Interface Management API Implementation Agreement (Specific Function Set for IPv6 Interfaces)

Revision 3.0

Editor: Erik B. Pedersen, Ericsson, erik.b.pedersen@ericsson.com
Karen Nielsen, Ericsson, karen.e.nielsen@ericsson.com

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

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

Table of Contents

Table of Figures

Glossary

API	Application Programming Interface
DAD	IPv6 Duplicate Address Detection
MTU	Maximum Transmission Unit
PDU	Protocol Data Unit

1 Revision History

Revision	Date	Reason for Changes
1.0	5/03/2004	Original text.
1.1	4/06/2004	Updated version
1.2	23/06/2004	Updated version including revised tunnel modeling

2 Introduction

This document defines the IPv6 interface types and functions under the NPF Interface Management API.

2.1 Assumptions and External Requirements

This API assumes the existence of the Interface Management API Core Function Set, and shares all the same assumptions and external requirements of that API.

2.2 Scope

This document is concerned only with definitions and functions supporting IPv6 interfaces (native IPv6, IPv6inv6 tunnels and IPv6inv4 tunnels) under NPF Interface Management.

2.3 Dependencies

This API is dependent on the Interface Management API Core Function Set and shares the same dependences as the Interface Management API Core Function Set.

The API is dependent on the IPv6 Unicast Forwarding Service API for the definition of the NPF_IPv6Prefix_t type.

2.3.1 Definition of attributes and functions provided by the IM API Core

2.3.1.1 NPF_IfMaxPDU_Size_Set

Using the NPF_IfMaxPDU_Size_Set on an IPv6 interface will set the MTU value. If Path MTU discovery (only tunnel type interfaces supported PMTU) is supported (and enabled) or the link has a dynamic MTU value, maxPDU value will be the maximum value allowed on an interface.

To read back the runtime MTU value, it is necessary to use an IPv6 specific function (NPF_IfIPv6RuntimeMTU_Get).

If the MaxPDU is left unspecified (zero), the default value on the particular link type should be chosen (par example 1500bytes on Ethernet).

2.3.1.2 Fwd_Enable/Disable

Enable/disabling forwarding on an IPv6 interface will start/end forwarding of packet out on the interface (egress direction). It is possible to receive local destined traffic as well as to send local originated traffic on a disabled interface.

`Interrelation with the IPv6 Router Advertisement Function`

While an IPv6 interface with forwarding enabled may not be an IPv6 advertising interface (i.e. an IPv6 interface with RA_mode on) an interface with forwarding mode disabled must not advertise it-self as a valid router to neighboring hosts. Consequently on an interface with RA_mode set and with forwarding mode disabled, the implementation must ensure that:

- the RA functions is stopped on the interface - or alternatively -
- that RAs are sent out with router lifetime set to 0 only.

2.4 IPv6 Interface Management Functions

The IPv6 specific part of the IM API provides an interface for configuring and managing IPv6 interfaces functions. The IPv6 interface functions considered are:

- IPv6 Address configuration and management, including functions governing IPv6 address auto configuration and duplicate address detection.
- Configuring and management of base IP functions such as runtime MTU and association of forwarding tables on IPv6 interfaces. IPv6 interface types considered include native IPv6 interfaces as well as various IPv6 tunnel interfaces.
- Configuration and management of Router Advertisement functions on IPv6 interfaces.

2.4.1 IPv6 link capability

The operation of various L3 applications on an IPv6 interface (e.g. routing protocols) as well as in particular IPv6 address configuration and link management operation depend on the capabilities of the underlying link-layer. For that purpose the IM API operates with the following three different link capabilities of IPv6 interfaces:

- **multicast -** an IPv6 link that supports a native mechanism at the link layer for sending a packet to all (i.e., broadcast) or a subset of all neighbors.
- **point-to-point** – an IPv6 link that connects exactly two IPv6 interfaces.
- **non-broadcast multi-access (NBMA)** – an IPv6 link via which more than two interfaces can attach, but that does not support a native form of multicast or broadcast

The link capability of an IPv6 interface is determined from the underlying link-layer and/or from the type of the IPv6 interface. The capability is deduced by the IM implementation and is required to be specified by the IM API implementation in a read-only interface attribute.

Examples: An IPv6 over Ethernet interface has multicast link capability. A configured IPv6-in-IPv4 tunnel interface or an IPv6-in-IPv6 tunnel interface with both IPv6 endpoint specified have point-to-point link capability. A 6to4 IPv6-in-IPv4 tunnel interface normally has NBMA link capability, but an implementation may also provide 6to4 interfaces with multicast link capability, e.g. by deploying IPv6 multicast emulation over IPv4 multicast capabilities of the underlying IPv4 layer.

`Multicast Enabled Interfaces`

In this document the term “multicast enabled interface” denotes an IPv6 interface of either multicast or point-to-point link capability.

The point-to-point case is a degenerated multicast scenario as a node always will be communication with one and only one receiver (the other end) over a point-to-point link.

A multicast enabled interface must, unless instantiated in pseudo-interface mode only (see 2.4.2.9) support setting of multicast receive address and must allow transmittal of packets addressed to multicast addresses.

2.4.2 IPv6 address configuration and management

The following subsections describe the base features of the IM API in relation to IPv6 address configuration and management as well as the base assumptions on the IM API usage model in this respect are described.

2.4.2.1 IPv6 unicast address assignment

`Preferred, Deprecated and Tentative address attribute assignment`

IPv6 addressing operates with three IPv6 unicast address attributes, which are relevant for IPv6 source address selection and interface packet processing: PREFERRED, DEPRECATED and TENTATIVE addresses:

- A TENTATIVE address is an address whose uniqueness on a link is being verified. A tentative address is not considered assigned to an interface in the true sense. An IPv6 interface generally

discards[1] received packets addressed to a tentative address, but accepts Neighbor Discovery packets related to Duplicate Address Detection for the tentative address.

- A PREFERRED address assigned on an interface is an address of which the uniqueness on the link has been verified and whose use by upper layer protocols is unrestricted.
- A DEPRECATED address assigned on an interface is an address whose use is discouraged, but not forbidden. A deprecated address should no longer be used as a source address in new communications, but packets sent from or to deprecated addresses are delivered as expected.

The IM API support for IPv6 interface address assignment of unicast addresses as either PREFERRED, DEPRECATED or TENTATIVE addresses.

The IM API specification makes no restrictions on the set of TENTATIVE addresses on IPv6 NBMA links. No assumptions, however, are made as to the possible operation of such addresses.

Address Assignment via IM API for Duplicate Address Detection

In order for the node to perform DAD for a unicast address on a multicast enabled interface, it is a prerequisite that the Solicited-Node Multicast Address associated with the unicast address as well as the All-Nodes Multicast Address have been assigned to the interface. The unicast address itself need not be assigned to the interface for the purposes of DAD, but it may tentatively be assigned to the interface for other purposes.

The IM API allow for instantiating of automated DAD validation of unicast addresses using the AUTODADVALIDATION flag attribute (see below). The following describes the envisaged IM API usage options when this feature isn't invoked.

The Solicited-Node Multicast Address may either be set explicitly by the IM API client using the IM API or it may be set implicitly by means of the IM API client setting the associated unicast address via the IM API (see Section 2.4.2.5).

The All-Nodes Multicast Address may be automatically generated as a result of interface initialization or it may have to be set explicitly by the IM API client (see Section 2.4.2.8).

If DAD succeeds, the unicast address must be set as PREFFERED address by the IM API client.

If DAD fails the Solicited-Node Multicast Address associated with the unicast address must be deleted as must the TENTATIVE unicast address (if set). The IM API client must do this.

2.4.2.2 IPv6 unicast address automatic DAD validation

The IM API allows for the management of automatic DAD validation of IPv6 unicast addresses on IPv6 interfaces. The feature is invoked by setting the AUTODADVALIDATION flag attribute on the address. Invoking this flag attribute on a unicast address means that it is the responsibility of the IM API implementation to perform DAD on the address and if successful to assign the address to the interface as a PREFERRED address.

Further it is the responsibility of the IM API implementation to perform DAD validation for the address anew whenever the interface comes up after an administratively or operational down period. In this case a deprecated address, if successfully validated, must be restored in DEPRECATED mode by the IM API implementation.

In case DAD fails during the process, it is the responsibility of the IM API client to remove the address.

Success and failure of the automated DAD validation process is reported to the IM API client via the NPF_IF_IPV6_ADDR_STATUS event.

DAD performed by the IM API implementation on a unicast address with the AUTODADVALIDATION flag set must be done in accordance with the IPv6 protocol DUPADDRDETECTTRANSMIT interface

[1] Actually a MUST discard, but optimistic DAD mechanisms emerging at the time of the writing of this document may allow for restrictive usage of the a TENTATIVE address prior to DAD completion.

attribute value. The IM API allow for the specification of this value in the `NPF_IfIPv6DAD_Transmits_t` interface attribute.

The AUTOVALIDATION flag attribute should only be set on addresses on which the `NPF_IfIPv6DAD_Transmits_t` is non-zero. The implementation may ignore the AUTOVALIDATION flag attribute on interfaces of `NPF_IfIPv6DAD_Transmits_t` equal to zero or it may choose to return an error message.

The support of `NPF_IfIPv6DAD_Transmits_t` values larger than zero is considered an optional feature.

The IM API makes no assumptions on the exact DAD functionality invoked by the set of `NPF_IfIPv6DAD_Transmits_t` values larger than zero on NBMA links.

For the purposes of the automated DAD validation function, the IM API implementation operates with the following address states:

- Valid – the address has been successfully DAD validated
- Probe – the address is undergoing DAD validation
- Invalid – DAD has failed for the address

The IM API client may at all times query the status of its addresses. The above interpretation applies to unicast addresses with the AUTODADVALIDATION flag attribute set. All other addresses are in this respect considered to be valid.

On an interface with `NPF_IfIPv6DAD_Transmits_t` equal to zero all addresses are valid.

2.4.2.3 Link-local address configuration

The explicit algorithm for generating the link identifier part of the link-local address of an IPv6 interface depends on the underlying layer.

The IM API specification makes no explicit assumptions as to whether link-local address assignment on a particular interface should be performed by the IM API client or by the IM API implementation.

The IM API specification defines the `NPF_IfIPv6LL_AddrMode_t` IPv6 interface attribute, which together with the `NPF_IfIPv6DAD_Transmits_t` interface attribute value governs link-local address assignment:

- On interfaces with `NPF_IfIPv6LL_AddrMode_t` disabled, a link-local address will have to be set from above as any other unicast address. A link-local address may be set with or without the AUTODADVALIDATION flag attribute.
- On interfaces with `NPF_IfIPv6LL_AddrMode_t` enabled, the IM API implementation will auto-generate, and optionally DAD validate depending on the `NPF_IfIPv6_DADTransmits_t` value, a link-local address once an interface has been initialized. Such an implementation does not allow for the specification of a link-local address from above. Upon having auto-generated, optionally DAD validated, a link-local address, the IM API implementation must generate the appropriate NPF_IF_IPV6_ADDR_STATUS event.

An IM API implementation may allow for specification of various combinations of the `NPF_IfIPv6DAD_Transmits_t` and the `NPF_IfIPv6LL_AddrMode_t` interface attributes. The exact range of combinations supported on each individual interface is likely to depend on the nature of the IPv6 interface and the underlying layer.

As a minimum normative requirement it is assumed that all interfaces will support the above described operation in accordance with set of `NPF_IfIPv6DAD_Transmits_t` equal to zero and `NPF_IfIPv6LL_AddrMode_t` in either (not both) disabled or enabled mode.

2.4.2.4 IPv6 Neighbor Discovery defense of assigned unicast addresses

It is assumed that any unicast address that is assigned to a multicast enabled interface via the IM API will be continuously defended against being configured on the interfaces of other nodes on the link by the implementations run of the IPv6 Neighbor Discovery protocol.

2.4.2.5 Solicited-Node Multicast Address Auto-generation

When setting a unicast/anycast IPv6 address on a multicast enabled interface it is required that the associated Solicited-Node Multicast Address automatically gets generated and assigned to the same interface by the IM API implementation.

No normative assumptions are made on the automatic generation of the Solicited-Node Multicast Address on IPv6 interfaces of NBMA link capability only.

Whether auto-generating a Solicited-Node Multicast Address on a multicast enabled or a NBMA link, the IM API implementation must always comply with the following:

- An auto-generated Solicited-Node Multicast Address shares faith with the unicast/anycast address from which it is generated. That is, it MUST be deleted by the implementation once the unicast/anycast address is deleted by the IM API client.
- The AUTOGENERATED flag attribute must be set by the IM API implementation on any auto-generated Solicited-Node Multicast Address.

A Solicited-Node Multicast Address which isn't auto-generated must be explicitly deleted by the IM API client.

It is assumed that an auto-generated Solicited-Node Multicast Address overwrite any Solicited-Node Multicast Address already explicitly set by the IM API client, that is the AUTOGENERATED flag gets set on the already existing Solicited-Node Multicast Address.

2.4.2.6 IPv6 anycast address assignment via IM API

IPv6 anycast addresses are syntactically indistinguishable from IPv6 unicast addresses. The IPv6 protocol operation on anycast interface addresses is different from the operation on unicast addresses. The IM API allow for the specification of an IPv6 unicast address as an anycast address by setting the ANYCAST flag attribute.

2.4.2.7 IPv6 multicast address assignment via IM API

IPv6 Multicast addresses are distinguishable from unicast (and anycast) addresses from the prefix. It is the responsibility of the IM API implementation to identify an assigned address as a multicast address.

2.4.2.8 Generic assumptions on existence and auto-generation of IPv6 addresses

IPv6 protocol operation on an IPv6 interface, IPv6 Neighbor Discovery protocol operation in particular, requires a minimal set of IPv6 addresses to be assigned on a multicast enabled interface.

`All IPv6 Interfaces:`

- Link-local address: An IPv6 Interface MUST have one and only one IPv6 link-local address assigned
- The All-Nodes Multicast Address: FF02::2.
- The Solicited-Node Multicast Address for its link-local address

`Plus in addition for IPv6 Router Interfaces:`

- The Subnet-Router Anycast Address
- The All-Routers Multicast Address

The IM API does not make any normative requirements on an IPv6 interface (whether multicast enabled or not) to have any of the above addresses in order to be operational (Section 2.4.2.9).

Further, no normative assumptions on the set of - or auto-generation of - any of the multicast or anycast addresses listed here are made.

It is however acknowledged that an implementation may choose to couple the existence of an auto-generated All-Nodes Multicast Address with interface initialization. Further it is acknowledged that an implementation may couple the auto-generation of the Subnet-Router Anycast Address and the All-Routers Multicast Address with the forwarding mode set on the interface.

Finally it is acknowledged that an implementation in accordance with IPv6 Protocol operation may choose to operational disable an IPv6 interface if DAD fails on an attempted auto-configured Link-local address.

The following normative requirement apply to auto-generation of the All-Nodes Multicast Address, the Subnet-Router Anycast Address and the All-Routers Multicast Address as well as to the auto-generation of addresses in general:

- Whenever an address is automatically assigned to an interface by the IM API implementation it must be assigned the AUTOGENERATED flag attribute.

2.4.2.9 IPv6 Pseudo-interfaces

The IM API allow for initialization and operation of IPv6 interfaces for packets transmit, receive and forwarding functions on which no addresses are assigned. In particular, the IM API makes no coupling in between the operational status of an IPv6 interface to the assignment of (valid) addresses.

An implementation may support certain types of interfaces in pseudo-interface mode only, i.e. it may not support the set of addresses on certain interfaces, e.g., Ip-in-Ip tunnel interfaces.

An implementation must support the set of addresses on IPv6 interfaces of NPF_IF_TYPE_IPV6 type.

2.4.3 IPv6 RA function

As an optional feature the IM API provides functions for configuring and managing of the IPv6 Router Advertisement function.

The functions are intended for use on multicast enabled IPv6 interfaces. No assumptions are made on the possible usage of these functions on IPv6 interfaces of NBMA link capability only.

2.4.4 IPv6-in-IP tunnel interface model

The NPF Interface Management API defines an IP-in-IP tunnel interface as an interface that is tightly coupled with a parent IP interface (which could be another tunnel, in case of nested tunnels). The tunnel interface has a primary role of encapsulating/decapsulating tunnel packets, while the tightly coupled parent IP interface has the primary role of packet delivery upwards or downwards to the next layer. The encapsulation/decapsulation and packet delivery are performed on the “outer header”, or “tunnel header”. Before encapsulation or after decapsulation the packet is exposing an ‘inner header” and “payload”. This is illustrated in Figure 1.

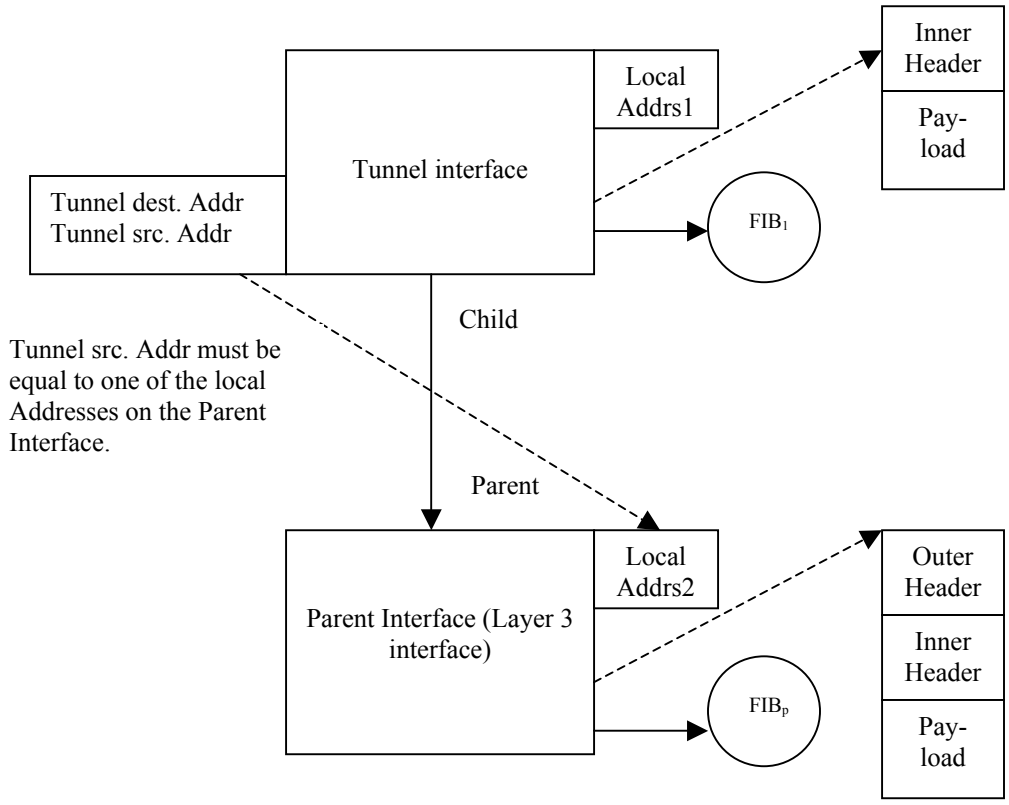

Figure 1 IM API tunnel model

The relation between the IP-in-IP tunnel interface and its parent IP interface is a normal parent-child relation (with the tunnel as child) with the additional requirement that the tunnel source addresses on which the tunnel interface operates (i.e. the address which is set as IP source address in the encapsulating tunnel header) should be set on the parent IP interface.

In the case of IPv6 tunnel types, the parent interface can be either an IPv6 or IPv4 interface.

An IP interface can have zero or more associated child tunnel interfaces, each identified by the combination of source and destination IP address (on the outer header) and protocol number. For "open-ended" tunnel interfaces, as e.g. a 6to4 tunnel interfaces, it is only the source IP address and protocol number that identify the tunnel interface.

The IM API makes no assumption on the order of interface creation; in particular it is possible to create the tunnel interface without having the parent interface ready.

2.4.4.1 IP-in-IP tunnel interface functionality

The tunnel interface can be instantiated as a full-fledged IP interface with IP addresses of its own (local addr's in Figure 1). A local IP address of the tunnel interface appears in the inner header of packets originated locally and going out to the tunnel interface.

IP-in-IP tunnel tunnel interfaces may also be instantiated as pseudo-interfaces only.

As any standard IP interface an IP-in-IP tunnel interface has a FIB assigned (used for forwarding of decapsulated packets). This FIB may be the same as the one used for the IP parent interface for IPv6inv6 tunnel interfaces.

2.4.4.2 Incoming Packet Flow

An incoming tunnelled packet arriving at an ordinary IPv6 or IPv4 interface with a destination address equal to one of the nodes interface addresses will be consumed by the node and the appropriate child

tunnel interface will be identified based on the outer header addresses and protocol number (as described above).

2.4.4.3 Transmit Packet flow

An outgoing packet arrives on an IP-in-IP tunnel interface as a result of a forwarding decision, i.e. the IP-in-IP tunnel interface is pointed out as the egress interface in the standard manner. After encapsulation the encapsulated packets is handed to the underlying parent IP interface for transmission. This may or may not involve an additional forwarding decision being taken on the encapsulating IP packet using the FIB associated with the parent IP interface.

3 Data Types

3.1 IPv6 interface Management API Types

3.1.1 Interface Type Code: NPF_IfType_t

The type code (value used in the **NPF_IfType_t** variable) for IPv6 interfaces is 6, for IPv6inIPv4 tunnel interfaces is 7 and for IPv6 in IPv6 is 9:

```
#define NPF_IF_TYPE_IPV6            7           /* IPv6 logical interface */
#define NPF_IF_TYPE_TUNNEL_IPV6INV4 8           /* IP6-in-IPV4 tunnel */
#define NPF_IF_TYPE_TUNNEL_IPV6     9           /* IP6-in-IPV6 tunnel */
```

3.1.2 IPv6 link capability: NPF_IfIPv6LinkCapability_t

```
/*
 * IPv6 link capability attribute
 */
typedef enum {
      NPF_IF_IPV6_MULTICAST        = 1,   /* multicast */
      NPF_IF_IPV6_POINT_TO_POINT   = 2,   /* point-to-point */
      NPF_IF_IPV6_NBMA             = 3    /* NBMA */
 }NPF_IfIPv6LinkCapability_t;
```

3.1.3 DAD transmission counter: NPF_IfIPv6DAD_Transmits_t

```
/*
 * Duplicate Address Detection transmit counter
 */
typedef NPF_uint8_t     NPF_IfIPv6DAD_Transmits_t;
```

- The IPv6 protocol constant DupAddrDetectTransmits of the IPv6 interface. Valid values are 0,1,2 and 3 (MAX_UNICAST_SOLICIT IPv6 protocol constant). A value of 0 mutes the DAD function on the interface.
- Support for set of NPF_IfIPv6_DADTransmits_t different from 0 is **optional**.

3.1.4 IPv6 Address structure: NPF_IfIPv6InterfaceAddr_t

```
/*
 * IPv6 address structure
 */
struct NPF_IPv6Address {
      union {
            NPF_uchar8_t      addr8[16];
            NPF_uint32_t      addr32[4];
      }u;
};
```

- The typedef declaration will appear in **npf_if_core.h**

```
/*
 * IPv6 Interface address structure
 */
typedef struct {
      NPF_IPv6Address_t      IPv6Addr;     /* IPv6 address */
      NPF_uchar8_t           IPv6Plen;     /* Prefix length in bits (0-128)*/
```

```
      NPF_IfIPv6AddrFlags_t  IPv6AddrFlags;/* IPv6 address flags */
} NPF_IfIPv6InterfaceAddr_t;

/*
 * IPv6 Address Flag
 */
typedef NPF_uchar8_t     NPF_IfIPv6AddrFlags_t

#define NPF_IF_IPV6_ADDR_FLAGS_NONE             0x00
#define NPF_IF_IPV6_ADDR_PREFERRED              0x01
#define NPF_IF_IPV6_ADDR_DEPRECATED             0x02
#define NPF_IF_IPV6_ADDR_ANYCAST                0x04
#define NPF_IF_IPV6_ADDR_TENTATIVE              0x08
#define NPF_IF_IPV6_ADDR_PUBLIC                 0x10
#define NPF_IF_IPV6_ADDR_TEMPORARY              0x20
#define NPF_IF_IPV6_ADDR_AUTODADVALIDATION      0x40
#define NPF_IF_IPV6_ADDR_AUTOGENERATED          0x80
```

- PREFERRED, DEPRECATED and TENTATIVE address flags are used in the source address selection process (for local originated traffic). The flags are only valid for unicast addresses which are not anycast addresses.
- The ANYCAST flag is only valid for anycast addresses and MUST be set for such addresses.
- The AUTODADVALIDATION flag indicates that the implementation should perform IPv6 auto DAD validation on the address in accordance with the NPF_IfIPv6_DADTransmits_t value set on the interface. The flag is only valid for unicast addresses that are not anycast addresses. The flag can be ignored on interfaces for which the NPF_IfIPv6_DADTransmits_t is set to zero.
- PUBLIC and TEMPORARY is used as defined in RFC3041.
- AUTOGENERATED is a read-only flag used when retrieving settings. It indicates that a specific address is auto-generated by the implementation. The flag is valid for all address types.

3.1.5 IPv6 Link Local Address Generation Mode: NPF_IfIPv6LL_AddrMode_t

```
/*
 * IPv6 Link Local address generation mode
 */

 typedef enum {
       NPF_IfIPv6_LL_ADDR_GENERATION_ENABLE   = 1, /*enable*/
       NPF_IfIPv6_LL_ADDR_GENERATION_DISABLE  = 2  /*disable*/
 }NPF_IfIPv6LL_AddrMode_t;
```

- NPF_IfIPv6LL_AddrMode_t defines whether link local addresses are automatically generated below the IM API (and not by the API user).

3.1.6 IPv6 Router Advertisements Mode: NPF_IfIPv6RA_Mode_t

```
/*
 * IPv6 RA (Router Advertisement) mode
 */
 typedef enum {
      NPF_IF_IPV6_RA_MODE_ENABLE   = 1,   /* Enable RA */
```

```
      NPF_IF_IPV6_RA_MODE_DISABLE   = 2   /* Disable RA */
 }NPF_IfIPv6RA_Mode_t;
```

3.1.7 IPv6 Interface attributes: NPF_IfIPv6_t

A forward reference to this structure must be added to the **NPF_IfGeneric_t** structure in **npf_if_core.h** if IPv6 interfaces are supported. Before the declaration of **NPF_IfGeneric_t**, the following must appear:

```
typedef struct NPF_IfIPv6        NPF_IfIPv6_t;
```

And the following must appear inside the union within the **NPF_IfGeneric_t** structure:

```
NPF_IfIPv6_t *IPv6_Attr; /*IPv6 interface attributes*/

/*
 * IPv6 Interface Attributes
 */
struct NPF_IfIPv6 {
   NPF_uint32_t                nAddr;     /*Number of IPv6 addresses*/
   NPF_IfIPv6InterfaceAddr_t   *Addrs;    /*Array of IPv6 addresses*/
   NPF_IPv6UC_FwdTableHandle_t FIB_Handle; /* IPv6 Forwarding info base*/
   NPF_IFIPv6LL_AddrMode_t     LL_mode; /*Enable autogen of Link Local addr*/
   NPF_IfIPv6DAD_Transmits_t   DAD_Transmits;/*nr of times DAD is performed*/
   NPF_IfIPv6RA_mode_t         IfIPv6RA_Mode;/* IPv6 RA on/off */
   NPF_IfIPv6LinkCapability_t  LinkCapability;  /*IPv6 link capability */
};
```

- Multicast and unicast addresses are specified in the same array. The IM API implementation (and client on attribute readback) must be able to split based on prefix.
- Upon readback the list of addresses also include addresses generated by Autoconfiguration. For autogenerated addresses and addresses with DADVALIDATION on, it is necessary that the client use the **NPF_IfIPv6AddrStatus_Get** function call to read the address state (valid, probing or invalid).
- RA mode is default disabled.
- DAD_Transmits specify the number of times Neighbor Solicitation messages is sent on this interface when performing DAD. A value of zero disables DAD on this interface. If DAD validation is not supported, setting a non-zero value returns an error.
- The LinkCapability attribute is read-only.

3.1.8 IPv6 address status readout: NPF_IfIPv6AddrStatus_t and NPF_IfIPv6AddrStatusEventData_t

```
/*
 * IPv6 address state
 */
typedef enum {
      NPF_IPv6_ADDR_VALID = 1,
      NPF_IPv6_ADDR_PROBING = 2,
      NPF_IPv6_ADDR_INVALID = 3
}NPF_IPv6AddrState_t;
```

- Invalid means that automated DAD has failed and the IM API client should remove the address.

```
/*
 * IPv6 address status structure for Address Status Event handling
 */
struct NPF_IfIPv6AddrStatusEventData {
      NPF_IPv6Prefix_t        addr;
      NPF_IPv6AddrState_t     addrState;
};
```

- The typedef declaration will appear in **npf_if_core.h**

```
/*
 * IPv6 address status structure used when reading back address state
 */
struct NPF_IfIPv6AddrStatus {
      NPF_IfIPv6InterfaceAddr_t     addr;
      NPF_IPv6AddrState_t           addrState;
};
```

- The typedef declaration will appear in **npf_if_core.h**

3.1.9 Tunnel Interface Path MTU config: NPF_IfIPv6_TunnelMTU_t

```
/*
 * Tunnel interface MTU mode
 */
 typedef enum {
      NPF_IFIPv6_TUNNEL_PMTU_DISABLED = 1,
      NPF_IFIPv6_TUNNEL_PMTU_ENABLED = 2
 } NPF_IfIPv6TunnelMTU_t;
```

3.1.10 Using DSCP from inner header: NPF_IfIPv6_InnerDSCP_t

```
/*
 * Tunnel DSCP mode
 */
 typedef enum {
      NPF_IFIPv6_INNER_DSCP_DISABLED = 1,
      NPF_IFIPv6_INNER_DSCP_ENABLED  = 2
 } NPF_IfIPv6InnerDSCP_t;
```

3.1.11 IPv6 in IPv4 Tunnel Attributes: NPF_IfIPv6TunnelV4_t

A forward reference to this structure must be added to the **NPF_IfGeneric_t** structure in **npf_if_core.h** if IPv6inv4 tunnel interfaces are supported. Before the declaration of **NPF_IfGeneric_t**, the following must appear:

```
typedef struct NPF_IfIPv6TunnelV4        NPF_IfIPv6TunnelV4_t;
```

And the following must appear inside the union within the **NPF_IfGeneric_t** structure:

```
NPF_IfIPv6TunnelV4_t *IPv6inv4_Attr; /*IPv6inv4 tunnel interface attributes*/

/*
 * IPv6 in IPv4 Tunnel types
```

```
 */
 typedef enum {
      NPF_IF_V6INV4_CONFIGURED = 1, /* Configured Tunnel */
      NPF_IF_V6INV4_6TO4 = 2        /* 6to4 Tunnel*/
 } NPF_IfIPv6TunnelV4Type_t;
```

- NPF_IF_V6INV4_AUTOMATIC present in the current revision has been deprecated.

```
/*
 * IPv6-in-IPv4 Tunnel Interface attributes
 */
 struct NPF_IfIPv6TunnelV4 {
      NPF_IfIPv6TunnelV4Type_t    tunnelType; /* Type of v6inv4 Tunnel */
      NPF_IfIPv4Address_t         IPv4DstAddr; /*Destination IPv4 address*/
      NPF_IfIPv4Address_t         IPv4SrcAddr; /* Source IPv4 address */
      NPF_uint8_t                 IPv4TTL; /* TTL in IPv4 header*/
      NPF_uint8_t                 IPv4DSCP; /*DSCP in IPv4 header*/
      NPF_IPv6UC_FwdTableHandle_t FIB_Handle;/*IPv6 Fwd info base*/
      NPF_IfIPv6RA_mode_t         ifIPv6RA_Mode;  /* IPv6 RA on/off */
      NPF_IfIPv6LL_AddrMode_t     LL_mode; /*Enable autogen of LL addr*/
      NPF_IfIPv6DAD_Transmits_t   DAD_Transmits; /*nr of DAD req*/
      NPF_IfIPv6InnerDSCP_t       IPv6DSCP;/*Use DSCP from IPv6 pck*/
      NPF_IfIPv6TunnelMTU_t       MTU_MODE; /* PMTU on/off */
      NPF_uint32_t                nAddr;  /*Number of IPv6 addresses*/
      NPF_IfIPv6InterfaceAddr_t   *Addrs; /*Array of IPv6 addresses*/
      NPF_IfIPv6LinkCapability_t  LinkCapability;  /*IPv6 link capability */
};
```

- For a tunnel interface of NPF_IF_IPV6INV4_CONFIGURED type, both the IPv4 destination and source address must be specified. For a tunnel interface of NPF_IF_IPV6INV4_6TO4 type, the IPv4 destination may be left unspecified
- The usage of the IPv4DstAddr for 6to4 is left to the implementation, it could however be used to point out a 6to4 Relay Router (for native IPv6 traffic) if not the destination address may be given as the unspecified address 0.0.0.0.
- If Link Local mode is on, the link local addr will be auto-generated
- Path MTU for 6to4 is not supported, ie. it will be ignored.
- If IPv4TTL is 0, the value from the inner packet (the IPv6 packet) is used (as specified in rfc2667).
- If DSCP is taken from the encapsulated packet (the inner packet) as specified by **IPv6DSCP** the value of **DSCP** is ignored.
- DADTransmits specify the number of times Neighbor Solicitation messages is sent on this interface when performing DAD. A value of zero disables DAD on this interface. If DAD validation is not supported, setting a non-zero value returns an error.
- Default values are as follows:
    - MTU_MODE = off
    - IPv6DSCP = do not use inner DSCP
    - IfIPv6RA_Mode = off
    - LL_mode = off
    - DAD_Transmits = 0
    - nAddr = 0
    - *Addrs = ignored as the number of addresses is zero

3.1.12 IPv6-in-IPv6 Tunnel Interface Attributes: NPF_IfIPv6TunnelV6_t

A forward reference to this structure must be added to the **NPF_IfGeneric_t** structure in **npf_if_core.h** if IPv6inv6 tunnel interfaces are supported. Before the declaration of **NPF_IfGeneric_t**, the following must appear:

```
typedef struct NPF_IfIPv6TunnelV6        NPF_IfIPv6TunnelV6_t;
```

And the following must appear inside the union within the **NPF_IfGeneric_t** structure:

```
NPF_IfIPv6TunnelV6_t *IPv6TunnelIPv6_Attr; /*IPv6inv6 tunnel interface
attributes*/

/*
 * IPv6-in-IPv6 (rfc2473 compliant) Tunnel Interface Attributes
 */
struct NPF_IfIPv6TunnelV6 {
      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-default 0 */
      NPF_IPv6UC_FwdTableHandle_t   FIB_Handle;/*FIB for inner hdr*/
      NPF_IfIPv6TunnelMTU_t         MTU_MODE;/* PMTU on/off */
      NPF_IfIPv6InnerDSCP_t         IPv6DSCP;/*use inner packet DSCP*/
      NPF_IFIPv6LL_AddrMode_t       LL_mode; /*Enable autogen of LL addr*/
      NPF_IfIPv6RA_mode_t           ifIPv6RA_Mode;  /* IPv6 RA on/off */
      NPF_IfIPv6DAD_Transmits_t     DAD_Transmits; /*nr of DAD req*/
      NPF_uint32_t                  nAddr;  /*Number of IPv6 addresses*/
      NPF_IPv6InterfaceAddr_t       *Addrs; /*Array of IPv6 addresses*/
      NPF_IfIPv6LinkCapability_t    LinkCapability; /*IPv6 link capability */
};
```

- If DSCP is taken from the encapsulated packet (the inner packet) as specified by **IPv6DSCP** the value of **DSCP** is ignored.
- If **maxHops** is 0, the value is taken from the inner packet (as specified in rfc2667)
- Default values are as follows:
  - MTU_MODE = off
  - IPv6DSCP = do not use inner DSCP
  - IfIPv6RA_Mode = off
  - DAD_Transmits = 0
  - nAddr = 0
  - *Addrs = ignored as the number of addresses is zero

3.1.13 Tunnel Address Structures: NPF_IfIPv6TunnelV4Addr_t and NPF_IfIPv6TunnelV6Addr_t

```
/*
 * IPv4 Tunnel Addresses
 */
typedef struct {
      NPF_IPv4Address_t destAddr; /* Tunnel destination address */
      NPF_IPv4Address_t srcAddr; /* Tunnel source address*/
} NPF_IfIPv6TunnelIPv4Addr_t;
```

```
/*
 * IPv6 Tunnel Addresses
 */
typedef struct {
      NPF_IPv6Address_t destAddr; /* Tunnel destination address */
      NPF_IPv6Address_t srcAddr; /* Tunnel source address*/
} NPF_IfIPv6TunnelIPv6Addr_t;
```

3.1.14 IPv6 RA Prefix attributes: NPF_IfIPv6PrefixAdv_t

```
/*
 * IPv6 Prefix Advertisement
 */
typedef struct {
      NPF_IPv6Prefix_t        adv_Prefix;      /* Prefix to be advertised */
      NPF_IfIPv6PrefixFlags_t prefix_Flags;    /* Prefix flags */
      NPF_IfIPv6PrefixLfts_t  prefix_Lifetimes;/* Prefix lifetime, preferred
                                                  and valid */
} NPF_IfIPv6PrefixAdv_t;

/*
 * IPv6 Prefix Flags
 */
typedef NPF_uint8_t      NPF_IfIPv6PrefixFlags_t

#define NPF_IF_IPV6_PREFIX_ONLINK          0x01
#define NPF_IF_IPV6_PREFIX_AUTO            0x02
#define NPF_IF_IPV6_VALIDLIFETIME_DECR     0x04
#define NPF_IF_IPV6_PREFFEREDLIFETIME_DECR 0x08
```

NPF_IF_IPV6_VALIDLIFETIME_DECR and NPF_IF_IPV6_PREFFEREDLIFETIME_DECR is used to control whether the lifetime counters should decrement or not.

```
/*
 * IPv6 Prefix Lifetimes
 */
typedef struct{
        NPF_uint32_t     IPv6ValidLft;             /*Time in seconds */
        NPF_uint16_t     IPv6PreferredLft;         /*Time in seconds */
} NPF_IfIPv6PrefixLfts_t;
```

3.1.15 IPv6 RA settings: NPF_IfIPv6RA_Settings_t

```
/*
 * IPv6 RA parameters
 */
struct NPF_IfIPv6RA_Params {
    NPF_IfIPv6RA_Flags_t ifIPv6RAFlags;      /* IPv6 RA Flags */
    NPF_IFIPv6RtrPref_t  routerPreference;   /* Router preference */
    NPF_uint32_t         maxRtrAdvInterval;  /* Time in msec */
    NPF_uint32_t         minRtrAdvInterval;  /* Time in msec */
    NPF_uint32_t         advReachableTime;   /* Time in msec */
    NPF_uint32_t         advRetransTimer;    /* Time in msec */
```

```
    NPF_uint8_t          advCurHopLimit;     /* Current Hop Limit value*/
    NPF_uint16_t         advDefaultLft;      /* Time in sec */
    NPF_uint32_t         minDelayBetweenRAs; /* for rate limiting – default
                                                3sec */
};
```

- The typedef declaration will appear in **npf_if_core.h**

```
/*
 * IPv6 RA Flags
 */
typedef NPF_uint8_t      NPF_IfIPv6RA_Flags_t
#define NPF_IF_IPV6_RA_MANAGED                          0x01
#define NPF_IF_IPV6_RA_OTHER_CONFIG                     0x02
#define NPF_IF_IPV6_RA_ADV_MTU                          0x04
#define NPF_IF_IPV6_RA_ADV_MAX_RA_ADV_INTERVAL          0x08
```

NPF_IF_IPV6_RA_ADV_MAX_RA_ADV_INTERVAL is for access router usage for networks with mobile nodes. It allows mobile nodes to detect whether it has moved to another access point (by checking if it has received RA messages within a number of consecutive intervals.

```
/*
 * Advertised router preference (High, medium or low)
 */
typedef enum {
      NPF_IF_IPV6_RTR_PREF_HIGH = 1,
      NPF_IF_IPV6_RTR_PREF_MEDIUM = 2,
      NPF_IF_IPV6_RTR_PREF_LOW = 3
} NPF_IFIPv6RtrPref_t;

/*
 * IPv6 RA settings (prefixes and RA parameters)
 */
struct NPF_IfIPv6RA_Settings {
      NPF_uint32_t                  nPrefix;
      NPF_IfIPv6PrefixAdv_t         *Prefix;
      NPF_IfIPv6RA_Params_t         RA_Params;
};
```

3.1.16 IPv6 RA inconsistency: NPF_IfIPv6RA_Inconsistency_t

```
/*
 * IPv6 RA inconsistency check – values extracted from received packet
 */
struct NPF_IfIPv6RA_Inconsistency {
      NPF_IPv6Prefix_t     RecvLL_addr; /*Link local addr of rec. RA msg*/
      NPF_IfIPv6RA_Flags_t RA_Flags; /* Only managed and other relevant */
      NPF_uint32_t         AdvReachableTime;   /* Time in msec */
      NPF_uint32_t         AdvRetransTimer;    /* Time in msec */
      NPF_uint8_t          AdvCurHopLimit;     /* Hop Limit value*/
      NPF_uint16_t         MTU;           /*value from MTU options field */
      NPF_uint32           nPrefix;       /*number of prefixes */
      NPF_IfIPv6PrefixAdv_t; *recvPrefix; /*adv. prefixes from other router*/
};
```

- The typedef declaration will appear in **npf_if_core.h**

3.2 Data Structures for Completion Callbacks

3.2.1 Completion Callback Type Codes: NPF_IfCallbackType_t

```
/*
 *    Completion Callback Types for IPv6 interfaces
 */
#define NPF_IF_IPV6_ADDR_SET                    ((NPF_IF_TYPE_IPV6<<16)+1)
#define NPF_IF_IPV6_ADDR_ADD                    ((NPF_IF_TYPE_IPV6<<16)+2)
#define NPF_IF_IPV6_ADDR_DELETE                 ((NPF_IF_TYPE_IPV6<<16)+3)
#define NPF_IF_IPV6_FIB_SET                     ((NPF_IF_TYPE_IPV6<<16)+4)
#define NPF_IF_IPV6_ADDR_STATUS_GET             ((NPF_IF_TYPE_IPV6<<16)+5)
#define NPF_IF_IPV6_TUNNEL_V6_ADDR_SET          ((NPF_IF_TYPE_IPV6<<16)+6)
#define NPF_IF_IPV6_TUNNEL_FLOW_LABEL_SET       ((NPF_IF_TYPE_IPV6<<16)+7)
#define NPF_IF_IPV6_TUNNEL_HOPS_SET             ((NPF_IF_TYPE_IPV6<<16)+8)
#define NPF_IF_IPV6_RUNTIME_MTU_GET             ((NPF_IF_TYPE_IPV6<<16)+9)
#define NPF_IF_IPV6_TUNNEL_V4_ADDR_SET          ((NPF_IF_TYPE_IPV6<<16)+10)
#define NPF_IF_IPV6_RA_PREFIX_ADD               ((NPF_IF_TYPE_IPV6<<16)+11)
#define NPF_IF_IPV6_RA_PREFIX_CLEAR             ((NPF_IF_TYPE_IPV6<<16)+12)
#define NPF_IF_IPV6_RA_PARAMS_SET               ((NPF_IF_TYPE_IPV6<<16)+13)
#define NPF_IF_IPV6_RA_SETTING_GET              ((NPF_IF_TYPE_IPV6<<16)+14)
```

3.2.2 Asynchronous Response Details

Asynchronous responses are returned by the mechanism defined in Interface Management API Core Function Set. The NPF_IfAsyncResponse_t structure defined there contains a (void *) pointer; in the case of IPv6 API function calls, this pointer can point to a structure containing IPv6-specific or generic information. The following table summarizes what is returned by each of the functions defined in this Implementation Agreement.

Function Name	Type Code	Structure Returned
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 (null pointer)
NPF_IfIPv6AddrStatusGet	NPF_IF_IPV6_ADDR_STATUS_GET	NPF_IfIPv6AddrStatus_t *
NPF_IfIPv6TunnelV6AddrSet	NPF_IF_IPV6_TUNNEL_V6_ADDR_SET	Unused (null pointer)
NPF_IfIPv6TunnelFlowLabelSet	NPF_IF_IPV6_TUNNEL_FLOW_LABEL_SET	Unused (null pointer)
NPF_IfIPv6TunnelHopsSet	NPF_IF_IPV6_TUNNEL_HOPS_SET	Unused (null pointer)
NPF_IfIPv6RuntimeMTU_Get	NPF_IF_IPV6_RUNTIME_MTU_GET	NPF_uint16_t *
NPF_IfIPv6TunnelIV4AddrSet	NPF_IF_IPV6_TUNNEL_V4_ADDR_SET	Unused (null pointer)
NPF_IfIPv6RA_PrefixAdd	NPF_IF_IPV6_RA_PREFIX_ADD	NPF_IPv6Prefix_t *
NPF_IfIPv6RA_PrefixClear	NPF_IF_IPV6_RA_PREFIX_CLEAR	NPF_IPv6Prefix_t *
NPF_IfIPv6RA_ParamsSet	NPF_IF_IPV6_RA_PARAMS_SET	NPF_IfIPv6RA_Params_t *
NPF_IfIPv6RA_SettingsGet	NPF_IF_IPV6_RA_SETTING_GET	NPF_IfIPv6RA_Settings_t *

Before declaration of the **NPF_IfAsyncResponse_t** structure the following must appear:

```
typedef struct NPF_IfIPv6AddrStatus       NPF_IfIPv6AddrStatus_t;
typedef struct NPF_IfIPv6RA_Params        NPF_IfIPv6RA_Params_t;
```

```
typedef struct NPF_IfIPv6RA_Settings      NPF_IfIPv6RA_Settings_t;
```

The following must be included in the union within the **NPF_IfAsyncResponse_t** structure:

```
      NPF_IPv6Prefix_t        *v6prefix;  /* NPF_IfIPv6AddrSet(), Add(),
                                             Delete(), RA_PrefixAdd() Clear()
                                          */

      NPF_uint16_t            *MTU_Value  /* NPF_IfIPv6RuntimeMTU_Get() */

      NPF_IfIPv6AddrStatus_t  *addrStatus /* NPF_IfIPv6AddrStatusGet() */

      NPF_IfIPv6RA_Params_t   *RA_ParamsSet/* NPF_IfRA_ParamsSet() */

      NPF_IfIPv6RA_Settings_t *RA_Settings /* NPF_IfIPv6RA_SettingsGet */
```

3.3 Error Codes

The following codes are used as values of **NPF_IfErrorType_t**.

```
/*
 *    Error codes returned in function callbacks and return
 *    values for function invocations used by IPv6 interface functions
 */

#define NPF_IF_E_IPV6_CODE(code) (0x10000+(NPF_IF_TYPE_IPV6<<8)+(code))

/* Invalid IP address */
#define NPF_IF_IPV6_E_INVALID_IPADDR      NPF_IF_E_IPV6_CODE(1)

/* Invalid IP prefix length */
#define NPF_IF_IPV6_E_INVALID_PLEN        NPF_IF_E_IPV6_CODE(2)

/* Invalid FIB handle */
#define NPF_IF_IPV6_E_INVALID_FIB_HANDLE  NPF_IF_E_IPV6_CODE(3)

/* Invalid TTL value */
#define NPF_IF_IPV6_E_INVALID_TTL         NPF_IF_E_IPV6_CODE(4)

/* Invalid DSCP value */
#define NPF_IF_IPV6_E_INVALID_DSCP        NPF_IF_E_IPV6_CODE(5)

/* IP address does not exist on this interface */
#define NPF_IF_IPV6_E_NO_SUCH_ADDRESS     NPF_IF_E_IPV6_CODE(6)

/* Error in parameters*/
#define NPF_IF_IPV6_E_INVALID_PARAM       NPF_IF_E_IPV6_CODE(7)

/* Optional feature not supported */
#define NPF_IF_IPV6_E_NO_SUCH_FEATURE     NPF_IF_E_IPV6_CODE(8)
```

3.3.1 Synchronous Error codes

In addition to the list of error code in section 3.3, codes defined in Software Convention (ref 1) may be returned, note in particular NPF_F_FUNCTION_NOT_SUPPORTED for optional functions not supported by the implementation.

3.4 Data Structures for Event Notifications

3.4.1 Event Types for IPv6

```
/*
 * Event types
 */
/* Duplicated Address Detection Event */
#define NPF_IF_IPV6_ADDR_STATUS ((NPF_IF_TYPE_IPV6<<16)+1)
/* Runtime MTU changed */
#define NPF_IF_IPV6_MTU_CHANGE ((NPF_IF_TYPE_IPV6<<16)+2)
/* RA inconsistency detected */
#define NPF_IF_IPV6_RA_INCONSISTENCY ((NPF_IF_TYPE_IPV6<<16)+3)
/* IPv6UC FIB changed */
#define NPF_IF_IPV6_FIB_CHANGE ((NPF_IF_TYPE_IPV6<<16)+4)
```

3.4.1.1 NPF_IF_IPV6_ADDR_STATUS Event

This event is generated as the result of auto-generation, optionally DAD validation depending on the `NPF_IfIPv6DAD_Transmits_t` value set on the interface, of a unicast address.

3.4.1.2 NPF_IF_IPV6_MTU_CHANGE Event

This event is generated whenever the runtime MTU on the interface changes.

3.4.1.3 NPF_IF_IPV6_RA_INCONSISTENCY Event

This event is generated when a Router Advertisement inconsistency has been detected by the implementation. The event is optional.

3.4.1.4 NPF_IF_IPV6_FIB_CHANGE Event

This event is generated when the FIB is changed on a particular interface. The event is optional.

3.4.2 Event Mask Bit Assignment

The following bit assignments represent mask bit assignment for IPv6 events.

#define NPF_IFIPv6_EVMASK_FIB	0x00000001 /*FIB and MTU change events*/
#define NPF_IFIPv6_EVMASK_MTU	0x00000002 /*MTU change events*/
#define NPF_IFIPv6_EVMASK_ADDR_STATUS	0x00000004 /*Addr status events*/
#define NPF_IFIPv6_EVMASK_RA	0x00000008 /*Ra inconsistency events*/
#define NPF_IFIPv6_EVMASK_ALL	0xFFFFFFFF /*All IPv6 event types*/
#define NPF_IFIPv6_EVMASK_NONE	0x00000000 /*No IPv6 events*/

3.4.3 Event Notification Structure: NPF_IfEventData_t

A forward reference to this structure must be added to the **NPF_IfEventData_t** structure in **npf_if_core.h** if IPv6 interfaces are supported. Before the declaration of **NPF_IfEventData_t**, the following must appear:

```
typedef struct    NPF_IfIPv6AddrStatusEventData
                  NPF_IfIPv6AddrStatusEventData_t;
```

```
typedef struct    NPF_IfIPv6RA_InConsistency
                  NPF_IfIPv6RA_InConsistency_t;
```

And the following must appear inside the union within the **NPF_IfEventData_t** structure:

```
NPF_IfIPv6AddrStatusEventData_t     *addr;      /*Addr and status*/
NPF_IfIPv6RA_InConsistency_t        *InC;       /*RA settings from packet*/
NPF_IPv6UC_FwdTableHandle_t         *newFIB;    /*FIB changed*/
NPF_uint16_t                        *rtMTU;     /*runtime MTU changed*/
```

3.5 Completion Callbacks and Error Returns

In addition to error codes defined in Interface Management API (core function set), the following error code will be returned:

NPF_IfAttrSet and NPF_IfCreateAndSet:

```
/* Optional feature not supported */
#define NPF_IF_IPV6_E_NO_SUCH_FEATURE     NPF_IF_E_IPV6_CODE(8)
```

- This error code is generated in response to the creation of an IPv6 interface for which the implementation does not support the requested NPF_IfIPv6DAD_Transmits_t and/or NPF_IfIPv6LL_AddrMode_t values. This value will be returned in response to the function invocation in the asynchronous callback and optionally the synchronous function return.

```
/* Error in parameters*/
#define NPF_IF_IPV6_E_INVALID_PARAM       NPF_IF_E_IPV6_CODE(7)
```

- This error code is generated if addresses are assigned to pseudo-interfaces. This value will be returned in response to the function invocation in the asynchronous callback and optionally the synchronous function return.

4 Functions

4.1 IPv6 Interface Management API

4.1.1 NPF_IfIPv6AddrSet: Set the Addresses for an IPv6 Interface

```
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 ifHandle,
        NPF_IN NPF_uint32_t nAddress,
        NPF_IN NPF_IPv6InterfaceAddr_t *if_IPv6AddrArray);
```

4.1.1.1 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 **nAddress** is zero, all settable addresses are removed.
Non-settable addresses are auto-generated link-local addresses (when `NPF_IfIPv6LL_AddrMode_t is enabled)` as well as other potential auto-generated addresses such as e.g. the All-Nodes Multicast Address, the Subnet-Router Anycast Address and the All-Routers Multicast Address. An auto-generated Solicited-Node Multicast address share faith with the associated unicast address.

4.1.1.2 Input Parameters

- **if_cbHandle:** the registered callback handle.
- **if_cbCorrelator:** the application's context for this call.
- **if_errorReporting:** the desired callback.
- **ifHandle:** the handle of the interface to modify.
- **nAddress**: the number addresses to assign to the interface.
- **if_IPv6AddrArray**: pointer to an array of IPv6 address structures.

4.1.1.3 Output Parameters

None

4.1.1.4 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.
- **NPF_IF_IPV6_E_NO_SUCH_FEATURE:** AUTODADVALIDATION flag on without DAD support, i.e. **`NPF_IfIPv6DAD_Transmits_t=0`**. The return of this error code is **optional**.
- **NPF_IF_IPV6_E_INVALID_PARAM:** Address setting are not supported for this particular interface type (pseudo-interface).

4.1.1.5 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.

4.1.2 NPF_IfIPv6AddrAdd: Add Addresses to an IPv6 Interface

```
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 ifHandle,
        NPF_IN NPF_uint32_t nAddress,
        NPF_IN NPF_IPv6InterfaceAddr_t *if_IPv6AddrArray);
```

4.1.2.1 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. If the address already exist, but with another flag setting or prefix length, this will replace the existing address.

4.1.2.2 Input Parameters

- **if_cbHandle:** the registered callback handle.
- **if_cbCorrelator:** the application's context for this call.
- **if_errorReporting:** the desired callback.
- **ifHandle:** the handle of the interface to modify.
- **nAddress:** the number of addresses to add. This must be greater than zero.
- **if_IPv6AddrArray:** pointer to an array of IPv6 address structures (may be NULL if nAddresses is zero).

4.1.2.3 Output Parameters

None

4.1.2.4 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.

- **NPF_IF_IPV6_E_NO_SUCH_FEATURE**: AUTODADVALIDATION flag on without DAD support, i.e. **`NPF_IfIPv6DAD_Transmits_t=0`**. This return of this error code is **optional**.
- **NPF_IF_IPV6_E_INVALID_PARAM:** Address setting are not supported for this particular interface type (pseudo-interface).

4.1.2.5 Asynchronous Response

A total of **nAddress** 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.

4.1.3 NPF_IfIPv6AddrDelete: Delete Addresses From an IPv6 Interface's List of IP Addresses

```
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 ifHandle,
        NPF_IN NPF_uint32_t nAddress,
        NPF_IN NPF_IPv6Prefix_t *if_IPv6PrefixArray);
```

4.1.3.1 Description

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

4.1.3.2 Input Parameters

- if_cbHandle: the registered callback handle.
- if_cbCorrelator: the application's context for this call.
- if_errorReporting: the desired callback.
- ifHandle: the handle of the affected interface.
- nAddress: the number of entries in the IP address array. This must not be zero.
- if_IPv6AddrArray: pointer to an array of IPv6 Prefix structures.

4.1.3.3 Output Parameters

None

4.1.3.4 Asynchronous Error Codes

- **NPF_NO_ERROR:** Operation successful.
- **NPF_IF_E_INVALID_HANDLE:** ifHandle 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_NO_SUCH_ADDRESS:** Address not found on this interface.

4.1.3.5 Asynchronous Response

A total of **nAddress** 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.

4.1.4 NPF_IfIPv6FIB_Set: Associate an IPv6 FIB with an Interface

```
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 nHandles,
        NPF_IN NPF_IfHandle_t *ifHandleArray,
        NPF_IN NPF_IPv6UC_FwdTableHandle_t if_FIB_Handle);
```

4.1.4.1 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.

4.1.4.2 Input Parameters

- **if_cbHandle:** the registered callback handle.
- **if_cbCorrelator:** the application's context for this call.
- **if_errorReporting:** the desired callback.
- **nHandle:** 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.

4.1.4.3 Output Parameters

None

4.1.4.4 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.

4.1.4.5 Asynchronous Response

A total of **`nHandle`** 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.

4.1.5 NPF_IfIPv6AddrStatus_Get: Retrieve address status

```
NPF_error_t NPF_IfIPv6AddrStatus_Get (
      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_HandleArray
        );
```

4.1.5.1 Description

This function returns via a callback, a pointer to an IPv6 address status structure (NPF_IfIPv6_AddrStatus_t) containing the current status of the addresses assigned to one or more of the indicated interfaces (All address assigned to an interface is returned, statically assigned address are returned with as being valid). The function will also return auto-generated addresses. This function is **optional** on interfaces with `NPF_IfIPv6DAD_Transmits_t=0`

4.1.5.2 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 interfaces to get attributes for.
- **if_HandleArray**: pointer to an array of interface handles.

4.1.5.3 Output Parameters

None.

4.1.5.4 Asynchronous Error Codes

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

4.1.5.5 Asynchronous Response

A total of **nHandles** times the number of address assigned to each interface 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_IfIPv6AddrStatus_t** structure for that interface.

4.1.6 NPF_IfIPv6RuntimeMTU_Get: retrieve runtime MTU value

```
NPF_error_t NPF_IfIPv6TunnelMTU_Get (
      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_HandleArray
      );
```

4.1.6.1 Description

This function returns via a callback, a pointer to an interface MTU value (NPF_uint16_t) containing the current value used by one or more of the indicated interfaces. **This is an optional function.**

4.1.6.2 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 interfaces to get attributes for.
- **if_HandleArray**: pointer to an array of interface handles.

4.1.6.3 Output Parameters

None.

4.1.6.4 Asynchronous Error Codes

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

4.1.6.5 Asynchronous Response

A total of **nHandles** 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 an **NPF_uint16_t** (carrying the MTU value).

4.1.7 NPF_IfIPv6RA_PrefixAdd : add Prefix to be advertised

```
NPF_error_t NPF_IfIPv6RA_PrefixAdd (
      NPF_IN NPF_callbackHandle_t if_cbHandle,
      NPF_IN NPF_correlator_t if_cbCorrelator,
      NPF_errorReporting_t if_errorReporting,
      NPF_IN NPF_IfHandle_t   ifHandle,
      NPF_IN NPF_uint32_t nPrefixes,
      NPF_IN NPF_IfIPv6PrefixAdv_t *prefix_Array
      );
```

4.1.7.1 Description

This function adds one or more Prefixes to the RA list of prefixes for a specific interface (given by **ifHandle**). If prefixes already exist, the list will just be extended. **This is an optional function.**

4.1.7.2 Input Parameters

- **if_cbHandle**: the registered callback handle.
- **if_cbCorrelator**: the application's context for this call.

- **if_errorReporting**: the desired callback.
- **ifHandle**: the relevant interface.
- **nPrefixes**: the number of prefixes to be advertised.
- **Prefix_Array**: pointer to an array of prefixes.

4.1.7.3 Output Parameters

None.

4.1.7.4 Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_E_INVALID_HANDLE**: An **if_Handle** is null or invalid.
- **NPF_IF_E_INVALID_IPADDR:** IP address is not a valid IPv6 address.
- **NPF_IF_E_INVALID_PLEN:** Invalid prefix length.
- **NPF_IF_IPV6_E_NO_SUCH_FEATURE:** The RA function is not supported on the interface

4.1.7.5 Asynchronous Response

A total of **nPrefixes** (if non-zero) asynchronous responses (**NPF_IfAsyncResponse_t**) will be passed to the callback function, in one or more invocations. Each response contains a prefix and an error code (success/failure).

4.1.8 NPF_IfIPv6RA_PrefixClear : deletes addresse(s) from advertising list

```
NPF_error_t NPF_IfIPv6RA_PrefixClear (
      NPF_IN NPF_callbackHandle_t if_cbHandle,
      NPF_IN NPF_correlator_t if_cbCorrelator,
      NPF_errorReporting_t if_errorReporting,
      NPF_IN NPF_IfHandle_t ifHandle,
      NPF_IN NPF_uint32_t nPrefixes,
      NPF_IN NPF_IfIPv6PrefixAdv_t *prefix_Array
      );
```

4.1.8.1 Description

This function deletes one or more Prefixes from the RA list of prefixes configured on a given interface (specified by **ifHandle**). A prefix is identified on the prefix only (excluding trailing bits). **This is an optional function.**

4.1.8.2 Input Parameters

- **if_cbHandle**: the registered callback handle.
- **if_cbCorrelator**: the application's context for this call.
- **if_errorReporting**: the desired callback.
- **ifHandle**: the specified interface.
- **nPrefixes**: the number of prefixes to delete.
- **prefix_Array**: pointer to an array of prefixes.

4.1.8.3 Output Parameters

None.

4.1.8.4 Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_E_INVALID_HANDLE**: An **if_Handle** is null or invalid.
- **NPF_IF_E_INVALID_IPADDR:** IP address is not a valid IPv6 address.
- **NPF_IF_E_INVALID_PLEN:** Invalid prefix length.
- **NPF_IF_IPV6_E_NO_SUCH_FEATURE:** The RA function is not supported on the interface

4.1.8.5 Asynchronous Response

A total of **nPrefixes** (if non-zero) asynchronous responses (**NPF_IfAsyncResponse_t**) will be passed to the callback function, in one or more invocations. Each response contains a prefix and an error code (success/failure).

4.1.9 NPF_IfIPv6RA_ParamsSet : set RA parameters

```
NPF_error_t NPF_IfIPv6RA_ParamsSet (
      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_HandleArray,
      NPF_IN NPF_IfIPv6RA_Params_t *params_Array
      );
```

4.1.9.1 Description

This function set the RA parameters related to one or more interface. **This is an optional function.**

4.1.9.2 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 interfaces to get attributes for.
- **`If_HandleArray`**: an array of interface handles.
- **`Params_Array`**: pointer to an array of RA parameters.

4.1.9.3 Output Parameters

None.

4.1.9.4 Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.

- **NPF_IF_E_INVALID_HANDLE**: An **if_Handle** is null or invalid.
- **NPF_IF_IPV6_E_NO_SUCH_FEATURE:** The RA function is not supported on the interface

4.1.9.5 Asynchronous Response

A total of **nHandles** (if non-zero) asynchronous responses (**NPF_IfAsyncResponse_t**) will be passed to the callback function, in one or more invocations. Each response contains an **NPF_IfIPv6RA_Params_t** struct and an error code (success/failure).

4.1.10 NPF_IfIPv6RA_SettingsGet : retrieve RA settings

```
NPF_error_t NPF_IfIPv6RA_SettingsGet (
      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_HandleArray
      );
```

4.1.10.1 Description

This function returns via a callback, a pointer to an IPv6 RA settings struct (NPF_IfIPv6RA_Settings_t) containing the current settings of the RA function on one or more of the indicated interfaces. **This is an optional function.**

4.1.10.2 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 interfaces to get attributes for.
- **if_HandleArray**: pointer to an array of interface handles.

4.1.10.3 Output Parameters

None.

4.1.10.4 Asynchronous Error Codes

- **NPF_NO_ERROR**: Operation successful.
- **NPF_IF_E_INVALID_HANDLE**: An **if_Handle** is null or invalid.
- **NPF_IF_IPV6_E_NO_SUCH_FEATURE:** The RA function is not supported on the interface

4.1.10.5 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_IfIPv6RA_Settings_t** structure for that interface.

4.1.11 NPF_IfIPv6TunnelV4AddrSet: Set IPv6-in-IPv4 Tunnel's IPv4 Addresses

NPF_error_t NPF_IfTunnelIPv6inv4AddrSet(
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_IfIPv6TunnelV4Addr_t *addrArray);

4.1.11.1 Description

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

4.1.11.2 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.

4.1.11.3 Output Parameters

None

4.1.11.4 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.

4.1.11.5 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.

4.1.12 NPF_IfIPv6TunnelV6AddrSet: Set IPv6-in-IPv6 Tunnel Addresses

```
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_IfIPv6TunnelV6Addr_t *addrArray);
```

4.1.12.1 Description

This function sets the source and destination IP addresses on one or more IPv6-in-IPv6 Tunnel interface(s).

4.1.12.2 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.

4.1.12.3 Output Parameters

None

4.1.12.4 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.

4.1.12.5 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.

4.1.13 NPF_IfIPv6TunnelFlowLabelSet: Set IPv6-in-IPv6 Tunnel Flow Label

```
NPF_error_t NPF_IfIPv6TunnelFlowLabelSet(
        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);
```

4.1.13.1 Description

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

4.1.13.2 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.

4.1.13.3 Output Parameters

None

4.1.13.4 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.

4.1.13.5 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.

4.1.14 NPF_IfIPv6TunnelHopsSet: Set IP-in-IP Tunnel Length (Hops)

```
NPF_error_t NPF_IfIPv6TunnelHopsSet(
        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);
```

4.1.14.1 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.

4.1.14.2 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.

4.1.14.3 Output Parameters

None

4.1.14.4 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.

4.1.14.5 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 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_IfIPv6AddrSet	Yes
NPF_IfIPv6AddrAdd	Yes
NPF_IfIPv6AddrDelete	Yes
NPF_IfIPv6FIB_Set	Yes
NPF_IfIPv6AddrStatusGet	No
NPF_IfIPv6TunnelV6AddrSet	Only if IPv6inv6 interfaces is supported
NPF_IfIPv6TunnelFlowLabelSet	Only if IPv6inv6 interfaces is supported
NPF_IfIPv6TunnelHopsSet	Only if IPv6inv6 interfaces is supported
NPF_IfIPv6RuntimeMTU_Get	No
NPF_IfIPv6TunnelV4AddrSet	Only if IPv6inv4 interfaces is supported
NPF_IfIPv6RA_PrefixAdd	Only if IPv6 RA is supported
NPF_IfIPv6RA_PrefixClear	Only if IPv6 RA is supported
NPF_IfIPv6RA_ParamsSet	Only if IPv6 RA is supported
NPF_IfIPv6RA_SettingsGet	Only if IPv6 RA is supported

6.3 API Events

Event Name	Required?
NPF_IF_IPV6_ADDR_STATUS	If NPF_IfIPv6LL_AddrMode_t is enabled – or- if NPF_IfIPv6DAD_Transmits_t greater than zero is supported
NPF_IF_IPV6_MTU_CHANGE	Only if Path MTU is supported is supported on interface
NPF_IF_IPV6_RA_INCONSISTENCY	Only if Router Advertisement function is supported on interface
NPF_IF_IPV6_FIB_CHNG	Yes

APPENDIX A HEADER FILE: NPF_IF_IPV6.H

```
/*
 * This header file defines typedefs, constants, and functions
 * for IPv6 extensions that apply to the NPF interface
 * management API version 3.0
 */

#ifndef __NPF_IF_IPv6_0_H__
#define __NPF_IF_IPv6_0_H__

#ifdef __cplusplus
extern "C" {
#endif

#ifndef NPF_IPv6UC_FwdTableHandle_t
/* Definition of Unicast Forwarding FIB handle */
typedef NPF_uint32_t NPF_IPv6UC_FwdTableHandle_t;
#endif

#ifndef NPF_IPv6Prefix_t
/*
 * IPv6 prefix structure
 * typedef declarations to appear in npf_if_core.h
 */
struct NPF_IPv6Prefix {
 NPF_IPv6Address_t IPv6Addr;
 NPF_uint8_t IPv6Plen;
};
#endif

/* IPv6 interface types */
#define NPF_TYPE_IPV6                7  /* native IPv6 interface */
#define NPF_TYPE_TUNNEL_IPV6INV4     8  /* IPv6inv4 tunnel interface */
#define NPF_TYPE_TUNNEL_IPV6         9  /* IPv6inv6 tunnel interface */

/*
 * IPv6 link capability attribute
 */
typedef enum {
      NPF_IF_IPV6_MULTICAST        = 1,   /* multicast */
      NPF_IF_IPV6_POINT_TO_POINT   = 2,   /* point-to-point */
      NPF_IF_IPV6_NBMA             = 3   /* NBMA */
 }NPF_IfIPv6LinkCapability_t;

#ifndef NPF_IPv6Address_t
/*
 * IPv6 address structure
 * typedef declarations to appear in npf_if_core.h
 */
struct NPF_IPv6Address {
      union {
            NPF_uchar8_t      b_addr[16];
            NPF_uint32_t      w_addr[4];
        }u;
};
```

```
#endif

/*
 * IPv6 Address Flag
 */
typedef NPF_uint8_t      NPF_IfIPv6AddrFlags_t;

#define NPF_IF_IPV6_ADDR_FLAGS_NONE             0x00
#define NPF_IF_IPV6_ADDR_PREFERRED              0x01
#define NPF_IF_IPV6_ADDR_DEPRECATED             0x02
#define NPF_IF_IPV6_ADDR_ANYCAST                0x04
#define NPF_IF_IPV6_ADDR_TENTATIVE              0x08
#define NPF_IF_IPV6_ADDR_PUBLIC                 0x10
#define NPF_IF_IPV6_ADDR_TEMPORARY              0x20
#define NPF_IF_IPV6_ADDR_AUTODADVALIDATION      0x40
#define NPF_IF_IPV6_ADDR_AUTOGENERATED          0x40

/*
 * IPv6 Interface address structure
 */
typedef struct {
  NPF_IPv6Address_t      IPv6Addr;       /* IPv6 address */
  NPF_uchar8_t           IPv6Plen;       /* Prefix length in bits (0-128) */
  NPF_IfIPv6AddrFlags_t  IPv6AddrFlags;  /* IPv6 address flags */
} NPF_IfIPv6InterfaceAddr_t;


/*
 * Duplicate Address Detection transmit counter
 */
typedef NPF_uint8_t     NPF_IfIPv6DAD_Transmits_t;

/*
* IPv6 Link Local address generation mode
 */
typedef enum {
   NPF_IfIPv6_LL_ADDR_GENERATION_ENABLE  = 1, /*Enable LL addr generation */
   NPF_IfIPv6_LL_ADDR_GENERATION_DISABLE = 2  /*Disable LL addr generation */
} NPF_IfIPv6LL_AddrMode_t;


/*
 * IPv6 RA mode
 */
typedef enum {
      NPF_IfIPv6_RA_MODE_ENABLE   = 1,   /* Enable RA */
      NPF_IfIPv6_RA_MODE_DISABLE   = 2   /* Disable RA */
} NPF_IfIPv6RA_Mode_t;

/*
 * IPv6 Interface Attributes
 * typedef declarations to appear in npf_if_core.h
 */
struct NPF_IfIPv6 {
   NPF_uint32_t                nAddr;  /*Number of IPv6 addresses*/
   NPF_IfIPv6InterfaceAddr_t   *Addrs;   /*Array of IPv6 addresses*/
   NPF_IPv6UC_FwdTableHandle_t FIB_Handle; /* IPv6 Forwarding info base*/
```

```
   NPF_IfIPv6LL_AddrMode_t     LL_mode; /*Enable autogen of Link Local addr*/
   NPF_IfIPv6DAD_Transmits_t   DAD_Transmits; /*nr of times DAD is perf*/
   NPF_IfIPv6RA_Mode_t         IfIPv6RA_Mode;  /* IPv6 RA on/off */
   NPF_IfIPv6LinkCapability_t  LinkCapability; /* MC, PPP or NBMA */
};


/*
 * IPv6 address state
 */
typedef enum {
      NPF_IPv6_ADDR_VALID = 1,
      NPF_IPv6_ADDR_PROBING = 2,
      NPF_IPv6_ADDR_INVALID = 3
} NPF_IPv6AddrState_t;


* typedef declarations to appear in npf_if_core.h
struct NPF_IfIPv6AddrStatusEventData {
      NPF_IPv6Prefix_t        addr;
      NPF_IPv6AddrState_t   addrState;
};

* typedef declarations to appear in npf_if_core.h
struct NPF_IfIPv6AddrStatus {
    NPF_IfIPv6InterfaceAddr_t    addr;
    NPF_IPv6AddrState_t  addrState;
};

/*
 * IPv4 Tunnel Addresses
 */
typedef struct {
NPF_IPv4Address_t destAddr; /* Tunnel destination address */
NPF_IPv4Address_t srcAddr; /* Tunnel source address*/
} NPF_IfIPv6TunnelV4Addr_t;

/*
 * IPv6 Tunnel Addresses
 */
typedef struct {
NPF_IPv6Address_t destAddr; /* Tunnel destination address */
NPF_IPv6Address_t srcAddr; /* Tunnel source address*/
} NPF_IfIPv6TunnelV6Addr_t;

/*
 * Tunnel interface MTU mode
 */
typedef enum {
      NPF_IFIPv6_TUNNEL_PMTU_DISABLED = 1,
      NPF_IFIPv6_TUNNEL_PMTU_ENABLED = 2
} NPF_IfIPv6TunnelMTU_t;

/*
 * Tunnel DSCP mode
 */
typedef enum {
```

```
 NPF_IFIPv6_INNER_DSCP_DISABLED = 1,
 NPF_IFIPv6_INNER_DSCP_ENABLED = 2
} NPF_IfIPv6InnerDSCP_t;

/*
 * IPv6 in IPv4 Tunnel types
 */
typedef enum {
 NPF_IF_V6INV4_CONFIGURED = 1, /* Configured Tunnel */
 NPF_IF_V6INV4_6TO4 = 2        /* 6to4 Tunnel*/
} NPF_IPv6TunnelV4Type_t;


/*
 * IPv6-in-IPv4 Tunnel Interface attributes
 * typedef declarations to appear in npf_if_core.h
 */
 struct NPF_IfIPv6TunnelV4 {
     NPF_IPv6TunnelV4Type_t         tunnelType; /* Type of v6inv4 Tunnel */
     NPF_IPv4Address_t              IPv4DstAddr; /*Destination IPv4 address*/
     NPF_IPv4Address_t              IPv4SrcAddr; /* Source IPv4 address */
     NPF_uint8_t                    IPv4TTL; /* TTL in IPv4 header*/
     NPF_uint8_t                    IPv4DSCP; /*DSCP in IPv4 header*/
     NPF_IPv6UC_FwdTableHandle_t    FIB_Handle;/*IPv6 Fwd info base*/
     NPF_IfIPv6RA_Mode_t            ifIPv6RA_Mode;  /* IPv6 RA on/off */
     NPF_IfIPv6LL_AddrMode_t        LL_mode; /*Enable autogen of LL addr*/
     NPF_IfIPv6DAD_Transmits_t      DAD_Transmits; /*nr of DAD req*/
     NPF_IfIPv6InnerDSCP_t          IPv6DSCP;/*Use DSCP from IPv6 pck*/
     NPF_IfIPv6TunnelMTU_t          MTU_MODE; /* PMTU on/off */
     NPF_uint32_t                   nAddr;  /*Number of IPv6 addresses*/
     NPF_IfIPv6InterfaceAddr_t      *Addrs; /*Array of IPv6 addresses*/
     NPF_IfIPv6LinkCapability_t     LinkCapability; /*IPv6 link capability */
};

/*
 * IPv6-in-IPv6 (rfc2473 compliant) Tunnel Interface Attributes
 * typedef declarations to appear in npf_if_core.h
 */
struct NPF_IfIPv6TunnelIPv6 {
      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-default 0 */
      NPF_IPv6UC_FwdTableHandle_t   FIB_Handle;/*FIB for inner hdr*/
      NPF_IfIPv6TunnelMTU_t         MTU_MODE;/* PMTU on/off */
      NPF_IfIPv6InnerDSCP_t         IPv6DSCP;/*use inner packet DSCP*/
      NPF_IfIPv6LL_AddrMode_t       LL_mode; /*Enable autogen of LL addr*/
      NPF_IfIPv6RA_Mode_t           ifIPv6RA_Mode;  /* IPv6 RA on/off */
      NPF_IfIPv6DAD_Transmits_t     DAD_Transmits; /*nr of DAD req*/
      NPF_uint32_t                  nAddr;  /*Number of IPv6 addresses*/
      NPF_IfIPv6InterfaceAddr_t     *Addrs; /*Array of IPv6 addresses*/
      NPF_IfIPv6LinkCapability_t    LinkCapability; /*IPv6 link capability */
};

/*
 * IPv6 Prefix Flags
```

```
 */
typedef NPF_uint8_t      NPF_IfIPv6PrefixFlags_t;

#define NPF_IF_IPV6_PREFIX_ONLINK          0x01
#define NPF_IF_IPV6_PREFIX_AUTO            0x02
#define NPF_IF_IPV6_VALIDLIFETIME_DECR     0x04
#define NPF_IF_IPV6_PREFFEREDLIFETIME_DECR 0x08

/*
 * IPv6 Prefix Lifetimes
 */
typedef struct {
  NPF_uint32_t     IPv6ValidLft;             /*Time in seconds */
  NPF_uint16_t     IPv6PreferredLft;         /*Time in seconds */
} NPF_IfIPv6PrefixLfts_t;

/*
 * IPv6 Prefix Advertisement
 */
typedef struct {
  NPF_IPv6Prefix_t            adv_Prefix;   /* Prefix to be advertised */
  NPF_IfIPv6PrefixFlags_t     prefix_Flags;       /* Prefix flags */
  NPF_IfIPv6PrefixLfts_t      prefix_Lifetimes; /* Prefix lifetime, preferred
   and valid */
}NPF_IfIPv6PrefixAdv_t;

/*
 * Advertised router preference (High, medium or low)
 */
typedef enum {
NPF_IF_IPv6_RTR_PREF_HIGH = 1,
NPF_IF_IPv6_RTR_PREF_MEDIUM = 2,
NPF_IF_IPv6_RTR_PREF_LOW = 3
}NPF_IFIPv6RtrPref_t;


/*
 * IPv6 RA Flags
 */
typedef NPF_uint8_t      NPF_IfIPv6RA_Flags_t;
#define NPF_IF_IPV6_RA_MANAGED                          0x01
#define NPF_IF_IPV6_RA_OTHER_CONFIG                     0x02
#define NPF_IF_IPV6_RA_ADV_MTU                          0x04
#define NPF_IF_IPV6_RA_ADV_MAX_RA_ADV_INTERVAL          0x08

/*
 * IPv6 RA parameters
 * typedef declarations to appear in npf_if_core.h
 */
struct NPF_IfIPv6RA_Params {
  NPF_IfIPv6RA_Flags_t IfIPv6RAFlags;      /* IPv6 RA Flags */
  NPF_IFIPv6RtrPref_t  routerPreference;   /* router preference*/
  NPF_uint32_t         MaxRtrAdvInterval;  /* Time in msec */
  NPF_uint32_t         MinRtrAdvInterval;  /* Time in msec */
  NPF_uint32_t         AdvReachableTime;   /* Time in msec */
  NPF_uint32_t         AdvRetransTimer;    /* Time in msec */
  NPF_uint8_t          AdvCurHopLimit;     /* Current Hop Limit value*/
```

```
  NPF_uint16_t         AdvDefaultLft;      /* Time in sec */
  NPF_uint32_t         MinDelayBetweenRAs; /* for rate limiting, def. 3sec*/
};


/*
 * IPv6 RA settings (prefixes and RA parameters)
 * typedef declarations to appear in npf_if_core.h
 */
struct NPF_IfIPv6RA_Settings {
  NPF_uint32_t                nPrefix;
  NPF_IfIPv6PrefixAdv_t       *Prefix;
  NPF_IfIPv6RA_Params_t RA_Params;
};

/*
 * IPv6 RA inconsistency check
 * typedef declarations to appear in npf_if_core.h
 */
struct NPF_IfIPv6RA_Inconsistency {
  NPF_IPv6Prefix_t      RecvLL_addr;/* Link local addr of received RA msg */
  NPF_IfIPv6RA_Flags_t  RA_Flags;   /* Only managed and other relevant */
  NPF_uint32_t          AdvReachableTime;   /* Time in msec */
  NPF_uint32_t          AdvRetransTimer;    /* Time in msec */
  NPF_uint8_t           AdvCurHopLimit;     /* Hop Limit value*/
  NPF_uint16_t          MTU;              /*value from MTU options field */
  NPF_uint32_t          nPrefix;            /*number of prefixes */
  NPF_IfIPv6PrefixAdv_t *recvPrefix;   /* adv. prefixes from other routers */
};

/*
 *    Completion Callback Types for IPv6 interfaces
 */
#define NPF_IF_IPV6_ADDR_SET                    ((NPF_IF_TYPE_IPV6<<16)+1)
#define NPF_IF_IPV6_ADDR_ADD                    ((NPF_IF_TYPE_IPV6<<16)+2)
#define NPF_IF_IPV6_ADDR_DELETE                 ((NPF_IF_TYPE_IPV6<<16)+3)
#define NPF_IF_IPV6_FIB_SET                     ((NPF_IF_TYPE_IPV6<<16)+4)
#define NPF_IF_IPV6_ADDR_STATUS_GET             ((NPF_IF_TYPE_IPV6<<16)+5)
#define NPF_IF_IPV6_TUNNEL_V6_ADDR_SET          ((NPF_IF_TYPE_IPV6<<16)+6)
#define NPF_IF_IPV6_TUNNEL_FLOW_LABEL_SET       ((NPF_IF_TYPE_IPV6<<16)+7)
#define NPF_IF_IPV6_TUNNEL_HOPS_SET             ((NPF_IF_TYPE_IPV6<<16)+8)
#define NPF_IF_IPV6_RUNTIME_MTU_GET             ((NPF_IF_TYPE_IPV6<<16)+9)
#define NPF_IF_IPV6_TUNNEL_V4_ADDR_SET          ((NPF_IF_TYPE_IPV6<<16)+10)
#define NPF_IF_IPV6_RA_PREFIX_ADD               ((NPF_IF_TYPE_IPV6<<16)+11)
#define NPF_IF_IPV6_RA_PREFIX_CLEAR             ((NPF_IF_TYPE_IPV6<<16)+12)
#define NPF_IF_IPV6_RA_PARAMS_SET               ((NPF_IF_TYPE_IPV6<<16)+13)
#define NPF_IF_IPV6_RA_SETTING_GET              ((NPF_IF_TYPE_IPV6<<16)+14)

/*
 * Asynchronous error codes (returned in function callbacks)
 * Used by IPv6 interface functions
 */

#define NPF_IF_E_IPV6_CODE(code) (0x10000+(NPF_IF_TYPE_IPV6<<8)+(code))

/* Invalid IP address */
#define NPF_IF_IPV6_E_INVALID_IPADDR      NPF_IF_E_IPV6_CODE(1)
```

```
/* Invalid IP prefix length */
#define NPF_IF_IPV6_E_INVALID_PLEN        NPF_IF_E_IPV6_CODE(2)

/* Invalid FIB handle */
#define NPF_IF_IPV6_E_INVALID_FIB_HANDLE  NPF_IF_E_IPV6_CODE(3)

/* Invalid TTL value */
#define NPF_IF_IPV6_E_INVALID_TTL         NPF_IF_E_IPV6_CODE(4)

/* Invalid DSCP value */
#define NPF_IF_IPV6_E_INVALID_DSCP        NPF_IF_E_IPV6_CODE(5)

/* IP address does not exist on this interface */
#define NPF_IF_IPV6_E_NO_SUCH_ADDRESS     NPF_IF_E_IPV6_CODE(6)

/* Error in parameters*/
#define NPF_IF_IPV6_E_INVALID_PARAM       NPF_IF_E_IPV6_CODE(7)

/* Optional feature not supported */
 #define NPF_IF_IPV6_E_NO_SUCH_FEATURE    NPF_IF_E_IPV6_CODE(8)

/*
 * Event types
 */

/* Duplicated Address Detection Event */
#define NPF_IF_IPV6_ADDR_STATUS ((NPF_IF_TYPE_IPV6<<16)+1)
/* Tunnel MTU changed */
#define NPF_IF_IPV6_MTU_CHANGE ((NPF_IF_TYPE_IPV6<<16)+2)
/* RA inconsistency detected */
#define NPF_IF_IPV6_RA_INCONSISTENCY ((NPF_IF_TYPE_IPV6<<16)+3)
/* IPv6UC FIB changed */
#define NPF_IF_IPV6_FIB_CHANGE ((NPF_IF_TYPE_IPV6<<16)+4)

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 ifHandle,
                        NPF_IN NPF_uint32_t nAddress,
                        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 ifHandle,
                        NPF_IN NPF_uint32_t nAddress,
                        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 ifHandle,
                        NPF_IN NPF_uint32_t nAddress,
```

```
                        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 nHandles,
                        NPF_IN NPF_IfHandle_t *ifHandleArray,
                        NPF_IN NPF_IPv6UC_FwdTableHandle_t if_FIB_Handle);

NPF_error_t NPF_IfIPv6AddrStatus_Get (
                        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 *ifHandleArray);

NPF_error_t NPF_IfIPv6RuntimeMTU_Get (
                        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 *ifHandleArray);

NPF_error_t NPF_IfIPv6RA_PrefixAdd (
                        NPF_IN NPF_callbackHandle_t if_cbHandle,
                        NPF_IN NPF_correlator_t if_cbCorrelator,
                        NPF_errorReporting_t if_errorReporting,
                        NPF_IN NPF_uint32_t nPrefixes,
                        NPF_IN NPF_IfIPv6PrefixAdv_t *prefix_Array);

NPF_error_t NPF_IfIPv6RA_PrefixClear (
                        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 nPrefixes,
                        NPF_IN NPF_IfIPv6PrefixAdv_t *prefix_Array);

NPF_error_t NPF_IfIPv6RA_ParamsSet (
                        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_HandleArray,
                        NPF_IN NPF_IfIPv6RA_Params_t *params_Array);

NPF_error_t NPF_IfIPv6RA_SettingsGet (
                        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_HandleArray);

NPF_error_t NPF_IfIPv6TunnelV4AddrSet(
                        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_IfIPv6TunnelV4Addr_t *addrArray);

NPF_error_t NPF_IfIPv6TunnelV6AddrSet(
                        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_IfIPv6TunnelV6Addr_t *addrArray);

NPF_error_t NPF_IfIPv6TunnelFlowLabelSet(
                        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);

NPF_error_t NPF_IfIPv6TunnelHopsSet(
                        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);

#ifdef __cplusplus
}
#endif
```

#endif

APPENDIX B ACKNOWLEDGEMENTS

Working Group Chair:

Alex Conta, Transwitch, aconta@txc.com

Task Group Chair:

Alex Conta, Transwitch, aconta@txc.com

Task Group Editor:

John Renwick, Agere Systems, jrenwick@agere.com

The following individuals are acknowledged for their participation to IM API TG teleconferences, plenary meetings, mailing list, and/or for their NPF contributions used for the development of this Implementation Agreement. This list may not be all-inclusive since only names supplied by member companies for inclusion here will be listed. The NPF wishes to thank all active participants to this Implementation Agreement, whether listed here or not.

The list is in alphabetical order of last names:

APPENDIX C LIST OF COMPANIES BELONGING TO NPF DURING APPROVAL PROCESS

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