Forwarding Plane Plug-in API

Reference Guide

Control Plane-Platform Development Kit 2.11

March 2004

Contents

Figures

Tables

Revision History

Revision	Description	Date	Author
2.11	Updated for Release 2.11	March 2004	Shailesh Suman
2.1	Updated for Release 2.1	December 2003	Shailesh Suman
2.0	Updated for Release 2.0	August 2003	Shailesh Suman

Part 1: Overview

1 Overview

Network elements such as switches and routers can be classified into three logical operational components: Control plane, Forwarding plane, and Management plane.

The control plane controls and configures the forwarding plane. The control plane executes different signaling or routing protocols and provides all the routing information to the forwarding plane.

The forwarding plane manipulates network traffic and makes decisions based on this information. The forwarding plane performs operations on packets such as forwarding, classification, filtering, and so on.

An orthogonal management plane manages the control and forwarding planes.

For example, the control plane in a router executes routing protocols, the forwarding plane performs hardware-based switching, and the management plane starts or stops routing process or performs logging.

The introduction of standardized APIs within the above-mentioned planes can help system vendors, OEMs, and end users of these network elements to mix and match components available from different vendors to achieve a device of their choice. The Network Processing Forum (NPF) API is designed for this purpose, as it presents a flexible and well-known programming interface to the control plane applications. It makes the existence of multiple forwarding planes, as well as vendor-specific details, transparent to control plane applications. Furthermore, the hardware properties and nature of interconnect used between the control and the forwarding planes are isolated. Thus, the protocol stacks and network processors available from different vendors can be easily integrated with the NPF APIs. The APIs included in the Control Plane Platform Development Kit are based on the NPF APIs. For more information about NPF, refer to http://www.npforum.org/.

1.1 Transport Plug-in

In CP-PDK architecture [1], the Transport Plug-in is defined to enable communication between the Control and Forwarding plane(s). Figure 1 shows the transport plug-in architecture. It uses the FP Plug-in API to abstract out the communication mechanism details from the rest of the CP-PDK implementation. This document specifies the FP Plug-in API, which is used by the CP-PDK implementation to communicate with the forwarding plane(s). The transport plug-in also exposes a Backend API that is used by the Forwarding Plane PDK implementation. Section 9, Backend API, describes this API.

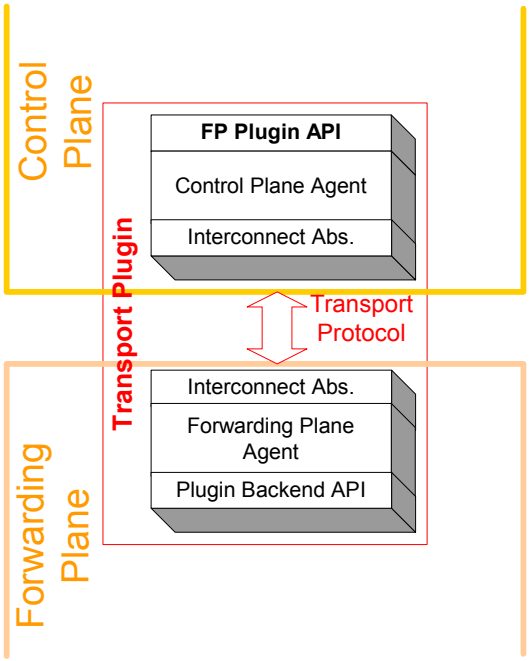

Figure 1. Transport plug-in architecture

The FP Plug-in API is designed to be similar to the actual NPF APIs in terms of data structures and API signatures. This is done to simplify the PDK implementation that uses the API, and to hide any communication protocol details. The API allows the CP-PDK to address and communicate with individual FEs.

1.2 Assumptions and Dependencies

The following assumptions and dependencies apply to this module:

- Most FP Plug-in API calls are asynchronous. Applications or API Clients must register callbacks for all API calls. This is because the PDK architecture supports remote FEs.
- The API callback model defines single register and de-register functions for all API calls. Differing callback types are used to distinguish between callbacks for different calls.

The data structures used in service-specific FP Plug-in APIs such as the IPv4 or MPLS APIs are the same as those defined in the respective NPF APIs (NPF IPv4 and NPF MPLS APIs).

1.3 Memory Management

The basic model for the FPP API or backend API is that whenever a data structure is passed to the API by the application or by the client, it makes an internal copy of the structure before the call returns, and vice-versa. Thus, the client can delete the data structure once the call returns. On the other hand, whenever the transport plug-in invokes a callback, it expects the application to make a copy of the data structure, which is passed in the callback. The plug-in deletes the structure after the callback returns.

1.4 References

Table 1 lists documents that are referenced in, or related to, this document.

Table 1. References

Reference	Document
[1]	Control Plane PDK Software Architecture Overview
[2]	Control Plane PDK API Framework Reference
[3]	Control Plane PDK Configuration and Management API Reference
[4]	Control Plane PDK ATM API Reference
[5]	NPF IPv4 API; Network Processing Forum (Implementation Agreement)
[6]	NPF IPv6 API; Network Processing Forum (Implementation Agreement)
[7]	NPF Differentiated Services API; Network Processing Forum (Implementation Agreement)

1.5 Terminology

Table 2 lists terms used in this document, and provides the expansion of each term.

Table 2. Terminology

Term	Description
AAL	ATM Adaptation Layer
ATM	Asynchronous Transfer Mode
CE	Control Element
CP	Control Plane
CPID	Connection Point Identifier
CP PDK	Control Plane Platform Development Kit
DiffServ	Differentiated Services
FE	Forwarding Element
ForCES	Forwarding and Control Element Separation protocol
FP	Forwarding Plane
ID	Identifier
IP	Internet Protocol
MFS	Maximum Frame Size

Term	Description
MIB	Management Information Base
MPLS	Multi-protocol Label Switching
NPF	Network Processing Forum
PDU	Protocol Data Unit
PVC	Permanent Virtual Connection
SVC	Switched Virtual Connection
TCP	Transmission Control Protocol
TM	Traffic Management
UBR	Unspecified Bit Rate
VC	Virtual Connection
VCC	Virtual Channel Connection
VCI	Virtual Channel Identifier
VPC	Virtual Path Connection
VPI	Virtual Path Identifier

Part 2: Generic Functions and Data Structures

2 Generic Functions and Data Structures

This section describes the generic functions and data structures used by the FP Plug-in API.

2.1 Data Types

Type

```
typedef uint32_t  FPPI_FEID;
```

Description

A unique FE ID generated by the transport plug-in. It must be used in subsequent FPP API calls to identify the FE.

Type

```
typedef uint32_t  FPPI_PortID;
```

Description

A unique port ID generated by the transport plug-in. It must be used in subsequent FPP API calls to identify a port.

Type

```
typedef uint32_t  FPPI_CBHANDLE;
```

Description

A unique callback ID generated by the transport plug-in when a callback is registered. It must be used in subsequent FPP API calls to identify the callback function to be invoked for that call.

Type

```
typedef void*     FPPI_CONTEXT;
```

Description

The client context passed by the client of the FPP API during callback registration, and returned to the client when the callback is invoked.

Type

```
typedef void*     FPPI_CORRELATOR;
```

Description

The client correlator passed by the client of the FPP API during the API call, and returned to the client when the callback is invoked for that particular call.

Type

```
typedef enum {
 FPPI_SUCCESS,
 FPPI_FAIL,
 FPPI_INVALID_PARAMETERS,
 FPPI_OUT_OF_MEMORY,
 FPPI_DUPLICATE_CONTEXT } FPPI_RET;
```

Description

The return value of FPP API.

Type

```
typedef enum { FPPI_SUCCESSFUL,
               FPPI_FAILURE } FPPI_Status;
```

Description

Reports the status of a call in the callback.

Type

```
typedef enum {
 FPPI_IPv4_AddRoute,
 FPPI_IPv4_DelRoute,
 FPPI_IPv4_PurgeRoute,
 FPPI_ARP_AddEntry,
 FPPI_ARP_DelEntry,
 FPPI_ARP_QueryEntry,
 FPPI_ARP_PurgeEntry,
 FPPI_IPv4_UniAddNextHop,
 FPPI_IPv4_UniDelNextHop,
 FPPI_IPv4_UniFlushNextHop,
 FPPI_IPv4_UniQueryNextHop,
 FPPI_IPv4_UniAddPrefix,
 FPPI_IPv4_UniDelPrefix,
 FPPI_IPv4_UniFlushPrefix,
 FPPI_IPv4_UniQueryPrefix,
 FPPI_IPv4_UniAddToL2,
 FPPI_IPv4_UniDelToL2,
 FPPI_IPv4_UniFlushToL2,
 FPPI_FEBind_Event,
 FPPI_FEUnBind_Event,
 FPPI_CEUnBind_Event,
 FPPI_LinkUp_Event,
 FPPI_LinkDown_Event,
 FPPI_ARP_AddEntry_Event,
```

```
  FPPI_ARP_DelEntry_Event,
  FPPI_FEPortCap_Event,
  FPPI_FECapability_Event,
  FPPI_CEReady_Event,
  FPPI_PacketIO_Event,
  FPPI_FE_GetProperties,
  FPPI_FE_SetProperties,
  FPPI_L2_GetProperties,
  FPPI_L2_SetProperties,
  FPPI_L2_DelProperties,
  FPPI_L2_GetStatistics,
  FPPI_L3_GetProperties,
  FPPI_L3_SetProperties,
  FPPI_L3_DelProperties,
  FPPI_L3_GetStatistics,
  FPPI_ICMP_GetStatistics
/* ATM  Enums */
   FPPI_ATM_IfAttrSet,
   FPPI_ATM_IfDelete,
   FPPI_ATM_IfEnable,
   FPPI_ATM_IfDisable,
   FPPI_ATM_IfGenStatsGet,
   FPPI_ATM_VccSet,
   FPPI_ATM_VccBind,
   FPPI_ATM_VccDelete,
   FPPI_ATM_VccEnable,
   FPPI_ATM_VccDisable,
   FPPI_ATM_VccOperStatusGet,
   FPPI_ATM_VccStatsGet,
   FPPI_ATM_AAL2_ChannelSet,
   FPPI_ATM_AAL2_ChannelDel
} NPF_CBCAT;
```

Description

Defines the callback types for which the client of the API can register the callbacks. There are different callback types for each different API call.

2.1.1 FP Plug-in API Generic Callback

Syntax

```
typedef void (*fppapi_callback) (
       FPPI_FEID         feid,
       FPPI_CORRELATOR   correlator,
       FPPI_CONTEXT      context,
       void              *response_data);
```

Description

The generic callback function defined for FP Plug-in API.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`context`	Unique value associated with the callback during registration
`response_data`	Void parameter containing the response data

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

2.2 Registering a Callback - Generic Prototype

Syntax

```
FPPI_RET FPPAPI_register_callback(
                fppapi_callback  callback,
                FPPI_CONTEXT     context,
                NPF_CBCAT        cbtype,
                FPPI_CBHANDLE*   cbhandle);
```

Description

Generic function used by the application to register a callback function for a particular event or call identified by the callback type.

Input Parameters

`callback`	The callback function being registered
`context`	Unique value passed back to the application during the callback
`cbtype`	The type of callback for which the function is being registered

Output Parameters

`cbhandle`	Unique value generated by the transport plug-in that should be used by the application to identify the callback function

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

2.3 De-registering a Callback - Generic Prototype

Syntax

```
FPPI_RET FPPAPI_deregister_callback(
         FPPI_CBHANDLE  cbhandle);
```

Description

Generic function used by the application to de-register a callback function

Input Parameters

`cbhandle`	Unique value passed to the application during callback registration

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Part 3: Initialization and Shutdown

3 Initialization and Shutdown

This section describes APIs used for initialization and shutdown of the transport plug-in module.

3.1 Initialization

Syntax

```
FPPI_RET FPPAPI_Initialize (void);
```

Description

Initializes the transport plug-in module in the control plane. This synchronous function returns only on completion of initialization. It initializes the module in terms of allocating memory for internal data structures. This function must be called before calling any other FP Plug-in API functions.

Input Parameters

None.

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

3.2 Shutdown

Syntax

```
FPPI_RET FPPAPI_Shutdown (void);
```

Description

Shuts down the transport plug-in module in the control plane. This is synchronous function performs the basic cleanup for the module, including de-allocating memory, closing server sockets, and shutting down the PCI driver.

Input Parameters

None.

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

3.3 Start

Syntax

```
FPPI_RET FPPAPI_Start (void);
```

Description

Starts the transport plug-in module in terms of receiving connections for FEs. This synchronous function opens a server socket if TCP/IP is used as the transport, or initializes the PCI driver if PCI is used.

Input Parameters

None.

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

3.4 Stop

Syntax

```
FPPI_RET FPPAPI_Stop (void);
```

Description

Stops the transport plug-in module from receiving any connection from the FEs. This synchronous function closes server sockets or shuts down the PCI driver.

Input Parameters

None.

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Part 4: Event Notification

4 Event Notification

This section describes APIs used by the transport plug-in module for event notification. This includes the FE bind/unbind events and the port up/down events.

4.1 FE Bind/Unbind Events

4.1.1 Registering Callback for FE Bind Event

The `FPPI_FEBind_Event` callback type is used to register for an FE Bind Event.

4.1.2 Registering Callback for FE Unbind Event

The `FPPI_FEUnBind_Event` callback type is used to register for an FE UnBind Event.

4.1.3 FE Bind Reply

Syntax

```
FPPI_RET FPPAPI_feBind_Reply(
         FPPI_FEID     feid,
         FPPI_Status   status);
```

Description

Responds to an FE Bind event. The client can decide to either accept or reject the FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`status`	Set to `SUCCESSFUL` or `FAILURE` depending on whether the client wants to accept or reject the FE

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

4.2 Port Up/Down Events

4.2.1 Registering Callback for Port State Change

The `FPPI_PortDown_Event` callback type is used to register for a port down event.

Part 5: Capability Discovery

5 Capability Discovery

This section describes APIs used by the transport plug-in module for capability discovery.

5.1 Registering Callback for FE Capability Event

The `FPPI_FECapability_Event` callback type is used to register for an FE capability event.

`FPPI_FE_Caps` is returned as the `response_data` in the callback. Refer the following definition:

```
#define FENAMELEN 4
typedef uint8_t FENAME[FENAMELEN];

typedef struct fpp_Port_Caps_tag {
               FPPI_PortID     portid;
               uint32_t        port_type;
               MACA            mac_addr;
               uint32_t        link_speed;
               uint32_t        mtu;
} FPPI_Port_Caps;

typedef struct fpp_Ports_Caps_tag {
               FPPI_Port_Caps  *port_caps_array;
               uint32_t        port_count;
} FPPI_Ports_Caps;

typedef struct fpp_FE_Caps_tag {
               FENAME          fe_name;
               FPPI_Ports_Caps ports_caps;
} FPPI_FE_Caps;
```

5.1.1 FE Capability Reply

Syntax

```
FPPI_RET FPPAPI_feCapability_Reply(
         FPPI_FEID      feid);
```

Description

This function is called after the FE sends its capabilities to the CE. It notifies the FE that the CE is ready to receive asynchronous events from the FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Part 6: IPv4/ARP API

6 IPv4/ARP API

This section describes APIs used by the transport plug-in module to provide functionality equivalent to the IPv4 API [5].

6.1 IPv4 API

6.1.1 Add Unicast Next-hop

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastAddNextHop (
         FPPI_FEID                          feid,
         FPPI_CORRELATOR                    correlator,
         FPPI_CBHANDLE                      cbhandle,
         NPF_uint32_t                       numEntries,
         FPPI_UnicastNextHopUpdateInfo_t*   nhInfo);
```

Description

Adds IPv4 unicast next-hop entries to an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`numEntries`	Number of entries to be added
`nhInfo`	Array of entries to add to the forwarding table, as defined in the IPv4 API

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`npf_IPv4EntryStatusList`, which is defined in the IPv4 API.

6.1.2 Delete Unicast Next-hop

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastDelNextHop(
         FPPI_FEID                          feid,
         FPPI_CORRELATOR                    correlator,
         FPPI_CBHANDLE                      cbhandle,
         uint32_t                           numEntries,
         FPPI_UnicastNextHopDeleteInfo_t*   nhInfo);
```

Description

Deletes IPv4 unicast next-hop entries from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`numEntries`	Number of entries in next-hop list
`nhInfo`	Next-hop list to delete

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`npf_IPv4EntryStatusList`, which is defined in the IPv4 API.

6.1.3 Flush Unicast Next-hop Table

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastFlushNextHop (
         FPPI_FEID              feid,
         FPPI_CORRELATOR        correlator,
         FPPI_CBHANDLE          cbhandle);
```

Description

Flushes all IPv4 unicast next-hop entries from the specified FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

6.1.4 Query Unicast Next-hop Entry

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastQueryNextHop (
         FPPI_FEID              feid,
         FPPI_CORRELATOR        correlator,
         FPPI_CBHANDLE          cbhandle,
         NPF_uint32_t           numEntries,
         NPF_uint32_t*          nhid);
```

Description

Queries all IPv4 unicast next-hop entries from the specified FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`numEntries`	Number of entries in the next hop ID list
`nhid`	The list of next hop IDs to be queried

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`NPF_IPv4UC_CallbackData_t`, which is defined in the IPv4 API.

6.1.5 Add Unicast Prefix

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastAddPrefix(
         FPPI_FEID                        feid,
         FPPI_CORRELATOR                  correlator,
         FPPI_CBHANDLE                    cbhandle,
         NPF_uint32_t                     numEntries,
         NPF_UnicastPrefixUpdateInfo_t*   prfxInfo);
```

Description

Adds IPv4 unicast prefix entries to an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`numEntries`	Number of entries in prefix list
`prfxInfo`	List of prefixes to add to a specified FE

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`npf_IPv4EntryStatusList`, which is defined in the IPv4 API.

6.1.6 Query Unicast Prefix

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastQueryPrefix(
         FPPI_FEID              feid,
         FPPI_CORRELATOR        correlator,
         FPPI_CBHANDLE          cbhandle,
         NPF_uint32_t           numEntries,
         NPF_IPv4UC_Prefix_t*   prfxInfo);
```

Description

Queries IPv4 unicast prefix entries from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`numEntries`	Number of entries in prefix list
`prfxInfo`	List of prefixes to query from the specified FE

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`NPF_IPv4UC_CallbackData_t`, which is defined in the IPv4 API.

6.1.7 Delete Unicast Prefix

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastDelPrefix(
         FPPI_FEID              feid,
         FPPI_CORRELATOR        correlator,
         FPPI_CBHANDLE          cbhandle,
         NPF_uint32_t           numEntries,
         NPF_IPv4UC_Prefix_t*   prfxInfo);
```

Description

Deletes IPv4 unicast prefix entries from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`numEntries`	Number of entries in prefix list
`prfxInfo`	List of prefixes to add to specified FE

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`npf_IPv4EntryStatusList`, which is defined in the IPv4 API.

6.1.8 Flush Unicast Prefix Table

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastFlushPrefix(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle);
```

Description

Flushes IPv4 unicast prefix entries from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

6.2 ARP API

6.2.1 Add ARP Entries

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastToL2Add (
         FPPI_FEID                        feid,
         FPPI_CORRELATOR                  correlator,
         FPPI_CBHANDLE                    cbhandle,
         NPF_uint32_t                     entry_count,
         FPPI_IPv4UniNHResolEntry_t*      entry_array);
```

Description

Adds ARP entries to an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`entry_count`	Number of entries in ARP list
`entry_array`	List of ARP entries to add to a specified FE

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`npf_IPv4EntryStatusList`, which is defined in the IPv4 API.

6.2.2 Delete ARP Entries

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastToL2Del(
         FPPI_FEID                          feid,
         FPPI_CORRELATOR                    correlator,
         FPPI_CBHANDLE                      cbhandle,
         NPF_uint32_t                       entry_count,
         FPPI_IPv4UniNHResolEntry_t*        entry_array
         );
```

Description

Deletes ARP entries from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`entry_count`	Number of entries in ARP list
`entry_array`	List of ARP entries

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`npf_IPv4EntryStatusList`, which is defined in the IPv4 API.

6.2.3 Flush ARP Entries

Syntax

```
FPPI_RET FPPAPI_ipv4_unicastToL2Flush(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle);
```

Description

Deletes all ARP entries from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Part 7: ATM API

7 ATM API

This section describes APIs used by the transport plug-in module to provide the functionality equivalent to the ATM API.

7.1 Set ATM Interface Attributes

Syntax

```
FPPI_RET FPPAPI_ATM_IfAttrSetFunc(
         FPPI_FEID             feid,
         FPPI_CORRELATOR       correlator,
         FPPI_CBHANDLE         cbhandle,
         IfATM_IfAtmAttr_t *   ifAtmAttr,
         NPF_uint32_t          count);
```

Description

Sets the ATM interface Attributes.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`ifAtmAttr`	Array of entries of ATM port attributes
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

The `ifATM_FpResponse_t` data type is:

```
typedef struct
{
      NPF_uint8_t status;           /* Status of response */
      NPF_uint8_t reason;           /* Reason of Failure */
}IfATM_FpResponse_t;
```

7.2 Delete ATM Interface

Syntax

```
FPPI_RET FPPAPI_ATM_IfDeleteFunc(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         FPPI_PortID *        portId,
         NPF_uint32_t         count);
```

Description

Deletes an ATM interface.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`portId`	Array of IDs of Ports to be deleted
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.3 Enable ATM Interface

Syntax

```
FPPI_RET FPPAPI_ATM_IfEnableFunc(
         FPPI_FEID                feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         FPPI_PortID *        portId,
         NPF_uint32_t         count);
```

Description

Enables an ATM interface.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`portId`	Array of entries of ATM Port IDs
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.4 Disable ATM Interface

Syntax

```
FPPI_RET FPPAPI_ATM_IfDisableFunc(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         FPPI_PortID *        portId,
         NPF_uint32_t         count);
```

Description

Disables an ATM interface

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`portId`	Array of entries of ATM Port ID
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.5 Get ATM Interface Statistics

Syntax

```
FPPI_RET FPPAPI_ATM_IfGenericStatsGetFunc(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         FPPI_PortID *        portId,
         NPF_uint32_t         count);
```

Description

Gets the generic ATM interface statistics.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`portId`	Array of entries of ATM Port IDs
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.6 Create an ATM VCC

Syntax

```
FPPI_RET FPPAPI_ATM_VccSetFunc(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         IfATM_IfVcc_t *      vccInfo,
         NPF_uint32_t         count);
```

Description

Creates an ATM Virtual Connection (VCC).

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`vccInfo`	Array of entries of ATM VCC attributes
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.7 Bind an ATM VCC to a Layer-3 interface

Syntax

```
FPPI_RET FPPAPI_ATM_VccBindFunc(
         FPPI_FEID             feid,
         FPPI_CORRELATOR       correlator,
         FPPI_CBHANDLE         cbhandle,
         IfATM_VccBind_t *     vccInfo,
         NPF_uint32_t          count);
```

Description

Binds an ATM VCC to a Layer-3 interface.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`vccInfo`	Array of entries of ATM VCC bind information elements
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.8 Delete an ATM VCC

Syntax

```
FPPI_RET FPPAPI_ATM_VccDeleteFunc(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         IfATM_VccAddr_t *    vccInfo,
         NPF_uint32_t         count);
```

Description

Deletes an ATM VCC

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`vccInfo`	Array of entries of ATM VCC Address type
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.9 Enable an ATM VCC

Syntax

```
FPPI_RET FPPAPI_ATM_VccEnableFunc(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         IfATM_VccAddr_t *    vccInfo,
         NPF_uint32_t         count);
```

Description

Enables an ATM VCC.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`vccInfo`	Array of entries of ATM VCC Address type
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.10 Disable an ATM VCC

Syntax

```
FPPI_RET FPPAPI_ATM_VccDisableFunc(
         FPPI_FEID             feid,
         FPPI_CORRELATOR       correlator,
         FPPI_CBHANDLE         cbhandle,
         IfATM_VccAddr_t *     vccInfo,
         NPF_uint32_t          count);
```

Description

Disables an ATM VCC.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`vccInfo`	Array of entries of ATM VCC Address type
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.11 Get Operational Status of an ATM VCC

Syntax

```
FPPI_RET FPPAPI_ATM_VccOperStatusGetFunc(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         IfATM_VccAddr_t *    vccInfo,
         NPF_uint32_t         count);
```

Description

Gets the operational status of an ATM VCC.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`vccInfo`	Array of entries of ATM VCC Address type
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.12 Get Statistics of an ATM VCC

Syntax

```
FPPI_RET FPPAPI_ATM_VccStatsGetFunc(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         IfATM_VccAddr_t *    vccInfo,
         NPF_uint32_t         count);
```

Description

Get the statistics of an ATM VCC.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`vccInfo`	Array of entries of ATM VCC Address type
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.13 Create an ATM AAL2 channel

Syntax

```
FPPI_RET FPPAPI_ATM_AAL2_ChannelSetFunc(
         FPPI_FEID             feid,
         FPPI_CORRELATOR       correlator,
         FPPI_CBHANDLE         cbhandle,
         IfATM_AAL2_ChannelInfo_t * aal2Info,
         NPF_uint32_t          count);
```

Description

Creates an ATM AAL2 channel.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`Aal2Info`	Array of entries of ATM AAL2 information elements
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

7.14 Delete an AAL2 channel

Syntax

```
FPPI_RET FPPAPI_ATM_AAL2_ChannelDeleteFunc(
         FPPI_FEID                feid,
         FPPI_CORRELATOR          correlator,
         FPPI_CBHANDLE            cbhandle,
         IfATM_AAL2_ConnId_t *    aal2Id,
         NPF_uint32_t             count);
```

Description

Deletes an AAL2 channel.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`Aal2Id`	Array of entries of ATM AAL2 IDs
`count`	Number of entries in the array

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`IfATM_FpResponse_t` information type for each ATM port attribute entry.

Part 8: Configuration and Management API

8 Configuration and Management API

This section describes APIs used by the transport plug-in module to provide the functionality equivalent to the Configuration and Management API. In particular, it describes L2 interface Management, and L3 configuration and management.

8.1 L2 - Interface Management

Syntax

```
typedef enum {
        NPF_L2ATTRS_ALL,
        NPF_L2ATTRS_MTU,
        NPF_L2ATTRS_LINKSPEED,
        NPF_L2ATTRS_STATUS,
        NPF_L2ATTRS_VPIBITS,
        NPF_L2ATTRS_VCIBITS
} NPF_L2ATTRS_TYPE;

typedef struct {
        NPF_L2ATTRS_TYPE        type;
        FPPI_PortID             portid;
        uint32_t                link_speed;
        uint32_t                mtu;
        int                     status;
        int                     vpiBits;
        int                     vciBits;
} npf_L2IntfAttrs_t;
```

8.1.1 Set L2 Interface Properties

Syntax

```
FPPI_RET FPPAPI_L2Interface_SetProperties(
         FPPI_FEID              feid,
         FPPI_CORRELATOR        correlator,
         FPPI_CBHANDLE          cbhandle,
         npf_L2IntfAttrs_t*     port_array,
         uint32_t               port_count);
```

Description

Sets L2 interface properties from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`port_array`	Array of `<port ID, L2_Attributes>` pairs
`port_count`	The number of entries contained in `port_array`

Return Values

FPPI_SUCCESS if successful, otherwise one of the error values listed in Section 2.1, Data Types.

8.1.2 Get L2 Interface Statistics

Syntax

```
FPPI_RET FPPAPI_L2Interface_GetStatistics(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         FPPI_PortID          *port_array,
         uint32_t             port_count);
```

Description

Gets the L2 interface statistics from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`port_array`	Array of port IDs
`port_count`	The number of entries contained in `port_array`

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`npf_L2StatsList_t`, which is defined in the Configuration and Management API.

8.2 L3 Configuration & Management

8.2.1 Set L3 Interface Properties

Syntax

```
FPPI_RET FPPAPI_L3Interface_SetProperties(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         npf_L3IntfAttrs_t*   port_array,
         uint32_t             port_count);
```

Description

Sets the L3 interface properties from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`port_array`	Array of `<port ID, L3_Attributes>` pairs
`port_count`	The number of entries contained in `port_array`

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

8.2.2 Delete L3 Interface Properties

Syntax

```
FPPI_RET FPPAPI_L3Interface_DelProperties(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         npf_L3IntfAttrs_t*   port_array,
         uint32_t             port_count);
```

Description

Deletes L3 interface properties from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`port_array`	Array of `<port ID, L3_Attributes>` pairs
`port_count`	The number of entries contained in `port_array`

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

8.2.3 Get L3 Interface Statistics

Syntax

```
FPPI_RET FPPAPI_L3Interface_GetStatistics(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         FPPI_PortID          *port_array,
         uint32_t             port_count);
```

Description

Gets the L3 interface statistics from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`port_array`	Array of port IDs
`port_count`	The number of entries contained in `port_array`

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`npf_L3StatsList_t`, which is defined in the Configuration and Management API.

8.3 ICMP

8.3.1 Get ICMP Statistics

Syntax

```
FPPI_RET FPPAPI_ICMP_GetStatistics(
         FPPI_FEID            feid,
         FPPI_CORRELATOR      correlator,
         FPPI_CBHANDLE        cbhandle,
         FPPI_PortID          *port_array,
         uint32_t             port_count);
```

Description

Gets the ICMP statistics from an FE.

Input Parameters

`feid`	Unique ID associated with an FE by the transport plug-in
`correlator`	Distinguishes between multiple invocations of the same API call
`cbhandle`	Unique value used to identify the callback function
`port_array`	Array of port IDs
`port_count`	The number of entries contained in `port_array`

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

Response Data in Callback

`npf_ICMPStatsList_t`, which is defined in the Configuration and Management API.

Part 9: Backend API

9 Backend API

The transport plug-in exposes a backend API, which is used by the forwarding plane PDK implementation. This section describes the backend API.

9.1 Generic Functions and Data Structures

This section describes the generic functions and data structures used by the Backend API.

9.1.1 Data Types

Syntax

```
typedef uint32_t  FPPI_CBCORRELATOR;
```

Description

The callback correlator generated by the transport plug-in, and passed to the client of the Backend API during callback invocation. This should be passed back by the client when it invokes the Report Status call for that particular callback.

9.1.2 Backend API Generic Callback

Syntax

```
typedef void (*backendapi_callback) (
        FPPI_CBCORRELATOR      cbcorrelator,
        FPPI_CONTEXT           context,
        void*                  response_data);
```

Description

Generic callback function defined for the Backend API.

Input Parameters

`cbcorrelator`	Described in Section 9.1.1, Data Types
`context`	Unique value associated with the callback during registration
`response_data`	Void parameter containing the response data

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

9.1.3 Registering a Callback - Generic Prototype

Syntax

```
FPPI_RET BENDAPI_register_callback(
         backendapi_callback  callback,
         FPPI_CBHANDLE*       cbhandle,
         NPF_CBCAT            cbtype,
         FPPI_CONTEXT         context);
```

Description

Generic function used by the application to register a callback function for a particular event or call identified by the callback type.

Input Parameters

`callback`	The callback function being registered
`cbhandle`	Unique value generated by the transport plug-in, which should be used by the application to identify the callback function
`cbtype`	The type of the callback for which the function is being registered
`context`	Unique value passed back to the application during the callback

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

9.1.4 De-registering a Callback - Generic Prototype

Syntax

```
FPPI_RET BENDAPI_deregister_callback(
         FPPI_CBHANDLE  cbhandle);
```

Description

Generic function used by the application to de-register a callback function.

Input Parameters

`cbhandle`	Unique value passed to the application during callback registration

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

9.2 Initialization and Shutdown

9.2.1 Initialization

Syntax

```
FPPI_RET BENDAPI_Initialize (void);
```

Description

Initializes the transport plug-in module in the forwarding plane. This synchronous function returns only after it completes initialization. It initializes the module in terms of allocating memory for internal data structures. This function must be called before calling any other Backend API functions.

Input Parameters

None.

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

9.2.2 Shutdown

Syntax

```
FPPI_RET BENDAPI_Shutdown (void);
```

Description

Shuts down the transport plug-in module in the forwarding plane. This is synchronous function performs the basic cleanup for the module, including de-allocating memory, and closing connections to the control plane.

Input Parameters

None.

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

9.3 Binding and Capability Notification

9.3.1 Sending Bind Request

Syntax

```
FPPI_RET BENDAPI_bindRequest(
        char*               ipaddr,
        char*               fename,
        FPPI_CBHANDLE       cbhandle);
```

Description

Sends a bind request to the control plane.

Input Parameters

`ipaddr`	IP address of the CE to which the bind request should be sent
`fename`	Name of the forwarding element to which the bind request should be sent
`cbhandle`	Unique value passed to the client during callback registration

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

9.3.2 Sending Unbind Request

Syntax

```
FPPI_RET BENDAPI_unbindRequest(void);
```

Description

Sends an unbind request to the control plane.

Input Parameters

None.

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

9.3.3 Capability Notification

Syntax

```
FPPI_RET BENDAPI_capabilityRequest(
         FPPI_FE_Caps         fe_caps,
         FPPI_CBHANDLE        cbhandle);
```

Description

Sends FE Capabilities to the control plane.

Input Parameters

`fe_caps`	The FE Capability structure defined previously
`cbhandle`	Unique value passed to the client during callback registration

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

9.4 Call/Event Notification from the Control Plane

The client of the Backend API (FE) registers for calls or events from the control plane, which are defined by `NPF_CBCAT` (refer Section 2.1, Data Types). The calls, such as `AddRouteEntry` made by the control plane are passed to the client or FE using the appropriate callback function that was registered. The status of the call is reported to the control plane using the Report Status call, after the call is executed on the FE.

9.5 Event Notification to the Control Plane

9.5.1 Generic Send Event

Syntax

```
FPPI_RET BENDAPI_sendEvent(
         NPF_CBCAT            event_type,
         void*                event_data);
```

Description

Notifies the control plane about asynchronous events in the FE, such as Port Down events.

Input Parameters

`event_type`	The type of the event which occurred on the FE
`event_data`	The data specific to the particular event

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

9.5.2 Send Packet

Syntax

```
FPPI_RET BENDAPI_sendPacket(
         npf_Packet_t         *packet_list,
         uint32_t             packet_count);
```

Description

Sends data packets to the control plane.

Input Parameters

`packet_list`	An array of `npf_Packet_t structs` to be sent to the CP
`packet_count`	The number of entries in `packet_array`

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.

9.6 Reporting Call Status

9.6.1 Report Status

Syntax

```
FPPI_RET BENDAPI_Report_Status(
         FPPI_Status          status,
         NPF_CBCAT            cbtype,
         FPPI_CBCORRELATOR    cbcorrelator,
         void                 *response_data);
```

Description

Notifies the control plane about the status of a call that was sent by the control plane to be executed on the forwarding plane.

Input Parameters

`status`	The status of the call (success or failure)
`cbtype`	The callback type for which the status is being reported
`cbcorrelator`	Described in Section 9.1.1, Data Types
`response_data`	Void parameter containing the response data

Return Values

`FPPI_SUCCESS` if successful, otherwise one of the error values listed in Section 2.1, Data Types.