



# Interface Management API Implementation Agreement

Revision 2.0

**Editor:** John Renwick, Agere Systems, [jrenwick@agere.com](mailto:jrenwick@agere.com)

Copyright © 2003 The Network Processing Forum (NPF). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction other than the following, (1) the above copyright notice and this paragraph must be included on all such copies and derivative works, and (2) this document itself may not be modified in any way, such as by removing the copyright notice or references to the NPF, except as needed for the purpose of developing NPF Implementation Agreements.

By downloading, copying, or using this document in any manner, the user consents to the terms and conditions of this notice. Unless the terms and conditions of this notice are breached by the user, the limited permissions granted above are perpetual and will not be revoked by the NPF or its successors or assigns.

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN "AS IS" BASIS WITHOUT ANY WARRANTY OF ANY KIND. THE INFORMATION, CONCLUSIONS AND OPINIONS CONTAINED IN THE DOCUMENT ARE THOSE OF THE AUTHORS, AND NOT THOSE OF NPF. THE NPF DOES NOT WARRANT THE INFORMATION IN THIS DOCUMENT IS ACCURATE OR CORRECT. THE NPF DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED THE IMPLIED LIMITED WARRANTIES OF MERCHANTABILITY, TITLE OR FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS.

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.

For additional information contact:  
The Network Processing Forum, 39355 California Street,  
Suite 307, Fremont, CA 94538  
+1 510 608-5990 phone [info@npforum.org](mailto:info@npforum.org)

# Table of Contents

|        |  |    |
|--------|--|----|
| 1      | Revision History .....   | 5  |
| 2      | Introduction.....  | 6  |
| 2.1    | ASSUMPTIONS AND EXTERNAL REQUIREMENTS.....   | 7  |
| 2.2    | SCOPE.....   | 8  |
| 2.3    | DEPENDENCIES.....  | 8  |
| 2.4    | INTERFACE MANAGEMENT STRUCTURES.....   | 8  |
| 2.4.1  | Common Interface Attributes.....   | 8  |
| 2.4.2  | Common Interface Statistics.....   | 8  |
| 2.4.3  | IPv4 Interface .....   | 9  |
| 2.4.4  | Generic LAN Interface .....  | 9  |
| 2.4.5  | ATM UNI Interface .....  | 9  |
| 2.4.6  | Packet Over Sonet (POS) Interface.....   | 10 |
| 2.4.7  | IPv6 Interface .....   | 10 |
| 2.4.8  | IPv6inV4 Interface.....  | 10 |
| 2.4.9  | IP-in-IP Tunnel Interfaces.....  | 10 |
| 2.4.10 | Interface Relatedness.....   | 11 |
| 2.4.11 | Interface Manager Application.....   | 11 |
| 3      | Data Types .....   | 12 |
| 3.1    | INTERFACE MANAGEMENT API TYPES.....  | 12 |
| 3.1.1  | Interface Identifier .....   | 12 |
| 3.1.2  | Generic Interface Structure: <i>NPF_IfGeneric_t</i> .....  | 12 |
| 3.1.3  | Interface Handle: <i>NPF_IfHandle_t</i> .....  | 12 |
| 3.1.4  | Interface Type Code: <i>NPF_IfType_t</i> .....   | 13 |
| 3.1.5  | Structure to Relate Two Interfaces: <i>NPF_IfBinding_t</i> .....                                 | 13 |
| 3.1.6  | Interface Statistics: <i>NPF_IfStatistics_t</i> .....  | 13 |
| 3.1.7  | Operational Status Code: <i>NPF_IfOperStatus_t</i> .....   | 13 |
| 3.1.8  | Administrative Status Code: <i>NPF_IfAdminStatus_t</i> .....                                     | 14 |
| 3.1.9  | IP Forwarding Mode : <i>NPF_IfIpFwdMode_t</i> .....  | 14 |
| 3.1.10 | IPv4 Interface Attributes: <i>NPF_IfIPv4_t</i> .....   | 14 |
| 3.1.11 | IPv4 Address Prefix: <i>NPF_IPv4Prefix_t</i> .....   | 14 |
| 3.1.12 | IPv6 Address Prefix: <i>NPF_IPv6Prefix_t</i> .....   | 14 |
| 3.1.13 | IPv6 Interface Attributes: <i>NPF_IfIPv6_t</i> .....   | 15 |
| 3.1.14 | IPv6 in IPv4 Tunnel Attributes: <i>NPF_IfIPv6inV4_t</i> .....                                    | 15 |
| 3.1.15 | LAN Speed Codes: <i>NPF_IfLAN_Speed_t</i> .....  | 15 |
| 3.1.16 | Generic LAN Interface Attributes: <i>NPF_IfLAN_t</i> .....                                       | 16 |
| 3.1.17 | ATM UNI Interface Attributes: <i>NPF_IfATM_t</i> .....   | 16 |
| 3.1.18 | ATM UNI Vcc Attributes: <i>NPF_IfVcc_t</i> .....   | 16 |
| 3.1.19 | ATM UNI VPI/VCI Structures: <i>NPF_VccAddr_t</i> .....   | 16 |
| 3.1.20 | ATM UNI Vcc AAL Type Code: <i>NPF_IfAAL_t</i> .....  | 17 |
| 3.1.21 | ATM UNI QoS Profile: <i>NPF_IfATM_QoS_t</i> .....  | 17 |
| 3.1.22 | ATM UNI Per-VCC Statistics: <i>NPF_IfATM_VccStats_t</i> .....                                    | 17 |
| 3.1.23 | Packet Over SONET (POS) Interface Attributes: <i>NPF_IfPOS_t</i> .....                           | 17 |
| 3.1.24 | IPv4-in-IPv4 Tunnel Interface Attributes: <i>NPF_IfTunnelIPv4_t</i> .....                        | 18 |
| 3.1.25 | IPv6-in-IPv6 Tunnel Interface Attributes: <i>NPF_IfTunnelIPv6_t</i> .....                        | 18 |
| 3.1.26 | Tunnel Address Structures: <i>NPF_IfTunnelIPv4Addr_t</i> and <i>NPF_IfTunnelIPv6Addr_t</i> ..... | 18 |
| 3.2    | DATA STRUCTURES FOR COMPLETION CALLBACKS.....  | 18 |
| 3.2.1  | Completion Callback Type Codes: <i>NPF_IfCallbackType_t</i> .....                                | 18 |
| 3.2.2  | Asynchronous Response Array Element: <i>NPF_IfAsyncResponse_t</i> .....                          | 20 |
| 3.2.3  | Callback Data Structure: <i>NPF_IfCallbackData_t</i> .....                                       | 21 |
| 3.3    | ERROR CODES .....  | 22 |
| 3.4    | DATA STRUCTURES FOR EVENT NOTIFICATIONS.....   | 25 |
| 3.4.1  | Event Types: <i>NPF_IfEvent_t</i> .....  | 25 |

|        |  |    |
|--------|--|----|
| 3.4.2  | <i>Event Notification Structure and Array: NPF_IfEventData_t and NPF_IfEventArray_t</i> .....                    | 25 |
| 4      | <b>Functions</b> .....   | 26 |
| 4.1    | COMPLETION CALLBACK .....  | 26 |
| 4.1.1  | <i>Completion Callback Function</i> .....  | 26 |
| 4.1.2  | <i>NPF_IfRegister: Completion Callback Registration Function</i> .....   | 27 |
| 4.1.3  | <i>NPF_IfDeregister: Completion Callback Deregistration Function</i> .....                                       | 27 |
| 4.2    | EVENT NOTIFICATION .....   | 28 |
| 4.2.1  | <i>Event Handler Function</i> .....  | 28 |
| 4.2.2  | <i>NPF_IfEventRegister: Event Handler Registration Function</i> .....  | 28 |
| 4.2.3  | <i>NPF_IfEventDeregister: Event Handler Deregistration Function</i> .....  | 29 |
| 4.3    | EVENT DEFINITION SIGNATURE .....   | 30 |
| 4.4    | ORDER OF OPERATIONS.....   | 30 |
| 4.5    | COMPLETION CALLBACKS AND ERROR RETURNS.....  | 30 |
| 4.6    | INTERFACE MANAGEMENT API .....   | 30 |
| 4.6.1  | <i>NPF_IfCreate: Create an Interface</i> .....   | 30 |
| 4.6.2  | <i>NPF_IfDelete: Delete an Interface</i> .....   | 31 |
| 4.6.3  | <i>NPF_IfBind: Bind Interfaces</i> .....   | 32 |
| 4.6.4  | <i>NPF_IfUnBind: Remove Interface Bindings</i> .....   | 33 |
| 4.6.5  | <i>NPF_IfGenericStatsGet: Read Interface Statistics</i> .....  | 34 |
| 4.6.6  | <i>NPF_IfVccStatsGet: Read ATM UNI Vcc Statistics</i> .....  | 34 |
| 4.6.7  | <i>NPF_IfAttrSet: Set All Interface Attributes</i> .....   | 35 |
| 4.6.8  | <i>NPF_IfCreateAndSet: Create an Interface and Set All of its Attributes</i> .....                               | 36 |
| 4.6.9  | <i>NPF_IfEnable: Enable an Interface</i> .....   | 38 |
| 4.6.10 | <i>NPF_IfDisable: Disable an Interface</i> .....   | 38 |
| 4.6.11 | <i>NPF_IfOperStatusGet: Return the Operational Status of an Interface</i> .....                                  | 39 |
| 4.6.12 | <i>NPF_IfMTU_Set: Set an Interface's MTU</i> .....   | 40 |
| 4.6.13 | <i>NPF_IfAttrGet: Read Interface Attributes</i> .....  | 41 |
| 4.6.14 | <i>NPF_IfLAN_SrcAddrGet: Return a LAN Interface's Source MAC Address</i> .....                                   | 41 |
| 4.6.15 | <i>NPF_IfLAN_SrcAddrSet: Set a LAN Interface's Source MAC Address</i> .....                                      | 42 |
| 4.6.16 | <i>NPF_IfLAN_MAC_RcvAddrListSet: Set a LAN Interface's List of Receive MAC Addresses</i> .....                   | 43 |
| 4.6.17 | <i>NPF_IfLAN_MAC_RcvAddrListAdd: Add to a LAN Interface's List of Receive MAC Addresses</i> ..                   | 44 |
| 4.6.18 | <i>NPF_IfLAN_PromiscSet: Set a LAN Interface in Promiscuous Mode</i> .....                                       | 45 |
| 4.6.19 | <i>NPF_IfLAN_PromiscClear: Turn off Promiscuous Mode in a LAN Interface</i> .....                                | 45 |
| 4.6.20 | <i>NPF_IfLAN_FullDuplexSet: Set Full Duplex Mode on a LAN Interface</i> .....                                    | 46 |
| 4.6.21 | <i>NPF_IfLAN_FullDuplexClear: Disable Full Duplex Mode on a LAN Interface</i> .....                              | 47 |
| 4.6.22 | <i>NPF_IfLAN_SpeedSet: Set Interface Speed or Autosense on a LAN Interface</i> .....                             | 47 |
| 4.6.23 | <i>NPF_IfLAN_FlowControlTxEnable: Enable Sending Flow Control on a LAN Interface</i> .....                       | 48 |
| 4.6.24 | <i>NPF_IfLAN_FlowControlTxDisable: Disable Sending Flow Control on a LAN Interface</i> .....                     | 49 |
| 4.6.25 | <i>NPF_IfLAN_FlowControlRxEnable: Enable Receiving Flow Control on a LAN Interface</i> .....                     | 50 |
| 4.6.26 | <i>NPF_IfLAN_FlowControlRxDisable: Disable Receiving Flow Control on a LAN Interface</i> .....                   | 50 |
| 4.6.27 | <i>NPF_IfIP_FwdEnable: Enable IP Forwarding on One or More IPv4 or IPv6 Interfaces</i> .....                     | 51 |
| 4.6.28 | <i>NPF_IfIP_FwdDisable: Disable IP Forwarding on one or more IP Interfaces</i> .....                             | 52 |
| 4.6.29 | <i>NPF_IfIPv4AddrSet: Set an IPv4 Interface's IP Address</i> .....   | 52 |
| 4.6.30 | <i>NPF_IfIPv4AddrClear: Clear an IPv4 Interface's Primary IP Address</i> .....                                   | 53 |
| 4.6.31 | <i>NPF_IfIPv4MTUSet: Set an IPv4 Interface's MTU</i> .....   | 54 |
| 4.6.32 | <i>NPF_IfIPv4FIBSet: Associate an IPv4 FIB to an Interface</i> .....   | 55 |
| 4.6.33 | <i>NPF_IfIPv4UC_AddrSet: Set an IPv4 Interface's Secondary List of IP Prefixes</i> .....                         | 55 |
| 4.6.34 | <i>NPF_IfIPv4UC_AddrAdd: Add to an IPv4 Interface's Secondary List of IP Prefixes</i> .....                      | 56 |
| 4.6.35 | <i>NPF_IfIPv4UC_AddrDelete: Delete Prefixes From an IPv4 Interface's Secondary List of IP<br/>Prefixes</i> ..... | 57 |
| 4.6.36 | <i>NPF_IfIPv4McastAddrSet: Set an IPv4 Interface's List of Receive Multicast IP Addresses</i> .....              | 58 |
| 4.6.37 | <i>NPF_IfIPv4McastAddrAdd: Add to an IPv4 Interface's List of Receive Multicast IP Addresses</i> ..              | 59 |
| 4.6.38 | <i>NPF_IfIPv6AddrSet: Set the Addresses for an IPv6 Interface</i> .....  | 59 |
| 4.6.39 | <i>NPF_IfIPv6AddrAdd: Add Addresses to an IPv6 Interface</i> .....   | 60 |
| 4.6.40 | <i>NPF_IfIPv6AddrDelete: Delete Addresses From an IPv6 Interface's List of Unicast IP Addresses</i> .....        | 61 |

|            |   |    |
|------------|---|----|
| 4.6.41     | <i>NPF&gt;IfIPv6FIB_Set: Associate an IPv6 FIB with an Interface</i> .....                    | 62 |
| 4.6.42     | <i>NPF&gt;IfATM_VccSet: Add or Modify an ATM UNI Vcc</i> .....                                | 63 |
| 4.6.43     | <i>NPF&gt;IfATM_VccBind: Bind a Higher Layer Interface to an ATM Vcc</i> .....                | 64 |
| 4.6.44     | <i>NPF&gt;IfATM_VccUnBind: Remove Binding of a Higher Layer Interface to an ATM Vcc</i> ..... | 64 |
| 4.6.45     | <i>NPF&gt;IfATM_VccDelete: Delete an ATM Vcc</i> .....  | 65 |
| 4.6.46     | <i>NPF&gt;IfTunnelHopsSet: Set IP-in-IP Tunnel Length (Hops)</i> .....                        | 66 |
| 4.6.47     | <i>NPF&gt;IfTunnelDSCP_Set Set IP Tunnel DSCP</i> .....                                       | 67 |
| 4.6.48     | <i>NPF&gt;IfTunnelIPv4AddrSet: Set Tunnel's IPv4 Addresses</i> .....                          | 68 |
| 4.6.49     | <i>NPF&gt;IfTunnelIPv6AddrSet: Set IPv6-in-IPv6 Tunnel Addresses</i> .....                    | 68 |
| 4.6.50     | <i>NPF&gt;IfTunnelIPv6FlowLabelSet: Set IPv6-in-IPv6 Tunnel Flow Label</i> .....              | 69 |
| 5          | References .....  | 71 |
| 6          | API Capabilities .....  | 72 |
| 6.1        | OPTIONAL SUPPORT OF SPECIFIC TYPES .....  | 72 |
| 6.2        | API FUNCTIONS.....  | 72 |
| 6.3        | API EVENTS.....   | 73 |
| Appendix A | Header File: npf_if.h.....  | 74 |
| Appendix B | List of companies belonging to NPF DURING APPROVAL PROCESS .....                              | 93 |

## Table of Figures

|          |  |   |
|----------|--|---|
| Figure 1 | Layer 2 interface configurations.....            | 6 |
| Figure 2 | Layer 2 and Layer 3 Interface Relationship ..... | 7 |
| Figure 3 | L2-L3 mappings .....                             | 7 |
| Figure 4 | Link/L2/L3 relationship .....                    | 7 |

# 1 Revision History

| <b>Revision</b> | <b>Date</b> | <b>Reason for Changes</b>   |
|-----------------|-------------|---|
| 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.  |
| 2.0             | 12/05/2003  | Created Rev 2.0 of the implementation agreement by adding support for multiple unicast IP addresses and multicast addresses on IPv4 interfaces, adding support for tunnel interfaces, adding support for IPv6 interfaces, and IPv6in4 tunnels |

## 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 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 such as link speed, 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.

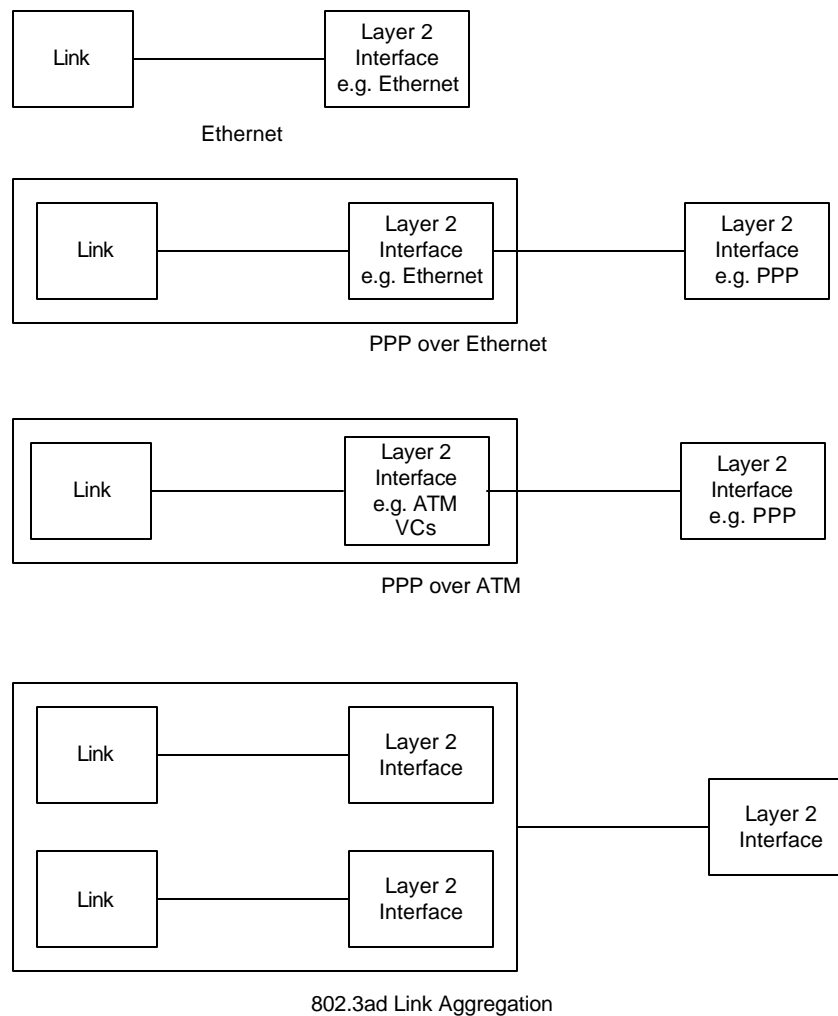


Figure 1 - Layer 2 interface configurations

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

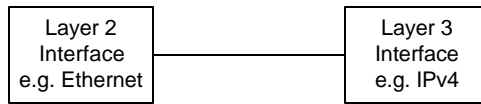


Figure 2 - Layer 2 and Layer 3 Interface Relationship

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 Layer 3 interface can be associated with multiple L2 interfaces.

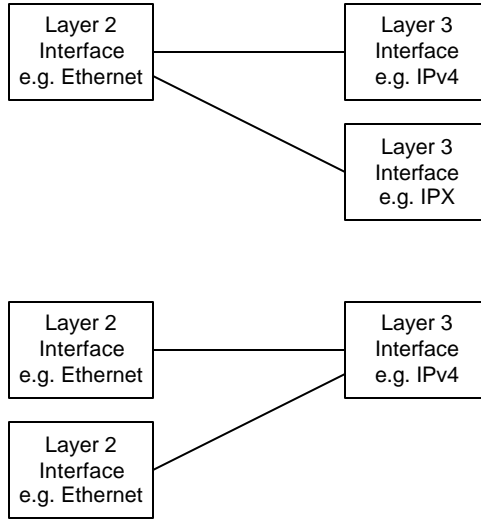


Figure 3 - L2-L3 mappings

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

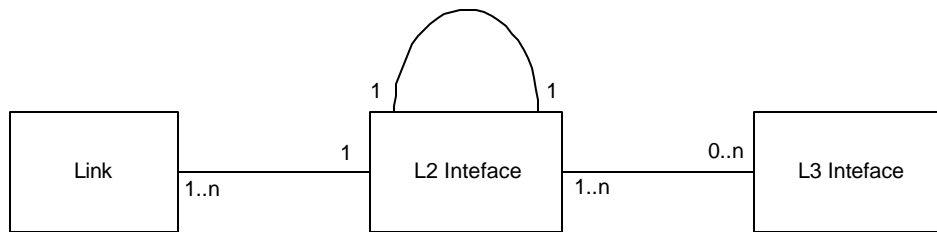


Figure 4 - Link/L2/L3 relationship

The Interface Management API provides a uniform interface for configuring and managing the physical and logical interfaces of which a network processor may need to be aware. For example, the API defined in this document will cover aspects of interface management related to Layer 2 (Bridging), Layer 3 (IP), 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.
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 2.4.11.)

## 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_IPv4UC_FwdTableHandle_t`.

The NP Forum IPv6 Unicast Forwarding API Implementation Agreement defines the Forwarding Information Base Handle, `NPF_IPv6UC_FwdTableHandle_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, such as statistics. See the individual structure descriptions below.

### 2.4.1 Common Interface Attributes

The common interface attribute structure contains the following, which 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.
- Interface Identifier: a nonzero integer value assigned by the application to each interface it creates. No two interfaces may have the same ID. This number can be anything of the application’s choosing; the value of `ifIndex` (see RFC 2863) is one possibility.

### 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”<sup>1</sup> 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)

---

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

- Output packets (multicast)
- Output packets (broadcast)
- Output packets dropped
- Output errors

### 2.4.3 IPv4 Interface

This interface 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:

- IP addresses and prefix lengths
- Receive IP multicast addresses
- MTU, or the largest IP datagram that can be sent on this interface
- FIB handle
- IPv4 forwarding mode

An IPv4 interface can have multiple IPv4 addresses on multiple IPv4 networks. The IPv4 interface attributes structure, `NPF_IFIPv4_t`, includes both a “Primary IP address” field and an array of IPv4 addresses and corresponding prefixes. The single address field existed in the first version of this Agreement, and the array of addresses was added later. When the array was added, the single primary address field was left in the structure for compatibility with existing applications. Together, these fields form a single set of IPv4 address/prefix pairs for the interface, but they are managed using different function calls. Arriving packets addressed to any of these addresses are considered locally-addressed packets.

### 2.4.4 Generic LAN Interface

This 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.
- Controls for half/full duplex, flow control.

### 2.4.5 ATM UNI Interface

This 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 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 IPv6 Interface

This interface type represents a Layer 3 interface for IPv6, essentially a binding of IPv6 attributes to a physical or logical port. Its application-settable attributes are:

- IPv6 addresses and their prefix length
- MTU, or the largest IP datagram that can be sent on this interface
- FIB handle
- Multicast receive addresses

### 2.4.8 IPv6in4 Interface

This interface type represents an IPv6 in IPv4 Tunnel. essentially a binding of tunnel attributes to a physical or logical port. Its application-settable attributes are:

- Type of IPv6in4 tunnel: Automatic, Configured or 6to4
- Destination IPv4 address in case of Configured Tunneling
- Tunnel source address

### 2.4.9 IP-in-IP Tunnel Interfaces

Tunneling is a method of routing IP datagrams by encapsulating them in a second IP header – the tunnel header. IP-in-IP tunnels are of two types, depending on the type of tunnel header used: IPv4-in-IPv4 tunnels, and IPv6-in-IPv6 tunnels. IP-in-IP tunnel interface attributes are:

- MTU (for sending)
- Tunnel hop count (for setting the TTL or Hop Limit when encapsulating for transmission)
- Source IP address
- Destination IP address
- DSCP value for the tunnel header (there are many ways to set the DSCP value depending on local policy, so this attribute might not be used)
- Flow label value for the tunnel header (IPv6 only)
- FIB handle; used for forwarding the encapsulated inner packet upon receive

The Source and Destination IP addresses are the addresses that are placed in the tunnel header when sending.

Tunneled packets arrive on an ordinary IPv4 or IPv6 interface with an IP address equal to the destination address in the tunnel header. That address also identifies the appropriate tunnel interface, whose FIB handle attribute can be used to identify the appropriate FIB for forwarding the encapsulated (inner) IP header.

### **2.4.10 Interface Relatedness**

The API includes a function (NPF>IfBind) that relates a pair of interfaces as parent and child. A parent interface represents a higher layer than that of its children, with reference to the OSI model. For example, layer 2 interfaces are lower in the hierarchy than (i.e. children of) layer 3 interfaces. An interface can be at the same time a parent of one interface and a child of another. The API places no restriction on:

- the number of levels of hierarchy
- the number 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.

### **2.4.11 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 4.4), 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

#### 3.1.1 Interface Identifier

The Interface Identifier is a nonzero integer value assigned by the application to each interface. No two interfaces may have the same Interface ID value. The Interface ID performs at least two functions: it aids in recovering from the event of a lost callback from an interface creation function, and it serves to identify the interface in callbacks. The Interface Management API implementation must remember the Interface ID value associated with each Interface Handle it creates. Any attempt by the application to create a new Interface Handle using an Interface ID value already associated with an existing handle must result in an error with the existing handle being returned to the application, and no new handle created. Callback information from functions that create, modify, destroy or query interfaces must always include both the Interface Handle and the Interface ID value for each interface referenced.

```
typedef NPF_uint32_t    NPF_IfID_t;        /* Interface Identifier */
```

#### 3.1.2 Generic Interface Structure: NPF\_IfGeneric\_t

```
/*
 *   The Interface structure:
 */
typedef struct {
    NPF_IfID_t        ifID;                /* Interface ID */
    NPF_IfType_t      type;                /* Logical interface type */
    NPF_uint64_t      speed;               /* Speed in Kbits/second */
    NPF_IfAdminStatus_t adminStatus;      /* Administrative up/down */
    NPF_uint32_t      nChildren;           /* Number of child interfaces */
    NPF_uint32_t      *childIDs;           /* Array of child interface IDs */
    NPF_uint32_t      nParents;            /* Number of parent interfaces */
    NPF_uint32_t      *parentIDs;          /* Array of parent i/f IDs */
    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 */
        NPF_IfIPv6_t      IPv6_Attr;       /* IPv6 interface attributes */
        NPF_IfIPv6inv4_t  IPv6inv4_Attr;   /* IPv6inv4 tunnel attributes */
        NPF_IfTunnelIPv4_t TunnelIPv4Attr; /* IP-in-IPv4 tunnel attributes */
        NPF_IfTunnelIPv6_t TunnelIPv6Attr; /* IP-in-IPv6 tunnel attributes */
    } u;
} NPF_IfGeneric_t;
```

The “speed” variable is read-only, or, if set by the application, might not affect the speed of any device.

#### 3.1.3 Interface Handle: NPF\_IfHandle\_t

```
/*
 *   Interface handle
 */
typedef NPF_uint32_t    NPF_IfHandle_t;
```

### 3.1.4 Interface Type Code: NPF\_IfType\_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_IPV6=6,         /* IPv6 logical interface */
    NPF_IF_TYPE_IPV6INV4=7,     /* IPv6inv4 logical interface */
    NPF_IF_TYPE_TUNNEL_IPV4=8,  /* IP-in-IPV4 tunnel */
    NPF_IF_TYPE_TUNNEL_IPV6=9,  /* IP-in-IPV6 tunnel */
    NPF_IF_TYPE_HIGHEST=9      /* Highest defined value */
} NPF_IfType_t;
```

### 3.1.5 Structure to Relate Two Interfaces: NPF\_IfBinding\_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;
```

### 3.1.6 Interface Statistics: NPF\_IfStatistics\_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;
```

### 3.1.7 Operational Status Code: NPF\_IfOperStatus\_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;
```

### 3.1.8 Administrative Status Code: NPF\_IfAdminStatus\_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;

```

### 3.1.9 IP Forwarding Mode : NPF\_IfIpFwdMode\_t

```

/*
 *   IP Forwarding mode code
 */
typedef enum      {
    NPF_IF_IP_FORWARDING_ENABLE = 1,     /* Enable IP Forwarding */
    NPF_IF_IP_FORWARDING_DISABLE = 2    /* Disable IP Forwarding */
} NPF_IfIpFwdMode_t;

```

### 3.1.10 IPv4 Interface Attributes: NPF\_IfIPv4\_t

```

/*
 *   IPv4 Interface attributes
 */
typedef struct {
    NPF_IPv4Prefix_t  addr;               /* Primary IPv4 Address and plen */
    NPF_uint16_t      mtu;                /* IPv4 Max Transmission Unit */
    NPF_IPv4UC_FwdTableHandle_t  fibHandle; /* Forwarding Info Base */
    NPF_uint32_t      nUcAddrs;          /* Number of unicast IP addrs */
    NPF_IPv4Prefix_t *ucAddrs;           /* Array of unicast IP addrs */
    NPF_uint32_t      nMcAddrs;          /* Number of mcast IP addrs */
    NPF_uint32_t      *mcAddrs;          /* Array of multicast IP addrs */
    NPF_IfIpFwdMode_t fwdMode;           /* IPv4 Forwarding Mode */
} NPF_IfIPv4_t;

```

### 3.1.11 IPv4 Address Prefix: NPF\_IPv4Prefix\_t

```

/*
 *   IPv4 address prefix structure (Defined globally)
 */
typedef struct {
    NPF_IPv4Address_t IPv4Addr;          /* IPv4 address */
    NPF_uint8_t        IPv4Plen;         /* Prefix length in bits (1-32) */
} NPF_IPv4Prefix_t;

```

### 3.1.12 IPv6 Address Prefix: NPF\_IPv6Prefix\_t

```

/*
 *   IPv6 Address Flags
 */

typedef NPF_uint8_t      NPF_IPv6AddrFlags_t;

#define NPF_IF_IPV6_ADDR_FLAGS_NONE 0x00

```

```

#define NPF_IF_IPV6_ADDR_ANYCAST    0x01
#define NPF_IF_IPV6_ADDR_PREFERRED  0x02
#define NPF_IF_IPV6_ADDR_DEPRECATED 0x04

/*
 * IPv6 address prefix structure (Defined globally)
 */
typedef struct {
    NPF_IPv6Address_t IPv6Addr;          /* IPv6 address */
    NPF_uint8_t       IPv6Plen;         /* Prefix length in bits (1-128) */
    NPF_IPv6AddrFlags_t IPv6Flags;      /* Anycast, preferred, deprecated*/
} NPF_IPv6Prefix_t;

```

### 3.1.13 IPv6 Interface Attributes: NPF>IfIPv6\_t

```

/*
 * IPv6 Interface attributes
 */
typedef struct {
    NPF_uint32_t      n_addr;           /* Number of IPv6 Addresses */
    NPF_IPv6Prefix_t *addr;            /* IPv6 Addresses */
    NPF_uint16_t      mtu;              /* IPv6 Max Transmission Unit */
    NPF_IPv6UC_FwdTableHandle_t fibHandle; /* Forwarding Info Base*/
    NPF>IfIpFwdMode_t fwdMode;         /* IPv6 Forwarding Mode */
} NPF>IfIPv6_t;

```

### 3.1.14 IPv6 in IPv4 Tunnel Attributes: NPF>IfIPv6inv4\_t

```

/*
 * IPv6 in IPv4 Tunnel types
 */
typedef enum {
    NPF_IF_V6INV4_AUTOMATIC = 1,      /* Automatic Tunnel */
    NPF_IF_V6INV4_CONFIGURED = 2,     /* Configured Tunnel */
    NPF_IF_V6INV4_6TO4 = 3           /* 6to4 Tunnel*/
} NPF_IPv6inv4TunnelType_t;

/*
 * IPv6inv4 Tunnel Interface attributes
 */
typedef struct {
    NPF_IPv6inv4TunnelType_t tunnelType; /* Type of v6inv4 Tunnel */
    NPF_IPv4Address_t         IPv4DstAddr; /* Destination IPv4 address */
    NPF_IPv4Address_t         IPv4SrcAddr; /* Source IPv4 address */
    NPF_uint8_t               IPv4TTL;     /* Time to live in IPv4 header*/
    NPF_uint8_t               IPv4DSCP;    /* DiffServ Code Point in IPv4
header*/
} NPF>IfIPv6inv4_t;

```

### 3.1.15 LAN Speed Codes: NPF>IfLAN\_Speed\_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;

```

### 3.1.16 Generic LAN Interface Attributes: NPF>IfLAN\_t

```

/*
 *   LAN Control Flags
 */
#define NPF_IF_LAN_FDX_ENABLE          0x01 /* TRUE to enable full duplex */
#define NPF_IF_LAN_PROMISC            0x02 /* TRUE for promiscuous mode */
#define NPF_IF_LAN_TX_FC_ENABLE       0x04 /* TRUE for transmit flow ctrl */
#define NPF_IF_LAN_RX_FC_ENABLE       0x08 /* TRUE for receive flow ctrl */

/*
 *   Generic LAN Interface attributes
 */
typedef struct {
    NPF_uint32_t      port;          /* Port number */
    NPF>IfLAN_Speed_t speed;        /* Speed control */
    NPF_uint32_t      flags;         /* Flags (see above) */
    NPF_MAC_Address_t srcAddr;      /* Source MAC address */
    NPF_uint32_t      nAddrs;       /* Number of receive MAC addrs */
    NPF_MAC_Address_t *rcvAddrs;    /* Array of receive MAC addrs */
} NPF>IfLAN_t;

```

### 3.1.17 ATM UNI Interface Attributes: NPF>IfATM\_t

```

/*
 *   ATM UNI Interface Attributes
 */
typedef struct {
    NPF_uint32_t      port;          /* Port number */
    NPF_uint32_t      nVccs;        /* Number of active VCCs */
    NPF>IfVcc_t       **vccArray;   /* Pointer to array of pointers */
                                     /* to VCC Attributes structures */
} NPF>IfATM_t;

```

### 3.1.18 ATM UNI Vcc Attributes: NPF>IfVcc\_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;

```

### 3.1.19 ATM UNI VPI/VCI Structures: NPF\_VccAddr\_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;

```

### 3.1.20 ATM UNI Vcc AAL Type Code: NPF\_IfAAL\_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;

```

### 3.1.21 ATM UNI QoS Profile: NPF\_IfATM\_QoS\_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;

```

### 3.1.22 ATM UNI Per-VCC Statistics: NPF\_IfATM\_VccStats\_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;

```

### 3.1.23 Packet Over SONET (POS) Interface Attributes: NPF\_IfPOS\_t

```

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

```

```
} NPF_IfPOS_t;
```

### 3.1.24 IPv4-in-IPv4 Tunnel Interface Attributes: NPF\_IfTunnelIPv4\_t

```
/*
 *   IP-in-IPv4 Tunnel Interface Attributes
 */
typedef struct {
    NPF_uint16_t      MTU;                /* Tunnel max transmission unit */
    NPF_uint8_t       maxHops;           /* Maximum hops in the tunnel */
    NPF_IPv4Address_t dstAddr;          /* Tunnel destination IP addr */
    NPF_IPv4Address_t srcAddr;          /* Tunnel source IP addr */
    NPF_uint8_t       DSCP;              /* DiffServ Code Point for hdr */
    NPF_IPv4UC_FwdTableHandle_t fibHandle; /* FIB for forwarding inner hdr */
} NPF_IfTunnelIPv4_t;
```

### 3.1.25 IPv6-in-IPv6 Tunnel Interface Attributes: NPF\_IfTunnelIPv6\_t

```
/*
 *   IP-in-IPv6 Tunnel Interface Attributes
 */
typedef struct {
    NPF_uint16_t      MTU;                /* Tunnel max transmission unit */
    NPF_uint8_t       maxHops;           /* Maximum hops in the tunnel */
    NPF_IPv6Address_t dstAddr;          /* Tunnel destination IP addr */
    NPF_IPv6Address_t srcAddr;          /* Tunnel source IP addr */
    NPF_uint8_t       DSCP;              /* DiffServ Code Point for hdr */
    NPF_uint32_t      flowLabel;         /* IPv6 Flow Label for hdr */
    NPF_IPv6UC_FwdTableHandle_t fibHandle; /* FIB for forwarding inner hdr */
} NPF_IfTunnelIPv6_t;
```

### 3.1.26 Tunnel Address Structures: NPF\_IfTunnelIPv4Addr\_t and NPF\_IfTunnelIPv6Addr\_t

```
/*
 *   IPv4 Tunnel Addresses
 */
typedef struct {
    NPF_IPv4Address_t destAddr;         /* Tunnel destination address */
    NPF_IPv4Address_t srcAddr;         /* Tunnel source address */
} NPF_IfTunnelIPv4Addr_t;

/*
 *   IPv6 Tunnel Addresses
 */
typedef struct {
    NPF_IPv6Address_t destAddr;         /* Tunnel destination address */
    NPF_IPv6Address_t srcAddr;         /* Tunnel source address */
} NPF_IfTunnelIPv6Addr_t;
```

## 3.2 Data Structures for Completion Callbacks

### 3.2.1 Completion Callback Type Codes: NPF\_IfCallbackType\_t

```
/*
```

```

*   Completion Callback Types
*/
typedef enum NPF_IfCallbackType {
    NPF_IF_CREATE = 1,
    NPF_IF_DELETE = 2,
    NPF_IF_BIND = 3,
    NPF_IF_UN_BIND = 4,
    NPF_IF_STATS_GET = 5,
    NPF_IF_VCC_STATS_GET = 6,
    NPF_IF_ATTR_SET = 7,
    NPF_IF_CREATE_AND_SET = 8,
    NPF_IF_ENABLE = 9,
    NPF_IF_DISABLE = 10,
    NPF_IF_OPER_STATUS_GET = 11,
    NPF_IF_MTU_SET = 12,
    NPF_IF_ATTR_GET = 13,

    NPF_IF_LAN_SRC_ADDR_GET = 14,
    NPF_IF_LAN_SRC_ADDR_SET = 15,
    NPF_IF_LAN_ADDR_LIST_SET = 16,
    NPF_IF_LAN_ADDR_LIST_ADD = 17,
    NPF_IF_LAN_PROMISC_SET = 18,
    NPF_IF_LAN_PROMISC_CLEAR = 19,
    NPF_IF_LAN_FULL_DUPLEX_SET = 20,
    NPF_IF_LAN_FULL_DUPLEX_CLEAR = 21,
    NPF_IF_LAN_SPEED_SET = 22,
    NPF_IF_LAN_FLOW_CONTROL_TX_ENABLE = 23,
    NPF_IF_LAN_FLOW_CONTROL_TX_DISABLE = 24,
    NPF_IF_LAN_FLOW_CONTROL_RX_ENABLE = 25,
    NPF_IF_LAN_FLOW_CONTROL_RX_DISABLE = 26,

    NPF_IF_IP_FWD_ENABLE = 27,
    NPF_IF_IP_FWD_DISABLE = 28,

    NPF_IF_IPV4ADDR_SET = 29,
    NPF_IF_IPV4ADDR_CLEAR = 30,
    NPF_IF_IPV4MTU_SET = 31,
    NPF_IF_IPV4FIB_SET = 32,
    NPF_IF_IPV4_UC_ADDR_SET = 33,
    NPF_IF_IPV4_UC_ADDR_ADD = 34,
    NPF_IF_IPV4_UC_ADDR_DELETE = 35,
    NPF_IF_IPV4_MCAST_ADDR_SET = 36,
    NPF_IF_IPV4_MCAST_ADDR_ADD = 37,

    NPF_IF_IPV6_ADDR_SET = 38,
    NPF_IF_IPV6_ADDR_ADD = 39,
    NPF_IF_IPV6_ADDR_DELETE = 40,
    NPF_IF_IPV6_FIB_SET = 41,

    NPF_IF_ATM_VCC_SET = 42,
    NPF_IF_ATM_VCC_BIND = 43,
    NPF_IF_ATM_VCC_UN_BIND = 44,
    NPF_IF_ATM_VCC_DELETE = 45,

    NPF_IF_TUNNEL_HOPS_SET = 46,
    NPF_IF_TUNNEL_DSCP_SET = 47,
    NPF_IF_TUNNEL_IPV4_ADDR_SET = 48,

```

```

    NPF_IF_TUNNEL_IPV6_ADDR_SET = 49,
    NPF_IF_TUNNEL_IPV6_FLOW_LABEL_SET = 50
} NPF_IfCallbackType_t;

```

### 3.2.2 Asynchronous Response Array Element: NPF\_IfAsyncResponse\_t

```

/*
 * An asynchronous response contains an interface handle,
 * an 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_IfID_t       ifID;     /* Interface ID */
    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_IfATM_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() */
        NPF_IfGeneric_t    *attrs; /* NPF_IfAttrGet() */
        NPF_IPv4Prefix_t    v4prefix; /* NPF_IPv4UC_AddrAdd(), Set(), */
        /* Delete() */
        NPF_IPv4Address_t    v4addr; /* NPF_IPv4McastAddrAdd(), Set() */
        NPF_IPv6Prefix_t    v6prefix; /* NPF_IPv6AddrAdd(), Set(), */
        /* Delete() */
    } u;
} NPF_IfAsyncResponse_t;

```

The following table summarizes the information returned by each function in this API.

| Function Name         | Type Code               | Structure Returned        |
|-----------------------|-------------------------|---------------------------|
| NPF_IfCreate          | NPF_IF_CREATE           | Unused                    |
| NPF_IfDelete          | NPF_IF_DELETE           | Unused                    |
| NPF_IfBind            | NPF_IF_BIND             | NPF_IfHandle_t (child)    |
| NPF_IfUnBind          | NPF_IF_UN_BIND          | NPF_IfHandle_t (child)    |
| NPF_IfGenericStatsGet | NPF_IF_STATS_GET        | NPF_IfStatistics_t        |
| NPF_IfVccStatsGet     | NPF_IF_VCC_STATS_GET    | NPF_ATM_VccStats_t        |
| NPF_IfAttrSet         | NPF_IF_ATTR_SET         | Unused                    |
| NPF_IfCreateAndSet    | NPF_IF_CREATE_AND_SET   | NPF_uint32_t (arrayIndex) |
| NPF_IfEnable          | NPF_IF_ENABLE           | Unused                    |
| NPF_IfDisable         | NPF_IF_DISABLE          | Unused                    |
| NPF_IfOperStatusGet   | NPF_IF_OPER_STATUS_GET  | NPF_IfOperStatus_t        |
| NPF_IfMTU_Set         | NPF_IF_MTU_SET          | Unused                    |
| NPF_IfAttrGet         | NPF_IF_ATTR_GET         | NPF_IfGeneric_t           |
| NPF_IfLAN_SrcAddrGet  | NPF_IF_LAN_SRC_ADDR_GET | NPF_MAC_Address_t         |

| Function Name                  | Type Code                          | Structure Returned |
|--------------------------------|------------------------------------|--------------------|
| NPF>IfLAN_SrcAddrSet           | NPF_IF_LAN_SRC_ADDR_SET            | Unused             |
| NPF>IfLAN_MAC_RcvAddrListSet   | NPF_IF_LAN_ADDR_LIST_SET           | Unused             |
| NPF>IfLAN_MAC_RcvAddrListAdd   | NPF_IF_LAN_ADDR_LIST_ADD           | Unused             |
| NPF>IfLAN_PromiscSet           | NPF_IF_LAN_PROMISC_SET             | Unused             |
| NPF>IfLAN_PromiscClear         | NPF_IF_LAN_PROMISC_CLEAR           | Unused             |
| NPF>IfLAN_FullDuplexSet        | NPF_IF_LAN_FULL_DUPLEX_SET         | Unused             |
| NPF>IfLAN_FullDuplexClear      | NPF_IF_LAN_FULL_DUPLEX_CLEAR       | Unused             |
| NPF>IfLAN_SpeedSet             | NPF_IF_LAN_SPEED_SET               | Unused             |
| NPF>IfLAN_FlowControlTxEnable  | NPF_IF_LAN_FLOW_CONTROL_TX_ENABLE  | Unused             |
| NPF>IfLAN_FlowControlTxDisable | NPF_IF_LAN_FLOW_CONTROL_TX_DISABLE | Unused             |
| NPF>IfLAN_FlowControlRxEnable  | NPF_IF_LAN_FLOW_CONTROL_RX_ENABLE  | Unused             |
| NPF>IfLAN_FlowControlRxDisable | NPF_IF_LAN_FLOW_CONTROL_RX_DISABLE | Unused             |
|                                |                                    |                    |
| NPF>IfIP_FwdEnable             | NPF_IF_IP_FWD_ENABLE               | Unused             |
| NPF>IfIP_FwdDisable            | NPF_IF_IP_FWD_DISABLE              | Unused             |
|                                |                                    |                    |
| NPF>IfIPv4AddrSet              | NPF_IF_IPV4ADDR_SET                | Unused             |
| NPF>IfIPv4AddrClear            | NPF_IF_IPV4ADDR_CLEAR              | Unused             |
| NPF>IfIPv4MTU_Set              | NPF_IF_IPV4MTU_SET                 | Unused             |
| NPF>IfIPv4FIBSet               | NPF_IF_IPV4FIB_SET                 | Unused             |
| NPF_IPv4UC_AddrSet             | NPF_IF_IPV4_UC_ADDR_SET            | NPF_IPv4Prefix_t   |
| NPF_IPv4UC_AddrAdd             | NPF_IF_IPV4_UC_ADDR_ADD            | NPF_IPv4Prefix_t   |
| NPF_IPv4UC_AddrDelete          | NPF_IF_IPV4_UC_ADDR_DELETE         | NPF_IPv4Prefix_t   |
| NPF_IPv4McastAddrSet           | NPF_IF_IPV4_MCAST_ADDR_SET         | NPF_IPv4Addr_t     |
| NPF_IPv4McastAddrAdd           | NPF_IF_IPV4_MCAST_ADDR_ADD         | NPF_IPv4Addr_t     |
|                                |                                    |                    |
| NPF>IfIPv6AddrSet              | NPF_IF_IPV6_ADDR_SET               | NPF_IPv6Prefix_t   |
| NPF>IfIPv6AddrAdd              | NPF_IF_IPV6_ADDR_ADD               | NPF_IPv6Prefix_t   |
| NPF>IfIPv6AddrDelete           | NPF_IF_IPV6_ADDR_DELETE            | NPF_IPv6Prefix_t   |
| NPF>IfIPv6FIB_Set              | NPF_IF_IPV6_FIB_SET                | Unused             |
|                                |                                    |                    |
| NPF>IfATM_VccSet               | NPF_IF_ATM_VCC_SET                 | NPF_VccAddr_t      |
| NPF>IfATM_VccBind              | NPF_IF_ATM_VCC_BIND                | NPF_VccAddr_t      |
| NPF>IfATM_VccUnBind            | NPF_IF_ATM_VCC_UN_BIND             | NPF_VccAddr_t      |
| NPF>IfATM_VccDelete            | NPF_IF_ATM_VCC_DELETE              | NPF_VccAddr_t      |
|                                |                                    |                    |
| NPF>IfTunnelHopsSet            | NPF_IF_TUNNEL_HOPS_SET             | Unused             |
| NPF>IfTunnelDSCP_Set           | NPF_IF_TUNNEL_DSCP_SET             | Unused             |
| NPF>IfTunnelIPv4AddrSet        | NPF_IF_TUNNEL_IPV4_ADDR_SET        | Unused             |
| NPF>IfTunnelIPv6AddrSet        | NPF_IF_TUNNEL_IPV6_ADDR_SET        | Unused             |
| NPF>IfTunnelIPv6FlowLabelSet   | NPF_IF_TUNNEL_IPV6_FLOW_LABEL_SET  | Unused             |

### 3.2.3 Callback Data Structure: NPF>IfCallbackData\_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            n_resp;       /* 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 prefix length */
#define NPF_IF_E_INVALID_PLEN ((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)

/* Invalid TTL value */
#define NPF_IF_E_INVALID_TTL ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+27)

/* Invalid DSCP value */
#define NPF_IF_E_INVALID_DSCP ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+28)

/* Invalid IPv4 Multicast Address */
#define NPF_IF_E_INVALID_IPV4_MCAST_ADDR ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+29)

/* Parent/child binding not found */
#define NPF_IF_E_NO_SUCH_BINDING ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+30)

/* IP address does not exist on this interface */
#define NPF_IF_E_NO_SUCH_ADDRESS ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+31)

/* Parent/child binding is circular */
#define NPF_IF_E_CIRCULAR_BINDING ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+32)

typedef NPF_uint32_t NPF_IfErrorType_t;
```

## 3.4 Data Structures for Event Notifications

### 3.4.1 Event Types: NPF\_IfEvent\_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;

```

### 3.4.2 Event Notification Structure and Array: NPF\_IfEventData\_t and NPF\_IfEventArray\_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;

```

## 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 the `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 NPF\_IfRegister: 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 or not this should be treated as an error is dependent on the application.

## 4.1.3 NPF\_IfDeregister: 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 an 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 NPF\_IfEventRegister: 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 first 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 or not this should be treated as an error is dependent on the application.

## 4.2.3 NPF>IfEventDeregister: 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 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]).

### 4.4 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.

### 4.5 Completion Callbacks and Error Returns

Each of the functions defined in section 4.6 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 4.6 with the definitions of the functions that return them.

### 4.6 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.

#### 4.6.1 NPF>IfCreate: 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,
NPF_IN NPF_IfID_t         *ifID);

```

## Description

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.
- **ifID**: a pointer to an array of Interface ID values. The number of elements in the array is given by **n\_if**. The values must all be different from each other, and none may be the same as the ID of an existing interface.

## Output Parameters

None

## Asynchronous Error Codes

- **NPF\_NO\_ERROR**: operation successful.
- **NPF\_E\_RESOURCE\_EXISTS**: an interface with the same Interface ID value already exists; its handle is returned in the callback, and no new interface is created.
- **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.

## 4.6.2 NPF\_IfDelete: 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

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.

## 4.6.3 NPF\_IfBind: 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            nbinds,
    NPF_IN NPF_IfBinding_t        *if_bindArray);
```

### Description

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.

Bindings have the following characteristics:

- Adding a parent to an interface can mean that a particular protocol can be carried on the link represented by the child.
- Setting a parent Administratively UP or DOWN controls the processing of the protocol represented by the parent; for instance, setting an IPv4 interface down would cause all incoming IPv4 packets received on any of that interface's child interfaces to be discarded.
- Removing a binding does not result in either interface being deleted.

### Input Parameters

- **if\_cbHandle**: the registered callback handle.

- **if\_cbCorrelator**: the application's context for this call.
- **if\_errorReporting**: the desired callback.
- **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.
- **NPF\_IF\_E\_CIRCULAR\_BINDING**: An interface would exist more than once in its own parent/child hierarchy. 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.

## 4.6.4 NPF>IfUnBind: Remove Interface Bindings

### Syntax

```
NPF_error_t NPF>IfUnBind(
    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 nbinds,
    NPF_IN NPF>IfBinding_t *if_bindArray);
```

### Description

This function removes one or more interface parent-child relationships previously set by **NPF>IfBind()** calls.

### Input Parameters

- **if\_cbHandle**: the registered callback handle.
- **if\_cbCorrelator**: the application's context for this call.
- **if\_errorReporting**: the desired callback.
- **nbinds**: number of bindings in the array.
- **if\_bindArray**: pointer to an array of interface handle parent/child bindings to be removed.

### Output Parameters

None

### Asynchronous Error Codes

- **NPF\_NO\_ERROR**: Operation successful.

- **NPF\_IF\_E\_NO\_SUCH\_BINDING**: A specified binding could not be found.

### 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.

## 4.6.5 NPF\_IfGenericStatsGet: 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

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.

## 4.6.6 NPF\_IfVccStatsGet: 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

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.

## 4.6.7 NPF\_IfAttrSet: 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

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. The Interface Handle value identifies the interface to be modified; the Interface ID value in the `NPF_ifGeneric_t` structure is ignored.

### 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 structure or 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_PLEN`: Invalid 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.

## 4.6.8 NPF\_ifCreateAndSet: 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

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). Each instance of the `NPF_IfGeneric_t` structure must contain a different, nonzero Interface ID value, and none may be the same as that of an existing interface.

## 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_E_RESOURCE_EXISTS`: an interface with the same Interface ID value already exists; its handle is returned in the callback, and no new interface is created.
- `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_PLEN`: Invalid 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.

## 4.6.9 NPF\_IfEnable: 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

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.

## 4.6.10 NPF\_IfDisable: 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

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.

## 4.6.11 NPF\_IfOperStatusGet: 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

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.

### 4.6.12 NPF\_IfMTU\_Set: Set an Interface's MTU

#### Syntax

```
NPF_error_t NPF_IfMTU_Set(
    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_MTU_Array);
```

#### Description

This function sets the Maximum Transmission Unit (MTU) of one or more IP interfaces. These may be IPv4 interfaces, IPv6 interfaces, or any kind of IP tunnel interface. The `if_HandleArray` and `if_MTU_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.

**Note:** this function is identical to `NPF_IfIPv4MTUSet()`, without the restriction to IPv4 type 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 set the MTU for.
- `if_HandleArray`: the handle of each interface.
- `if_MTU_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 IPv6 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.

### 4.6.13 NPF\_IfAttrGet: Read Interface Attributes

#### Syntax

```

NPF_error_t  NPF_IfAttrGet(
    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

This function returns, via a callback, a pointer to a generic interface structure (`NPF_IfGeneric_t`) containing the current attributes of one or more indicated interfaces. **This is an optional function.**

#### 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 attributes 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_IfGeneric_t` structure for that interface.

### 4.6.14 NPF\_IfLAN\_SrcAddrGet: 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

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.

## 4.6.15 NPF>IfLAN\_SrcAddrSet: 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

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.

## 4.6.16 NPF>IfLAN\_MAC\_RcvAddrListSet: 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

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.

## 4.6.17 NPF>IfLAN\_MAC\_RcvAddrListAdd: 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

This function adds addresses to a list of receive MAC addresses (represented as an array) in 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.

## 4.6.18 NPF\_IflAN\_PromiscSet: 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

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.

## 4.6.19 NPF\_IflAN\_PromiscClear: 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

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.

## 4.6.20 NPF\_IfLAN\_FullDuplexSet: 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

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.

## 4.6.21 NPF>IfLAN\_FullDuplexClear: 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

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.

## 4.6.22 NPF>IfLAN\_SpeedSet: 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

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.

## 4.6.23 NPF>IfLAN\_FlowControlTxEnable: 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

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.

## 4.6.24 NPF>IfLAN\_FlowControlTxDisable: 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

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.

## 4.6.25 NPF>IfLAN\_FlowControlRxEnable: 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

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.

## 4.6.26 NPF>IfLAN\_FlowControlRxDisable: 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

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.

## 4.6.27 NPF>IfIP\_FwdEnable: Enable IP Forwarding on One or More IPv4 or IPv6 Interfaces

### Syntax

```
NPF_error_t NPF>IfIP_FwdEnable(
    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

This function enables IP forwarding on one or more IPv4 or IPv6 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 on which to enable forwarding.
- **if\_HandleArray**: pointer to an array of IP (IPv4 or IPv6) interface handles.

### Output Parameters

None

### Asynchronous Error Codes

- **NPF\_NO\_ERROR**: Operation successful.
- **NPF\_IF\_E\_INVALID\_HANDLE**: An interface handle is null or invalid, or not an IPv4 or IPv6 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.

### 4.6.28 NPF\_IfIP\_FwdDisable: Disable IP Forwarding on one or more IP Interfaces

#### Syntax

```
NPF_error_t NPF_IfIP_FwdDisable(
    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

This function disables IP forwarding on one or more IP interfaces. When IP forwarding is disabled, the interface can still send and receive IP datagrams as long as the interface is administratively UP and the underlying L2 interface is operationally and administratively UP.

#### 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 on which to disable IP forwarding.
- `if_HandleArray`: pointer to an array of IP (IPv4 or IPv6) interface handles.

#### Output Parameters

None

#### Asynchronous Error Codes

- `NPF_NO_ERROR`: Operation successful.
- `NPF_IF_E_INVALID_HANDLE`: An interface handle is null or invalid, or not an IPv4 or IPv6 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.

### 4.6.29 NPF\_IfIPv4AddrSet: Set an IPv4 Interface's IP Address

#### 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_IPv4Prefix_t      *if_IPv4AddrArray);
```

## Description

This function sets the Primary IP address and 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_PLEN`: Invalid 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.

## 4.6.30 NPF>IfIPv4AddrClear: Clear an IPv4 Interface's Primary IP Address

### Syntax

```
NPF_error_t NPF>IfIPv4AddrClear(
    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

This function clears the Primary IP address and prefix length of one or more IPv4 interfaces. If the secondary list of IP addresses is empty, the interface has no IP address (it is “unnumbered”) after this call is complete. Note that only point-to-point interfaces can function without an 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 IP 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 an IPv4 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.

## 4.6.31 NPF>IfIPv4MTUSet: 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

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 (too large for the underlying device, less than 576, or other error as determined by the implementation)

## 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.

### 4.6.32 NPF\_IfIPv4FIBSet: Associate an IPv4 FIB to an 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_IPv4UC_FwdTableHandle_t    if_FIB_Handle);
```

#### Description

This function associates an IPv4 Forwarding Table (FIB) with one or more IPv4 or IPv4 tunnel 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.

### 4.6.33 NPF\_IfIPv4UC\_AddrSet: Set an IPv4 Interface's Secondary List of IP Prefixes

#### Syntax

```
NPF_error_t NPF_IfIPv4UC_AddrSet(
    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           nAddrs,
NPF_IN NPF_IPv4Prefix_t       *if_IPv4AddrArray);

```

## Description

This function sets (or replaces) an interface's secondary list of unicast IPv4 prefixes. The effect of placing a prefix in this array is to cause arriving packets addressed to that prefix to be delivered locally. If the given array is empty, all secondary unicast IPv4 prefixes are removed.

## 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 affected interface.
- **nAddrs**: the number of entries in the IP prefix array. Set this to zero to remove all prefixes.
- **if\_IPv4AddrArray**: pointer to an array of IPv4 prefix structures. This may be NULL if **nAddrs** is zero.

## 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 IPv4 interface.
- **NPF\_IF\_E\_INVALID\_IPADDR**: IP address is not a valid unicast address.
- **NPF\_IF\_E\_INVALID\_PLEN**: Invalid prefix length.

## Asynchronous Response

If **nAddrs** is nonzero, a total of **nAddrs** asynchronous responses (**NPF\_IfAsyncResponse\_t**) will be passed to the callback function, in one or more invocations. Each response contains one of the given prefixes and a success code or a possible error code for that prefix. If **nAddrs** is given as zero, a single response (**NPF\_IfAsyncResponse\_t**) is returned, containing only a success/failure code.

## 4.6.34 NPF\_IfIPv4UC\_AddrAdd: Add to an IPv4 Interface's Secondary List of IP Prefixes

### Syntax

```

NPF_error_t NPF_IfIPv4UC_AddrAdd(
    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           nAddrs,
    NPF_IN NPF_IPv4Prefix_t       *if_IPv4AddrArray);

```

## Description

This function adds more IPv4 prefixes to an interface's secondary list of unicast IPv4 prefixes. The effect of adding an prefix to this array is to cause arriving packets addressed to that prefix to be delivered

locally. If a given prefix is already in the array, or is already the Primary IP address, that prefix is not added a second time.

### 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 affected interface.
- **nAddrs**: the number of entries in the IP prefix array. This must not be zero.
- **if\_IPv4AddrArray**: pointer to an array of IPv4 prefix 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 IPv4 interface.
- **NPF\_IF\_E\_INVALID\_IPADDR**: IP address is not a valid unicast address.
- **NPF\_IF\_E\_INVALID\_PLEN**: Invalid prefix length.

### Asynchronous Response

A total of **nAddrs** asynchronous responses (**NPF>IfAsyncResponse\_t**) will be passed to the callback function, in one or more invocations. Each response contains one of the given prefixes and a success code or a possible error code for that prefix.

## 4.6.35 NPF>IfIPv4UC\_AddrDelete: Delete Prefixes From an IPv4 Interface's Secondary List of IP Prefixes

### Syntax

```
NPF_error_t  NPF>IfIPv4UC_AddrAdd(
    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          nAddrs,
    NPF_IN NPF_IPv4Prefix_t      *if_IPv4AddrArray);
```

### Description

This function deletes IPv4 prefixes from an interface's secondary list of unicast IPv4 prefixes.

### 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 affected interface.
- **nAddrs**: the number of entries in the IP prefix array. This must not be zero.
- **if\_IPv4AddrArray**: pointer to an array of IPv4 prefix 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 IPv4 interface.
- **NPF\_IF\_E\_INVALID\_IPADDR**: IP address is not a valid unicast address.
- **NPF\_IF\_E\_INVALID\_PLEN**: Invalid prefix length.
- **NPF\_IF\_E\_NO\_SUCH\_ADDRESS**: Address not found on this interface.

## Asynchronous Response

A total of **nAddrs** asynchronous responses (**NPF>IfAsyncResponse\_t**) will be passed to the callback function, in one or more invocations. Each response contains one of the given prefixes and a success code or a possible error code for that prefix.

## 4.6.36 NPF>IfIPv4McastAddrSet: Set an IPv4 Interface's List of Receive Multicast IP Addresses

### Syntax

```
NPF_error_t NPF>IfIPv4McastAddrSet(
    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 nAddrs,
    NPF_IN NPF_IPv4Address_t *mcAddrArray);
```

### Description

This function sets (or replaces) an interface's list of receive multicast IPv4 addresses. The effect of placing an address in this array is to cause arriving multicast packets addressed to that address to be delivered locally. If the given array length is zero, all multicast IPv4 addresses are removed.

### 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 affected interface.
- **nAddrs**: the number of entries in the IP address array; zero to remove all addresses.
- **mcAddrArray**: pointer to an array of IPv4 multicast addresses. This may be NULL if **nAddrs** is zero.

### 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 IPv4 interface.
- **NPF\_IF\_E\_INVALID\_IPMCAST\_ADDR**: IP address is not a valid multicast address.

## Asynchronous Response

If `nAddrs` is nonzero, a total of `nAddresses` asynchronous responses (`NPF>IfAsyncResponse_t`) will be passed to the callback function, in one or more invocations. Each response contains one of the given addresses and a success code or a possible error code for that address. If `nAddrs` is given as zero, a single response (`NPF>IfAsyncResponse_t`) is returned, containing only a success/failure code.

### 4.6.37 NPF>IfIPv4McastAddrAdd: Add to an IPv4 Interface's List of Receive Multicast IP Addresses

#### Syntax

```
NPF_error_t NPF>IfIPv4McastAddrAdd(
    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            nAddrs,
    NPF_IN NPF_IPv4Address_t       *mcAddrArray);
```

#### Description

This function adds new addresses to an interface's list of receive multicast IPv4 addresses. The effect of adding an address to this array is to cause arriving multicast packets addressed to that address to be delivered locally. If a given address is already in the array, that address is not added a second time.

#### 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 affected interface.
- `nAddrs`: the number of entries in the IP address array.
- `mcAddrArray`: pointer to an array of IPv4 multicast 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 IPv4 interface.
- `NPF_IF_E_INVALID_IPMCAST_ADDR`: IP address is not a valid multicast address.

#### Asynchronous Response

A total of `nAddrs` asynchronous responses (`NPF>IfAsyncResponse_t`) will be passed to the callback function, in one or more invocations. Each response contains one of the given addresses and a success code or a possible error code for that address.

### 4.6.38 NPF>IfIPv6AddrSet: Set the Addresses for an IPv6 Interface

#### Syntax

```
NPF_error_t NPF>IfIPv6AddrSet(
    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         handle,
NPF_IN NPF_uint32_t          nAddresses,
NPF_IN NPF_IPv6Prefix_t      *if_IPv6AddrArray);

```

## Description

This function sets (or replaces) one or more IPv6 addresses on an IPv6 interface. If the interface already has one or more addresses assigned, they are all removed and replaced by the new set. If **nAddresses** is zero, all addresses are removed; the interface becomes an “unnumbered” interface. Note that only point-to-point interfaces can function without an address.

## Input Parameters

- **if\_cbHandle**: the registered callback handle.
- **if\_cbCorrelator**: the application’s context for this call.
- **if\_errorReporting**: the desired callback.
- **handle**: the handle of the interface to modify.
- **nAddresses**: the number addresses to assign to the interface.
- **if\_IPv6AddrArray**: pointer to an array of IPv6 address structures.

## Output Parameters

None

## Asynchronous Error Codes

- **NPF\_NO\_ERROR**: Operation successful.
- **NPF\_IF\_E\_INVALID\_HANDLE**: The **handle** argument is null or invalid, or is not an IPv6 interface.
- **NPF\_IF\_E\_INVALID\_IPADDR**: IP address is not a valid IPv6 address.
- **NPF\_IF\_E\_INVALID\_PLEN**: Invalid prefix length.

## Asynchronous Response

If **nAddresses** is nonzero, a total of **nAddresses** asynchronous responses (**NPF\_IfAsyncResponse\_t**) will be passed to the callback function, in one or more invocations. Each response contains one of the given addresses and a success code or a possible error code for that address. If **nAddresses** is given as zero, a single response (**NPF\_IfAsyncResponse\_t**) is returned, containing only a success/failure code.

## 4.6.39 NPF\_IfIPv6AddrAdd: Add Addresses to an IPv6 Interface

### Syntax

```

NPF_error_t NPF_IfIPv6AddrAdd(
    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         handle,
    NPF_IN NPF_uint32_t          nAddresses,
    NPF_IN NPF_IPv6Prefix_t      *if_IPv6AddrArray);

```

## Description

This function adds IPv6 addresses to an IPv6 interface. If the interface already has one or more addresses assigned, they are not removed; the set of addresses is extended by adding the new ones to it.

### Input Parameters

- **if\_cbHandle**: the registered callback handle.
- **if\_cbCorrelator**: the application's context for this call.
- **if\_errorReporting**: the desired callback.
- **handle**: the handle of the interface to modify.
- **nAddresses**: the number of addresses add. This must be greater than zero.
- **if\_IPv6AddrArray**: pointer to an array of IPv6 address structures (may be NULL if **nAddresses** is zero).

### Output Parameters

None

### Asynchronous Error Codes

- **NPF\_NO\_ERROR**: Operation successful.
- **NPF\_IF\_E\_INVALID\_HANDLE**: The **handle** argument is null or invalid, or is not an IPv6 interface.
- **NPF\_IF\_E\_INVALID\_IPADDR**: IP address is not a valid IPv6 address.
- **NPF\_IF\_E\_INVALID\_PLEN**: Invalid prefix length.

### Asynchronous Response

A total of **nAddresses** asynchronous responses (**NPF>IfAsyncResponse\_t**) will be passed to the callback function, in one or more invocations. Each response contains one of the given addresses and a success code or a possible error code for that address.

## 4.6.40 NPF>IfIPv6AddrDelete: Delete Addresses From an IPv6 Interface's List of Unicast IP Addresses

### Syntax

```
NPF_error_t NPF>IfIPv6AddrDelete(
    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            nAddrs,
    NPF_IN NPF_IPv6Prefix_t        *if_IPv6AddrArray);
```

### Description

This function deletes addresses from an IPv6 interface's list of unicast addresses.

### 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 affected interface.

- **nAddrs**: the number of entries in the IP address array. This must not be zero.
- **if\_IPv6AddrArray**: pointer to an array of IPv6 Prefix 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 IPv6 interface.
- **NPF\_IF\_E\_INVALID\_IPADDR**: IP address is not a valid unicast address.
- **NPF\_IF\_E\_INVALID\_PLEN**: Invalid prefix length.
- **NPF\_IF\_E\_NO\_SUCH\_ADDRESS**: Address not found on this interface.

### Asynchronous Response

One asynchronous response structure (**NPF\_IfAsyncResponse\_t**) will be passed to the callback function, containing the interface handle and a success code or a possible error code. The union in the callback response structure is unused.

## 4.6.41 NPF\_IfIPv6FIB\_Set: Associate an IPv6 FIB with an Interface

### Syntax

```
NPF_error_t  NPF_IfIPv6FIB_Set(
    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_IPv6UC_FwdTableHandle_t  if_FIB_Handle);
```

### Description

This function associates an IPv6 Forwarding Table (FIB) with one or more IPv6 or IPv6 tunnel 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 IPv6 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.

## 4.6.42 NPF\_IfATM\_VccSet: 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

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.

#### 4.6.43 NPF\_IfATM\_VccBind: 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

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 the ATM interface handle and a success code or a possible error code. The union in the callback response structure contains the VPI/VCI address of the pertinent Vcc.

#### 4.6.44 NPF\_IfATM\_VccUnBind: Remove Binding of a Higher Layer Interface to an ATM Vcc

##### Syntax

```

NPF_error_t  NPF_IfATM_VccUnBind(
    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

This function removes the binding of a particular higher-layer interface with one or more Vccs on the same ATM UNI interface. The bindings must have been previously created by `NPF>IfATM_VccBind()` calls.

## 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 unbind.
- `n_Vccs`: the number of Vccs to remove the binding from.
- `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_NO_SUCH_BINDING`: The specified binding was not found.
- `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 the ATM interface handle and a success code or a possible error code. The union in the callback response structure contains the VPI/VCI address of the pertinent Vcc.

### 4.6.45 NPF>IfATM\_VccDelete: 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

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.

## 4.6.46 NPF>IfTunnelHopsSet: Set IP-in-IP Tunnel Length (Hops)

### Syntax

```
NPF_error_t NPF>IfTunnelHopsSet(
    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            nHandles,
    NPF_IN NPF>IfHandle_t          *if_Handle,
    NPF_IN NPF_uint8_t            hopcount);
```

### Description

This function sets the tunnel length (hop count) on one or more IP-in-IP Tunnel interfaces. If multiple interface handles are given, the same length value is applied to each. This value can be used in setting the TTL or Hop Limit field of the outgoing tunnel header.

### Input Parameters

- **if\_cbHandle**: the registered callback handle.
- **if\_cbCorrelator**: the application's context for this call.
- **if\_errorReporting**: the desired callback.
- **nHandles**: The number of handles in the Interface Handle array.
- **if\_Handle**: pointer to an array of one or more Interface Handles.
- **hopcount**: The tunnel length value.

### 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 a tunnel interface.

## Asynchronous Response

One asynchronous response structure (**NPF>IfAsyncResponse\_t**) for each interface will be passed to the callback function, each containing an interface handle and a success code or a possible error code. The union in the callback response structure is unused.

## 4.6.47 NPF>IfTunnelDSCP\_Set Set IP Tunnel DSCP

### Syntax

```
NPF_error_t NPF>IfTunnelDSCP_Set(
    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_uint8_t             *if_DSCPArray);
```

### Description

This function assigns IPv4 DiffServ Code Point value to one or more IP tunnel interfaces (IPv4, IPv6, or IPv6in4 tunnel type). The `if_Handle` and `if_IPv4DSCP` arrays must both contain the same number of entries, equal to the value of `n_handles`; and an element in the `n`th position in one array must correspond to the element in the `n`th position in the other.

### 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 DSCP for.
- **if\_HandleArray**: pointer to an array of interface handles.
- **if\_DSCPArray**: pointer to an array of IPv4 DSCP values.

### 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 IPv6in4 interface.
- **NPF\_IF\_E\_INVALID\_DSCP**: A IPv4DSCP is not a valid IPv4 DiffServ code point value.

### 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.

## 4.6.48 NPF\_IfTunnelIPv4AddrSet: Set Tunnel's IPv4 Addresses

### Syntax

```
NPF_error_t NPF_IfTunnelIPv4AddrSet(
    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            nHandles,
    NPF_IN NPF_IfHandle_t          *if_Handle,
    NPF_IN NPF_IfTunnelIPv4Addr_t *addrArray);
```

### Description

This function sets the source and destination IP addresses on one or more IPv4-in-IPv4 or IPv6-in-IPv4 Tunnel interfaces.

### Input Parameters

- **if\_cbHandle**: the registered callback handle.
- **if\_cbCorrelator**: the application's context for this call.
- **if\_errorReporting**: the desired callback.
- **nHandles**: The number of handles in the Interface Handle array and the address array.
- **if\_Handle**: pointer to an array of one or more Interface Handles.
- **addrArray**: Array of source/destination IPv4 address pairs, one pair for each interface handle.

### 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 a tunnel interface.
- **NPF\_IF\_E\_INVALID\_IPADDR**: Invalid source or destination IP address given.

### Asynchronous Response

One asynchronous response structure (**NPF\_IfAsyncResponse\_t**) for each interface will be passed to the callback function, each containing an interface handle and a success code or a possible error code. The union in the callback response structure is unused.

## 4.6.49 NPF\_IfTunnelIPv6AddrSet: Set IPv6-in-IPv6 Tunnel Addresses

### Syntax

```
NPF_error_t NPF_IfTunnelIPv6AddrSet(
    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            nHandles,
    NPF_IN NPF_IfHandle_t          *if_Handle,
    NPF_IN NPF_IfTunnelIPv6Addr_t *addrArray);
```

### Description

This function sets the source and destination IP addresses on one or more IPv6-in-IPv6 Tunnel interfaces.

## Input Parameters

- **if\_cbHandle**: the registered callback handle.
- **if\_cbCorrelator**: the application's context for this call.
- **if\_errorReporting**: the desired callback.
- **nHandles**: The number of handles in the Interface Handle array and the address array.
- **if\_Handle**: pointer to an array of one or more Interface Handles.
- **addrArray**: Array of source/destination IPv6 address pairs, one pair for each interface handle.

## 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 a tunnel interface.
- **NPF\_IF\_E\_INVALID\_IPADDR**: Invalid source or destination IP address given.

## Asynchronous Response

One asynchronous response structure (**NPF>IfAsyncResponse\_t**) for each interface will be passed to the callback function, each containing an interface handle and a success code or a possible error code. The union in the callback response structure is unused.

## 4.6.50 NPF>IfTunnelIPv6FlowLabelSet: Set IPv6-in-IPv6 Tunnel Flow Label

### Syntax

```
NPF_error_t NPF>IfTunnelIPv6FlowLabelSet(
    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            nHandles,
    NPF_IN NPF>IfHandle_t          *if_Handle,
    NPF_IN NPF_uint32_t            *labelArray);
```

### Description

This function sets the Flow Label on one or more IP-in-IPv6 Tunnel interfaces. This value is placed in the Flow Label field of the tunnel header when a packet is encapsulated for sending.

### Input Parameters

- **if\_cbHandle**: the registered callback handle.
- **if\_cbCorrelator**: the application's context for this call.
- **if\_errorReporting**: the desired callback.
- **nHandles**: The number of handles in the Interface Handle array and the address array.
- **if\_Handle**: pointer to an array of one or more Interface Handles.
- **labelArray**: Array of Flow Label values, one for each interface handle.

### 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 a tunnel interface.

## Asynchronous Response

One asynchronous response structure (**NPF>IfAsyncResponse\_t**) for each interface will be passed to the callback function, each containing an interface handle and a success code or a possible error code. The union in the callback response structure is unused.

## 5 References

- 1 McCloughrie, K., Kastenholz, F., “The Interfaces Group MIB”, Internet Engineering Task Force RFC 2863, June 2000.
- 2 NP Forum – Software API Conventions Implementation Agreement Revision 2.0.

## 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_IfUnBind()                   | 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_IfMTU_Set                    | Yes                                       |
| NPF_IfAttrGet                    | No  |
| 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_IfIP_FwdEnable()             | Only if IPv4 or IPv6 interfaces supported |
| NPF_IfIP_FwdDisable()            | Only if IPv4 or IPv6 interfaces supported |
| NPF_IfIPv4AddrSet()              | Only if IPv4 interfaces supported         |
| NPF_IfIPv4MTUSet()               | Only if IPv4 interfaces supported         |
| NPF_IfIPv4FIBSet()               | Only if IPv4 interfaces supported         |
| NPF_IPv4UC_AddrSet()             | Only if IPv4 interfaces supported         |
| NPF_IPv4UC_AddrAdd()             | Only if IPv4 interfaces supported         |
| NPF_IPv4UC_AddrDelete()          | Only if IPv4 interfaces supported         |
| NPF_IPv4McastAddrSet()           | Only if IPv4 interfaces supported         |

| <b>Function Name</b>           | <b>Required?</b>                         |
|--------------------------------|--|
| NPF_IPv4McastAddrAdd()         | Only if IPv4 interfaces supported        |
| NPF_IfIPv6AddrSet()            | Only if IPv6 interfaces supported        |
| NPF_IfIPv6AddrAdd()            | Only if IPv6 interfaces supported        |
| NPF_IfIPv6AddrDelete()         | Only if IPv6 interfaces supported        |
| NPF_IfIPv6FIB_Set()            | Only if IPv6 interfaces supported        |
| NPF_IfATM_VccSet()             | Only if ATM UNI interfaces supported     |
| NPF_IfATM_VccBind()            | Only if ATM UNI interfaces supported     |
| NPF_IfATM_VccUnBind()          | Only if ATM UNI interfaces supported     |
| NPF_IfATM_VccDelete()          | Only if ATM UNI interfaces supported     |
| NPF_IfTunnelHopsSet()          | Only if Tunnel interfaces supported      |
| NPF_IfTunnelDSCP_Set()         | Only if Tunnel interfaces supported      |
| NPF_IfTunnelIPv4AddrSet()      | Only if IPv4 Tunnel interfaces supported |
| NPF_IfTunnelIPv6AddrSet()      | Only if IPv6 Tunnel interfaces supported |
| NPF_IfTunnelIPv6FlowLabelSet() | Only if IPv6 Tunnel interfaces supported |

### 6.3 API Events

| <b>Event Name</b>            | <b>Required?</b> |
|------------------------------|------------------|
| NPF_IF_UP                    | Yes              |
| NPF_IF_DOWN                  | Yes              |
| NPF_IF_COUNTER_DISCONTINUITY | Yes              |

**APPENDIX A    HEADER FILE: NPF\_IF.H**

```

/*
 * This header file defines typedefs, constants, and functions
 * that apply to the NPF Interface Management API, version 2.0.
 */
#ifndef __NPF_IF_V2_0_H__
#define __NPF_IF_V2_0_H__

#ifdef __cplusplus
extern "C" {
#endif

typedef NPF_uint32_t      NPF_IfID_t;          /* Interface Identifier */

#ifndef __NPF_H__
/*
 * Interface handle
 */
typedef NPF_uint32_t      NPF_IfHandle_t;
#endif

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_IPV6=6,        /* IPv6 logical interface */
    NPF_IF_TYPE_IPV6INV4=7,    /* IPv6inv4 logical interface */
    NPF_IF_TYPE_TUNNEL_IPV4=8, /* IP-in-IPV4 tunnel */
    NPF_IF_TYPE_TUNNEL_IPV6=9, /* IP-in-IPV6 tunnel */
    NPF_IF_TYPE_HIGHEST=9     /* 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;

/*
 * IP Forwarding mode code
 */
typedef enum          {
        NPF_IF_IP_FORWARDING_ENABLE = 1,   /* Enable IP Forwarding */
        NPF_IF_IP_FORWARDING_DISABLE = 2   /* Disable IP Forwarding */
} NPF_IfIpFwdMode_t;

/*
 * IPv4 address prefix structure (Defined globally)
 */
typedef struct {
        NPF_IPv4Address_t IPv4Addr;        /* IPv4 address */
        NPF_uint8_t       IPv4Plen;        /* Prefix length in bits (1-32) */
} NPF_IPv4Prefix_t;

/*
 * IPv6 Address Flags
 */

typedef NPF_uint8_t    NPF_IPv6AddrFlags_t;

#define NPF_IF_IPV6_ADDR_FLAGS_NONE 0x00
#define NPF_IF_IPV6_ADDR_ANYCAST 0x01
#define NPF_IF_IPV6_ADDR_PREFERRED 0x02
#define NPF_IF_IPV6_ADDR_DEPRECATED 0x04

/*
 * IPv6 address prefix structure (Defined globally)
 */
typedef struct {
        NPF_IPv6Address_t IPv6Addr;        /* IPv6 address */

```

```

        NPF_uint8_t      IPv6Plen;    /* Prefix length in bits (1-128) */
        NPF_IPv6AddrFlags_t IPv6Flags; /* Anycast, preferred, deprecated*/
} NPF_IPv6Prefix_t;

/*
 * IPv4 Interface attributes
 */
typedef struct {
    NPF_IPv4Prefix_t addr;          /* Primary IPv4 Address and plen */
    NPF_uint16_t      mtu;          /* IPv4 Max Transmission Unit */
    NPF_IPv4UC_FwdTableHandle_t fibHandle; /* Forwarding Info Base*/
    NPF_uint32_t      nUcAddrs;    /* Number of unicast IP addrs */
    NPF_IPv4Prefix_t *ucAddrs;    /* Array of unicast IP addrs */
    NPF_uint32_t      nMcAddrs;    /* Number of mcast IP addrs */
    NPF_uint32_t      *mcAddrs;   /* Array of multicast IP addrs */
    NPF_IfIpFwdMode_t fwdMode;    /* IPv4 Forwarding Mode */
} NPF_IfIPv4_t;

/*
 * IPv6 Interface attributes
 */
typedef struct {
    NPF_uint32_t      n_addr;      /* Number of IPv6 Addresses */
    NPF_IPv6Prefix_t *addr;       /* IPv6 Addresses */
    NPF_uint16_t      mtu;        /* IPv6 Max Transmission Unit */
    NPF_IPv6UC_FwdTableHandle_t fibHandle; /* Forwarding Info Base*/
    NPF_IfIpFwdMode_t fwdMode;   /* IPv6 Forwarding Mode */
} NPF_IfIPv6_t;

/*
 * IPv6 in IPv4 Tunnel types
 */
typedef enum {
    NPF_IF_V6INV4_AUTOMATIC = 1, /* Automatic Tunnel */
    NPF_IF_V6INV4_CONFIGURED = 2, /* Configured Tunnel */
    NPF_IF_V6INV4_6TO4 = 3 /* 6to4 Tunnel*/
} NPF_IPv6inv4TunnelType_t;

/*
 * IPv6inv4 Tunnel Interface attributes
 */
typedef struct {
    NPF_IPv6inv4TunnelType_t tunnelType; /* Type of v6inv4 Tunnel */
    NPF_IPv4Address_t IPv4DstAddr; /* Destination IPv4 address */
    NPF_IPv4Address_t IPv4SrcAddr; /* Source IPv4 address */
    NPF_uint8_t IPv4TTL; /* Time to live in IPv4 header*/
    NPF_uint8_t IPv4DSCP; /* DiffServ Code Point in IPv4
header*/
} NPF_IfIPv6inv4_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;

/*
 *   LAN Control Flags
 */
#define NPF_IF_LAN_FDX_ENABLE           0x01 /* TRUE to enable full duplex */
#define NPF_IF_LAN_PROMISC             0x02 /* TRUE for promiscuous mode */
#define NPF_IF_LAN_TX_FC_ENABLE        0x04 /* TRUE for transmit flow ctrl */
#define NPF_IF_LAN_RX_FC_ENABLE        0x08 /* TRUE for receive flow ctrl */

/*
 *   Generic LAN Interface attributes
 */
typedef struct {
    NPF_uint32_t      port;              /* Port number */
    NPF>IfLAN_Speed_t speed;            /* Speed control */
    NPF_uint32_t      flags;            /* Flags (see above) */
    NPF_MAC_Address_t srcAddr;         /* Source MAC address */
    NPF_uint32_t      nAddrs;          /* Number of receive MAC addrs */
    NPF_MAC_Address_t *rcvAddrs;      /* Array of receive MAC addrs */
} NPF>IfLAN_t;

/*
 *   ATM UNI Interface Attributes
 */
typedef struct {
    NPF_uint32_t      port;              /* Port number */
} NPF>IfATM_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 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;

#ifndef __NPF_H__
/*
 * 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;
#endif

/*
 * 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;

/*
 * IP-in-IPv4 Tunnel Interface Attributes
 */
typedef struct {
    NPF_uint16_t MTU; /* Tunnel max transmission unit */
    NPF_uint8_t maxHops; /* Maximum hops in the tunnel */
    NPF_IPv4Address_t dstAddr; /* Tunnel destination IP addr */
    NPF_IPv4Address_t srcAddr; /* Tunnel source IP addr */
    NPF_uint8_t DSCP; /* DiffServ Code Point for hdr */
    NPF_IPv4UC_FwdTableHandle_t fibHandle; /*FIB for forwarding inner hdr*/
} NPF_IfTunnelIPv4_t;

/*
 * IP-in-IPv6 Tunnel Interface Attributes
 */
typedef struct {
    NPF_uint16_t MTU; /* Tunnel max transmission unit */
    NPF_uint8_t maxHops; /* Maximum hops in the tunnel */
    NPF_IPv6Address_t dstAddr; /* Tunnel destination IP addr */

```

```

        NPF_IPv6Address_t srcAddr;      /* Tunnel source IP addr */
        NPF_uint8_t      DSCP;          /* DiffServ Code Point for hdr */
        NPF_uint32_t     flowLabel;     /* IPv6 Flow Label for hdr */
        NPF_IPv6UC_FwdTableHandle_t fibHandle; /* FIB for forwarding inner hdr */
} NPF_IfTunnelIPv6_t;

/*
 * IPv4 Tunnel Addresses
 */
typedef struct {
    NPF_IPv4Address_t destAddr;        /* Tunnel destination address */
    NPF_IPv4Address_t srcAddr;        /* Tunnel source address */
} NPF_IfTunnelIPv4Addr_t;

/*
 * IPv6 Tunnel Addresses
 */
typedef struct {
    NPF_IPv6Address_t destAddr;        /* Tunnel destination address */
    NPF_IPv6Address_t srcAddr;        /* Tunnel source address */
} NPF_IfTunnelIPv6Addr_t;

/*
 * The Interface structure:
 */
typedef struct {
    NPF_IfID_t      ifID;              /* Interface ID */
    NPF_IfType_t    type;              /* Logical interface type */
    NPF_uint64_t    speed;             /* Speed in Kbits/second */
    NPF_IfAdminStatus_t adminStatus;  /* Administrative up/down */
    NPF_uint32_t    nChildren;         /* Number of child interfaces */
    NPF_uint32_t    *childIDs;        /* Array of child interface IDs */
    NPF_uint32_t    nParents;         /* Number of parent interfaces */
    NPF_uint32_t    *parentIDs;       /* Array of parent i/f IDs */
    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 */
        NPF_IfIPv6_t    IPv6_Attr;    /* IPv6 Interface attributes */
        NPF_IfIPv6inv4_t IPv6inv4_Attr; /* IPv6inv4 Tunnel attributes */
        NPF_IfTunnelIPv4_t TunnelIPv4_Attr; /* IP-in-IPv4 tunnel attributes */
        NPF_IfTunnelIPv6_t TunnelIPv6_Attr; /* IP-in-IPv6 tunnel attributes */
    } u;
} NPF_IfGeneric_t;

/*
 * Completion Callback Types
 */
typedef enum NPF_IfCallbackType {
    NPF_IF_CREATE = 1,
    NPF_IF_DELETE = 2,
    NPF_IF_BIND = 3,
    NPF_IF_UN_BIND = 4,
    NPF_IF_STATS_GET = 5,

```

```

NPF_IF_VCC_STATS_GET = 6,
NPF_IF_ATTR_SET = 7,
NPF_IF_CREATE_AND_SET = 8,
NPF_IF_ENABLE = 9,
NPF_IF_DISABLE = 10,
NPF_IF_OPER_STATUS_GET = 11,
NPF_IF_MTU_SET = 12,
NPF_IF_ATTR_GET = 13,

NPF_IF_LAN_SRC_ADDR_GET = 14,
NPF_IF_LAN_SRC_ADDR_SET = 15,
NPF_IF_LAN_ADDR_LIST_SET = 16,
NPF_IF_LAN_ADDR_LIST_ADD = 17,
NPF_IF_LAN_PROMISC_SET = 18,
NPF_IF_LAN_PROMISC_CLEAR = 19,
NPF_IF_LAN_FULL_DUPLEX_SET = 20,
NPF_IF_LAN_FULL_DUPLEX_CLEAR = 21,
NPF_IF_LAN_SPEED_SET = 22,
NPF_IF_LAN_FLOW_CONTROL_TX_ENABLE = 23,
NPF_IF_LAN_FLOW_CONTROL_TX_DISABLE = 24,
NPF_IF_LAN_FLOW_CONTROL_RX_ENABLE = 25,
NPF_IF_LAN_FLOW_CONTROL_RX_DISABLE = 26,

NPF_IF_IP_FWD_ENABLE = 27,
NPF_IF_IP_FWD_DISABLE = 28,

NPF_IF_IPV4ADDR_SET = 29,
NPF_IF_IPV4ADDR_CLEAR = 30,
NPF_IF_IPV4MTU_SET = 31,
NPF_IF_IPV4FIB_SET = 32,
NPF_IF_IPV4_UC_ADDR_SET = 33,
NPF_IF_IPV4_UC_ADDR_ADD = 34,
NPF_IF_IPV4_UC_ADDR_DELETE = 35,
NPF_IF_IPV4_MCAST_ADDR_SET = 36,
NPF_IF_IPV4_MCAST_ADDR_ADD = 37,

NPF_IF_IPV6_ADDR_SET = 38,
NPF_IF_IPV6_ADDR_ADD = 39,
NPF_IF_IPV6_ADDR_DELETE = 40,
NPF_IF_IPV6_FIB_SET = 41,

NPF_IF_ATM_VCC_SET = 42,
NPF_IF_ATM_VCC_BIND = 43,
NPF_IF_ATM_VCC_UN_BIND = 44,
NPF_IF_ATM_VCC_DELETE = 45,

NPF_IF_TUNNEL_HOPS_SET = 46,
NPF_IF_TUNNEL_DSCP_SET = 47,
NPF_IF_TUNNEL_IPV4_ADDR_SET = 48,
NPF_IF_TUNNEL_IPV6_ADDR_SET = 49,
NPF_IF_TUNNEL_IPV6_FLOW_LABEL_SET = 50
} NPF_IfCallbackType_t;

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      {
    NPF_IfHandle_t    ifHandle;    /* Asynchronous Response Structure */
    NPF_IfID_t        ifID;        /* Interface handle for this response */
    NPF_IfErrorType_t error;       /* Interface ID */
    union {           /* Error code for this response */
        /* 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_IfATM_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() */
        NPF_IfGeneric_t *attrs;    /* NPF_IfAttrGet() */
        NPF_IPv4Prefix_t v4prefix; /* NPF_IPv4UC_AddrAdd(), */
                                   /* Set(),Delete() */
        NPF_IPv4Address_t v4addr;  /* NPF_IPv4McastAddrAdd(), */
                                   /* Set() */
        NPF_IPv6Prefix_t v6prefix; /* NPF_IPv6AddrAdd(), Set(), */
                                   /* Delete() */
    } 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           n_resp; /* Number of responses in array */
    NPF_IfAsyncResponse_t *resp; /* Pointer to response structures*/
} NPF_IfCallbackData_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 prefix length */
#define NPF_IF_E_INVALID_PLEN ((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)

/* Invalid TTL value */
#define NPF_IF_E_INVALID_TTL ((NPF_IfErrorType_t) NPF_INTERFACES_BASE_ERR+27)

/* Invalid DSCP value */
#define NPF_IF_E_INVALID_DSCP ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+28)

/* Invalid IPv4 Multicast Address */
#define NPF_IF_E_INVALID_IPV4_MCAST_ADDR ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+29)

/* Parent/child binding not found */
#define NPF_IF_E_NO_SUCH_BINDING ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+30)

/* IP address does not exist on this interface */
#define NPF_IF_E_NO_SUCH_ADDRESS ((NPF_IfErrorType_t)
NPF_INTERFACES_BASE_ERR+31)

/*
 * 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;

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

NPF_error_t NPF_IfRegister(
    NPF_IN NPF_userContext_t userContext,
    NPF_IN NPF_IfCallbackFunc_t ifCallbackFunc,

```

```

    NPF_OUT NPF_callbackHandle_t *ifCallbackHandle);

NPF_error_t NPF_IfDeregister(
    NPF_IN NPF_callbackHandle_t ifCallbackHandle);

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

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

NPF_error_t NPF_IfEventDeregister(
    NPF_IN NPF_IfEventHandlerHandle_t ifEventHandlerHandle);

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,
    NPF_IN NPF_IfID_t *ifID);

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);

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 nbinds,
    NPF_IN NPF_IfBinding_t *if_bindArray);

NPF_error_t NPF_IfUnBind(
    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 nbinds,
    NPF_IN NPF_IfBinding_t *if_bindArray);

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);

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);

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);

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);

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);

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);

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);

NPF_error_t NPF_IfMTU_Set(
    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_MTU_Array);

NPF_error_t NPF_IfAttrGet(
    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_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);

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);

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);

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);

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);

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);

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);

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);

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);

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);

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);

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);

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);

NPF_error_t NPF_IfIP_FwdEnable(
    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_error_t NPF_IfIP_FwdDisable(
    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_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_IPv4Prefix_t       *if_IPv4AddrArray);

NPF_error_t NPF_IfIPv4AddrClear(
    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_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);

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_IPv4UC_FwdTableHandle_t  if_FIB_Handle);

NPF_error_t NPF_IfIPv4UC_AddrSet(
    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         nAddrs,
    NPF_IN NPF_IPv4Prefix_t     *if_IPv4AddrArray);

NPF_error_t NPF_IfIPv4UC_AddrAdd(
    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         nAddrs,
    NPF_IN NPF_IPv4Prefix_t     *if_IPv4AddrArray);

NPF_error_t NPF_IfIPv4UC_AddrAdd(
    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         nAddrs,
    NPF_IN NPF_IPv4Prefix_t     *if_IPv4AddrArray);

NPF_error_t NPF_IfIPv4McastAddrSet(
    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         nAddrs,
    NPF_IN NPF_IPv4Address_t     *mcAddrArray);

```

```

NPF_error_t NPF_IfIPv4McastAddrAdd(
    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           nAddrs,
    NPF_IN NPF_IPv4Address_t      *mcAddrArray);

NPF_error_t NPF_IfIPv6AddrSet(
    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         handle,
    NPF_IN NPF_uint32_t           nAddresses,
    NPF_IN NPF_IPv6Prefix_t       *if_IPv6AddrArray);

NPF_error_t NPF_IfIPv6AddrAdd(
    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         handle,
    NPF_IN NPF_uint32_t           nAddresses,
    NPF_IN NPF_IPv6Prefix_t       *if_IPv6AddrArray);

NPF_error_t NPF_IfIPv6AddrDelete(
    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           nAddrs,
    NPF_IN NPF_IPv6Prefix_t       *if_IPv6AddrArray);

NPF_error_t NPF_IfIPv6FIB_Set(
    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_IPv6UC_FwdTableHandle_t if_FIB_Handle);

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);

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);

```

```

NPF_error_t NPF_IfATM_VccUnBind(
    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);

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);

NPF_error_t NPF_IfTunnelHopsSet(
    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           nHandles,
    NPF_IN NPF_IfHandle_t         *if_Handle,
    NPF_IN NPF_uint8_t            hopcount);

NPF_error_t NPF_IfTunnelDSCP_Set(
    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_uint8_t            *if_DSCPArray);

NPF_error_t NPF_IfTunnelIPv4AddrSet(
    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           nHandles,
    NPF_IN NPF_IfHandle_t         *if_Handle,
    NPF_IN NPF_IfTunnelIPv4Addr_t *addrArray);

NPF_error_t NPF_IfTunnelIPv6AddrSet(
    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           nHandles,
    NPF_IN NPF_IfHandle_t         *if_Handle,
    NPF_IN NPF_IfTunnelIPv6Addr_t *addrArray);

NPF_error_t NPF_IfTunnelIPv6FlowLabelSet(
    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           nHandles,
    NPF_IN NPF_IfHandle_t         *if_Handle,
    NPF_IN NPF_uint32_t           *labelArray);

```

```
#ifdef __cplusplus  
}  
#endif  
  
#endif
```

**APPENDIX B LIST OF COMPANIES BELONGING TO NPF DURING APPROVAL PROCESS**

|                        |                 |                            |
|------------------------|-----------------|----------------------------|
| Agere Systems          | IBM             | Samsung Electronics        |
| Alcatel                | IDT             | Sandburst Corporation      |
| Altera                 | Intel           | Silicon & Software Systems |
| AMCC                   | IP Infusion     | Silicon Access             |
| Analog Devices         | Kawasaki LSI    | Sony Electronics           |
| Avici Systems          | LSI Logic       | STMicroelectronics         |
| Azanda Network Devices | Modelware       | Sun Microsystems           |
| Cypress Semiconductor  | Mosaid          | Teja Technologies          |
| Ericsson               | Motorola        | TranSwitch                 |
| Erlang Technologies    | NEC             | U4EA Group                 |
| EZ Chip                | NetLogic        | Xelerated                  |
| Flextronics            | Nokia           | Xilinx                     |
| Fujitsu Ltd.           | Paion Co., Ltd. | Zettacom                   |
| FutureSoft             | PMC Sierra      | ZTE                        |
| HCL Technologies       | RadiSys         |                            |
| Hi/fn                  |                 |                            |