rtn = fci_write(p_cl, FPP_CMD_FP_TABLE, sizeof(fpp_fp_table_cmd_t),
                                                 (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_fp_table_del() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to insert a FP rule at a given position of a FP table in PFE.
 * @param[in]   p_cl          FCI client
 * @param[in]   p_table_name  Name of an existing FP table.
 * @param[in]   p_rule_name   Name of an existing FP rule.
 * @param[in]   position      Index where to insert the rule. Starts at 0.
 * @return      FPP_ERR_OK : The rule was successfully inserted into the table.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_fp_table_insert_rule(FCI_CLIENT* p_cl, const char* p_table_name,
                              const char* p_rule_name, uint16_t position)
{
    assert(NULL != p_cl);
    assert(NULL != p_table_name);
    assert(NULL != p_rule_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_fp_table_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((char*)(cmd_to_fci.table_info.t.table_name), p_table_name, IFNAMSIZ);
    if (FPP_ERR_OK == rtn)
    {
        rtn = set_text((char*)(cmd_to_fci.table_info.t.rule_name), p_rule_name, IFNAMSIZ);
    }
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.table_info.t.position = htons(position);
    }

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_USE_RULE;
        rtn = fci_write(p_cl, FPP_CMD_FP_TABLE, sizeof(fpp_fp_table_cmd_t),
                                                 (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_fp_table_insert_rule() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to remove a FP rule from a FP table in PFE.
 * @param[in]   p_cl          FCI client
 * @param[in]   p_table_name  Name of an existing FP table.
 * @param[in]   p_rule_name   Name of a FP rule which is present in the FP table.
 * @return      FPP_ERR_OK : The rule was successfully removed from the table.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_fp_table_remove_rule(FCI_CLIENT* p_cl, const char* p_table_name,
                              const char* p_rule_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_table_name);
    assert(NULL != p_rule_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_fp_table_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((char*)(cmd_to_fci.table_info.t.table_name), p_table_name, IFNAMSIZ);
    if (FPP_ERR_OK == rtn)
    {
        rtn = set_text((char*)(cmd_to_fci.table_info.t.rule_name), p_rule_name, IFNAMSIZ);
    }

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_UNUSE_RULE;
        rtn = fci_write(p_cl, FPP_CMD_FP_TABLE, sizeof(fpp_fp_table_cmd_t),
                                                 (unsigned short*)(&cmd_to_fci));
```

```
    }

    print_if_error(rtn, "demo_fp_table_remove_rule() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : modify local data (no FCI calls) ================ */
/*
 * @defgroup    localdata_fprule  [localdata_fprule]
 * @brief:      Functions marked as [localdata_fprule] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_rule (a struct with configuration data).
 *              For addition of FP rules, there are no "initial data" to be obtained from PFE.
 *              Simply declare a local data struct and configure it.
 *              Then, after all modifications are done and finished,
 *              call demo_fp_rule_add() to create a new FP rule in PFE.
 */


/*
 * @brief          Set a data "template" of a FP rule.
 * @details        [localdata_fprule]
 * @param[in,out]  p_rule  Local data to be modified.
 * @param[in]      data    Data "template" (a value)
 *                         This value will be compared with a selected value from
 *                         the inspected traffic.
 */
void demo_fp_rule_ld_set_data(fpp_fp_rule_cmd_t* p_rule, uint32_t data)
{
    assert(NULL != p_rule);
    p_rule->r.data = htonl(data);
}


/*
 * @brief          Set a bitmask of a FP rule.
 * @details        [localdata_fprule]
 * @param[in,out]  p_rule  Local data to be modified.
 * @param[in]      mask    Bitmask for more precise data selection.
 *                         This bitmask is applied on the selected 32bit value from
 *                         the inspected traffic.
 */
void demo_fp_rule_ld_set_mask(fpp_fp_rule_cmd_t* p_rule, uint32_t mask)
{
    assert(NULL != p_rule);
    p_rule->r.mask = htonl(mask);
}


/*
 * @brief          Set an offset and a base for the offset ("offset from") of a FP rule.
 * @details        [localdata_fprule]
 * @param[in,out]  p_rule  Local data to be modified.
 * @param[in]      offset  Offset (in bytes) into traffic's data.
 *                         The offset is applied from the respective base ("offset_from").
 *                         Data value (32bit) which lies on the offset is the value selected
 *                         for comparison under the given FP rule.
 * @param[in]      offset_from  Base for an offset calculation.
 *                              See description of the fpp_fp_offset_from_t type
 *                              in FCI API Reference.
 */
void demo_fp_rule_ld_set_offset(fpp_fp_rule_cmd_t* p_rule, uint16_t offset,
                                fpp_fp_offset_from_t offset_from)
{
    assert(NULL != p_rule);

    p_rule->r.offset = htons(offset);

    hton_enum(&offset_from, sizeof(fpp_fp_offset_from_t));
    p_rule->r.offset_from = offset_from;
}


/*
 * @brief          Set/unset an inverted mode of a FP rule match evaluation.
 * @details        [localdata_fprule]
 * @param[in,out]  p_rule  Local data to be modified.
 * @param[in]      invert  Request to set/unset the inverted mode of evaluation.
 */
void demo_fp_rule_ld_set_invert(fpp_fp_rule_cmd_t* p_rule, bool invert)
{
    assert(NULL != p_rule);
    p_rule->r.invert = invert;  /* NOTE: Implicit cast from bool to uint8_t */
}
```



```
/*
 * @brief          Set action to be done if inspected traffic satisfies a FP rule.
 * @details        [localdata_fprule]
 * @param[in,out]  p_rule  Local data to be modified.
 * @param[in]      match_action      Action to be done.
 *                                   See description of the fpp_fp_rule_match_action_t type
 *                                   in FCI API Reference.
 * @param[in]      p_next_rule_name  Name of a next FP rule to execute.
 *                                   Meaningful only if the match action is FP_NEXT_RULE.
 *                                   Can be NULL. If NULL or "" (empty string),
 *                                   then no rule is set as the next rule.
 */
void demo_fp_rule_ld_set_match_action(fpp_fp_rule_cmd_t* p_rule,
                                      fpp_fp_rule_match_action_t match_action,
                                      const char* p_next_rule_name)
{
    assert(NULL != p_rule);
    /* 'p_next_rule_name' is allowed to be NULL */

    hton_enum(&match_action, sizeof(fpp_fp_rule_match_action_t));
    p_rule->r.match_action = match_action;

    set_text((char*)(p_rule->r.next_rule_name), p_next_rule_name, IFNAMSIZ);
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */


/*
 * @brief      Query the status of an invert mode of a FP rule.
 * @details    [localdata_fprule]
 * @param[in]  p_rule  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the FP rule:
 *             true  : was running in the inverted mode
 *             false : was NOT running in the inverted mode
 */
bool demo_fp_rule_ld_is_invert(const fpp_fp_rule_cmd_t* p_rule)
{
    assert(NULL != p_rule);
    return (bool)(p_rule->r.invert);
}


/*
 * @brief      Query the name of a FP rule.
 * @details    [localdata_fprule]
 * @param[in]  p_rule  Local data to be queried.
 * @return     Name of the FP rule.
 */
const char* demo_fp_rule_ld_get_name(const fpp_fp_rule_cmd_t* p_rule)
{
    assert(NULL != p_rule);
    return (const char*)(p_rule->r.rule_name);
}


/*
 * @brief      Query the name of a "next FP rule".
 * @details    [localdata_fprule]
 *             "Next FP rule" is meaningful only when "match_action == FP_NEXT_RULE"
 * @param[in]  p_rule  Local data to be queried.
 * @return     Name of the "next FP rule".
 */
const char* demo_fp_rule_ld_get_next_name(const fpp_fp_rule_cmd_t* p_rule)
{
    assert(NULL != p_rule);
    return (const char*)(p_rule->r.next_rule_name);
}


/*
 * @brief      Query the data "template" of a FP rule.
 * @details    [localdata_fprule]
 * @param[in]  p_rule  Local data to be queried.
 * @return     Data "template" used by the FP rule.
 */
uint32_t demo_fp_rule_ld_get_data(const fpp_fp_rule_cmd_t* p_rule)
{
    assert(NULL != p_rule);
    return ntohl(p_rule->r.data);
}
```

```
/*
 * @brief      Query the bitmask of a FP rule.
 * @details    [localdata_fprule]
 * @param[in]  p_rule  Local data to be queried.
 * @return     Bitmask used by the FP rule.
 */
uint32_t demo_fp_rule_ld_get_mask(const fpp_fp_rule_cmd_t* p_rule)
{
    assert(NULL != p_rule);
    return ntohl(p_rule->r.mask);
}


/*
 * @brief      Query the offset of a FP rule.
 * @details    [localdata_fprule]
 * @param[in]  p_rule  Local data to be queried.
 * @return     Offset where to find the inspected value in the traffic data.
 */
uint16_t demo_fp_rule_ld_get_offset(const fpp_fp_rule_cmd_t* p_rule)
{
    assert(NULL != p_rule);
    return ntohs(p_rule->r.offset);
}


/*
 * @brief      Query the offset base ("offset from") of a FP rule.
 * @details    [localdata_fprule]
 * @param[in]  p_rule  Local data to be queried.
 * @return     Base position in traffic data to use for offset calculation.
 */
fpp_fp_offset_from_t demo_fp_rule_ld_get_offset_from(const fpp_fp_rule_cmd_t* p_rule)
{
    assert(NULL != p_rule);

    fpp_fp_offset_from_t tmp_offset_from = (p_rule->r.offset_from);
    ntoh_enum(&tmp_offset_from, sizeof(fpp_fp_offset_from_t));

    return (tmp_offset_from);
}


/*
 * @brief      Query the match action of a FP rule.
 * @details    [localdata_fprule]
 * @param[in]  p_rule  Local data to be queried.
 * @return     Match action of the FP rule.
 */
fpp_fp_rule_match_action_t demo_fp_rule_ld_get_match_action(const fpp_fp_rule_cmd_t* p_rule)
{
    assert(NULL != p_rule);

    fpp_fp_rule_match_action_t tmp_match_action = (p_rule->r.match_action);
    ntoh_enum(&tmp_match_action, sizeof(fpp_fp_rule_match_action_t));

    return (tmp_match_action);
}


/* ==== PUBLIC FUNCTIONS : misc ============================================ */


/*
 * @brief      Use FCI calls to iterate through all available FP rules of a given FP table
 *             in PFE. Execute a callback print function for each applicable FP rule.
 * @param[in]  p_cl           FCI client
 * @param[in]  p_cb_print     Callback print function.
 *                            --> If the callback returns ZERO, then all is OK and
 *                                a next FP rule in table is picked for a print process.
 *                            --> If the callback returns NON-ZERO, then some problem is
 *                                assumed and this function terminates prematurely.
 * @param[in]  p_table_name   Name of a FP table.
 *                            Names of FP tables are user-defined. See demo_fp_table_add().
 * @param[in]  position_init  Start invoking a callback print function from
 *                            this position in the FP table.
 *                            If 0, start from the very first FP rule in the table.
 * @param[in]  count          Print only this count of FP rules, then end.
 *                            If 0, keep printing FP rules till the end of the table.
 * @return     FPP_ERR_OK : Successfully iterated through all FP rules of the given FP table.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_fp_table_print(FCI_CLIENT* p_cl, demo_fp_rule_cb_print_t p_cb_print,
                        const char* p_table_name, uint16_t position_init, uint16_t count)
{
    assert(NULL != p_cl);
```

```
    assert(NULL != p_cb_print);
    assert(NULL != p_table_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_fp_table_cmd_t cmd_to_fci = {0};
    fpp_fp_table_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((char*)(cmd_to_fci.table_info.t.table_name), p_table_name, IFNAMSIZ);
    if (0u == count)  /* if 0, set max possible count of items */
    {
        count--;  /* WARNING: intentional use of owf behavior */
    }

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_FP_TABLE,
                        sizeof(fpp_fp_table_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

        /* query loop */
        uint16_t position = 0u;
        while ((FPP_ERR_OK == rtn) && (0u != count))
        {
            if (position >= position_init)
            {
                const fpp_fp_rule_cmd_t tmp_rule = {0u, (reply_from_fci.table_info.r)};
                rtn = p_cb_print(&tmp_rule, position);
                count--;
            }

            position++;

            if (FPP_ERR_OK == rtn)
            {
                cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
                rtn = fci_query(p_cl, FPP_CMD_FP_TABLE,
                                sizeof(fpp_fp_table_cmd_t), (unsigned short*)(&cmd_to_fci),
                                &reply_length, (unsigned short*)(&reply_from_fci));
            }
        }

        /* query loop runs till there are no more FP rules to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_FP_RULE_NOT_FOUND == rtn)
        {
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_fp_table_print() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to iterate through all available FP rules in PFE (regardless
 *             of table affiliation). Execute a print function for each applicable FP rule.
 * @param[in]  p_cl           FCI client
 * @param[in]  p_cb_print     Callback print function.
 *                            --> If the callback returns ZERO, then all is OK and
 *                                a next FP rule is picked for a print process.
 *                            --> If the callback returns NON-ZERO, then some problem is
 *                                assumed and this function terminates prematurely.
 * @param[in]  idx_init       Start invoking a callback print function from
 *                            this index of FP rule query.
 *                            If 0, start from the very first queried FP rule.
 * @param[in]  count          Print only this count of FP rules, then end.
 *                            If 0, keep printing FP rules till there is no more available.
 * @return     FPP_ERR_OK : Successfully iterated through all available FP rules.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_fp_rule_print_all(FCI_CLIENT* p_cl, demo_fp_rule_cb_print_t p_cb_print,
                           uint16_t idx_init, uint16_t count)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
```

```
    fpp_fp_rule_cmd_t cmd_to_fci = {0};
    fpp_fp_rule_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    if (0u == count)  /* if 0, set max possible count of items */
    {
        count--;  /* WARNING: intentional use of owf behavior */
    }

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_FP_RULE,
                    sizeof(fpp_fp_rule_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    uint16_t idx = 0u;
    while ((FPP_ERR_OK == rtn) && (0u != count))
    {
        if (idx >= idx_init)
        {
            rtn = p_cb_print(&reply_from_fci, idx);
            count--;
        }

        idx++;

        if (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_FP_RULE,
                            sizeof(fpp_fp_rule_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more FP rules to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_FP_RULE_NOT_FOUND == rtn)
    {
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_fp_rule_print_all() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available FP rules in PFE (regardless
 *              of table affiliation).
 * @param[in]   p_cl         FCI client instance
 * @param[out]  p_rtn_count  Space to store the count of FP rules.
 * @return      FPP_ERR_OK : Successfully counted all available FP rules.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_fp_rule_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_fp_rule_cmd_t cmd_to_fci = {0};
    fpp_fp_rule_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint32_t count = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_FP_RULE,
                    sizeof(fpp_fp_rule_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        count++;

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_FP_RULE,
                        sizeof(fpp_fp_rule_cmd_t), (unsigned short*)(&cmd_to_fci),
```

```
                         &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* query loop runs till there are no more FP rules to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_FP_RULE_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_fp_rule_get_count() failed!");

    return (rtn);
}


/* ========================================================================= */
```

15.18 demo_rt_ct.c

```
/* =========================================================================
 *  Copyright 2020-2022 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_rt_ct.h"


/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from PFE ============== */


/*
 * @brief       Use FCI calls to get configuration data of a requested route from PFE.
 *              Identify the route by its ID.
 * @param[in]   p_cl      FCI client
 * @param[out]  p_rtn_rt  Space for data from PFE.
 * @param[in]   id        ID of the requested route.
 *                        Route IDs are user-defined. See demo_rt_add().
 * @return      FPP_ERR_OK : The requested route was found.
 *                           A copy of its configuration data was stored into p_rtn_rt.
 *                           REMINDER: Data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
```

```
 *                            No data copied.
 */
int demo_rt_get_by_id(FCI_CLIENT* p_cl, fpp_rt_cmd_t* p_rtn_rt, uint32_t id)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_rt);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_rt_cmd_t cmd_to_fci = {0};
    fpp_rt_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_IP_ROUTE,
                    sizeof(fpp_rt_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop (with a search condition) */
    while ((FPP_ERR_OK == rtn) && (ntohl(reply_from_fci.id) != id))
    {
        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_IP_ROUTE,
                        sizeof(fpp_rt_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_rt = reply_from_fci;
    }

    print_if_error(rtn, "demo_rt_get_by_id() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get configuration data of a requested IPv4 conntrack
 *              from PFE. Identify the conntrack by a specific tuple of parameters.
 * @param[in]   p_cl       FCI client
 * @param[out]  p_rtn_ct   Space for data from PFE.
 * @param[in]   p_ct_data  Configuration data for IPv4 conntrack identification.
 *                         To identify a conntrack, all the following data must be
 *                         correctly set:
 *                           --> protocol
 *                           --> saddr
 *                           --> daddr
 *                           --> sport
 *                           --> dport
 *                         REMINDER: It is assumed that data are in a network byte order.
 * @return      FPP_ERR_OK : The requested IPv4 conntrack was found.
 *                           A copy of its configuration was stored into p_rtn_ct.
 *                           REMINDER: Data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_ct_get_by_tuple(FCI_CLIENT* p_cl, fpp_ct_cmd_t* p_rtn_ct,
                                     const fpp_ct_cmd_t* p_ct_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_ct);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_ct_cmd_t cmd_to_fci = {0};
    fpp_ct_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_IPV4_CONNTRACK,
                    sizeof(fpp_ct_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop (with a search condition) */
    while ((FPP_ERR_OK == rtn) &&
           !(
             /* both sides are in network byte order (thus no byte order conversion needed) */
             ((reply_from_fci.protocol) == (p_ct_data->protocol)) &&
             ((reply_from_fci.sport) == (p_ct_data->sport)) &&
             ((reply_from_fci.dport) == (p_ct_data->dport)) &&
             ((reply_from_fci.saddr) == (p_ct_data->saddr)) &&
```

```
              ((reply_from_fci.daddr) == (p_ct_data->daddr))
             )
           )
    {
        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_IPV4_CONNTRACK,
                        sizeof(fpp_ct_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_ct = reply_from_fci;
    }

    print_if_error(rtn, "demo_ct_get_by_tuple() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get configuration data of a requested IPv6 conntrack
 *              from PFE. Identify the conntrack by a specific tuple of parameters.
 * @param[in]   p_cl        FCI client
 * @param[out]  p_rtn_ct6   Space for data from PFE.
 * @param[in]   p_ct6_data  Configuration data for IPv6 conntrack identification.
 *                          To identify a conntrack, all the following data must be
 *                          correctly set:
 *                            --> protocol
 *                            --> saddr
 *                            --> daddr
 *                            --> sport
 *                            --> dport
 *                          REMINDER: It is assumed that data are in a network byte order.
 * @return      FPP_ERR_OK : The requested IPv6 conntrack was found.
 *                           A copy of its configuration was stored into p_rtn_ct6.
 *                           REMINDER: Data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_ct6_get_by_tuple(FCI_CLIENT* p_cl, fpp_ct6_cmd_t* p_rtn_ct6,
                                      const fpp_ct6_cmd_t* p_ct6_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_ct6);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_ct6_cmd_t cmd_to_fci = {0};
    fpp_ct6_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_IPV6_CONNTRACK,
                    sizeof(fpp_ct6_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop (with a search condition) */
    while ((FPP_ERR_OK == rtn) &&
           !(
              /* both sides are in network byte order (thus no byte order conversion needed) */
              ((reply_from_fci.protocol) == (p_ct6_data->protocol)) &&
              ((reply_from_fci.sport) == (p_ct6_data->sport)) &&
              ((reply_from_fci.dport) == (p_ct6_data->dport)) &&
              (0 == memcmp(reply_from_fci.saddr, p_ct6_data->saddr, (4 * sizeof(uint32_t)))) &&
              (0 == memcmp(reply_from_fci.daddr, p_ct6_data->daddr, (4 * sizeof(uint32_t))))
             )
           )
    {
        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_IPV6_CONNTRACK,
                        sizeof(fpp_ct6_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_ct6 = reply_from_fci;
    }

    print_if_error(rtn, "demo_ct6_get_by_tuple() failed!");
```

```
    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to update data in PFE ============= */


/*
 * @brief       Use FCI calls to update configuration of a target IPv4 conntrack in PFE.
 * @details     For conntracks, only a few selected parameters can be modified.
 *              See FCI API Reference, chapter FPP_CMD_IPV4_CONNTRACK,
 *              subsection "Action FPP_ACTION_UPDATE".
 * @param[in]   p_cl       FCI client
 * @param[in]   p_ct_data  Local data struct which represents a new configuration of
 *                         the target IPv4 conntrack.
 *                         Initial data can be obtained via demo_ct_get_by_tuple().
 * @return      FPP_ERR_OK : Configuration of the target IPv4 conntrack was
 *                           successfully updated in PFE.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_ct_update(FCI_CLIENT* p_cl, const fpp_ct_cmd_t* p_ct_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_ct_data);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_ct_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci = *p_ct_data;

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_IPV4_CONNTRACK, sizeof(fpp_ct_cmd_t),
                                                 (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_ct_update() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to update configuration of a target IPv6 conntrack in PFE.
 * @details     For conntracks, only a few selected parameters can be modified.
 *              See FCI API Reference, chapter FPP_CMD_IPV6_CONNTRACK,
 *              subsection "Action FPP_ACTION_UPDATE".
 * @param[in]   p_cl        FCI client
 * @param[in]   p_ct6_data  Local data struct which represents a new configuration of
 *                          the target IPv6 conntrack.
 *                          Initial data can be obtained via demo_ct6_get_by_tuple().
 * @return      FPP_ERR_OK : Configuration of the target IPv6 conntrack was
 *                           successfully updated in PFE.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_ct6_update(FCI_CLIENT* p_cl, const fpp_ct6_cmd_t* p_ct6_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_ct6_data);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_ct6_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci = *p_ct6_data;

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_IPV6_CONNTRACK, sizeof(fpp_ct6_cmd_t),
                                                 (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_ct6_update() failed!");

    return (rtn);
}




/*
 * @brief       Use FCI calls to set timeout for IPv4 TCP conntracks in PFE.
 * @param[in]   p_cl     FCI client
 * @param[in]   timeout  Timeout [seconds]
 * @return      FPP_ERR_OK : New timeout was set.
 *              other      : Some error occurred (represented by the respective error code).
 */
```

```
int demo_ct_timeout_tcp(FCI_CLIENT* p_cl, uint32_t timeout)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_timeout_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci.protocol = htons(6u);  /* 6 == tcp */
    cmd_to_fci.timeout_value1 = htonl(timeout);

    /* send data */
    rtn = fci_write(p_cl, FPP_CMD_IPV4_SET_TIMEOUT, sizeof(fpp_timeout_cmd_t),
                                                   (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_ct_timeout_tcp() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to set timeout for IPv4 UDP conntracks in PFE.
 * @param[in]   p_cl      FCI client
 * @param[in]   timeout   Timeout [seconds]
 * @return      FPP_ERR_OK : New timeout was set.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_ct_timeout_udp(FCI_CLIENT* p_cl, uint32_t timeout)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_timeout_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci.protocol = htons(17u);  /* 17 == udp */
    cmd_to_fci.timeout_value1 = htonl(timeout);

    /* send data */
    rtn = fci_write(p_cl, FPP_CMD_IPV4_SET_TIMEOUT, sizeof(fpp_timeout_cmd_t),
                                                   (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_ct_timeout_udp() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to set timeout for all other IPv4 conntracks than TCP/UDP ones.
 * @param[in]   p_cl     FCI client
 * @param[in]   timeout  Timeout [seconds]
 * @return      FPP_ERR_OK : New timeout was set.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_ct_timeout_others(FCI_CLIENT* p_cl, uint32_t timeout)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_timeout_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci.protocol = htons(0u);  /* 0 == others */
    cmd_to_fci.timeout_value1 = htonl(timeout);

    /* send data */
    rtn = fci_write(p_cl, FPP_CMD_IPV4_SET_TIMEOUT, sizeof(fpp_timeout_cmd_t),
                                                   (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_ct_timeout_others() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to add/del items in PFE =========== */


/*
 * @brief       Use FCI calls to create a new route in PFE.
 * @details     In the context of PFE, a "route" is a configuration data element that
 *              specifies which physical interface of PFE shall be used as an egress interface
 *              and what destination MAC address shall be set in the routed traffic.
 *              These "routes" are used as a part of IPv4/IPv6 conntracks.
```



```
 * @param[in]   p_cl       FCI client
 * @param[in]   id         ID of the new route.
 * @param[in]   p_rt_data  Configuration data of the new route.
 *                         To create a new route, a local data struct must be created,
 *                         configured and then passed to this function.
 *                         See [localdata_rt] to learn more.
 * @return      FPP_ERR_OK : New route was created.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_rt_add(FCI_CLIENT* p_cl, uint32_t id, const fpp_rt_cmd_t* p_rt_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_rt_data);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_rt_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci = *p_rt_data;
    cmd_to_fci.id = htonl(id);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_REGISTER;
    rtn = fci_write(p_cl, FPP_CMD_IP_ROUTE, sizeof(fpp_rt_cmd_t),
                                       (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_rt_add() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to destroy the target route in PFE.
 * @param[in]   p_cl  FCI client
 * @param[in]   id    ID of the route to destroy.
 * @return      FPP_ERR_OK : The route was destroyed.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_rt_del(FCI_CLIENT* p_cl, uint32_t id)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_rt_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci.id = htonl(id);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_DEREGISTER;
    rtn = fci_write(p_cl, FPP_CMD_IP_ROUTE, sizeof(fpp_rt_cmd_t),
                                       (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_rt_del() failed!");

    return (rtn);
}




/*
 * @brief       Use FCI calls to create a new IPv4 conntrack in PFE.
 * @param[in]   p_cl       FCI client
 * @param[in]   p_ct_data  Configuration data of the new IPv4 conntrack.
 *                         To create a new IPv4 conntrack, a local data struct must
 *                         be created, configured and then passed to this function.
 *                         See [localdata_ct] to learn more.
 * @return      FPP_ERR_OK : New IPv4 conntrack was created.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_ct_add(FCI_CLIENT* p_cl, const fpp_ct_cmd_t* p_ct_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_ct_data);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_ct_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci = *p_ct_data;

    /* send data */
    cmd_to_fci.action = FPP_ACTION_REGISTER;
    rtn = fci_write(p_cl, FPP_CMD_IPV4_CONNTRACK, sizeof(fpp_ct_cmd_t),
```

```
                                                  (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_ct_add() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to destroy the target IPv4 conntrack in PFE.
 * @param[in]   p_cl       FCI client
 * @param[in]   p_ct_data  Configuration data for IPv4 conntrack identification.
 *                         To identify a conntrack, all the following data must be
 *                         correctly set:
 *                           --> protocol
 *                           --> saddr
 *                           --> daddr
 *                           --> sport
 *                           --> dport
 *                         REMINDER: It is assumed that data are in a network byte order.
 * @return      FPP_ERR_OK : The IPv4 conntrack was destroyed.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_ct_del(FCI_CLIENT* p_cl, const fpp_ct_cmd_t* p_ct_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_ct_data);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_ct_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci = *p_ct_data;

    /* send data */
    cmd_to_fci.action = FPP_ACTION_DEREGISTER;
    rtn = fci_write(p_cl, FPP_CMD_IPV4_CONNTRACK, sizeof(fpp_ct_cmd_t),
                                                  (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_ct_del() failed!");

    return (rtn);
}




/*
 * @brief       Use FCI calls to create a new IPv6 conntrack in PFE.
 * @param[in]   p_cl        FCI client
 * @param[in]   p_ct6_data  Configuration data of the new IPv6 conntrack.
 *                          To create a new IPv6 conntrack, a local data struct must
 *                          be created, configured and then passed to this function.
 *                          See [localdata_ct6] to learn more.
 * @return      FPP_ERR_OK : New IPv6 conntrack was created.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_ct6_add(FCI_CLIENT* p_cl, const fpp_ct6_cmd_t* p_ct6_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_ct6_data);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_ct6_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci = *p_ct6_data;

    /* send data */
    cmd_to_fci.action = FPP_ACTION_REGISTER;
    rtn = fci_write(p_cl, FPP_CMD_IPV6_CONNTRACK, sizeof(fpp_ct6_cmd_t),
                                                  (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_ct6_add() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to destroy the target IPv6 conntrack in PFE.
 * @param[in]   p_cl        FCI client
 * @param[in]   p_ct6_data  Configuration data for IPv6 conntrack identification.
 *                          To identify a conntrack, all the following data must be
 *                          correctly set:
 *                            --> protocol
```

```
 *                              --> saddr
 *                              --> daddr
 *                              --> sport
 *                              --> dport
 *                            REMINDER: It is assumed that data are in a network byte order.
 * @return      FPP_ERR_OK : The IPv6 conntrack was destroyed.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_ct6_del(FCI_CLIENT* p_cl, const fpp_ct6_cmd_t* p_ct6_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_ct6_data);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_ct6_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci = *p_ct6_data;

    /* send data */
    cmd_to_fci.action = FPP_ACTION_DEREGISTER;
    rtn = fci_write(p_cl, FPP_CMD_IPV6_CONNTRACK, sizeof(fpp_ct6_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));

    print_if_error(rtn, "demo_ct6_del() failed!");

    return (rtn);
}




/*
 * @brief       Use FCI calls to reset (clear) all IPv4 routes & conntracks in PFE.
 * @param[in]   p_cl  FCI client
 * @return      FPP_ERR_OK : All IPv4 routes & conntracks were cleared.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_rtct_reset_ip4(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    /* prepare data */
    /* empty */

    /* send data */
    rtn = fci_write(p_cl, FPP_CMD_IPV4_RESET, 0, NULL);

    print_if_error(rtn, "demo_rtct_reset_ip4() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to reset (clear) all IPv6 routes & conntracks in PFE.
 * @param[in]   p_cl  FCI clientf
 * @return      FPP_ERR_OK : All IPv6 routes & conntracks were cleared.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_rtct_reset_ip6(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    /* prepare data */
    /* empty */

    /* send data */
    rtn = fci_write(p_cl, FPP_CMD_IPV6_RESET, 0, NULL);

    print_if_error(rtn, "demo_rtct_reset_ip6() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : modify local data (no FCI calls) ================ */
/*
 * @defgroup    localdata_rt  [localdata_rt]
 * @brief:      Functions marked as [localdata_rt] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_rt (a struct with configuration data).
```

```
 *              When adding a new route, there are no "initial data" to be obtained from PFE.
 *              Simply declare a local data struct and configure it.
 *              Then, after all modifications are done and finished,
 *              call demo_rt_add() to create a new route in PFE.
 *              REMINDER: In the context of PFE, a "route" is a configuration data element
 *                        which is used as a part of IPv4/IPv6 conntracks.
 */


/*
 * @brief          Set a route as an IPv4 route. If the route was previously set as
 *                 an IPv6 route, then the IPv6 flag is removed.
 * @details        [localdata_rt]
 *                 Symbol names are a bit confusing (inherited from another project).
 *                 FPP_IP_ROUTE_6o4 == route is an IPv4 route
 *                 FPP_IP_ROUTE_4o6 == route is an IPv6 route
 *                 It is forbidden to set both flags at the same time (undefined behavior).
 * @param[in,out]  p_rt  Local data to be modified.
 */
void demo_rt_ld_set_as_ip4(fpp_rt_cmd_t* p_rt)
{
    assert(NULL != p_rt);
    p_rt->flags &= htonl(~FPP_IP_ROUTE_4o6);
    p_rt->flags |= htonl(FPP_IP_ROUTE_6o4);
}


/*
 * @brief          Set a route as an IPv6 route. If the route was previously set as
 *                 an IPv4 route, then the IPv4 flag is removed.
 * @details        [localdata_rt]
 * @param[in,out]  p_rt  Local data to be modified.
 *                 Symbol names are a bit confusing (inherited from another project).
 *                 FPP_IP_ROUTE_6o4 == route is an IPv4 route
 *                 FPP_IP_ROUTE_4o6 == route is an IPv6 route
 *                 It is forbidden to set both flags at the same time (undefined behavior).
 * @param[in,out]  p_rt  Local data to be modified.
 */
void demo_rt_ld_set_as_ip6(fpp_rt_cmd_t* p_rt)
{
    assert(NULL != p_rt);
    p_rt->flags &= htonl(~FPP_IP_ROUTE_6o4);
    p_rt->flags |= htonl(FPP_IP_ROUTE_4o6);
}


/*
 * @brief          Set a source MAC address of a route.
 * @details        [localdata_rt]
 * @param[in,out]  p_rt  Local data to be modified.
 *                 p_src_mac  Source MAC address.
 */
void demo_rt_ld_set_src_mac(fpp_rt_cmd_t* p_rt, const uint8_t p_src_mac[6])
{
    assert(NULL != p_rt);
    assert(NULL != p_src_mac);
    memcpy((p_rt->src_mac), p_src_mac, (6 * sizeof(uint8_t)));
}


/*
 * @brief          Set a destination MAC address of a route.
 * @details        [localdata_rt]
 * @param[in,out]  p_rt       Local data to be modified.
 *                 p_dst_mac  Destination MAC address.
 */
void demo_rt_ld_set_dst_mac(fpp_rt_cmd_t* p_rt, const uint8_t p_dst_mac[6])
{
    assert(NULL != p_rt);
    assert(NULL != p_dst_mac);
    memcpy((p_rt->dst_mac), p_dst_mac, (6 * sizeof(uint8_t)));
}


/*
 * @brief          Set an egress physical interface of a route.
 * @details        [localdata_rt]
 * @param[in,out]  p_rt          Local data to be modified.
 * @param[in]      p_phyif_name  Name of a physical interface which shall be used as egress.
 *                               Names of physical interfaces are hardcoded.
 *                               See the FCI API Reference, chapter Interface Management.
 */
void demo_rt_ld_set_egress_phyif(fpp_rt_cmd_t* p_rt, const char* p_phyif_name)
{
    assert(NULL != p_rt);
    assert(NULL != p_phyif_name);
```

```
    set_text((p_rt->output_device), p_phyif_name, IFNAMSIZ);
}




/*
 * @defgroup    localdata_ct  [localdata_ct]
 * @brief:      Functions marked as [localdata_ct] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_ct (a struct with configuration data).
 *              When adding a new IPv4 conntrack, there are no "initial data" to be obtained
 *              from PFE. Simply declare a local data struct and configure it.
 *              Then, after all modifications are done and finished,
 *              call demo_ct_add() to create a new IPv4 conntrack in PFE.
 */


/*
 * @brief          Set a protocol type of an IPv4 conntrack.
 * @details        [localdata_ct]
 * @param[in,out]  p_ct      Local data to be modified.
 * @param[in]      protocol  IP protocol ID
 *                           See "IANA Assigned Internet Protocol Number":
 *                https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
 */
void demo_ct_ld_set_protocol(fpp_ct_cmd_t* p_ct, uint16_t protocol)
{
    assert(NULL != p_ct);
    p_ct->protocol = htons(protocol);
}


/*
 * @brief          Set a ttl decrement flag of an IPv4 conntrack.
 * @details        [localdata_ct]
 *                 If set, then the TTL value of a packet is decremented when
 *                 the packet is routed by the IPv4 conntrack.
 * @param[in,out]  p_ct  Local data to be modified.
 * @param[in]      set   Request to enable/disable the ttl decrement.
 */
void demo_ct_ld_set_ttl_decr(fpp_ct_cmd_t* p_ct, bool set)
{
    assert(NULL != p_ct);

    if (set)
    {
        p_ct->flags |= htons(CTCMD_FLAGS_TTL_DECREMENT);
    }
    else
    {
        p_ct->flags &= htons((uint16_t)(~CTCMD_FLAGS_TTL_DECREMENT));
    }
}


/*
 * @brief          Set "orig direction" data of an IPv4 conntrack.
 * @details        [localdata_ct]
 * @param[in,out]  p_ct      Local data to be modified.
 * @param[in]      saddr     IPv4 source address.
 * @param[in]      daddr     IPv4 destination address.
 * @param[in]      sport     Source port.
 * @param[in]      dport     Destination port.
 * @param[in]      vlan      VLAN tag.
 *                             ZERO     : No VLAN tag modifications in this direction.
 *                             non ZERO : --> If a packet is not tagged,
 *                                                then a VLAN tag is added.
 *                                        --> If a packet is already tagged,
 *                                                then the VLAN tag is replaced.
 * @param[in]      route_id  ID of a route for the orig direction.
 *                           The route must already exist in PFE.
 *                           See demo_rt_add().
 * @param[in]      unidir_orig_only  Request to make the conntrack unidirectional
 *                                   (orig direction only).
 *                                   If true and the conntrack was previously
 *                                   configured as "reply direction only",
 *                                   it gets newly reconfigured as "orig direction only".
 */
void demo_ct_ld_set_orig_dir(fpp_ct_cmd_t* p_ct, uint32_t saddr, uint32_t daddr,
                             uint16_t sport, uint16_t dport, uint16_t vlan,
                             uint32_t route_id, bool unidir_orig_only)
{
    assert(NULL != p_ct);

    p_ct->saddr = htonl(saddr);
```



```
    p_ct->daddr = htonl(daddr);
    p_ct->sport = htons(sport);
    p_ct->dport = htons(dport);
    p_ct->vlan  = htons(vlan);
    p_ct->route_id = htonl(route_id);

    if (unidir_orig_only)
    {
        p_ct->route_id_reply = htonl(0u);
        p_ct->flags |= htons(CTCMD_FLAGS_REP_DISABLED);
        p_ct->flags &= htons((uint16_t)(~CTCMD_FLAGS_ORIG_DISABLED));
    }
}


/*
 * @brief          Set "reply direction" data of an IPv4 conntrack.
 * @details        [localdata_ct]
 * @param[in,out]  p_ct               Local data to be modified.
 * @param[in]      saddr_reply        IPv4 source address (reply direction).
 * @param[in]      daddr_reply        IPv4 destination address (reply direction).
 * @param[in]      sport_reply        Source port (reply direction).
 * @param[in]      dport_reply        Destination port (reply direction).
 * @param[in]      vlan_reply         VLAN tag (reply direction).
 *                                      ZERO     : No VLAN tag modifications in this direction
 *                                      non ZERO : --> If a packet is not tagged,
 *                                                     then a VLAN tag is added.
 *                                                 --> If a packet is already tagged,
 *                                                     then the VLAN tag is replaced.
 * @param[in]      route_id_reply     ID of a route for the reply direction.
 *                                    The route must already exist in PFE.
 *                                    See demo_rt_add().
 * @param[in]      unidir_reply_only  Request to make the conntrack unidirectional
 *                                    (reply direction only).
 *                                    If true and the conntrack was previously
 *                                    configured as "orig direction only",
 *                                    it gets newly reconfigured as "reply direction only".
 */
void demo_ct_ld_set_reply_dir(fpp_ct_cmd_t* p_ct, uint32_t saddr_reply, uint32_t daddr_reply,
                              uint16_t sport_reply, uint16_t dport_reply, uint16_t vlan_reply,
                              uint32_t route_id_reply, bool unidir_reply_only)
{
    assert(NULL != p_ct);

    p_ct->saddr_reply = htonl(saddr_reply);
    p_ct->daddr_reply = htonl(daddr_reply);
    p_ct->sport_reply = htons(sport_reply);
    p_ct->dport_reply = htons(dport_reply);
    p_ct->vlan_reply  = htons(vlan_reply);
    p_ct->route_id_reply = htonl(route_id_reply);

    if (unidir_reply_only)
    {
        p_ct->route_id = htonl(0u);
        p_ct->flags |= htons(CTCMD_FLAGS_ORIG_DISABLED);
        p_ct->flags &= htons((uint16_t)(~CTCMD_FLAGS_REP_DISABLED));
    }
}




/*
 * @defgroup    localdata_ct6  [localdata_ct6]
 * @brief:      Functions marked as [localdata_ct6] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_ct6 (a struct with configuration data).
 *              When adding a new IPv6 conntrack, there are no "initial data" to be obtained
 *              from PFE. Simply declare a local data struct and configure it.
 *              Then, after all modifications are done and finished,
 *              call demo_ct6_add() to create a new IPv6 conntrack in PFE.
 */


/*
 * @brief          Set a protocol type of an IPv6 conntrack.
 * @details        [localdata_ct6]
 * @param[in,out]  p_ct6     Local data to be modified.
 * @param[in]      protocol  IP protocol ID
 *                           See "IANA Assigned Internet Protocol Number":
 *               https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
 */
void demo_ct6_ld_set_protocol(fpp_ct6_cmd_t* p_ct6, uint16_t protocol)
{
    assert(NULL != p_ct6);
    p_ct6->protocol = htons(protocol);
```

```
}


/*
 * @brief          Set a ttl decrement flag of an IPv6 conntrack.
 * @details        [localdata_ct6]
 *                 If set, then the TTL value of a packet is decremented when
 *                 the packet is routed by the IPv6 conntrack.
 * @param[in,out]  p_ct6  Local data to be modified.
 * @param[in]      set    Request to enable/disable the ttl decrement.
 */
void demo_ct6_ld_set_ttl_decr(fpp_ct6_cmd_t* p_ct6, bool set)
{
    assert(NULL != p_ct6);

    if (set)
    {
        p_ct6->flags |= htons(CTCMD_FLAGS_TTL_DECREMENT);
    }
    else
    {
        p_ct6->flags &= htons((uint16_t)(~CTCMD_FLAGS_TTL_DECREMENT));
    }
}


/*
 * @brief          Set "orig direction" data of an IPv6 conntrack.
 * @details        [localdata_ct6]
 * @param[in,out]  p_ct6     Local data to be modified.
 * @param[in]      p_saddr   IPv6 source address.
 * @param[in]      p_daddr   IPv6 destination address.
 * @param[in]      sport     Source port.
 * @param[in]      dport     Destination port.
 * @param[in]      vlan      VLAN tag
 *                             ZERO     : No VLAN tag modifications in this direction.
 *                             non ZERO : --> If a packet is not tagged,
 *                                            then a VLAN tag is added.
 *                                        --> If a packet is already tagged,
 *                                            then the VLAN tag is replaced.
 * @param[in]      route_id  ID of a route for the orig direction.
 *                           The route must already exist in PFE.
 *                           See demo_rt_add().
 * @param[in]      unidir_orig_only  Request to make the conntrack unidirectional
 *                                   (orig direction only).
 *                                   If true and the conntrack was previously
 *                                   configured as "reply direction only",
 *                                   it gets newly reconfigured as "orig direction only".
 */
void demo_ct6_ld_set_orig_dir(fpp_ct6_cmd_t* p_ct6, const uint32_t p_saddr[4],
                              const uint32_t p_daddr[4],
                              uint16_t sport, uint16_t dport, uint16_t vlan,
                              uint32_t route_id, bool unidir_orig_only)
{
    assert(NULL != p_ct6);
    assert(NULL != p_saddr);
    assert(NULL != p_daddr);

    p_ct6->saddr[0] = htonl(p_saddr[0]);
    p_ct6->saddr[1] = htonl(p_saddr[1]);
    p_ct6->saddr[2] = htonl(p_saddr[2]);
    p_ct6->saddr[3] = htonl(p_saddr[3]);

    p_ct6->daddr[0] = htonl(p_daddr[0]);
    p_ct6->daddr[1] = htonl(p_daddr[1]);
    p_ct6->daddr[2] = htonl(p_daddr[2]);
    p_ct6->daddr[3] = htonl(p_daddr[3]);

    p_ct6->sport = htons(sport);
    p_ct6->dport = htons(dport);
    p_ct6->vlan  = htons(vlan);
    p_ct6->route_id = htonl(route_id);

    if (unidir_orig_only)
    {
        p_ct6->route_id_reply = htonl(0u);
        p_ct6->flags |= htons(CTCMD_FLAGS_REP_DISABLED);
        p_ct6->flags &= htons((uint16_t)(~CTCMD_FLAGS_ORIG_DISABLED));
    }
}


/*
 * @brief          Set "reply direction" data of an IPv6 conntrack.
 * @details        [localdata_ct6]
 * @param[in,out]  p_ct6              Local data to be modified.
```

```
 * @param[in]      p_saddr_reply      IPv6 source address (reply direction).
 * @param[in]      p_daddr_reply      IPv6 destination address (reply direction).
 * @param[in]      sport_reply        Source port (reply direction).
 * @param[in]      dport_reply        Destination port (reply direction).
 * @param[in]      vlan_reply         VLAN tag (reply direction).
 *                                      ZERO     : No VLAN tag modifications in this direction
 *                                      non ZERO : --> If a packet is not tagged,
 *                                                     then a VLAN tag is added.
 *                                                 --> If a packet is already tagged,
 *                                                     then the VLAN tag is replaced.
 * @param[in]      route_id_reply     ID of a route for the reply direction.
 * @param[in]      unidir_reply_only  Request to make the conntrack unidirectional
 *                                    (reply direction only).
 *                                    If true and the conntrack was previously
 *                                    configured as "orig direction only",
 *                                    it gets newly reconfigured as "reply direction only".
 */
void demo_ct6_ld_set_reply_dir(fpp_ct6_cmd_t* p_ct6, const uint32_t p_saddr_reply[4],
                               const uint32_t p_daddr_reply[4],
                               uint16_t sport_reply, uint16_t dport_reply,uint16_t vlan_reply,
                               uint32_t route_id_reply, bool unidir_reply_only)
{
    assert(NULL != p_ct6);
    assert(NULL != p_saddr_reply);
    assert(NULL != p_daddr_reply);

    p_ct6->saddr_reply[0] = htonl(p_saddr_reply[0]);
    p_ct6->saddr_reply[1] = htonl(p_saddr_reply[1]);
    p_ct6->saddr_reply[2] = htonl(p_saddr_reply[2]);
    p_ct6->saddr_reply[3] = htonl(p_saddr_reply[3]);

    p_ct6->daddr_reply[0] = htonl(p_daddr_reply[0]);
    p_ct6->daddr_reply[1] = htonl(p_daddr_reply[1]);
    p_ct6->daddr_reply[2] = htonl(p_daddr_reply[2]);
    p_ct6->daddr_reply[3] = htonl(p_daddr_reply[3]);

    p_ct6->sport_reply = htons(sport_reply);
    p_ct6->dport_reply = htons(dport_reply);
    p_ct6->vlan_reply  = htons(vlan_reply);
    p_ct6->route_id_reply = htonl(route_id_reply);

    if (unidir_reply_only)
    {
        p_ct6->route_id = htonl(0u);
        p_ct6->flags |= htons(CTCMD_FLAGS_ORIG_DISABLED);
        p_ct6->flags &= htons((uint16_t)(~CTCMD_FLAGS_REP_DISABLED));
    }
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */


/*
 * @brief      Query whether a route is an IPv4 route.
 * @details    [localdata_rt]
 * @param[in]  p_rt  Local data to be queried.
 * @return     The route:
 *             true  : is an IPv4 route.
 *             false : is NOT an IPv4 route.
 */
bool demo_rt_ld_is_ip4(const fpp_rt_cmd_t* p_rt)
{
    assert(NULL != p_rt);
    return (bool)(ntohl(p_rt->flags) & FPP_IP_ROUTE_6o4);
}


/*
 * @brief      Query whether a route is an IPv6 route.
 * @details    [localdata_rt]
 * @param[in]  p_rt  Local data to be queried.
 * @return     The route:
 *             true  : is an IPv6 route.
 *             false : is NOT an IPv6 route.
 */
bool demo_rt_ld_is_ip6(const fpp_rt_cmd_t* p_rt)
{
    assert(NULL != p_rt);
    return (bool)(ntohl(p_rt->flags) & FPP_IP_ROUTE_4o6);
}


/*
 * @brief      Query the ID of a route.
 * @details    [localdata_rt]
```

```
 * @param[in]  p_rt  Local data to be queried.
 * @return     ID of the route.
 */
uint32_t demo_rt_ld_get_route_id(const fpp_rt_cmd_t* p_rt)
{
    assert(NULL != p_rt);
    return ntohl(p_rt->id);
}


/*
 * @brief      Query the source MAC of a route.
 * @details    [localdata_rt]
 * @param[in]  p_rt  Local data to be queried.
 * @return     Source MAC which shall be set in the routed traffic.
 */
const uint8_t* demo_rt_ld_get_src_mac(const fpp_rt_cmd_t* p_rt)
{
    assert(NULL != p_rt);
    return (p_rt->src_mac);
}


/*
 * @brief      Query the destination MAC of a route.
 * @details    [localdata_rt]
 * @param[in]  p_rt  Local data to be queried.
 * @return     Destination MAC which shall be set in the routed traffic.
 */
const uint8_t* demo_rt_ld_get_dst_mac(const fpp_rt_cmd_t* p_rt)
{
    assert(NULL != p_rt);
    return (p_rt->dst_mac);
}


/*
 * @brief      Query the egress interface of a route.
 * @details    [localdata_rt]
 * @param[in]  p_rt  Local data to be queried.
 * @return     Name of a physical interface which shall be used as an egress interface.
 */
const char* demo_rt_ld_get_egress_phyif(const fpp_rt_cmd_t* p_rt)
{
    assert(NULL != p_rt);
    return (p_rt->output_device);
}




/*
 * @brief      Query whether an IPv4 conntrack serves as a NAT.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     The IPv4 conntrack:
 *             true  : does serve as a NAT.
 *             false : does NOT serve as a NAT.
 */
bool demo_ct_ld_is_nat(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    /* no need to transform byte order when comparing members of one struct */
    return (bool)(((p_ct->daddr_reply) != (p_ct->saddr)) ||
                  ((p_ct->saddr_reply) != (p_ct->daddr)));
}


/*
 * @brief      Query whether an IPv4 conntrack serves as a PAT.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     The IPv4 conntrack:
 *             true  : does serve as a PAT.
 *             false : does NOT serve as a PAT.
 */
bool demo_ct_ld_is_pat(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    /* no need to transform byte order when comparing members of one struct */
    return (bool)(((p_ct->dport_reply) != (p_ct->sport)) ||
                  ((p_ct->sport_reply) != (p_ct->dport)));
}


/*
```

```
 * @brief      Query whether an IPv4 conntrack modifies VLAN tags.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     The IPv4 conntrack:
 *             true  : does modify VLAN tags of serviced packets.
 *             false : does NOT modify VLAN tags of serviced packets.
 */
bool demo_ct_ld_is_vlan_tagging(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    /* no need to transform byte order when comparing with ZERO */
    return (bool)((0u != p_ct->vlan) || (0u != p_ct->vlan_reply));
}


/*
 * @brief      Query whether an IPv4 conntrack decrements packet's TTL counter or not.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     The IPV4 conntrack:
 *             true  : does decrement TTL counter.
 *             false : does NOT decrement TTL counter.
 */
bool demo_ct_ld_is_ttl_decr(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return (bool)(ntohs(p_ct->flags) & CTCMD_FLAGS_TTL_DECREMENT);
}


/*
 * @brief      Query whether an IPv4 conntrack is orig direction only.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     The IPv4 conntrack:
 *             true  : is orig direction only.
 *             false : is NOT orig direction only.
 */
bool demo_ct_ld_is_orig_only(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return (bool)(ntohs(p_ct->flags) & CTCMD_FLAGS_REP_DISABLED);
}


/*
 * @brief      Query whether an IPv4 conntrack is reply direction only.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     The IPv4 conntrack:
 *             true  : is reply direction only.
 *             false : is NOT reply direction only.
 */
bool demo_ct_ld_is_reply_only(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return (bool)(ntohs(p_ct->flags) & CTCMD_FLAGS_ORIG_DISABLED);
}


/*
 * @brief      Query the protocol of an IPv4 conntrack.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     IP Protocol ID
 */
uint16_t demo_ct_ld_get_protocol(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohs(p_ct->protocol);
}


/*
 * @brief      Query the source address of an IPv4 conntrack.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Source IPv4 address.
 */
uint32_t demo_ct_ld_get_saddr(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohl(p_ct->saddr);
}
```

```
/*
 * @brief      Query the destination address of an IPv4 conntrack.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Destination IPv4 address.
 */
uint32_t demo_ct_ld_get_daddr(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohl(p_ct->daddr);
}


/*
 * @brief      Query the source port of an IPv4 conntrack.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Source port.
 */
uint16_t demo_ct_ld_get_sport(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohs(p_ct->sport);
}


/*
 * @brief      Query the destination port of an IPv4 conntrack.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Destination port.
 */
uint16_t demo_ct_ld_get_dport(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohs(p_ct->dport);
}


/*
 * @brief      Query the used VLAN tag of an IPv4 conntrack.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     VLAN tag. 0 == no VLAN tagging in this direction.
 */
uint16_t demo_ct_ld_get_vlan(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohs(p_ct->vlan);
}


/*
 * @brief      Query the route ID of an IPv4 conntrack.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Route ID.
 */
uint32_t demo_ct_ld_get_route_id(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohl(p_ct->route_id);
}


/*
 * @brief      Query the source address of an IPv4 conntrack (reply direction).
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Source IPv4 address (reply direction).
 */
uint32_t demo_ct_ld_get_saddr_reply(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohl(p_ct->saddr_reply);
}


/*
 * @brief      Query the destination address of an IPv4 conntrack (reply direction).
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Destination IPv4 address (reply direction).
 */
uint32_t demo_ct_ld_get_daddr_reply(const fpp_ct_cmd_t* p_ct)
{
```

```
    assert(NULL != p_ct);
    return ntohl(p_ct->daddr_reply);
}


/*
 * @brief      Query the source port of an IPv4 conntrack (reply direction).
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Source port (reply direction).
 */
uint16_t demo_ct_ld_get_sport_reply(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohs(p_ct->sport_reply);
}


/*
 * @brief      Query the destination port of an IPv4 conntrack (reply direction).
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Destination port (reply direction).
 */
uint16_t demo_ct_ld_get_dport_reply(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohs(p_ct->dport_reply);
}


/*
 * @brief      Query the used VLAN tag of an IPv4 conntrack (reply direction).
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     VLAN tag (reply direction). 0 == no VLAN tagging in this direction.
 */
uint16_t demo_ct_ld_get_vlan_reply(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohs(p_ct->vlan_reply);
}


/*
 * @brief      Query the route ID of an IPv4 conntrack (reply direction).
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Route ID (reply direction).
 */
uint32_t demo_ct_ld_get_route_id_reply(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohl(p_ct->route_id_reply);
}


/*
 * @brief      Query the flags of an IPv4 conntrack.
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Flags bitset at time when the data was obtained from PFE.
 */
uint16_t demo_ct_ld_get_flags(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohs(p_ct->flags);
}


/*
 * @brief      Query the statistics of an IPv4 conntrack (number of frames).
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Number of frames that used the conntrack.
 */
uint32_t demo_ct_ld_get_stt_hit(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohl(p_ct->stats.hit);
}


/*
 * @brief      Query the statistics of an IPv6 conntrack (number of bytes).
 * @details    [localdata_ct]
```

```
 * @param[in]  p_ct  Local data to be queried.
 * @return     Sum of bytesizes of all frames that used the conntrack.
 */
uint32_t demo_ct_ld_get_stt_hit_bytes(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohl(p_ct->stats.hit_bytes);
}


/*
 * @brief      Query the statistics of an IPv4 conntrack (number of frames).
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Number of frames that used the conntrack (reply direction).
 */
uint32_t demo_ct_ld_get_stt_reply_hit(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohl(p_ct->stats_reply.hit);
}


/*
 * @brief      Query the statistics of an IPv4 conntrack (number of bytes).
 * @details    [localdata_ct]
 * @param[in]  p_ct  Local data to be queried.
 * @return     Sum of bytesizes of all frames that used the conntrack (reply direction).
 */
uint32_t demo_ct_ld_get_stt_reply_hit_bytes(const fpp_ct_cmd_t* p_ct)
{
    assert(NULL != p_ct);
    return ntohl(p_ct->stats_reply.hit_bytes);
}




/*
 * @brief      Query whether an IPv6 conntrack serves as a NAT.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     The IPv6 conntrack:
 *             true  : does serve as a NAT.
 *             false : does NOT serve as a NAT.
 */
bool demo_ct6_ld_is_nat(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    /* no need to transform byte order when comparing members of one struct */
    return (bool)((0 != memcmp(p_ct6->daddr_reply, p_ct6->saddr, (4 * sizeof(uint32_t)))) ||
                  (0 != memcmp(p_ct6->saddr_reply, p_ct6->daddr, (4 * sizeof(uint32_t)))));
}


/*
 * @brief      Query whether an IPv6 conntrack serves as a PAT.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     The IPv6 conntrack:
 *             true  : does serve as a PAT.
 *             false : does NOT serve as a PAT.
 */
bool demo_ct6_ld_is_pat(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    /* no need to transform byte order when comparing members of one struct */
    return (bool)(((p_ct6->dport_reply) != (p_ct6->sport)) ||
                  ((p_ct6->sport_reply) != (p_ct6->dport)));
}


/*
 * @brief      Query whether an IPv6 conntrack modifies VLAN tags.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     The IPv6 conntrack:
 *             true  : does modify VLAN tags of serviced packets.
 *             false : does NOT modify VLAN tags of serviced packets.
 */
bool demo_ct6_ld_is_vlan_tagging(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    /* no need to transform byte order when comparing with ZERO */
    return (bool)((0u != p_ct6->vlan) || (0u != p_ct6->vlan_reply));
}
```



```
/*
 * @brief      Query whether an IPv6 conntrack decrements packet's TTL counter or not.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     The IPV6 conntrack:
 *             true  : does decrement TTL counter.
 *             false : does NOT decrement TTL counter.
 */
bool demo_ct6_ld_is_ttl_decr(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return (bool)(ntohs(p_ct6->flags) & CTCMD_FLAGS_TTL_DECREMENT);
}


/*
 * @brief      Query whether an IPv6 conntrack is orig direction only.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     The IPv6 conntrack:
 *             true  : is orig direction only.
 *             false : is NOT orig direction only.
 */
bool demo_ct6_ld_is_orig_only(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return (bool)(ntohs(p_ct6->flags) & CTCMD_FLAGS_REP_DISABLED);
}


/*
 * @brief      Query whether an IPv6 conntrack is reply direction only.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     The IPv6 conntrack:
 *             true  : is reply direction only.
 *             false : is NOT reply direction only.
 */
bool demo_ct6_ld_is_reply_only(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return (bool)(ntohs(p_ct6->flags) & CTCMD_FLAGS_ORIG_DISABLED);
}


/*
 * @brief      Query the protocol of an IPv6 conntrack.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     IP Protocol ID.
 */
uint16_t demo_ct6_ld_get_protocol(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohs(p_ct6->protocol);
}


/*
 * @brief      Query the source address of an IPv6 conntrack.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Source IPv6 address.
 */
const uint32_t* demo_ct6_ld_get_saddr(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    static uint32_t rtn_saddr[4] = {0u};

    rtn_saddr[0] = ntohl(p_ct6->saddr[0]);
    rtn_saddr[1] = ntohl(p_ct6->saddr[1]);
    rtn_saddr[2] = ntohl(p_ct6->saddr[2]);
    rtn_saddr[3] = ntohl(p_ct6->saddr[3]);

    return (rtn_saddr);
}


/*
 * @brief      Query the destination address of an IPv6 conntrack.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Destination IPv4 address.
 */
```

```
const uint32_t* demo_ct6_ld_get_daddr(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    static uint32_t rtn_daddr[4] = {0u};

    rtn_daddr[0] = ntohl(p_ct6->daddr[0]);
    rtn_daddr[1] = ntohl(p_ct6->daddr[1]);
    rtn_daddr[2] = ntohl(p_ct6->daddr[2]);
    rtn_daddr[3] = ntohl(p_ct6->daddr[3]);

    return (rtn_daddr);
}


/*
 * @brief      Query the source port of an IPv6 conntrack.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Source port.
 */
uint16_t demo_ct6_ld_get_sport(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohs(p_ct6->sport);
}


/*
 * @brief      Query the destination port of an IPv6 conntrack.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Destination port.
 */
uint16_t demo_ct6_ld_get_dport(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohs(p_ct6->dport);
}


/*
 * @brief      Query the used VLAN tag of an IPv6 conntrack.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     VLAN tag. 0 == no VLAN tagging in this direction.
 */
uint16_t demo_ct6_ld_get_vlan(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohs(p_ct6->vlan);
}


/*
 * @brief      Query the route ID of an IPv6 conntrack.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Route ID
 */
uint32_t demo_ct6_ld_get_route_id(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohl(p_ct6->route_id);
}


/*
 * @brief      Query the source address of an IPv6 conntrack (reply direction).
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Source IPv6 address (reply direction).
 */
const uint32_t* demo_ct6_ld_get_saddr_reply(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    static uint32_t rtn_saddr_reply[4] = {0u};

    rtn_saddr_reply[0] = ntohl(p_ct6->saddr_reply[0]);
    rtn_saddr_reply[1] = ntohl(p_ct6->saddr_reply[1]);
    rtn_saddr_reply[2] = ntohl(p_ct6->saddr_reply[2]);
    rtn_saddr_reply[3] = ntohl(p_ct6->saddr_reply[3]);

    return (rtn_saddr_reply);
}
```

```
/*
 * @brief      Query the destination address of an IPv6 conntrack (reply direction).
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Destination IPv6 address (reply direction).
 */
const uint32_t* demo_ct6_ld_get_daddr_reply(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    static uint32_t rtn_daddr_reply[4] = {0u};

    rtn_daddr_reply[0] = ntohl(p_ct6->daddr_reply[0]);
    rtn_daddr_reply[1] = ntohl(p_ct6->daddr_reply[1]);
    rtn_daddr_reply[2] = ntohl(p_ct6->daddr_reply[2]);
    rtn_daddr_reply[3] = ntohl(p_ct6->daddr_reply[3]);

    return (rtn_daddr_reply);
}


/*
 * @brief      Query the source port of an IPv6 conntrack (reply direction).
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Source port (reply direction).
 */
uint16_t demo_ct6_ld_get_sport_reply(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohs(p_ct6->sport_reply);
}


/*
 * @brief      Query the destination port of an IPv6 conntrack (reply direction).
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Destination port (reply direction).
 */
uint16_t demo_ct6_ld_get_dport_reply(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohs(p_ct6->dport_reply);
}


/*
 * @brief      Query the used VLAN tag of an IPv6 conntrack (reply direction).
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     VLAN tag (reply direction). 0 == no VLAN tagging in this direction.
 */
uint16_t demo_ct6_ld_get_vlan_reply(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohs(p_ct6->vlan_reply);
}


/*
 * @brief      Query the route ID of an IPv6 conntrack (reply direction).
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Route ID (reply direction).
 */
uint32_t demo_ct6_ld_get_route_id_reply(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohl(p_ct6->route_id_reply);
}


/*
 * @brief      Query the flags of an IPv6 conntrack.
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Flags bitset at time when the data was obtained from PFE.
 */
uint16_t demo_ct6_ld_get_flags(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohs(p_ct6->flags);
}


/*
```

```
 * @brief      Query the statistics of an IPv6 conntrack (number of frames).
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Number of frames that used the conntrack.
 */
uint32_t demo_ct6_ld_get_stt_hit(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohl(p_ct6->stats.hit);
}


/*
 * @brief      Query the statistics of an IPv6 conntrack (number of bytes).
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Sum of bytesizes of all frames that used the conntrack.
 */
uint32_t demo_ct6_ld_get_stt_hit_bytes(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohl(p_ct6->stats.hit_bytes);
}


/*
 * @brief      Query the statistics of an IPv6 conntrack (number of frames).
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Number of frames that used the conntrack (reply direction).
 */
uint32_t demo_ct6_ld_get_stt_reply_hit(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohl(p_ct6->stats_reply.hit);
}


/*
 * @brief      Query the statistics of an IPv6 conntrack (number of bytes).
 * @details    [localdata_ct6]
 * @param[in]  p_ct6  Local data to be queried.
 * @return     Sum of bytesizes of all frames that used the conntrack (reply direction).
 */
uint32_t demo_ct6_ld_get_stt_reply_hit_bytes(const fpp_ct6_cmd_t* p_ct6)
{
    assert(NULL != p_ct6);
    return ntohl(p_ct6->stats_reply.hit_bytes);
}


/* ==== PUBLIC FUNCTIONS : misc ============================================ */


/*
 * @brief      Use FCI calls to iterate through all available routes in PFE and
 *             execute a callback print function for each applicable route.
 * @param[in]  p_cl        FCI client
 * @param[in]  p_cb_print  Callback print function.
 *                         --> If the callback returns ZERO, then all is OK and
 *                             a next route is picked for a print process.
 *                         --> If the callback returns NON-ZERO, then some problem is
 *                             assumed and this function terminates prematurely.
 * @param[in]  print_ip4   Set true to print IPv4 routes.
 * @param[in]  print_ip6   Set true to print IPv6 routes.
 * @return     FPP_ERR_OK : Successfully iterated through all applicable routes.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_rt_print_all(FCI_CLIENT* p_cl, demo_rt_cb_print_t p_cb_print,
                      bool print_ip4, bool print_ip6)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_rt_cmd_t cmd_to_fci = {0};
    fpp_rt_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_IP_ROUTE,
                    sizeof(fpp_rt_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));
```

```
    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        if ((print_ip4) && demo_rt_ld_is_ip4(&reply_from_fci))
        {
            rtn = p_cb_print(&reply_from_fci);  /* print IPv4 route */
        }
        if ((print_ip6) && demo_rt_ld_is_ip6(&reply_from_fci))
        {
            rtn = p_cb_print(&reply_from_fci);  /* print IPv6 route */
        }

        if (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_IP_ROUTE,
                            sizeof(fpp_rt_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more routes to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_RT_ENTRY_NOT_FOUND == rtn)
    {
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_rt_print_all() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available routes in PFE.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_count  Space to store the count of routes.
 * @return      FPP_ERR_OK : Successfully counted all available routes.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_rt_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_rt_cmd_t cmd_to_fci = {0};
    fpp_rt_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint32_t count = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_IP_ROUTE,
                    sizeof(fpp_rt_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /*  query loop  */
    while (FPP_ERR_OK == rtn)
    {
        count++;

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_IP_ROUTE,
                        sizeof(fpp_rt_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* query loop runs till there are no more routes to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_RT_ENTRY_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_rt_get_count() failed!");

    return (rtn);
}
```

```
/*
 * @brief      Use FCI calls to iterate through all available IPv4 conntracks in PFE and
 *             execute a callback print function for each reported IPv4 conntrack.
 * @param[in]  p_cl        FCI client
 * @param[in]  p_cb_print  Callback print function.
 *                         --> If the callback returns ZERO, then all is OK and
 *                             a next IPv4 conntrack is picked for a print process.
 *                         --> If the callback returns NON-ZERO, then some problem is
 *                             assumed and this function terminates prematurely.
 * @return     FPP_ERR_OK : Successfully iterated through all available IPv4 conntracks.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_ct_print_all(FCI_CLIENT* p_cl, demo_ct_cb_print_t p_cb_print)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_ct_cmd_t cmd_to_fci = {0};
    fpp_ct_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_IPV4_CONNTRACK,
                    sizeof(fpp_ct_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        rtn = p_cb_print(&reply_from_fci);

        if (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_IPV4_CONNTRACK,
                            sizeof(fpp_ct_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more IPv4 conntracks to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_CT_ENTRY_NOT_FOUND == rtn)
    {
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_ct_print_all() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available IPv4 conntracks in PFE.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_count  Space to store the count of IPv4 conntracks.
 * @return      FPP_ERR_OK : Successfully counted all available IPv4 conntracks.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_ct_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_ct_cmd_t cmd_to_fci = {0};
    fpp_ct_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint32_t count = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_IPV4_CONNTRACK,
                    sizeof(fpp_ct_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
```

```
    while (FPP_ERR_OK == rtn)
    {
        count++;

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_IPV4_CONNTRACK,
                        sizeof(fpp_ct_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* query loop runs till there are no more IPv4 conntracks to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_CT_ENTRY_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_ct_get_count() failed!");

    return (rtn);
}




/*
 * @brief      Use FCI calls to iterate through all available IPv6 conntracks in PFE and
 *             execute a callback print function for each reported IPv6 conntrack.
 * @param[in]  p_cl        FCI client
 * @param[in]  p_cb_print  Callback print function.
 *                         --> If the callback returns ZERO, then all is OK and
 *                             a next IPv6 conntrack is picked for a print process.
 *                         --> If the callback returns NON-ZERO, then some problem is
 *                             assumed and this function terminates prematurely.
 * @return     FPP_ERR_OK : Successfully iterated through all available IPv6 conntracks.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_ct6_print_all(FCI_CLIENT* p_cl, demo_ct6_cb_print_t p_cb_print)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_ct6_cmd_t cmd_to_fci = {0};
    fpp_ct6_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_IPV6_CONNTRACK,
                    sizeof(fpp_ct6_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        rtn = p_cb_print(&reply_from_fci);

        if (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_IPV6_CONNTRACK,
                            sizeof(fpp_ct6_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more IPv6 conntracks to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_CT_ENTRY_NOT_FOUND == rtn)
    {
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_ct6_print_all() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available IPv6 conntracks in PFE.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_count  Space to store the count of IPv6 conntracks.
```

```
 * @return      FPP_ERR_OK : Successfully counted all available IPv6 conntracks.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_ct6_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_ct6_cmd_t cmd_to_fci = {0};
    fpp_ct6_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint32_t count = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_IPV6_CONNTRACK,
                    sizeof(fpp_ct6_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        count++;

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_IPV6_CONNTRACK,
                        sizeof(fpp_ct6_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* query loop runs till there are no more IPv6 conntracks to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_CT_ENTRY_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_ct6_get_count() failed!");

    return (rtn);
}


/* ========================================================================= */
```

15.19 demo_spd.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
 * ======================================================================== */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_spd.h"


/* ==== PRIVATE FUNCTIONS ================================================== */


/*
 * @brief       Set/unset a flag in a SecurityPolicy struct.
 * @param[out]  p_rtn_spd  Struct to be modified.
 * @param[in]   enable     New state of a flag.
 * @param[in]   flag       The flag.
 */
static void set_spd_flag(fpp_spd_cmd_t* p_rtn_spd, bool enable, fpp_spd_flags_t flag)
{
    assert(NULL != p_rtn_spd);

    hton_enum(&flag, sizeof(fpp_spd_flags_t));

    if (enable)
    {
        p_rtn_spd->flags |= flag;
    }
    else
    {
        p_rtn_spd->flags &= (fpp_spd_flags_t)(~flag);
    }
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from PFE ============== */


/*
 * @brief       Use FCI calls to get configuration data of a requested SecurityPolicy
 *              from PFE. Identify the SecurityPolicy by a name of the parent
 *              physical interface (each physical interface has its own SPD) and by
 *              a position of the policy in the SPD.
 * @param[in]   p_cl           FCI client
 * @param[out]  p_rtn_spd      Space for data from PFE.
 * @param[in]   p_phyif_name   Name of a parent physical interface.
 *                             Names of physical interfaces are hardcoded.
 *                             See FCI API Reference, chapter Interface Management.
 *              position       Position of the requested SecurityPolicy in the SPD.
 * @return      FPP_ERR_OK : The requested SecurityPolicy was found.
 *                           A copy of its configuration data was stored into p_rtn_spd.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_spd_get_by_position(FCI_CLIENT* p_cl, fpp_spd_cmd_t* p_rtn_spd,
                             const char* p_phyif_name, uint16_t position)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_spd);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_spd_cmd_t cmd_to_fci = {0};
    fpp_spd_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_SPD,
                        sizeof(fpp_spd_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
```

```
        /* query loop (with a search condition) */
        while ((FPP_ERR_OK == rtn) && (ntohs(reply_from_fci.position) != position))
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_SPD,
                            sizeof(fpp_spd_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_spd = reply_from_fci;
    }

    print_if_error(rtn, "demo_spd_get_by_position() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to add/del items in PFE =========== */


/*
 * @brief       Use FCI calls to create a new SecurityPolicy in PFE.
 *              The new policy is added into SPD of a provided parent physical interface.
 * @param[in]   p_cl          FCI client instance
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @param[in]   position      Position of the new SecurityPolicy in the SPD.
 * @param[in]   p_spd_data    Configuration data for the new SecurityPolicy.
 *                            To create a new SecurityPolicy, a local data struct must be
 *                            created, configured and then passed to this function.
 *                            See [localdata_spd] to learn more.
 * @return      FPP_ERR_OK : New SecurityPolicy was created.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_spd_add(FCI_CLIENT* p_cl, const char* p_phyif_name, uint16_t position,
                 const fpp_spd_cmd_t* p_spd_data)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_spd_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci = (*p_spd_data);
    cmd_to_fci.position = htons(position);
    rtn = set_text((cmd_to_fci.name), p_phyif_name, IFNAMSIZ);

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_REGISTER;
        rtn = fci_write(p_cl, FPP_CMD_SPD, sizeof(fpp_spd_cmd_t),
                                          (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_spd_add() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to destroy the target SecurityPolicy in PFE.
 * @param[in]   p_cl          FCI client instance
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @param[in]   position      Position of the target SecurityPolicy in the SPD.
 * @return      FPP_ERR_OK : The SecurityPolicy was destroyed.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_spd_del(FCI_CLIENT* p_cl, const char* p_phyif_name, uint16_t position)
{
    assert(NULL != p_cl);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_spd_cmd_t cmd_to_fci = {0};

    /*  prepare data  */
```

```
    cmd_to_fci.position = htons(position);
    rtn = set_text((cmd_to_fci.name), p_phyif_name, IFNAMSIZ);

    /*  send data  */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_DEREGISTER;
        rtn = fci_write(p_cl, FPP_CMD_SPD, sizeof(fpp_spd_cmd_t),
                                          (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_spd_del() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : modify local data (no FCI calls) ================ */
/*
 * @defgroup    localdata_spd  [localdata_spd]
 * @brief:      Functions marked as [localdata_spd] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_spd (a struct with configuration data).
 *              When adding a new SecurityPolicy, there are no "initial data"
 *              to be obtained from PFE. Simply declare a local data struct and configure it.
 *              Then, after all modifications are done and finished,
 *              call demo_spd_add() to create a new SecurityPolicy in PFE.
 */


/*
 * @brief          Set a protocol type of a SecurityPolicy.
 * @details        [localdata_spd]
 * @param[in,out]  p_spd     Local data to be modified.
 * @param[in]      protocol  IP protocol ID
 *                           See "IANA Assigned Internet Protocol Number":
 *                https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
 */
void demo_spd_ld_set_protocol(fpp_spd_cmd_t* p_spd, uint8_t protocol)
{
    assert(NULL != p_spd);
    p_spd->protocol = protocol;
}


/*
 * @brief          Set a source/destination IP address of a SecurityPolicy.
 * @details        [localdata_spd]
 *                 BEWARE: Address type (IPv4/IPv6) of p_saddr and p_daddr must be the same!
 * @param[in,out]  p_spd    Local data to be modified.
 * @param[in]      p_saddr  Source IP address (IPv4 or IPv6).
 * @param[in]      p_daddr  Destination IP address (IP4 or IP6).
 * @param[in]      is_ip6   Set if provided addresses are IPv6 type addresses.
 */
void demo_spd_ld_set_ip(fpp_spd_cmd_t* p_spd, const uint32_t p_saddr[4],
                        const uint32_t p_daddr[4], bool is_ip6)
{
    assert(NULL != p_spd);
    assert(NULL != p_saddr);
    assert(NULL != p_daddr);

    if (is_ip6)
    {
        p_spd->saddr[0] = htonl(p_saddr[0]);
        p_spd->saddr[1] = htonl(p_saddr[1]);
        p_spd->saddr[2] = htonl(p_saddr[2]);
        p_spd->saddr[3] = htonl(p_saddr[3]);

        p_spd->daddr[0] = htonl(p_daddr[0]);
        p_spd->daddr[1] = htonl(p_daddr[1]);
        p_spd->daddr[2] = htonl(p_daddr[2]);
        p_spd->daddr[3] = htonl(p_daddr[3]);
    }
    else
    {
        p_spd->saddr[0] = htonl(p_saddr[0]);
        p_spd->saddr[1] = 0u;
        p_spd->saddr[2] = 0u;
        p_spd->saddr[3] = 0u;

        p_spd->daddr[0] = htonl(p_daddr[0]);
        p_spd->daddr[1] = 0u;
        p_spd->daddr[2] = 0u;
        p_spd->daddr[3] = 0u;
    }
```

```
    set_spd_flag(p_spd, is_ip6, FPP_SPD_FLAG_IPv6);
}


/*
 * @brief          Set a source/destination port of a SecurityPolicy.
 * @details        [localdata_spd]
 * @param[in,out]  p_spd      Local data to be modified.
 * @param[in]      use_sport  Prompt to use the source port value of this SecurityPolicy
 *                            during SPD matching process (evaluation which policy to use).
 *                            If false, then source port of the given SecurityPolicy is
 *                            ignored (not tested) when the policy is evaluated.
 * @param[in]      sport      Source port
 * @param[in]      use_dport  Prompt to use the destination port value of this SecurityPolicy
 *                            during SPD matching process (evaluation which policy to use).
 *                            If false, then destination port of the given SecurityPolicy is
 *                            ignored (not tested) when the policy is evaluated.
 * @param[in]      dport      Destination port
 */
void demo_spd_ld_set_port(fpp_spd_cmd_t* p_spd, bool use_sport, uint16_t sport,
                          bool use_dport, uint16_t dport)
{
    assert(NULL != p_spd);

    p_spd->sport = ((use_sport) ? htons(sport) : (0u));
    p_spd->dport = ((use_dport) ? htons(dport) : (0u));
    set_spd_flag(p_spd, (!use_sport), FPP_SPD_FLAG_SPORT_OPAQUE);  /* inverted logic */
    set_spd_flag(p_spd, (!use_dport), FPP_SPD_FLAG_DPORT_OPAQUE);  /* inverted logic */
}


/*
 * @brief          Set action of a SecurityPolicy.
 * @details        [localdata_spd]
 * @param[in,out]  p_spd       Local data to be modified.
 * @param[in]      spd_action  Action to do if traffic matches this SecurityPolicy.
 *                             See description of the fpp_spd_action_t type in
 *                             FCI API Reference.
 * @param[in]      sa_id  Meaningful ONLY if the action is FPP_SPD_ACTION_PROCESS_ENCODE.
 *                        ID of an item in the SAD (Security Association Database).
 *                        SAD is stored in the HSE FW (Hardware Security Engine).
 * @param[in]      spi    Meaningful ONLY if the action is FPP_SPD_ACTION_PROCESS_DECODE.
 *                        Security Parameter Index (will be looked for in the traffic data).
 */
void demo_spd_ld_set_action(fpp_spd_cmd_t* p_spd, fpp_spd_action_t spd_action,
                            uint32_t sa_id, uint32_t spi)
{
    assert(NULL != p_spd);

    {
        fpp_spd_action_t tmp_action = spd_action;
        hton_enum(&tmp_action, sizeof(fpp_spd_action_t));
        p_spd->spd_action = tmp_action;
    }

    p_spd->sa_id = ((FPP_SPD_ACTION_PROCESS_ENCODE == spd_action) ? htonl(sa_id) : (0uL));
    p_spd->spi   = ((FPP_SPD_ACTION_PROCESS_DECODE == spd_action) ? htonl(spi)   : (0uL));
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */


/*
 * @brief      Query address type of a SecurityPolicy.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     IP address of the policy:
 *             true  : is IPv6 type.
 *             false : is NOT IPv6 type.
 */
bool demo_spd_ld_is_ip6(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);

    fpp_spd_flags_t tmp_flags = (p_spd->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_spd_flags_t));

    return (bool)(tmp_flags & FPP_SPD_FLAG_IPv6);
}


/*
 * @brief      Query whether the source port value is used during SPD matching process.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
```



```
 * @return     Source port value:
 *             true  : is used in a matching process.
 *             false : is NOT used in a matching process.
 */
bool demo_spd_ld_is_used_sport(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);

    fpp_spd_flags_t tmp_flags = (p_spd->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_spd_flags_t));

    return !(tmp_flags & FPP_SPD_FLAG_SPORT_OPAQUE);  /* the flag has inverted logic */
}


/*
 * @brief      Query whether the destination port value is used during SPD matching process.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     Destination port value:
 *             true  : is used in a matching process.
 *             false : is NOT used in a matching process.
 */
bool demo_spd_ld_is_used_dport(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);

    fpp_spd_flags_t tmp_flags = (p_spd->flags);
    ntoh_enum(&tmp_flags, sizeof(fpp_spd_flags_t));

    return !(tmp_flags & FPP_SPD_FLAG_DPORT_OPAQUE);  /* the flag has inverted logic */
}




/*
 * @brief      Query the position of a SecurityPolicy.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     Position of the Security Policy within the SPD.
 */
uint16_t demo_spd_ld_get_position(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);
    return ntohs(p_spd->position);
}


/*
 * @brief      Query the source IP address of a SecurityPolicy.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     Source IP address.
 *             Use demo_spd_ld_is_ip6() to distinguish between IPv4 and IPv6.
 */
const uint32_t* demo_spd_ld_get_saddr(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);
    static uint32_t rtn_saddr[4] = {0u};

    rtn_saddr[0] = htonl(p_spd->saddr[0]);
    rtn_saddr[1] = htonl(p_spd->saddr[1]);
    rtn_saddr[2] = htonl(p_spd->saddr[2]);
    rtn_saddr[3] = htonl(p_spd->saddr[3]);

    return (rtn_saddr);
}


/*
 * @brief      Query the destination IP address of a SecurityPolicy.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     Destination IP address.
 *             Use demo_spd_ld_is_ip6() to distinguish between IPv4 and IPv6.
 */
const uint32_t* demo_spd_ld_get_daddr(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);
    static uint32_t rtn_daddr[4] = {0u};

    rtn_daddr[0] = htonl(p_spd->daddr[0]);
    rtn_daddr[1] = htonl(p_spd->daddr[1]);
    rtn_daddr[2] = htonl(p_spd->daddr[2]);
    rtn_daddr[3] = htonl(p_spd->daddr[3]);
```

```
    return (rtn_daddr);
}


/*
 * @brief      Query the source port of a SecurityPolicy.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     Source port.
 */
uint16_t demo_spd_ld_get_sport(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);
    return ntohs(p_spd->sport);
}


/*
 * @brief      Query the destination port of a SecurityPolicy.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     Destination port.
 */
uint16_t demo_spd_ld_get_dport(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);
    return ntohs(p_spd->dport);
}


/*
 * @brief      Query the destination port of a SecurityPolicy.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     IP Protocol ID
 */
uint8_t demo_spd_ld_get_protocol(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);
    return (p_spd->protocol);
}


/*
 * @brief      Query the ID of an item in the SAD (Security Association Database).
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     ID of an associated item in the SAD.
 *             Meaningful ONLY if the action is FPP_SPD_ACTION_PROCESS_ENCODE.
 */
uint32_t demo_spd_ld_get_sa_id(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);
    return ntohl(p_spd->sa_id);
}


/*
 * @brief      Query the SPI tag of a SecurityPolicy.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     SPI tag associated with the SecurityPolicy.
 *             Meaningful ONLY if the action is FPP_SPD_ACTION_PROCESS_DECODE.
 */
uint32_t demo_spd_ld_get_spi(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);
    return ntohl(p_spd->spi);
}


/*
 * @brief      Query the action of a SecurityPolicy.
 * @details    [localdata_spd]
 * @param[in]  p_spd  Local data to be queried.
 * @return     Action to be done if this SecurityPolicy is utilized.
 */
fpp_spd_action_t demo_spd_ld_get_action(const fpp_spd_cmd_t* p_spd)
{
    assert(NULL != p_spd);

    fpp_spd_action_t tmp_action = (p_spd->spd_action);
    ntoh_enum(&tmp_action, sizeof(fpp_spd_action_t));

    return (tmp_action);
```

```
}


/* ==== PUBLIC FUNCTIONS : misc ============================================ */


/*
 * @brief      Use FCI calls to iterate through all SecurityPolicies of a given physical
 *             interface and execute a callback print function for each SecurityPolicy.
 * @param[in]  p_cl           FCI client
 * @param[in]  p_cb_print     Callback print function.
 *                            --> If the callback returns ZERO, then all is OK and
 *                                a next SecurityPolicy is picked for a print process.
 *                            --> If the callback returns NON-ZERO, then some problem is
 *                                assumed and this function terminates prematurely.
 * @param[in]  p_phyif_name   Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @param[in]  position_init  Start invoking a callback print function from
 *                            this position in the SPD.
 *                            If 0, start from the very first SecurityPolicy in the SPD.
 * @param[in]  count          Print only this count of SecurityPolicies, then end.
 *                            If 0, keep printing SecurityPolicies till the end of the SPD.
 * @return     FPP_ERR_OK : Successfully iterated through all SecrityPolicies of
 *                          the given physical interface.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_spd_print_by_phyif(FCI_CLIENT* p_cl, demo_spd_cb_print_t p_cb_print,
                            const char* p_phyif_name, uint16_t position_init, uint16_t count)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_spd_cmd_t cmd_to_fci = {0};
    fpp_spd_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_phyif_name, IFNAMSIZ);
    if (0u == count)  /* if 0, set max possible count of items */
    {
        count--;  /* WARNING: intentional use of owf behavior */
    }

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_SPD,
                        sizeof(fpp_spd_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

        /* query loop */
        uint16_t position = 0u;
        while ((FPP_ERR_OK == rtn) && (0u != count))
        {
            if (position >= position_init)
            {
                rtn = p_cb_print(&reply_from_fci);
                count--;
            }

            position++;

            if (FPP_ERR_OK == rtn)
            {
                cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
                rtn = fci_query(p_cl, FPP_CMD_SPD,
                                sizeof(fpp_spd_cmd_t), (unsigned short*)(&cmd_to_fci),
                                &reply_length, (unsigned short*)(&reply_from_fci));
            }
        }

        /* query loop runs till there are no more SecurityPolicies to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_IF_ENTRY_NOT_FOUND == rtn)
        {
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_spd_print_by_phyif() failed!");
```

```
    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all SecurityPolicies in PFE which are
 *              associated with the given physical interface.
 * @param[in]   p_cl          FCI client
 * @param[out]  p_rtn_count   Space to store the count of SecurityPolicies.
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : Successfully counted all available SecurityPolicies of
 *                           the given physical interface. Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_spd_get_count_by_phyif(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                                const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_spd_cmd_t cmd_to_fci = {0};
    fpp_spd_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint16_t count = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_SPD,
                        sizeof(fpp_spd_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

        /*  query loop  */
        while (FPP_ERR_OK == rtn)
        {
            count++;

            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_SPD,
                            sizeof(fpp_spd_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }

        /* query loop runs till there are no more SecurityPolicies to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_IF_ENTRY_NOT_FOUND == rtn)
        {
            *p_rtn_count = count;
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_spd_get_count_by_phyif() failed!");

    return (rtn);
}


/* ========================================================================= */
```

15.20 demo_qos.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
```

```
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_qos.h"


/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from PFE ============== */


/*
 * @brief       Use FCI calls to get configuration data of a requested QoS queue
 *              from PFE. Identify the QoS queue by the name of a parent
 *              physical interface and by the queue's ID.
 * @param[in]   p_cl          FCI client
 * @param[out]  p_rtn_que     Space for data from PFE.
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 *              que_id        ID of the requested QoS queue.
 * @return      FPP_ERR_OK : The requested QoS queue was found.
 *                           A copy of its configuration data was stored into p_rtn_que.
 *                           REMINDER: Data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_qos_que_get_by_id(FCI_CLIENT* p_cl, fpp_qos_queue_cmd_t* p_rtn_que,
                           const char* p_phyif_name, uint8_t que_id)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_que);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_queue_cmd_t cmd_to_fci = {0};
    fpp_qos_queue_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    cmd_to_fci.id = que_id;
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query (get the QoS queue directly; no need for a loop) */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_QOS_QUEUE,
                        sizeof(fpp_qos_queue_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
```

```
    {
        *p_rtn_que = reply_from_fci;
    }

    print_if_error(rtn, "demo_qos_que_get_by_id() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get configuration data of a requested QoS scheduler
 *              from PFE. Identify the QoS scheduler by the name of a parent
 *              physical interface and by the scheduler's ID.
 * @param[in]   p_cl          FCI client
 * @param[out]  p_rtn_que     Space for data from PFE.
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 *              sch_id        ID of the requested QoS scheduler.
 * @return      FPP_ERR_OK : The requested QoS scheduler was found.
 *                           A copy of its configuration data was stored into p_rtn_sch.
 *                           REMINDER: Data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_qos_sch_get_by_id(FCI_CLIENT* p_cl, fpp_qos_scheduler_cmd_t* p_rtn_sch,
                           const char* p_phyif_name, uint8_t sch_id)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_sch);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_scheduler_cmd_t cmd_to_fci = {0};
    fpp_qos_scheduler_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    cmd_to_fci.id = sch_id;
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query (get the QoS scheduler directly; no need for a loop) */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_QOS_SCHEDULER,
                        sizeof(fpp_qos_scheduler_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_sch = reply_from_fci;
    }

    print_if_error(rtn, "demo_qos_sch_get_by_id() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get configuration data of a requested QoS shaper
 *              from PFE. Identify the QoS shaper by the name of a parent
 *              physical interface and by the shaper's ID.
 * @param[in]   p_cl          FCI client
 * @param[out]  p_rtn_que     Space for data from PFE.
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 *              shp_id        ID of the requested QoS shaper.
 * @return      FPP_ERR_OK : The requested QoS shaper was found.
 *                           A copy of its configuration data was stored into p_rtn_shp.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_qos_shp_get_by_id(FCI_CLIENT* p_cl, fpp_qos_shaper_cmd_t* p_rtn_shp,
                           const char* p_phyif_name, uint8_t shp_id)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_shp);
    assert(NULL != p_phyif_name);
```

```
    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_shaper_cmd_t cmd_to_fci = {0};
    fpp_qos_shaper_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    cmd_to_fci.id = shp_id;
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query (get the QoS shaper directly; no need for a loop) */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_QOS_SHAPER,
                        sizeof(fpp_qos_shaper_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_shp = reply_from_fci;
    }

    print_if_error(rtn, "demo_qos_shp_get_by_id() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to update data in PFE ============= */


/*
 * @brief          Use FCI calls to update configuration of a target QoS queue
 *                 in PFE.
 * @param[in]      p_cl   FCI client
 * @param[in,out]  p_que  Local data struct which represents a new configuration of
 *                        the target QoS queue.
 *                        Initial data can be obtained via demo_qos_que_get_by_id().
 * @return        FPP_ERR_OK : Configuration of the target QoS queue was
 *                             successfully updated in PFE.
 *                             The local data struct was automatically updated with
 *                             readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                             The local data struct not updated.
 */
int demo_qos_que_update(FCI_CLIENT* p_cl, fpp_qos_queue_cmd_t* p_que)
{
    assert(NULL != p_cl);
    assert(NULL != p_que);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_qos_queue_cmd_t cmd_to_fci = (*p_que);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_QOS_QUEUE, sizeof(fpp_qos_queue_cmd_t),
                                    (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_qos_que_get_by_id(p_cl, p_que, (p_que->if_name), (p_que->id));
    }

    print_if_error(rtn, "demo_qos_que_update() failed!");

    return (rtn);
}


/*
 * @brief          Use FCI calls to update configuration of a target QoS scheduler
 *                 in PFE.
 * @param[in]      p_cl   FCI client
 * @param[in,out]  p_sch  Local data struct which represents a new configuration of
 *                        the target QoS scheduler.
 *                        Initial data can be obtained via demo_qos_sch_get_by_id().
 * @return        FPP_ERR_OK : Configuration of the target QoS scheduler was
 *                             successfully updated in PFE.
 *                             The local data struct was automatically updated with
 *                             readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                             The local data struct not updated.
```

```
 */
int demo_qos_sch_update(FCI_CLIENT* p_cl, fpp_qos_scheduler_cmd_t* p_sch)
{
    assert(NULL != p_cl);
    assert(NULL != p_sch);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_qos_scheduler_cmd_t cmd_to_fci = (*p_sch);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_QOS_SCHEDULER, sizeof(fpp_qos_scheduler_cmd_t),
                                            (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_qos_sch_get_by_id(p_cl, p_sch, (p_sch->if_name), (p_sch->id));
    }

    print_if_error(rtn, "demo_qos_sch_update() failed!");

    return (rtn);
}


/*
 * @brief          Use FCI calls to update configuration of a target QoS shaper
 *                 in PFE.
 * @param[in]      p_cl   FCI client
 * @param[in,out]  p_shp  Local data struct which represents a new configuration of
 *                        the target QoS shaper.
 *                        Initial data can be obtained via demo_qos_shp_get_by_id().
 * @return        FPP_ERR_OK : Configuration of the target QoS shaper was
 *                             successfully updated in PFE.
 *                             The local data struct was automatically updated with
 *                             readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                             The local data struct not updated.
 */
int demo_qos_shp_update(FCI_CLIENT* p_cl, fpp_qos_shaper_cmd_t* p_shp)
{
    assert(NULL != p_cl);
    assert(NULL != p_shp);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_qos_shaper_cmd_t cmd_to_fci = (*p_shp);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_QOS_SHAPER, sizeof(fpp_qos_shaper_cmd_t),
                                         (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_qos_shp_get_by_id(p_cl, p_shp, (p_shp->if_name), (p_shp->id));
    }

    print_if_error(rtn, "demo_qos_shp_update() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : modify local data (no FCI calls) ================ */
/*
 * @defgroup    localdata_que  [localdata_que]
 * @brief:      Functions marked as [localdata_que] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_que (a struct with configuration data).
 *              Initial data for p_que can be obtained via demo_qos_que_get_by_id().
 *              If some modifications are made to local data, then after all modifications
 *              are done and finished, call demo_qos_que_update() to update
 *              the configuration of a real QoS queue in PFE.
 */


/*
 * @brief          Set a mode (queue discipline) of a QoS queue.
 * @details        [localdata_que]
 * @param[in,out]  p_que     Local data to be modified.
 * @param[in]      que_mode  Queue mode (queue discipline).
 *                           For valid modes, see FCI API Reference,
 *                           chapter 'fpp_qos_queue_cmd_t'.
 */
```

```
void demo_qos_que_ld_set_mode(fpp_qos_queue_cmd_t* p_que, uint8_t que_mode)
{
    assert(NULL != p_que);
    p_que->mode = que_mode;
}


/*
 * @brief          Set a minimal threshold of a QoS queue.
 * @details        [localdata_que]
 *                 Meaning of a minimal threshold depends on
 *                 a queue mode of the given QoS queue.
 * @param[in,out]  p_que  Local data to be modified.
 * @param[in]      min    Minimal threshold.
 */
void demo_qos_que_ld_set_min(fpp_qos_queue_cmd_t* p_que, uint32_t min)
{
    assert(NULL != p_que);
    p_que->min = htonl(min);
}


/*
 * @brief          Set a maximal threshold of a QoS queue.
 * @details        [localdata_que]
 *                 Meaning of a maximal threshold depends on
 *                 a queue mode of the given QoS queue.
 * @param[in,out]  p_que  Local data to be modified.
 * @param[in]      max    Maximal threshold.
 */
void demo_qos_que_ld_set_max(fpp_qos_queue_cmd_t* p_que, uint32_t max)
{
    assert(NULL != p_que);
    p_que->max = htonl(max);
}


/*
 * @brief          Set packet drop probability of a particular QoS queue's zone.
 * @details        [localdata_que]
 *                 Meaningful only for the que mode WRED.
 * @param[in,out]  p_que     Local data to be modified.
 * @param[in]      zprob_id  ID of a probability zone.
 *                           There may be less than 32 zones actually implemented in PFE.
 *                           (32 is just the max array limit)
 *                           See FCI API Reference, chapter Egress QoS.
 * @param[in]      percentage  Drop probability in [%].
 */
void demo_qos_que_ld_set_zprob(fpp_qos_queue_cmd_t* p_que, uint8_t zprob_id,
                               uint8_t percentage)
{
    assert(NULL != p_que);
    if (32u > zprob_id)
    {
        p_que->zprob[zprob_id] = percentage;
    }
}




/*
 * @defgroup    localdata_sch  [localdata_sch]
 * @brief:      Functions marked as [localdata_sch] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_sch (a struct with configuration data).
 *              Initial data for p_sch can be obtained via demo_qos_sch_get_by_id().
 *              If some modifications are made to local data, then after all modifications
 *              are done and finished, call demo_qos_sch_update() to update
 *              the configuration of a real QoS scheduler in PFE.
 */


/*
 * @brief          Set a mode of a QoS scheduler.
 * @details        [localdata_sch]
 * @param[in,out]  p_sch     Local data to be modified.
 * @param[in]      sch_mode  Scheduler mode.
 *                           For valid modes, see FCI API Reference,
 *                           chapter 'fpp_qos_scheduler_cmd_t'.
 */
void demo_qos_sch_ld_set_mode(fpp_qos_scheduler_cmd_t* p_sch, uint8_t sch_mode)
{
    assert(NULL != p_sch);
    p_sch->mode = sch_mode;
}
```

```
/*
 * @brief          Set a selection algorithm of a QoS scheduler.
 * @details        [localdata_sch]
 * @param[in,out]  p_sch  Local data to be modified.
 * @param[in]      algo   Selection algorithm.
 *                        For valid modes, see the FCI API Reference,
 *                        chapter 'fpp_qos_scheduler_cmd_t'.
 */
void demo_qos_sch_ld_set_algo(fpp_qos_scheduler_cmd_t* p_sch, uint8_t algo)
{
    assert(NULL != p_sch);
    p_sch->algo = algo;
}


/*
 * @brief          Set an input (and its properties) of a QoS scheduler.
 * @details        [localdata_sch]
 * @param[in,out]  p_sch     Local data to be modified.
 * @param[in]      input_id  ID of the scheduler's input.
 *                           There may be less than 32 inputs per scheduler
 *                           actually implemented in PFE. (32 is just the max array limit)
 *                           See FCI API Reference, chapter Egress QoS.
 *                 enable    Request to enable/disable the given scheduler input.
 *                 src       Data source which is connected to the given sscheduler input.
 *                           See FCI API Reference, chapter Egress QoS.
 *                 weight    Weight ("importance") of the given scheduler input.
 */
void demo_qos_sch_ld_set_input(fpp_qos_scheduler_cmd_t* p_sch, uint8_t input_id,
                               bool enable, uint8_t src, uint32_t weight)
{
    assert(NULL != p_sch);

    if (32u > input_id)
    {
        if (enable)
        {
            p_sch->input_en |= htonl(1uL << input_id);
        }
        else
        {
            p_sch->input_en &= htonl((uint32_t)(~(1uL << input_id)));
        }

        p_sch->input_w[input_id] = htonl(weight);
        p_sch->input_src[input_id] = src;
    }
}




/*
 * @defgroup    localdata_shp  [localdata_shp]
 * @brief:      Functions marked as [localdata_shp] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_shp (a struct with configuration data).
 *              Initial data for p_shp can be obtained via demo_qos_shp_get_by_id().
 *              If some modifications are made to local data, then after all modifications
 *              are done and finished, call demo_shp_sch_update() to update
 *              the configuration of a real QoS shaper in PFE.
 */


/*
 * @brief          Set a mode of a QoS shaper.
 * @details        [localdata_shp]
 * @param[in,out]  p_shp     Local data to be modified.
 * @param[in]      shp_mode  Shaper mode.
 *                           For valid modes, see FCI API Reference,
 *                           chapter 'fpp_qos_shaper_cmd_t'.
 */
void demo_qos_shp_ld_set_mode(fpp_qos_shaper_cmd_t* p_shp, uint8_t shp_mode)
{
    assert(NULL != p_shp);
    p_shp->mode = shp_mode;
}


/*
 * @brief          Set a position of a QoS shaper.
 * @details        [localdata_shp]
 * @param[in,out]  p_shp     Local data to be modified.
 * @param[in]      position  Position of the QoS shaper in a QoS configuration.
```



```
 *                           For valid positions, see FCI API Reference, chapter Egress QoS.
 */
void demo_qos_shp_ld_set_position(fpp_qos_shaper_cmd_t* p_shp, uint8_t position)
{
    assert(NULL != p_shp);
    p_shp->position = position;
}


/*
 * @brief          Set an idle slope rate of a QoS shaper.
 * @details        [localdata_shp]
 * @param[in,out]  p_shp  Local data to be modified.
 * @param[in]      isl    Idle slope rate (units per second).
 *                        Units depend on the mode of a QoS shaper.
 */
void demo_qos_shp_ld_set_isl(fpp_qos_shaper_cmd_t* p_shp, uint32_t isl)
{
    assert(NULL != p_shp);
    p_shp->isl = htonl(isl);
}


/*
 * @brief          Set a minimal credit of a QoS shaper.
 * @details        [localdata_shp]
 * @param[in,out]  p_shp       Local data to be modified.
 * @param[in]      min_credit  Minimal credit.
 */
void demo_qos_shp_ld_set_min_credit(fpp_qos_shaper_cmd_t* p_shp, int32_t min_credit)
{
    assert(NULL != p_shp);
    p_shp->min_credit = (int32_t)(htonl(min_credit));
}


/*
 * @brief          Set a maximal credit of a QoS shaper.
 * @details        [localdata_shp]
 * @param[in,out]  p_shp       Local data to be modified.
 * @param[in]      min_credit  Maximal credit.
 */
void demo_qos_shp_ld_set_max_credit(fpp_qos_shaper_cmd_t* p_shp, int32_t max_credit)
{
    assert(NULL != p_shp);
    p_shp->max_credit = (int32_t)(htonl(max_credit));
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */


/*
 * @brief      Query the name of a parent physical interface of a QoS queue.
 * @details    [localdata_que]
 * @param[in]  p_que  Local data to be queried.
 * @return     Name of a parent physical interface.
 */
const char* demo_qos_que_ld_get_if_name(const fpp_qos_queue_cmd_t* p_que)
{
    assert(p_que);
    return (p_que->if_name);
}


/*
 * @brief      Query the ID of a QoS queue.
 * @details    [localdata_que]
 * @param[in]  p_que  Local data to be queried.
 * @return     ID of a QoS queue.
 */
uint8_t demo_qos_que_ld_get_id(const fpp_qos_queue_cmd_t* p_que)
{
    assert(p_que);
    return (p_que->id);
}


/*
 * @brief      Query the mode of a QoS queue.
 * @details    [localdata_que]
 * @param[in]  p_que  Local data to be queried.
 * @return     Mode of a QoS queue.
 */
uint8_t demo_qos_que_ld_get_mode(const fpp_qos_queue_cmd_t* p_que)
{
```

```
    assert(p_que);
    return (p_que->mode);
}


/*
 * @brief      Query the minimal threshold of a QoS queue.
 * @details    [localdata_que]
 * @param[in]  p_que  Local data to be queried.
 * @return     Minimal threshold of a QoS queue.
 */
uint32_t demo_qos_que_ld_get_min(const fpp_qos_queue_cmd_t* p_que)
{
    assert(p_que);
    return ntohl(p_que->min);
}


/*
 * @brief      Query the maximal threshold of a QoS queue.
 * @details    [localdata_que]
 * @param[in]  p_que  Local data to be queried.
 * @return     Maximal threshold of a QoS queue.
 */
uint32_t demo_qos_que_ld_get_max(const fpp_qos_queue_cmd_t* p_que)
{
    assert(p_que);
    return ntohl(p_que->max);
}


/*
 * @brief      Query the percentage chance for packet drop.
 * @details    [localdata_que]
 * @param[in]  p_que     Local data to be queried.
 * @param[in]  zprob_id  ID of a probability zone.
 *                       There may be less than 32 zones actually implemented in PFE.
 *                       (32 is just the max array limit)
 *                       See FCI API Reference, chapter Egress QoS.
 * @return     Percentage drop chance of the given probability zone.
 */
uint8_t demo_qos_que_ld_get_zprob_by_id(const fpp_qos_queue_cmd_t* p_que, uint8_t zprob_id)
{
    assert(p_que);
    return ((32u > zprob_id) ? (p_que->zprob[zprob_id]) : (255u));
}




 /*
 * @brief      Query the name of a parent physical interface of a QoS scheduler.
 * @details    [localdata_sch]
 * @param[in]  p_sch  Local data to be queried.
 * @return     Name of a parent physical interface.
 */
const char* demo_qos_sch_ld_get_if_name(const fpp_qos_scheduler_cmd_t* p_sch)
{
    assert(p_sch);
    return (p_sch->if_name);
}


/*
 * @brief      Query the ID of a QoS scheduler.
 * @details    [localdata_sch]
 * @param[in]  p_sch  Local data to be queried.
 * @return     ID of a QoS scheduler.
 */
uint8_t demo_qos_sch_ld_get_id(const fpp_qos_scheduler_cmd_t* p_sch)
{
    assert(p_sch);
    return (p_sch->id);
}


/*
 * @brief      Query the mode of a QoS scheduler.
 * @details    [localdata_sch]
 * @param[in]  p_sch  Local data to be queried.
 * @return     Mode of a QoS scheduler.
 */
uint8_t demo_qos_sch_ld_get_mode(const fpp_qos_scheduler_cmd_t* p_sch)
{
    assert(p_sch);
    return (p_sch->mode);
```

```
}


/*
 * @brief      Query the selection algorithm of a QoS scheduler.
 * @details    [localdata_sch]
 * @param[in]  p_sch  Local data to be queried.
 * @return     Selection algorithm of a QoS scheduler.
 */
uint8_t demo_qos_sch_ld_get_algo(const fpp_qos_scheduler_cmd_t* p_sch)
{
    assert(p_sch);
    return (p_sch->algo);
}


/*
 * @brief      Query whether an input of a QoS scheduler is enabled or not.
 * @details    [localdata_sch]
 * @param[in]  p_sch     Local data to be queried.
 * @param[in]  input_id  Queried scheduler input.
 * @return     At time when the data was obtained from PFE, the input of the QoS scheduler:
 *             true  : was enabled
 *             false : was disabled
 */
bool demo_qos_sch_ld_is_input_enabled(const fpp_qos_scheduler_cmd_t* p_sch, uint8_t input_id)
{
    assert(NULL != p_sch);
    return (bool)((32u > input_id) ? (ntohl(p_sch->input_en) & (1uL << input_id)) : (0u));
}


/*
 * @brief      Query the weight of a QoS scheduler input.
 * @details    [localdata_sch]
 * @param[in]  p_sch     Local data to be queried.
 * @param[in]  input_id  Queried scheduler input.
 * @return     Weight of a QoS scheduler input.
 */
uint32_t demo_qos_sch_ld_get_input_weight(const fpp_qos_scheduler_cmd_t* p_sch,
                                          uint8_t input_id)
{
    assert(NULL != p_sch);
    return ((32u > input_id) ? (ntohl(p_sch->input_w[input_id])) : (0uL));
}


/*
 * @brief      Query the traffic source of a QoS scheduler input.
 * @details    [localdata_sch]
 * @param[in]  p_sch     Local data to be queried.
 * @param[in]  input_id  Queried scheduler input.
 * @return     Traffic source of a QoS scheduler input.
 */
uint8_t demo_qos_sch_ld_get_input_src(const fpp_qos_scheduler_cmd_t* p_sch, uint8_t input_id)
{
    assert(NULL != p_sch);
    return ((32u > input_id) ? (p_sch->input_src[input_id]) : (0uL));
}




 /*
 * @brief      Query the name of a parent physical interface of a QoS shaper.
 * @details    [localdata_shp]
 * @param[in]  p_shp  Local data to be queried.
 * @return     Name of a parent physical interface.
 */
const char* demo_qos_shp_ld_get_if_name(const fpp_qos_shaper_cmd_t* p_shp)
{
    assert(p_shp);
    return (p_shp->if_name);
}


/*
 * @brief      Query the ID of a QoS shaper.
 * @details    [localdata_shp]
 * @param[in]  p_shp  Local data to be queried.
 * @return     ID of a QoS shaper.
 */
uint8_t demo_qos_shp_ld_get_id(const fpp_qos_shaper_cmd_t* p_shp)
{
    assert(p_shp);
    return (p_shp->id);
```

```
}


 /*
 * @brief      Query the position of a QoS shaper.
 * @details    [localdata_shp]
 * @param[in]  p_shp  Local data to be queried.
 * @return     Position of a QoS shaper.
 */
uint8_t demo_qos_shp_ld_get_position(const fpp_qos_shaper_cmd_t* p_shp)
{
    assert(p_shp);
    return (p_shp->position);
}


/*
 * @brief      Query the mode of a QoS shaper.
 * @details    [localdata_shp]
 * @param[in]  p_shp  Local data to be queried.
 * @return     Mode of a QoS shaper.
 */
uint8_t demo_qos_shp_ld_get_mode(const fpp_qos_shaper_cmd_t* p_shp)
{
    assert(p_shp);
    return (p_shp->mode);
}


/*
 * @brief      Query the idle slope of a QoS shaper.
 * @details    [localdata_shp]
 * @param[in]  p_shp  Local data to be queried.
 * @return     Idle slope of a QoS shaper.
 */
uint32_t demo_qos_shp_ld_get_isl(const fpp_qos_shaper_cmd_t* p_shp)
{
    assert(p_shp);
    return ntohl(p_shp->isl);
}


/*
 * @brief      Query the maximal credit of a QoS shaper.
 * @details    [localdata_shp]
 * @param[in]  p_shp  Local data to be queried.
 * @return     Maximal credit of a QoS shaper.
 */
int32_t demo_qos_shp_ld_get_max_credit(const fpp_qos_shaper_cmd_t* p_shp)
{
    assert(p_shp);
    return (int32_t)(ntohl(p_shp->max_credit));
}


/*
 * @brief      Query the minimal credit of a QoS shaper.
 * @details    [localdata_shp]
 * @param[in]  p_shp  Local data to be queried.
 * @return     Minimal credit of a QoS shaper.
 */
int32_t demo_qos_shp_ld_get_min_credit(const fpp_qos_shaper_cmd_t* p_shp)
{
    assert(p_shp);
    return (int32_t)(ntohl(p_shp->min_credit));
}


/* ==== PUBLIC FUNCTIONS : misc =========================================== */


/*
 * @brief      Use FCI calls to iterate through all available QoS queues of
 *             a given physical interface and execute a callback print function for
 *             each QoS queue.
 * @param[in]  p_cl          FCI client
 * @param[in]  p_cb_print    Callback print function.
 *                           --> If the callback returns ZERO, then all is OK and
 *                               a next QoS queue is picked for a print process.
 *                           --> If the callback returns NON-ZERO, then some problem is
 *                               assumed and this function terminates prematurely.
 * @param[in]  p_phyif_name  Name of a parent physical interface.
 *                           Names of physical interfaces are hardcoded.
 *                           See FCI API Reference, chapter Interface Management.
 * @return     FPP_ERR_OK : Successfully iterated through all available QoS queues of
 *                          the given physical interface.
```



```
 *             other      : Some error occurred (represented by the respective error code).
 *                          No count was stored.
 */
int demo_qos_que_print_by_phyif(FCI_CLIENT* p_cl, demo_qos_que_cb_print_t p_cb_print,
                                const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_queue_cmd_t cmd_to_fci = {0};
    fpp_qos_queue_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* query loop */
        uint8_t que_id = 0u;
        while (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.id = que_id;
            cmd_to_fci.action = FPP_ACTION_QUERY;
            rtn = fci_query(p_cl, FPP_CMD_QOS_QUEUE,
                            sizeof(fpp_qos_queue_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));

            if (FPP_ERR_OK == rtn)
            {
                rtn = p_cb_print(&reply_from_fci);
            }

            que_id++;
        }

        /* query loop runs till there are no more QoS queues to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_QOS_QUEUE_NOT_FOUND == rtn)
        {
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_qos_que_print_by_phyif() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available QoS queues in PFE which
 *              are a part of a given parent physical interface.
 * @param[in]   p_cl          FCI client
 * @param[out]  p_rtn_count   Space to store the count of QoS queues.
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : Successfully counted all applicable QoS queues.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_qos_que_get_count_by_phyif(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                                    const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_queue_cmd_t cmd_to_fci = {0};
    fpp_qos_queue_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* query loop */
```

```
        uint8_t que_id = 0u;
        while (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.id = que_id;
            cmd_to_fci.action = FPP_ACTION_QUERY;
            rtn = fci_query(p_cl, FPP_CMD_QOS_QUEUE,
                            sizeof(fpp_qos_queue_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));

            if (FPP_ERR_OK == rtn)
            {
                que_id++;
            }
        }

        /* query loop runs till there are no more QoS queues to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_QOS_QUEUE_NOT_FOUND == rtn)
        {
            *p_rtn_count = que_id;
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_qos_que_get_count_by_phyif() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to iterate through all available QoS schedulers of
 *             a given physical interface and execute a callback print function for
 *             each QoS scheduler.
 * @param[in]  p_cl           FCI client
 * @param[in]  p_cb_print     Callback print function.
 *                            --> If the callback returns ZERO, then all is OK and
 *                                a next QoS scheduler is picked for a print process.
 *                            --> If the callback returns NON-ZERO, then some problem is
 *                                assumed and this function terminates prematurely.
 * @param[in]  p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @return     FPP_ERR_OK : Successfully iterated through QoS schedulers of
 *                           the given physical interface.
 *             other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_qos_sch_print_by_phyif(FCI_CLIENT* p_cl, demo_qos_sch_cb_print_t p_cb_print,
                                const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_scheduler_cmd_t cmd_to_fci = {0};
    fpp_qos_scheduler_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* query loop */
        uint8_t sch_id = 0u;
        while (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.id = sch_id;
            cmd_to_fci.action = FPP_ACTION_QUERY;
            rtn = fci_query(p_cl, FPP_CMD_QOS_SCHEDULER,
                            sizeof(fpp_qos_scheduler_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));

            if (FPP_ERR_OK == rtn)
            {
                rtn = p_cb_print(&reply_from_fci);
            }

            sch_id++;
        }

        /* query loop runs till there are no more QoS schedulers to report */
```

```
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_QOS_SCHEDULER_NOT_FOUND == rtn)
        {
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_qos_sch_print_by_phyif() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available QoS schedulers in PFE which
 *              are a part of a given parent physical interface.
 * @param[in]   p_cl           FCI client
 * @param[out]  p_rtn_count    Space to store the count of QoS schedulers.
 * @param[in]   p_phyif_name   Name of a parent physical interface.
 *                             Names of physical interfaces are hardcoded.
 *                             See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : Successfully counted all applicable QoS schedulers.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_qos_sch_get_count_by_phyif(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                                    const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_scheduler_cmd_t cmd_to_fci = {0};
    fpp_qos_scheduler_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* query loop */
        uint8_t sch_id = 0u;
        while (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.id = sch_id;
            cmd_to_fci.action = FPP_ACTION_QUERY;
            rtn = fci_query(p_cl, FPP_CMD_QOS_SCHEDULER,
                            sizeof(fpp_qos_scheduler_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));

            if (FPP_ERR_OK == rtn)
            {
                sch_id++;
            }
        }

        /* query loop runs till there are no more QoS schedulers to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_QOS_SCHEDULER_NOT_FOUND == rtn)
        {
            *p_rtn_count = sch_id;
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_qos_sch_get_count_by_phyif() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to iterate through all available QoS shapers of
 *             a given physical interface and execute a callback print function for
 *             each QoS shaper.
 * @param[in]  p_cl          FCI client
 * @param[in]  p_cb_print    Callback print function.
 *                           --> If the callback returns ZERO, then all is OK and
 *                               a next QoS shaper is picked for a print process.
 *                           --> If the callback returns NON-ZERO, then some problem is
 *                               assumed and this function terminates prematurely.
 * @param[in]  p_phyif_name  Name of a parent physical interface.
```



```
 *                           Names of physical interfaces are hardcoded.
 *                           See FCI API Reference, chapter Interface Management.
 * @return     FPP_ERR_OK : Successfully iterated through all available QoS shapers of
 *                          the given physical interface.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_qos_shp_print_by_phyif(FCI_CLIENT* p_cl, demo_qos_shp_cb_print_t p_cb_print,
                                const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_shaper_cmd_t cmd_to_fci = {0};
    fpp_qos_shaper_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* query loop */
        uint8_t shp_id = 0u;
        while (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.id = shp_id;
            cmd_to_fci.action = FPP_ACTION_QUERY;
            rtn = fci_query(p_cl, FPP_CMD_QOS_SHAPER,
                            sizeof(fpp_qos_shaper_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));

            if (FPP_ERR_OK == rtn)
            {
                rtn = p_cb_print(&reply_from_fci);
            }

            shp_id++;
        }

        /* query loop runs till there are no more QoS shapers to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_QOS_SHAPER_NOT_FOUND == rtn)
        {
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_qos_shp_print_by_phyif() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available QoS shapers in PFE which
 *              are a part of a given parent physical interface.
 * @param[in]   p_cl          FCI client
 * @param[out]  p_rtn_count   Space to store the count of QoS shapers.
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : Successfully counted all applicable QoS shapers.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_qos_shp_get_count_by_phyif(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                                    const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_shaper_cmd_t cmd_to_fci = {0};
    fpp_qos_shaper_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
```



```
    if (FPP_ERR_OK == rtn)
    {
        /* query loop */
        uint8_t shp_id = 0u;
        while (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.id = shp_id;
            cmd_to_fci.action = FPP_ACTION_QUERY;
            rtn = fci_query(p_cl, FPP_CMD_QOS_SHAPER,
                            sizeof(fpp_qos_shaper_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));

            if (FPP_ERR_OK == rtn)
            {
                shp_id++;
            }
        }

        /* query loop runs till there are no more QoS shapers to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_QOS_SHAPER_NOT_FOUND == rtn)
        {
            *p_rtn_count = shp_id;
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_qos_shp_get_count_by_phyif() failed!");

    return (rtn);
}


/* ========================================================================= */
```

15.21 demo_qos_pol.c

```
/* =========================================================================
 *  Copyright 2020-2021 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_qos_pol.h"
```

```
/* ==== PRIVATE FUNCTIONS ================================================== */


/*
 * @brief       Set/unset a flow type flag (from argumentless set) in a policer flow struct.
 * @param[out]  p_rtn_polflow  Struct to be modified.
 * @param[in]   enable         New state of a flag.
 * @param[in]   flag           The flag.
 */
static void set_polflow_m_flag(fpp_qos_policer_flow_cmd_t* p_rtn_polflow, bool enable,
                               fpp_iqos_flow_type_t flag)
{
    assert(NULL != p_rtn_polflow);

    hton_enum(&flag, sizeof(fpp_iqos_flow_type_t));

    if (enable)
    {
        p_rtn_polflow->flow.type_mask |= flag;
    }
    else
    {
        p_rtn_polflow->flow.type_mask &= (fpp_iqos_flow_type_t)(~flag);
    }
}


/*
 * @brief       Set/unset a flow type flag (from argumentful set) in a policer flow struct.
 * @param[out]  p_rtn_polflow  Struct to be modified.
 * @param[in]   enable         New state of a flag.
 * @param[in]   flag           The flag.
 */
static void set_polflow_am_flag(fpp_qos_policer_flow_cmd_t* p_rtn_polflow, bool enable,
                                fpp_iqos_flow_arg_type_t flag)
{
    assert(NULL != p_rtn_polflow);

    hton_enum(&flag, sizeof(fpp_iqos_flow_arg_type_t));

    if (enable)
    {
        p_rtn_polflow->flow.arg_type_mask |= flag;
    }
    else
    {
        p_rtn_polflow->flow.arg_type_mask &= (fpp_iqos_flow_arg_type_t)(~flag);
    }
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from PFE ============== */


/*
 * @brief       Use FCI calls to get configuration data of a requested Ingress QoS policer
 *              from PFE.
 * @param[in]   p_cl           FCI client
 * @param[out]  p_rtn_pol      Space for data from PFE.
 * @param[in]   p_phyif_name   Name of a parent physical interface.
 *                             Names of physical interfaces are hardcoded.
 *                             See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : The requested Ingress QoS policer was found.
 *                           A copy of its configuration data was stored into p_rtn_pol.
 *                           REMINDER: Data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_pol_get(FCI_CLIENT* p_cl, fpp_qos_policer_cmd_t* p_rtn_pol, const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_pol);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_cmd_t cmd_to_fci = {0};
    fpp_qos_policer_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query (get the Ingress QoS policer directly; no need for a loop) */
```

```
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER,
                        sizeof(fpp_qos_policer_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_pol = reply_from_fci;
    }

    print_if_error(rtn, "demo_pol_get() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get configuration data of a requested Ingress QoS wred
 *              from PFE. Identify the Ingress QoS wred by the name of a parent
 *              physical interface and by the associated wred queue.
 * @param[in]   p_cl            FCI client
 * @param[out]  p_rtn_polwred  Space for data from PFE.
 * @param[in]   p_phyif_name   Name of a parent physical interface.
 *                              Names of physical interfaces are hardcoded.
 *                              See FCI API Reference, chapter Interface Management.
 *              polwred_que    Associated queue.
 * @return      FPP_ERR_OK : The requested Ingress QoS wred was found.
 *                            A copy of its configuration data was stored into p_rtn_polwred.
 *                            REMINDER: Data from PFE are in a network byte order.
 *              other      : Some error occurred (represented by the respective error code).
 *                            No data copied.
 */
int demo_polwred_get_by_que(FCI_CLIENT* p_cl, fpp_qos_policer_wred_cmd_t* p_rtn_polwred,
                            const char* p_phyif_name, fpp_iqos_queue_t polwred_que)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_polwred);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_wred_cmd_t cmd_to_fci = {0};
    fpp_qos_policer_wred_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    cmd_to_fci.queue = polwred_que;
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query (get the Ingress QoS shaper directly; no need for a loop) */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_WRED,
                        sizeof(fpp_qos_policer_wred_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_polwred = reply_from_fci;
    }

    print_if_error(rtn, "demo_polwred_get_by_que() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get configuration data of a requested Ingress QoS shaper
 *              from PFE. Identify the Ingress QoS shaper by the name of a parent
 *              physical interface and by the shaper's ID.
 * @param[in]   p_cl            FCI client
 * @param[out]  p_rtn_polshp   Space for data from PFE.
 * @param[in]   p_phyif_name   Name of a parent physical interface.
 *                              Names of physical interfaces are hardcoded.
 *                              See FCI API Reference, chapter Interface Management.
 *              polshp_id      ID of the requested Ingress QoS shaper.
 * @return      FPP_ERR_OK : The requested Ingress QoS shaper was found.
 *                            A copy of its configuration data was stored into p_rtn_polshp.
```

```
 *                            REMINDER: Data from PFE are in a network byte order.
 *              other       : Some error occurred (represented by the respective error code).
 *                            No data copied.
 */
int demo_polshp_get_by_id(FCI_CLIENT* p_cl, fpp_qos_policer_shp_cmd_t* p_rtn_polshp,
                          const char* p_phyif_name, uint8_t polshp_id)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_polshp);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_shp_cmd_t cmd_to_fci = {0};
    fpp_qos_policer_shp_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    cmd_to_fci.id = polshp_id;
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query (get the Ingress QoS shaper directly; no need for a loop) */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_SHP,
                        sizeof(fpp_qos_policer_shp_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_polshp = reply_from_fci;
    }

    print_if_error(rtn, "demo_polshp_get_by_id() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get configuration data of a requested Ingress QoS flow
 *              from PFE. Identify the Ingress QoS flow by the name of a parent
 *              physical interface and by the flow's ID.
 * @param[in]   p_cl           FCI client
 * @param[out]  p_rtn_polflow  Space for data from PFE.
 * @param[in]   p_phyif_name   Name of a parent physical interface.
 *                             Names of physical interfaces are hardcoded.
 *                             See FCI API Reference, chapter Interface Management.
 *              polflow_id     ID of the requested Ingress QoS flow.
 * @return      FPP_ERR_OK : The requested Ingress QoS flow was found.
 *                            A copy of its configuration data was stored into p_rtn_polflow.
 *                            REMINDER: Data from PFE are in a network byte order.
 *              other       : Some error occurred (represented by the respective error code).
 *                            No data copied.
 */
int demo_polflow_get_by_id(FCI_CLIENT* p_cl, fpp_qos_policer_flow_cmd_t* p_rtn_polflow,
                           const char* p_phyif_name, uint8_t polflow_id)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_polflow);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_flow_cmd_t cmd_to_fci = {0};
    fpp_qos_policer_flow_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    cmd_to_fci.id = polflow_id;
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_FLOW,
                        sizeof(fpp_qos_policer_flow_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

        /* query loop (with a search condition) */
        while ((FPP_ERR_OK == rtn) && (reply_from_fci.id != polflow_id))
```

```
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_FLOW,
                           sizeof(fpp_qos_policer_flow_cmd_t), (unsigned short*)(&cmd_to_fci),
                           &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_polflow = reply_from_fci;
    }

    print_if_error(rtn, "demo_polflow_get_by_id() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to update data in PFE ============= */


/*
 * @brief       Use FCI calls to enable/disable Ingress QoS block of a physical interface.
 * @param[in]   p_cl  FCI client
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                             Names of physical interfaces are hardcoded.
 *                             See FCI API Reference, chapter Interface Management.
 * @param[in]   enable        Enable/disable Ingress QoS block of a physical interface.
 * @return      FPP_ERR_OK : Static MAC table entries of all bridge domains were
 *                            successfully flushed in PFE.
 *              other      : Some error occurred (represented by the respective error code).
 */
int demo_pol_enable(FCI_CLIENT* p_cl, const char* p_phyif_name, bool enable)
{
    assert(NULL != p_cl);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci.enable = enable;
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_UPDATE;
        rtn = fci_write(p_cl, FPP_CMD_QOS_POLICER, sizeof(fpp_qos_policer_cmd_t),
                                                  (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_pol_enable() failed!");

    return (rtn);
}


/*
 * @brief          Use FCI calls to update configuration of a target Ingress QoS wred
 *                 in PFE.
 * @param[in]      p_cl   FCI client
 * @param[in,out]  p_polwred  Local data struct which represents a new configuration of
 *                             the target Ingress QoS wred.
 *                             Initial data can be obtained via demo_polwred_get_by_que().
 * @return        FPP_ERR_OK : Configuration of the target Ingress QoS wred was
 *                              successfully updated in PFE.
 *                              The local data struct was automatically updated with
 *                              readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                              The local data struct not updated.
 */
int demo_polwred_update(FCI_CLIENT* p_cl, fpp_qos_policer_wred_cmd_t* p_polwred)
{
    assert(NULL != p_cl);
    assert(NULL != p_polwred);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_qos_policer_wred_cmd_t cmd_to_fci = (*p_polwred);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_QOS_POLICER_WRED, sizeof(fpp_qos_policer_wred_cmd_t),
```

```
                                              (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_polwred_get_by_que(p_cl, p_polwred, (p_polwred->if_name),
                                                        (p_polwred->queue));
    }

    print_if_error(rtn, "demo_polwred_update() failed!");

    return (rtn);
}


/*
 * @brief          Use FCI calls to update configuration of a target Ingress QoS shaper
 *                 in PFE.
 * @param[in]      p_cl   FCI client
 * @param[in,out]  p_polshp   Local data struct which represents a new configuration of
 *                            the target Ingress QoS shaper.
 *                            Initial data can be obtained via demo_polshp_get_by_id().
 * @return        FPP_ERR_OK : Configuration of the target Ingress QoS shaper was
 *                             successfully updated in PFE.
 *                             The local data struct was automatically updated with
 *                             readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                             The local data struct not updated.
 */
int demo_polshp_update(FCI_CLIENT* p_cl, fpp_qos_policer_shp_cmd_t* p_polshp)
{
    assert(NULL != p_cl);
    assert(NULL != p_polshp);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_qos_policer_shp_cmd_t cmd_to_fci = (*p_polshp);

    /* send data */
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_QOS_POLICER_SHP, sizeof(fpp_qos_policer_shp_cmd_t),
                                             (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_polshp_get_by_id(p_cl, p_polshp, (p_polshp->if_name), (p_polshp->id));
    }

    print_if_error(rtn, "demo_polwred_update() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to add/del items in PFE =========== */


/*
 * @brief       Use FCI calls to create a new Ingress QoS flow for a target physical interface
 *              in PFE.
 * @param[in]   p_cl           FCI client
 * @param[out]  p_rtn_polflow  Space for data from PFE.
 *                             This will contain a copy of configuration data of
 *                             the newly created Ingress QoS flow.
 *                             Can be NULL. If NULL, then there is no local data to fill.
 * @param[in]   p_phyif_name   Name of a parent physical interface.
 *                             Names of physical interfaces are hardcoded.
 *                             See FCI API Reference, chapter Interface Management.
 * @param[in]   polflow_id     ID of the requested Ingress QoS flow.
 * @param[in]   p_polflow_data Configuration data of the new Ingress QoS flow.
 *                             To create a new flow, a local data struct must be created,
 *                             configured and then passed to this function.
 *                             See [localdata_polflow] to learn more.
 * @return      FPP_ERR_OK : New Ingress QoS flow was created.
 *                           If applicable, then its configuration data were
 *                           copied into p_rtn_polflow.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_polflow_add(FCI_CLIENT* p_cl, const char* p_phyif_name, uint8_t polflow_id,
                     fpp_qos_policer_flow_cmd_t* p_polflow_data)
{
    assert(NULL != p_cl);
    assert(NULL != p_phyif_name);
    /* 'p_rtn_polflow' is allowed to be NULL */
```

```
    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_qos_policer_flow_cmd_t cmd_to_fci = (*p_polflow_data);

    /* prepare data */
    cmd_to_fci.id = polflow_id;
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_REGISTER;
        rtn = fci_write(p_cl, FPP_CMD_QOS_POLICER_FLOW, sizeof(fpp_qos_policer_flow_cmd_t),
                                                        (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_polflow_add() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to destroy the target Ingress QoS flow in PFE.
 * @param[in]  p_cl    FCI client
 * @param[in]  p_phyif_name   Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 *             polflow_id     ID of the Ingress QoS flow to destroy.
 * @return     FPP_ERR_OK : The Ingress QoS flow was destroyed.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_polflow_del(FCI_CLIENT* p_cl, const char* p_phyif_name, uint8_t polflow_id)
{
    assert(NULL != p_cl);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_qos_policer_flow_cmd_t cmd_to_fci = {0};

    /* prepare data */
    cmd_to_fci.id = polflow_id;
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_DEREGISTER;
        rtn = fci_write(p_cl, FPP_CMD_QOS_POLICER_FLOW, sizeof(fpp_qos_policer_flow_cmd_t),
                                                        (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_polflow_del() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : modify local data (no FCI calls) ================ */
/*
 * @defgroup    localdata_polflow  [localdata_polflow]
 * @brief:      Functions marked as [localdata_polflow] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_polflow (a struct with configuration data).
 *              Initial data for p_polflow can be obtained via demo_polflow_get_by_id().
 *              If some modifications are made to local data, then after all modifications
 *              are done and finished, call demo_polflow_update() to update
 *              the configuration of a real Ingress QoS flow in PFE.
 */


/*
 * @brief          Clear all argumentless flow types of an Ingress QoS flow.
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      action     Requested action for Ingress QoS flow.
 */
void demo_polflow_ld_set_action(fpp_qos_policer_flow_cmd_t* p_polflow,
                                fpp_iqos_flow_action_t action)
{
    assert(NULL != p_polflow);
    p_polflow->flow.action = action;
}


/*
 * @brief          Clear all argumentless flow types of an Ingress QoS flow.
```

```
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 */
void demo_polflow_ld_clear_m(fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    p_polflow->flow.type_mask = 0u;
}


/*
 * @brief          Set/unset the given argumentless flow type (TYPE_ETH).
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow tpye.
 */
void demo_polflow_ld_set_m_type_eth(fpp_qos_policer_flow_cmd_t* p_polflow, bool set)
{
    assert(NULL != p_polflow);
    set_polflow_m_flag(p_polflow, set, FPP_IQOS_FLOW_TYPE_ETH);
}


/*
 * @brief          Set/unset the given argumentless flow type (TYPE_PPPOE).
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow tpye.
 */
void demo_polflow_ld_set_m_type_pppoe(fpp_qos_policer_flow_cmd_t* p_polflow, bool set)
{
    assert(NULL != p_polflow);
    set_polflow_m_flag(p_polflow, set, FPP_IQOS_FLOW_TYPE_PPPOE);
}


/*
 * @brief          Set/unset the given argumentless flow type (TYPE_ARP).
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow tpye.
 */
void demo_polflow_ld_set_m_type_arp(fpp_qos_policer_flow_cmd_t* p_polflow, bool set)
{
    assert(NULL != p_polflow);
    set_polflow_m_flag(p_polflow, set, FPP_IQOS_FLOW_TYPE_ARP);
}


/*
 * @brief          Set/unset the given argumentless flow type (TYPE_IP4).
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow tpye.
 */
void demo_polflow_ld_set_m_type_ip4(fpp_qos_policer_flow_cmd_t* p_polflow, bool set)
{
    assert(NULL != p_polflow);
    set_polflow_m_flag(p_polflow, set, FPP_IQOS_FLOW_TYPE_IPV4);
}


/*
 * @brief          Set/unset the given argumentless flow type (TYPE_IP6).
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow tpye.
 */
void demo_polflow_ld_set_m_type_ip6(fpp_qos_policer_flow_cmd_t* p_polflow, bool set)
{
    assert(NULL != p_polflow);
    set_polflow_m_flag(p_polflow, set, FPP_IQOS_FLOW_TYPE_IPV6);
}


/*
 * @brief          Set/unset the given argumentless flow type (TYPE_IPX).
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow tpye.
 */
void demo_polflow_ld_set_m_type_ipx(fpp_qos_policer_flow_cmd_t* p_polflow, bool set)
{
    assert(NULL != p_polflow);
    set_polflow_m_flag(p_polflow, set, FPP_IQOS_FLOW_TYPE_IPX);
}
```

```
/*
 * @brief          Set/unset the given argumentless flow type (TYPE_MCAST).
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow tpye.
 */
void demo_polflow_ld_set_m_type_mcast(fpp_qos_policer_flow_cmd_t* p_polflow, bool set)
{
    assert(NULL != p_polflow);
    set_polflow_m_flag(p_polflow, set, FPP_IQOS_FLOW_TYPE_MCAST);
}


/*
 * @brief          Set/unset the given argumentless flow type (TYPE_BCAST).
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow tpye.
 */
void demo_polflow_ld_set_m_type_bcast(fpp_qos_policer_flow_cmd_t* p_polflow, bool set)
{
    assert(NULL != p_polflow);
    set_polflow_m_flag(p_polflow, set, FPP_IQOS_FLOW_TYPE_BCAST);
}


/*
 * @brief          Set/unset the given argumentless flow type (TYPE_VLAN).
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow tpye.
 */
void demo_polflow_ld_set_m_type_vlan(fpp_qos_policer_flow_cmd_t* p_polflow, bool set)
{
    assert(NULL != p_polflow);
    set_polflow_m_flag(p_polflow, set, FPP_IQOS_FLOW_TYPE_VLAN);
}




/*
 * @brief          Clear all argumentful flow types of an Ingress QoS flow.
 *                 (also zeroify all associated flow type arguments)
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 */
void demo_polflow_ld_clear_am(fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    p_polflow->flow.arg_type_mask = 0u;
    memset(&(p_polflow->flow.args), 0, sizeof(fpp_iqos_flow_args_t));
}


/*
 * @brief          Set/unset the given argumentful flow type (VLAN) and its argument.
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow type.
 * @param[in]      vlan       New VLAN ID for this flow type.
 *                            When this flow type is active, it compares value of its
 *                            'vlan' argument with the value of traffic's 'VID' field.
 *                            Comparison is bitmasked by value from vlan_m argument.
 * @param[in]      vlan_m     New bitmask for VLAN ID.
 */
void demo_polflow_ld_set_am_vlan(fpp_qos_policer_flow_cmd_t* p_polflow, bool set,
                                 uint16_t vlan, uint16_t vlan_m)
{
    assert(NULL != p_polflow);

    set_polflow_am_flag(p_polflow, set, FPP_IQOS_ARG_VLAN);
    p_polflow->flow.args.vlan = htons(vlan);
    p_polflow->flow.args.vlan_m = htons(vlan_m);
}


/*
 * @brief          Set/unset the given argumentful flow type (TOS) and its argument.
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow type.
 * @param[in]      tos        New TOS/TCLASS value for this flow type to match.
 *                            When this flow type is active, it compares value of its
```

```
 *                              'tos' argument with the value of traffic's 'TOS' field.
 *                              Comparison is bitmasked by value from tos_m argument.
 * @param[in]      tos_m      New bitmask for TOS/TCLASS.
 */
void demo_polflow_ld_set_am_tos(fpp_qos_policer_flow_cmd_t* p_polflow, bool set,
                                uint8_t tos, uint8_t tos_m)
{
    assert(NULL != p_polflow);

    set_polflow_am_flag(p_polflow, set, FPP_IQOS_ARG_TOS);
    p_polflow->flow.args.tos = tos;
    p_polflow->flow.args.tos_m = tos_m;
}


/*
 * @brief          Set/unset the given argumentful flow type (L4PROTO) and its argument.
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow type.
 * @param[in]      tos        New PROTOCOL value for this flow type to match.
 *                            When this flow type is active, it compares value of its
 *                            'l4proto' argument with the value of traffic's 'Protocol' field.
 *                            Comparison is bitmasked by value from l4proto_m argument.
 * @param[in]      tos_m      New bitmask for PROTOCOL.
 */
void demo_polflow_ld_set_am_proto(fpp_qos_policer_flow_cmd_t* p_polflow, bool set,
                                  uint8_t proto, uint8_t proto_m)
{
    assert(NULL != p_polflow);

    set_polflow_am_flag(p_polflow, set, FPP_IQOS_ARG_L4PROTO);
    p_polflow->flow.args.l4proto = proto;
    p_polflow->flow.args.l4proto_m = proto_m;
}


/*
 * @brief          Set/unset the given argumentful flow type (SIP) and its argument.
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow type.
 * @param[in]      sip        New source IP address for this flow type to match.
 *                            When this flow type is active, it compares value of its
 *                            'sip' argument with the value of traffic's
 *                            'source address'.
 *                            Comparison is bitmasked by source address subnet prefix.
 * @param[in]      sip_m      New subnet prefix for source IP address.
 */
void demo_polflow_ld_set_am_sip(fpp_qos_policer_flow_cmd_t* p_polflow, bool set,
                                uint32_t sip, uint8_t sip_m)
{
    assert(NULL != p_polflow);

    set_polflow_am_flag(p_polflow, set, FPP_IQOS_ARG_SIP);
    p_polflow->flow.args.sip = htonl(sip);
    p_polflow->flow.args.sip_m = sip_m;
}


/*
 * @brief          Set/unset the given argumentful flow type (DIP) and its argument.
 * @details        [localdata_polflow]
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow type.
 * @param[in]      dip        New destination IP address for this flow type to match.
 *                            When this flow type is active, it compares value of its
 *                            'dip' argument with the value of traffic's
 *                            'destination address'.
 *                            Comparison is bitmasked by destination address subnet prefix.
 * @param[in]      dip_m      New subnet prefix for destination IP address.
 */
void demo_polflow_ld_set_am_dip(fpp_qos_policer_flow_cmd_t* p_polflow, bool set,
                                uint32_t dip, uint8_t dip_m)
{
    assert(NULL != p_polflow);

    set_polflow_am_flag(p_polflow, set, FPP_IQOS_ARG_DIP);
    p_polflow->flow.args.dip = htonl(dip);
    p_polflow->flow.args.dip_m = dip_m;
}


/*
 * @brief          Set/unset the given argumentful flow type (DIP) and its argument.
 * @details        [localdata_polflow]
```

```
 *                  When this flow type is active, it compares traffic's 'source port'
 *                  with a defined range of source ports (from min to max).
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow type.
 * @param[in]      sport_min  New range of source ports - minimal port.
 * @param[in]      sport_max  New range of source ports - maximal port.
 */
void demo_polflow_ld_set_am_sport(fpp_qos_policer_flow_cmd_t* p_polflow, bool set,
                                  uint16_t sport_min, uint16_t sport_max)
{
    assert(NULL != p_polflow);

    set_polflow_am_flag(p_polflow, set, FPP_IQOS_ARG_SPORT);
    p_polflow->flow.args.sport_min = htons(sport_min);
    p_polflow->flow.args.sport_max = htons(sport_max);
}


/*
 * @brief          Set/unset the given argumentful flow type (DIP) and its argument.
 * @details        [localdata_polflow]
 *                  When this flow type is active, it compares traffic's 'destination port'
 *                  with a defined range of destination ports (from min to max).
 * @param[in,out]  p_polflow  Local data to be modified.
 * @param[in]      set        Request to set/unset the given flow type.
 * @param[in]      dport_min  New range of destination ports - minimal port.
 * @param[in]      dport_max  New range of destination ports - maximal port.
 */
void demo_polflow_ld_set_am_dport(fpp_qos_policer_flow_cmd_t* p_polflow, bool set,
                                  uint16_t dport_min, uint16_t dport_max)
{
    assert(NULL != p_polflow);

    set_polflow_am_flag(p_polflow, set, FPP_IQOS_ARG_DPORT);
    p_polflow->flow.args.dport_min = htons(dport_min);
    p_polflow->flow.args.dport_max = htons(dport_max);
}




/*
 * @defgroup    localdata_wred  [localdata_polwred]
 * @brief:      Functions marked as [localdata_polwred] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_polwred (a struct with configuration data).
 *              Initial data for p_polwred can be obtained via demo_polwred_get_by_que().
 *              If some modifications are made to local data, then after all modifications
 *              are done and finished, call demo_polwred_update() to update
 *              the configuration of a real Ingress QoS wred in PFE.
 */


/*
 * @brief          Enable/disable Ingress QoS wred.
 * @details        [localdata_polwred]
 * @param[in,out]  p_polwred  Local data to be modified.
 * @param[in]      enable     Enable/disable Ingress QoS wred.
 */
void demo_polwred_ld_enable(fpp_qos_policer_wred_cmd_t* p_polwred, bool enable)
{
    assert(NULL != p_polwred);
    p_polwred->enable = enable;
}


/*
 * @brief          Set a minimal threshold of Ingress QoS wred.
 * @details        [localdata_polwred]
 * @param[in,out]  p_polwred  Local data to be modified.
 * @param[in]      min        Minimal threshold.
 */
void demo_polwred_ld_set_min(fpp_qos_policer_wred_cmd_t* p_polwred, uint16_t min)
{
    assert(NULL != p_polwred);
    p_polwred->thr[FPP_IQOS_WRED_MIN_THR] = htons(min);
}


/*
 * @brief          Set a maximal threshold of Ingress QoS wred.
 * @details        [localdata_polwred]
 * @param[in,out]  p_polwred  Local data to be modified.
 * @param[in]      max        Maximal threshold.
 */
void demo_polwred_ld_set_max(fpp_qos_policer_wred_cmd_t* p_polwred, uint16_t max)
```

```
{
    assert(NULL != p_polwred);
    p_polwred->thr[FPP_IQOS_WRED_MAX_THR] = htons(max);
}


/*
 * @brief          Set a queue length ('full' threshold) of Ingress QoS wred.
 * @details        [localdata_polwred]
 * @param[in,out]  p_polwred  Local data to be modified.
 * @param[in]      full       Maximal threshold.
 */
void demo_polwred_ld_set_full(fpp_qos_policer_wred_cmd_t* p_polwred, uint16_t full)
{
    assert(NULL != p_polwred);
    p_polwred->thr[FPP_IQOS_WRED_FULL_THR] = htons(full);
}


/*
 * @brief          Set packet drop probability of a particular Ingress QoS wred's zone.
 * @details        [localdata_polwred]
 * @param[in,out]  p_polwred   Local data to be modified.
 * @param[in]      zprob_id    ID of a probability zone.
 * @param[in]      percentage  Drop probability in [%].
 */
void demo_polwred_ld_set_zprob(fpp_qos_policer_wred_cmd_t* p_polwred, uint8_t zprob_id,
                               uint8_t percentage)
{
    assert(NULL != p_polwred);
    if (FPP_IQOS_WRED_ZONES_COUNT > zprob_id)
    {
        /* FCI command for Ingress QoS wred expects drop probability in compressed format */
        const uint8_t compressed = (uint8_t)((percentage * 0x0Fu) / 100u);

        p_polwred->zprob[zprob_id] = compressed;
    }
}




/*
 * @defgroup    localdata_polshp  [localdata_polshp]
 * @brief:      Functions marked as [localdata_polshp] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_polshp (a struct with configuration data).
 *              Initial data for p_polshp can be obtained via demo_polshp_get_by_id().
 *              If some modifications are made to local data, then after all modifications
 *              are done and finished, call demo_polshp_update() to update
 *              the configuration of a real Ingress QoS shaper in PFE.
 */


/*
 * @brief          Enable/disable Ingress QoS shaper.
 * @details        [localdata_polshp]
 * @param[in,out]  p_polshp  Local data to be modified.
 * @param[in]      enable    Enable/disable Ingress QoS shaper.
 */
void demo_polshp_ld_enable(fpp_qos_policer_shp_cmd_t* p_polshp, bool enable)
{
    assert(NULL != p_polshp);
    p_polshp->enable = enable;
}


/*
 * @brief          Set a type of Ingress QoS shaper.
 * @details        [localdata_polshp]
 * @param[in,out]  p_polshp  Local data to be modified.
 * @param[in]      shp_type  Shaper type.
 */
void demo_polshp_ld_set_type(fpp_qos_policer_shp_cmd_t* p_polshp,
                             fpp_iqos_shp_type_t shp_type)
{
    assert(NULL != p_polshp);
    p_polshp->type = shp_type;
}


/*
 * @brief          Set a mode of Ingress QoS shaper.
 * @details        [localdata_polshp]
 * @param[in,out]  p_polshp  Local data to be modified.
 * @param[in]      shp_mode  Shaper mode.
```

```
 */
void demo_polshp_ld_set_mode(fpp_qos_policer_shp_cmd_t* p_polshp,
                             fpp_iqos_shp_rate_mode_t shp_mode)
{
    assert(NULL != p_polshp);
    p_polshp->mode = shp_mode;
}


/*
 * @brief          Set an idle slope rate of Ingress QoS shaper.
 * @details        [localdata_polshp]
 * @param[in,out]  p_polshp  Local data to be modified.
 * @param[in]      isl       Idle slope rate (units per second).
 *                           Units depend on the mode of a QoS shaper.
 */
void demo_polshp_ld_set_isl(fpp_qos_policer_shp_cmd_t* p_polshp, uint32_t isl)
{
    assert(NULL != p_polshp);
    p_polshp->isl = htonl(isl);
}


/*
 * @brief          Set a minimal credit of  Ingress QoS shaper.
 * @details        [localdata_polshp]
 * @param[in,out]  p_polshp    Local data to be modified.
 * @param[in]      min_credit  Minimal credit.
 */
void demo_polshp_ld_set_min_credit(fpp_qos_policer_shp_cmd_t* p_polshp, int32_t min_credit)
{
    assert(NULL != p_polshp);
    p_polshp->min_credit = (int32_t)(htonl(min_credit));
}


/*
 * @brief          Set a maximal credit of Ingress QoS shaper.
 * @details        [localdata_polshp]
 * @param[in,out]  p_polshp    Local data to be modified.
 * @param[in]      max_credit  Maximal credit.
 */
void demo_polshp_ld_set_max_credit(fpp_qos_policer_shp_cmd_t* p_polshp, int32_t max_credit)
{
    assert(NULL != p_polshp);
    p_polshp->max_credit = (int32_t)(htonl(max_credit));
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */


/*
 * @brief      Query the name of a parent physical interface of Ingress QoS policer.
 * @details    p_pol  Local data of Ingress QoS policer.
 * @return     Name of a parent physical interface.
 */
const char* demo_pol_ld_get_if_name(const fpp_qos_policer_cmd_t* p_pol)
{
    assert(p_pol);
    return (p_pol->if_name);
}


/*
 * @brief      Query the status of Ingress QoS policer "enable" flag.
 * @details    p_pol  Local data of Ingress QoS policer.
 * @return     At time when the data was obtained from PFE, the Ingress QoS policer:
 *             true  : was enabled
 *             false : was disabled
 */
bool demo_pol_ld_is_enabled(const fpp_qos_policer_cmd_t* p_pol)
{
    assert(p_pol);
    return (p_pol->enable);
}




/*
 * @brief      Query the name of a parent physical interface of Ingress QoS flow.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Name of a parent physical interface.
 */
```

```
const char* demo_polflow_ld_get_if_name(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(p_polflow);
    return (p_polflow->if_name);
}


/*
 * @brief      Query the ID of Ingress QoS flow.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     ID of Ingress QoS flow.
 */
uint8_t demo_polflow_ld_get_id(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(p_polflow);
    return (p_polflow->id);
}


/*
 * @brief      Query the action of Ingress QoS flow.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Action
 */
fpp_iqos_flow_action_t demo_polflow_ld_get_action(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(p_polflow);
    return (p_polflow->flow.action);
}

/*
 * @brief      Query the argumentless flow types bitset of Ingress QoS flow.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Bitset of argumentless flow types.
 */
fpp_iqos_flow_type_t demo_polflow_ld_get_m_bitset(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);

    fpp_iqos_flow_type_t type_mask = (p_polflow->flow.type_mask);
    ntoh_enum(&type_mask, sizeof(fpp_iqos_flow_type_t));

    return (type_mask);
}


/*
 * @brief      Query the argumentful flow types bitset of Ingress QoS flow.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Bitset of argumentful flow types.
 */
fpp_iqos_flow_arg_type_t demo_polflow_ld_get_am_bitset(
    const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);

    fpp_iqos_flow_arg_type_t arg_type_mask = (p_polflow->flow.arg_type_mask);
    ntoh_enum(&arg_type_mask, sizeof(fpp_iqos_flow_arg_type_t));

    return (arg_type_mask);
}


/*
 * @brief      Query the argument of the argumentful flow type VLAN.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Argument (VLAN ID) of the given flow type.
 */
uint16_t demo_polflow_ld_get_am_vlan(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return ntohs(p_polflow->flow.args.vlan);
}


/*
 * @brief      Query the bitmask of the argumentful flow type VLAN.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Bitmask for VLAN ID.
 */
```

```
uint16_t demo_polflow_ld_get_am_vlan_m(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return ntohs(p_polflow->flow.args.vlan_m);
}


/*
 * @brief      Query the argument of the argumentful flow type TOS.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Argument (TOS) of the given flow type.
 */
uint8_t  demo_polflow_ld_get_am_tos(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return (p_polflow->flow.args.tos);
}


/*
 * @brief      Query the bitmask of the argumentful flow type TOS.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Bitmask for TOS.
 */
uint8_t demo_polflow_ld_get_am_tos_m(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return (p_polflow->flow.args.tos_m);
}


/*
 * @brief      Query the argument of the argumentful flow type PROTOCOL.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Argument (Protocol ID) of the given flow type.
 */
uint8_t  demo_polflow_ld_get_am_proto(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return (p_polflow->flow.args.l4proto);
}


/*
 * @brief      Query the bitmask of the argumentful flow type PROTOCOL.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Bitmask for Protocol ID.
 */
uint8_t demo_polflow_ld_get_am_proto_m(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return (p_polflow->flow.args.l4proto_m);
}


/*
 * @brief      Query the argument of the argumentful flow type SIP.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Argument (source IP address) of the given flow type.
 */
uint32_t demo_polflow_ld_get_am_sip(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return ntohl(p_polflow->flow.args.sip);
}


/*
 * @brief      Query the bitmask of the argumentful flow type SIP.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Bitmask for source IP address.
 */
uint8_t demo_polflow_ld_get_am_sip_m(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return (p_polflow->flow.args.sip_m);
}


/*
```

```
 * @brief      Query the argument of the argumentful flow type DIP.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Argument (destination IP address) of the given flow type.
 */
uint32_t demo_polflow_ld_get_am_dip(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return ntohl(p_polflow->flow.args.dip);
}


/*
 * @brief      Query the bitmask of the argumentful flow type SIP.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Bitmask for destination IP address.
 */
uint8_t demo_polflow_ld_get_am_dip_m(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return (p_polflow->flow.args.dip_m);
}


/*
 * @brief      Query the argument of the argumentful flow type SPORT.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Argument (source port range - minimal port) of the given flow type.
 */
uint16_t demo_polflow_ld_get_am_sport_min(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return ntohs(p_polflow->flow.args.sport_min);
}


/*
 * @brief      Query the argument of the argumentful flow type SPORT.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Argument (source port range - maximal port) of the given flow type.
 */
uint16_t demo_polflow_ld_get_am_sport_max(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return ntohs(p_polflow->flow.args.sport_max);
}


/*
 * @brief      Query the argument of the argumentful flow type DPORT.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Argument (destination port range - minimal port) of the given flow type.
 */
uint16_t demo_polflow_ld_get_am_dport_min(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return ntohs(p_polflow->flow.args.dport_min);
}


/*
 * @brief      Query the argument of the argumentful flow type DPORT.
 * @details    [localdata_polflow]
 * @param[in]  p_polflow  Local data to be queried.
 * @return     Argument (destination port range - maximal port) of the given flow type.
 */
uint16_t demo_polflow_ld_get_am_dport_max(const fpp_qos_policer_flow_cmd_t* p_polflow)
{
    assert(NULL != p_polflow);
    return ntohs(p_polflow->flow.args.dport_max);
}




/*
 * @brief      Query the name of a parent physical interface of Ingress QoS wred.
 * @details    [localdata_polwred]
 * @param[in]  p_polwred  Local data to be queried.
 * @return     Name of a parent physical interface.
 */
const char* demo_polwred_ld_get_if_name(const fpp_qos_policer_wred_cmd_t* p_polwred)
```

```
{
    assert(p_polwred);
    return (p_polwred->if_name);
}


/*
 * @brief      Query the queue of Ingress QoS wred.
 * @details    [localdata_polwred]
 * @param[in]  p_polwred  Local data to be queried.
 * @return     Queue of the given Ingress QoS wred.
 */
fpp_iqos_queue_t demo_polwred_ld_get_que(const fpp_qos_policer_wred_cmd_t* p_polwred)
{
    assert(p_polwred);
    return (p_polwred->queue);
}


/*
 * @brief      Query the status of Ingress QoS wred "enable" flag.
 * @param[in]  p_polwred  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the Ingress QoS wred:
 *             true  : was enabled
 *             false : was disabled
 */
bool demo_polwred_ld_is_enabled(const fpp_qos_policer_wred_cmd_t* p_polwred)
{
    assert(p_polwred);
    return (p_polwred->enable);
}


/*
 * @brief      Query the minimal threshold of Ingress QoS wred.
 * @details    [localdata_polwred]
 * @param[in]  p_polwred  Local data to be queried.
 * @return     Minimal threshold of Ingress QoS wred.
 */
uint16_t demo_polwred_ld_get_min(const fpp_qos_policer_wred_cmd_t* p_polwred)
{
    assert(p_polwred);
    return ntohs(p_polwred->thr[FPP_IQOS_WRED_MIN_THR]);
}


/*
 * @brief      Query the maximal threshold of Ingress QoS wred.
 * @details    [localdata_polwred]
 * @param[in]  p_polwred  Local data to be queried.
 * @return     Maximal threshold of Ingress QoS wred.
 */
uint16_t demo_polwred_ld_get_max(const fpp_qos_policer_wred_cmd_t* p_polwred)
{
    assert(p_polwred);
    return ntohs(p_polwred->thr[FPP_IQOS_WRED_MAX_THR]);
}


/*
 * @brief      Query the queue length (full threshold) of Ingress QoS wred.
 * @details    [localdata_polwred]
 * @param[in]  p_polwred  Local data to be queried.
 * @return     Queue length (full threshold) of Ingress QoS wred.
 */
uint16_t demo_polwred_ld_get_full(const fpp_qos_policer_wred_cmd_t* p_polwred)
{
    assert(p_polwred);
    return ntohs(p_polwred->thr[FPP_IQOS_WRED_FULL_THR]);
}


/*
 * @brief      Query the percentage chance for packet drop.
 * @details    [localdata_polwred]
 * @param[in]  p_que      Local data to be queried.
 * @param[in]  zprob_id   ID of a probability zone.
 *                        There may be less than 32 zones actually implemented in PFE.
 *                        (32 is just the max array limit)
 *                        See FCI API Reference, chapter Egress QoS.
 * @return     Percentage drop chance of the given probability zone.
 */
uint8_t demo_polwred_ld_get_zprob_by_id(const fpp_qos_policer_wred_cmd_t* p_polwred,
                                        uint8_t zprob_id)
{
    assert(p_polwred);
```

```
    uint8_t percentage = 255u;  /* default value */

    if (FPP_IQOS_WRED_ZONES_COUNT > zprob_id)
    {
        /* FCI command for Ingress QoS wred provides drop probability in compressed format */
        percentage = (uint8_t)((p_polwred->zprob[zprob_id] * 100u) / 0x0Fu);
    }

    return (percentage);
}




 /*
 * @brief      Query the name of a parent physical interface of Ingress QoS shaper.
 * @details    [localdata_polshp]
 * @param[in]  p_polshp  Local data to be queried.
 * @return     Name of a parent physical interface.
 */
const char* demo_polshp_ld_get_if_name(const fpp_qos_policer_shp_cmd_t* p_polshp)
{
    assert(p_polshp);
    return (p_polshp->if_name);
}


/*
 * @brief      Query the ID of Ingress QoS shaper.
 * @details    [localdata_polshp]
 * @param[in]  p_polshp  Local data to be queried.
 * @return     ID of Ingress QoS shaper.
 */
uint8_t demo_polshp_ld_get_id(const fpp_qos_policer_shp_cmd_t* p_polshp)
{
    assert(p_polshp);
    return (p_polshp->id);
}


/*
 * @brief      Query the status of Ingress QoS shaper "enable" flag.
 * @param[in]  p_polshp  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the Ingress QoS wred:
 *             true  : was enabled
 *             false : was disabled
 */
bool demo_polshp_ld_is_enabled(const fpp_qos_policer_shp_cmd_t* p_polshp)
{
    assert(p_polshp);
    return (p_polshp->enable);
}


/*
 * @brief      Query the type of Ingress QoS shaper.
 * @details    [localdata_polshp]
 * @param[in]  p_polshp  Local data to be queried.
 * @return     Type of Ingress QoS shaper.
 */
fpp_iqos_shp_type_t demo_polshp_ld_get_type(const fpp_qos_policer_shp_cmd_t* p_polshp)
{
    assert(p_polshp);
    return (p_polshp->type);
}


/*
 * @brief      Query the mode of Ingress QoS shaper.
 * @details    [localdata_polshp]
 * @param[in]  p_polshp  Local data to be queried.
 * @return     Mode of Ingress QoS shaper.
 */
fpp_iqos_shp_rate_mode_t demo_polshp_ld_get_mode(const fpp_qos_policer_shp_cmd_t* p_polshp)
{
    assert(p_polshp);
    return (p_polshp->mode);
}


/*
 * @brief      Query the idle slope of Ingress QoS shaper.
 * @details    [localdata_polshp]
 * @param[in]  p_polshp  Local data to be queried.
 * @return     Idle slope of Ingress QoS shaper.
```

```
 */
uint32_t demo_polshp_ld_get_isl(const fpp_qos_policer_shp_cmd_t* p_polshp)
{
    assert(p_polshp);
    return ntohl(p_polshp->isl);
}


/*
 * @brief      Query the maximal credit of Ingress QoS shaper.
 * @details    [localdata_polshp]
 * @param[in]  p_polshp  Local data to be queried.
 * @return     Maximal credit of Ingress QoS shaper.
 */
int32_t demo_polshp_ld_get_max_credit(const fpp_qos_policer_shp_cmd_t* p_polshp)
{
    assert(p_polshp);
    return (int32_t)(ntohl(p_polshp->max_credit));
}


/*
 * @brief      Query the minimal credit of Ingress QoS shaper.
 * @details    [localdata_polshp]
 * @param[in]  p_polshp  Local data to be queried.
 * @return     Minimal credit of a QoS shaper.
 */
int32_t demo_polshp_ld_get_min_credit(const fpp_qos_policer_shp_cmd_t* p_polshp)
{
    assert(p_polshp);
    return (int32_t)(ntohl(p_polshp->min_credit));
}


/* ==== PUBLIC FUNCTIONS : misc ============================================ */


/*
 * @brief      Use FCI calls to iterate through all available Ingress QoS wreds of
 *             a given physical interface and execute a callback print function for
 *             each Ingress QoS wred.
 * @param[in]  p_cl          FCI client
 * @param[in]  p_cb_print    Callback print function.
 *                           --> If the callback returns ZERO, then all is OK and
 *                               a next Ingress QoS wred is picked for a print process.
 *                           --> If the callback returns NON-ZERO, then some problem is
 *                               assumed and this function terminates prematurely.
 * @param[in]  p_phyif_name  Name of a parent physical interface.
 *                           Names of physical interfaces are hardcoded.
 *                           See FCI API Reference, chapter Interface Management.
 * @return     FPP_ERR_OK : Successfully iterated through all available Ingress QoS wred of
 *                          the given physical interface.
 *             other      : Some error occurred (represented by the respective error code).
 *                          No count was stored.
 */
int demo_polwred_print_by_phyif(FCI_CLIENT* p_cl, demo_polwred_cb_print_t p_cb_print,
                                const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_wred_cmd_t cmd_to_fci = {0};
    fpp_qos_policer_wred_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* query loop */
        uint8_t wred_queue = 0u;
        while (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.queue = wred_queue;
            cmd_to_fci.action = FPP_ACTION_QUERY;
            rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_WRED,
                            sizeof(fpp_qos_policer_wred_cmd_t),
                            (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));

            if (FPP_ERR_OK == rtn)
```

```
            {
                rtn = p_cb_print(&reply_from_fci);
            }

            wred_queue++;
        }

        /* query loop runs till there are no more Ingress QoS wreds to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_INTERNAL_FAILURE == rtn)
        {
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_polwred_print_by_phyif() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available Ingress QoS wreds in PFE which
 *              are a part of a given parent physical interface.
 * @param[in]   p_cl          FCI client
 * @param[out]  p_rtn_count   Space to store the count of Ingress QoS wreds.
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : Successfully counted all applicable Ingress QoS wreds.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_polwred_get_count_by_phyif(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                                    const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_wred_cmd_t cmd_to_fci = {0};
    fpp_qos_policer_wred_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* query loop */
        uint8_t wred_queue = 0u;
        while (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.queue = wred_queue;
            cmd_to_fci.action = FPP_ACTION_QUERY;
            rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_WRED,
                            sizeof(fpp_qos_policer_wred_cmd_t),
                            (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));

            wred_queue++;
        }

        /* query loop runs till there are no more Ingress QoS wreds to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_INTERNAL_FAILURE == rtn)
        {
            *p_rtn_count = wred_queue;
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_polwred_get_count_by_phyif() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to iterate through all available Ingress QoS shapers of
 *             a given physical interface and execute a callback print function for
 *             each Ingress QoS shaper.
 * @param[in]  p_cl          FCI client
```

```
 * @param[in]  p_cb_print    Callback print function.
 *                           --> If the callback returns ZERO, then all is OK and
 *                               a next Ingress QoS shaper is picked for a print process.
 *                           --> If the callback returns NON-ZERO, then some problem is
 *                               assumed and this function terminates prematurely.
 * @param[in]  p_phyif_name  Name of a parent physical interface.
 *                           Names of physical interfaces are hardcoded.
 *                           See FCI API Reference, chapter Interface Management.
 * @return     FPP_ERR_OK : Successfully iterated through all available Ingress QoS shapers of
 *                          the given physical interface.
 *             other      : Some error occurred (represented by the respective error code).
 *                          No count was stored.
 */
int demo_polshp_print_by_phyif(FCI_CLIENT* p_cl, demo_polshp_cb_print_t p_cb_print,
                               const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_shp_cmd_t cmd_to_fci = {0};
    fpp_qos_policer_shp_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* query loop */
        uint8_t shp_id = 0u;
        while (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.id = shp_id;
            cmd_to_fci.action = FPP_ACTION_QUERY;
            rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_SHP,
                            sizeof(fpp_qos_policer_shp_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));

            if (FPP_ERR_OK == rtn)
            {
                rtn = p_cb_print(&reply_from_fci);
            }

            shp_id++;
        }

        /* query loop runs till there are no more Ingress QoS shapers to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_INTERNAL_FAILURE == rtn)
        {
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_polshp_print_by_phyif() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available Ingress QoS shapers in PFE which
 *              are a part of a given parent physical interface.
 * @param[in]   p_cl          FCI client
 * @param[out]  p_rtn_count   Space to store the count of Ingress QoS shapers.
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : Successfully counted all applicable Ingress QoS shapers.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_polshp_get_count_by_phyif(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                                   const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_shp_cmd_t cmd_to_fci = {0};
```

```
    fpp_qos_policer_shp_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* query loop */
        uint8_t shp_id = 0u;
        while (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.id = shp_id;
            cmd_to_fci.action = FPP_ACTION_QUERY;
            rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_SHP,
                            sizeof(fpp_qos_policer_shp_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));

            shp_id++;
        }

        /* query loop runs till there are no more Ingress QoS shapers to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_INTERNAL_FAILURE == rtn)
        {
            *p_rtn_count = shp_id;
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_polshp_get_count_by_phyif() failed!");

    return (rtn);
}


/*
 * @brief      Use FCI calls to iterate through all available Ingress QoS flows of
 *             a given physical interface and execute a callback print function for
 *             each Ingress QoS flow.
 * @param[in]  p_cl          FCI client
 * @param[in]  p_cb_print    Callback print function.
 *                           --> If the callback returns ZERO, then all is OK and
 *                               a next Ingress QoS flow is picked for a print process.
 *                           --> If the callback returns NON-ZERO, then some problem is
 *                               assumed and this function terminates prematurely.
 * @param[in]  p_phyif_name  Name of a parent physical interface.
 *                           Names of physical interfaces are hardcoded.
 *                           See FCI API Reference, chapter Interface Management.
 * @return     FPP_ERR_OK : Successfully iterated through all available Ingress QoS flows of
 *                          the given physical interface.
 *             other      : Some error occurred (represented by the respective error code).
 *                          No count was stored.
 */
int demo_polflow_print_by_phyif(FCI_CLIENT* p_cl, demo_polflow_cb_print_t p_cb_print,
                                const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_flow_cmd_t cmd_to_fci = {0};
    fpp_qos_policer_flow_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_FLOW,
                        sizeof(fpp_qos_policer_flow_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

        /* query loop */
        while (FPP_ERR_OK == rtn)
        {
            rtn = p_cb_print(&reply_from_fci);

            print_if_error(rtn, "demo_polflow_print_by_phyif() --> "
```

```
                                 "non-zero return from callback print function!");

            if (FPP_ERR_OK == rtn)
            {
                cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
                rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_FLOW,
                                sizeof(fpp_qos_policer_flow_cmd_t),
                                (unsigned short*)(&cmd_to_fci),
                                &reply_length, (unsigned short*)(&reply_from_fci));
            }
        }

        /* query loop runs till there are no more Ingress QoS flows to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_QOS_POLICER_FLOW_NOT_FOUND == rtn)
        {
            rtn = FPP_ERR_OK;
        }
    }

    print_if_error(rtn, "demo_polflow_print_by_phyif() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available Ingress QoS flows in PFE which
 *              are a part of a given parent physical interface.
 * @param[in]   p_cl          FCI client
 * @param[out]  p_rtn_count   Space to store the count of Ingress QoS flows.
 * @param[in]   p_phyif_name  Name of a parent physical interface.
 *                            Names of physical interfaces are hardcoded.
 *                            See FCI API Reference, chapter Interface Management.
 * @return      FPP_ERR_OK : Successfully counted all applicable Ingress QoS flows.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_polflow_get_count_by_phyif(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                                    const char* p_phyif_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_phyif_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_qos_policer_flow_cmd_t cmd_to_fci = {0};
    fpp_qos_policer_flow_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint32_t count = 0u;

    /* prepare data */
    rtn = set_text((cmd_to_fci.if_name), p_phyif_name, IFNAMSIZ);

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_FLOW,
                        sizeof(fpp_qos_policer_flow_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

        /* query loop */
        while (FPP_ERR_OK == rtn)
        {
            count++;

            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_QOS_POLICER_FLOW,
                            sizeof(fpp_qos_policer_flow_cmd_t),
                            (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more logical interfaces to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_IF_ENTRY_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_polflow_get_count_by_phyif() failed!");
```

```
    return (rtn);
}

/* ========================================================================= */
```

15.22 demo_fwfeat.c

```
/* =========================================================================
 *  Copyright 2020-2022 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_fwfeat.h"

/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from the PFE ========== */


/*
 * @brief       Use FCI calls to get configuration data of a requested FW feature
 *              from PFE. Identify the FW feature by its name.
 * @param[in]   p_cl            FCI client
 * @param[out]  p_rtn_fwfeat    Space for data from PFE.
 * @param[in]   p_feature_name  Name of the requested FW feature.
 *                              Names of FW features are hardcoded.
 *                              Use FPP_ACTION_QUERY+FPP_ACTION_QUERY_CONT to get a list of
 *                              available FW features (and their names) from PFE.
 *                              See demo_fwfeat_print_all().
 * @return      FPP_ERR_OK : The requested FW feature was found.
 *                           A copy of its configuration data was stored into p_rtn_fwfeat.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_fwfeat_get_by_name(FCI_CLIENT* p_cl, fpp_fw_features_cmd_t* p_rtn_fwfeat,
                            const char* p_feature_name)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_fwfeat);
    assert(NULL != p_feature_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
```

```
    fpp_fw_features_cmd_t cmd_to_fci = {0};
    fpp_fw_features_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE,
                    sizeof(fpp_fw_features_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop (with a search condition) */
    while ((FPP_ERR_OK == rtn) && (strcmp(p_feature_name, reply_from_fci.name)))
    {
        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE,
                        sizeof(fpp_fw_features_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_fwfeat = reply_from_fci;
    }

    print_if_error(rtn, "demo_fwfeat_get_by_name() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get data of a requested FW feature element
 *              from PFE. Identify the element by name of its parent FW feature and
 *              by name of the target element.
 * @param[in]   p_cl             FCI client
 * @param[out]  p_rtn_fwfeat_el  Space for data from PFE.
 * @param[in]   p_feature_name   Name of the requested FW feature.
 *                               Names of FW features are hardcoded.
 *                               Use FPP_ACTION_QUERY+FPP_ACTION_QUERY_CONT to get a list of
 *                               available FW features (and their names) from PFE.
 *                               See demo_fwfeat_print_all().
 * @param[in]   p_element_name   Name of the requested FW feature element.
 *                               Names of FW feature elements are hardcoded.
 *                               Use FPP_ACTION_QUERY+FPP_ACTION_QUERY_CONT to get a list of
 *                               available FW feature elements from PFE.
 * @param[in]   group            Element group where to search.
 *                               Groups are described in struct definition of
 *                               fpp_fw_features_element_cmd_t.
 * @param[in]   index            Element can have an array of data units. This parameter is
 *                               an index that specifies where to start querying within
 *                               element's data array. Quried data will be in the .payload.
 * @return      FPP_ERR_OK : The requested FW feature element was found.
 *                           A copy of its data was stored into p_rtn_fwfeat_el.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_fwfeat_el_get_by_name(FCI_CLIENT* p_cl,
                               fpp_fw_features_element_cmd_t* p_rtn_fwfeat_el,
                               const char* p_feature_name, const char* p_element_name,
                               uint8_t group, uint8_t index)
{
    assert(NULL != p_cl);
    assert(NULL != p_feature_name);
    assert(NULL != p_element_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_fw_features_element_cmd_t cmd_to_fci = {0};
    fpp_fw_features_element_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    cmd_to_fci.group = group;
    cmd_to_fci.index = index;
    rtn = set_text((cmd_to_fci.fw_feature_name), p_feature_name,
                   (FPP_FEATURE_NAME_SIZE + 1));

    if (FPP_ERR_OK == rtn)
    {
        rtn = set_text((cmd_to_fci.element_name), p_element_name,
                       (FPP_FEATURE_NAME_SIZE + 1));
    }

    /* do the query (get the element directly; no need for a loop) */
    if (FPP_ERR_OK == rtn)
```

```
    {
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE_ELEMENT,
                        sizeof(fpp_fw_features_element_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* if a query is successful, then assign the data */
    if (FPP_ERR_OK == rtn)
    {
        *p_rtn_fwfeat_el = reply_from_fci;
    }

    print_if_error(rtn, "demo_fwfeat_el_get_by_name() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : use FCI calls to update data in PFE ============= */


/*
 * @brief      Use FCI calls to enable/disable a target FW feature in PFE.
 * @param[in]  p_cl            FCI client
 * @param[in]  p_feature_name  Name of a FW feature.
 *                             Names of FW features are hardcoded.
 *                             Use FPP_ACTION_QUERY+FPP_ACTION_QUERY_CONT to get a list of
 *                             available FW features (and their names) from PFE.
 *                             See demo_fwfeat_print_all().
 * @param[in]  enable          Request to set/unset the FW feature.
 * @return     FPP_ERR_OK : FW feature was successfully enabled/disabled in PFE.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_fwfeat_set(FCI_CLIENT* p_cl, const char* p_feature_name, bool enable)
{
    assert(NULL != p_cl);
    assert(NULL != p_feature_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_fw_features_cmd_t cmd_to_fci = {0};

    /* prepare data */
    rtn = set_text((cmd_to_fci.name), p_feature_name, (FPP_FEATURE_NAME_SIZE + 1));
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.val = enable;  /* NOTE: Implicit cast from bool to uintX_t */
    }

    /* send data */
    if (FPP_ERR_OK == rtn)
    {
        cmd_to_fci.action = FPP_ACTION_UPDATE;
        rtn = fci_write(p_cl, FPP_CMD_FW_FEATURE, sizeof(fpp_fw_features_cmd_t),
                                                  (unsigned short*)(&cmd_to_fci));
    }

    print_if_error(rtn, "demo_fwfeat_set() failed!");

    return (rtn);
}


/*
 * @brief          Use FCI calls to update data of a FW feature element in PFE.
 * @param[in]      p_cl         FCI client
 * @param[in,out]  p_fwfeat_el  Local data struct which represents new data of
 *                              the target FW feature element.
 *                              It is assumed that the struct contains a valid data of some
 *                              FW feature element, just modified via some fwfeat_el setters.
 * @return        FPP_ERR_OK : Data of the target FW feature element were
 *                             successfully updated in PFE.
 *                             The local data struct was automatically updated with
 *                             readback data from PFE.
 *                other      : Some error occurred (represented by the respective error code).
 *                             The local data struct was not updated.
 */
int demo_fwfeat_el_set(FCI_CLIENT* p_cl, fpp_fw_features_element_cmd_t* p_fwfeat_el)
{
    assert(NULL != p_cl);
    assert(NULL != p_fwfeat_el);

    int rtn = FPP_ERR_INTERNAL_FAILURE;
    fpp_fw_features_element_cmd_t cmd_to_fci = *p_fwfeat_el;

    /* send data */
```



```
    cmd_to_fci.action = FPP_ACTION_UPDATE;
    rtn = fci_write(p_cl, FPP_CMD_FW_FEATURE_ELEMENT,
                    sizeof(fpp_fw_features_element_cmd_t), (unsigned short*)(&cmd_to_fci));

    /* read back and update caller data */
    if (FPP_ERR_OK == rtn)
    {
        rtn = demo_fwfeat_el_get_by_name(p_cl, p_fwfeat_el,
                                         (p_fwfeat_el->fw_feature_name),
                                         (p_fwfeat_el->element_name),
                                         (p_fwfeat_el->group),
                                         (p_fwfeat_el->index));
    }

    print_if_error(rtn, "demo_fwfeat_el_set() failed!");

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : modify local data (no FCI calls) ================ */
/*
 * @defgroup    localdata_fwfeat_el  [localdata_fwfeat_el]
 * @brief:      Functions marked as [localdata_fwfeat_el] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_fwfeat_el (a struct with element data).
 *              Initial data for p_fwfeat_el can be obtained via demo_fwfeat_el_get_by_name().
 */


/*
 * @brief          Set the element group of a FW feature element.
 * @details        [localdata_fwfeat_el]
 *                 This setter should be rarely needed. If FW element data were obtained
 *                 from PFE via demo_fwfeat_el_get_by_name(), then the data should already
 *                 have a correct group set.
 * @param[in,out]  p_fwfeat_el  Local data to be modified.
 * @param[in]      group        Element group. For explanation about element groups, see
 *                              description of fpp_fw_features_element_cmd_t.
 */
void demo_fwfeat_el_set_group(fpp_fw_features_element_cmd_t* p_fwfeat_el, uint8_t group)
{
    assert(NULL != p_fwfeat_el);
    p_fwfeat_el->group = group;
}


/*
 * @brief          Set the index of a FW feature element.
 * @details        [localdata_fwfeat_el]
 *                 What is index:
 *                   [*] FW feature element (as stored in PFE firmware) can have
 *                       an array of data units.
 *                   [*] FCI command allows querying or updating a particular item from
 *                       such array by specifying index of the target item.
 *                   [*] A consecutive series of array items can be queried or updated by
 *                       a single FCI command. The index specifies starting point for such
 *                       query/update operation.
 * @param[in,out]  p_fwfeat_el  Local data to be modified.
 * @param[in]      index        Index into element's data array in PFE.
 */
void demo_fwfeat_el_set_index(fpp_fw_features_element_cmd_t* p_fwfeat_el, uint8_t index)
{
    assert(NULL != p_fwfeat_el);
    p_fwfeat_el->index = index;
}


/*
 * @brief          Set the payload of a FW feature element.
 * @details        [localdata_fwfeat_el]
 * @param[in,out]  p_fwfeat_el Local data to be modified.
 * @param[in]      p_payload   New payload.
 * @param[in]      count       Count of data units in the new payload.
 * @param[in]      unit_size   Bytesize of a data unit.
 */
int demo_fwfeat_el_set_payload(fpp_fw_features_element_cmd_t* p_fwfeat_el,
                               const uint8_t* p_payload, uint8_t count, uint8_t unit_size)
{
    assert(NULL != p_fwfeat_el);
    assert(NULL != p_payload);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    if (sizeof(p_fwfeat_el->payload) >= (count * unit_size))
    {
```

```
        p_fwfeat_el->count = count;
        p_fwfeat_el->unit_size = unit_size;
        memcpy(p_fwfeat_el->payload, p_payload, (count * unit_size));

        rtn = FPP_ERR_OK;
    }

    return (rtn);
}


/* ==== PUBLIC FUNCTIONS : query local data (no FCI calls) ================= */
/*
 * @defgroup    localdata_fwfeat  [localdata_fwfeat]
 * @brief:      Functions marked as [localdata_fwfeat] access only local data.
 *              No FCI calls are made.
 * @details:    These functions have a parameter p_fwfeat (a struct with configuration data).
 *              Initial data for p_fwfeat can be obtained via demo_fwfeat_get_by_name().
 */


/*
 * @brief      Query the current status of a FW feature.
 * @details    [localdata_fwfeat]
 * @param[in]  p_fwfeat  Local data to be queried.
 * @return     At time when the data was obtained from PFE, the FW feature:
 *             true  : was enabled
 *             false : was disabled
 */
bool demo_fwfeat_ld_is_enabled(const fpp_fw_features_cmd_t* p_fwfeat)
{
    assert(NULL != p_fwfeat);
    return (bool)(p_fwfeat->val);
}


/*
 * @brief      Query the default status of a FW feature.
 * @details    [localdata_fwfeat]
 * @param[in]  p_fwfeat  Local data to be queried.
 * @return     By default, the FW feature:
 *             true  : is initially enabled
 *             false : is initially disabled
 */
bool demo_fwfeat_ld_is_enabled_by_def(const fpp_fw_features_cmd_t* p_fwfeat)
{
    assert(NULL != p_fwfeat);
    return (bool)(p_fwfeat->def_val);
}




/*
 * @brief      Query the name of a FW feature.
 * @details    [localdata_fwfeat]
 * @param[in]  p_fwfeat  Local data to be queried.
 * @return     Name of the FW feature.
 */
const char* demo_fwfeat_ld_get_name(const fpp_fw_features_cmd_t* p_fwfeat)
{
    assert(NULL != p_fwfeat);
    return (p_fwfeat->name);
}


/*
 * @brief      Query the description text of a FW feature.
 * @details    [localdata_fwfeat]
 * @param[in]  p_fwfeat  Local data to be queried.
 * @return     Description text of the FW feature.
 */
const char* demo_fwfeat_ld_get_desc(const fpp_fw_features_cmd_t* p_fwfeat)
{
    assert(NULL != p_fwfeat);
    return (p_fwfeat->desc);
}


/*
 * @brief      Query the variant of a FW feature.
 * @details    [localdata_fwfeat]
 * @param[in]  p_fwfeat  Local data to be queried.
 * @return     Flags (bitset) of a FW feature.
 */
fpp_fw_feature_flags_t demo_fwfeat_ld_get_flags(const fpp_fw_features_cmd_t* p_fwfeat)
```

```
{
    assert(NULL != p_fwfeat);
    return (p_fwfeat->flags);
}




/*
 * @brief      Query the name of a FW feature element.
 * @details    [localdata_fwfeat_el]
 * @param[in]  p_fwfeat_el  Local data to be queried.
 * @return     Name of the FW feature element.
 */
const char* demo_fwfeat_el_ld_get_name(const fpp_fw_features_element_cmd_t* p_fwfeat_el)
{
    assert(NULL != p_fwfeat_el);
    return (p_fwfeat_el->element_name);
}


/*
 * @brief      Query the name of element's parent FW feature.
 * @details    [localdata_fwfeat_el]
 * @param[in]  p_fwfeat_el  Local data to be queried.
 * @return     Name of the element's parent FW feature.
 */
const char* demo_fwfeat_el_ld_get_feat_name(const fpp_fw_features_element_cmd_t* p_fwfeat_el)
{
    assert(NULL != p_fwfeat_el);
    return (p_fwfeat_el->fw_feature_name);
}


/*
 * @brief      Query the element group of a FW feature element.
 * @details    [localdata_fwfeat_el]
 * @param[in]  p_fwfeat_el  Local data to be queried.
 * @return     Element group. For explanation about element groups, see
 *             description of fpp_fw_features_element_cmd_t.
 */
uint8_t demo_fwfeat_el_ld_get_group(const fpp_fw_features_element_cmd_t* p_fwfeat_el)
{
    assert(NULL != p_fwfeat_el);
    return (p_fwfeat_el->group);
}


/*
 * @brief      Query the index of a FW feature element.
 * @details    [localdata_fwfeat_el]
 *             What is index:
 *               [*] FW feature element (as stored in PFE firmware) can have
 *                   an array of data units.
 *               [*] FCI command allows querying or updating a particular item from such array
 *                   by specifying index of the target item.
 *               [*] A consecutive series of array items can be queried or updated by a single
 *                   FCI command. The index specifies starting point for such query/update
 *                   operation.
 * @param[in]  p_fwfeat_el  Local data to be queried.
 * @return     index
 */
uint8_t demo_fwfeat_el_ld_get_index(const fpp_fw_features_element_cmd_t* p_fwfeat_el)
{
    assert(NULL != p_fwfeat_el);
    return (p_fwfeat_el->index);
}


/*
 * @brief       Query the payload of a FW feature element.
 * @details     [localdata_fwfeat_el]
 * @param[in]   p_fwfeat_el   Local data to be queried.
 * @param[out]  pp_rtn_payload   Passback value. Pointer to payload data bytearray.
 * @param[out]  p_rtn_count      Passback value. Count of data units in payload.
 * @param[out]  p_rtn_unit_size  Passback value. Bytesize of a data unit.
 */
void demo_fwfeat_el_ld_get_payload(const fpp_fw_features_element_cmd_t* p_fwfeat_el,
                                   const uint8_t** pp_rtn_payload, uint8_t* p_rtn_count,
                                   uint8_t* p_rtn_unit_size)
{
    assert(NULL != p_fwfeat_el);
    assert((NULL != pp_rtn_payload) && (NULL != p_rtn_count) && (NULL != p_rtn_unit_size));

    *pp_rtn_payload = p_fwfeat_el->payload;
    *p_rtn_count = p_fwfeat_el->count;
```

```
    *p_rtn_unit_size = p_fwfeat_el->unit_size;
}


/* ==== PUBLIC FUNCTIONS : misc ============================================ */


/*
 * @brief      Use FCI calls to iterate through all available FW features in PFE and
 *             execute a callback print function for each reported FW feature.
 * @param[in]  p_cl        FCI client
 * @param[in]  p_cb_print  Callback print function.
 *                         --> If the callback returns ZERO, then all is OK and
 *                             a next FW feature is picked for a print process.
 *                         --> If the callback returns NON-ZERO, then some problem is
 *                             assumed and this function terminates prematurely.
 * @return     FPP_ERR_OK : Successfully iterated through all available FW features.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_fwfeat_print_all(FCI_CLIENT* p_cl, demo_fwfeat_cb_print_t p_cb_print)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_fw_features_cmd_t cmd_to_fci = {0};
    fpp_fw_features_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
    rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE,
                    sizeof(fpp_fw_features_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        rtn = p_cb_print(&reply_from_fci);

        if (FPP_ERR_OK == rtn)
        {
            cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
            rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE,
                            sizeof(fpp_fw_features_cmd_t), (unsigned short*)(&cmd_to_fci),
                            &reply_length, (unsigned short*)(&reply_from_fci));
        }
    }

    /* query loop runs till there are no more FW features to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_FW_FEATURE_NOT_FOUND == rtn)
    {
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_fwfeat_print_all() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all available FW features in PFE.
 * @param[in]   p_cl         FCI client
 * @param[out]  p_rtn_count  Space to store the count of FW features.
 * @return      FPP_ERR_OK : Successfully counted all available FW features.
 *                           Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_fwfeat_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_fw_features_cmd_t cmd_to_fci = {0};
    fpp_fw_features_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    uint32_t count = 0u;

    /* start query process */
    cmd_to_fci.action = FPP_ACTION_QUERY;
```

```
    rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE,
                    sizeof(fpp_fw_features_cmd_t), (unsigned short*)(&cmd_to_fci),
                    &reply_length, (unsigned short*)(&reply_from_fci));

    /* query loop */
    while (FPP_ERR_OK == rtn)
    {
        count++;

        cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
        rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE,
                        sizeof(fpp_fw_features_cmd_t), (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));
    }

    /* query loop runs till there are no more FW features to report */
    /* the following error is therefore OK and expected (it ends the query loop) */
    if (FPP_ERR_FW_FEATURE_NOT_FOUND == rtn)
    {
        *p_rtn_count = count;
        rtn = FPP_ERR_OK;
    }

    print_if_error(rtn, "demo_fwfeat_get_count() failed!");

    return (rtn);
}




/*
 * @brief      Use FCI calls to iterate through all available elements of a target FW feature
 *             in PFE and execute a callback print function for each reported element.
 * @param[in]  p_cl        FCI client
 * @param[in]  p_cb_print  Callback print function.
 *                         --> If the callback returns ZERO, then all is OK and
 *                             a next element is picked for a print process.
 *                         --> If the callback returns NON-ZERO, then some problem is
 *                             assumed and this function terminates prematurely.
 * @param[in]  p_feature_name  Name of the requested FW feature.
 *                             Names of FW features are hardcoded.
 *                             Use FPP_ACTION_QUERY+FPP_ACTION_QUERY_CONT to get a list of
 *                             available FW features (and their names) from PFE.
 *                             See demo_fwfeat_print_all().
 * @param[in]  group           Element group where to search.
 *                             Groups are described in struct definition of
 *                             fpp_fw_features_element_cmd_t.
 * @return     FPP_ERR_OK : Successfully iterated through all applicable elements of the
 *                          target FW feature.
 *             other      : Some error occurred (represented by the respective error code).
 */
int demo_fwfeat_el_print_all(FCI_CLIENT* p_cl, demo_fwfeat_el_cb_print_t p_cb_print,
                             const char* p_feature_name, uint8_t group)
{
    assert(NULL != p_cl);
    assert(NULL != p_cb_print);
    assert(NULL != p_feature_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_fw_features_element_cmd_t cmd_to_fci = {0};
    fpp_fw_features_element_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;

    /* prepare data */
    cmd_to_fci.group = group;
    rtn = set_text((cmd_to_fci.fw_feature_name), p_feature_name, (FPP_FEATURE_NAME_SIZE + 1));

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE_ELEMENT,
                        sizeof(fpp_fw_features_element_cmd_t),
                        (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

        /* query loop */
        while (FPP_ERR_OK == rtn)
        {
            rtn = p_cb_print(&reply_from_fci);

            if (FPP_ERR_OK == rtn)
            {
```



```
                cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
                rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE_ELEMENT,
                                sizeof(fpp_fw_features_element_cmd_t),
                                (unsigned short*)(&cmd_to_fci),
                                &reply_length, (unsigned short*)(&reply_from_fci));
            }
        }

        /* query loop runs till there are no more FW feature elements to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_FW_FEATURE_ELEMENT_NOT_FOUND == rtn)
        {
            rtn = FPP_ERR_OK;
        }
    }
    print_if_error(rtn, "demo_fwfeat_el_print_all() failed!");

    return (rtn);
}


/*
 * @brief       Use FCI calls to get a count of all applicable elements of a target FW feature
 *              in PFE.
 * @param[in]   p_cl            FCI client
 * @param[out]  p_rtn_count     Space to store the count of FW features.
 * @param[in]   p_feature_name  Name of the requested FW feature.
 *                              Names of FW features are hardcoded.
 *                              Use FPP_ACTION_QUERY+FPP_ACTION_QUERY_CONT to get a list of
 *                              available FW features (and their names) from PFE.
 *                              See demo_fwfeat_print_all().
 * @param[in]   group           Element group where to search.
 *                              Groups are described in struct definition of
 *                              fpp_fw_features_element_cmd_t.
 * @return      FPP_ERR_OK : Successfully counted all applicable elements of
 *                           the target FW feature. Count was stored into p_rtn_count.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No count was stored.
 */
int demo_fwfeat_el_get_count(FCI_CLIENT* p_cl, uint32_t* p_rtn_count,
                             const char* p_feature_name, uint8_t group)
{
    assert(NULL != p_cl);
    assert(NULL != p_rtn_count);
    assert(NULL != p_feature_name);

    int rtn = FPP_ERR_INTERNAL_FAILURE;

    fpp_fw_features_element_cmd_t cmd_to_fci = {0};
    fpp_fw_features_element_cmd_t reply_from_fci = {0};
    unsigned short reply_length = 0u;
    int32_t count = 0u;

    /* prepare data */
    cmd_to_fci.group = group;
    rtn = set_text((cmd_to_fci.fw_feature_name), p_feature_name, (FPP_FEATURE_NAME_SIZE + 1));

    /* do the query */
    if (FPP_ERR_OK == rtn)
    {
        /* start query process */
        cmd_to_fci.action = FPP_ACTION_QUERY;
        rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE_ELEMENT,
                        sizeof(fpp_fw_features_element_cmd_t),
                        (unsigned short*)(&cmd_to_fci),
                        &reply_length, (unsigned short*)(&reply_from_fci));

        /* query loop */
        while (FPP_ERR_OK == rtn)
        {
            count++;

            if (FPP_ERR_OK == rtn)
            {
                cmd_to_fci.action = FPP_ACTION_QUERY_CONT;
                rtn = fci_query(p_cl, FPP_CMD_FW_FEATURE_ELEMENT,
                                sizeof(fpp_fw_features_element_cmd_t),
                                (unsigned short*)(&cmd_to_fci),
                                &reply_length, (unsigned short*)(&reply_from_fci));
            }
        }

        /* query loop runs till there are no more FW feature elements to report */
        /* the following error is therefore OK and expected (it ends the query loop) */
        if (FPP_ERR_FW_FEATURE_ELEMENT_NOT_FOUND == rtn)
        {
```



```
            *p_rtn_count = count;
            rtn = FPP_ERR_OK;
        }
    }
    print_if_error(rtn, "demo_fwfeat_el_get_count() failed!");

    return (rtn);
}


/* ========================================================================= */
```

15.23 demo_fci_owner.c

```
/* =========================================================================
 *  Copyright 2022 NXP
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * ========================================================================= */


#include <assert.h>
#include <string.h>
#include <arpa/inet.h>

#include <stdint.h>
#include <stdbool.h>
#include "fpp.h"
#include "fpp_ext.h"
#include "libfci.h"

#include "demo_common.h"
#include "demo_fci_owner.h"

/* ==== PUBLIC FUNCTIONS : use FCI calls to get data from PFE ============== */

/*
 * @brief       Use FCI calls to get FCI ownership.
 * @param[in]   p_cl         FCI client
 * @return      FPP_ERR_OK : The FCI ownership was granted.
 *              other      : Some error occurred (represented by the respective error code).
 *                           No data copied.
 */
int demo_fci_ownership_lock(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = fci_write(p_cl, FPP_CMD_FCI_OWNERSHIP_LOCK, 0u, NULL);
    print_if_error(rtn, "demo_fci_ownership_lock() failed!");
    return (rtn);
}

/*
 * @brief       Use FCI calls to release FCI ownership.
 * @param[in]   p_cl         FCI client
 * @return      FPP_ERR_OK : The FCI ownership was released.
 *              other      : Some error occurred (represented by the respective error code).
```

```
 *                              No data copied.
 */
int demo_fci_ownership_unlock(FCI_CLIENT* p_cl)
{
    assert(NULL != p_cl);
    int rtn = fci_write(p_cl, FPP_CMD_FCI_OWNERSHIP_UNLOCK, 0u, NULL);
    print_if_error(rtn, "demo_fci_ownership_unlock() failed!");
    return (rtn);
}

/* ========================================================================= */
```


Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 18 September 2023
Document identifier: PFE FCI API Reference