ATM API

Reference Guide

Control Plane-Platform Development Kit 2.11

March 2004

Contents

Tables

Revision History

Revision	Description	Date	Author
2.11	Updated for Release 2.11	March 2004	Shashank S. Prasad
2.1	Updated for Release 2.1	December 2003	Shashank S. Prasad
2.0	Updated for Release 2.0	August 2003	Shashank S. Prasad

Part 1: Overview

1 Overview

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

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

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

An orthogonal management plane manages the control and forwarding planes.

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

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

1.1 ATM API Overview

This document presents the ATM API for defining and managing ATM functionality in forwarding elements. This document along with the NPF Interface Management document, implements the complete functionality for an ATM forwarding element.

1.2 Assumptions and Dependencies

The following assumptions and dependencies apply to the ATM API module:

- NPF interface management module implements a few ATM technology APIs like connection management and statistics handling
- ATM interface refers to the logical entity that maps to a physical ATM port. There is one interface per physical port.
- ATM connection or a Virtual Channel Connection (VCC) is a virtual connection over an ATM interface, which is identified by VPI/VCI
- ATM connection and VCC are interchangeably used in this document
- An AAL2 channel is created over an AAL2 connection. A single AAL2 connection can hold up to 256 AAL2 channels.
- Forward direction refers to egress and backward direction refers to ingress

- The APIs defined in the subsequent sections pertain to specific Virtual Path (VP) connections, if VCI = 0, for the corresponding VCC address. The VPI value determines the specific VP connection.

1.3 References

Table 1 lists the documents that are referenced in this document.

Table 1. References

Reference	Document
[1]	NPF Software API conventions Implementation Agreement, Revision 1.0, Network Processing Forum - http://www.npforum.org/
[2]	Software Architecture Overview, Control Plane PDK 1.0.5, June 2002, Intel Corporation
[3]	API Framework Reference, Control Plane PDK 1.0.5, August 2002, Intel Corporation
[4]	Traffic Management Specification, Version 4.1, AF-TM-0121.000, ATM Forum, March 1999
[5]	NPF Software API Framework Implementation Agreement, Revision 1.0, Network Processing Forum - http://www.npforum.org/
[6]	NPF Software API Framework Lexicon Implementation Agreement, Revision 1.0, Network Processing Forum - http://www.npforum.org/
[7]	NPF Interface Management API Implementation Agreement, Revision 1.0, Network Processing Forum - http://www.npforum.org/
[8]	Intel Network Processor Division, ATM Extensions to CPPDK, Technical Proposal, 9001214 1.2
[9]	B-ISDN ATM Adaptation Layer Specification: Type 1 AAL, I.363.1 (08/96)
[10]	B-ISDN ATM Adaptation Layer Specification: Type 2 AAL, I.363.2 (11/2000)
[11]	B-ISDN ATM Adaptation Layer Specification: Type 3/4 AAL, I.363.3 (08/96)
[12]	B-ISDN ATM Adaptation Layer Specification: Type 5 AAL, I.363.5 (08/96)
[13]	Segmentation and Reassembly Service Specific Convergence Sub Layer for AAL Type 2, I.366.1 (06/98)
[14]	AAL Type 2 Service Specific Convergence Sub Layer for Narrow-band Services, I.366.2 (11/2000)
[15]	B-ISDN Operations and Maintenance Principles and Functions, I.610 (02/99)
[16]	ATM User-Network Interface (UNI) Signalling Specification, Version 4.1
[17]	Integrated Local Management Interface (ILMI) Specification Version 4.0

1.4 Terminology

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

Table 2. Terminology

Term	Description
AAL	ATM Adaptation Layer
ABR	Available Bit Rate
AIS	Alarm Indication Signal
ATM	Asynchronous Transfer Mode
BR	Backward Reporting
CBR	Constant Bit Rate
CC	Continuity Check
CDV	Cell Delay Variation
CDVT	CDV Tolerance
CE	Control Element
CLP	Cell Loss Priority
CP	Control Plane
CPCS	Common Part Convergence Sub layer
CP-PDK	Control Plane Platform Development Kit
CPS	Common Part Sub layer
F4	OAM flow on virtual path level
F5	OAM flow on virtual channel level
FE	Forwarding Element
FEC	Forward Error Correction
FPM	Forward Performance Monitoring
GFR	Guaranteed Frame Rate
ICR	Initial Cell Rate
ID	Identifier
IE	Information Element
IP	Internet Protocol

Term	Description
LB	Loopback
LES	Loop Emulation Service
MBS	Maximum Burst Size
MCR	Minimum Cell Rate
MFS	Maximum Frame Size
MID	Message Identifier
NNI	Network Node Interface
Nrt-VBR	Non-real-time VBR
OAM	Operation and Maintenance
PCR	Peak Cell Rate
PM	Performance Monitoring
PTI	Payload Type Indicator
PVC	Permanent Virtual Connection
QoS	Quality of Service
RDI	Remote Defect Indication
RM-cell	Resource Management Cell
rt-VBR	Real-time VBR
SCR	Sustainable Cell Rate
SDT	Structured Data Transfer
SDU	Service Data Unit
SRTS	Synchronous Residual Time Stamp
SVC	Switched Virtual Connection
TM	Traffic Management
UBR	Unspecified Bit Rate
UNI	User Network Interface
UPC	Usage Parameter Control
VBR	Variable Bit Rate

Term	Description
VC	Virtual Connection
VCC	Virtual Channel Connection
VCI	Virtual Channel Identifier
VPC	Virtual Path Connection
VPI	Virtual Path Identifier

Part 2: ATM Interface Management APIs

2 ATM Interface Management APIs

2.1 Create an ATM Interface

Syntax

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

Description

This function creates an ATM interface with the administratively disabled (`NPF_IF_ADMIN_STATUS_DOWN`) state by default. The ATM interface created is undifferentiated until the attributes in them are set using `NPF_IfAttrSet()` or other functions.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`n_if`	Number of interfaces to create
`if_Type`	Interface type: `NPF_IF_TYPE_ATM`

Output Parameters

None.

Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_PARAM`	Operation failed, interface not created

Asynchronous response

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

2.2 Set all ATM Attributes of an ATM Interface

Syntax

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

Description

This function sets all the attributes of ATM interfaces, from the contents of an array of structures passed by the caller, as defined in `NPF_IfGeneric_t`. Ownership of the structure memory remains with the caller, that is, the API implementation must copy all the necessary contents before it returns. Any single attribute can be set with its own function call. This function sets multiple attributes automatically and efficiently.

Note: The number of `NPF_IfGeneric_t` structures and the number of interface handles in the two arrays must be the same, and equal to the `n_handles` argument. This function sets different set of attributes for each named interface.

As defined by the NP Forum's Interface Management API, the following ATM interface attributes, in addition to the port number, are proposed to be set through `NPF_IfAttrSet` API:

- Port number
- ATM address
- Maximum number of active VPI bits
- Maximum number of active VCI bits
- ATM network interface (UNI/NNI)
- ATM interface type (public/private)
- ATM UNI device type (user/node)
- CPID in the forward direction (primarily for OAM operation)
- CPID in the reverse direction (primarily for OAM operation)

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`n_handles`	Number of interfaces to set attributes for
`if_HandleArray`	Pointer to an array of interface handles
`if_StructArray`	Pointer to an array of structures containing the new interface attributes

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_ATTRIBUTE`	An invalid attribute (other than those mentioned below)

Generic Interface Errors

`NPF_IF_E_INVALID_HANDLE`	`if_Handle` is null or invalid
`NPF_IF_E_INVALID_PORT`	Port is invalid
`NPF_IF_E_INVALID_SPEED`	Speed is invalid
`NPF_IF_E_INVALID_IF_TYPE`	Interface type is invalid
`NPF_IF_E_INVALID_ADMIN_STATUS`	Admin status is invalid

ATM Interface Errors

`NPF_IF_E_INVALID_ATM_ADDRESS`	ATM address is invalid
`NPF_IF_E_INVALID_MAX_VPI_BITS`	A maximum VPI bit is invalid
`NPF_IF_E_INVALID_MAX_VCI_BITS`	A maximum VCI bit is invalid

Asynchronous response

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

2.3 Create an ATM Interface and Set All its Attributes

Syntax

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

Description

This function simultaneously creates and sets all the attributes of ATM interfaces, from the contents of an array of structures passed by the caller (`NPF_IfGeneric_t`). Ownership of the structure memory remains with the caller. The API implementation must copy all contents before returning.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`n_if`	Number of interfaces for which attributes are to be set
`if_StructArray`	Pointer to an array of structures containing new interface attributes

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_ATTRIBUTE`	An invalid attribute (other than those mentioned below)

Generic Interface Errors:

`NPF_IF_E_INVALID_HANDLE`	`if_Handle` is null or invalid
`NPF_IF_E_INVALID_PORT`	Port is invalid
`NPF_IF_E_INVALID_SPEED`	Invalid interface speed parameter
`NPF_IF_E_INVALID_IF_TYPE`	Invalid interface type code
`NPF_IF_E_INVALID_ADMIN_STATUS`	Invalid administrative status code

ATM Interface Errors

`NPF_IF_E_INVALID_ATM_ADDRESS` ATM address is invalid

`NPF_IF_E_INVALID_MAX_VPI_BITS` Maximum VPI bit is invalid

`NPF_IF_E_INVALID_MAX_VCI_BITS` Maximum VCI bit is invalid

Asynchronous response

A total of `n_if` asynchronous responses (`NPF_IfAsyncResponse_t`) are passed to the callback function, in one or more invocations. Each response contains the following:

- New interface handle
- A success code or a possible error code, if an interface is not created or any attributes are not set

Responses are linked to interface attributes in the following way:

For each response, the union in the response structure contains the corresponding index of the `if_StructArray` element that contains its attributes. For example, the response for the first array element will include an interface handle and an `arrayIndex` value of zero, the tenth array element, an `arrayIndex` of 9, and so on.

In addition to the port number, defined by NP Forum's Interface Management API, the following ATM interface attributes are also proposed to be set through `NPF_IfCreateAndSet` API.

- Port number
- ATM address
- Maximum number of active VPI Bits
- Maximum number of active VCI Bits
- ATM network interface (UNI/NNI)
- ATM interface type (public/private)
- ATM UNI device type (user/node)
- CPID in the forward direction (primarily for OAM operation)
- CPID in the reverse direction (primarily for OAM operation)
- The NPF_IfATM_t structure is redefined.

2.4 Enable an ATM Interface

Syntax

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

Description

This function administratively enables an ATM interface. It unilaterally enables the administrative status of all the VCCs on the said ATM interface. If the ATM interface is ready for operation, then the interface can transmit and receive packets.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`n_handles`	Number of interfaces to enable
`if_HandleArray`	Pointer to an array of interface handles

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_HANDLE`	An `if_Handle` is null or invalid

Asynchronous response

A total of `n_handles` asynchronous responses (`NPF_IfAsyncResponse_t`) are 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.

2.5 Disable an ATM Interface

Syntax

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

Description

This function disables the ATM interfaces administratively and not operationally. Once disabled, none of the configured VCCs on the said interface can transmit or receive packets.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`n_handles`	Number of interfaces to disable
`if_HandleArray`	Pointer to an array of interface handles

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_HANDLE`	An `if_Handle` is null or invalid

Asynchronous response

A total of `n_handles` asynchronous responses (`NPF_IfAsyncResponse_t`) are 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.

2.6 Delete an ATM Interface

Syntax

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

Description

This function deletes one or more ATM interfaces. This handle should not be used after the return of the call. All the VCCs associated to the interface are also deleted.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`n_handles`	Number of ATM interfaces to delete
`if_HandleArray`	Pointer to an array of handles of the ATM interfaces, which are to be deleted

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_PARAM`	Interface not deleted

Asynchronous response

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

2.7 Read ATM Interface Statistics

Syntax

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

Description

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

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`n_handles`	Number of interfaces for which statistics have to be obtained
`if_HandleArray`	Pointer to an array of interface handles

Output Parameters

None.

Asynchronous Error Codes

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

Asynchronous response

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

2.8 Fetch Operational Status of an Interface

Syntax

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

Description

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

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`n_handles`	Number of interfaces to query
`if_HandleArray`	Pointer to an array of interface handles

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_HANDLE`	A null or invalid `if_Handle`

Asynchronous response

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

Part 3: ATM VCC Management APIs

3 ATM VCC Management APIs

All ATM interfaces are created and their attributes are defined from the APIs defined Section 2, ATM Interface Management APIs. New virtual channel connections are created to carry the ATM traffic after the creation of an ATM interface.

The connection can be a permanent virtual connection or a switched virtual connection. Virtual circuit connection is a circuit or path between points in a network that appears to be a discrete physical path, but is actually a managed pool of circuit resources. Specific circuits are allocated to meet traffic requirements from this pool of circuit resources.

Permanent Virtual Circuit (PVC) is a virtual circuit that is permanently available to the user, like a dedicated or leased line reserved for the user.

Switched Virtual Circuit (SVC) is a virtual circuit in which a connection session is set up for a user only for the duration of the connection.

At present, the ATM API interface does not strictly conform to the Network Processing Forum Standards for ATM Interface.

The following section defines the Control Plane APIs for ATM Virtual Channel Connection management.

3.1 Add or Modify an ATM VCC

Syntax

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

Description

The function adds or creates one or more VCCs on a single ATM interface, or modifies the existing ATM attributes (AAL type, parent interface handle and QOS profile) of VCCs with the given VPI or VCI address.

Ownership of the VCC structure memory stays with the caller. The API implementation must copy all VCC attributes before returning. Each VCC is defined to cater to the traffic services of specific AAL type users. The AAL profile of the VCC is defined by the `NPF_IfAALProfile_t`.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`if_Handle`	Handle of the interface
`n_Vccs`	Number of VCCs to be set
`if_VccStructArray`	Pointer to an array of ATM VCC attribute structures

Output Parameters

None.

Asynchronous Error Codes

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

AAL1 Error Codes

`NPF_IF_E_INVALID_SUBTYPE`	Invalid subtype
`NPF_IF_E_INVALID_CBR`	Invalid CBR or beyond the QoS of the VCC
`NPF_IF_E_INVALID_CLK_REC_TYPE`	Invalid clock recovery type
`NPF_IF_E_INVALID_FEC`	Invalid FEC
`NPF_IF_E_INVALID_SDT`	Invalid SDT
`NPF_IF_E_INVALID_PAR_FILL_CELL`	Invalid partially filled cells
`NPF_IF_E_INVALID_CELL_LOSS_PER`	Invalid cell loss integration period

AAL2 Error Codes

NPF_IF_E_INVALID_APPID	Invalid App ID
NPF_IF_E_INVALID_CPS_ATTR	Invalid CPS attribute
NPF_IF_E_INVALID_SSCS_I3661_ATTR	Invalid I.366.1 SSCS attribute
NPF_IF_E_INVALID_SSCS_I3662_ATTR	Invalid I.366.2 SSCS attribute
NPF_IF_E_INVALID_ATM_TRUNK_ATTR	Invalid ATM trunking attribute
NPF_IF_E_INVALID_LES_ATTR	Invalid LES attribute

AAL3/4 Error Codes

NPF_IF_E_INVALID_MAX_SIZE_FWD	Invalid forward maximum CPCS SDU size
NPF_IF_E_INVALID_MAX_SIZE_BWD	Invalid backward maximum CPCS SDU size
NPF_IF_E_INVALID_MID_RANGE_LOW	Invalid MID range low
NPF_IF_E_INVALID_MID_RANGE_HIGH	Invalid MID range high
NPF_IF_E_INVALID_AAL_MODE	Invalid AAL mode
NPF_IF_E_INVALID_SSCS_TYPE	Invalid SSCS type

AAL5 Error Codes

NPF_IF_E_INVALID_MAX_SIZE_FWD	Invalid forward maximum CPCS SDU size
NPF_IF_E_INVALID_MAX_SIZE_BWD	Invalid backward maximum CPCS SDU size
NPF_IF_E_INVALID_AAL_MODE	Invalid AAL mode
NPF_IF_E_INVALID_SSCS_TYPE	Invalid SSCS type

Asynchronous response

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

3.2 Bind a Higher Layer Interface to an ATM VCC

Syntax

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

Description

This function associates a single higher-layer interface with one or more VCCs on the same ATM interface. The VCCs must be created before making the call. This API must bind other applications over the ATM, like IP over ATM, and so on.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`if_ATM_Handle`	Handle of the ATM interface
`if_ParentHandle`	Handle of the higher layer interface to bind
`n_Vccs`	Number of VCCs to be set
`if_VccAddrArray`	Pointer to an array of VPI/VCI addresses

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_ATM_HANDLE`	Invalid ATM UNI interface handle
`NPF_IF_E_INVALID_L3_HANDLE`	Invalid L3 interface handle
`NPF_IF_E_NO_SUCH_VCC`	VCC does not exist

Asynchronous response

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

3.3 Delete an ATM VCC

Syntax

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

Description

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

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`if_Handle`	Handle of the ATM UNI interface
`n_Vccs`	Number of VCCs to delete
`if_VccAddrArray`	Pointer (to an array of VPI/VCI addresses) to be deleted

Output Parameters

None.

Asynchronous Error Codes

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

Asynchronous response

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

3.4 Enable a VCC on an ATM Interface

Syntax

```
NPF_error_t NPF_IfATM_VccEnable(
            NPF_IN NPF_callbackHandle_t  atm_cbHandle,
            NPF_IN NPF_correlator_t      atm_cbCorrelator,
            NPF_IN NPF_errorReporting_t  atm_errorReporting,
            NPF_IN NPF_IfHandle_t        ifHandle,
            NPF_IN NPF_uint32_t          n_Vccs,
            NPF_IN NPF_VccAddr_t         *vcc_AddrArray);
```

Description

This function enables one more VCCs identified by the VPI/VCI on an ATM interface administratively. The interface can receive and transmit packets if the ATM VCC is ready for operation.

Input Parameters

`atm_cbHandle`	Registered callback handle
`atm_cbCorrelator`	Application's context for this call
`atm_errorReporting`	Desired callback
`ifHandle`	Handle of an `NPF_IF_TYPE_ATM` type interface
`n_Vccs`	Number of VCCs to be enabled administratively on the specified ATM interface
`vcc_AddrArray`	Pointer to an array of VPI/VCI addresses

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_HANDLE`	`if_Handle` is null or invalid, or it is not an ATM interface
`NPF_IF_E_INVALID_VCC_ADDRESS`	ATM VCC address is invalid on the specified ATM interface

Asynchronous response

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

3.5 Disable a VCC on an ATM Interface

Syntax

```
NPF_error_t NPF_IfATM_VccDisable(
            NPF_IN NPF_callbackHandle_t  atm_cbHandle,
            NPF_IN NPF_correlator_t      atm_cbCorrelator,
            NPF_IN NPF_errorReporting_t  if_errorReporting,
            NPF_IN NPF_IfHandle_t        ifHandle,
            NPF_IN NPF_uint32_t          n_Vccs,
            NPF_IN NPF_VccAddr_t         *vcc_AddrArray);
```

Description

This function disables one or more VCCs on the specified the ATM interface administratively. If disabled, it can no longer send or receive packets.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`ifHandle`	Handle of the ATM type interface
`n_Vccs`	Number of VCCs to be administratively enabled on the specified ATM interface
`vcc_AddrArray`	Pointer to an array of VPI/VCI addresses

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_HANDLE`	`if_Handle` is null or invalid, or it is not an ATM interface
`NPF_IF_E_INVALID_VCC_ADDRESS`	The ATM VCC address is invalid on the specified ATM interface

Asynchronous response

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

3.6 Fetch Operational Status of a VCC on an ATM Interface

Syntax

```
NPF_error_t NPF_IfATM_VccOperStatusGet(
            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           n_Vccs,
            NPF_IN NPF_VccAddr_t          *vcc_AddrArray);
```

Description

This function returns the operational status of one or more VCCs on the ATM interface.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`ifHandle`	Handle of the ATM interface
`n_Vccs`	Number of VCCs to be administratively enabled on the specified ATM interface
`Vcc_AddrArray`	Pointer to an array of VPI/VCI addresses

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_HANDLE`	`if_Handle` is null or invalid, or it is not an ATM interface
`NPF_IF_E_INVALID_VCC_ADDRESS`	Invalid ATM VCC address on the specified ATM interface

Asynchronous response

A total of `n_Vccs` asynchronous responses (`NPF_IfAsyncResponse_t`) are passed to the callback function, in one or more invocations. Each response contains an interface handle and a success code or a possible error code for that interface.

The union in the callback response structure contains the operational status of the interface and the VPI/VCI address of the relevant VCC if the code indicates success.

3.7 Read ATM VCC Statistics

Syntax

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

Description

This function returns, via a callback, a pointer to an ATM VCC statistics structure that contains the current counter values for each of one or more indicated ATM VCCs on a single ATM interface.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`if_Handle`	Handle of an interface of type `NPF_IF_TYPE_ATM`
`n_Vccs`	The number of VCCs for which statistics need to be obtained
`if_VccAddrArray`	Pointer to an array of VPI/VCI addresses

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_HANDLE`	`if_Handle` is null or invalid, or is not an ATM UNI interface
`NPF_IF_E_NO_SUCH_VCC`	Indicated VCC does not exist

Asynchronous response

A total of `n_Vccs` asynchronous responses (`NPF_IfAsyncResponse_t`) are passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code.

If the error code indicates success, the union in the response structure contains a pointer to a VCC statistics structure (`NPF_ATM_VccStats_t`). This structure contains the VCC's VPI/VCI address and its counter values.

In addition to the statistics attributes defined by the NPF's Interface Management API, the following ATM VCC statistics attributes can be fetched through `NPF_IfAtm_VccStatsGet API`:

- Ingress total cells (CLP0 + CLP1)
- Egress total cells (CLP0 + CLP1)
- Ingress AAL frames
- Egress AAL frames
- Ingress bytes
- Egress bytes

3.8 Add Connection Cross-connect

Syntax

```
NPF_error_t NPF_IfATM_ConnectionXcSet (
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_Vccs,
            NPF_IN NPF_ATM_ConnectionXc_t *connXc);
```

Description

This function allows a client to cross-connect ATM connection endpoints (VP cross-connects when VCI value is zero). Both the cross-connect endpoints must exist before invoking the API. Depending on the error reporting setting, an asynchronous callback can be invoked upon the completion of the call.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`if_Handle`	Handle of an interface of type `NPF_IF_TYPE_ATM`
`n_Vccs`	Number of ATM connection endpoint cross-connect information
`connXc`	Pointer to an array structures containing the ATM cross-connect information

Output Parameters

None.

Return Values

NPF_NO_ERROR	Operation successful
NPF_IF_E_INVALID_HANDLE	if_Handle is null or invalid, or is not an ATM interface
NPF_IF_E_NO_SUCH_VCC	Indicated VCC does not exist

Asynchronous Callback

A total of n_Vccs asynchronous responses (NPF_IfAsyncResponse_t) are passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code.

The union in the callback response structure contains cross-connect information to correlate the response with a specific request.

3.9 Delete Connection Cross-connect

Syntax

```
NPF_error_t NPF_ATM_ConnectionXcDelete (
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_Vccs,
            NPF_IN NPF_ATM_ConnectionXc_t *connXc);
```

Description

This function allows a client to disassociate one or more cross-connects of ATM connection endpoints. Depending on the error reporting setting, an asynchronous callback can be invoked upon completion of the call.

Input Parameters

if_cbHandle	Registered callback handle
if_cbCorrelator	Application's context for this call
if_errorReporting	Desired callback
if_Handle	Handle of an interface of type NPF_IF_TYPE_ATM
n_Vccs	Number of ATM connection endpoint cross-connect information
connXc	Pointer to an array structures containing the ATM cross-connect information

Output Parameters

None.

Return Values

NPF_NO_ERROR	Operation successful
NPF_IF_E_INVALID_HANDLE	if_Handle is null or invalid, or is not an ATM interface
NPF_IF_E_NO_SUCH_VCC	Indicated VCC does not exist

Asynchronous Callback

A total of n_Vccs asynchronous responses (NPF_IfAsyncResponse_t) are passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code. The union in the callback response structure contains the cross connect information, to correlate the response with the specific request.

Part 4: OAM Services

4 OAM Services

This section describes the set of functions required to operate and maintain the ATM layer aspects for permanent, semi-permanent, reserved, and on-demand virtual connections.

For maintenance purposes, F4 and F5 flows are defined in the ATM layer covering the VP and VC level, respectively. Both flows are bi-directional and follow the same physical route (VCC) as the user-data cells and constitutes an in-band maintenance flow.

Besides the vertical subdivision at F4 and F5 levels, a horizontal partition also exists. Both flows can either cover the entire virtual connection (end-to-end flow), or only parts of the virtual connection (segment flow).

Dedicated OAM cells with the pre-assigned VCI and PTI values are used to implement the F4 and F5 flows at segment or end-to-end levels respectively.

This section specifies the APIs for realization (using different OAM cell types) of the following functions:

- Fault management, using AIS, RDI, CC and LB cells
- Performance management, using FPM and BR cells
- Activation/deactivation of PM and/or CC, using activation/deactivation cells

4.1 Configure a Connection Point with OAM Attributes

Syntax

```
NPF_error_t NPF_IfATM_OAM_CP_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_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_CPs,
            NPF_IN NPF_ATM_OAM_CP_t       *oamCPArray);
```

Description

This function `NPF_IfATM_OAM_CP_Set` configures the connection points for the OAM services. The `NPF_IfATM_VccSet` must precede this operation to have the VCC configured. All the specified connection points are set with the OAM profile parameters, from the contents of an array of structures passed by the caller, as defined in `NPF_ATM_OAM_CP_t`.

OAM flows are related to bi-directional Maintenance Entities (MEs) corresponding either to the entire ATM VPC/VCC, referred to as the VPC/VCC ME, or to a portion of this connection referred to as a VPC/VCC segment ME.

Before the start of any OAM operation, the boundary needs to be drawn for the paired endpoints. The MEs terminating the ATM links are configured before as an endpoint of the VPC/VCC or endpoint of the VPC/VCC segment.

End-to-end F5 flows terminate at the endpoints of a VCC, while the segment F5 flows terminate at the VCC segment endpoints.

Similarly, the end-to-end F4 flows terminate at the endpoints of a VPC, while the segment F4 flows terminate at the VPC segment endpoints.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`if_Handle`	Handle of an interface of type `NPF_IF_TYPE_ATM`
`n_CPs`	Number of connection points
`oamCPArray`	Pointer to an array structures containing the OAM profile parameters for the specified connection point

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`:	Operation successful
`NPF_IF_E_INVALID_HANDLE`:	`if_Handle` is null or invalid, or is not an ATM UNI interface
`NPF_IF_E_NO_SUCH_CP`:	Indicated connection point does not exist

Asynchronous response

A total of `n_Vccs` asynchronous responses **(**`NPF_IfAsyncResponse_t`**)** are passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code. The union in the callback response structure contains the VPI/VCI address of the relevant connection point.

4.2 Activate, Deactivate, Start and Stop OAM Continuity Check

Syntax

```
NPF_error_t NPF_IfATM_OAM_CC_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_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_CPs,
            NPF_IN NPF_ATM_OAM_CC_t       *oamCCArray);
```

Description

This function `NPF_IfATM_OAM_CC_Set` activates and de-activates the CC cells on end-to-end and segment level on a certain number of active VCCs per interface in each direction (forward/backward). The forward direction is the direction followed by monitored user cells flow. CC can be established during connection establishment or at anytime after the connection has been established. The `NPF_IfATM_VccSet` must precede this operation to create the VCC.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`if_Handle`	Handle of an interface of type `NPF_IF_TYPE_ATM`
`n_CPs`	Number of CPs to monitor the OAM traffic
`oamCCArray`	Pointer to an array structure that contains the CPs to be initiated with CC cells

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`	Operation successful
`NPF_IF_E_INVALID_HANDLE`	`if_Handle` is null or invalid, or is not an ATM UNI interface
`NPF_IF_E_NO_SUCH_CP`	Indicated CP does not exist

Asynchronous response

A total of `n_CPs` asynchronous responses **(**`NPF_IfAsyncResponse_t`**)** are passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface,

and a success code or a possible error code. The union in the callback response structure contains the VPI/VCI address of the relevant CP.

4.3 Activate, Deactivate, Start and Stop OAM Performance Management

Syntax

```
NPF_error_t NPF_IfATM_OAM_PM_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_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_CPs,
            NPF_IN NPF_ATM_OAM_PM_t       *oamPMArray);
```

Description

This function `NPF_IfATM_OAM_PM_Set` activates and de-activates the PM (FPM and associated BR, FPM only) cells on an end-to-end and segment level on a certain number of active VCCs per interface in each direction (forward/backward).

The forward direction is the direction followed by monitored user cells flow. The PM can be established during connection establishment or at anytime after the connection has been established. The `NPF_IfATM_VccSet` must precede this operation to create the VCC.

Input Parameters

`if_cbHandle`	Registered callback handle
`if_cbCorrelator`	Application's context for this call
`if_errorReporting`	Desired callback
`if_Handle`	Handle of an interface of type `NPF_IF_TYPE_ATM`
`n_CPs`	Number of CPs to monitor the OAM traffic
`oamPMArray`	Pointer to an array structures containing the CPs and OAM cell attributes to enable the PM

Output Parameters

None.

Asynchronous Error Codes

NPF_NO_ERROR	Operation successful
NPF_IF_E_INVALID_HANDLE	if_Handle is null or invalid, or is not an ATM UNI interface
NPF_IF_E_NO_SUCH_CP	Indicated CP does not exist

Asynchronous response

A total of n_Vccs asynchronous responses (NPF_IfAsyncResponse_t) are passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface, and a success code or a possible error code. The union in the callback response structure contains the VPI/VCI address of the relevant CP.

4.4 Initiate an OAM Loopback Procedure

Syntax

```
NPF_error_t NPF_IfATM_OAM_LB_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_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_CPs,
            NPF_IN NPF_ATM_OAM_LB_t       *oamLBArray);
```

Description

This function NPF_IfATM_OAM_LB_Set initiates the LB procedure of inserting the LB cell without disrupting the sequence of the user cells and minimizes the user cell transfer delay. This LB cell is looped back at the downstream point. The NPF_IfATM_VccSet must precede this operation to create the VCC.

Input Parameters

if_cbHandle	Registered callback handle
if_cbCorrelator	Application's context for this call
if_errorReporting	Desired callback
if_Handle	Handle of an interface of type NPF_IF_TYPE_ATM
n_CPs	Number of CPs to monitor the OAM traffic
oamLBArray	Pointer to an array structures containing the OAM cell attributes to enable the LB

Output Parameters

None.

Asynchronous Error Codes

NPF_NO_ERROR	Operation successful
NPF_IF_E_INVALID_HANDLE	if_Handle is null or invalid, or is not an ATM UNI interface
NPF_IF_E_NO_SUCH_CP	Indicated CP does not exist

Asynchronous response

A total of n_Vccs asynchronous responses (NPF_IfAsyncResponse_t) are passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code. The union in the callback response structure contains the VPI/VCI address of the relevant CP.

4.5 Monitor OAM Cells

Syntax

```
NPF_error_t NPF_IfATM_OAM_Monitor(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_CPs,
            NPF_IN NPF_ATM_OAM_Mon_t      *oamMonArray);
```

Description

The function NPF_IfATM_OAM_Monitor, activates the non-intrusive monitoring of any type of end-to-end or segment fault and performance management OAM flows, without modifying the characteristics of the aggregate OAM flow.

The purpose of the non-intrusive monitoring function is to provide to the network providers additional OAM information, which cannot be derived from the content of the segment OAM flows.

For example, monitoring RDI and BR flows gives the possibility to assess, from any intermediate point, both the status (available/unavailable) and end-to-end performance. The NPF_IfATM_VccSet must precede this operation to create the VCC.

Input Parameters

`if_cbHandle`: Registered callback handle

`if_cbCorrelator`: Application's context for this call

`if_errorReporting`: Desired callback

`if_Handle`: Handle of an interface of type `NPF_IF_TYPE_ATM`

`n_CPs`: Number of CPs to monitor the OAM traffic

`oamMonArray`: Pointer to an array structures containing the OAM cell types to be monitored

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`: Operation successful

`NPF_IF_E_INVALID_HANDLE`: `if_Handle` is null or invalid, or is not an ATM UNI interface

`NPF_IF_E_NO_SUCH_CP`: Indicated CP does not exist

Asynchronous response

A total of `n_Vccs` asynchronous responses **(**`NPF_IfAsyncResponse_t`**)** is passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code. The union in the callback response structure contains the VPI/VCI address of the relevant CP.

4.6 Enable AIS/RDI Alarms

Syntax

```
NPF_error_t NPF_IfATM_OAM_AlarmSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_CPs,
            NPF_IN NPF_ATM_OAM_Alarm_t    *oamAlarmArray);
```

Description

The function `NPF_IfATM_OAM_AlarmSet`, enables the alarm state. The connection point must have received the AIS or the RDI.

Input Parameters

`if_cbHandle`: Registered callback handle

`if_cbCorrelator`: Application's context for this call

`if_errorReporting`: Desired callback

`if_Handle`: Handle of an interface of type `NPF_IF_TYPE_ATM`

`n_CPs`: Number of CPs to monitor the OAM traffic

`oamAlarmArray`: Pointer to an array structures containing the OAM alarms at CPs to set

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`: Operation successful

`NPF_IF_E_INVALID_HANDLE`: `if_Handle` is null or invalid, or is not an ATM UNI interface

`NPF_IF_E_NO_SUCH_CP`: Indicated CP does not exist

Asynchronous response

A total of `n_Vccs` asynchronous responses **(**`NPF_IfAsyncResponse_t`**)** is passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code. The union in the callback response structure contains the VPI/VCI address of the relevant CP.

4.7 Disable AIS/RDI Alarms

Syntax

```
NPF_error_t NPF_IfATM_OAM_AlarmClear(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_CPs,
            NPF_IN NPF_ATM_OAM_Alarm_t    *oamAlarmArray);
```

Description

This function `NPF_IfATM_OAM_AlarmClear` clears the AIS/RDI alarms received at the corresponding connection point.

Input Parameters

`if_cbHandle`: Registered callback handle

`if_cbCorrelator`: Application's context for this call

`if_errorReporting`: Desired callback

`if_Handle`: Handle of an interface of type `NPF_IF_TYPE_ATM`

`n_CPs`: Number of CPs to monitor the OAM traffic

`oamAlarmArray`: Pointer to an array structures containing the OAM alarms at the CPs to clear

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`: Operation successful

`NPF_IF_E_INVALID_HANDLE`: `if_Handle` is null or invalid, or is not an ATM interface

`NPF_IF_E_NO_SUCH_CP`: Indicated CP does not exist

Asynchronous response

A total of `n_CP`s asynchronous responses (`NPF_IfAsyncResponse_t`) is passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code. The union in the callback response structure contains the VPI/VCI address of the relevant CP.

Part 5: AAL2 Channel Management

5 AAL2 Channel Management

The AAL type 2 makes use of the service provided by the underlying ATM layer. Multiple AAL connections can be associated with a single ATM layer connection, allowing multiplexing at the AAL in the Common Part Sublayer (CPS). The AAL user selects the QoS provided by the ATM layer and there exists no standardized means to provide QoS at the AAL type 2 layer.

This section specifies the APIs for AAL2 channel management.

5.1 Create a AAL2 Channel

Syntax

```
NPF_error_t NPF_IfATM_AAL2_ChannelSet(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_Channels,
            NPF_ATM_AAL2_ChannelInfo_t    *aal2ChannelArray);
```

Description

This function `NPF_IfATM_AAL2_ChannelSet` creates a new AAL2 channel on the specified interface. The `NPF_IfATM_VccSet` must precede this operation so as to have the AAL2 VCC configured. All the AAL2 channels are set with the profile parameters from the contents of an array of structures passed by the caller, as defined in the `NPF_ATM_AAL2_ChannelInfo_t`.

Input Parameters

`if_cbHandle`: Registered callback handle

`if_cbCorrelator`: Application's context for this call

`if_errorReporting`: Desired callback

`if_Handle`: Handle of an interface of type `NPF_IF_TYPE_ATM`

`n_Channels`: Number of AAL2 channels

`aal2ChannelArray`: Pointer to an array structures containing the AAL2 channel profile on the specified AAL2 VCC connection.

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`: Operation successful

`NPF_IF_E_INVALID_HANDLE`: `if_Handle` is null or invalid, or is not an ATM interface

`NPF_IF_E_NO_SUCH_VCC`: Indicated VCC does not exist

Asynchronous response

A total of `n_Channels` asynchronous responses (`NPF_IfAsyncResponse_t`) is passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code. The union of the callback response structure contains the AAL2 channel IDs of the relevant AAL2 channels.

5.2 Delete an AAL2 Channel

Syntax

```
NPF_error_t NPF_IfATM_AAL2_ChannelDelete(
            NPF_IN NPF_callbackHandle_t   if_cbHandle,
            NPF_IN NPF_correlator_t       if_cbCorrelator,
            NPF_IN NPF_errorReporting_t   if_errorReporting,
            NPF_IN NPF_IfHandle_t         if_Handle,
            NPF_IN NPF_uint32_t           n_Channels,
            NPF_IN NPF_ATM_AAL2_ConnId_t  *connIdArray);
```

Description

This function `NPF_IfATM_AAL2_ChannelDelete` deletes an AAL2 channel on the specified interface. The `NPF_IfATM_AAl2_ChannelSet` must precede this operation so as to have the AAL2 VCC configured. The AAL2 channels to be deleted are specified in the contents of an array of structures passed by the caller, as defined in the `NPF_ATM_AAL2_ConnId_t`.

Input Parameters

`if_cbHandle`: Registered callback handle

`if_cbCorrelator`: Application's context for this call

`if_errorReporting`: Desired callback

`if_Handle`: Handle of an interface of type `NPF_IF_TYPE_ATM`

`n_Channels`: Number of AAL2 channels

`connIdArray`: Pointer to an array structures containing the AAL2 CID multiplexed on an ATM VCC of the specified interface

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`: Operation successful

`NPF_IF_E_INVALID_HANDLE`: `if_Handle` is null or invalid, or is not an ATM interface

`NPF_IF_E_NO_SUCH_CID`: Indicated CID does not exist

Asynchronous response

A total of `n_Channels` asynchronous responses (`NPF_IfAsyncResponse_t`) is passed to the callback function, in one or more invocations. Each response contains the interface handle of the ATM interface and a success code or a possible error code. The union of the callback response structure contains the AAL2 channel IDs of the relevant AAL2 channels.

Part 6: ATM Event Handler Function

6 ATM Event Handler Function

Syntax

```
typedef void (*NPF_ATMEventHandlerFunc_t) (
               NPF_IN NPF_userContext_t   userContext,
               NPF_IN NPF_ATMEventArray_t atmEventArray);
```

Description

The ATM Interface Management API allows registration for the ATM related events per VCC via the Event handler function. One or more events can be notified through the single invocation of the event handler function.

Information on each event is represented in an array in the `atmEventArray` structure, where application can traverse through the array and the process of each of the events. This event handler function is intended to be implemented by the application, and is registered to the API implementation through the `NPF_ATMEventRegister()` function.

Input Parameters

`userContext`: The context item that is supplied by the application during the registration of the event handler function.

`atmEventArray`: Data structure that contains an array of event information. Refer to `NPF_ATMEventArray_t` for details

Output Parameters

None.

Return Values

None.

6.1 ATM Event Handler Registration Function

Syntax

```
NPF_error_t NPF_ATMEventRegister (
      NPF_IN NPF_userContext_t            userContext,
      NPF_IN NPF_ATMEventHandlerFunc      atmEventHandlerFunc,
      NPF_IN NPF_IfHandle_t               ifHandle,
      NPF_IN NPF_uint32_t                 n_Vccs,
      NPF_IN NPF_VccAddr_r                *vcc_AddrArray,
      NPF_OUT NPF_ATMEventHandlerHandle_t *atmEventHandlerHandle);
```

Description

This function is used by an application to register its event handler function for receiving asynchronous ATM event notifications. The `NPF_ATMEventRegister` registers for ATM events on a specific ATM VCC (`NPF_VccAddr_t`) of an interface (`ifHandle`). The `ifHandle` and `vcc_AddrArray` together define a unique ATM connection across the multiple ATM interface.

Input Parameters

`userContext`:	Context item that is supplied by the application during the registration of the event handler function
`atmEventHandlerFunc`:	Pointer to the event handler function, to be registered
`ifHandle`:	Handle of an `NPF_IF_TYPE_ATM` type interface
`n_Vccs`:	The number of VCCs registering for an ATM event on a particular interface
`vcc_AddrArray`:	Pointer to an array of VPI/VCI addresses

Output Parameters

`atmEventHandlerHandle`:	Unique ID assigned for the registered `userContext` and `atmEventHandlerFunc` pair. This handle is used by the application for de-registering the `userContext` and the `atmEventHandlerFunc` pair.

Return Codes

`NPF_NO_ERROR`:	Registration completed successfully
`NPF_ATM_E_BAD_CALLBACK_HANDLE`:	`ifEventHandlerFunc` is `NULL` or not recognized
`NPF_ATM_E_ALREADY_REGISTERED`:	No new registration is made since the `userContext` and the `ifEventhandlerFunc` pair is registered already

Note: This should be treated as an error or not is dependent on the application.

6.2 ATM Event Handler De-registration Function

Syntax

```
NPF_error_t NPF_ATMEventDeregister (
                NPF_OUT NPF_ATMEventHandlerHandle_t
*atmEventHandlerHandle);
```

Description

This function is used by an application to de-register a pair of `userContext` and Event Handler function.

Input Parameters

`atmEventHandlerHandle`:	Unique ID, representing the pair of `userContext` and event handler function, (`atmEventHandlerFunc`) to be de-registered.

Output Parameters

None.

Return Codes

NPF_NO_ERROR:	**De-registration** completed successfully
NPF_E_BAD_CALLBACK_HANDLE:	ifEventHandlerFunc is NULL or not recognized

Part 7: Data Structures

7 Data Structures

The following sections describe the `typedefs` and `constants` that apply to the NPF ATM Interface Management API.

7.1 Data Structures for ATM APIs

```
/*
* ATM Interface Attributes, used in the union within
* NPF_IfGeneric_t.
*/
typedef struct {
NPF_uint32_t      port;           /* Port number */
NPF_ATM_Address_t atmAddress;     /* ATM address */
NPF_uint16_t      max_VpiBits,    /* Max VPI bits */
NPF_uint16_t      max_VciBits,    /* Max VCI bits */
NPF_ATM_Inf_t     atmInf;         /* ATM interface (UNI/NNI) */
NPF_ATM_InfType_t infType;        /* Interface Type (Public/Private)
*/
NPF_ATM_DevType_t devType;        /* UNI Device Type(End User/ N/w
Node) */
NPF_uchar8_t      rxCpId[16];     /* Ingress CPID */
NPF_uchar8_t      txCpId[16];     /* Egress CPID */
} NPF_IfATM_t;

/*
* ATM Address:
*   E.164 Address.
*   NSAP Address.
*/
typedef struct {
NPF_uint32_t addrCh;          /* Address Choice */
union {
NPF_E164_Address e164Address;       /* E.164 ATM address */
NPF_NSAP_Address nsapAddress;       /* NSAP ATM address */
}u;
} NPF_ATM_Address_t;

/*
* Interface – UNI/NNI
*/
typedef enum {
      NPF_NNI = 0
NPF_UNI = 1
} NPF_InfType_t;
```

```
/*
* UNI Device Type – User Side/Network Side
*/
typedef struct {
      NPF_USER = 0,
      NPF_NETWORK = 1
} NPF_DevType_t;
```

The following modifications need to be done for the VCC stats over the NP Forum's standard `NPF_IfVccStatsGet` API:

```
/*
* ATM UNI Per-Vcc Statistics, returned in asynchronous
* response from NPF_IfVccStatsGet().
*/
typedef struct {
NPF_VccAddr_t vccaddr;         /* VPI/VCI to which stats apply */

NPF_uint64_t cellsClp1Rx;      /* Ingress Total Cells (CLP0 + CLP1)
*/
NPF_uint64_t cellsClp0Rx;      /* Ingress High Priority Cells (CLP0)
*/
NPF_uint64_t cellsTaggedRx;    /* Ingress Cells changed from CLP0 to
CLP1 */
NPF_uint64_t cellsDiscardedRx; /* Ingress Cells discarded */

NPF_uint64_t cellsClp1Tx;      /* Egress Total Cells (CLP0 + CLP1) */
NPF_uint64_t cellsClp0Tx;      /* Egress High Priority Cells (CLP0)
*/

NPF_uint64_t framesRx;         /* Ingress AAL Frames */
NPF_uint64_t framesDiscardedRx;/* Ingress AAL Frames Discarded */
NPF_uint64_t framesTx;         /* Egress AAL Frames */

NPF_uint64_t bytesRx;          /* Ingress Bytes */
NPF_uint64_t bytesTx;          /* Egress Bytes */
} NPF_ATM_VccStats_t;

/*
* Service Category
* The service category of this virtual channel connection.
*/
typedef enum {
NPF_OTHER   =    0,
NPF_CBR     =    1,
NPF_rtVBR   =    2,
NPF_nrtVBR  =    3,
```

```
NPF_ABR     =    4,
NPF_UBR     =    5,
NPF_GFR     =    6
} NPF_ServiceCategory_t


NPF_IfATM_QoS_t needs to be modified to carry more information.
/*
* ATM UNI QOS profile -- can be shared by multiple Vccs,
* used in the ATM Vcc attribute structure, NPF_IfVcc_t.
*/
typedef struct {
NPF_ServiceCategory_t serviceCategory;    /* Service Category */
NPF_uint32_t rxPCR1;          /* Ingress Peak Cell Rate (CLP = 0+1)
*/
NPF_uint32_t rxPCR0;          /* Ingress Peak Cell Rate (CLP = 0) */
NPF_uint32_t rxSCR1 ;         /* Ingress Sustainable Cell
Rate(CLP=0+1) */
NPF_uint32_t rxSCR0;          /* Ingress Sustainable Cell Rate(CLP=0)
*/
NPF_uint32_t rxMBS1;          /* Ingress Maximum Burst Size(CLP=0+1)
*/
NPF_uint32_t rxMBS0;          /* Ingress Maximum Burst Size (CLP=0)
*/
NPF_uint32_t rxMCR;           /* Ingress Minimum Cell Rate */
NPF_uint32_t rxMaxFrmSize;    /* Ingress Maximum Frame Size */
NPF_uint32_t rxCTD;           /* Ingress Cell Transfer Delay */
NPF_uint32_t rxCDV;           /* Ingress Peak-to-Peak Cell Delay
Variation */
NPF_uint32_t rxCLR;           /* Ingress Acceptable Cell Loss Ratio
*/
NPF_uint32_t rxCDVT;          /* Ingress Cell Delay Variation
Tolerance */
NPF_uint16_t rxQosClass;      /* QoS Class */
NPF_boolean_t rxFrameDiscard; /* Receive Frame Discard */
NPF_boolean_t rxTagging       /* Tagging Option */


NPF_uint32_t  txPCR1;         /* Egress Peak Cell Rate (CLP = 0+1) */
NPF_uint32_t  txPCR0;         /* Egress Peak Cell Rate (CLP = 0) */
NPF_uint32_t  txSCR1;         /* Egress Sustainable Cell
Rate(CLP=0+1) */
NPF_uint32_t  txSCR0;         /* Egress Sustainable Cell Rate(CLP=0)
*/
NPF_uint32_t  txMBS1;         /* Egress Maximum Burst Size (CLP=0+1)
*/
NPF_uint32_t  txMBS0;         /* Egress Maximum Burst Size (CLP=0) */
NPF_uint32_t  txMCR;          /* Egress Minimum Cell Rate */
NPF_uint32_t  txMaxFrmSize;   /* Egress Maximum Frame Size */
NPF_uint32_t  txCTD;           /* Egress Cell Transfer Delay */
```

```
NPF_uint32_t  txCDV;          /* Egress Peak-to-Peak Cell Delay
Variation */
NPF_uint32_t  txCLR;          /* Egress Acceptable Cell Loss Ratio */
NPF_uint32_t  txCDVT;         /* Egress Cell Delay Variation
Tolerance */
NPF_uint16_t  txQosClass      /* QoS Class */
NPF_boolean_t txFrameDiscard; /* Transmit Frame Discard */
NPF_boolean_t txTagging;      /* Egress Tagging Option */

NPF_boolean_t bestEffIndicator;/* Best Effort Indicator */

} NPF_IfATM_QoS_t;


/*
* AAL Profile
*/
typedef union {
NPF_AAL1Profile_t   aal1Profile,
NPF_AAL2Profile_t   aal2Profile,
NPF_AAL34Profile_t  aal34Profile,
NPF_AAL5Profile_t   aal5Profile
} NPF_IfAAL_Profile;


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


/*
* AAL1 Profile
*/
typedef struct {
NPF_AAL1Subtype_t         subtype,            /* AAL1 sub-type */
NPF_CBR_t                 cbrRate,            /* rate of CBR service
*/
NPF_uint32_t              rateMultiplier,     /* Rate multiplier */
```

```
NPF_AAL1ClkRecoveryType_t clkRecoveryType,     /* Clock Recovery Type
*/
NPF_AAL1FEC_t             fecType,             /* Error Correction
Method */
NPF_boolean_t             sdtSupport,          /* Whether SDT
configured */
NPF_boolean_t             sdtBlockSize,        /* SDT Block Size */
NPF_uint32_t              partiallyFilledCells,/* No of part. filled
cells */
NPF_uint32_t              cellLossIntegrPeriod
} NPF_AAL1Profile_t

/*
*   AAL type 1 subtype used by the CBR service application (e.g. 64
*   KBPS voice band signal transport, circuit transport)
*/
typedef enum {
NPF_NULL = 0,
NPF_VOICEBAND = 1,
NPF_CIRCUIT_EMULATION_SYNCHRONOUS = 2,
NPF_CIRCUIT_EMULATION_ASYNCHRONOUS =3,
NPF_HIGH_QUALITY_AUDIO = 4,
NPF_VIDEO = 5
} NPF_AAL1Subtype_t

/*
*   CBR Rate
*/
typedef enum {
    NPF_64_KBPS     = 0,
    NPF_1544_KBPS   = 1,
    NPF_6312_KBPS   = 2,
    NPF_32064_KBPS  = 3,
    NPF_44736_KBPS  = 4,
    NPF_97728_KBPS  = 5,
    NPF_2048_KBPS   = 6,
    NPF_8448_KBPS   = 7,
    NPF_34368_KBPS  = 8,
    NPF_139264_KBPS = 9,
    NPF_N64_KBPS    = 10,
    NPF_N8_KBPS     = 11
} NPF_CBR_t


/*  Clock recovery type :
*   Synchronous,
*   Asynchronous-SRTS(Synchronous Residual Time Stamp) or
```

```
*   Asynchronous-Adaptive Clock Recovery.
*/
typedef enum {
            NPF_SYNCHRONOUS = 0,
            NPF_SRTS = 1,
            NPF_ADAPTIVE = 2
} NPF_AAL1ClkRecoveryType_t


/*   FEC method:
*     no FEC,
*     FEC for Loss Sensitive Signal Transport or
*     FEC for Delay Sensitive
*/
typedef enum {
NPF_NO_FEC = 0,
NPF_LOSS_SENSITIVE_SIGNAL_FEC = 1,
NPF_DELAY_SENSITIVE_SIGNAL_FEC = 2
} NPF_AAL1FEC_t


/*
* AAL2 Profile
*/

typedef enum {
    NPF_AUDIO = 1,
    NPF_MULTIRATE = 2,
    NPF_AUDIO_AND_MULTIRATE =3
} NPF_AAL2SscsServiceCategory

typedef enum {
{
    NPF_ITUT  = 1,
    NPF_OTHER = 2
} NPF_AAL2SscsProfileSource_t

typedef enum {
   NPF_ALAW  = 1,
   NPF_ULAW = 2
} NPF_AAL2SscsPcmEncoding_t

typedef struct{

/* Common AAL2 CPS Parameters I.363.2 */
```

```
NPF_uint32_t    cpsMaxMultiplexedChannels, /* Max mux channels. */
NPF_uint32_t    cpsMaxSduLength,           /* Maximum CPS-SDU size */
NPF_uint32_t    cpsTimer_CU,               /* Combined Use Timer_CU
*/
NPF_SscsType_t  sscsType                   /* SSCS Type */

NPF_uint32_t    appId,    /* Application Identifier */


/* Common SSCS Parameters (SSCS type = SAR)as specified in I.366.1 */

NPF_boolean_t sstedStatus,        /* SSTED selected */
NPF_boolean_t ssadtStatus,        /* SSADT selected */
NPF_uint32_t  maxSssarSduLength,  /* Max SSSAR-SDU size*/
NPF_uint32_t  maxSssarSegLength,  /* Max SSSAR-PDU size
                                     (1..MaxCPSDeliverLength)*/
NPF_uint32_t  rasTimer,           /* RAS Timer */

/* Common SSCS Parameters (SSCS type = Trunking) as specified in
I.366.2 */


NPF_AAL2SscsServiceCategory srvCategory,   /* Service category  */
NPF_boolean_t circuitModeDataTransport,    /* Circuit mode data
support */
NPF_boolean_t frameModeDataTransport,      /* Frame mode data support
*/
NPF_boolean_t faxDemodTransport,           /* demodulated fax data
support */
NPF_boolean_t casTransport,                /* CAS support */
NPF_boolean_t dtmfDigitPacketTransport,    /* DTMF dialed digit
support */
NPF_boolean_t mfR1DigitPacketTransport,    /* MF-R1 dialed digit
support */
NPF_boolean_t mfR2DigitPacketTransport,    /* MF-R2 dialed digit
support */
NPF_boolean_t audioServiceTransport,       /* Audio Transport */
NPF_AAL2SscsPcmEncoding_t pcmEncoding,     /* PCM encoding type */
NPF_uint32_t maxFrameModeDataLength,       /* Max I.366.2 frame data
len */
NPF_AAL2SscsProfileSource_t profileSource, /* profile source */
NPF_uint32_t predefinedProfileIdentifier,  /* predefined profile id
*/
NPF_uint32_t  ieeeOui,                     /* IEEE OUI */
NPF_uint32_t circuitModeDataNumChannels,   /* Multiplier N in
N*64kbit/s circuit mode data */


/* Common SSCOP Parameters as specified in Q.2110 */
```

```
NPF_uint32_t sscopSduLength,              /* Maximum SSCOP-SDU
length */
NPF_uint32_t sscopUuFieldLength           /* Maximum SSCOP-UU
length. */

/* ATM Trunking AAL2 Profile */
NPF_AAL2TrunkingProfileEntry_t aal2TrunkingProfile,

/* AAL2 CPS Profile for LES (af-vmoa-0145.000 shashank] */
NPF_AAL2LESProfileEntry_t aal2LESProfile,

} NPF_AAL2Profile_t


/* ATM Trunking AAL2 Profile */
typedef struct {
NPF_uint32_t vcci,              /* Uniquely identifies a VCC between
IWFs */
NPF_uint32_t sigVcci            /* VCCI to carry the CCS */
} NPF_AAL2TrunkingProfileEntry_t


/* ATM LES AAL2 Profile */
typedef struct {
NPF_uint32_t             cpsCIDLowerLimit, /* Min value the CID */
NPF_uint32_t             cpsCIDUpperLimit, /* Max value of CID */
NPF_AAL2CpsOptimisation_t cpsOptimisation,
} NPF_AAL2LESProfileEntry_t

typedef enum {
    NPF_SINGLE_CPS_PACKET_PER_CPS_PDU_NO_OVERLAP = 1,
    NPF_MULTIPLE_CPS_PACKETS_PER_CPS_PDU_WITH_OVERLAP = 2
} NPF_AAL2CpsOptimisation_t


/*
* AAL3/4 Profile
*/

typedef struct {
NPF_uint32_t    maxCpcsSduSizeForward,  /* Max i/c CPCS_PDU size */
NPF_uint32_t    maxCpcsSduSizeBackward, /* Max o/g CPCS_PDU size */
NPF_uint32_t    midRangeLow,            /* Low MID value for the VCC
*/
NPF_uint32_t    midRangeHigh,           /* High MID value for the VCC
*/
```

```
NPF_AALMode_t   aalMode,                /* AAL Mode */
NPF_SscsType_t  sscsType                /* SSCS Type */
}NPF_AAL3_4_Profile_t

/*
*  This attribute indicates whether the AAL for the supporting VCC
*  operating in message mode or streaming mode, assured or unassured
*/
typedef enum {
            NPF_MESSAGE_ASSURED = 0,
            NPF_MESSAGE_UNASSURED = 1,
            NPF_STREAMING_ASSURED = 2,
            NPF_STREAMING_UNASSURED = 3
} NPF_AALMode_t


/*
* SSCS type
*/
typedef enum{
        NULL                = 0,
        NPF_DATA_ASSURED    = 1,
        NPF_DATA_NON_ASSURED = 2,
        NPF_FRAME_RELAY     = 4,
        NPF_SSCF_CONS       = 8,
        NPF_SSCF_COTS       = 9,
        NPF_SAR             = 16,
        NPF_TRUNKING        = 17
} NPF_SscsType_t


/*
* AAL5 Profile
*/
typedef struct{
NPF_uint32_t     maxCpcsSduSizeForward, /* Max o/g CPCS_PDU size */
NPF_uint32_t     maxCpcsSduSizeBackward /* Max i/c CPCS_PDU size */
NPF_AALMode_t    aalMode,               /* AAL Mode */
NPF_SscsType_t   sscsType,              /* SSCS Type */
}NPF_AAL5Profile_t


/*
* ATM Cross-Connect
*/
typedef struct {
```

```
    NPF_IfHandle_t   sourceIf;        /* Source Interface */
    NPF_VccAddr_t    sourceVcc;       /* Source VPI/VCI */
    NPF_IfHandle_t   destinationIf;   /* destination interface */
    NPF_VccAddr_t    destinationVcc;  /* Destination VPI/VCI */
} NPF_ATM_ConnectionXc_t;


/*
* OAM CP Configuration
*/
typedef struct {
    NPF_VccAddr_t       connection;/* VPI/VCI */
    NPF_OAM_CP_Type_t   cpType;    /*
endpoint/segment/both/intermediate */
    NPF_OAM_Role_t      roleType;  /* Source/Sink/Both */
    NPF_OAM_Direction_t oamDir     /* OAM
direction(Forward/Backward/Both) */
} NPF_ATM_OAM_CP_t;


/*
* OAM CC Activation/Deactivation
*/
typedef struct {
    NPF_VccAddr_t         connection;  /* VPI/VCI */
    NPF_OAM_FlowType_t     flowType;   /* segment/end-to-end */
    NPF_OAM_OperType_t     operType;   /*
Activate/Deactivate/Start/Stop */
} NPF_ATM_OAM_CC_t;


/*
* OAM PM Activation/Deactivation – FPM-BR/FPM
*/
typedef struct {
    NPF_VccAddr_t          connection;  /* VPI/VCI */
    NPF_OAM_FlowType_t     flowType;    /* segment/end-to-end */
    NPF_OAM_FuncType_t     funcType;    /* FPM-BR/FPM */
    NPF_OAM_Direction_t    dir;         /* away/towards/both */
    NPF_OAM_BlkSize_t      fwdSize;     /* A-B block size */
    NPF_OAM_BlkSize_t      bwdSize;     /* B-A block size */
    NPF_OAM_OperType_t     operType;    /*
Activate/Deactivate/Start/Stop */
} NPF_ATM_OAM_PM_t;
```

```
/*
* OAM Loopback
*/
typedef struct {
    NPF_VccAddr_t          connection;  /* VPI/VCI */
    NPF_OAM_FlowType_t     flowType;    /* segment/end-to-end */
    NPF_OAM_LB_AppType_t   lbAppType;   /* Loopback Application Type
*/
    NPF_uchar8_t           llId[16];    /* Loopback location Id */
    NPF_boolean_t          remCell;     /* Remove Loopback Cells */
} NPF_ATM_OAM_LB_t;


/*
* OAM Non-Intrusive Monitoring
*/
typedef struct {
    NPF_VccAddr_t          connection;  /* VPI/VCI */
    NPF_OAM_FlowType_t     flowType;    /* segment/end-to-end */
    NPF_OAM_CellType_t     cellType;    /* OAM FM and PM Cell Type */
} NPF_ATM_OAM_Mon_t;


/*
* OAM Alarms - Declare and Release
*/
typedef struct {
    NPF_VccAddr_t         connection;   /* VPI/VCI */
    NPF_OAM_AlarmType_t   cellType;     /* OAM AIS/RDI Cell Type */
} NPF_ATM_OAM_Alarm_t;


/*
* OAM Alarm Type
*/
typedef enum{
NPF_IF_ATM_F4_SEG_AIS = 0,  /* F4 Segment AIS alarm */
NPF_IF_ATM_F4_SEG_RDI = 1,  /* F4 Segment RDI alarm */
NPF_IF_ATM_F4_EP_AIS  = 2,  /* F4 End Point AIS alarm */
NPF_IF_ATM_F4_EP_RDI  = 3,  /* F4 End Point RDI alarm */
NPF_IF_ATM_F5_SEG_AIS = 4,  /* F5 Segment AIS alarm */
NPF_IF_ATM_F5_SEG_RDI = 5,  /* F5 Segment RDI alarm */
NPF_IF_ATM_F5_EP_AIS  = 6,  /* F5 End Point AIS alarm */
NPF_IF_ATM_F5_EP_RDI  = 7,  /* F5 End Point RDI alarm */
} NPF_OAM_AlarmType_t
```

```
/*
* OAM Fault Management(AIS/RDI/CC/LB) and Performance
Management(FPM/BR)
* cell type
*/
typedef enum{
        NPF_AIS_CELL = 0,  /* Alarm Indication Signal Cell */
        NPF_RDI_CELL = 1,  /* Remote Defect Indicator Cell */
        NPF_CC_CELL  = 2,  /* Continuity Check Cell */
        NPF_LB_CELL  = 3,  /* Loopback Cell */
        NPF_FPM_CELL = 4,  /* Forward Performance Monitoring Cell */
        NPF_BR_CELL  = 5   /* Backward Reporting Cell */
} NPF_OAM_CellType_t


/*
* OAM Loopback Application Type
*/
typedef enum{
        NPF_END_TO_END   = 0,
        NPF_ACCESS_LINE  = 1,
        NPF_INTER_DOMAIN = 2,
        NPF_NET_TO_EP    = 3,
        NPF_INTRA_DOMAIN = 4,
} NPF_OAM_LB_AppType_t


/*
* OAM Flow Type
*/
typedef enum{
        NPF_SEGMENT = 0,
        NPF_END_TO_END = 1,
} NPF_OAM_FlowType_t


/*
* OAM Oper Type
*/
typedef enum{
        NPF_ACTIVATE    = 0,
        NPF_DEACTIVATE  = 1,
        NPF_START       = 2,
        NPF_STOP        = 3,
```

```
} NPF_OperType_t


/*
* OAM CP Type
*/
typedef enum{
        NPF_ENDPOINT             = 0,
        NPF_SEGMENT              = 1,
        NPF_ENDPOINT_AND_SEGMENT = 2,
        NPF_INTERMEDIATE         = 3
} NPF_OAM_CP_Type_t


/*
* OAM Role
*/
typedef enum{
        NPF_SOURCE          = 0,
        NPF_SINK            = 1,
        NPF_SOURCE_AND_SINK = 2
} NPF_OAM_Role_t


/*
* OAM PM Block Size
*/
typedef enum{
        NPF_SIZE_32768 = 0,
        NPF_SIZE_16384 = 1,
        NPF_SIZE_8192  = 2,
        NPF_SIZE_4096  = 3,
        NPF_SIZE_2048  = 4,
        NPF_SIZE_1024  = 5,
        NPF_SIZE_512   = 6,
        NPF_SIZE_256   = 7,
        NPF_SIZE_128   = 8
} NPF_OAM_BlkSize_t


/*
* OAM Direction
*/
typedef enum{
        NPF_FORWARD        = 0,    /* A-B */
        NPF_BACKWARD       = 1,    /* B-A */
```

```
        NPF_TWO_WAY        = 2     /* TWO-WAY */
        NPF_NOT_APPLICABLE = 3     /* NOT APPLICABLE */
} NPF_OAM_Direction_t


/*
*   AAL2 Connection ID.
*/
typedef struct {
NPF_VccAddr_t  connection;     /* VPI/VCI */
NPF_uint16_t   aal2Cid;        /* AAL2 Channel Id */
} NPF_AAL2_ConnId_t;


/*
*   AAL2 Connection Info.
*/
typedef struct {
NPF_AAL2_ConnId  aal2ConnId;             /* AAL2 Connection Id */

/* Specified in I.363.2 */
NPF_uint16_t     maxCpsSduDeliverLength; /* Maximum length of CPS SDU
*/
} NPF_ATM_AAL2_ChannelInfo_t;


/*
* Completion Callback Types, to be found in the callback
* data structure, NPF_IfCallbackData_t.
*/
typedef enum NPF_IfCallbackType {
NPF_IF_CREATE = 1,
NPF_IF_DELETE = 2,
NPF_IF_BIND = 3,
NPF_IF_STATS_GET = 4,
NPF_IF_VCC_STATS_GET = 5,
NPF_IF_ATTR_SET = 6,
NPF_IF_CREATE_AND_SET = 7,
NPF_IF_ENABLE = 8,
NPF_IF_DISABLE = 9,
NPF_IF_OPER_STATUS_GET = 10,
NPF_IF_LAN_SRC_ADDR_GET = 11,
NPF_IF_LAN_SRC_ADDR_SET = 12,
NPF_IF_LAN_ADDR_LIST_SET = 13,
NPF_IF_LAN_ADDR_LIST_ADD = 14,
```

```
NPF_IF_LAN_PROMISC_SET = 15,
NPF_IF_LAN_PROMISC_CLEAR = 16,
NPF_IF_LAN_FULL_DUPLEX_SET = 17,
NPF_IF_LAN_FULL_DUPLEX_CLEAR = 18,
NPF_IF_LAN_SPEED_SET = 19,
NPF_IF_LAN_FLOW_CONTROL_TX_ENABLE = 20,
NPF_IF_LAN_FLOW_CONTROL_TX_DISABLE = 21,
NPF_IF_LAN_FLOW_CONTROL_RX_ENABLE = 22,
NPF_IF_LAN_FLOW_CONTROL_RX_DISABLE = 23,
NPF_IF_IPV4ADDR_SET = 24,
NPF_IF_IPV4MTU_SET = 25,
NPF_IF_IPV4FIB_SET = 26,
NPF_IF_ATM_VCC_SET = 27,
NPF_IF_ATM_VCC_BIND = 28,
NPF_IF_ATM_VCC_DELETE = 29,

NPF_IF_ATM_VCC_ENABLE = 30,
NPF_IF_ATM_VCC_DISABLE = 31,
NPF_IF_ATM_VCC_OPER_STATUS_GET = 32,
NPF_IF_ATM_VCC_CROSSCONNECT_SET = 33,
NPF_IF_ATM_VCC_CROSSCONNECT_DELETE = 34,

NPF_IF_ATM_OAM_CP_SET = 35,
NPF_IF_ATM_OAM_CC_SET = 36,
NPF_IF_ATM_OAM_PM_SET = 37,
NPF_IF_ATM_OAM_LB_SET = 38,
NPF_IF_ATM_OAM_MONITOR = 39,
NPF_IF_ATM_OAM_ALARM_SET = 40,
NPF_IF_ATM_OAM_ALARM_CLEAR = 41,

NPF_IF_ATM_AAL2_CHANNEL_SET = 42,
NPF_IF_ATM_AAL2_CHANNEL_DELETE = 43
} NPF_IfCallbackType_t;
```

7.2 Data Structures for ATM Event Notification

```
/*
*   ATM Event Handler Function
*/
typedef void (*NPF_ATM_EventHandlerFunc_t) (
         NPF_IN NPF_userContext_t     userContext,
         NPF_IN NPF_ATM_EventArray_t  atmEventArray);
```

```
/*
*   ATM Event Types
*/
typedef enum {
NPF_IF_ATM_VCC_UP = 1,  /* VCC on the Interface is UP */
NPF_IF_ATM_VCC_DWN = 2, /* VCC on the Interface is DOWN */

NPF_IF_ATM_F4_SEG_AIS =  3,  /* F4 Segment AIS alarm */
NPF_IF_ATM_F4_SEG_RDI =  4,  /* F4 Segment RDI alarm */
NPF_IF_ATM_F4_EP_AIS  =  5,  /* F4 End Point AIS alarm */
NPF_IF_ATM_F4_EP_RDI  =  6,  /* F4 End Point RDI alarm */
NPF_IF_ATM_F5_SEG_AIS =  7,  /* F5 Segment AIS alarm */
NPF_IF_ATM_F5_SEG_RDI =  8,  /* F5 Segment RDI alarm */
NPF_IF_ATM_F5_EP_AIS  =  9,  /* F5 End Point AIS alarm */
NPF_IF_ATM_F5_EP_RDI  = 10,  /* F5 End Point RDI alarm */

NPF_IF_ATM_F4_SEG_CC = 11,   /* F4 Segment CC */
NPF_IF_ATM_F4_EP_CC  = 12,   /* F4 End Point CC */
NPF_IF_ATM_F5_SEG_CC = 13,   /* F5 Segment CC */
NPF_IF_ATM_F5_EP_CC  = 14,   /* F5 End Point CC */

NPF_IF_ATM_F4_SEG_FPM  = 15, /* F4 Segment FPM */
NPF_IF_ATM_F4_EP_FPM   = 16, /* F4 End Point FPM */
NPF_IF_ATM_F5_SEG_FPM  = 17, /* F5 Segment FPM */
NPF_IF_ATM_F5_EP_FPM   = 18, /* F5 End Point FPM */

NPF_IF_ATM_F4_SEG_BR = 19, /* F4 Segment BR */
NPF_IF_ATM_F4_EP_BR  = 20, /* F4 End Point BR */
NPF_IF_ATM_F5_SEG_BR = 21, /* F5 Segment BR */
NPF_IF_ATM_F5_EP_BR  = 22, /* F5 End Point BR */
} NPF_ATM_Event_t;


/*
*   AIS/RDI Fault Management Cell Info
*/
typedef struct {
NPF_uint16_t defectType,         /* Defect type */,
NPF_char8_t  defectLocation[16], /* Defect Location */
} NPF_OAM_AisRdi_EventInfo_t;


/*
*   Performance Management Cell Info
```

```
*/
typedef union {
   NPF_OAM_FPM_EventInfo_t   fpmStats;
   NPF_OAM_BR_EventInfo_t    brStats
} u;
} NPF_OAM_PM_EventInfo_t;


/*
*   FPM Cell Info
*/
typedef struct {
NPF_uchar8_t  mscn,           /* FPM Monitoring Cell Seq. No. */
NPF_uint16_t  totUsrCell1,    /* Total User Cell(CLP-0+1) */
NPF_uint16_t  totUsrCell0,    /* Total User Cell(CLP-0) */
NPF_uint16_t  blkErrDetCode,  /* Block Error Detection Code(CLP-0+1)
*/
NPF_uint32_t  timeStamp,      /* Time Stamp */
} NPF_OAM_FPM_EventInfo_t;


/*
*   BR Cell Info
*/
typedef struct {
NPF_uchar8_t  mscn,           /* BR Monitoring Cell Seq. No. */
NPF_uint16_t  totUsrCell1,    /* Total User Cell(CLP-0+1) */
NPF_uint16_t  totUsrCell0,    /* Total User Cell(CLP-0) */
NPF_uint32_t  timeStamp,      /* Time Stamp */
NPF_uchar8_t  repMscn,        /* Reported Monitoring Cell Seq. No. */
NPF_uchar8_t  secbc,          /* Severely Errored Cell Sequence
Number */
NPF_uint16_t  totRcvdUsrCell0,/* Total Received User Cell(CLP-0) */
NPF_uint8_t   blkErr,         /* Block Error Result(CLP-0+1) */
NPF_uint16_t  totRcvdUsrCell1,/* Total Received User Cell(CLP-0+1) */
} NPF_OAM_BR_EventInfo_t;


/*
*   Loopback Cell Info
*/
typedef struct {
NPF_uchar8_t  llId[16];          /* Loopback Location Id */
} NPF_OAM_LB_EventInfo_t;
```

```
/*
*   ATM Event Notification
*/
typedef struct {
NPF_ATMEvent_t eventType;     /* Event type */,
NPF_IfHandle_t if_handle;     /* Interface Handle */
NPF_VccAddr_r  vcc_Addr;      /* VPI/VCI */
union {
   NPF_OAM_AisRdi_EventInfo_t oamAisRdiInfo; /* AIS/RDI event info */
   NPF_OAM_PM_EventInfo_t     oamPMInfo;     /* FPM-BR/FPM event info
*/
   NPF_OAM_LB_EventInfo_t     oamLBInfo;     /* LB event info */
}u;
} NPF_ATM_EventData_t;

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