Interface Management API Implementation Agreement

Revision 1.0

Editor(s):

John Renwick, Agere Systems, jrenwick@agere.com

The words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", “MAY", and "OPTIONAL" in the remainder of this document are to be interpreted as described in the NPF Software API Conventions Implementation Agreement revision 1.0.

Table of Contents

Table of Figures

1 Revision History

Revision	Date	Reason for Changes
1.0	09/13/2002	Created Rev 1.0 of the implementation agreement by taking the Interface Management APIs (npf2002.471.00) and making minor editorial corrections.

2 Introduction

A network element, for instance a router, has one or more physical connection points usually called *links*, through which it is connected to other network elements. Packets are received over a link by the network element for processing. A link usually has an associated framing protocol, or Layer 2 (L2) protocol, that is used to transfer packets over the media of the link. A L2 protocol typically either implements and/or negotiates standards based link characteristics e.g., link speed, maximum transfer unit (MTU), single or full duplex transmission mode, etc. PPP, Ethernet, etc. are examples of layer 2 protocols commonly deployed in today's networks. It is quite possible that more than one L2 protocol can be running on a single link e.g. PPP over Ethernet. Also, multiple L2 interfaces of the same type can be combined into a single logical L2 interface to create a trunk, as in 802.3ad link aggregation and other multilink techniques.

The figures below show some of the relationships that exist in today's networks. Other configurations are possible including combining the forms below into more deeply nested hierarchies.

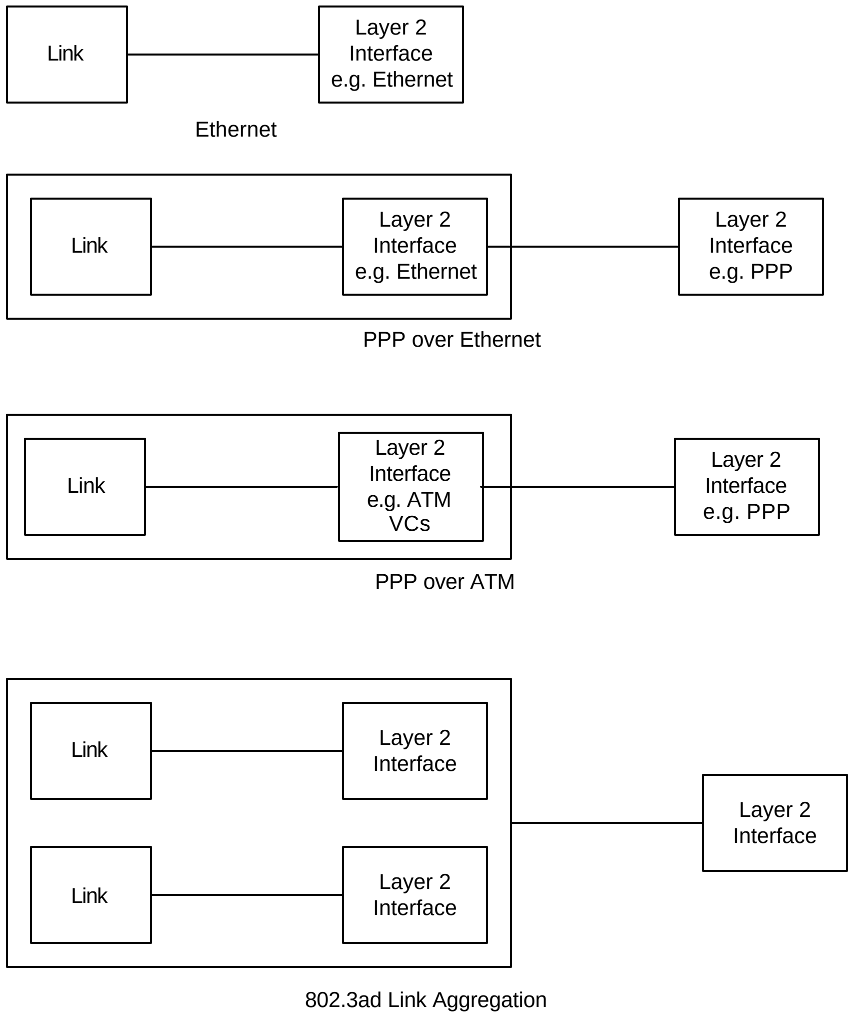
Link
Layer 2 Interface e.g. Ethernet
Ethernet
Link
Layer 2 Interface e.g. Ethernet
Layer 2 Interface e.g. PPP
PPP over Ethernet
Link
Layer 2 Interface e.g. ATM VCs
Layer 2 Interface e.g. PPP
PPP over ATM
Link
Layer 2 Interface
Link
Layer 2 Interface
Layer 2 Interface
802.3ad Link Aggregation

Figure 1 Layer 2 interface configurations

One or more Layer 3 protocols, for instance IPv4, IPv6 or IPX, can be used on an L2 interface. A L3 interface captures the properties of the corresponding L3 protocol. For example in case of IPv4, IP address and network prefix length are associated with the L3 interfaces.

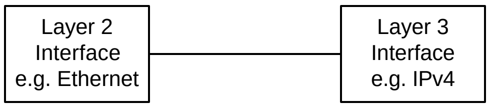

Figure 2 Layer 2 and Layer 3 Interface Relationship

There is a many to many relationship between L2 and L3 interfaces. Thus, as shown by Figure 3, multiple Layer 3 interfaces can be associated with a single L2 interface, and a single L3 interface can be associated with multiple L2 interfaces.

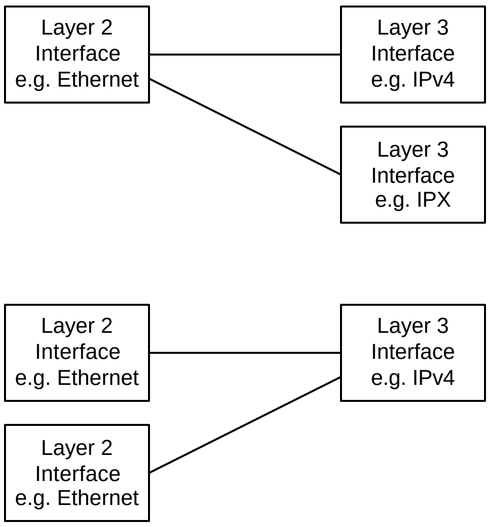

Figure 3 L2-L3 mappings

Figure 4 shows the overall relationships between links, L2 Interfaces and L3 Interfaces.

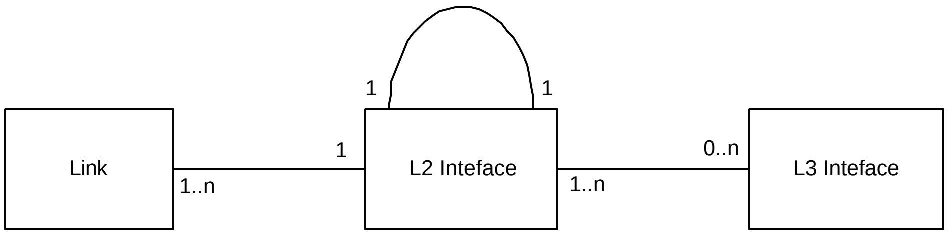

Figure 4 Link/L2/L3 relationship

The Interface Management API provides a uniform interface for configuring and managing the physical and logical interfaces a network processor may need to be aware of. For example, the API defined in this document will cover aspects of interface management related to Layer 2 (Bridging), Layer 3 (IP), VLAN, media-specific management (Ethernet, ATM UNI, SONET, etc.), and so on.

2.1 Assumptions and External Requirements

1. In the scope of this API, the structure and attributes of interfaces reflect what is needed by the forwarding plane, not by the application. Applications are expected to maintain their own representations of interfaces, very likely in a system that permits more and different interface attributes than are recognized by this API. The API specifically does not provide a way for the application to read back interface attributes or values it has previously given to the API.
2. Memory allocation and usage model for the API implementation will be as dictated by the

NPF Software Conventions Implementation Agreement.
3. The API does not determine any policy with respect to operations on interfaces or their events. It is assumed that policy will be embodied in an Interface Manager module that is part of the application. (See 2.4.8.)

2.2 Scope

The “interfaces” addressed by this API are those related to external network ports only. Other internal interfaces defined by NP Forum, such as streaming and lookaside interfaces, are outside the scope of this document.

2.3 Dependencies

The NP Forum IPv4 Unicast Forwarding API Implementation Agreement defines the Forwarding Information Base Handle, **`NPF_IPv4UnicastPrefixTableHandle_t`**.

2.4 Interface Management Structures

We represent an interface with a hierarchy of structures. At the top level is a structure containing attributes that can be set from the application, and are common to all interface types. Within that is a union containing objects that are attributes of a specific type or family of interface types. This nested structure contains only attributes that can be set on an interface. There are other structures for reading interface attributes, like statistics. See the individual structure descriptions below.

2.4.1 Common Interface Attributes

The common interface attribute structure contains the following that the application can set on most interfaces:

- Interface type code, which indicates which of several different type-specific groups of attributes are being used: LAN, IPv4, ATM UNI, POS.
- Administrative Status (up or down): a global enable/disable control on the interface
- Link speed

2.4.2 Common Interface Statistics

These attributes have a structure of their own, and can be retrieved by an application, but not set. They apply to all interface types.

- Counters (64 bits – wide enough “never”[1] to wrap):
  - Bytes received
  - Input packets (unicast)
  - Input packets (multicast)
  - Input packets (broadcast)
  - Input packets dropped
  - Input errors
  - Input packets of unknown protocol
  - Bytes sent
  - Output packets (unicast)
  - Output packets (multicast)
  - Output packets (broadcast)

[1] At OC768 sustained full speed, or 39813 megabits/second, a 64-bit byte counter will wrap in approximately 117 years.

  - Output packets dropped
  - Output errors

2.4.3 IPv4 Interface

This interface sub type represents a Layer 3 interface for IPv4, essentially a binding of IP attributes to a physical or logical port. Its application-settable attributes are:

- Primary IP address and prefix length
- MTU, or the largest IP datagram that can be sent on this interface
- FIB handle.

2.4.4 Generic LAN Interface

This sub-interface type represents all broadcast media types that use IEEE MAC addressing: all the Ethernet flavors, FDDI, etc. Its application-settable attributes are:

- Port number. This is a 32-bit value whose internal structure is implementation-dependent. It identifies the physical location of the LAN connector in terms of the system view: by chassis number and board number, for example.
- Promiscuous Mode flag (promiscuous mode means receiving all frames, regardless of destination MAC address)
- LAN speed selection or autoselect enable code.

2.4.5 ATM UNI Interface

This sub-interface type represents a SONET or DS3 port that carries ATM cells. This is a “UNI” port, or one that serves as an end-point for ATM virtual connections. Its application-settable attributes are:

- Port number. This is a 32-bit value whose internal structure is implementation-dependent. It identifies the physical location of the SONET or DS3 connector.

2.4.5.1 ATM Vcc

While there is no interface type for an ATM Virtual Connection, a number of Virtual Connections can belong to an ATM UNI interface. Each VCC has the following attributes:

- VPI/VCI address of this Vcc
- AAL type (AAL2, AAL3/4, AAL5/LLC, AAL5/VCMux, none)
- Parent interface handle
- Traffic shaping/policing profile handle

2.4.5.2 ATM Vcc Statistics

In addition to the interface-generic statistics available on an ATM UNI port, each Vcc has its own small set of counters that can be read, but not set, by an application:

- Bytes received
- Cells received
- Frames received
- Bytes transmitted
- Cells transmitted
- Frames transmitted

2.4.6 Packet Over Sonet (POS) Interface

This sub-interface type represents a point-to-point SONET link carrying datagrams encapsulated in PPP protocol. Its application-settable attributes are:

- Port number. This is a 32-bit value whose internal structure is implementation-dependent. It identifies the physical location of the SONET connector.

2.4.7 Interface Relatedness

The API includes a function (NPF_IfBind) that relates a pair of interfaces as parent and child. An interface can be at the same time the parent of one interface and a child of another. The API places no restriction on

- the number of levels of hierarchy
- the numer of child interfaces any can have
- the number of parent interfaces any can have
- the types of interfaces that can be bound together as parent and child.

This last point means, for instance, that binding a LAN interface as the parent of an IPv4 interface is permitted as far as the API specification is concerned, even though such a binding might make no sense in the context of a given implementation (for another implementation, it might make perfect sense). Implementations MAY place their own restrictions on the way interfaces of certain types can be related, in what multiplicity, and to what depth of hierarchy.

The API offers no way for an application to discover the relationships between interfaces; these are set by the application, and it must remember them if it needs this information. Similarly, the API specification takes no position on how these relationships might be represented within an implementation.

2.4.8 Interface Manager Application

Because interfaces can be related, an application may require that an event on one interface causes a related event to be registered on a related interface; or it may require that operations on related interfaces be done in a certain way, or in a certain order. The API imposes a few necessary restrictions on the order of operations (see section 4.5), but there are other matters of policy that belong to the application and are outside the scope of the API to regulate. Where there are significant policy considerations, the client application should include an Interface Manager module that brokers transactions or intercedes between the Interface Management API and its clients, and ensures that the application's requirements are satisfied.

3 Data Types

3.1 Interface Management API Types

```
/*
 *    The Interface structure:
 */
typedef struct  {
      NPF_IfType_t        type;           /* Logical interface type */
      NPF_uint64_t        speed;          /* Speed in Kbits/second */
      NPF_IfAdminStatus_t adminStatus;    /* Administrative up/down */
      union {             /* Type specific attributes (by if_type code) */
        NPF_IfIPv4_t      IPv4_Attr;      /* IPv4 Interface attributes */
        NPF_IfLAN_t       LAN_Attr;       /* LAN interface attributes */
        NPF_IfATM_t       ATM_Attr;       /* ATM UNI interface attributes */
        NPF_IfPOS_t       POS_Attr;       /* POS interface attributes */
      } u;
} NPF_IfGeneric_t;

/*
 * Interface handle
 */
typedef NPF_uint32_t    NPF_IfHandle_t;

typedef enum {
  NPF_IF_TYPE_UNK=1,                      /* Interface type unknown */
  NPF_IF_TYPE_LAN=2,                      /* Generic LAN interface */
  NPF_IF_TYPE_ATM=3,                      /* ATM interface */
  NPF_IF_TYPE_POS=4,                      /* Packet over SONET interface */
  NPF_IF_TYPE_IPV4=5,                     /* IPv4 logical interface */
  NPF_IF_TYPE_HIGHEST=5                   /* Highest defined value */
} NPF_IfType_t;

/*
 *    Structure to relate two interfaces
 */
typedef struct {
      NPF_IfHandle_t    parent;           /* Parent interface handle */
      NPF_IfHandle_t    child;            /* Child interface handle */
} NPF_IfBinding_t;

/*
 *    Statistics
 */
typedef struct {
      NPF_uint64_t      bytesRx;          /* Receive Bytes */
      NPF_uint64_t      ucPackRx;         /* Receive Unicast Packets */
      NPF_uint64_t      mcPackRx;         /* Receive Multicast Packets */
      NPF_uint64_t      bcPackRx;         /* Receive Broadcast packets */
      NPF_uint64_t      dropRx;           /* Receive packets dropped */
      NPF_uint64_t      errorRx;          /* Receive errors */
      NPF_uint64_t      protoRx;          /* Receive unknown protocol */
      NPF_uint64_t      bytesTx;          /* Transmit bytes */
      NPF_uint64_t      ucPackTx;         /* Transmit Unicast Packets */
      NPF_uint64_t      mcPackTx;         /* Transmit Multicast Packets */
      NPF_uint64_t      bcPackTx;         /* Transmit Broadcast Packets */
      NPF_uint64_t      dropTx;           /* Transmit dropped packets */
      NPF_uint64_t      errorTx;          /* Transmit errors */
} NPF_IfStatistics_t;

/*
 *    Operational Status code
 */
typedef enum      {
      NPF_IF_OPER_STATUS_UP = 1,          /* Operationally UP */
      NPF_IF_OPER_STATUS_DOWN = 2,        /* Operationally DOWN */
      NPF_IF_OPER_STATUS_UNKNOWN = 3      /* Status unknown */
} NPF_IfOperStatus_t;

/*
 *    Administrative Status code
 */
typedef enum      {
      NPF_IF_ADMIN_STATUS_UP = 1,         /* Administratively UP */
      NPF_IF_ADMIN_STATUS_DOWN = 2        /* Administratively DOWN */
} NPF_IfAdminStatus_t;

/*
 *    IPv4 Interface attributes
 */
typedef struct {
      NPF_IPv4NetAddr_t addr;             /* Primary IPv4 Address and plen */
      NPF_uint16_t      mtu;              /* IPv4 Max Transmission Unit */
      NPF_IPv4UnicastPrefixTableHandle_t FIB_Handle;/* Forwarding Info Base*/
} NPF_IfIPv4_t;

/*
 * IPv4 address prefix structure (Defined globally)
 */
typedef struct {
      NPF_IPv4Address_t IPv4Addr;         /* IPv4 address */
      NPF_uint8_t       IPv4NetPlen;      /* Prefix length in bits (1-32) */
} NPF_IPv4NetAddr_t;

/*
 * LAN Speed codes for setting Ethernet speed.
 */
typedef enum      {
      NPF_IF_LAN_SPEED_10M = 1,           /* 10 megabits/second */
      NPF_IF_LAN_SPEED_100M = 2,          /* 100 megabits/second */
      NPF_IF_LAN_SPEED_1G = 3,            /* 1 gigabit/second */
      NPF_IF_LAN_SPEED_10G = 4,           /* 10 gigabit/second */
      NPF_IF_LAN_SPEED_AUTO = 5           /* Set autosense */
} NPF_IfLAN_Speed_t;

/*
 *    Generic LAN Interface attributes
 */
typedef struct {
      NPF_uint32_t      port;             /* Port number */
      NPF_boolean_t     promisc;          /* Promiscuous Mode */
      NPF_IfLAN_Speed_t speed;            /* Speed control */
} NPF_IfLAN_t;
```

```
/*
 *    ATM UNI Interface Attributes
 */
typedef struct {
      NPF_uint32_t      port;             /* Port number */
} NPF_IfATM_t;

/*
 *    ATM UNI Vcc attributes
 */
typedef struct {
      NPF_VccAddr_t     vcc;              /* VPI/VCI of this Vcc */
      NPF_IfAAL_t       AAL;              /* AAL type */
      NPF_IfHandle_t    parent;           /* Parent interface handle */
      NPF_IfATM_QoS_t   QoS;              /* QoS profile */
} NPF_IfVcc_t;

/*
 *    ATM UNI VPI/VCI types
 */
typedef NPF_uint16_t NPF_VccVPI_t;        /* VPI is 8 or 12 bits */
typedef NPF_uint16_t NPF_VccVCI_t;        /* VCI is 16 bits */

typedef struct {        /* ATM Vcc Address (VPI/VCI) structure */
      NPF_VccVPI_t      VPI;              /* VPI number */
      NPF_VccVCI_t      VCI;              /* VCI number */
} NPF_VccAddr_t;

/*
 *    ATM UNI Vcc AAL type code
 */
typedef enum {
      NPF_IF_VCC_AAL1 = 1,                /* AAL 1 */
      NPF_IF_VCC_AAL2 = 2,                /* AAL 2 */
      NPF_IF_VCC_AAL3_4 = 3,              /* AAL 3&4 */
      NPF_IF_VCC_AAL5_LLC = 4,            /* AAL 5 with LLC/SNAP */
      NPF_IF_VCC_AAL5_VCMUX = 5           /* AAL 5, VC-multiplexed */
} NPF_IfAAL_t;

/*
 *    ATM UNI QOS profile – can be shared by multiple Vccs
 */
typedef struct {
      NPF_uint32_t      SCR;              /* Sust. cell rate in cells/sec */
      NPF_uint32_t      PCR;              /* Peak cell rate in cells/sec */
      NPF_uint32_t      MBS;              /* Max burst size in cells */
} NPF_IfATM_QoS_t;

/*
 *    ATM UNI Per-Vcc Statistics
 */
typedef struct {
      NPF_VccAddr_t     vccaddr;          /* VPI/VCI to which stats apply */
      NPF_uint64_t      bytesRx;          /* Received Bytes */
      NPF_uint64_t      cellsRx;          /* Received Cells */
      NPF_uint64_t      framesRx;         /* Received Frames */
      NPF_uint64_t      bytesTx;          /* Transmitted Bytes */
      NPF_uint64_t      cellsTx;          /* Transmitted Cells */
      NPF_uint64_t      framesTx;         /* Transmitted Frames */
} NPF_IfATM_VccStats_t;

/*
 *    Packet over SONET (POS) Interface Attributes
 */
typedef struct {
      NPF_uint32_t      port;             /* Port number */
} NPF_IfPOS_t;
```

3.2 Data Structures for Completion Callbacks

```
/*
 *    Completion Callback Types
 */
typedef enum NPF_IfCallbackType {
      NPF_IF_CREATE = 1,
      NPF_IF_DELETE = 2,
      NPF_IF_BIND = 3,
      NPF_IF_STATS_GET = 4,
      NPF_IF_VCC_STATS_GET = 5,
      NPF_IF_ATTR_SET = 6,
      NPF_IF_CREATE_AND_SET = 7,
      NPF_IF_ENABLE = 8,
      NPF_IF_DISABLE = 9,
      NPF_IF_OPER_STATUS_GET = 10,
      NPF_IF_LAN_SRC_ADDR_GET = 11,
      NPF_IF_LAN_SRC_ADDR_SET = 12,
      NPF_IF_LAN_ADDR_LIST_SET = 13,
      NPF_IF_LAN_ADDR_LIST_ADD = 14,
      NPF_IF_LAN_PROMISC_SET = 15,
      NPF_IF_LAN_PROMISC_CLEAR = 16,
      NPF_IF_LAN_FULL_DUPLEX_SET = 17,
      NPF_IF_LAN_FULL_DUPLEX_CLEAR = 18,
      NPF_IF_LAN_SPEED_SET = 19,
      NPF_IF_LAN_FLOW_CONTROL_TX_ENABLE = 20,
      NPF_IF_LAN_FLOW_CONTROL_TX_DISABLE = 21,
      NPF_IF_LAN_FLOW_CONTROL_RX_ENABLE = 22,
      NPF_IF_LAN_FLOW_CONTROL_RX_DISABLE = 23,
      NPF_IF_IPV4ADDR_SET = 24,
      NPF_IF_IPV4MTU_SET = 25,
      NPF_IF_IPV4FIB_SET = 26,
      NPF_IF_ATM_VCC_SET = 27,
      NPF_IF_ATM_VCC_BIND = 28,
      NPF_IF_ATM_VCC_DELETE = 29
} NPF_IfCallbackType_t;

/*
 *    An asynchronous response contains an interface handle,
 *    a error or success code, and in some cases a function-
 *    specific structure embedded in a union.  One or more of
 *    these is passed to the callback function as an array
 *    within the NPF_IfCallbackData_t structure (below).
 */
```

```
typedef struct    {                 /* Asynchronous Response Structure */
      NPF_IfHandle_t    ifHandle;   /* Interface handle for this response */
      NPF_IfErrorType_t error;      /* Error code for this response */
      union {                             /* Function-specific structures: */
        NPF_uint32_t          unused;     /* Default */
        NPF_uint32_t          arrayIndex; /* NPF_IfCreateAndSet index */
        NPF_IfStatistics_t   *ifStats;    /* NPF_IfGenericStatsGet() */
        NPF_IfOperStatus_t    operStat;   /* NPF_IfOperStatusGet() */
        NPF_ATM_VccStats_t   *vccStats;   /* NPF_IfVccStatsGet() */
        NPF_IfHandle_t        child;      /* NPF_IfBind(), handle=parent*/
        NPF_MAC_Address_t     MACaddr;    /* NPF_IfLAN_SrcAddrGet() */
        NPF_VccAddr_t         VccAddr;    /* NPF_IfATM_VccSet(), */
                                          /* NPF_IfATM_VccBind(), and */
                                          /* NPF_IfATM_VccDelete() */
      }     u;
} NPF_IfAsyncResponse_t;

/*
 *    The callback function receives the following structure containing
 *    one or more asynchronous responses from a single function call.
 *    There are several possibilities:
 *    1. The called function does a single request
 *       - n_resp = 1, and the resp array has just one element.
 *       - allOK = TRUE if the request completed without error
 *         and the only return value is the response code.
 *       - if allOK = FALSE, the "resp" structure has the error code.
 *    2. the called function supports an array of requests
 *       a. All completed successfully, at the same time, and the
 *          only returned value is the response code:
 *          - allOK = TRUE, n_resp = 0.
 *       b. Some completed, but not all, or there are values besides
 *          the response code to return:
 *          - allOK = FALSE, n_resp = the number completed
 *          - the "resp" array will contain one element for
 *            each completed request, with the error code
 *            in the NPF_IfAsyncResponse_t structure, along
 *            with any other information needed to identify
 *            which request element the response belongs to.
 *          - Callback function invocations are repeated in
 *            this fashion until all requests are complete.
 *          Responses are not repeated for request elements
 *          already indicated as complete in earlier callback
 *          function invocations.
 */
typedef struct {
      NPF_IfCallbackType_t    type;       /* Which function was called? */
      NPF_boolean_t           allOK;      /* TRUE if all completed OK */
      NPF_uint32_t            nResp;      /* Number of responses in array */
      NPF_IfAsyncResponse_t  *resp;       /* Pointer to response structures*/
} NPF_IfCallbackData_t;
```

3.3 Error Codes

```
/*
 *    Asynchronous error codes (returned in function callbacks)
 */
```

```
/* Callback/event reg. error */
#define NPF_IF_E_ALREADY_REGISTERED ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR)

/* Callback/event handle invalid */
#define NPF_IF_E_BAD_CALLBACK_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+1)

/* Callback function is NULL */
#define NPF_IF_E_BAD_CALLBACK_FUNCTION ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+2)

/* Invalid parameter */
#define NPF_IF_E_INVALID_PARAM ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+3)

/* Invalid child i/f handle */
#define NPF_IF_E_INVALID_CHILD_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+4)

/* Invalid parent i/f handle */
#define NPF_IF_E_INVALID_PARENT_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+5)

/* Invalid interface handle */
#define NPF_IF_E_INVALID_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+6)

/* Invalid VPI/VCI address */
#define NPF_IF_E_NO_SUCH_VCC ((NPF_IfErrorType_t) NPF_INTERFACES_BASE_ERR+7)

/* Invalid IP address */
#define NPF_IF_E_INVALID_IPADDR ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+8)

/* Invalid IP net prefix length */
#define NPF_IF_E_INVALID_NETPLEN ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+9)

/* Invalid IP MTU specification */
#define NPF_IF_E_INVALID_MTU ((NPF_IfErrorType_t) NPF_INTERFACES_BASE_ERR+10)

/* Invalid interface attribute */
#define NPF_IF_E_INVALID_ATTRIBUTE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+11)

/* Error – interface not created */
#define NPF_IF_E_NOT_CREATED ((NPF_IfErrorType_t) NPF_INTERFACES_BASE_ERR+12)

/* Invalid MAC address */
#define NPF_IF_E_INVALID_MAC_ADDR ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+13)

/* Invalid FIB handle */
#define NPF_IF_E_INVALID_FIB_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+14)
```

```
/* Invalid ATM UNI i/f handle */
#define NPF_IF_E_INVALID_ATM_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+15)

/* Invalid layer 3 i/f handle */
#define NPF_IF_E_INVALID_L3_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+16)

/* Array length <= 0 or too big */
#define NPF_IF_E_BAD_ARRAY_LENGTH ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+17)

/* Interface has no source addr. */
#define NPF_IF_E_NO_SRC_ADDRESS ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+18)

/* Invalid generic interface speed */
#define NPF_IF_E_INVALID_SPEED ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+19)

/* Invalid VPI/VCI */
#define NPF_IF_E_INVALID_VCC_ADDRESS ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+20)

/* Invalid Interface Type */
#define NPF_IF_E_INVALID_IF_TYPE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+21)

/* Invalid Administrative Status code */
#define NPF_IF_E_INVALID_ADMIN_STATUS ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+22)

/* Invalid Port number */
#define NPF_IF_E_INVALID_PORT_NUMBER ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+22)

/* Invalid Promiscuous Mode code */
#define NPF_IF_E_INVALID_PROMISC_MODE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+23)

/* Invalid LAN speed code */
#define NPF_IF_E_INVALID_LAN_SPEED ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+24)

/* Invalid ATM AAL code */
#define NPF_IF_E_INVALID_ATM_AAL ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+25)

/* Invalid ATM QoS specification */
#define NPF_IF_E_INVALID_ATM_QOS ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+26)

typedef NPF_uint32_t NPF_IfErrorType_t;
```

3.4 Data Structures for Event Notifications

```
/*
```

```
 *    Event types
 */
typedef enum {
      NPF_IF_UP = 1,                     /* Interface went oper UP */
      NPF_IF_DOWN = 2,                   /* Interface went oper DOWN */
      NPF_IF_COUNTER_DISCONTINUITY = 3    /* Counter discontinuity occurred*/
} NPF_IfEvent_t;

/*
 *    Event notification structure and array
 */
typedef struct NPF_IfEventData      {
      NPF_IfEvent_t  eventType;          /* Event type */
      NPF_IfHandle_t handle;             /* Interface handle */
} NPF_IfEventData_t;

typedef struct    {
      NPF_uint16_t       n_data;          /* Number of events in array */
      NPF_IfEventData_t *eventData;       /* Array of event notifications */
} NPF_IfEventArray_t;
```

4 Functions

The Interface management API will provide for setting the interface properties and reading statistics in accordance with the Interface MIB, RFC 2863 [1] and other MIBs (although it makes no attempt to support any MIB fully).

4.1 Completion Callback

4.1.1 Completion Callback Function

Syntax

```
typedef void (*NPF_IfCallbackFunc_t) (
      NPF_IN NPF_userContext_t      userContext,
      NPF_IN NPF_correlator_t       correlator,
      NPF_IN NPF_IfCallbackData_t  *ifCallbackData);
```

Description

The application registers this asynchronous response handling routine to the API implementation. The callback function is implemented by the application, and is registered to the API implementation through **`NPF_IfRegister()`** function.

The callback data structure contains an array of responses, so that callbacks for multiple interfaces or ATM UNI Vccs referenced in a single API function call can be aggregated into fewer (perhaps just one) callback function invocations. The application can expect to receive exactly the same number of responses (callback array elements) as the multiplicity of the request, but the responses may be spread over multiple callback function invocations. How the API implementation allocates responses to callback invocations is up to the API implementor.

As an optimization: if the implementation is able to return success indications (**`NPF_NO_ERROR`**) for all responses from a single request in a single invocation of the callback function, and there is no information to return besides the success/failure code: instead of returning an array of responses, the implementation SHALL return a simple code indicating that all requested actions completed without error. See section 3.2.

Input Parameters

- **`userContext:`** The context item that was supplied by the application when the completion callback function was registered.
- **`correlator:`** The correlator item that was supplied by the application when the an API function call was made. The correlator is used by the application mainly to distinguish between multiple invocations of the same function.
- **`ifCallbackData`**: Pointer to a structure containing an array of response information related to the API function call. Contains information that is common among all functions, as well as information specific to a particular function. See **`NPF_IfCallbackData_t`** definition for details.

Output Parameters

None.

Return Codes

None.

4.1.2 Completion Callback Registration Function

Syntax

```
NPF_error_t NPF_IfRegister(
      NPF_IN  NPF_userContext_t     userContext,
      NPF_IN  NPF_IfCallbackFunc_t  ifCallbackFunc,
      NPF_OUT NPF_callbackHandle_t *ifCallbackHandle);
```

Description

This function is used by an application to register its completion callback function for receiving asynchronous responses related to API function calls. The application may register multiple callback functions using this function. The callback function is identified by the pair of **`userContext`** and **`ifCallbackFunc`**, and for each individual pair, a unique **`ifCallbackHandle`** will be assigned for future reference. Since the callback function is identified by both **`userContext`** and **`ifCallbackFunc`**, duplicate registration of the same callback function with different **`userContext`** is allowed. Also, the same **`userContext`** can be shared among different callback functions. Duplicate registration of the same **`userContext`** and **`ifCallbackFunc`** pair has no effect, will output a handle that is already assigned to the pair, and will return **`NPF_IF_E_ALREADY_REGISTERED`**.

Note: **`NPF_IfRegister()`** is a synchronous function and has no completion callback associated with it.

Input Parameters

- **`userContext`**: A context item for uniquely identifying the context of the application registering the completion callback function. The exact value will be provided back to the registered completion callback function as its first parameter when it is called. Application can assign any value to the `userContext` and the value is completely opaque to the API implementation.
- **`ifCallbackFunc`**: Pointer to the completion callback function to be registered.

Output Parameters

- **`ifCallbackHandle`**: A unique identifier assigned for the registered userContext and ifCallbackFunc pair. This handle will be used by the application to specify which callback to be called when invoking asynchronous API functions. It will also be used when de-registering the `userContext` and **`ifCallbackFunc`** pair.

Return Codes

- **`NPF_NO_ERROR`**: The registration completed successfully.
- **`NPF_IF_E_BAD_CALLBACK_FUNCTION`**: `ifCallbackFunc` is `NULL`.
- NPF_IF_E_ALREADY_REGISTERED: No new registration was made since the `userContext` and `ifCallbackFunc` pair was already registered.
  Note: Whether this should be treated as an error or not is dependent on the application.

4.1.3 Completion Callback Deregistration Function

Syntax

```
NPF_error_t NPF_IfDeregister(
      NPF_IN NPF_callbackHandle_t ifCallbackHandle);
```

Description
This function is used by an application to de-register a pair of user context and callback function. After the Deregister function returns, no more function calls can be made using the deregistered callback handle.

Input Parameters

- **`ifCallbackHandle:`** The unique identifier representing the pair of user context and callback function to be de-registered.

Output Parameters
None.

Return Codes

- **`NPF_NO_ERROR`**: The de-registration completed successfully.
- **`NPF_IF_E_BAD_CALLBACK_HANDLE`**: The API implementation does not recognize the callback handle. There is no effect to the registered callback functions.

4.2 Event Notification

4.2.1 Event Handler Function

Syntax

```
typedef void (*NPF_IfEventHandlerFunc_t) (
      NPF_IN NPF_userContext_t  userContext,
      NPF_IN NPF_IfEventArray_t ifEventArray);
```

Description
This handler function is for the application to register an event handling routine to the API implementation. One or more events can be notified to the application through a single invocation of this event handler function. Information on each event is represented in an array in the **`ifEventArray`** structure, where application can traverse through the array and process each of the events. This event handler function is intended to be implemented by the application, and be registered to the API implementation through **`NPF_IfEventRegister()`** function.

Note: This function may be called any time after **`NPF_IfEventRegister()`** is called for it.

Input Parameters

- **`userContext`**: The context item that was supplied by the application when the event handler function was registered.
- **`ifEventArray`**: Data structure that contains an array of event information. See **`NPF_IfEventArray_t`** definition for details.

Output Parameters
None.

Return Codes
None.

4.2.2 Event Handler Registration Function

Syntax

```
NPF_error_t NPF_IfEventRegister(
      NPF_IN  NPF_userContext_t           userContext,
      NPF_IN  NPF_IfEventHandlerFunc_t    ifEventHandlerFunc,
      NPF_OUT NPF_IfEventHandlerHandle_t *ifEventHandlerHandle);
```

Description

This function is used by an application to register its event handler function for receiving asynchronous event notifications from this API. Application may register multiple handler functions using this function. The event handler function is identified by the pair of **`userContext`** and **`ifEventHandlerFunc`**, and for each individual pair, a unique **`ifEventHandlerHandle`** will be assigned for future reference. Since the event handler function is identified by both **`userContext`** and **`ifEventHandlerFunc`**, duplicate registration of same event handler function with different **`userContext`** is allowed. Also, same **`userContext`** can be shared among different event handler functions. Duplicate registration of the same **`userContext`** and **`ifEventHandlerFunc`** pair has no effect, and will output a handle that is already assigned to the pair, and will return **`NPF_IF_E_ALREADY_REGISTERED`**.

Notes: Besides registering a handler function, this call enables events. The handler function could be called at any time following the invocation of `IfEventRegister()`. **`NPF_IfEventRegister()`** is a synchronous function and has no completion callback associated with it.

Input Parameters

- **`userContext`**: A context item for uniquely identifying the context of the application registering the event handler function. The exact value will be provided back to the registered event handler function as its 1st parameter when it is called. Application can assign any value to the userContext and the value is completely opaque to the API implementation.
- **`ifEventHandlerFunc`**: Pointer to the event handler function to be registered.

Output Parameters

- **`ifEventHandlerHandle`**: A unique identifier assigned for the registered `userContext` and `ifEventHandlerFunc` pair. This handle will be used by the application de-registering the `userContext` and `ifEventHandlerFunc` pair.

Return Codes

- **`NPF_NO_ERROR`**: The registration completed successfully.
- **`NPF_IF_E_BAD_CALLBACK_HANDLE`**: `ifEventHandlerFunc` is `NULL` or not recognized.
- **`NPF_IF_E_ALREADY_REGISTERED`**: No new registration was made since the `userContext` and `ifEventHandlerFunc` pair was already registered.
- Note: Whether this should be treated as an error or not is dependent on the application.

4.2.3 Event Handler Deregistration Function

Syntax

```
      NPF_error_t NPF_IfEventDeregister(
            NPF_IN NPF_IfEventHandlerHandle_t ifEventHandlerHandle);
```

Description

This function is used by an application to de-register a pair of user context and event handler function.

Input Parameters

- **`ifEventHandlerHandle`**: The unique identifier representing the pair of user context and event handler function to be de-registered.

Output Parameters

None.

Return Codes

- **`NPF_NO_ERROR:`** The de-registration completed successfully.
- **`NPF_E_BAD_CALLBACK_HANDLE:`** The API implementation does not recognize the event handler handle. There is no effect to the registered event handler functions.

4.3 Interface Management API

The API includes the following functions:

- **`NPF_IfCreate()`** creates one or more interfaces of a given type and returns a new interface handle for each.
- **`NPF_IfDelete()`** removes one or more interfaces and invalidates their handles.
- **`NPF_IfBind()`** binds a higher layer interface to lower layer interface, for one or more interface pairs.
- **`NPF_IfGenericStatsGet()`** returns the Statistics Structure containing the current value of the generic interface counters for one or more interfaces.
- **`NPF_IfVccStatsGet()`** returns an ATM Vcc Statistics structure for one or more Vccs on a single ATM UNI interface.
- **`NPF_IfAttrSet()`** sets all the settable generic and type-specific attributes for one or more interfaces.
- **`NPF_IfCreateAndSet()`** creates one or more interfaces and sets their attributes.
- **`NPF_IfEnable()`** Sets AdminUp status on one or more interfaces.
- **`NPF_IfDisable()`** Clears AdminUp status on one or more interfaces.
- **`NPF_IfOperStatusGet()`** returns the operational status of one or more interfaces.
- **`NPF_IfLAN_SrcAddrGet()`** returns the source MAC address of one or more LAN interfaces.
- **`NPF_IfLAN_SrcAddrSet()`** sets the source MAC address of one or more LAN interfaces.
- **`NPF_IfLAN_MAC_RcvAddrListSet()`** sets a list of receive MAC addresses on one or more LAN interfaces.
- **`NPF_IfLAN_MAC_RcvAddrListAdd()`** adds to the list of receive MAC addresses on one or more LAN interfaces.
- **`NPF_IfLAN_PromiscSet()`** sets Promiscuous mode on one or more LAN interfaces.
- **`NPF_IfLAN_PromiscClear()`** clears Promiscuous mode on one or more LAN interfaces.
- **`NPF_IfLAN_FullDuplexSet()`** sets full duplex mode on one or more Ethernet LAN interfaces.
- **`NPF_IfLAN_FullDuplexClear()`** clears full duplex mode on one or more Ethernet LAN interfaces.

- **`NPF_IfLAN_SpeedSet()`** sets the speed or Autosense mode on one or more Ethernet LAN interfaces.
- **`NPF_IfLAN_FlowControlTxEnable()`** enables one or more Ethernet LAN interfaces to transmit flow control messages.
- **`NPF_IfLAN_FlowControlTxDisable()`** disables one or more Ethernet LAN interfaces from sending flow control messages.
- **`NPF_IfLAN_FlowControlRxEnable()`** enables one or more Ethernet LAN interfaces to respond to received flow control messages.
- **`NPF_IfLAN_FlowControlRxDisable()`** disables one or more Ethernet LAN interfaces from responding to received flow control messages.
- **`NPF_IfIPv4AddrSet()`** sets the Primary IP address and prefix length for one or more IPv4 interfaces.
- **`NPF_IfIPv4MTUSet()`** sets the MTU on one or more IPv4 interfaces.
- **`NPF_IfIPv4FIBSet()`** associates IPv4 forwarding tables (FIBs) with one or more IPv4 interfaces.
- **`NPF_IfATM_VccSet()`** adds one or more Vccs to a single ATM UNI interface, or modifies existing Vccs.
- **`NPF_IfATM_VccBind()`** binds a higher layer interface to one or more Vccs on a single ATM UNI interface.
- **`NPF_IfATM_VccDelete()`** deletes one or more Vccs from a single ATM UNI interface.

4.4 Event definition signature

NPF Interfaces can generate the following events:

- **`NPF_IF_UP`** indicates the interface's OperUp status became FALSE
- **`NPF_IF_DOWN`** indicates the interface's OperUp status became TRUE
- **`NPF_IF_COUNTER_DISCONTINUITY`** indicates a discontinuity occurred in one or more of the statistics counters belonging to the interface. This event is intended to help a MIB implementation support **`ifCounterDiscontinuityTime`** (RFC 2863 [1]).

4.5 Order of Operations

There are a few restrictions on the order of operations on interfaces:

1. **`NPF_IfCreate()`** or **`NPF_IfCreateAndSet()`** must precede any other operations on an interface, because those functions assign the **`if_Handle`** value required by all other functions.
2. **`NPF_IfATM_VccSET()`** must precede any other operations on an ATM UNI Vcc.
3. There are no other restrictions, except as may be imposed by a particular implementation.

4.6 Completion Callbacks and Error Returns

Each of the functions defined in section 4.7 can return an immediate error, and each makes asynchronous callbacks. The only error codes eligible for immediate return are those defined in "NPF Software API Conventions Implementation Agreement". They are:

- **`NPF_NO_ERROR:`** This value is returned when a function was successfully invoked.

- **`NPF_E_UNKNOWN:`** An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative.
- **`NPF_BAD_CALLBACK_HANDLE:`** A function was invoked with a callback handle that did not correspond to a valid NPF callback handle as returned by a registration function, or a callback handle was registered with a registration function belonging to a different API than the function call where the handle was passed in.
- **`NPF_E_BAD_CALLBACK_FUNCTION:`** A callback registration was invoked with a function pointer parameter that was invalid.

All other error codes must be returned in an asynchronous callback response. They are defined in section 4.7 with the definitions of the functions that return them.

4.7 Interface Management API

This section will define functions for querying and modifying the interface properties and attributes.

Note: These functions follow a convention permitting multiple interface handles or ATM Vcc addresses to be passed for action in a single function invocation. In each case there is an argument that indicates the size of the array of interface handles or addresses. No limit on the size of such arrays is specified by this agreement; however an implementation MAY impose a size limit of its own choosing. If an application exceeds such limit, the implementation SHALL return the response code **`NPF_IF_E_BAD_ARRAY_LENGTH`** synchronously.

4.7.1 Function to Create an Interface

Syntax

```
NPF_error_t  NPF_IfCreate(
       NPF_IN NPF_callbackHandle_t   if_cbHandle,
       NPF_IN NPF_correlator_t       if_cbCorrelator,
       NPF_IN NPF_errorReporting_t   if_errorReporting,
       NPF_IN NPF_uint32_t           n_if,
       NPF_IN NPF_IfType_t           if_Type);
```

Description of function

This function creates one or more interfaces of a given type, including “typeless” (type unknown). Interfaces created by this function are in the Administratively Disabled (**`NPF_IF_ADMIN_STATUS_DOWN`**) state by default. The newly created interfaces are all alike, and blank except for type. The callback function will receive as many handles as **`NPF_IfCreate()`** could successfully create, and error codes for the rest. The created interfaces are undifferentiated until you set some attributes in them using **`NPF_IfAttrSet()`** or other functions.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application’s context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_if`**: number of interfaces to create.
- **`if_Type`**: the interface type: **`NPF_IF_TYPE_LAN`**, **`NPF_IF_TYPE_IPv4`**, **`NPF_IF_TYPE_ATM`**, **`NPF_IF_TYPE_POS`**, or **`NPF_IF_TYPE_UNK`**. All interfaces created by one function invocation are of the same type.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`** : operation successful.
- **`NPF_IF_E_INVALID_PARAM`**: operation failed, interface not created.

Asynchronous response

A total of **`n_if`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains the new interface handle or a possible error code. The union in the callback response structure is unused.

4.7.2 Function to Delete an Interface

Syntax

```
NPF_error_t  NPF_IfDelete(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function deletes one or more interfaces. The handle may not be used after this call returns.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to delete.
- **`if_HandleArray`**: pointer to an array of handles of the interfaces to be deleted.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_PARAM`**: Interface not deleted.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains the handle of the deleted interface, or a possible error code. The union in the callback response structure is unused.

4.7.3 Function to Bind Interfaces

Syntax

```
NPF_error_t  NPF_IfBind(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t if_       nbinds,
      NPF_IN NPF_IfBinding_t       *if_bindArray);
```

Description of function

This function binds one or more pairs of interfaces in parent-child relationships. Each binding associates two interfaces with each other, one as parent, and one as child. Multiple bindings can be made in a single call. An interface can have multiple parents; it can also have multiple children. Such relationships are indicated by multiple one-to-one binding entries, since a single many-to-one binding entry is not supported. An interface can be at the same time the parent of one and the child of another. An implementation SHOULD return an error if cycles occur (e.g. an interface is the child of one of its own children: "I'm my own grandpa"). An implementation MAY limit how many associations an interface can have, or restrict the depth of the hierarchy.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`if_nbinds`**: number of bindings in the array.
- **`if_bindArray`**: pointer to an array of interface handle parent/child bindings.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_CHILD_HANDLE`**: Child Handle is null or invalid; no binding done.
- **`NPF_IF_E_INVALID_PARENT_HANDLE`**: Parent Handle is null or invalid; no binding done.
- **`NPF_IF_E_INVALID_PARAM`**; Binding failed.No binding done.

Asynchronous response

A total of **`n_binds`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains the parent interface handle and a possible error code. The particular binding to which the response code pertains is identified in the callback by the two handles: the parent handle is in the usual **`ifHandle`** position, and the child handle is in the union part of the callback structure.

4.7.4 Function to Read Interface Statistics

Syntax

```
NPF_error_t  NPF_IfGenericStatsGet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function returns, via a callback, a pointer to a generic interface statistics structure containing the current counter values for one or more indicated interfaces.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.

- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to get statistics for.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE:`** An if_Handle is null or invalid.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle or a possible error code. If the error code indicates success, the union in the callback response structure contains a pointer to the **`NPF_IfStatistics_t`** structure for that interface.

4.7.5 Function to Read ATM UNI Vcc Statistics

Syntax

```
NPF_error_t  NPF_IfVccStatsGet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_Handle,
      NPF_IN NPF_uint32_t           n_Vccs,
      NPF_IN NPF_VccAddr_t         *if_VccAddrArray);
```

Description of function

This function returns, via a callback, a pointer to an ATM UNI Vcc statistics structure containing the current counter values for each of one or more indicated ATM UNI Vccs on a single ATM UNI interface.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`if_Handle`**: the handle of an interface of type **`NPF_IF_TYPE_ATM`**.
- **`n_Vccs`**: the number of Vccs to get statistics for.
- **`if_VccAddrArray`**: pointer to an array of VPI/VCI addresses.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: **`if_Handle`** is null or invalid, or is not an ATM UNI interface.
- **`NPF_IF_E_NO_SUCH_VCC`**: Indicated Vcc does not exist.

Asynchronous response

A total of **`n_Vccs`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code. If the error code indicates success, the union in the response structure contains a pointer to a Vcc statistics structure (**`NPF_ATM_VccStats_t`**) containing the Vcc's VPI/VCI address and its counter values.

4.7.6 Function to Set all Interface Attributes

Syntax

```
NPF_error_t  NPF_IfAttrSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray,
      NPF_IN NPF_IfGeneric_t       *if_StructArray);
```

Description of function

This function sets all the attributes of one or more interfaces, from the contents of an array of structures passed by the caller, as defined in **`NPF_IfGeneric_t`**. Ownership of the structure memory remains with the caller (the API implementation must copy all needed contents before returning). Any single attribute can be set with its own function call; this function is included as a way to set multiple attributes atomically and efficiently. Note: the number of **`NPF_IfGeneric_t`** structures and the number of interface handles in the two arrays must be the same, equal to the **`n_handles`** argument. This function sets a *different* set of attributes for each named interface.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to set attributes for.
- **`if_HandleArray`**: pointer to an array of interface handles.
- **`if_StructArray`**: pointer to a structurean array of structures containing the new interface attributes.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_ATTRIBUTE`**: An attribute (other than those mentioned below) was invalid.

Generic Interface Errors:

- **`NPF_IF_E_INVALID_HANDLE`**: if_Handle is null or invalid.
- **`NPF_IF_E_INVALID_SPEED`**: Invalid interface speed parameter.
- **`NPF_IF_E_INVALID_IF_TYPE`**: Invalid interface type code.
- **`NPF_IF_E_INVALID_ADMIN_STATUS`**: Invalid administrative status code.

IPv4 Interface Errors:

- NPF_IF_E_INVALID_IPADDR: Invalid IP address
- **`NPF_IF_E_INVALID_NETPLEN`**: Invalid network prefix length
- **`NPF_IF_E_INVALID_MTU`**: MTU value is invalid.
- NPF_IF_E_INVALID_FIB_HANDLE: FIB handle is invalid.

LAN Interface Errors:

- **`NPF_IF_E_INVALID_PORT`**: Port specification is invalid.
- **`NPF_IF_E_INVALID_PROMISC_MODE`**: Promiscuous Mode code is invalid.
- **`NPF_IF_E_INVALID_LAN_SPEED`**: LAN speed code is invalid.

ATM and POS Interface Errors:

- **`NPF_IF_E_INVALID_PORT`**: Port specification is invalid.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.7 Function to Create an Interface and Set All of its Attributes

Syntax

```
NPF_error_t  NPF_IfCreateAndSet(
      NPF_IN NPF_callbackHandle_t if_cbHandle,
      NPF_IN NPF_correlator_t if_cbCorrelator,
      NPF_IN NPF_errorReporting_t if_errorReporting,
      NPF_IN NPF_uint32_t n_if,
      NPF_IN NPF_IfGeneric_t *if_StructArray);
```

Description of function

This function simultaneously creates and sets all the attributes of one or more interfaces, from the contents of an array of structures passed by the caller (**`NPF_IfGeneric_t`**). Each interface is created with a *different* set of attributes. Ownership of the structure memory remains with the caller (the API implementation must copy all contents before returning).

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_if`**: the number of interfaces to set attributes for.
- **`if_StructArray`**: pointer to an array of structures containing the new interface attributes.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_ATTRIBUTE`**: An attribute (other than those mentioned below) was invalid.

Generic Interface Errors:

- **`NPF_IF_E_INVALID_HANDLE`** : if_Handle is null or invalid.
- **`NPF_IF_E_INVALID_SPEED`**: Invalid interface speed parameter.
- **`NPF_IF_E_INVALID_IF_TYPE`**: Invalid interface type code.
- **`NPF_IF_E_INVALID_ADMIN_STATUS`**: Invalid administrative status code.

IPv4 Interface Errors:

- **`NPF_IF_E_INVALID_IPADDR:`** Invalid IP address
- **`NPF_IF_E_INVALID_NETPLEN`**: Invalid network prefix length
- **`NPF_IF_E_INVALID_MTU`**: MTU value is invalid.
- **`NPF_IF_E_INVALID_FIB_HANDLE:`** FIB handle is invalid.

LAN Interface Errors:

- **`NPF_IF_E_INVALID_PORT`**: Port specification is invalid.
- **`NPF_IF_E_INVALID_PROMISC_MODE`**: Promiscuous Mode code is invalid.
- **`NPF_IF_E_INVALID_LAN_SPEED`**: LAN speed code is invalid.

ATM and POS Interface Errors:

- **`NPF_IF_E_INVALID_PORT`**: Port specification is invalid.

Asynchronous response

A total of **`n_if`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains the new interface handle and a success code or a possible error code if an interface could not be created or any attributes could not be set. Responses are linked to interface attributes in the following way: for each response, the union in the response structure contains the corresponding index of the **`if_StructArray`** element that contained its attributes. For example, the response for the first array element will include an Interface Handle and an **`arrayIndex`** value of zero; the response for the tenth array element an **`arrayIndex`** of 9, and so on.

4.7.8 Function to Enable an Interface

Syntax

```
NPF_error_t  NPF_IfEnable(
       NPF_IN NPF_callbackHandle_t   if_cbHandle,
       NPF_IN NPF_correlator_t       if_cbCorrelator,
       NPF_IN NPF_errorReporting_t   if_errorReporting,
       NPF_IN NPF_uint32_t           n_handles,
       NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function administratively enables one or more interfaces: if the interface is operationally ready, it can now send and receive packets.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application’s context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to enable.

- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE:`** An if_Handle is null or invalid.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.9 Function to Disable an Interface

Syntax

```
NPF_error_t  NPF_IfDisable(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function disables one or more interfaces, administratively (but not operationally). Once disabled, it can no longer send or receive packets.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to disable.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE:`** an `if_Handle` is null or invalid.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.10 Function to Return the Operational Status of an Interface

Syntax

```
NPF_error_t  NPF_IfOperStatusGet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function returns the operational status of one or more interfaces.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to query.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE:`** An `if_Handle` is null or invalid.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. If the code indicates success, the union in the callback response structure contains the operational status of the interface.

4.7.11 Function to Return a LAN Interface's Source MAC Address

Syntax

```
NPF_error_t  NPF_IfLAN_SrcAddrGet(
          NPF_IN NPF_callbackHandle_t   if_cbHandle,
          NPF_IN NPF_correlator_t       if_cbCorrelator,
          NPF_IN NPF_errorReporting_t   if_errorReporting,
          NPF_IN NPF_uint32_t           n_handles,
          NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function returns, via a callback, the Source MAC address of a LAN interface. The Source MAC address is the address to be sent by default as Source Address in packets sent by the interface, and the default Destination Address to "listen" for in received packets.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to get the MAC address for.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: an **`if_Handle`** is null or invalid, or is not a LAN interface.
- **`NPF_IF_E_NO_SRC_ADDRESS`**: No source address is present in this interface.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. If the code indicates success, the union in the callback response structure contains the Source MAC address of the interface.

4.7.12 Function to Set a LAN Interface's Source MAC Address

Syntax

```
NPF_error_t  NPF_IfLAN_SrcAddrSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray,
      NPF_IN NPF_MAC_Address_t     *if_LAN_SrcAddrArray);
```

Description of function

This function sets the Source MAC address of one or more LAN interfaces. If more than one LAN interface is specified, each receives its MAC address from a different element of the source address array. The Source MAC address is the address to be sent by default as Source Address in packets sent by the interface, and the default Destination Address to "listen" for in received packets.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to set the MAC addresses for.
- **`if_HandleArray`**: pointer to an array of interface handles.
- **`if_LAN_SrcAddrArray`**: pointer to an array of MAC addresses containing one address for each interface handle.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: an **`if_Handle`** is null or invalid, or is not a LAN interface.
- **`NPF_IF_E_INVALID_MAC_ADDR`**: Source MAC address is invalid.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.13 Function to Set a LAN Interface's List of Receive MAC Addresses

Syntax

```
NPF_error_t  NPF_IfLAN_MAC_RcvAddrListSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray,
      NPF_IN NPF_uint32_t           if_n_MAC_Addrs,
      NPF_IN NPF_MAC_Address_t     *if_LAN_AddrArray);
```

Description of function

This function sets a list of receive MAC addresses (represented as an array) for one or more interfaces. The given list replaces any list already present in the interface. If the **`if_n_MAC_Addrs`** parameter is zero, the entire list is deleted. The same address list is applied to all indicated interfaces.

Note: it is not necessary to include the Source MAC Address in an interface's list of receive MAC addresses; LAN interfaces should receive on this address by default, and the list of receive MAC addresses is in addition to the source address.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to set the address list for.
- **`if_HandleArray`**: pointer to an array of interface handles.
- **`if_n_MAC_Addrs`**: number of MAC addresses in the array
- **`if_LAN_AddrArray`**: pointer to the array of receive MAC addresses to set on all the interfaces.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: an **`if_Handle`** is null or invalid, or is not a LAN interface.
- **`NPF_IF_E_INVALID_MAC_ADDR`**: One or more of the MAC addresses is invalid for receiving.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.14 Function to Add to a LAN Interface's List of Receive MAC Addresses

Syntax

```
NPF_error_t  NPF_IfLAN_MAC_RcvAddrListAdd(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray,
      NPF_IN NPF_uint32_t           if_n_MAC_Addrs,
      NPF_IN NPF_MAC_Address_t     *if_LAN_AddrArray);
```

Description of function

This function adds addresses to a list of receive MAC addresses (represented as an array) in thefor one or more interfaces. The same list of addresses is added to each interface.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to add the addresses to.
- **`if_HandleArray`**: the handle of the interface.
- **`if_n_MAC_Addrs`**: number of MAC addresses in the array
- **`if_LAN_AddrArray`**: pointer to an array of receive MAC addresses to add to all the indicated interfaces.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful
- **`NPF_IF_E_INVALID_HANDLE`**: an **`if_Handle`** is null or invalid, or not an LAN interface.
- **`NPF_IF_E_INVALID_MAC_ADDR`**: One or more of the MAC addresses is invalid for receiving.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.15 Function to Set a LAN Interface in Promiscuous Mode

Syntax

```
NPF_error_t  NPF_IfLAN_PromiscSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function puts one or more LAN interfaces in "promiscuous" mode (i.e. receives all packets, regardless of the destination MAC address).

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces in the **`if_HandleArray`** array.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: an **`if_Handle`** is null or invalid, or is not a LAN interface.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.16 Function to Turn off Promiscuous Mode in a LAN Interface

Syntax

```
NPF_error_t  NPF_IfLAN_PromiscClear(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function reverses the effect of **`NPF_IfLAN_PromiscSet()`** on one or more interfaces.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces in the **`if_Handle`** array.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: an **`if_Handle`** is null or invalid, or is not a LAN interface.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.17 Optional Function to Set Full Duplex Mode on a LAN Interface

Syntax

```
NPF_error_t NPF_IfLAN_FullDuplexSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function sets Full Duplex Mode on one or more LAN interfaces. It is meaningful on 10/100 megabit Ethernet interfaces, where this option is selectable. Implementation of this function is optional.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces in the **`if_HandleArray`** array.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not a LAN interface.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.18 Optional Function to Disable Full Duplex Mode on a LAN Interface

Syntax

```
NPF_error_t NPF_IfLAN_FullDuplexClear(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function reverses the effect of NPF_IfLAN_FullDuplexSet() on one or more LAN interfaces. It is meaningful on 10/100 megabit Ethernet interfaces, where this option is selectable. Implementation of this function is optional.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces in the **`if_HandleArray`** array.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not a LAN interface.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.19 Optional Function to Set Interface Speed or Autosense on a LAN Interface

Syntax

```
NPF_error_t NPF_IfLAN_SpeedSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray,
      NPF_IN NPF_IfLAN_Speed_t     *speedArray);
```

Description of function

This function sets the speed (10 or 100 megabit/second) or Autosense on one or more LAN interfaces. It is meaningful on 10/100 megabit Ethernet interfaces, where this option is selectable. Implementation of this function is optional.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces in the **`if_HandleArray`** array.
- **`if_HandleArray`**: pointer to an array of interface handles.
- **`speedArray`**: pointer to an array of LAN Speed codes, one for each interface in the **`if_HandleArray`** array.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`** – Operation successful.

- **`NPF_IF_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not a LAN interface.
- **`NPF_IF_E_INVALID_SPEED`**: One of the codes in the **`speedArray`** array is not valid.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.20 Optional Function to Enable Sending Flow Control on a LAN Interface

Syntax

```
NPF_error_t NPF_IfLAN_FlowControlTxEnable(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function enables sending flow control messages on one or more LAN interfaces. It is meaningful on 10/100 megabit interfaces in full duplex mode, or on gigabit Ethernet interfaces. Implementation of this function is optional.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces in the **`if_HandleArray`** array.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not a LAN interface.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.21 Optional Function to Disable Sending Flow Control on a LAN Interface

Syntax

```
NPF_error_t NPF_IfLAN_FlowControlTxDisable(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
          NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function disables the sending of flow control messages on one or more LAN interfaces. It is meaningful on 10/100 megabit interfaces in full duplex mode, or on gigabit Ethernet interfaces. Implementation of this function is optional.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces in the **`if_HandleArray`** array.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not a LAN interface.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.22 Optional Function to Enable Receiving Flow Control on a LAN Interface

Syntax

```
NPF_error_t NPF_IfLAN_FlowControlRxEnable(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function enables responding to received flow control messages on one or more LAN interfaces. It is meaningful on 10/100 megabit interfaces in full duplex mode, or on gigabit Ethernet interfaces. Implementation of this function is optional.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces in the **`if_HandleArray`** array.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not a LAN interface.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.23 Optional Function to Disable Receiving Flow Control on a LAN Interface

Syntax

```
NPF_error_t NPF_IfLAN_FlowControlRxDisable(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

Description of function

This function disables responding to received flow control messages on one or more LAN interfaces. It is meaningful on 10/100 megabit interfaces in full duplex mode, or on gigabit Ethernet interfaces. Implementation of this function is optional.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces in the **`if_HandleArray`** array.
- **`if_HandleArray`**: pointer to an array of interface handles.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not a LAN interface.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.24 Function to Set an IPv4 Interface's IP Address and Prefix length

Syntax

```
NPF_error_t  NPF_IfIPv4AddrSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray,
      NPF_IN NPF_IPv4NetAddr_t     *if_IPv4AddrArray);
```

Description of function

This function sets the Primary IP address and network prefix length of one or more IPv4 interfaces. The if_Handle and if_IPv4Addr arrays must both contain the same number of entries, equal to the value of n_handles. The address of each interface is set from a *different* element of the address array.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to set the IP address for.
- **`if_HandleArray`**: pointer to an array of interface handles.
- **`if_IPv4AddrArray`**: pointer to an array of IPv4 address/length structures.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not an IPv4 interface.
- **`NPF_IF_E_INVALID_IPADDR`**: IP address is not a valid unicast address.
- **`NPF_IF_E_INVALID_NETPLEN`**: Invalid network prefix length.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.25 Function to Set an IPv4 Interface's MTU

Syntax

```
NPF_error_t  NPF_IfIPv4MTUSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_uint32_t           n_handles,
      NPF_IN NPF_IfHandle_t        *if_HandleArray,
      NPF_IN NPF_uint16_t          *if_IPv4MTU_Array);
```

Description of function

This function sets the IPv4 Maximum Transmission Unit (MTU) of one or more IPv4 interfaces. The **`if_HandleArray`** and **`if_IPv4MTU_Array`** arrays must both contain the same number of entries, equal to the value of n_handles. The MTU of each interface is set from a *different* element of the MTU array.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to set the IP address for.
- **`if_HandleArray`**: the handle of each interface.
- **`if_IPv4MTU_Array`**: the corresponding MTU values to be set.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not an IPv4 interface.
- **`NPF_IF_E_INVALID_MTU`**: MTU value is invalid.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.26 Function to Associate a FIB to an IPv4 Interface

Syntax

```
NPF_error_t  NPF_IfIPv4FIBSet(
      NPF_IN NPF_callbackHandle_t              if_cbHandle,
      NPF_IN NPF_correlator_t                  if_cbCorrelator,
      NPF_IN NPF_errorReporting_t              if_errorReporting,
      NPF_IN NPF_uint32_t                      n_handles,
      NPF_IN NPF_IfHandle_t                   *if_HandleArray,
      NPF_IN NPF_IPv4UnicastPrefixTableHandle_t if_FIB_Handle);
```

Description of function

This function associates an IPv4 Forwarding Table (FIB) with one or more IPv4 interfaces. The new FIB handle becomes the **`if_FIB_Handle`** attribute of the interface. A single FIB is associated with all the interfaces in the **`if_HandleArray`** array.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`n_handles`**: the number of interfaces to set the FIB for.
- **`if_HandleArray`**: pointer to an array of interface handles.
- **`if_FIB_Handle`**: the new FIB handle.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation complete.
- **`NPF_IF_E_INVALID_HANDLE`**: An **`if_Handle`** is null or invalid, or is not an IPv4 interface.
- **`NPF_IF_E_INVALID_FIB_HANDLE:`** Invalid FIB handle.

Asynchronous response

A total of **`n_handles`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure is unused.

4.7.27 Function to Add or Modify an ATM UNI Vcc

Syntax

```
NPF_error_t  NPF_IfATM_VccSet(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_Handle,
      NPF_IN NPF_uint32_t           n_Vccs,
      NPF_IN NPF_IfVcc_t           *if_VccStructArray);
```

Description of function

This function adds one or more Vccs to a single ATM UNI interface, or modifies existing Vccs where the interface already has a Vcc with the given VPI/VCI address. Ownership of the Vcc structure memory stays with the caller: the API implementation must copy all Vcc attributes before returning.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`if_Handle`**: the handle of the interface.
- **`n_Vccs`**: the number of Vccs to set.
- **`if_VccStructArray`**: pointer to an array of ATM Vcc attribute structures.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: **`if_Handle`** is null or invalid, or is not an ATM UNI interface.
- **`NPF_IF_E_INVALID_VCC_ADDRESS`**: ATM Vcc address is invalid.
- **`NPF_IF_E_INVALID_ATM_AAL`**: ATM AAL type code is invalid.
- **`NPF_IF_E_INVALID_PARENT_HANDLE`**: Invalid handle of parent interface on an ATM Vcc.
- **`NPF_IF_E_INVALID_ATM_QOS`**: Invalid ATM QoS specification for a Vcc.
- **`NPF_IF_E_INVALID_ATTRIBUTE:`** Invalid attribute.

Asynchronous response

A total of **`n_Vccs`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure contains the VPI/VCI address of the pertinent Vcc.

4.7.28 Function to Bind a higher layer Interface to an ATM Vcc

Syntax

```
NPF_error_t  NPF_IfATM_VccBind(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_ATM_Handle,
      NPF_IN NPF_IfHandle_t         if_ParentHandle,
      NPF_IN NPF_uint32_t           n_Vccs,
      NPF_IN NPF_VccAddr_t         *if_VccAddrArray);
```

Description of function

This function associates a single higher-layer interface with one or more Vccs on the same ATM UNI interface. The Vccs must have been created before this call is made.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`if_ATM_Handle`**: the handle of the ATM interface.
- **`if_ParentHandle`**: the handle of the higher layer interface to bind.
- **`n_Vccs`**: the number of Vccs to set.
- **`if_VccAddrArray`**: pointer to an array of VPI/VCI addresses.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_ATM_HANDLE`**: Invalid ATM UNI interface handle.
- **`NPF_IF_E_INVALID_L3_HANDLE`**: Invalid L3 interface handle.
- **`NPF_IF_E_NO_SUCH_VCC`**: Vcc does not exist.

Asynchronous response

A total of **`n_Vccs`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure contains the VPI/VCI address of the pertinent Vcc.

4.7.29 Function to Delete an ATM Vcc

Syntax

```
NPF_error_t  NPF_IfATM_VccDelete(
      NPF_IN NPF_callbackHandle_t   if_cbHandle,
      NPF_IN NPF_correlator_t       if_cbCorrelator,
      NPF_IN NPF_errorReporting_t   if_errorReporting,
      NPF_IN NPF_IfHandle_t         if_Handle,
      NPF_IN NPF_uint32_t           n_Vccs,
      NPF_IN NPF_VccAddr_t         *if_VccAddrArray);
```

Description of function

This function deletes one or more Vccs on a single ATM UNI interface.

Input Parameters

- **`if_cbHandle`**: the registered callback handle.
- **`if_cbCorrelator`**: the application's context for this call.
- **`if_errorReporting`**: the desired callback.
- **`if_Handle`**: the handle of the ATM UNI interface.
- **`n_Vccs`**: the number of Vccs to delete.
- **`if_VccAddrArray`**: pointer to an array of VPI/VCI addresses of Vccs to be deleted.

Output Parameters

None

Asynchronous Error Codes

- **`NPF_NO_ERROR`**: Operation successful.
- **`NPF_IF_E_INVALID_HANDLE`**: **`if_Handle`** is null or invalid, or is not an ATM UNI interface.

Asynchronous response

A total of **`n_Vccs`** asynchronous responses (**`NPF_IfAsyncResponse_t`**) will be passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface. The union in the callback response structure contains the VPI/VCI address of the pertinent Vcc.

5 References

1 McCloughrie, K., Kastenholtz, F., “The Interfaces Group MIB”, Internet Engineering Task Force RFC 2863, June 2000.

6 API Capabilities

This section defines the capabilities of the Interface Management API.
It summarizes the defined APIs and Events and defines the mandatory and optional features.

6.1 Optional support of specific types

The support of any specific type of interface is optional in an implementation. An implementation MAY support exclusively one type of interface, and still claim compliance to the NP Forum Interface Management API.

6.2 API Functions

Function Name	**Required?**
`NPF_IfRegister()`	Yes
`NPF_IfDeregister()`	Yes
`NPF_IfEventRegister()`	Yes
`NPF_IfEventDeregister()`	Yes
`NPF_IfCreate()`	Yes
`NPF_IfDelete()`	Yes
`NPF_IfBind()`	Yes
`NPF_IfGenericStatsGet()`	Yes
`NPF_IfVccStatsGet()`	Only if ATM UNI interfaces supported
`NPF_IfAttrSet()`	Yes
`NPF_IfCreateAndSet()`	Yes
`NPF_IfEnable()`	Yes
`NPF_IfDisable()`	Yes
`NPF_IfOperStatusGet()`	Yes
`NPF_IfLAN_SrcAddrGet()`	Only if LAN interfaces supported
`NPF_IfLAN_SrcAddrSet()`	Only if LAN interfaces supported
`NPF_IfLAN_MAC_RcvAddrListSet()`	Only if LAN interfaces supported
`NPF_IfLAN_MAC_RcvAddrListAdd()`	Only if LAN interfaces supported
`NPF_IfLAN_PromiscSet()`	Only if LAN interfaces supported
`NPF_IfLAN_PromiscClear()`	Only if LAN interfaces supported
`NPF_IfLAN_FullDuplexSet()`	No
`NPF_IfLAN_FullDuplexClear()`	No
`NPF_IfLAN_SpeedSet()`	No
`NPF_IfLAN_FlowControlTxEnable()`	No
`NPF_IfLAN_FlowControlTxDisable()`	No
`NPF_IfLAN_FlowControlRxEnable()`	No
`NPF_IfLAN_FlowControlRxDisable()`	No
`NPF_IfIPv4AddrSet()`	Only if IPv4 interfaces supported
`NPF_IfIPv4MTUSet()`	Only if IPv4 interfaces supported
`NPF_IfIPv4FIBSet()`	Only if IPv4 interfaces supported

Function Name	**Required?**
`NPF_IfATM_VccSet()`	Only if ATM UNI interfaces supported
`NPF_IfATM_VccBind()`	Only if ATM UNI interfaces supported
`NPF_IfATM_VccDelete()`	Only if ATM UNI interfaces supported

6.3 API Events

Event Name	**Required?**
`NPF_IF_UP`	Yes
`NPF_IF_DOWN`	Yes
`NPF_IF_COUNTER_DISCONTINUITY`	Yes

7 Appendix A

Header File: npf_if.h

```
/*
 * This header file defines typedefs, constants, and functions
 * that apply to the NPF Interface Management API
 */
#ifndef __NPF_IF_H_
#define __NPF_IF_H_

#ifdef __cplusplus
extern "C"  {
#endif

typedef     NPF_uint32_t      NPF_IfHandle_t;

/*
 *    Interface Types
 */
typedef enum {
      NPF_IF_TYPE_UNK=1,            /* Interface type unknown */
      NPF_IF_TYPE_LAN=2,            /* Generic LAN interface */
      NPF_IF_TYPE_ATM=3,            /* ATM UNI interface */
      NPF_IF_TYPE_POS=4,            /* Packet over SONET interface */
      NPF_IF_TYPE_IPV4=5,           /* IPv4 logical interface */
      NPF_IF_TYPE_HIGHEST=5         /* Highest defined value */
} NPF_IfType_t;

/*
 *    Operational Status code, returned in the asynchronous
 *    response from NPF_IfOperStatusGet().
 */
typedef enum      {
      NPF_IF_OPER_STATUS_UP = 1,    /* Operationally UP */
      NPF_IF_OPER_STATUS_DOWN = 2,  /* Operationally DOWN */
      NPF_IF_OPER_STATUS_UNKNOWN = 3/* Status unknown */
} NPF_IfOperStatus_t;

/*
 *    Administrative Status code, used in the NPF_IfGeneric_t
 *    structure passed to NPF_IfAttrSet() and NPF_IfCreateAndSet().
 */
typedef enum      {
      NPF_IF_ADMIN_STATUS_UP = 1,   /* Administratively UP */
      NPF_IF_ADMIN_STATUS_DOWN = 2  /* Administratively DOWN */
} NPF_IfAdminStatus_t;

/*
 *    Structure to relate two interfaces, passed to NPF_IfBind().
 */
typedef struct {
      NPF_IfHandle_t    parent;     /* Parent interface handle */
      NPF_IfHandle_t    child;      /* Child interface handle */
} NPF_IfBinding_t;

/*
```

```
 *    Statistics, from MIB-II Interface Group (RFC 2863), returned
 *    by asynchronous callback from NPF_IfGenericStatsGet().
 */
typedef struct {
      NPF_uint64_t      bytesRx;    /* Receive Bytes */
      NPF_uint64_t      ucPackRx;   /* Receive Unicast Packets */
      NPF_uint64_t      mcPackRx;   /* Receive Multicast Packets */
      NPF_uint64_t      bcPackRx;   /* Receive Broadcast packets */
      NPF_uint64_t      dropRx;     /* Receive packets dropped */
      NPF_uint64_t      errorRx;    /* Receive errors */
      NPF_uint64_t      protoRx;    /* Receive unknown protocol */
      NPF_uint64_t      bytesTx;    /* Transmit bytes */
      NPF_uint64_t      ucPackTx;   /* Transmit Unicast Packets */
      NPF_uint64_t      mcPackTx;   /* Transmit Multicast Packets */
      NPF_uint64_t      bcPackTx;   /* Transmit Broadcast Packets */
      NPF_uint64_t      dropTx;     /* Transmit dropped packets */
      NPF_uint64_t      errorTx;    /* Transmit errors */
} NPF_IfStatistics_t;

/*
 *    IPv4 Interface attributes, used in the union within
 *    NPF_IfGeneric_t.
 */
typedef struct {
      NPF_IPv4NetAddr_t addr;       /* Primary IPv4 Address and plen */
      NPF_uint16_t      mtu;        /* IPv4 Max Transmission Unit */
      NPF_IPv4UnicastPrefixTableHandle_t FIB_Handle;/* Forwarding Info Base
*/
} NPF_IfIPv4_t;

/*
 * LAN Speed codes for setting Ethernet speed.
 */
typedef enum      {
      NPF_IF_LAN_SPEED_10M = 1,           /* 10 megabits/second */
      NPF_IF_LAN_SPEED_100M = 2,          /* 100 megabits/second */
      NPF_IF_LAN_SPEED_1G = 3,            /* 1 gigabit/second */
      NPF_IF_LAN_SPEED_10G = 4,           /* 10 gigabit/second */
      NPF_IF_LAN_SPEED_AUTO = 5           /* Set autosense */
} NPF_IfLAN_Speed_t;

/*
 *    Generic LAN Interface attributes, used in the union within
 *    NPF_IfGeneric_t.
 */
typedef struct {
      NPF_uint32_t      port;       /* Port number */
      NPF_boolean_t     promisc;    /* Promiscuous Mode */
      NPF_IfLAN_Speed_t speed;      /* Speed control */
} NPF_IfLAN_t;

/*
 *    ATM VPI/VCI types, used in various structures to
 *    identify ATM Vccs.
 */
typedef NPF_uint16_t NPF_VccVPI_t;  /* VPI is 8 or 12 bits */
typedef NPF_uint16_t NPF_VccVCI_t;  /* VCI is 16 bits */
typedef struct {        /* ATM Vcc Address (VPI/VCI) structure */
      NPF_VccVPI_t      VPI;        /* VPI number */
      NPF_VccVCI_t      VCI;        /* VCI number */
} NPF_VccAddr_t;

/*
 *    ATM UNI Vcc AAL type code, used in the ATM Vcc
 *    attribute structure, NPF_IfVcc_t.
 */
typedef enum {
      NPF_IF_VCC_AAL1 = 1,          /* AAL 1 */
      NPF_IF_VCC_AAL2 = 2,          /* AAL 2 */
      NPF_IF_VCC_AAL3_4 = 3,        /* AAL 3&4 */
      NPF_IF_VCC_AAL5_LLC = 4,      /* AAL 5 with LLC/SNAP */
      NPF_IF_VCC_AAL5_VCMUX = 5     /* AAL 5, VC-multiplexed */
} NPF_IfAAL_t;

/*
 *    ATM UNI QOS profile -- can be shared by multiple Vccs,
 *    used in the ATM Vcc attribute structure, NPF_IfVcc_t.
 */
typedef struct {
      NPF_uint32_t      SCR;        /* Sust. cell rate in cells/sec */
      NPF_uint32_t      PCR;        /* Peak cell rate in cells/sec */
      NPF_uint32_t      MBS;        /* Max burst size in cells */
} NPF_IfATM_QoS_t;

/*
 *    ATM UNI Interface Attributes, used in the union within
 *    NPF_IfGeneric_t.
 */
typedef struct {
      NPF_uint32_t      port;       /* Port number */
} NPF_IfATM_t;

/*
 *    ATM Vcc attributes, passed to NPF_IfATM_VccSet().
 */
typedef struct {
      NPF_VccAddr_t     vcc;        /* VPI/VCI of this Vcc */
      NPF_IfAAL_t       AAL;        /* AAL type */
      NPF_IfHandle_t    parent;     /* Parent interface handle */
      NPF_IfATM_QoS_t   QoS;        /* QoS profile */
} NPF_IfVcc_t;

/*
 *    ATM UNI Per-Vcc Statistics, returned in asynchronous
 *    response from NPF_IfVccStatsGet().
 */
typedef struct {
      NPF_VccAddr_t     vccaddr;    /* VPI/VCI to which stats apply */
      NPF_uint64_t      bytesRx;    /* Received Bytes */
      NPF_uint64_t      cellsRx;    /* Received Cells */
      NPF_uint64_t      framesRx;   /* Received Frames */
      NPF_uint64_t      bytesTx;    /* Transmitted Bytes */
      NPF_uint64_t      cellsTx;    /* Transmitted Cells */
      NPF_uint64_t      framesTx;   /* Transmitted Frames */
} NPF_ATM_VccStats_t;

/*
 *    Packet over SONET (POS) Interface Attributes, used in the
 *    union within NPF_IfGeneric_t.
 */
typedef struct {
      NPF_uint32_t      port;       /* Port number */
} NPF_IfPOS_t;

/*
 *    The Interface structure, passed to NPF_IfAttrSet()
 *    and NPF_IfCreateAndSet().
 */
typedef struct  {
      NPF_IfType_t      type;             /* Logical interface type */
      NPF_uint64_t      speed;            /* Speed in Kbits/second */
      NPF_IfAdminStatus_t adminStatus;    /* Administrative up/down */
      union {           /* Type specific attributes (by if_type code) */
        NPF_IfIPv4_t    IPv4_Attr;        /* IPv4 Interface attributes */
        NPF_IfLAN_t     LAN_Attr;         /* LAN interface attributes */
        NPF_IfATM_t     ATM_Attr;         /* ATM UNI interface attributes */
        NPF_IfPOS_t     POS_Attr;         /* POS interface attributes */
      } u;
} NPF_IfGeneric_t;


/*
 *    Completion Callback Types, to be found in the callback
 *    data structure, NPF_IfCallbackData_t.
 */
typedef enum NPF_IfCallbackType {
      NPF_IF_CREATE = 1,
      NPF_IF_DELETE = 2,
      NPF_IF_BIND = 3,
      NPF_IF_STATS_GET = 4,
      NPF_IF_VCC_STATS_GET = 5,
      NPF_IF_ATTR_SET = 6,
      NPF_IF_CREATE_AND_SET = 7,
      NPF_IF_ENABLE = 8,
      NPF_IF_DISABLE = 9,
      NPF_IF_OPER_STATUS_GET = 10,
      NPF_IF_LAN_SRC_ADDR_GET = 11,
      NPF_IF_LAN_SRC_ADDR_SET = 12,
      NPF_IF_LAN_ADDR_LIST_SET = 13,
      NPF_IF_LAN_ADDR_LIST_ADD = 14,
      NPF_IF_LAN_PROMISC_SET = 15,
      NPF_IF_LAN_PROMISC_CLEAR = 16,
      NPF_IF_LAN_FULL_DUPLEX_SET = 17,
      NPF_IF_LAN_FULL_DUPLEX_CLEAR = 18,
      NPF_IF_LAN_SPEED_SET = 19,
      NPF_IF_LAN_FLOW_CONTROL_TX_ENABLE = 20,
      NPF_IF_LAN_FLOW_CONTROL_TX_DISABLE = 21,
      NPF_IF_LAN_FLOW_CONTROL_RX_ENABLE = 22,
      NPF_IF_LAN_FLOW_CONTROL_RX_DISABLE = 23,
      NPF_IF_IPV4ADDR_SET = 24,
      NPF_IF_IPV4MTU_SET = 25,
      NPF_IF_IPV4FIB_SET = 26,
      NPF_IF_ATM_VCC_SET = 27,
      NPF_IF_ATM_VCC_BIND = 28,
      NPF_IF_ATM_VCC_DELETE = 29
} NPF_IfCallbackType_t;

/*
 *    Asynchronous error codes (returned in function callbacks)
 */

/* Callback/event reg. error */
#define NPF_IF_E_ALREADY_REGISTERED ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR)

/* Callback/event handle invalid */
#define NPF_IF_E_BAD_CALLBACK_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+1)

/* Callback function is NULL */
#define NPF_IF_E_BAD_CALLBACK_FUNCTION ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+2)

/* Invalid parameter */
#define NPF_IF_E_INVALID_PARAM ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+3)

/* Invalid child i/f handle */
#define NPF_IF_E_INVALID_CHILD_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+4)

/* Invalid parent i/f handle */
#define NPF_IF_E_INVALID_PARENT_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+5)

/* Invalid interface handle */
#define NPF_IF_E_INVALID_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+6)

/* Invalid VPI/VCI address */
#define NPF_IF_E_NO_SUCH_VCC ((NPF_IfErrorType_t) NPF_INTERFACES_BASE_ERR+7)

/* Invalid IP address */
#define NPF_IF_E_INVALID_IPADDR ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+8)

/* Invalid IP net prefix length */
#define NPF_IF_E_INVALID_NETPLEN ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+9)

/* Invalid IP MTU specification */
#define NPF_IF_E_INVALID_MTU ((NPF_IfErrorType_t) NPF_INTERFACES_BASE_ERR+10)

/* Invalid interface attribute */
#define NPF_IF_E_INVALID_ATTRIBUTE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+11)
```

```
/* Error – interface not created */
#define NPF_IF_E_NOT_CREATED ((NPF_IfErrorType_t) NPF_INTERFACES_BASE_ERR+12)

/* Invalid MAC address */
#define NPF_IF_E_INVALID_MAC_ADDR ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+13)

/* Invalid FIB handle */
#define NPF_IF_E_INVALID_FIB_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+14)

/* Invalid ATM UNI i/f handle */
#define NPF_IF_E_INVALID_ATM_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+15)

/* Invalid layer 3 i/f handle */
#define NPF_IF_E_INVALID_L3_HANDLE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+16)

/* Array length <= 0 or too big */
#define NPF_IF_E_BAD_ARRAY_LENGTH ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+17)

/* Interface has no source addr. */
#define NPF_IF_E_NO_SRC_ADDRESS ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+18)

/* Invalid generic interface speed */
#define NPF_IF_E_INVALID_SPEED ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+19)

/* Invalid VPI/VCI */
#define NPF_IF_E_INVALID_VCC_ADDRESS ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+20)

/* Invalid Interface Type */
#define NPF_IF_E_INVALID_IF_TYPE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+21)

/* Invalid Administrative Status code */
#define NPF_IF_E_INVALID_ADMIN_STATUS ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+22)

/* Invalid Port number */
#define NPF_IF_E_INVALID_PORT_NUMBER ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+22)

/* Invalid Promiscuous Mode code */
#define NPF_IF_E_INVALID_PROMISC_MODE ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+23)

/* Invalid LAN speed code */
#define NPF_IF_E_INVALID_LAN_SPEED ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+24)

/* Invalid ATM AAL code */
```

```
#define NPF_IF_E_INVALID_ATM_AAL ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+25)

/* Invalid ATM QoS specification */
#define NPF_IF_E_INVALID_ATM_QOS ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+26)

typedef NPF_uint32_t NPF_IfErrorType_t;

/*
 *    An asynchronous response contains an interface handle,
 *    a error or success code, and in some cases a function-
 *    specific structure embedded in a union.  One or more of
 *    these is passed to the callback function as an array
 *    within the NPF_IfCallbackData_t structure (below).
 */
typedef struct    {           /* Asynchronous Response Structure */
      NPF_IfHandle_t    ifHandle;   /* Interface handle for this response */
      NPF_IfErrorType_t error;      /* Error code for this response */
      union {                 /* Function-specific structures: */
        NPF_uint32_t          unused;     /* Default */
        NPF_uint32_t          arrayIndex; /* NPF_IfCreateAndSet index */
        NPF_IfStatistics_t   *ifStats;    /* NPF_IfGenericStatsGet() */
        NPF_IfOperStatus_t    operStat;   /* NPF_IfOperStatusGet() */
        NPF_ATM_VccStats_t   *vccStats;   /* NPF_IfVccStatsGet() */
        NPF_IfHandle_t        child;      /* NPF_IfBind(), handle=parent*/
        NPF_MAC_Address_t     MACaddr;    /* NPF_IfLAN_SrcAddrGet() */
        NPF_VccAddr_t         VccAddr;    /* NPF_IfATM_VccSet(), */
                                          /* NPF_IfATM_VccBind(), and */
                                          /* NPF_IfATM_VccDelete() */
      }     u;
} NPF_IfAsyncResponse_t;

/*
*     The callback function receives the following structure containing
 *    one or more asynchronous responses from a single function call.
 *    There are several possibilities:
 *    1. The called function does a single request
 *       - n_resp = 1, and the resp array has just one element.
 *       - allOK = TRUE if the request completed without error
 *         and the only return value is the response code.
 *       - if allOK = FALSE, the "resp" structure has the error code.
 *    2. the called function supports an array of requests
 *       a. All completed successfully, at the same time, and the
 *          only returned value is the response code:
 *          - allOK = TRUE, n_resp = 0.
 *       b. Some completed, but not all
 *          - allOK = FALSE, n_resp = the number completed
 *          - the "resp" array will contain one element for
 *            each completed request, with the error code
 *            in the NPF_IfAsyncResponse_t structure, along
 *            with any other information needed to identify
 *            which request element the response belongs to.
 *          - Callback function invocations are repeated in
 *            this fashion until all requests are complete.
 *          Responses are not repeated for request elements
 *          already indicated as complete in earlier callback
 *          function invocations.
 */
typedef struct {
      NPF_IfCallbackType_t    type;       /* Which function was called? */
      NPF_boolean_t           allOK;      /* TRUE if all completed OK */
      NPF_uint32_t            n_resp;     /* Number of responses in array */
      NPF_IfAsyncResponse_t   *resp;      /* Pointer to response structures*/
}     NPF_IfCallbackData_t;

/*
 *    Event types
 */
typedef enum {
      NPF_IF_UP = 1,               /* Interface went oper UP */
      NPF_IF_DOWN = 2,             /* Interface went oper DOWN */
      NPF_IF_COUNTER_DISCONTINUITY = 3 /* Counter discontinuity occurred*/
} NPF_IfEvent_t;

/*
 *    Event notification structure and array
 */
typedef struct NPF_IfEventData      {
      NPF_IfEvent_t eventType;      /* Event type */
      NPF_IfHandle_t handle;        /* Interface handle */
} NPF_IfEventData_t;

typedef struct    {
      NPF_uint16_t       n_data;          /* Number of events in array */
      NPF_IfEventData_t *eventData; /* Array of event notifications */
} NPF_IfEventArray_t;

typedef     NPF_uint32_t NPF_IfEventHandlerHandle_t;

/*
 * Completion Callback Function
 */
typedef void (*NPF_IfCallbackFunc_t) (
            NPF_IN NPF_userContext_t      userContext,
            NPF_IN NPF_correlator_t       correlator,
            NPF_IN NPF_IfCallbackData_t  *ifCallbackData);

/*
 * Completion Callback Registration Function
 */
NPF_error_t NPF_IfRegister(
            NPF_IN NPF_userContext_t userContext,
            NPF_IN NPF_IfCallbackFunc_t   ifCallbackFunc,
            NPF_OUT NPF_callbackHandle_t *CallbackHandle);

/*
 * Completion Callback Deregistration Function
 */
NPF_error_t NPF_IfDeregister(
            NPF_IN NPF_callbackHandle_t CallbackHandle);

/*
```

```
 * Event Handler Function
 */
typedef void (*NPF_IfEventHandlerFunc_t) (
            NPF_IN NPF_userContext_t  userContext,
            NPF_IN NPF_IfEventArray_t ifEventArray);

/*
 * Event Handler Registration Function
 */
NPF_error_t NPF_IfEventRegister(
            NPF_IN NPF_userContext_t            userContext,
            NPF_IN NPF_IfEventHandlerFunc_t     ifEventHandlerFunc,
            NPF_OUT NPF_IfEventHandlerHandle_t *ifEventHandlerHandle);

/*
 * Event Handler Deregistration Function
 */
NPF_error_t NPF_IfEventDeregister(
            NPF_IN NPF_IfEventHandlerHandle_t ifEventHandlerHandle);

/*
 *    Function to Create one or more Interfaces
 */
NPF_error_t  NPF_IfCreate(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_if,
            NPF_IN NPF_IfType_t           if_Type);

/*
 *    Function to Delete one or more Interfaces
 */
NPF_error_t  NPF_IfDelete(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to Bind one or more pairs of Interfaces
 */
NPF_error_t  NPF_IfBind(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           if_nbinds,
            NPF_IN NPF_IfBinding_t       *if_bindArray);

/*
 *    Function to Read Statistics from one or more Interfaces
 */
NPF_error_t  NPF_IfGenericStatsGet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to Read Statistics from one or
 *    more Vccs on a single ATM UNI interface
 */
NPF_error_t  NPF_IfVccStatsGet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_Vccs,
            NPF_IN NPF_VccAddr_t         *if_VccAddrArray);

/*
 *    Function to Set all Interface Attributes for one
 *    or more interfaces
 */
NPF_error_t  NPF_IfAttrSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray,
            NPF_IN NPF_IfGeneric_t       *if_StructArray);

/*
 *    Function to create one or more interfaces and set all of
 *    their attributes
 */
NPF_error_t  NPF_IfCreateAndSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_if,
            NPF_IN NPF_IfGeneric_t       *if_StructArray);

/*
 *    Function to Enable one or more Interfaces
 */
NPF_error_t  NPF_IfEnable(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to Disable one or more Interfaces
 */
NPF_error_t  NPF_IfDisable(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);
```

```
/*
 *    Function to Return the Operational Status of one
 *    or more Interfaces
 */
NPF_error_t  NPF_IfOperStatusGet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to Return the Source MAC Address of one
 *    or more LAN Interfaces
 */
NPF_error_t  NPF_IfLAN_SrcAddrGet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to Set the Source MAC Address of one
 *    or more LAN Interfaces
 */
NPF_error_t  NPF_IfLAN_SrcAddrSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray,
            NPF_IN NPF_MAC_Address_t     *if_LAN_SrcAddrArray);

/*
 *    Function to Set a  List of Receive MAC Addresses for
 *    one or more LAN Interfaces
 */
NPF_error_t  NPF_IfLAN_MAC_RcvAddrListSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray,
            NPF_IN NPF_uint32_t           if_n_MAC_Addrs,
            NPF_IN NPF_MAC_Address_t     *if_LAN_AddrArray);

/*
 *    Function to Add to the List of Receive MAC Addresses
 *    for one or more LAN Interfaces
 */
NPF_error_t  NPF_IfLAN_MAC_RcvAddrListAdd(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray,
            NPF_IN NPF_uint32_t           if_n_MAC_Addrs,
            NPF_IN NPF_MAC_Address_t     *if_LAN_AddrArray);

/*
 *    Function to Set one or more LAN Interfaces
 *    in Promiscuous Mode
 */
NPF_error_t  NPF_IfLAN_PromiscSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to Turn off Promiscuous Mode in one or more
 *    LAN Interfaces
 */
NPF_error_t  NPF_IfLAN_PromiscClear(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to set full duplex mode on one or more
 *    10/100 and gigabit Ethernet LAN Interfaces
 */
NPF_error_t NPF_IfLAN_FullDuplexSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to disable full duplex mode on one or more
 *    10/100 and gigabit Ethernet LAN Interfaces
 */
NPF_error_t NPF_IfLAN_FullDuplexClear(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to set interface speed or autosense
 *    on one or more 10/100 Ethernet LAN Interfaces
 */
NPF_error_t NPF_IfLAN_SpeedSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray,
            NPF_IN NPF_IfLAN_Speed_t     *speedArray);

/*
 *    Function to enable transmitting Flow Control messages on
 *    one or more 10/100 and gigabit Ethernet LAN Interfaces
 */
NPF_error_t NPF_IfLAN_FlowControlTxEnable(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to disable transmitting Flow Control messages
 *    on one or more 10/100 and gigabit Ethernet LAN Interfaces
 */
NPF_error_t NPF_IfLAN_FlowControlTxDisable(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to enable responding to Flow Control messages
 *    on one or more 10/100 and gigabit Ethernet LAN Interfaces
 */
NPF_error_t NPF_IfLAN_FlowControlRxEnable(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to disable responding to Flow Control messages
 *    on one or more 10/100 and gigabit Ethernet LAN Interfaces
 */
NPF_error_t NPF_IfLAN_FlowControlRxDisable(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray);

/*
 *    Function to set the primary IP address and prefix length
 *    on one or more IPv4 Interfaces
 */
NPF_error_t  NPF_IfIPv4AddrSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray,
            NPF_IN NPF_IPv4NetAddr_t     *if_IPv4AddrArray);

/*
 *    Function to set the MTU on one or more IPv4 interfaces
 */
NPF_error_t  NPF_IfIPv4MTUSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_uint32_t           n_handles,
            NPF_IN NPF_IfHandle_t        *if_HandleArray,
            NPF_IN NPF_uint16_t          *if_IPv4MTUArray);

/*
 *    Function to Associate a FIB to one or more IPv4 Interfaces
 */
NPF_error_t  NPF_IfIPv4FIBSet(
            NPF_IN NPF_callbackHandle_t               if_cbHandle,
            NPF_IN NPF_correlator_t                   if_cbCorrelator,
            NPF_IN NPF_errorReporting_t               if_errorReporting,
            NPF_IN NPF_uint32_t                       n_handles,
            NPF_IN NPF_IfHandle_t                    *if_HandleArray,
            NPF_IN NPF_IPv4UnicastPrefixTableHandle_t if_FIB_Handle);

/*
 *    Function to Add or Modify one or more Vccs on
 *    a single ATM UNI interface
 */
NPF_error_t  NPF_IfATM_VccSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_Vccs,
            NPF_IN NPF_IfVcc_t           *if_VccStructArray);

/*
 *    Function to bind a higher layer interface to one
 *    or more Vccs on a single ATM UNI interface
 */
NPF_error_t  NPF_IfATM_VccBind(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_ATM_Handle,
            NPF_IN NPF_IfHandle_t         if_ParentHandle,
            NPF_IN NPF_uint32_t           n_Vccs,
            NPF_IN NPF_VccAddr_t         *if_VccAddrArray);

/*
 *    Function to delete one or more Vccs on a single
 *    ATM UNI interface
 */
NPF_error_t  NPF_IfATM_VccDelete(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_Vccs,
            NPF_IN NPF_VccAddr_t         *if_VccAddrArray);

#ifdef __cplusplus
}
#endif

#endif /* __NPF_IF_ */
```