FP Module Core Components

Design Specification

Control Plane-Platform Development Kit 2.11

March 2004

intel

Contents

Tables

Revision History

Revision	Description	Date	Author
2.11	Updated for release 2.11	March 2004	Sreedhara D.S, Shabbir Ali
2.1	Updated for release 2.1	December 2003	Sreedhara D.S, Shabbir Ali
2.0	Updated for release 2.0	August 2003	Sreedhara D.S, Shabbir Ali

Part 1: Introduction

1 Introduction

This document describes the design of the Forwarding Plane (FP) module, ingress and egress core components. These are a part of the CP-PDK release 2.11, targeted for IXA SDK 3.51 running on the IXP 2400 and 2800 series of network processors. The functionality of these components is as follows:

The FP module ingress CC is responsible for communicating with the FP module. It also receives the following events/exceptions:

- Receive Link state change events.
- No Direct Route Exception (sent up by the transmit microblock). This is required for supporting directly connected hosts.

The egress CC interfaces with the IXA SDK L2 Table Manager to:

- Add/delete/purge the L2 Table.
- Add/delete/purge ARP entries.
- Retrieve transmit statistics for Ethernet and POS.

This document is intended for developers and testers of the FP module ingress and egress core components, and also for anyone that wants to make extensions to the functionality of this component.

1.1 Terminology

The following table lists terms used in this document and provides a definition for each term.

Table 1. Terminology table

Term	Description
ARP	Address Resolution Protocol
Control Element (CE), Control Plane (CP)	In a separated control/data system, refers to the processor(s) responsible for control and configuration of forwarding elements. Used interchangeable with Control Plane (CP)
CP-PDK	Control Plane Platform Development Kit
Forwarding Element (FE), Forwarding Plane (FP)	In a separated control/data system, refers to the processor(s) responsible for fast path forwarding of data. Used interchangeably with FP.
ICMP	Internet Control Message Protocol
IXA	Internet eXchange Architecture
IXP 2XXX	Network processors in the Intel IXA family. There are two versions of the IXP2X00 – IXP2400 with 8 microengines targeted at OC-48 POS line rates and IXP2800 with 16 microengine targeted at OC-192 POS line rates.
MPLS	Multiprotocol Label Switching

Term	Description
NPF	Network Processing Forum
OSPF	Open Shortest Path First (routing protocol)
RIP	Routing Information Protocol
Intel® XScale™ core	Forms the core of the IXP 2400 and 2800
ATM	Asynchronous Transfer Mode
CC	Core Component
CCI	Core Component Infrastructure
CSIX	Common Switch Interface – protocol for switch fabric that connects two or more forwarding blades
ME or Microengine	One of many (8 for IXP2400, 16 for IXP2800) programmable, specialized processors on the IXP2X00.
Microblock	An IXP microC or microcode function that follows the guidelines provided in the Level 0 Software model. The function conforms to one of three different types: source, transform or sink. Typically, a microblock has an XScale component that is used to configure and manage the microblock.
POS	Packet Over SONET (Synchronous Optical NETwork)
RTM	Route Table Manager.

1.2 References

The following table lists documents referenced in, or related to, this document.

Table 2. Reference table

Reference	Document Name
[1]	CP-PDK Software Architecture Overview
[2]	CP-PDK Configuration and Management API Reference
[3]	CP-PDK Forwarding Plane Plug-in API Reference
[4]	CP-PDK Forwarding Plane Module Design Reference

1.3 Document Organization

This document contains the following sections:

1. Overview of the interaction between FP module of the CP-PDK and the IXA SDK core components.
2. Overview of design of the FP module ingress and egress core components
3. Interaction between the FP module ingress core component and the rest of the FP module
4. Interaction between the FP module core components and the IXA SDK egress components (L2 Table Manager and Interface Transmit core component).

Threading model, Algorithms and data structures for the FP module core components.

Part 2: Overview

2 Overview

CP-PDK 2.11 is specifically targeted for IXA SDK 3.51, designed to work on the IXP 2400 and 2800 network processors. Please refer to the appropriate IXA SDK documentation for details on the differences between IXA SDK 2.0 and IXA SDK 3.5.

This document describes the Forwarding Plane module (FP module) ingress and egress core components. These are part of CP-PDK 2.11 and are specific to IXA SDK 3.51; they reside on the ingress and egress sides of the IXP2400/2800 network processors. The FPM ingress CC interacts directly with the FP module of the CP-PDK. It also receives the following events/exceptions from the other SDK core components:

- **Link state change events** - Link status change events are generated internally by the SDK (the Interface Tx CC generates this & sends it across to the stack driver on the ingress) and are then sent up to the FP module ingress CC. From here, the status is propagated to the CP-PDK (via the FP module on ingress). Routing protocols such as OSPF are interested in this event to perform routing decisions (switching to an alternate route, and so on).
- **No Direct Route Host Route Exception** - This is required for supporting directly connected hosts. Transmit microblocks generate the No direct host route exception when they get a packet destined to a directly connected host, but there is no L2 information for the host. This exception is sent to the Interface Tx CC. FP module ingress CC receives this exception (through a message) and communicates this to the FP module, from where it goes to the Control Plane of the PDK. The IPv4 component (of the CP-PDK) then downloads a Direct Route for the directly connected host. Now packets can be sent to the directly connected host.

The egress core component interfaces with the L2 Table Manager of the IXA SDK to update the L2 Table with Next hop information propagated from the CP-PDK.

The egress core component interfaces with the Interface TX core component for the following:

- Adding/deleting ARP entries, purging ARP table
- Retrieve transmit statistics for Ethernet, ATM and POS.

2.1 Overview of Core Component

One of the main components of the IXA SDK is the core component. The core component is a packet processing entity. The PDK interfaces with several IXA SDK core components – IPv4 core component (CC), Stack driver, Interface CC, and so on.

The core component is a user-defined component that is required to export a set of functions:

- An initialization function
- A termination function
- One or more packet handlers
- One or more message handlers

This CC need not perform initialization of any microblocks. The initialization function of the CC is invoked by the Execution Engine it is running in, when the Execution Engine starts up.

2.1.1 Inputs and Outputs

Each core component can have multiple inputs, each of which is associated with a different packet/message handler. Each input is associated with a queue, and has a globally unique ID. Each core component registers a packet and/or message handler using a communication ID that is set at compile time in a header file (bindings.h).

Outputs are logical outputs that determine the flow of data through the system. core components could conceivably be configured without defining outputs; a packet handler could use the input ID of another core component directly in order to send it a packet. Typically, this is used for passing packets around in the system.

2.1.2 Execution Engine

An execution engine is the thread of execution that runs one or more core components. This is a kernel thread for Linux* and a task for VxWorks*. The developer of the system application controls how many execution engines there are and which core components run in each execution engine. The system requires the developer to write an initialization and termination function for each execution engine. The system provides an API function that spawns a thread/task/process to run the execution engine, invoking the user-defined initialization function.

2.1.3 Communication with Other Core Components

All IXA SDK core components support the following types of APIs:

- Messaging API
- Library API

Messaging API is for clients who communicate with the core components through the messaging mechanism of core component Infrastructure and the Message Support Library and, at the same time, want access to direct C-style functions. These API are provided with help of Message Helper Library. The Library API is for clients using only Framework (formerly Level 0 Framework) to access services of the core component. In that case, the core component can be used as C-library without support of core component infrastructure. It is assumed that clients using Library APIs have their own implementation of packet and message passing mechanism on the core.

All core components communicate using messages. Sending and receiving messages is simplified by using the Messaging Support Library.

2.2 FP Module and IXA SDK 3.51

A brief introduction about how core components work is required here to understand some of the design decisions that have been made in the PDK (especially the FP module) to work with the IXA SDK.

2.2.1 Execution Engines and Core Components

All core components execute in the context of an *Execution Engine*. A typical configuration would have an execution engine on the ingress IXP running all the core components (IPv4, Stack driver, ingress Rx, and so on). An execution engine is simply a thread of execution – it is a kernel thread in Linux and a task in VxWorks. Each core component is required to export a set of user-defined functions –initialization and termination functions, one or more packet handlers, one or more message handlers. The core component Infrastructure (CCI) provides support for handling packets and messages in the core components. It also provides an Execution Engine that can be fine-tuned depending on the system requirements.

The execution engine in which the IXA SDK core components execute must not block since IXA SDK core components process packets (exception packets such as IP packets with options, and so on.) and messages. Blocked callbacks might lead to the core component being blocked, which is not desirable. In order to facilitate this, the FP module callbacks must be in a different context.

The FP module will run on the ingress side on the XScale core. The Interface Transmit core component of the IXA SDK and the L2 Table Manager run on the egress side. In order to interact with these components, the FP module egress CC will run on the egress and interact with the FPM ingress CC. These components are shown in Figure 1. In the figure, there are two execution engines – this is a configuration issue and does not affect the design. All core components execute in the context of one of these Execution Engines. The FP module ingress CC could be configured to run in either execution engine.

Most of the APIs exposed by the core components are asynchronous in nature. Due to this, the FP module has to register callbacks that can report any error/status conditions, or responses for queries such as statistics, interface attributes, and so on. from the underlying core components. IXA SDK core components process these messages (the API calls reach the corresponding core component as a message) and return responses as messages. The reply message is converted to a callback in the Utility Execution Engine. Subsequently, the callback function registered by the client (in this case the FP module) gets invoked.

The figure also shows some details of how packets and messages flow in the system. Packets typically will just flow through the micro blocks, as indicated by the green arrows, unless these are exception packets that are sent to the corresponding core component for processing (for example, an IP packet with options set). The FP module components interact with the core components (both PR CCs such as IPv4, Stack Driver, as well as the FP module ingress CC). Some of the interactions are shown in the figure in blue arrows in the figure. The IPv4 Manager of the FP module, that is, the platform-specific component of the IPv4 Manager must interact with the IPv4 core component for adding routes. In addition to this, the IPv4 Manager must to populate the L2 and ARP Tables located on the egress side. In order to accomplish this, it interacts with the FP module ingress core component. The FP module ingress CC communicates with the corresponding FP module egress core component to interface with the L2 Table Manager and ARP modules. Similarly, the Configuration and Management Manager of the FP module (this module maps all NPF Interfaces APIs to the underlying device) has a platform specific component that interacts with the Stack Driver core component of the IXA SDK.

In addition to this, some events and exceptions from the egress processor have to be propagated to the FP module. One such event is the Link status event – this is propagated by the Interface Tx core component into the stack driver CC. From here, it reaches the FP module ingress CC as a property update. The Event Manager of the FP module receives a callback for this event, from where it makes it way into the rest of the PDK components on the Control Plane. In addition to this, the Interface Tx core component generates the No Direct Host Route Found Exception, as described earlier. This exception also propagates to the rest of the PDK through the FP module ingress CC.

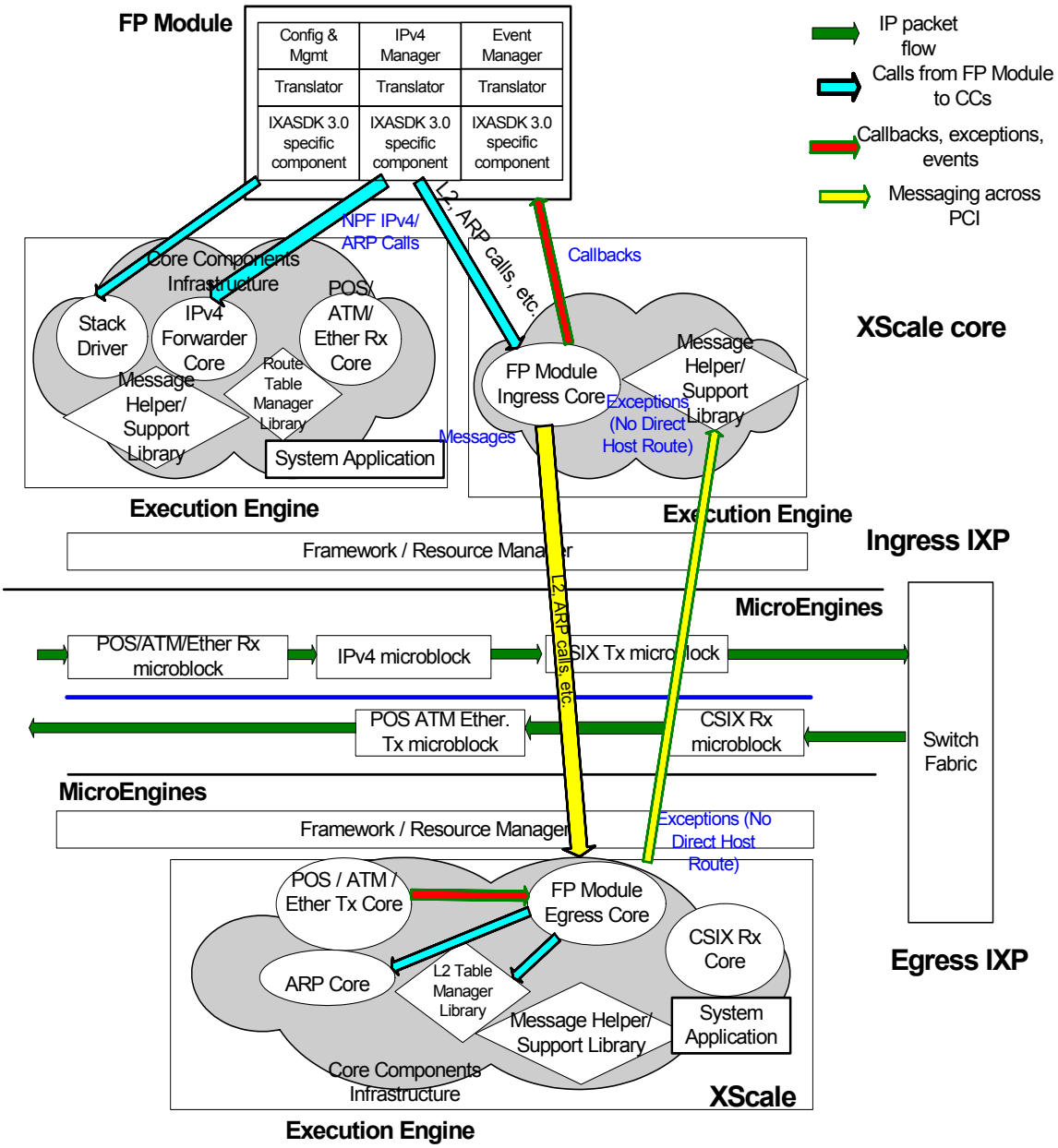

Figure 2: Overview of FP module Component Interaction with core components

2.3 Callbacks and the Message Support Library

This section gives a very brief overview of how callbacks work internally. For a more detailed description, examples, and so on, please refer to the Messaging Support Library information in the IXA SDK documentation. Each API call exposed by the core component has to internally use the Message Support Library to send messages to the correct core component. It will subsequently get

a reply message from the core component containing the result. The ingress core component has to receive the message and invoke the user callback. The Message Support Library only provides a generic callback function that passes the return message as a void pointer.

Each core component will have a specific callback type defined for each of their API calls. To support these specific callback functions, a generic callback function must exist internally to the message helper API for each call or group of calls. These callbacks of type **`ix_cc_msghlp_callback,`** reads the **void *** return message and makes a call to the specific API function passing appropriate arguments.

Internally, the ingress core component has to perform the following sequence of actions for each API call that it exposes (all the calls exposed by the ingress CC are asynchronous):

1. A Message-Helper function must perform several operations in order to implement a call to a Core-Component API function.
2. Register a message-handler function for the COMID defined (each core component is identified by a unique Communication ID, fixed at compile time) to receive API call messages. This function is called whenever a message is sent via the Message-Helper API.
3. Verify the validity of the arguments. If an error occurs, an ix_error should be returned.
4. Create a structure containing all of the arguments. If the call takes a structure as an argument, this step is not necessary.
5. Call the appropriate Message-Support function for the type of call, i.e. Synchronous, Asynchronous or Fire-and-forget.
6. Process post-call data. For Asynchronous and Fire-and-forget messages this means simply return any ix_error returned from the Message-Support function. For a Synchronous call, a return message is available when the Message-Support function returns, this data must be passed back to the caller of the Message-Helper function.

Each API call into the ingress core component works as shown in the Figure 3. As mentioned previously, the System Application of the IXA SDK starts the Utility Execution Engine. This execution engine is responsible for callbacks; the FP module ingress core component will also be executing in the context of this Execution Engine. The example below considers the NPF API to add an ARP entry. This results in making a call to the ingress Core component to add an ARP entry into the L2 Table.

1. IPv4 Manager's translator receives the API call to add an ARP entry.
2. The Translator calls into the platform-specific component with the NPF parameters.

   Note: It is to be noted that the actual process is a little more complicated since the calls into all the core components are asynchronous in nature. Please refer to the FP module Design Reference for more details.
3. The IPv4 Manager's platform-specific component makes a call into the FP module ingress core Component to add the ARP entry, passing in a callback function that is to be invoked when the result is ready.
4. The FPM ingress core component validates the parameters, creates a message and uses the Message Support Library to send a message to the egress core component. It additionally registers an internal callback with the Messaging library. This callback is invoked later when the reply message arrives.
5. The result of the ADD ARP ENTRY is not ready yet, so the API call into the FPM ingress core component returns back into the FP module.

6. At some later point of time, the result from the ADD ARP ENTRY arrives at the Utility Execution Engine.
7. The Messaging Library invokes the internal callback of the FP module ingress core component. This function then invokes the user-registered callback with the result of the ADD ARP ENTRY.

It is to be noted that the user callback executes in the context of the Execution Engine. One limitation is that all callbacks from this EE is serialized and the PDK can get only one callback at a time.

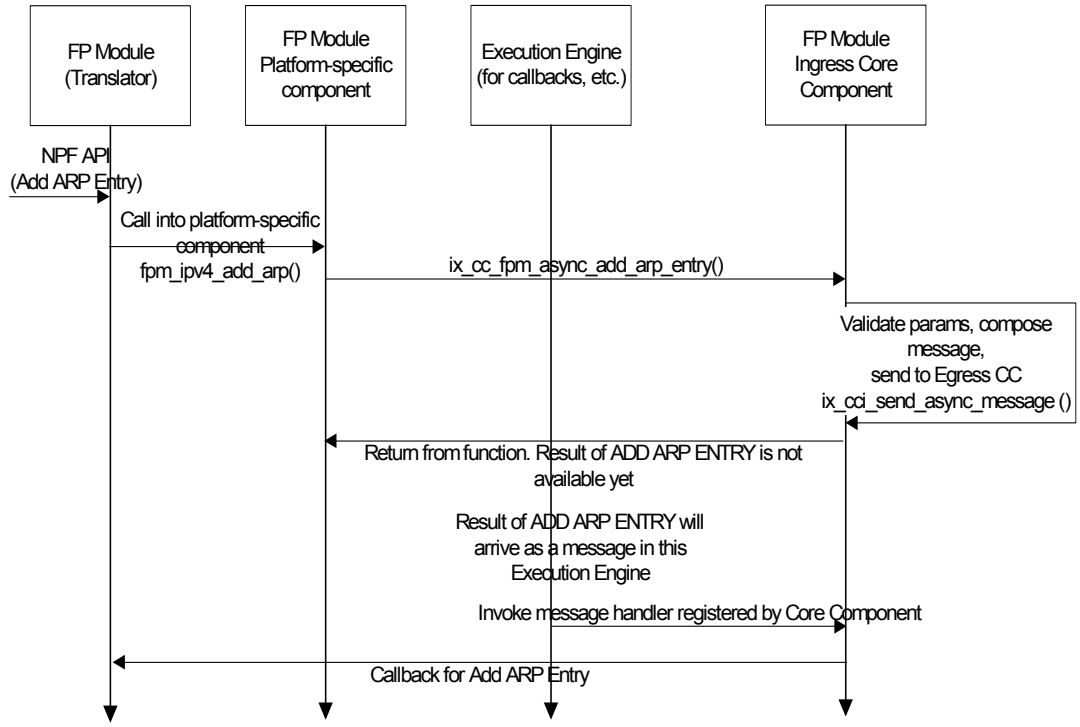

Figure 3: Callbacks from the FP module Execution Engine

2.4 Design Considerations/Assumptions and Dependencies

As mentioned earlier, most of the APIs exposed by the core components are asynchronous in nature. Due to this, the FP module has to register callbacks with the underlying CCs. However, one constraint imposed by the IXA SDK is that callbacks from the CCs must not block. In order to facilitate this, the FP module callbacks must be in a different context. There are several ways this could be done:

1. A separate Execution Engine could be created during initialization and this thread of execution could be used for making all callbacks.

2. The FP module could create a simple core component that would execute in the same context as the other CCs (IPv4, Stack driver, and so on). When a message arrived, this CC would spawn a callback (in another thread, for example). Alternatively, this CC could use a pool of callback threads and spawn callbacks in different threads.

For reasons of simplicity, we have chosen Option 1 and a separate Execution Engine created by the IXA SDK System Application (refer to the appropriate IXA SDK documentation for more details) during initialization delivers all callbacks.

2.5 Initialization

The FP module ingress and egress core components have to be started up along with the other IXA SDK core components. By the time FP module is started up, all Execution Engines & all core components would be initialized.

Part 3: FP Module Ingress Core Component

3 FP Module Ingress Core Component

3.1 Requirements

The basic requirements of the FP module ingress CC are as follows:

1. It should be a core component as required by the IXA SDK. It has to expose all the interfaces/APIs required by the core components. It is designed and implemented in a manner similar to other IXA SDK core components and hence is compliant in a system with Level-0 framework alone also.
2. It must expose a simple API that can be used from the FP module Managers. The API must be similar to the APIs exposed by the other IXA SDK core components such as IPv4, Stack driver, and so on.

3.2 Dependencies

core component Infrastructure (CCI): provides set of API for defining thread of execution for the CC, initialization and termination of CC and provides mechanism for component to component (inter-process and inter-task) packet and message handling.

Message Support Library – this provides a set of API for communicating with other core components in the system.

- FP module egress core component
- L2 Table Manager
- Ethernet Transmit Core component.

3.3 FP Module Core Component APIs

The subsequent sections describe in detail the API exposed by the FP module ingress core component. The clients of the FP module core component (for example, the IPv4 Manager of the PDK FP module) use only the Message Helper APIs. These are all asynchronous and synchronous functions, named as **ix_cc_fpm_async_xxx** or **ix_cc_fpm_sync_xxx**, where xxx signifies the different operations. Also, there are no asynchronous functions for the events (Link status change, No direct host route exception), instead there are registration and de-registration functions so that FP module can register for these events. The rest of the functions – Initialization, Shutdown, message handler, packet handler, and so on. are all used internally.

3.3.1 Data Structures

These data structures are defined by other IXA SDK core components and used here.

ix_cc_eth_tx_next_hop_info

```
typedef struct ix_cc_s_eth_tx_next_hop_info
{
ix_int32 l2Index;             /* L2 id */
ix_ip_addr nextHopIp;   /* next hop IP in network byte order */
ix_uint16 outputPort;   /* egress output port in network byte
order */
ix_l2_addr l2Addr;            /* layer-2 address */
} ix_cc_eth_tx_next_hop_info;
```

ix_l2_addr

```
typedef struct ix_s_l2_addr
{
ix_uint8 etherDstAddr[6]; /* Ethernet address in network byte
order */
} ix_l2_addr;
```

ix_cc_fpm_link_status

These structure reports changes in the link status of a physical port.

```
typedef enum ix_cc_fpm_link_status
{
      IX_CC_FPM_LINK_STATUS_ON,
      IX_CC_FPM_LINK_STATUS_OFF
};
```

ix_cc_fpm_dpe_op

This enum defines the DPE operations.

```
typedef enum ix_s_fpm_dpe_op
{
    IX_CC_FPM_DPE_ADD,        /* Add DPE entry */
    IX_CC_FPM_DPE_UPDATE,     /* Update DPE Entry */
    IX_CC_FPM_DPE_REMOVE,     /* Delete DPE Entry */
    IX_CC_FPM_DPE_STATS       /* Get DPE statistics */
}ix_cc_fpm_dpe_op;
```

ix_cc_fpm_dpe_type

This enum defines the DPE types.

```
typedef enum ix_s_fpm_dpe_type
{
    IX_CC_FPM_DPE_DSCP_CLASSIFIER,
    IX_CC_FPM_DPE_METER,
    IX_CC_FPM_DPE_MARKER,
    IX_CC_FPM_DPE_WRED_DROPPER
}ix_cc_fpm_dpe_type;
```

ix_cc_fpm_dpe

This structure defines the structure for DPE operations. This structure is passed across the message support library between the ingress and egress FP module core components.

```
typedef struct ix_s_cc_fpm_ix_cc_fpm_dpe
{
    ix_cc_fpm_dpe_op   op;      /*Add, Update, Remove, statistics
*/
    ix_cc_fpm_dpe_type type;    /* DPE type */
    ix_uint32          fid;     /* Flow Id */
    ix_uint16          port;    /* Port Id */
    ix_uint8           dscp;    /* Port Id */
    union
    {
        ix_cc_tc_meter_parameters  tc_meter; /* tc-meter
parameters */
        ix_cc_wred_parameters      wred_dropper; /* WRED dropper
parameters */
        ix_cc_wred_statistics      wred_stats; /* WRED statistics
*/
        ix_cc_tc_meter_statistics  meter_stats; /* tc-meter
statistics */
    }dpe;
} ix_cc_fpm_dpe;
```

ix_cc_fpm_dpe_callback_data

```
typedef struct ix_s_cc_fpm_dpe_callback_data
{
    ix_cc_fpm_dpe_cb                        pUserCallback;
    ix_cc_fpm_dpe_wred_dropper_stats_cb
      p_wred_dropper_stats_callback;
    ix_cc_fpm_dpe_meter_stats_cb            p_meter_stats_callback;
    void                                    *pUserContext;
    ix_error               error;       /* Error of DPE operation
response */
    ix_cc_fpm_dpe          dpe;
} ix_cc_fpm_dpe_callback_data;
```

ix_cc_fpm_atm_port_descr_t

This data type describes the attributes of a physical ATM port.

```
typedef struct ix_cc_fpm_atm_port_descr
{
ix_cc_atmsar_instance_t arg_sarInstance; /* ATMSAR instance */
ix_cc_atmsar_port_descr_t arg_pPortDescr; /* ATM Port descriptors
*/
} ix_cc_fpm_atm_port_descr_t;
```

ix_cc_fpm_atm_port_create_cb_t

This data type stores the callback function for the ATM port create call.

```
typedef struct ix_cc_fpm_atm_port_create_cb
```

```
{
ix_cc_atmsar_cb_port_create_t arg_Callback; /* Call back Function
*/
void *arg_pUserContext; /* User Context */
} ix_cc_fpm_atm_port_create_cb_t;
```

ix_cc_fpm_atm_port_create_resp_t

This data type stores the status/response of the ATM port create call.

```
typedef struct ix_cc_fpm_atm_port_create_resp
{
ix_error arg_Callback; /* Call back Function */
void *arg_pUserContext; /* User Context */
} ix_cc_fpm_atm_port_create_resp_t;
```

ix_cc_fpm_atm_port_handle_t

This data type describes the handle of a physical ATM port.

```
typedef struct ix_cc_fpm_atm_port_handle
{
ix_cc_atmsar_instance_t arg_sarInstance; /* ATM SAR instance */
ix_cc_atmsar_port_handle_t arg_port; /* ATM port Handle */
} ix_cc_fpm_atm_port_handle_t;
```

ix_cc_fpm_atm_port_del_cb_t

This data type stores the callback function for the ATM port delete call function.

```
typedef struct ix_cc_fpm_atm_port_del_cb
{
ix_cc_atmsar_cb_port_remove_t arg_Callback; /* Call back function
*/
void *arg_pUserContext; /* User Context */
} ix_cc_fpm_atm_port_del_cb_t;
```

ix_cc_fpm_atm_port_stats_get_cb_t

This data type stores the callback function for the ATM port statistics get function.

```
typedef struct ix_cc_fpm_atm_port_stats_get_cb
{
ix_cc_atmsar_cb_get_port_stats_t arg_Callback; /* Call back
Function */
void *arg_pUserContext; /* User Context */
} ix_cc_fpm_atm_port_stats_get_cb_t;
```

ix_cc_fpm_atm_port_stats_get_resp_t

This data type stores the status/response of the ATM port statistics get function.

```
typedef struct ix_cc_fpm_atm_port_stats_get_resp
{
ix_error arg_Result; /* Status/Result of the statistics get call
*/
```

```
ix_cc_atmsar_port_stats_t arg_pLookupResult; /* Port Statistics */
} ix_cc_fpm_atm_port_stats_get_resp_t;
```

ix_cc_fpm_atm_vc_descr_t

This data type describes the attributes of an ATM VCC.

```
typedef struct ix_cc_fpm_atm_vc_descr
{
ix_cc_atmsar_instance_t arg_sarInstance; /* ATMSAR instance */
ix_cc_atmsar_vc_descr_t arg_pVcDescr; /* VCC attributes descriptor
*/
} ix_cc_fpm_atm_vc_descr_t;
```

ix_cc_fpm_atm_vc_create_cb_t

This data type stores the callback function for the ATM VCC create function call.

```
typedef struct ix_cc_fpm_atm_vc_create_cb
{
ix_cc_atmsar_cb_vc_create_t arg_Callback; /* Call back Function */
void *arg_pUserContext; /* User Context */
} ix_cc_fpm_atm_vc_create_cb_t;
```

ix_cc_fpm_atm_vc_create_resp_t

This data type stores the status/response of the previous ATM VCC create call.

```
typedef struct ix_cc_fpm_atm_vc_create_resp
{
ix_error arg_Result; /* Result of the VCC Create request */
ix_cc_atmsar_vc_handle_t arg_hVc; /* Handle of the created VCC */
} ix_cc_fpm_atm_vc_create_resp_t;
```

ix_cc_fpm_atm_vc_update_t

This data type describes the parameters used in the ATM VC update call.

```
typedef struct ix_cc_fpm_atm_vc_update
{
ix_cc_atmsar_instance_t arg_sarInstance; /* ATMSAR instance */
ix_cc_atmsar_vc_handle_t arg_hVc; /* VCC handle */
ix_cc_atmsar_update_data_t arg_pUpdateData; /* VC Update data */
} ix_cc_fpm_atm_vc_update_t;
```

ix_cc_fpm_atm_vc_update_cb_t

This data type stbores the callback function for the ATM VCC update function call.

```
typedef struct ix_cc_fpm_atm_vc_update_cb
{
ix_cc_atmsar_cb_vc_update_t arg_Callback; /* Call back Function */
void *arg_pUserContext; /* User Context */
} ix_cc_fpm_atm_vc_update_cb_t;
```

ix_cc_fpm_atm_vc_handle_t

This data type describes the attributes of an ATM VC handle.

```
typedef struct ix_cc_fpm_atm_vc_handle
{
ix_cc_atmsar_instance_t arg_sarInstance; /* ATMSAR instance */
ix_cc_atmsar_vc_handle_t arg_hVc; /* VC Handle */
} ix_cc_fpm_atm_vc_handle_t;
```

ix_cc_fpm_atm_vc_remove_cb_t

This data type stores the callback function for the ATM VCC delete function call.

```
typedef struct ix_cc_fpm_atm_vc_remove_cb
{
ix_cc_atmsar_cb_vc_remove_t arg_Callback; /* Call back Function */
void *arg_pUserContext; /* User Context */
} ix_cc_fpm_atm_vc_remove_cb_t;
```

ix_cc_fpm_atm_vc_stats_get_cb_t

This data type stores the callback function for the ATM VCC statistics get function call.

```
typedef struct ix_cc_fpm_atm_vc_stats_get_cb
{
ix_cc_atmsar_cb_get_vc_stats_t arg_Callback; /* Call back Function
*/
void *arg_pUserContext; /* User Context */
} ix_cc_fpm_atm_vc_stats_get_cb_t;
```

ix_cc_fpm_atm_vc_stats_get_resp_t

This data type stores the status/response of the ATM VC statistics get call.

```
typedef struct ix_cc_fpm_atm_vc_stats_get_resp
{
ix_error arg_Result; /* Status/result of the VCC statistics get
call */
ix_cc_atmsar_vc_stats_t arg_pLookupResult; /* VC statistics */
} ix_cc_fpm_atm_vc_stats_get_resp_t;
```

ix_cc_fpm_llc_snap_encap_add_next_hop_t

This data type describes the Attributes of a next Hop used in case of Ip over ATM.

```
typedef struct ix_cc_fpm_llc_snap_encap_add_next_hop
{
ix_uint16 nextHop;
ix_uint8 encapBit;
ix_uint16 vpi;
ix_uint16 vci;
ix_uint8 dSap;
ix_uint8 sSap;
ix_uint8 ctrl;
ix_uint32 oui;
```

```
ix_uint16 pid;
} ix_cc_fpm_llc_snap_encap_add_next_hop_t;
```

ix_cc_fpm_llc_snap_encap_add_next_hop_cb_t

This data type stores the callback function for the LLC Snap Add next hop entry function call.

```
typedef struct ix_cc_fpm_llc_snap_encap_add_next_hop_cb
{
ix_cc_llc_snap_encap_cb_add_next_hop pCallback; /* Call back
Function */
void *pUserContext; /* User Context */
} ix_cc_fpm_llc_snap_encap_add_next_hop_cb_t;
```

3.3.2 Error Codes

ix_cc_fpm_error_code

```
typedef enum ix_cc_fpm_error_code
{
IX_ERROR_CC_FPM_ERROR,
IX_ERROR_CC_FPM_ERROR_CCI,
IX_ERROR_CC_FPM_INVALID_INPUT_PARAM,
IX_ERROR_CC_FPM_INVALID_INPUT_CONTEXT,
IX_ERROR_CC_MSG_LIBRARY,
IX_ERROR_CC_FPM_L2TM
};
```

3.3.3 Messages

The FP module ingress CC registers a generic message handler during initialization. core components communicate by sending messages to each using (possibly by using the Message Support Library). The FP module ingress core component receives and interprets the following messages. For L2 and ARP operations, the FP module ingress CC sends the respective messages to the FPM egress CC, where the message handler invokes the corresponding library function to perform the L2/ARP operation. Results from these operations arrive back as messages and the Message Support Library invokes the corresponding L2/ARP callback. Subsequently, user callbacks are invoked. Events/exception messages behave slightly differently. The link status update message reaches the FP module ingress CC as a property update and a client (who must have registered a callback for this earlier) will then get notified. In the same way, the Direct Host Route exception arrives at the FP module ingress CC as a message, and a user callback (registered earlier) gets invoked.

Table 6. Messages description table

Message	Description
IX_CC_FPM_MSG_ADD_L2_ENTRY	This message helps client to add a next hop into L2 Table maintained on egress.
IX_CC_FPM_MSG_DELETE_L2_ENTRY	This message helps client to delete a next hop from L2

	Table maintained on egress.
IX_CC_FPM_MSG_ADD_ARP_ENTRY	This message helps client to add an ARP entry into the ARP module maintained on egress.
IX_CC_FPM_MSG_DELETE_ARP_ENTRY	This message helps client to delete an ARP entry from the ARP module maintained on egress.
IX_CC_FPM_MSG_PURGE_ARP_TABLE	This message helps client to purge the ARP table from the ARP module maintained on egress.
IX_CC_FPM_COMM_MSG_ID_PROP_UPDATE	This is a property update message that conveys a link status change for a port in the system.
IX_NEW_DIRECT_HOST_MSG	This message originates on the egress (Interface CC) and is delivered to the client (IPv4 of FP module) reporting an exception raised by the transmit block.
IX_CC_FPM_MSG_ETH_TX_STATS	This message retrieves the Ethernet Transmit statistics from the egress Interface transmit component.
IX_CC_FPM_MSG_ATM_POS_TX_STATS	This message retrieves the ATM Transmit statistics from the egress Interface transmit component.
IX_CC_FPM_MSG_ATM_PORT_CREATE	This message helps the client to create an ATM port. This message is needed as currently the ATMSAR MAIN CC resides on the Egress.
IX_CC_FPM_MSG_ATM_PORT_DELETE	This message helps the client to delete/remove an ATM port. This message is needed as currently the ATMSAR MAIN CC resides on the Egress.
IX_CC_FPM_MSG_ATM_PORT_STATS_GET	This message helps the client to get the statistics of an ATM port. This message is needed as currently the ATMSAR MAIN CC resides on Egress.
IX_CC_FPM_MSG_ATM_VCC_CREATE	This message helps the client to create an ATM VCC. This message is needed as currently the ATMSAR MAIN CC resides on the Egress.
IX_CC_FPM_MSG_ATM_VCC_UPDATE	This message helps the client to update an ATM VCC. This message is needed as currently the ATMSAR MAIN CC resides on the Egress.
IX_CC_FPM_MSG_ATM_VCC_DELETE	This message helps the client to delete an ATM VCC. This message is needed as currently the ATMSAR MAIN CC resides on the Egress.
IX_CC_FPM_MSG_ATM_VCC_STATS_GET	This message helps the client to get statistics of an ATM VCC. This message is needed as currently the ATMSAR MAIN CC resides on the Egress.
IX_CC_FPM_MSG_LLC_SNAP_ENCAP_ADD_NEXT_HOP	This message helps the client to add the next hop ID in case of IP over ATM.
IX_CC_FPM_MSG_DPE_OP	This message is passed between the ingress and the egress to perform DPE operations.

3.3.4 Callback Functions

The FP module ingress core component defines these callbacks. As described earlier, the FP module makes all L2, ARP and statistics calls into the FP module ingress core component. These callbacks are defined for each operation that is supported by the FP module core component. The callbacks are defined only for the asynchronous calls exposed by the FP module core component.

3.3.4.1 ix_cc_fpm_arp_op_cb

Message Support invokes this callback function when a reply arrives from the FPM egress CC after completing an ARP operation. This function interprets the reply message and invokes the user callback function with the required result/response data of the operation.

ix_cc_fpm_arp_op_cb

```
ix_error (*ix_cc_fpm_arp_op_cb) (
ix_error    arg_Result,
void*       arg_pContext
);
```

3.3.4.2 ix_cc_fpm_eth_tx_stats_op_cb

Message Support invokes this callback function when a reply arrives from the FPM egress CC after retrieving Ethernet Transmit statistics. This function interprets the reply message and invokes the user callback function with the required result/response data of the operation.

ix_cc_fpm_eth_tx_stats_op_cb

```
ix_error (*ix_cc_fpm_eth_tx_stats_op_cb) (
ix_error    arg_Result,
void*       arg_pContext,
ix_uint64*  arg_pBuffer
);
```

3.3.4.3 ix_cc_fpm_atm_pos_tx_stats_op_cb

Message Support invokes this callback function when a reply arrives from the FPM egress CC after retrieving POS/ATM transmit statistics. This function will interpret the reply message and invoke the user callback function with the required result/response data of the operation.

ix_cc_fpm_atm_pos_tx_stats_op_cb

```
ix_error (*ix_cc_fpm_atm_pos_tx_stats_op_cb) (
ix_error    arg_Result,
void*       arg_pContext,
ix_uint64*  arg_pBuffer
);
```

3.3.4.4 ix_cc_fpm_link_status_op_cb

The FP module ingress CC receives the link status update message as a property update. A client of the FPM ingress CC must have registered a callback earlier in order to receive this notification. The signature of the callback function is given here.

ix_cc_fpm_link_status_op_cb

```
ix_error (*ix_cc_fpm_link_status_op_cb) (
ix_uint32               arg_PortId,
ix_cc_fpm_link_status   arg_PortStatus
);
```

3.3.4.5 ix_cc_fpm_host_route_exception_op_cb

The FP module ingress CC receives the No Direct Host Route exception as a message from the Interface Transmit CC. A client of the FPM ingress CC must have registered a callback earlier in order to receive this notification. The signature of the callback function is given here.

ix_cc_fpm_host_route_exception_op_cb

```
ix_error (*ix_cc_fpm_host_route_exception_op_cb) (
ix_uint32   arg_DestIP,
ix_uint32   arg_PortID
);
```

3.3.4.6 ix_cc_llc_snap_encap_fpm_add_next_hop_cb_core

Message Support invokes this callback function when a reply arrives from the FPM egress CC after adding an LLC SNAP next hop entry for IP over ATM case. This function interprets the reply message and invokes the user callback function with the required result/response data of the operation.

ix_cc_llc_snap_encap_fpm_add_next_hop_cb_core

```
ix_error (*ix_cc_llc_snap_encap_fpm_add_next_hop_cb_core) (
ix_error    arg_Result,
void*       arg_pContext,
void*       arg_pMsg,
ix_uint32   arg_MsgLen
);
```

3.3.4.7 ix_cc_fpm_atm_port_create_cb_core

Message Support invokes this callback function when a reply arrives from the FPM egress CC after creating an ATM port. This function interprets the reply message and invokes the user callback function with the required result/response data of the operation.

ix_cc_fpm_atm_port_create_cb_core

```
ix_error (*ix_cc_fpm_atm_port_create_cb_core) (
```

```
ix_error    arg_Result,
void*       arg_pContext,
void*       arg_pMsg,
ix_uint32   arg_MsgLen
);
```

3.3.4.8 ix_cc_fpm_atm_port_del_cb_core

Message Support invokes this callback function when a reply arrives from the FPM egress CC after deleting an ATM port. This function interprets the reply message and invokes the user callback function with the required result/response data of the operation.

ix_cc_fpm_atm_port_del_cb_core

```
ix_error (*ix_cc_fpm_atm_port_del_cb_core) (
ix_error    arg_Result,
void*       arg_pContext,
void*       arg_pMsg,
ix_uint32   arg_MsgLen
);
```

3.3.4.9 ix_cc_fpm_atm_port_stats_get_cb_core

Message Support invokes this callback function when a reply arrives from the FPM egress CC after getting statistics of an ATM port. This function will interpret the reply message and invoke the user callback function with the required result/response data of the operation.

ix_cc_fpm_atm_port_stats_get_cb_core

```
ix_error (*ix_cc_fpm_atm_port_stats_get_cb_core) (
ix_error    arg_Result,
void*       arg_pContext,
void*       arg_pMsg,
ix_uint32   arg_MsgLen
);
```

3.3.4.10 ix_cc_fpm_atm_vc_create_cb_core

Message Support invokes this callback function when a reply arrives from the FPM egress CC after creating an ATM VC. This function interprets the reply message and invokes the user callback function with the required result/response data of the operation.

ix_cc_fpm_atm_vc_create_cb_core

```
ix_error (*ix_cc_fpm_atm_vc_create_cb_core) (
ix_error    arg_Result,
void*       arg_pContext,
void*       arg_pMsg,
ix_uint32   arg_MsgLen
);
```

3.3.4.11 ix_cc_fpm_atm_vc_update_cb_core

Message Support invokes this callback function when a reply arrives from the FPM egress CC after updating an ATM VC. This function interprets the reply message and invokes the user callback function with the required result/response data of the operation.

ix_cc_fpm_atm_vc_update_cb_core

```
ix_error (*ix_cc_fpm_atm_vc_update_cb_core) (
ix_error    arg_Result,
void*       arg_pContext,
void*       arg_pMsg,
ix_uint32   arg_MsgLen
);
```

3.3.4.12 ix_cc_fpm_atm_vc_remove_cb_core

Message Support invokes this callback function when a reply arrives from the FPM egress CC after deleting an ATM VC. This function interprets the reply message and invokes the user callback function with the required result/response data of the operation.

ix_cc_fpm_atm_vc_remove_cb_core

```
ix_error (*ix_cc_fpm_atm_vc_remove_cb_core) (
ix_error    arg_Result,
void*       arg_pContext,
void*       arg_pMsg,
ix_uint32   arg_MsgLen
);
```

3.3.4.13 ix_cc_fpm_atm_vc_stats_get_cb_core

Message Support invokes this callback function when a reply arrives from the FPM egress CC after getting statistics of an ATM VC. This function interprets the reply message and invokes the user callback function with the required result/response data of the operation.

ix_cc_fpm_atm_vc_stats_get_cb_core

```
ix_error (*ix_cc_fpm_atm_vc_stats_get_cb_core) (
ix_error    arg_Result,
void*       arg_pContext,
void*       arg_pMsg,
ix_uint32   arg_MsgLen
);
```

ix_cc_fpm_dpe_callback

This is the async message callback function for DIffServ egress CC Api.

3.4 Initialization

`ix_cc_fpm_init()`

Syntax

```
ix_error ix_cc_fpm_init(
ix_cc_handle      arg_hFPMCCHandle,
void**            arg_ppContext);
```

Description

This function is the initialization function of the core component and must be invoked by the Execution Engine in which it is running. It must be invoked once or the behavior is undefined. This function is responsible for performing the basic initializations of the core component such as registering message and/or packet handlers, and so on. This function must be called before requesting any service before the core component. Since this core component is not responsible for any micro blocks, there is no need for patching symbols, initializing micro block memory, and so on.

On the egress side, this function also creates the L2 Table Manager and stores the corresponding table handle into the context. This is used during shutdown to terminate L2 Table Manager services.

Input Parameters

`arg_hFPMCCHandle` handle to FP module core component, created by core component infrastructure; this shall be used later to get other (to add event handler) services from core component infrastructure.

Output Parameters

`arg_ppContext` location where the pointer to the control block allocated by the core component is stored. The control block is internal to the core component and contains variables and internal data structures. This pointer is use later to be passed into the **`ix_cc_fpm_fini()`** function by core component infrastructure for memory freeing when the core component is being destroyed.

Return Values

`IX_SUCCESS` if the initialization is succees, else a valid ix_error.

Error Codes

`IX_CC_ERROR_NULL`	null input parameter
`IX_CC_ERROR_OOM_SYSTEM`	memory allocation failure
`IX_ERROR_CC_FPM_CCI`	failure from CCI

Logic Description

1. Validity check for Parameters

- If arg_ppContex is null, then return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocation of memory for FPM CC Context

- Call ix_ossl_malloc() for memory allocation of ix_cc_fpm_context. If it fails, then return IX_ERROR with error code IX_CC_ ERROR_OOM
- Store FP module core component handle in global variable p_fpm_cc_Context

3. If this is on the egress, initialize L2 Table Manager

- Call ix_cc_l2tm_create() to initialize L2 Table Manager. If this fails, free memory allocated for ix_cc_fpm_context; return IX_ERROR with error code IX_ERROR_CC_FPM_L2TM.
- Store L2 Table Manager handle in ix_cc_fpm_context.

4. Registration of message handler

- Call ix_cci_add_message_handler() to register FPM CC handler. If it fails, then return IX_ERROR with error code IX_ERROR_CC_FPM_CCI
- If there is no CCI, then call ix_rm_message_handler_register() to register FPM CC message handler. If the call fails, then return IX_ERROR with error code IX_ERROR_CC_FPM_CCI

5. Return IX_SUCCESS

3.5 Shutdown

```
ix_cc_fpm_fini()
```

Syntax

```
ix_error ix_cc_fpm_fini(
ix_cc_handle      arg_hFPMCCHandle,
void*             arg_pContext);
```

Description

This function is the termination function of the core component and must be invoked by the Execution Engine it is running in. It must be invoked exactly once. This function is responsible for cleaning up all memory allocated by the core component, deregistering message and/or packet handlers, and so on. On the egress side, this function terminates the services of the L2 Table Manager.

Input Parameters

```
arg_hFPMCCHandle  handle to FP module core component
arg_pContext            pointer to the control block allocated
earlier in ix_cc_fpm_init function
```

Return Values

```
IX_SUCCESS        if the shutdown is success else a valid ix_error
```

Error Codes

```
IX_CC_ERROR_NULL        null input parameter
IX_CC_ERROR_OOM_SYSTEM             memory allocation failure
IX_CC_ERROR_CCI              failure from CCI
```

Logic Description

1. Validity check for Parameters

- If arg_pContext is null, then return IX_ERROR with the error code IX_CC_ERROR_NULL
- If arg_pContext is not context of FPM core component, return IX_ERROR with the error code IX_ ERROR_CC_FPM_INVALID_INPUT_PARAM
- If arg_hFPMCCHandle does not match the FPM core component handle stored in p_fpm_cc_context, return IX_ERROR with the error code IX_ERROR_CC_FPM_INVALID_INPUT_PARAM

2. On the egress, terminate the L2 Table Manager services

- Call ix_cc_l2tm_fini with L2 Table Handle stored in ix_cc_fpm_context. If this fails, set IX_ERROR with the error code IX_ERROR_CC_FPM_L2TM.

3. Remove the message handler

- If CCI is present, call ix_cci_cc_remove_message_handler with FPM CC Id. If this fails, return IX_ERROR with error code IX_ERROR_CC_FPM_CCI.
- If there is no CCI, call ix_rm_remove_message_handler with the FPM CC ID. If this fails, return IX_ERROR with the error code IX_ERROR_CC_FPM_RM.

4. Return IX_SUCCESS

3.6 Message Handler

`ix_cc_fpm_message_handler()`

Syntax

```
ix_error ix_cc_fpm_message_handler(
ix_buffer_handle  arg_hDataToken,
ix_uint32         arg_UserData,
void*                   arg_pComponentContext);
```

Description

This primitive is the message handler function for FP module core component. This function is invoked by the CCI when a message arrives for the FP module ingress/egress core component. The FPM ingress/egress CC then internally calls the appropriate library function to process the message. Communication between client and core components for message passing and retrieve result is described in Messaging API and Support Library section of the IXA SDK documentation.

Input Parameters

```
arg_hDataToken          buffer handle embedding information for
the message passed in arg_UserData
arg_UserData                    Message type. The supported messages
are as defined in 3.3.3.
arg_pComponentContext   pointer to FP module core component
context that is passed to the core component when a message
arrives. This context was defined by the core component and passed
to core components infrastructure through the ix_cc_fpm_init
function.
```

Return Values

IX_SUCCESS	if success, else a valid ix_error for failure

Error Codes

IX_CC_ERROR_NULL	null input parameter
IX_CC_ERROR_OOM	memory allocation failure
IX_ERROR_CC_FPM_CCI	failure from CCI
IX_ERROR_CC_MSG_LIBRARY	failure from CCI
IX_ERROR_CC_FPM_UNDEFINED_MSG	failure from CCI
IX_ERROR_FPM_CC_INVALID_INPUT_PARAM	failure from CCI

Logic Description

1. Validity check for Parameters

- If arg_pComponentContext is null, then return IX_ERROR with error code IX_CC_ERROR_NULL
- If arg_pComponentContext is not context of FPM core component, return IX_ERROR with error code IX_ERROR_CC_FPM_INVALID_INPUT_CONTEXT

2. Validity check for message type

- If arg_UserData does not contain one of the messages supported by the FPM core component, then return IX_ERROR with error code IX_ERROR_CC_FPM_UNDEFINED_MSG

3. Retrieve data from the buffer

- Call IX_MSUP_EXTRACT_MSG (this function is exported by Message Support Library) with parameter arg_hDataToken, to retrieve message and return context.
- If the buffer is null, return IX_ERROR with error code IX_CC_ERROR_NULL

4. Invoke Library API (on the ingress side, all functions except the events and exception handlers are no-ops – on the egress, these functions interact with the L2 TM and Interface Tx CC).

- Call appropriate library function with the retrieved data
    - If arg_UserData == IX_CC_COMMON_MSG_ID_PROP_UPDATE, then call ix_cc_fpm_on_link_status_change. This should happen only on the ingress.
    - If arg_UserData == IX_CC_FPM_MSG_ADD_L2_ENTRY, then call ix_cc_fpm_add_l2_entry. This should happen only on the egress.
    - If arg_UserData == IX_CC_FPM_MSG_DELETE_L2_ENTRY, then call ix_cc_fpm_delete_l2_entry. This should happen only on the egress.
    - If arg_UserData == IX_CC_FPM_MSG_ADD_ARP_ENTRY, then call ix_cc_fpm_add_arp_entry. This should happen only on the egress.
    - If arg_UserData == IX_CC_FPM_MSG_DELETE_ARP_ENTRY, then call ix_cc_fpm_delete_arp_entry. This should happen only on the egress.
    - If arg_UserData == IX_CC_FPM_MSG_PURGE_ARP_TABLE, then call ix_cc_fpm_purge_arp_table. This should happen only on the egress.

- If arg_UserData == IX_CC_FPM_NO_DIRECT_HOST, then call ix_cc_fpm_report_ipv4_exception. This should happen only on the ingress.
- If arg_UserData == IX_CC_FPM_MSG_ETH_STATS, then call ix_cc_fpm_l2_eth_stats. This should happen only on the egress.
- If arg_UserData == IX_CC_FPM_MSG_ATM_POS_STATS, then call ix_cc_fpm_l2_atm_stats. This should happen only on the egress.

5. Send data to message support library

- If a reply must be sent, call ix_cc_msup_send_reply_msg with the retrieved context, return data, and return message length.
- If error, return IX_ERROR with the return code IX_ERROR_CC_MSG_LIBRARY.

6. Free arg_hDataToken (ix_buffer).
7. Return IX_SUCCESS.

3.7 Message Helpers

3.7.1 ix_cc_fpm_sync_add_l2_entry

```
ix_cc_fpm_sync_add_l2_entry()
```

Syntax

```
ix_error ix_cc_fpm_sync_add_l2_entry (
            ix_uint32   arg_L2ID,
            ix_uint32   arg_NextHopIP,
            ix_uint32   arg_OutputPort
);
```

Description

This function helps the client to add a L2 entry into the L2 Table Manager maintained on the egress. The FP module of the CP-PDK invokes this function.

Input Parameters

```
arg_L2ID                L2 ID to be added into the L2 Table.
arg_NexthopIP                 Next Hop IP address for the L2 ID.
arg_OutputPort                Output port to which the specified
Next Hop IP address is attached.
```

Return Values

```
IX_SUCCESS              if the operation is success else a valid
ix_error
```

Error Codes

```
IX_CC_ERROR_NULL        null input parameter
IX_CC_ERROR_OOM               memory allocation failure
IX_ERROR_CC_FPM_CCI     failure from CCI
```

IX_ERROR_CC_MSG_LIBRARY failure from CCI

Logic Description

1. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_eth_tx_next_hop_info structure.
- Call ix_cc_msup_send_sync_msg with IX_CC_FP_MODULE_EGRESS_INPUT identifier, IX_CC_FPM_MSG_ADD_L2_ENTRY, ix_cc_eth_tx_next_hop_info and length of message. If the call fails, return IX_ERROR with error code IX_ERROR_CC_MSG_LIBRARY.

2. The above synchronous call to Message Support blocks waiting for a message to arrive from the FP module egress CC. Once the call returns interpret the success code and return the error/success to the user.

3.7.2 ix_cc_fpm_sync_delete_l2_entry

ix_cc_fpm_sync_delete_l2_entry()

Syntax

```
ix_error ix_cc_fpm_sync_delete_l2_entry (
            ix_int32          arg_L2Id
);
```

Description

This function helps the client to delete a L2 entry into the L2 Table Manager maintained on the egress. The FP module of the CP-PDK invokes this function.

Input Parameters

arg_L2ID	L2 ID to be deleted from the L2 Table.

Return Values

IX_SUCCESS	if the operation is success, or a valid ix_error.

Error Codes

IX_CC_ERROR_NULL	null input parameter
IX_CC_ERROR_OOM	memory allocation failure
IX_ERROR_CC_FPM_CCI	failure from CCI
IX_ERROR_CC_MSG_LIBRARY	failure from CCI

Logic Description

1. Create and send appropriate message-to-message support library.

- Call ix_cc_msup_send_sync_msg with IX_CC_FP_MODULE_EGRESS_INPUT identifier, IX_CC_FPM_MSG_DELETE_L2_ENTRY, ix_int32 (L2Id) and length of message. If the call fails, return IX_ERROR with error code IX_ERROR_CC_MSG_LIBRARY.

2. The above synchronous call to Message Support will block waiting for a message to arrive from the FP module egress CC. Once the call returns interpret the success code and return the error/success to the user.

3.7.3 ix_cc_fpm_async_add_arp_entry

```
ix_cc_fpm_async_add_arp_entry()
```

Syntax

```
ix_error ix_cc_fpm_async_add_arp_entry (
ix_uint32               arg_L2ID,
ix_uint32               arg_NextHopIP,
ix_uint32               arg_OutputPort,
ix_uint32               arg_L2Addr,
ix_cc_fpm_arp_op_cb     arg_Callback,
void*                   arg_pContext);
```

Description

This function helps the client to add an ARP entry into the ARP module maintained on the egress. The FP module of the CP-PDK invokes this function.

Input Parameters

```
arg_L2ID                L2 ID to be added into the L2 Table
arg_NexthopIP                 Next Hop IP address for the L2 ID
arg_OutputPort                Output port to which the specified
Next Hop IP address is attached
arg_L2Addr              L2 Address (MAC Address) corresponding to
the L2 ID being added
```

Return Values

```
IX_SUCCESS              if the operation is success else a valid
ix_error.
```

Error Codes

```
IX_CC_ERROR_NULL              null input parameter
IX_CC_ERROR_OOM                     memory allocation failure
IX_ERROR_CC_FPM_CCI           failure from CCI
IX_ERROR_CC_MSG_LIBRARY       failure from CCI
```

Logic Description

1. Validity check for Parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for ARP callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.

3. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_eth_tx_next_hop_info structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_arp_callback_data
- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_EGRESS_INPUT identifier, ix_cc_fpm_cb_general, ix_cc_general_callback_data, IX_CC_FPM_MSG_ADD_ARP_ENTRY, ix_cc_eth_tx_next_hop_info and length of

message. If the call fails, return IX_ERROR with error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return the success/error code.

3.7.4 ix_cc_fpm_async_delete_arp_entry

`ix_cc_fpm_async_delete_arp_entry()`

Syntax

```
ix_error ix_cc_fpm_async_delete_arp_entry (
ix_uint32               arg_L2Id,
ix_cc_fpm_arp_op_cb     arg_Callback,
void*                   arg_pContext);
```

Description

This function helps the client to delete an ARP entry from the ARP module maintained on the egress. The FP module of the CP-PDK invokes this function.

Input Parameters

```
arg_L2ID                L2 ID that is to be deleted from the ARP table
arg_Callback                  pointer to client provided callback function
arg_ppContext                 pointer to the client defined context
```

Return Values

```
IX_SUCCESS              if the operation is success else a valid ix_error.
```

Error Codes

```
IX_CC_ERROR_NULL        null input parameter
IX_CC_ERROR_OOM               memory allocation failure
IX_ERROR_CC_FPM_CCI     failure from CCI
IX_ERROR_CC_MSG_LIBRARY failure from CCI
```

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for ARP callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.

3. Create and send appropriate message-to-message support library.

- Copy arg_Callback and arg_pContext into ix_cc_fpm_arp_callback_data
- Call ix_cc_msup_send_async_msg with FPM EGRESS CC identifier, ix_cc_fpm_arp_cb, ix_cc_arp_callback_data, IX_CC_FPM_MSG_DELTE_ARP_ENTRY, ix_int32 and length of message. If the call fails, return IX_ERROR with error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return success/failure from the above operations.

3.7.5 ix_cc_fpm_sync_purge_arp_entry

```
ix_cc_fpm_sync_purge_arp_table ()
```

Syntax

```
ix_error ix_cc_fpm_async_purge_arp_table ();
```

Description

This function helps the client to purge the ARP table from the ARP module maintained on the egress. The FP module of the CP-PDK invokes this function.

Input Parameters

None

Return Values

```
IX_SUCCESS                  if the operation is success else a
valid ix_error.
```

Error Codes

```
IX_CC_ERROR_NULL            null input parameter
IX_CC_ERROR_OOM                   memory allocation failure
IX_ERROR_CC_FPM_CCI         failure from CCI
IX_ERROR_CC_MSG_LIBRARY     failure from CCI
```

Logic Description

1. Create and send appropriate message-to-message support library.

- Call ix_cc_msup_send_sync_msg with IX_CC_FP_MODULE_EGRESS_INPUT identifier, ix_cc_fpm_cb_general, ix_cc_general_callback_data, IX_CC_FPM_MSG_PURGE_ARP_TABLE and length of message. If the call fails, return IX_ERROR with error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

2. Return to user (this call is a fire and forget call).

3.7.6 ix_cc_fpm_async_get_eth_tx_stats

```
ix_cc_fpm_async_get_eth_tx_stats()
```

Syntax

```
ix_error ix_cc_fpm_async_get_eth_tx_stats (
ix_cc_eth_statistics_info_data*     arg_pStatsInfo,
ix_cc_fpm_eth_tx_stats_op_cb        arg_Callback);
```

Description

This function helps the client to get the L2 statistics for Ethernet maintained on the egress for the specified port. The FP module of the CP-PDK invokes this function.

Input Parameters

```
arg_pStatsInfo          pointer to client provided statistics
information structure. Includes the counters to be retrieved, user
context, and so on.
arg_Callback            user provided callback function.
```

Return Values

```
IX_SUCCESS        if the operation is success else a valid
ix_error.
```

Error Codes

```
IX_CC_ERROR_NULL              null input parameter
IX_CC_ERROR_OOM                     memory allocation failure
IX_ERROR_CC_FPM_CCI           failure from CCI
IX_ERROR_CC_MSG_LIBRARY       failure from CCI
```

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for Ethernet Transmit Statistics callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.
3. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_eth_tx_statistics_info structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_eth_tx_stats_callback_data
- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_EGRESS_INPUT identifier, ix_cc_fpm_eth_tx_stats_cb, ix_cc_eth_tx_stats_callback_data, IX_CC_FPM_MSG_ETH_STATS, ix_cc_fpm_eth_tx_stats_data and length of message.
- If the call fails, return IX_ERROR with error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return success of the operation.

3.7.7 ix_cc_fpm_async_get_atm_pos_tx_stats

```
ix_cc_fpm_async_get_atm_pos_tx_stats()
```

Syntax

```
ix_error ix_cc_fpm_async_get_atm_pos_tx_stats (
ix_cc_atm_pos_tx_statistics_info_data*    arg_pStatsInfo,
ix_cc_fpm_atm_pos_tx_stats_op_cb          arg_Callback);
```

Description

This function helps a client to get the L2 statistics for ATM maintained on the egress. FP module of the CP-PDK invokes this function.

Input Parameters

```
arg_pStatsInfo          pointer to client provided statistics
information structure. Includes the counters to be retrieved, user
context, and so on.
arg_Callback            user provided callback function.
```

Return Values

```
IX_SUCCESS        if the operation is success else a valid
ix_error.
```

Error Codes

```
IX_CC_ERROR_NULL        null input parameter
IX_CC_ERROR_OOM               memory allocation failure
IX_ERROR_CC_FPM_CCI           failure from CCI
IX_ERROR_CC_MSG_LIBRARY failure from CCI
```

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for POS/ATM Transmit Statistics callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.
3. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_pos_atm_tx_statistics_info structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_eth_tx_stats_callback_data
- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_EGRESS_INPUT identifier, ix_cc_fpm_pos_atm_tx_stats_cb, ix_cc_pos_atm_tx_stats_callback_data, IX_CC_FPM_MSG_ETH_STATS, ix_cc_fpm_pos_atm_tx_stats_data and length of message.
- If the call fails, return IX_ERROR with error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return success of operation.

3.7.8 ix_cc_llc_snap_encap_fpm_add_next_hop

```
ix_cc_llc_snap_encap_fpm_add_next_hop()
```

Syntax

```
ix_error ix_cc_llc_snap_encap_fpm_add_next_hop (
ix_uint16 nextHop,
ix_uint8 encapBit,
ix_uint16 vpi,
ix_uint16 vci,
ix_uint16 vcq,
ix_uint8 dSap,
ix_uint8 sSap,
ix_uint8 ctrl,
ix_uint32 oui,
```

```
ix_uint16 pid,
ix_cc_llc_snap_encap_cb_add_next_hop pCallback,
void *pUserContext);
```

Description

This function helps a client to add the next Hop for the IP over ATM case. The FP module of the CP-PDK invokes this function.

Input Parameters

```
nextHop          nexthop ID
encapBit         Encapsulation bit (VCMUX / LLCSNAP)
vpi         VPI
vci         VCI
vcq         VC Queue number
dsap        Destination Service Access Point
sSap        Source Service Access Point
ctrl        Control Bit
oui         Organizational Unique ID
pid         Protocol ID
pCallback        User provided callback function
```

Return Values

```
IX_SUCCESS       if the operation is success, else a valid
ix_error
```

Error Codes

```
IX_CC_ERROR_NULL             null input parameter
IX_CC_ERROR_OOM                    memory allocation failure
IX_ERROR_CC_FPM_CCI          failure from CCI
IX_ERROR_CC_MSG_LIBRARY      failure from CCI
```

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.
3. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_fpm_llc_snap_encap_add_next_hop_t structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_llc_snap_encap_add_next_hop_cb_t
- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_INGRESS_OUTPUT identifier, ix_cc_llc_snap_encap_fpm_add_next_hop_cb_core, ix_cc_fpm_llc_snap_encap_add_next_hop_cb_t, IX_CC_FPM_MSG_LLC_SNAP_ENCAP_ADD_NEXT_HOP, ix_cc_fpm_llc_snap_encap_add_next_hop_t and length of message.
- If the call fails, return IX_ERROR with error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return success of the operation.

3.7.9 ix_cc_fpm_async_port_create

`ix_cc_fpm_async_port_create()`

Syntax

```
ix_error ix_cc_fpm_async_port_create (
ix_cc_atmsar_instance_t arg_sarInstance,
ix_cc_atmsar_port_descr_t* arg_pPortDescr,
ix_cc_atmsar_cb_port_create_t arg_Callback,
void *pUserContext);
```

Description

This function helps a client to create an ATM Port. FP module of the CP-PDK invokes this function.

Input Parameters

`arg_sarInstance`	`ATMSAR instance`
`arg_pPortDescr`	`user provided ATM port attributes descriptor`
`arg_Callback`	`user provided callback function.`
`pUserContext`	`user provided context`

Return Values

`IX_SUCCESS`	`if the operation is success else a valid ix_error`

Error Codes

`IX_CC_ERROR_NULL`	`null input parameter`
`IX_CC_ERROR_OOM`	`memory allocation failure`
`IX_ERROR_CC_FPM_CCI`	`failure from CCI`
`IX_ERROR_CC_MSG_LIBRARY`	`failure from CCI`

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.
3. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_fpm_atm_port_descr_t structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_atm_port_create_cb_t
- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_INGRESS_OUTPUT identifier, ix_cc_fpm_atm_port_create_cb_core, ix_cc_fpm_atm_port_create_cb_t, IX_CC_FPM_MSG_ATM_PORT_CREATE, ix_cc_fpm_atm_port_descr_t and length of message.

- If the call fails, return IX_ERROR with error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return success of the operation

3.7.10 ix_cc_fpm_async_port_remove

`ix_cc_fpm_async_port_remove()`

Syntax

```
ix_error ix_cc_fpm_async_port_remove (
ix_cc_atmsar_instance_t arg_sarInstance,
ix_cc_atmsar_port_handle_t arg_port,
ix_cc_atmsar_cb_port_remove_t arg_Callback,
void *pUserContext);
```

Description

This function helps the client to delete an ATM Port. The FP module of the CP-PDK invokes this function.

Input Parameters

arg_sarInstance	ATMSAR instance
arg_port	user provided ATM port handle
arg_Callback function.	user provided callback
pUserContext	user provided context

Return Values

IX_SUCCESS	if the operation is success else a valid ix_error

Error Codes

IX_CC_ERROR_NULL	null input parameter
IX_CC_ERROR_OOM	memory allocation failure
IX_ERROR_CC_FPM_CCI	failure from CCI
IX_ERROR_CC_MSG_LIBRARY	failure from CCI

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.
3. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_fpm_atm_port_handle_t structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_atm_port_del_cb_t
- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_INGRESS_OUTPUT identifier, ix_cc_fpm_atm_port_del_cb_core, ix_cc_fpm_atm_port_del_cb_t,

IX_CC_FPM_MSG_ATM_PORT_DELETE, ix_cc_fpm_atm_port_handle_t and length of message.

- If the call fails, return IX_ERROR with error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return success of the operation.

3.7.11 ix_cc_fpm_async_get_port_stats

`ix_cc_fpm_async_get_port_stats()`

Syntax

```
ix_error ix_cc_fpm_async_get_port_stats (
ix_cc_atmsar_instance_t arg_sarInstance,
ix_cc_atmsar_port_handle_t arg_port,
ix_cc_atmsar_cb_get_port_stats_t arg_Callback,
void *pUserContext);
```

Description

This function helps the client to get statistics of an ATM Port. The FP module of the CP-PDK invokes this function.

Input Parameters

arg_sarInstance	ATMSAR instance
arg_port	user provided ATM port handle
arg_Callback function.	user provided callback
pUserContext	user provided context

Return Values

IX_SUCCESS	if the operation is success else a valid ix_error

Error Codes

IX_CC_ERROR_NULL	null input parameter
IX_CC_ERROR_OOM	memory allocation failure
IX_ERROR_CC_FPM_CCI	failure from CCI
IX_ERROR_CC_MSG_LIBRARY	failure from CCI

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.
3. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_fpm_atm_port_handle_t structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_atm_port_stats_get_cb_t

- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_INGRESS_OUTPUT identifier, ix_cc_fpm_atm_port_stats_get_cb_core, ix_cc_fpm_atm_port_stats_get_cb_t, IX_CC_FPM_MSG_ATM_PORT_STATS_GET, ix_cc_fpm_atm_port_handle_t and length of message.
- If the call fails, return IX_ERROR with the error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return success of the operation.

3.7.12 ix_cc_fpm_async_vc_create

`ix_cc_fpm_async_vc_create()`

Syntax

```
ix_error ix_cc_fpm_async_vc_create (
ix_cc_atmsar_instance_t arg_sarInstance,
ix_cc_atmsar_vc_descr_t* arg_pVcDescr,
ix_cc_atmsar_cb_vc_create_t arg_Callback,
void *pUserContext);
```

Description

This function helps a client to create an ATM VC. FP module of the CP-PDK invokes this function.

Input Parameters

arg_sarInstance	ATMSAR instance
arg_pVcDescr	user provided ATM VCC attributes descriptor
arg_Callback	user provided callback function.
pUserContext	user provided context

Return Values

IX_SUCCESS	if the operation is success else a valid ix_error

Error Codes

IX_CC_ERROR_NULL	null input parameter
IX_CC_ERROR_OOM	memory allocation failure
IX_ERROR_CC_FPM_CCI	failure from CCI
IX_ERROR_CC_MSG_LIBRARY	failure from CCI

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.
3. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_fpm_atm_vc_descr_t structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_atm_vc_create_cb_t
- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_INGRESS_OUTPUT identifier, ix_cc_fpm_atm_vc_create_cb_core, ix_cc_fpm_atm_vc_create_cb_t, IX_CC_FPM_MSG_ATM_VCC_CREATE, ix_cc_fpm_atm_vc_descr_t and length of message.
- If the call fails, return IX_ERROR with error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return success of the operation.

3.7.13 ix_cc_fpm_async_vc_update

`ix_cc_fpm_async_vc_update()`

Syntax

```
ix_error ix_cc_fpm_async_vc_update (
ix_cc_atmsar_instance_t arg_sarInstance,
ix_cc_atmsar_vc_handle_t arg_hVc,
ix_cc_atmsar_vc_descr_t* arg_pUpdateData,
ix_cc_atmsar_cb_vc_create_t arg_Callback,
void *pUserContext);
```

Description

This function helps the client to create an ATM VC. The FP module of the CP-PDK invokes this function.

Input Parameters

arg_sarInstance	ATMSAR instance
arg_pVcDescr	user provided ATM VCC attributes descriptor
arg_Callback	user provided callback function.
pUserContext	user provided context

Return Values

IX_SUCCESS	if the operation is success else a valid ix_error

Error Codes

IX_CC_ERROR_NULL	null input parameter
IX_CC_ERROR_OOM	memory allocation failure
IX_ERROR_CC_FPM_CCI	failure from CCI
IX_ERROR_CC_MSG_LIBRARY	failure from CCI

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.
3. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_fpm_atm_vc_update_t structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_atm_vc_update_cb_t
- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_INGRESS_OUTPUT identifier, ix_cc_fpm_atm_vc_update_cb_core, ix_cc_fpm_atm_vc_update_cb_t, IX_CC_FPM_MSG_ATM_VCC_UPDATE, ix_cc_fpm_atm_vc_update_t and length of message.
- If the call fails, return IX_ERROR with the error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return success of the operation.

3.7.14 ix_cc_fpm_async_vc_remove

`ix_cc_fpm_async_vc_remove()`

Syntax

```
ix_error ix_cc_fpm_async_vc_remove (
ix_cc_atmsar_instance_t arg_sarInstance,
ix_cc_atmsar_vc_handle_t arg_hVc,
ix_cc_atmsar_cb_vc_remove_t arg_Callback,
void *pUserContext);
```

Description

This function helps the client to delete an ATM Port. The FP module of the CP-PDK invokes this function.

Input Parameters

arg_sarInstance	ATMSAR instance
arg_hVc	user provided ATM VC handle
arg_Callback function.	user provided callback
pUserContext	user provided context

Return Values

IX_SUCCESS	if the operation is success else a valid ix_error

Error Codes

IX_CC_ERROR_NULL	null input parameter
IX_CC_ERROR_OOM	memory allocation failure
IX_ERROR_CC_FPM_CCI	failure from CCI
IX_ERROR_CC_MSG_LIBRARY	failure from CCI

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocate memory for callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.
3. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_fpm_atm_vc_handle_t structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_atm_vc_remove_cb_t
- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_INGRESS_OUTPUT identifier, ix_cc_fpm_atm_vc_remove_cb_core, ix_cc_fpm_atm_vc_remove_cb_t, IX_CC_FPM_MSG_ATM_VCC_DELETE, ix_cc_fpm_atm_vc_handle_t and length of message.
- If the call fails, return IX_ERROR with the error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

4. Return success of the operation.

3.7.15 ix_cc_fpm_async_get_vc_stats

`ix_cc_fpm_async_get_vc_stats()`

Syntax

```
ix_error ix_cc_fpm_async_get_vc_stats (
ix_cc_atmsar_instance_t arg_sarInstance,
ix_cc_atmsar_vc_handle_t arg_hVc,
ix_cc_atmsar_cb_get_vc_stats_t arg_Callback,
void *pUserContext);
```

Description

This function helps the client to get statistics of an ATM VCC. The FP module of the CP-PDK invokes this function.

Input Parameters

arg_sarInstance	ATMSAR instance
arg_hVc	user provided ATM VCC handle
arg_Callback function	user provided callback
pUserContext	user provided context

Return Values

IX_SUCCESS	if the operation is successful else a valid ix_error.

Error Codes

IX_CC_ERROR_NULL	null input parameter
IX_CC_ERROR_OOM	memory allocation failure
IX_ERROR_CC_FPM_CCI	failure from CCI
IX_ERROR_CC_MSG_LIBRARY	failure from CCI

Logic Description

5. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with error code IX_CC_ERROR_NULL

6. Allocate memory for callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM.

7. Create and send appropriate message-to-message support library.

- Copy arguments into ix_cc_fpm_atm_vc_handle_t structure.
- Copy arg_Callback and arg_pContext into ix_cc_fpm_atm_vc_stats_get_cb_t
- Call ix_cc_msup_send_async_msg with IX_CC_FP_MODULE_INGRESS_OUTPUT identifier, ix_cc_fpm_atm_vc_stats_get_cb_core, ix_cc_fpm_atm_vc_stats_get_cb_t, IX_CC_FPM_MSG_ATM_VCC_STATS_GET, ix_cc_fpm_atm_vc_handle_t and length of message.
- If the call fails, return IX_ERROR with the error code IX_ERROR_CC_FPM_ERROR_MSG_LIBRARY.

8. Return success of the operation.

3.7.16 ix_cc_fpm_dpe_op

Syntax

```
ix_error ix_cc_fpm_async_dpe_op (
ix_cc_fpm_dpe *arg_DpeInfo,
ix_cc_fpm_dpe_cb arg_Callback,
void* arg_pUserContext
)
```

Description

This function helps the client to perform DPE operations on the egress CC.

Input Parameters

arg_DpeInfo	DPE details
arg_Callback function	user provided callback
pUserContext	user provided context

Return Values

IX_SUCCESS	if the operation is successful else a valid ix_error.

Error Codes

IX_CC_ERROR_NULL	null input parameter
IX_CC_ERROR_OOM	memory allocation failure
IX_ERROR_CC_FPM_CCI	failure from CCI
IX_ERROR_CC_MSG_LIBRARY	failure from CCI

Logic Description

1. Validity check for parameters

- If arg_Callback is NULL, return IX_ERROR with the error code IX_CC_ERROR_NULL.

2. Allocate memory for the callback data. If not successful, return IX_CC_ERROR_OOM_SYSTEM
3. Create and send appropriate message-to-message support library.

Return success/failure of the operation.

Part 4: FP Module Egress Core Component

4 FP Module Egress Core Component

The FPM egress CC is a very simple **proxy** core component. It receives the message from the ingress and processes it by interacting with the L2 Table Manager, Ethernet Tx CC, and so on.

4.1 Initialization

```
ix_cc_fpm_proxy_init()
```

Syntax

```
ix_error ix_cc_fpm_proxy_init(
ix_cc_handle      arg_hFPMCCHandle,
void**            arg_ppContext);
```

Description

This function is the initialization function of the core component and must be invoked by the Execution Engine it is running in. It must be invoked once otherwise the behavior is undefined. This function is responsible for performing the basic initializations of the core component such as registering message and/or packet handlers, and so on. This function must be called before requesting any service before the core component. Since this core component is not responsible for any micro blocks, there is no need for patching symbols, initializing micro block memory, and so on.

This function also creates the L2 Table Manager and stores the corresponding table handle into the context. This is used during shutdown to terminate L2 Table Manager services.

Input Parameters

```
arg_hFPMCCHandle  handle to FP module core component, created by
core component infrastructure; this shall be used later to get
other (to add event handler) services from core component
infrastructure.
```

Output Parameters

```
arg_ppContext           location where the pointer to the control
block allocated by the core component is stored.  The control
block is internal to the core component and contains variables and
internal data structures.  This pointer is use later to be passed
into the ix_cc_fpm_proxy_fini() function by core component
infrastructure for memory freeing when the core component is being
destroyed.
```

Return Values

```
IX_SUCCESS        if the initialization is successful else a valid
ix_error
```

Error Codes

```
IX_CC_ERROR_NULL        null input parameter
IX_CC_ERROR_OOM_SYSTEM  memory allocation failure
IX_ERROR_CC_FPM_CCI     failure from CCI
```

Logic Description

1. Validity check for Parameters

- If arg_ppContex is null, then return IX_ERROR with error code IX_CC_ERROR_NULL

2. Allocation of memory for FPM Proxy CC Context

- Call ix_ossl_malloc() for memory allocation of ix_cc_fpm_context. If it fails, then return IX_ERROR with error code IX_CC_ ERROR_OOM
- Store FP module core component handle in global variable p_fpm_cc_Context

3. If this is on the egress, initialize L2 Table Manager

- Call ix_cc_l2tm_create() to initialize L2 Table Manager. If this fails, free memory allocated for ix_cc_fpm_context; return IX_ERROR with error code IX_ERROR_CC_FPM_L2TM.
- Store L2 Table Manager handle in ix_cc_fpm_context.

4. Registration of message handler

- Call ix_cci_add_message_handler() to register FPM CC handler. If it fails, then return IX_ERROR with error code IX_ERROR_CC_FPM_CCI
- If there is no CCI, then call ix_rm_message_handler_register() to register FPM CC message handler. If the call fails, then return IX_ERROR with error code IX_ERROR_CC_FPM_CCI

5. Return IX_SUCCESS.

4.2 Shutdown

```
ix_cc_fpm_proxy_fini()
```

Syntax

```
ix_error ix_cc_fpm_proxy_fini(
ix_cc_handle      arg_hFPMCCHandle,
void*             arg_pContext);
```

Description

This function is the termination function of the core component and must be invoked by the Execution Engine it is running in. It must be invoked exactly once. This function is responsible for cleaning up all memory allocated by the core component, deregistering message and/or packet handlers, and so on. On the egress side, this function terminates the services of the L2 Table Manager.

Input Parameters

```
arg_hFPMCCHandle  handle to FP module core component.
arg_pContext            pointer to the control block allocated
earlier in ix_cc_fpm_init function.
```

Return Values

```
IX_SUCCESS        if shutdown is success else a valid ix_error
```

Error Codes

```
IX_CC_ERROR_NULL        null input parameter
IX_CC_ERROR_OOM_SYSTEM        memory allocation failure
IX_CC_ERROR_CCI         failure from CCI
```

Logic Description

1. Validity check for Parameters

- If arg_pContex is null, then return IX_ERROR with the error code IX_CC_ERROR_NULL
- If arg_pContext is not context of FPM core component, return IX_ERROR with error code IX_ ERROR_CC_FPM_ INVALID_INPUT_PARAM
- If arg_hFPMCCHandle does not match the FPM core component handle stored in p_fpm_cc_context, return IX_ERROR with error code IX_ERROR_CC_FPM_INVALID_INPUT_PARAM

2. Terminate L2 Table Manager services

- Call ix_cc_l2tm_fini with L2 Table Handle stored in ix_cc_fpm_context. If this fails, set IX_ERROR with error code IX_ERROR_CC_FPM_L2TM.

3. Remove message handler

- If CCI is present, call ix_cci_cc_remove_message_handler with FPM CC Id. If this fails, return IX_ERROR with error code IX_ERROR_CC_FPM_CCI.
- If there is no CCI, call ix_rm_remove_message_handler with the FPM CC ID. If this fails, return IX_ERROR with the error code IX_ERROR_CC_FPM_RM.

4. Return IX_SUCCESS

4.3 Message Handler

```
ix_cc_fpm_proxy_msg_handler()
```

Syntax

```
ix_error ix_cc_fpm_proxy_message_handler(
ix_buffer_handle  arg_hDataToken,
ix_uint32         arg_UserData,
void*                   arg_pComponentContext);
```

Description

This primitive is the message handler function for the FP module egress core component. Then the FPM egress CC internally calls the appropriate library function to process the message. The communication between the client and core components for message passing and retrieving of result is described in Messaging API and Support Library section of the IXA SDK documentation.

Input Parameters

```
arg_hDataToken          buffer handle embedding information for
the message passed in arg_UserData
arg_UserData                    Message type. The supported messages
are as defined in 3.3.3.
arg_pComponentContext   pointer to FP module core component
context that is passed to the core component when a message
arrives. This context was defined by the core component and passed
```

to core components infrastructure through the ix_cc_fpm_init function.

Return Values

IX_SUCCESS if success, or a valid ix_error for failure

Error Codes

IX_CC_ERROR_NULL	null input parameter
IX_CC_ERROR_OOM	memory allocation failure
IX_ERROR_CC_FPM_CCI	failure from CCI
IX_ERROR_CC_MSG_LIBRARY	failure from CCI
IX_ERROR_CC_FPM_UNDEFINED_MSG	failure from CCI
IX_ERROR_FPM_CC_INVALID_INPUT_PARAM	failure from CCI

Logic Description

1. Validity check for Parameters

- If arg_pComponentContext is null, then return IX_ERROR with error code IX_CC_ERROR_NULL
- If arg_pComponentContext is not context of FPM core component, return IX_ERROR with error code IX_ERROR_CC_FPM_INVALID_INPUT_CONTEXT

2. Validity check for message type

- If arg_UserData does not contain one of the messages supported by the FPM core component, then return IX_ERROR with error code IX_ERROR_CC_FPM_UNDEFINED_MSG

3. Retrieve data from the buffer

- Call IX_MSUP_EXTRACT_MSG (this function is exported by Message Support Library) with parameter arg_hDataToken, to retrieve message and return context.
- If the buffer is null, return IX_ERROR with error code IX_CC_ERROR_NULL

4. Invoke Library API. The corresponding reply message is sent by the library functions after appropriate processing.

- Call appropriate library function with the retrieved data
    - If arg_UserData == IX_CC_FPM_MSG_ADD_L2_ENTRY, then call ix_cc_fpm_add_l2_entry.
    - If arg_UserData == IX_CC_FPM_MSG_DELETE_L2_ENTRY, then call ix_cc_fpm_delete_l2_entry.
    - If arg_UserData == IX_CC_FPM_MSG_ADD_ARP_ENTRY, then call ix_cc_fpm_add_arp_entry.
    - If arg_UserData == IX_CC_FPM_MSG_DELETE_ARP_ENTRY, then call ix_cc_fpm_delete_arp_entry.
    - If arg_UserData == IX_CC_FPM_MSG_PURGE_ARP_TABLE, then call ix_cc_fpm_purge_arp_table.
    - If arg_UserData == IX_CC_FPM_MSG_ETH_STATS, then call ix_cc_fpm_l2_eth_stats.

- If arg_UserData == IX_CC_FPM_MSG_ATM_POS_STATS, then call ix_cc_fpm_l2_atm_pos_stats.
- If arg_UserData == IX_CC_FPM_MSG_LLC_SNAP_ENCAP_ADD_NEXT_HOP, then call ix_cc_llc_snap_encap_async_add_next_hop. The internal callback function is invoked at a later time with the result of the operation.A reply message is now sent back to ingress CC with the retrieved result & the return codes
- If arg_UserData == IX_CC_FPM_MSG_ATM_PORT_CREATE, then call ix_cc_atmsar_async_port_create. The internal callback function is invoked at a later time with the result of the operation and port handle.A reply message is now sent back to ingress CC with the retrieved port handle & the return codes
- If arg_UserData == IX_CC_FPM_MSG_ATM_PORT_DELETE, then call ix_cc_atmsar_async_port_remove. The internal callback function is invoked at a later time with the result of the operation. A reply message is now sent back to ingress CC with the retrieved result & the return codes
- If arg_UserData == IX_CC_FPM_MSG_ATM_PORT_STATS_GET, then call ix_cc_atmsar_async_get_port_stats. The internal callback function is invoked at a later time with the result of the operation and the statistics.A reply message is now sent back to ingress CC with the retrieved statistics & the return codes
- If arg_UserData == IX_CC_FPM_MSG_ATM_VCC_CREATE, then call ix_cc_atmsar_async_vc_create. The internal callback function is invoked at a later time with the result of the operation and vc handle.A reply message is now sent back to ingress CC with the retrieved vc handle and the return codes
- If arg_UserData == IX_CC_FPM_MSG_ATM_VCC_UPDATE, then call ix_cc_atmsar_async_vc_update. The internal callback function is invoked at a later time with the result of the operation.A reply message is now sent back to ingress CC with the retrieved result & the return codes
- If arg_UserData == IX_CC_FPM_MSG_ATM_VCC_DELETE, then call ix_cc_atmsar_async_vc_remove. The internal callback function is invoked at a later time with the result of the operation.A reply message is now sent back to ingress CC with the retrieved result & the return codes
- If arg_UserData == IX_CC_FPM_MSG_ATM_VCC_STATS_GET, then call ix_cc_atmsar_async_get_vc_stats. The internal callback function is invoked at a later time with the result of the operation and the statistics.A reply message is now sent back to ingress CC with the retrieved statistics & the return codes
- If arg_UserData == IX_CC_FPM_MSG_DPE_OP, invoke appropriate DiffServ CC api.

5. Free arg_hDataToken (ix_buffer).
6. Return IX_SUCCESS.

4.4 Library API

4.4.1 ix_cc_fpm_add_l2_entry

`ix_cc_fpm_add_l2_entry()`

Syntax

```
ix_error ix_cc_fpm_add_l2_entry (
ix_uint32               arg_L2ID,
ix_uint32               arg_NextHopIP,
ix_uint32               arg_OutputPort
);
```

Description

This function adds a L2 entry into the L2 Table Manager. Subsequently, it calls into the Ethernet Tx CC to create an ARP request using the Ethernet TX CC API.

Input Parameters

```
arg_L2ID                L2 ID to be added into the L2 Table.
arg_NexthopIP                 Next Hop IP address for the L2 ID.
arg_OutputPort                Output port to which the specified
Next Hop IP address is attached.
```

Return Values

```
IX_SUCCESS              if the operation is success else a valid
ix_error
```

Error Codes

IX_ERROR_INVALID_HANDLE	invalid Handle to L2 Table
IX_CC_L2TM_ERROR_INVALID_L2_ENTRY	invalid L2 entry
IX_CC_L2TM_ERROR_INVALID_L2ID	invalid L2 ID
IX_ERROR_INVALID_POINTER	invalid input parameter
IX_ERROR_CC_ARP_ENTRY_EXISTS	ARP entry with the NH ID already exists
IX_ERROR_CC_ARP_INVALID_L2ID	invalid L2 ID
IX_ERROR_CC_ARP_ERROR	Internal ARP error

Logic Description

1. Retrieve L2 Table Handle from FPM context data structure (ix_cc_fpm_context).
2. Invoke L2 Table API

- Construct ix_cc_l2tm_l2_entry structure with Next Hop IP from ix_cc_eth_tx_next_hop_info structure.
- Call ix_cc_l2tm_add_l2_entry() with parameters ix_cc_l2tm (handle), ix_cc_rtmv4_nhid (next hop id) and ix_cc_l2tm_l2_entry structure.
- Return error code as returned by the call into L2 Table Manager. Possible errors:
  - IX_ERROR_INVALID_HANDLE
  - IX_CC_L2TM_ERROR_INVALID_L2_ENTRY
  - IX_CC_L2TM_ERROR_INVALID_NHID
  - IX_ERROR_INVALID_POINTER

3. Invoke Interface Tx CC API to create an ARP entry

- Call ix_cc_eth_tx_async_create_arp_entry with ix_cc_eth_tx_next_hop_info structure as parameter. Pass an internal function as callback.

- When this function gets called, send the reply message back to the caller (FP module ingress CC). Return the error code as returned by the call. Possible errors:
  - IX_ERROR_CC_ARP_INVALID_NEXT_HOP_ID
  - IX_ERROR_CC_ARP_ENTRY_EXISTS – The existing entry is not changed
  - IX_ERROR_CC_ARP_ERROR – internal error.

4. Return success/failure of operation.

4.4.2 ix_cc_fpm_delete_l2_entry

```
ix_cc_fpm_delete_l2_entry()
```

Syntax

```
ix_error ix_cc_fpm_delete_l2_entry (
ix_int32          arg_L2Id
);
```

Description

This function deletes a L2 entry from the L2 Table Manager maintained on the egress.

Input Parameters

```
arg_L2ID                        L2 ID to be deleted from the L2
Table.
```

Return Values

```
IX_SUCCESS                      if operation is success else a valid
ix_error
```

Error Codes

```
IX_ERROR_INVALID_HANDLE             invalid Handle to L2 Table
IX_CC_L2TM_ERROR_INVALID_L2ID       invalid L2 ID
```

Logic Description

1. Retrieve L2 Table Handle from FPM context data structure (ix_cc_fpm_context).
2. Invoke L2 Table API

- Call ix_cc_l2tm_delete_l2_entry() with parameters ix_cc_l2tm (handle), ix_cc_rtmv4_nhid (next hop id).
- Return error code as returned by the call into L2 Table Manager. Possible errors:
  - IX_ERROR_INVALID_HANDLE
  - IX_CC_L2TM_ERROR_INVALID_L2ID

3. Send a reply message back to the ingress. This can be sent now itself, since the L2 Table Manager is a library and hence the calls are synchronous.
4. Return success/failure of the operation.

4.4.3 ix_cc_fpm_add_arp_entry

```
ix_cc_fpm_add_arp_entry()
```

Syntax

```
ix_error ix_cc_fpm_add_arp_entry (
ix_uint32               arg_L2ID,
ix_uint32               arg_NextHopIP,
ix_uint32               arg_OutputPort,
ix_uint32               arg_L2Addr
);
```

Description

This function adds an ARP entry into the ARP module and a corresponding L2 entry into the L2 Table Manager. The functionality of this function is same as the ix_cc_fpm_add_l2_entry, except that the L2 address passed in here is a valid L2 address.

Input Parameters

```
arg_L2ID                L2 ID to be added into the L2 Table.
arg_NexthopIP                 Next Hop IP address for the L2 ID.
arg_OutputPort                Output port to which the specified
Next Hop IP address is attached.
arg_L2Addr              L2 Address (MAC Address) corresponding to
the L2 ID being added.
```

Return Values

```
IX_SUCCESS              if operation is success else a valid
ix_error.
```

Error Codes

```
IX_ERROR_INVALID_HANDLE             invalid Handle to L2 Table
IX_CC_L2TM_ERROR_INVALID_L2_ENTRY   invalid L2 entry
IX_CC_L2TM_ERROR_INVALID_L2ID       invalid L2 ID
IX_ERROR_INVALID_POINTER            invalid input parameter
IX_ERROR_CC_ARP_ENTRY_EXISTS        ARP entry with the L2 ID
already exists
IX_ERROR_CC_ARP_INVALID_L2ID        invalid L2 ID
IX_ERROR_CC_ARP_ERROR               Internal ARP error
```

Logic Description

1. Retrieve L2 Table Handle from FPM context data structure (ix_cc_fpm_context).
2. Invoke L2 Table API

- Construct ix_cc_l2tm_l2_entry structure with Next Hop IP from ix_cc_eth_tx_next_hop_info structure.
- Call ix_cc_l2tm_add_l2_entry() with parameters ix_cc_l2tm (handle), ix_cc_rtmv4_nhid (next hop id) and ix_cc_l2tm_l2_entry structure.
- Return error code as returned by the call into L2 Table Manager. Possible errors:

- IX_ERROR_INVALID_HANDLE
- IX_CC_L2TM_ERROR_INVALID_L2_ENTRY
- IX_CC_L2TM_ERROR_INVALID_L2ID
- IX_ERROR_INVALID_POINTER

3. Invoke Interface Tx CC API to create an ARP entry

- Call ix_cc_eth_tx_create_arp_entry with ix_cc_eth_tx_next_hop_info structure as parameter.
- Return the error code as returned by the call. Possible errors:
  - IX_ERROR_CC_ARP_INVALID_L2_ID
  - IX_ERROR_CC_ARP_ENTRY_EXISTS – The existing entry is not changed
  - IX_ERROR_CC_ARP_ERROR – internal error

4. Return success/failure of the operation.

4.4.4 ix_cc_fpm_delete_arp_entry

`ix_cc_fpm_delete_arp_entry()`

Syntax

```
ix_error ix_cc_fpm_delete_arp_entry (
ix_int32          arg_L2Id
);
```

Description

This function deletes an ARP entry from the ARP module maintained on the egress. The message handler invokes this function only on the egress side.

Input Parameters

`arg_L2ID` L2 ID to be deleted from the L2 Table.

Return Values

`IX_SUCCESS` if the operation is success else a valid ix_error

Error Codes

IX_ERROR_INVALID_HANDLE	invalid Handle to L2 Table
IX_ERROR_CC_ARP_ENTRY_NOT_FOUND	ARP entry with the L2 ID already exists
IX_ERROR_CC_ARP_INVALID_L2ID	invalid L2ID
IX_ERROR_CC_ARP_ENTRY_NOT_VALID	the entry is not a static entry. The entry is not removed

Logic Description

1. Invoke Interface Tx API

- Call ix_cc_eth_tx_async_del_arp_entry() with parameters ix_uint32 (next hop id). Pass an internal function as callback.

2. When this function gets called, send the reply message back to the caller (FP module ingress CC). Return error code as returned by the call into L2 Table Manager. Possible errors:

- IX_ERROR_CC_ARP_ENTRY_NOT_FOUND
- IX_ERROR_CC_ARP_INVALID_L2ID
- IX_ERROR_CC_ARP_ENTRY_NOT_VALID

3. Return IX_SUCCESS.

4.4.5 ix_cc_fpm_purge_arp_table

`ix_cc_fpm_purge_arp_table()`

Syntax

```
ix_error ix_cc_fpm_purge_arp_table();
```

Description

This function purges the ARP cache from the ARP module.

Input Parameters

None

Return Values

```
IX_SUCCESS                    if the operation is success else a
valid ix_error
```

Error Codes

```
IX_ERROR_INVALID_HANDLE       invalid Handle to L2 Table
IX_ERROR_CC_ARP_ERROR         internal ARP error
```

Logic Description

4. Invoke Interface Tx API

- Call ix_cc_eth_tx_purge_arp_cache(). This is a fire-and-forget API and hence there are no return values to expect.
- Send reply message back to ingress. Return error code as returned by the call. Possible errors:
  - IX_ERROR_CC_ARP_ERROR

5. Return IX_SUCCESS.

4.4.6 ix_cc_fpm_eth_tx_stats

`ix_cc_fpm_eth_tx_stats()`

Syntax

```
ix_error ix_cc_fpm_eth_tx_stats (
ix_uint32              arg_portIndex
```

```
);
```

Description

This function queries the Ethernet transmit statistics maintained by the Ethernet Transmit CC. Subsequently a reply message with the statistics counters is sent back to the FPM ingress CC.

Input Parameters

```
arg_portIndex          port for which the statistics is desired
```

Return Values

```
IX_SUCCESS        if the operation is success else a valid
ix_error
```

Error Codes

Error core as returned by the Ethernet Tx CC

Logic Description

1. Create ix_cc_eth_tx_statistics_info structure. This contains the type of statistics desired. Since we get all counters in one shot, this contains the value IX_CC_ETH_TX_ALL_DRIVER_COUNTERS.
2. Invoke Interface Tx API

- Allocate memory for ix_cc_statistics_info_data that will contain the statistics collected.
- Call ix_cc_eth_tx_async_get_statistics_info () with parameters ix_uint32 (port index), the ix_cc_eth_tx_statistics_info and ix_cc_statistics_info_data structures created above, and an internal callback function.
    - The internal callback function is invoked at a later time with the statistics counters.
    - A reply message is now sent back to ingress CC with the retrieved statistics counters & the return codes.

3. Return result of the operation.

4.4.7 ix_cc_fpm_atm_pos_tx_stats

```
ix_cc_fpm_atm_pos_tx_stats()
```

Syntax

```
ix_error ix_cc_fpm_atm_pos_tx_stats (
ix_uint32              arg_portIndex
);
```

Description

This function queries the L2 ATM/POS statistics.

Input Parameters

```
arg_portIndex          port for which statistics is desired
```

Return Values

```
IX_SUCCESS        if the shutdown is success else a valid ix_error
```

Logic Description

1. Create ix_cc_atm_pos_tx_statistics_info structure. This contains the type of statistics desired. Since we get all counters in one shot, this contains the value IX_CC_ATM_TX_ALL_COUNTERS_PORT logically OR-ed with IX_CC_POS_TX_ALL_COUNTERS.
2. Invoke Interface Tx API

- Allocate memory for ix_cc_statistics_info_data that will contain the statistics collected.
- Call ix_cc_atm_pos_tx_async_get_statistics_info () with parameters ix_uint32 (port index), the ix_cc_atm_pos_tx_statistics_info and ix_cc_statistics_info_data structures created above, and an internal callback function.
    - The internal callback function is invoked at a later time with the statistics counters.
    - A reply message is now sent back to ingress CC with the retrieved statistics counters & the return codes.

3. Return IX_SUCCESS.