ATM Configuration Manager Functional API Implementation Agreement

April 11, 2005
Revision 1.0

Editor:

Vedvyas Shanbhogue, Intel, vedvyas.shanbhogue@intel.com

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

List of Tables

1 Revision History

Revision	Date	Reason for Changes
1.0	4/11/2004	Rev 1.0 of the ATM Configuration Manager Functional API Implementation Agreement. Source was npf2004.165.30.

2 Introduction

This document specifies the APIs for the ATM configuration Manager LFB. The ATM configuration manager LFB is the entity responsible for configuration and management of ATM LFBs on a FE. The ATM configuration manager provides APIs for

- ATM connection management
- AAL2 channel management
- ATM OAM services
- There are two types of ATM connections:
- Virtual Path Connections (VPC)
- Virtual Circuit Connections (VCC)

An ATM virtual connection (VPC or VCC) consists of series of connection virtual links between two end points. The cross connection of these virtual links is performed at the ATM switches along the path set up for the connection between the end points. The virtual links which are part of a VCC are called Virtual Channel (VC) links and the virtual links which are part of a VPC are called a Virtual Path (VP) link. Cross connects established (VP or VC cross connects) route the cells between the virtual links based on the VPI value (for VPC) or the VPI/VCI value (for VCC). When the VCC or VPC are provisioned by the network management functions, such VCC or VPC is called Permanent Virtual Connections (PVC). When the VCC or VPC is setup by means of a dynamic signaling procedure, such VCC or VPC are called Switched Virtual Connections (SVC). A Virtual Connection (SVC or PVC) can be a point-to-point or a point-to-multipoint connection. Each Virtual Connection (SVC or PVC) has associated with it a traffic descriptor. The virtual links which form this virtual connection inherit the traffic descriptors of the virtual connection. A Virtual Circuit Connection is illustrated by Figure 2.1: ATM Virtual Circuit Connection.

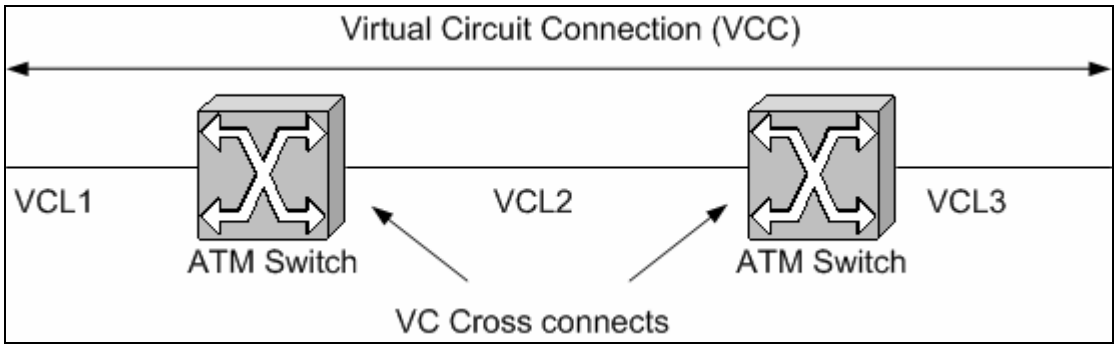

Figure 2.1: ATM Virtual Circuit Connection

A Virtual Path Connection is illustrated by Figure 2.2: ATM Virtual Path Connection.

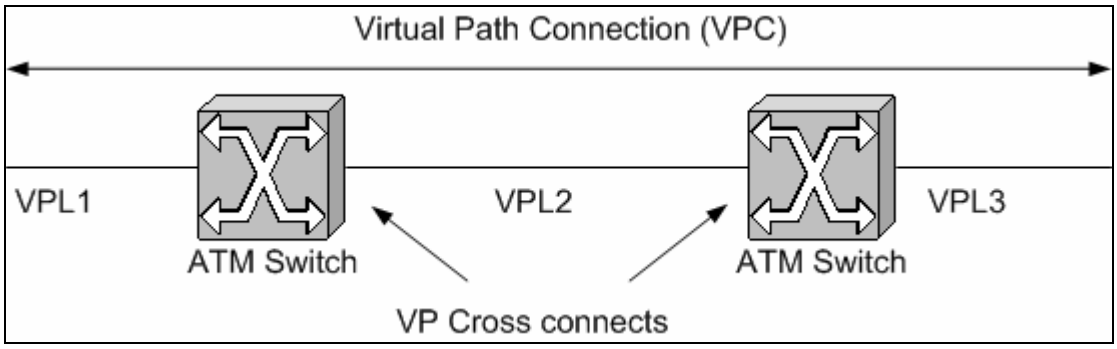

Figure 2.2: ATM Virtual Path Connection

2.1 Modeling the system

This specification defines the APIs at the Element Abstraction Layer in the NPF architecture for configuration and control of ATM objects on forwarding elements. The ATM Configuration Manager functional APIs hide less of the details of the system, exposing the presence of multiple forwarding devices and their individual capacities. However the ATM Configuration Manager functional APIs are vendor neutral, and a program using these APIs should be able to execute correctly on any conforming vendor's platform as long as the needed set of logical functions are present.

The physical realization of the NPF model may consist of multiple blades interconnected over a backplane in the system. In one realization this backplane could be a gigabit Ethernet connecting the blades in a rack mount system. In other realizations this backplane could be an ATM based switch fabric. Each forwarding element may terminate zero or more backplane interfaces. The backplane interfaces are identified by a unique identifier called the backplane interface ID. The traffic is carried between the blades in the system using an abstraction called the backplane switch link. The backplane switch links are cross connected to one or more external links on the blade. The external link could be an ATM virtual channel, ATM virtual path or an AAL2 channel.

Certain systems would need to create a VP/VC link or an AAL2 channel which is visible only within the forwarding element on which it is provisioned i.e. the scope of the link is limited to the FE on which it is created as opposed to external VP/VC links or AAL2 channels which are visible outside of the forwarding element over the ATM interface. To model such systems a concept of an internal and external link is introduced. An external link (VC link or AAL2 channel) is one which is visible external to the system and carries traffic across an ATM network interface. Internal links however may not be visible outside of the system and may be used to carry traffic only within the system.

These internal links may be cross connected to backplane links or to external links. One example of a cross connection to an external link would be where an external VC link is cross connected to an internal VC link with the internal VC link being bound to a packet handler interface. This is illustrated in using the example system below where in the external link #7 is cross connected to internal link #3 which is bound to a packet handler interface towards the host processor.

As discussed above setting up VPC/VCC would require two external VC or VP links to be cross connected. Similarly setting up an AAL2 CID cross connects would require two external AAL2 channels to be cross connected. The cross connect may be realized using an VP/VC cross connect LFB or an AAL2 channel cross connect LFB when the two external VP/VC links or channels to be cross connected are configured on the same forwarding element. When the external VP/VC links or AAL2 channels are configured on different forwarding elements, the cross connect is realized by using a backplane switch link

This concept is illustrated using the example system shown in 0 where two forwarding elements are connected over a backplane. To cross connect external link 1 on Forwarding element #2 to external link 4 on Forwarding element #1, a backplane link i.e. backplane link #1 is used as the interconnect. The

external links 2 and 3 on the forwarding element are locally interconnected on the Forwarding element #2.

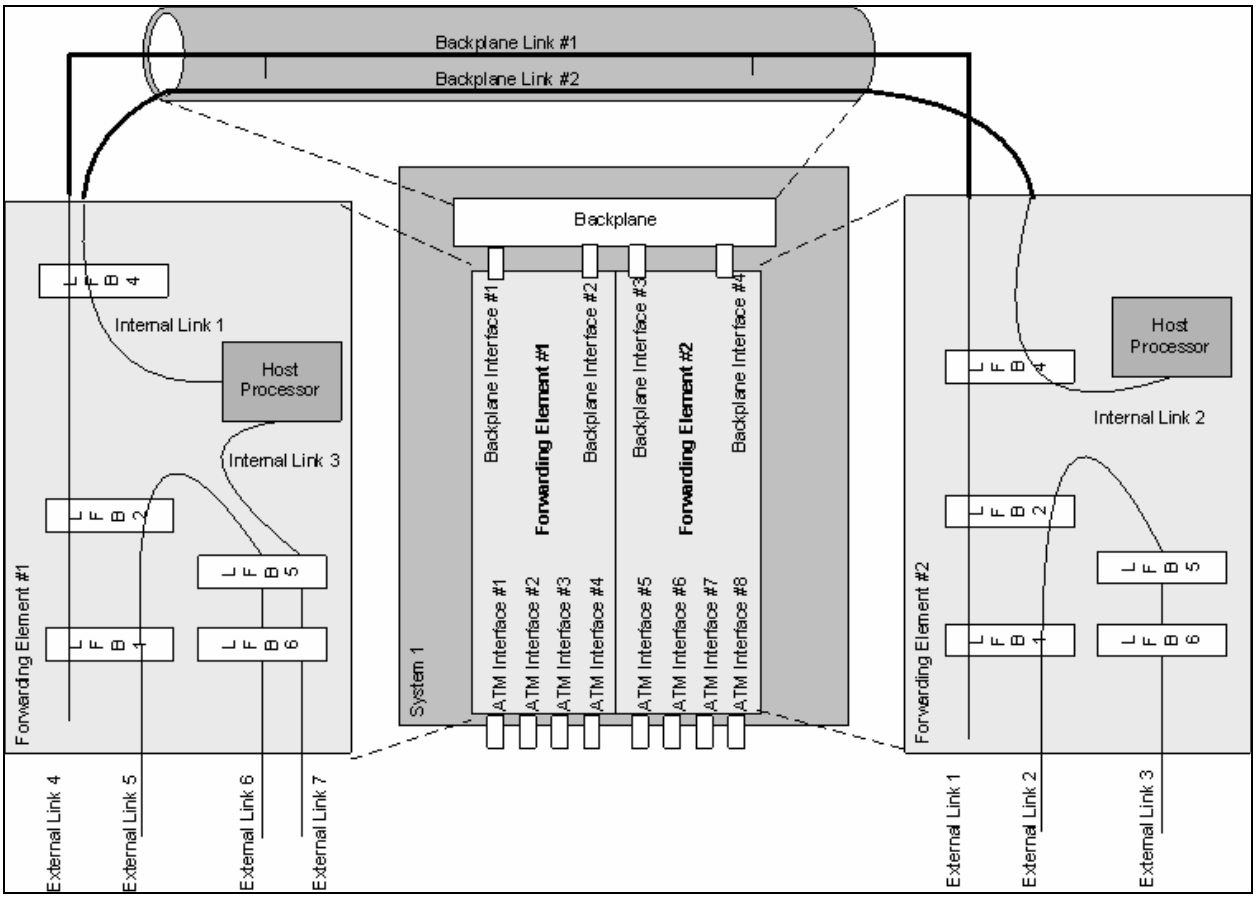

Figure 2.3: Example system configuration

Likewise the internal link 1 and internal link 2 are examples of VC links. These links do not terminate on any ATM interface. They are cross connected using a backplane switch link i.e. backplane link #2. In this example configuration, the internal link 1 and 2 are attached to a packet handler interface to carry the data received on these links to the host processor.

When an external link is cross connected to a backplane link, the payload received over the external link may be encapsulated with a header containing a backplane switch address to allow routing/switching over the backplane. The backplane link connects two end points ‘A’ and ‘B’ over the backplane. The headers used for sending data in the ‘A’ to ‘B’ direction may not be same as those used to send data in the ‘B’ to ‘A’ direction as illustrated by 0.

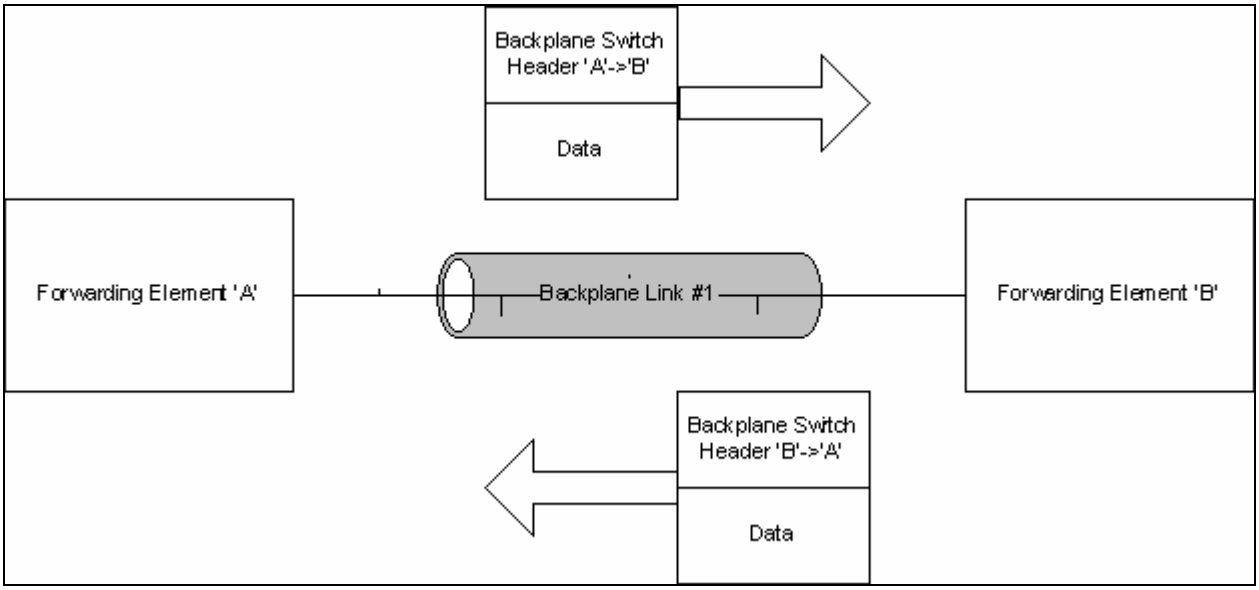

Figure 2.4: Backplane switch Headers

In order to allow such configurations the backplane switch link on a forwarding element it is associated with a pair of backplane switch addresses i.e. a receive switch address and a transmit switch address. When a FE sends data to the backplane, it creates the header using the transmit switch address configured for the backplane link on which the data is to be sent. When data is received from the backplane by an FE, the switch address in the header is used to determine the ID of the backplane link on which the data was received.

2.1.1 Data Directions

The conventions used when specifying the direction of data flow for external links and backplane links is as illustrated in 0.

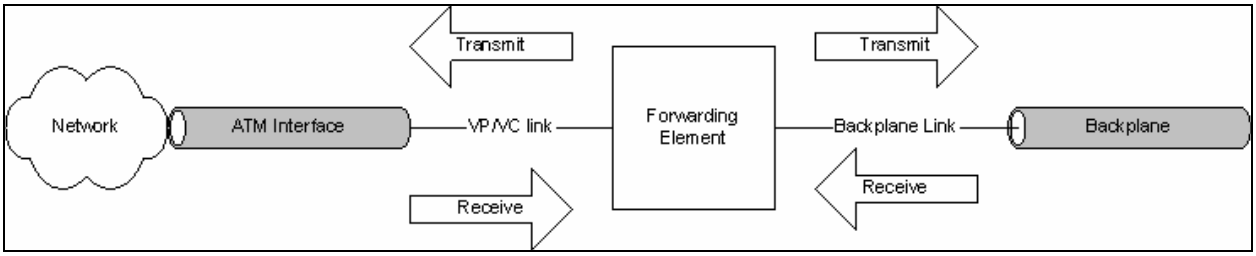

Figure 2.5: Data Flow Directions with reference to Forwarding Element

When a VP/VC link direction is configured as receive, the direction of traffic flow on that link is from the network to the forwarding element. When the VP/VC link direction is configured as transmit, the direction of traffic flow on that link is from the forwarding element to the network. When the VP/VC link direction is configured as duplex, the traffic flows in both directions i.e. from the network to the forwarding element as well as from the forwarding element to the network.
When a backplane switch link is configured as receive, the direction of traffic flow on that link is from the backplane to the forwarding element. When the backplane switch link is configured as transmit, the direction of the traffic flow on that link is from the forwarding element to the backplane. When the backplane switch link is configured as duplex, the traffic flows in both directions i.e. from the backplane to the forwarding element as well as from the forwarding element to the backplane.

2.2 Configuration Objects and Operations

The ATM configuration manager functional API is used to manipulate the following objects in the forwarding element:

- ATM Interfaces
- Virtual Channel (VC) Links
- Virtual Path (VP) Links
- AAL2 channels
- VC Link cross connect
- VP Link cross connect
- AAL2 channel cross connect

Each object is identified by a unique identifier (ID) that is assigned by the FAPI client. The objects ID can be assigned any value by the FAPI client and is completely opaque to the FAPI implementation. In general each object has the following basic type of operations defined:

Table 2.1: Operations on Configuration Objects

Operation	Description
`Set`	This operation is used to configure the object or to modify the characteristics of a previously configured object
`Delete`	This operation is used to delete a previously configured object
`Query`	This operation is used to query the configuration of a previously configured object. The API functions allow performing a bulk query to retrieve the configurations of all objects of a given type configured in the forwarding element.
`StatsGet`	Operation is used to query the statistics accumulated by the queried object
`StatsEnable`	Enable statistics collection for the specified object
`StatsDisable`	Disable statistics collection for the specified object

Additional operations in addition to these basic operations are defined as needed for the objects.

2.3 Acronyms / Definitions

- **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
- **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
- **ID:** Identifier
- **IP:** Internet Protocol
- **LB:** Loopback
- **LES:** Loop Emulation Service
- **MBS:** Maximum Burst Size
- **MCR:** Minimum Cell Rate
- **MFS:** Maximum Frame Size
- **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
- **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
- **VC:** Virtual Connection
- **VCC:** Virtual Channel Connection
- **VCI:** Virtual Channel Identifier
- **VPC:** Virtual Path Connection
- **VPI:** Virtual Path Identifier

2.4 Assumptions

None

2.5 Scope

This document covers ATM Configuration Manager related data structure definitions. The NPF Software Conventions Implementation Agreement Document [SWAPICON] documents data types and structures

generally used by all API specifications; however, specific structures are defined in this document. The document also specifies the syntax and semantics of the functions provided by the ATM configuration manager.

2.6 External Requirements and Dependencies

This document depends on the following documents:

- This document depends on the NPF Software API Conventions Implementation Agreement document [SWAPICON] for basic type definitions (Refer section 5.1 of Software API Conventions IA Revision 2.0).
- ATM Software API Architecture Framework defines the architectural framework for the ATM FAPIs
- This document depends on Interface Management API Implementation Agreement (Core Function Set) Revision 3.0 for below data types definitions:
    - `NPF_IfHandle_t` – Defined in section 3.1.3 of on Interface Management API Implementation Agreement (Core Function Set) Revision 3.0.
- This document depends on Software API Conventions Implementation agreement Revision 2.0 for below type definitions
    - `NPF_error_t` – Refer section 5.2 of Software API Conventions IA Rev 2.0
    - `NPF_callbackHandle_t` - Refer section 5.2 of Software API Conventions IA Rev 2.0
    - `NPF_callbackType_t` - Refer section 5.2 of Software API Conventions IA Rev 2.0
    - `NPF_userContext_t` - Refer section 5.2 of Software API Conventions IA Rev 2.0
    - `NPF_eventMask_t` - Refer section 5.2 of Software API Conventions IA Rev 2.0
    - `NPF_errorReporting_t` - Refer section 5.2 of Software API Conventions IA Rev 2.0
- This document depends on Topology Manager Functional API Implementation Agreement Revision 1.0 for below type definitions
    - `NPF_BlockId_t` – Refer section 3.1.1 of Topology Manager Functional API IA Rev 1.0
    - `NPF_FE_Handle_t` – Refer section 3.1.1 of Topology Manager Functional API IA Rev 1.0

3 ATM Configuration Manager Description

The ATM Configuration Manager provides an aggregation of the FAPI function calls provided by the ATM LFB to present a unified view to the ATM FAPI clients. The FAPI client invokes FAPI function calls provided by the ATM Configuration Manager. The ATM Configuration Manager then distributes the configurations to the underlying LFBs. The mechanism used to distribute such configurations is vendor specific and thus not in the scope of NPF.

ATM being connection oriented by nature requires that multiple LFBs in an FE be updated for a given circuit establishment. For example, establishment of a bi-directional AAL2 terminated virtual connection requires the following LFB data structures to be updated:

- ATM Header Classifier
- ATM OAM Receive
- ATM Policer
- AAL2 CPS Receive
- AAL2 SAR SSCS Receive
- AAL2 CPS Transmit
- AAL2 SAR SSCS Transmit
- ATM Traffic Manager
- ATM OAM Transmit
- ATM Header Generator

When these LFBs are located on the same FE, the number of function calls required to be made by the FAPI client to carry out operations (create, delete etc.) is optimized by having a ATM Configuration Manager that aggregates the parameters of the function calls to be invoked on individual LFBs to configure various ATM objects like connections, channels, interfaces etc.

3.1 ATM Configuration Manager Inputs

The ATM Configuration Manager has no packet/cell inputs and does not do any data path processing.

3.2 ATM Configuration Manager Outputs

The ATM Configuration Manager has no packet/cell outputs and does not do any data path processing.

3.3 Accepted Cell Types

The ATM Configuration Manager has no packet/cell outputs and does not do any data path processing

3.4 Cell Modifications

The ATM Configuration Manager has no packet/cell outputs and does not do any data path processing.

3.5 Relationship with Other LFBs

The relationship of the ATM Configuration Manager with other LFBs in the ATM group is as shown below:

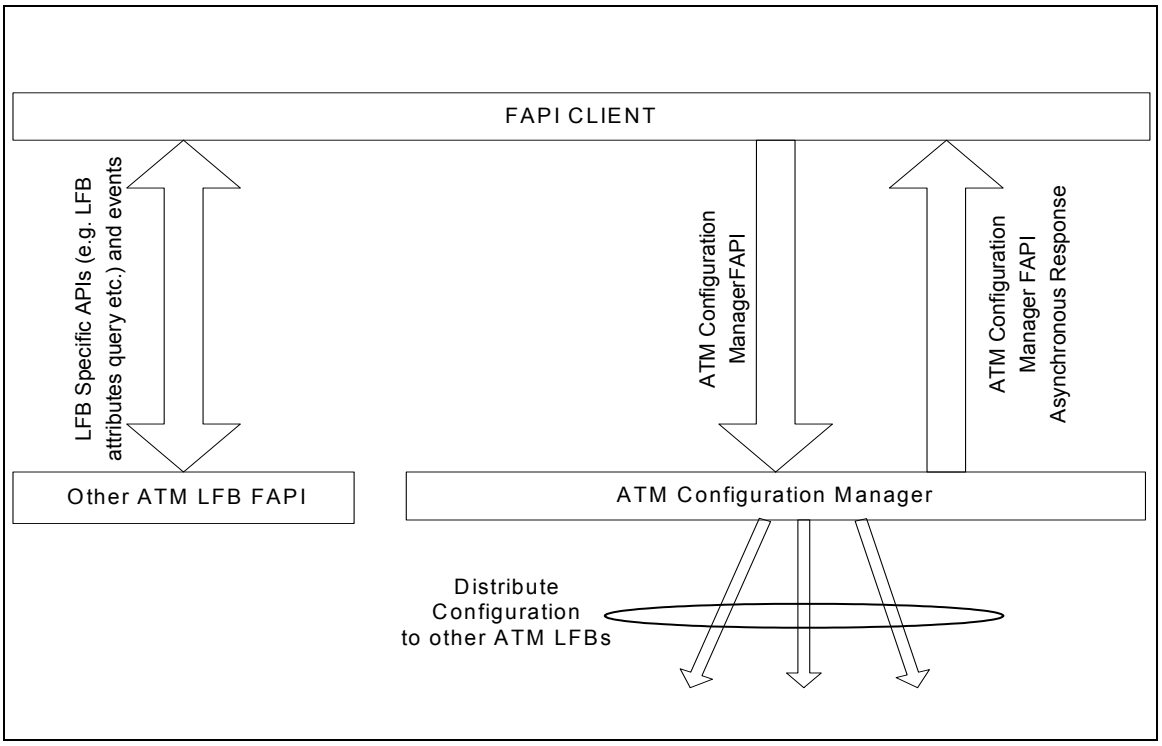

Figure 3.1: Relationship of ATM Configuration Manager to other ATM group LFBs

The ATM Configuration Manager exports APIs for carrying out specific operations like creating a connection, deleting a connection, querying statistics etc. In order to carry out the configuration, query or control operations required by a given ATM Configuration Manager functions one or more underlying ATM LFBs in the FE may need to be configured or their configuration queried.

The mechanism used by the ATM Configuration Manager to distribute configurations, query statistics or retrieve configurations – like messaging, shared memory etc. – from underlying ATM LFBs is not in the scope of NPF.

On successfully carrying out the requested operation, the ATM Configuration Manager invokes the completion callback into the FAPI client to indicate the result of the operation.

4 Data Types

4.1 Common LFB Data Types

4.1.1 Relationship with the Topology Discovery API

It is possible to use the FAPI Topology Discovery APIs to discover an ATM. Configuration Manager in a forwarding element using the block type value defined here.

```
#define NPF_F_ATMCONFIGMGR_LFB_TYPE 40   /* ATM Config. Mgr. LFB type code */
```

4.1.2 ATM Virtual Path Identifier

The simple data types below provide meaningful names for the VPI.

```
/*
* ATM VPI types
*/
typedef NPF_uint16_t NPF_F_ATM_VPI_t; /* VPI is 8 or 12 bits */
```

4.1.3 ATM Virtual Channel Identifier

The simple data types below provide meaningful names for the VCI.

```
/*
* ATM VCI types
*/
typedef NPF_uint16_t NPF_F_ATM_VCI_t; /* VCI is 16 bits      */
```

4.1.4 ATM Virtual Channel Address Types

The VPI and VCI in the ATM header are used to identify the VC on an interface.

```
typedef struct {          /* ATM Vc Address (VPI/VCI) structure */
    NPF_F_ATM_VPI_t vpi; /* VPI number */
    NPF_F_ATM_VCI_t vci; /* VCI number */
} NPF_F_ATM_VcAddr_t;
```

4.1.5 ATM Virtual Link Type

This section defines the ATM virtual link type which is used to identify a virtual link as a VP link or a VC link

```
typedef enum {
    NPF_F_ATM_VP_LINK = 0,       /* Virtual Path Link    */
    NPF_F_ATM_VC_LINK = 1        /* Virtual channel link */
} NPF_F_ATM_VirtLinkType_t;
```

4.1.6 Virtual link Scope

Virtual links which are visible outside of a forwarding element are distinguished from Virtual links which are not visible outside.

```
/* Scope of an ATM virtual link. Internal virtual links are not visible *
 * to the network and are used for transporting data within the system  */
typedef enum {
    NPF_F_ATM_VIRT_LINK_INTERNAL = 0,       /* Internal VP/VC link    */
    NPF_F_ATM_VIRT_LINK_EXTERNAL = 1        /* External VP/VC link    */
} NPF_F_ATM_VirtLinkScope_t;
```

4.1.7 ATM Virtual Link Identifier

This section defines the ATM virtual link identifier that is used to uniquely identify a virtual path or virtual channel link. The FAPI client can assign any value to the ATM virtual link identifier and the value is completely opaque to the implementation.

```
typedef NPF_uint32_t NPF_F_ATM_VirtLinkID_t;    /* Virtual Link ID      */

typedef struct {
    NPF_F_ATM_VirtLinkType_t atm_linkType;    /* VP link or VC link   */
    NPF_F_ATM_VirtLinkID_t   atm_linkID;      /* Virtual Link ID      */
} NPF_F_ATM_VirtLink_t;
```

4.1.8 Backplane Switch Link Identifier

This section defines the backplane interface identifier that is used to uniquely identify a backplane switch link in the system.

```
/* Identifies a backplane switch link */
typedef NPF_uint32_t NPF_F_ATM_BpLinkID_t;   /* Backplane Link ID    */
```

4.1.9 ATM Interface Identifier

The ATM interface identifier is used to uniquely identify an ATM interface in the system. The ATM interfaces are created using the Interface Management API functions. This variable is the ID assigned to the ATM interface by the Interface Management API.

```
typedef NPF_uint32_t NPF_F_ATM_IfID_t;         /* Interface ID      */
```

4.1.10 Backplane Interface Identifier

This section defines the backplane interface identifier that is used to uniquely identify a backplane interface in the system.

```
/* Identifies a backplane interface to a FE */
typedef NPF_uint32_t NPF_F_ATM_BpIfID_t;      /* Backplane Interface ID    */
```

4.1.11 AAL2 Channel Identifier

This section defines the AAL2 channel identifier that is used to uniquely identify an AAL2 channel. The FAPI client can assign any value to the AAL2 channel identifier and the value is completely opaque to the implementation.

```
typedef NPF_uint32_t NPF_F_AAL2ChanId_t;        /* AAL2 channel ID   */
```

4.1.12 Virtual Channel Cross Connect Identifier

This section defines the identifier that is used to uniquely identify a virtual channel cross connection. The FAPI client can assign any value to the virtual channel cross connect identifier and the value is completely opaque to the implementation.

```
typedef NPF_uint32_t NPF_F_VcXcId_t;        /* ATM VC Cross Connect ID   */
```

4.1.13 Virtual Path Cross Connect Identifier

This section defines the identifier that is used to uniquely identify a virtual path cross connection. The FAPI client can assign any value to the virtual path cross connect identifier and the value is completely opaque to the implementation.

```
typedef NPF_uint32_t NPF_F_VpXcId_t;        /* ATM VP Cross Connect ID   */
```

4.1.14 AAL2 Channel Cross Connect Identifier

This section defines the identifier that is used to uniquely identify an AAL2 channel cross connection. The FAPI client can assign any value to the AAL2 channel cross connect identifier and the value is completely opaque to the implementation.

```
typedef NPF_uint32_t NPF_F_AAL2ChnlXcId_t;   /* AAL2 channel Xconnect ID */
```

4.1.15 Timer Value

The FAPI client configures timers using a timer unit and a value. This data type provides enumerations for possible timer units.

```
typedef enum {
    NPF_F_ATM_TIME_UNIT_NS = 0,       /* Timers specified in nanoseconds  */
    NPF_F_ATM_TIME_UNIT_US = 1,       /* Timers specified in microseconds */
    NPF_F_ATM_TIME_UNIT_MS = 2,       /* Timers specified in milliseconds */
    NPF_F_ATM_TIME_UNIT_1S = 3        /* Timers specified in seconds      */
} NPF_F_ATM_TimerUnit_t;
```

4.1.16 Timer Configuration

The FAPI client configures timers using a timer unit and a value. This data type is used to configure timers.

```
typedef struct {
    NPF_F_ATM_TimerUnit_t  timeUnit;  /* Unit in which timer specified    */
    NPF_uint32_t           timeValue; /* Timer value in units specified   */
} NPF_F_ATM_Timers_t;
```

4.1.17 Object Status

This data type provides the possible values that can be assumed by the status of an object. This enumeration could be used to specify either the operational or administrative status of an object. This is a global type defined at NPF level

```
typedef enum {
    /* The below status can be used to denote operational or admin status
     * of an object, depending on the context in which they are used
     */
    NPF_STATUS_UP = 1,         /* Status marked UP                         */
    NPF_STATUS_DOWN = 2,       /* Status marked DOWN                       */
    NPF_STATUS_TESTING = 3,    /* Object in some test mode                 */

    /* The below status can be used to denote operational status of object */
    NPF_STATUS_UNKNOWN = 4,    /* Cannot be determined due to some reason  */
    NPF_STATUS_DORMANT = 5,    /* Ready but waiting for external action    */
    NPF_STATUS_NOT_PRESENT = 6,/* Object has missing components like h/w   */
    NPF_STATUS_LOWER_LYR_DOWN = 7 /* Lower layer down                      */
} NPF_ObjStatus_t;
```

4.1.18 Direction

The below data type is used to specify the direction of data flow on the configured link with respect to FE being configured.

```
/*
* Link direction
*/
typedef enum{
    NPF_F_ATM_RECEIVE  = 0,        /* link is receive  only */
    NPF_F_ATM_TRANSMIT = 1,        /* link is transmit only */
    NPF_F_ATM_DUPLEX = 2           /* link is bidirectional */
} NPF_F_ATM_Direction_t;
```

4.1.19 Backplane Switch Address

This data type is used to configure the headers to be used to encapsulate data (ATM SDU/AAL2 SDU) received on a VP/VC link or an AAL2 channel to be transported over the backplane used to connect forwarding elements. The format used to encapsulate data is out of the scope of NPF. The txAddress is used in the header to encapsulate the data when sending to the backplane. The data received from the backplane would be encapsulated with a header using the rxAddress and is used to

identify the backplane switch link on which the data is received. The rxBpIfId is the backplane interface on which the data is received from backplane switch link. The txBpIfId is the backplane interface on which data is transmitted to the backplane switch link.

```
/* Structure is used to pass the headers to be used to tunnel data         *
 * received over AAL2 channels or VP/VC links to backplane                 */
typedef struct {                              /* Switch Address structure     */
    NPF_F_ATM_BpLinkID_t  bpLinkID;           /* Backplane switch Link ID     */
    NPF_F_ATM_Direction_t direction;          /* Direction of data flow       */
    NPF_F_ATM_BpIfID_t    rxBpIfId;           /* Receive Backplane I/F ID     */
    NPF_uint32_t          rxaddressLength;    /* Length of the Switch Address */
    NPF_uint8_t           *rxAddress;         /* Array of octets containing   */
                                              /* the Switch Address.          */
    NPF_F_ATM_BpIfID_t    txBpIfId;           /* Transmit Backplane I/F ID    */
    NPF_uint32_t          txaddressLength;    /* Length of the Switch Address */
    NPF_uint8_t           *txAddress;         /* Array of octets containing   */
                                              /* the Switch Address.          */
} NPF_F_SwitchAddress_t;
```

The direction field usage and the relationship between the various parameters in the `NPF_F_SwitchAddress_t` structure are as described in the table below.

Table 4.1: Switch address configuration structure parameter usage

direction	rxBpIfId/ rxaddressLength/ rxAddress	txBpIfId/ txaddressLength/ txAddress
NPF_F_ATM_RECEIVE	Mandatory The backplane switch link is only used to receive data from the backplane. These fields are mandatory for the backplane switch link configuration.	Not applicable The backplane switch link is only used to receive data from the backplane. These fields are ignored in the NPF_F_SwitchAddress_t structure
NPF_F_ATM_TRANSMIT	Not applicable. The backplane switch link is only used to transmit data to the backplane. These fields are ignored in the NPF_F_SwitchAddress_t structure	Mandatory The backplane switch link is only used to transmit data to the backplane. These fields are mandatory for the backplane switch link configuration.
NPF_F_ATM_DUPLEX	Mandatory The backplane switch link is only used to receive data from the backplane. These fields are mandatory for the backplane switch link configuration.	Mandatory The backplane switch link is used to receive and transmit data to the backplane. These fields are mandatory for the

backplane switch link configuration.

4.2 ATM Traffic Management and Quality of Service Types

Providing differentiated quality of service (QoS) is a key element of traffic management in ATM networks that allows ATM networks to support a wide array of services and applications.

```
/*
* Service Category
* The service category of this virtual connection.
*/
typedef enum {
    NPF_F_ATM_OTHER_SRV_CAT = 0,       /* Unspecified service cat          */
    NPF_F_ATM_CBR = 1,                 /* Constant bit rate                */
    NPF_F_ATM_rtVBR = 2,               /* Variable bit rate - real time    */
    NPF_F_ATM_nrtVBR = 3,              /* Variable bit rate - non real time*/
    NPF_F_ATM_ABR = 4,                 /* Available bit rate               */
    NPF_F_ATM_UBR_WITHOUT_PCR = 5,     /* Unspecified bit rate-no peak rate*/
    NPF_F_ATM_UBR_WITH_PCR = 6,        /* Unspecified bit rate w/ peak rate*/
    NPF_F_ATM_UBR_WITH_MDCR = 7,       /* Unspecified bit rate w/ minimum  */
                                       /* Desired cell rate                */
    NPF_F_ATM_UBR_WITH_MDCR_PCR = 8,   /* Unspecified bit rate w/ minimum  */
                                       /* Desired cell rate and peak rate  */
    NPF_F_ATM_GFR = 9                  /* Guaranteed frame rate            */
} NPF_F_ATM_ServiceCategory_t;
```

The UPC/NPC functions or buffer congestion within the system may lead to dropping of cells as per the configured policy. The drop policy is configured using a bit mask as defined below. Setting both `NPF_F_ATM_CELL_CLP0_DROP` and `NPF_F_ATM_CELL_CLP1_DROP` bits will cause dropping to occur irrespective of the CLP value. The `NPF_F_ATM_PACKET_EP_DROP` and `NPF_F_ATM_PACKET_PP_DROP` bits are used to indicate how the frames discard should be performed. Setting drop policy bit mask to 0 indicates no dropping of cells is performed.

```
typedef NPF_uint16_t NPF_F_ATM_DropPolicy_t;

#define NPF_F_ATM_CELL_CLP0_DROP       0x0001 /* Drop CLP=0 cells    */
#define NPF_F_ATM_CELL_CLP1_DROP       0x0002 /* Drop CLP=1 cells    */
#define NPF_F_ATM_PACKET_EP_DROP       0x0004 /* Drop frames;        *
                                               * Early pkt discard   */
#define NPF_F_ATM_PACKET_PP_DROP       0x0008 /* Drop frames;        *
                                               * Partial pkt discard */
```

The `NPF_F_ATM_QoS_t` structure is used to configure the traffic management parameters for a virtual link.

```
/*
* ATM QOS profile,
* used in the ATM virtual link attribute structure
*/
typedef struct {
    NPF_F_ATM_ServiceCategory_t rxSrvCat;  /* ATM Service Category        */
    NPF_uint32_t     pcr;          /* Ingress Peak Cell Rate              */
    NPF_uint32_t     scr;          /* Ingress Sustainable Cell Rate       */
    NPF_uint32_t     mbs;          /* Ingress Maximum Burst Size          */
    NPF_uint32_t     mcr;          /* Ingress Minimum Cell Rate           */
    NPF_uint32_t     maxFrmSize;   /* Ingress Maximum Frame Size          */
    NPF_uint32_t     cdvt;         /* Ingress Cell Delay Var. Tolerance   */
    NPF_uint16_t     ubrPrio;      /* Ingress Priority for UBR with prio. */
```

```
    NPF_F_ATM_DropPolicy_t upcPolicy; /* Usage Parameter control policy    */
    NPF_boolean_t    tagging;      /* Tag CLP=0 cells as per UPC          */
} NPF_F_ATM_Policing_Char_t;

typedef struct {
    NPF_F_ATM_ServiceCategory_t srvCat;   /* ATM Service Category         */
    NPF_uint32_t     pcr;          /* Egress Peak Cell Rate               */
    NPF_uint32_t     scr;          /* Egress Sustainable Cell Rate        */
    NPF_uint32_t     mbs;          /* Egress Maximum Burst Size           */
    NPF_uint32_t     mcr;          /* Egress Minimum Guaranteed Cell Rate  */
    NPF_uint32_t     maxFrmSize;   /* Egress Maximum Frame Size           */
    NPF_uint32_t     cdvt;         /* Egress Cell Delay Var. Tolerance    */
    NPF_uint16_t     ubrPrio;      /* Egress Priority for UBR with prio.  */
    NPF_uint32_t     bfrThresh;    /* Buffer congestion threshold for VC  */
    NPF_uint32_t     warnThresh;   /* Overflow early warn threshold for VC */
    NPF_F_ATM_DropPolicy_t dropPolicy;  /* Drop policy based on buffer use */
} NPF_F_ATM_Queuing_Char_t;
```

The `bfrThresh` and `warnThresh` parameters should be specified in units of cells.

```
typedef struct {
    NPF_F_ATM_Policing_Char_t policeParams; /* Policing parameters        */
    NPF_F_ATM_Queuing_Char_t  queueParams;  /* Shaping/queuing parameters  */
} NPF_F_ATM_QoS_t;
```

4.3 ATM Adaptation Layer Types

The ATM adaptation layers to support multiple protocols to fit the needs of different service users enhance the services provided by the ATM layer. This data type is used to identify the various adaptation layers in used in ATM networks. The value `NPF_F_ATM_AAL_UNKNOWN` indicates that the AAL cannot be determined on this connection.

```
/*
 * ATM Vc AAL type code, used in the ATM Vc
 * attribute structure
 */
typedef enum {
    NPF_F_ATM_AAL0       = 0, /* RAW cell transfer. No AAL     */
    NPF_F_ATM_AAL1       = 1, /* AAL 1                         */
    NPF_F_ATM_AAL2       = 2, /* AAL 2                         */
    NPF_F_ATM_AAL5       = 3, /* AAL 5                         */
    NPF_F_ATM_AAL_UNKNOWN= 4, /* AAL cannot be determined      */
} NPF_F_ATM_AAL_t;
```

4.3.1 ATM Adaptation Layer Profile Types

The ATM AAL profile is used to configure AAL specific parameters for the VC. This is a union type and the AAL type component of the enclosing structure is used to select the appropriate union member.

```
/*
* AAL Profile
*/
typedef union {
    NPF_F_ATM_AAL1Profile_t   aal1Profile;  /* ATM Adaptation Layer Type 1 */
    NPF_F_ATM_AAL2Profile_t   aal2Profile;  /* ATM Adaptation Layer Type 2 */
    NPF_F_ATM_AAL5Profile_t   aal5Profile;  /* ATM Adaptation Layer Type 5 */
} NPF_F_ATM_AAL_Profile_t;
```

4.3.2 ATM Adaptation Layer Type 1 (AAL1) Profile

```
/*
```

```
*   AAL type 1 subtype used by the CBR service application (e.g. 64
*   KBPS voice band signal transport, circuit transport)
*/
typedef enum {
    NPF_F_ATM_NULL = 0,
    NPF_F_ATM_VOICEBAND = 1,
    NPF_F_ATM_CIRCUIT_EMULATION_SYNCHRONOUS = 2,
    NPF_F_ATM_CIRCUIT_EMULATION_ASYNCHRONOUS =3,
    NPF_F_ATM_HIGH_QUALITY_AUDIO = 4,
    NPF_F_ATM_VIDEO = 5
} NPF_F_ATM_AAL1Subtype_t;

/*
*   Rate of CBR service supported by the AAL
*/
typedef enum {
    NPF_F_ATM_64_KBPS     = 0,
    NPF_F_ATM_1544_KBPS   = 1,
    NPF_F_ATM_6312_KBPS   = 2,
    NPF_F_ATM_32064_KBPS  = 3,
    NPF_F_ATM_44736_KBPS  = 4,
    NPF_F_ATM_97728_KBPS  = 5,
    NPF_F_ATM_2048_KBPS   = 6,
    NPF_F_ATM_8448_KBPS   = 7,
    NPF_F_ATM_34368_KBPS  = 8,
    NPF_F_ATM_139264_KBPS = 9,
    NPF_F_ATM_N64_KBPS    = 10,
    NPF_F_ATM_N8_KBPS     = 11
} NPF_F_ATM_CBR_t;

/*  Clock recovery type :
*   Synchronous,
*   Asynchronous-SRTS(Synchronous Residual Time Stamp) or
*   Asynchronous-Adaptive Clock Recovery.
*/
typedef enum {
    NPF_F_ATM_SYNCHRONOUS = 0,          /* Synchronous clock recovery     */
    NPF_F_ATM_ASYNCHRONOUS_SRTS = 1,    /* Synchronous Residual Time Stamp*/
    NPF_F_ATM_SYNCHRONOUS_ADAPTIVE = 2 /* Adaptive clock recovery        */
} NPF_F_ATM_AAL1ClkRecType_t;

/*   FEC method:
*     no FEC,
*     FEC for Loss Sensitive Signal Transport or
*     FEC for Delay Sensitive
*/
typedef enum {
    NPF_F_ATM_NO_FEC = 0,
    NPF_F_ATM_LOSS_SENSITIVE_SIGNAL_FEC = 1,
    NPF_F_ATM_DELAY_SENSITIVE_SIGNAL_FEC = 2
} NPF_F_ATM_AAL1FEC_t;

/*   CAS Mode: Valid only for structured interfaces
*     No CAS bits carried or
*     Carry CAS Bits in E1 multiframe structure or
*     Carry CAS bits in DS1 SF multiframe structure or
*     Carry CAS bits in DS1 ESF multiframe structure
*/
typedef enum {
```

```
    NPF_F_ATM_BASIC_MODE = 0,
    NPF_F_ATM_CAS_MODE_E1 = 1,
    NPF_F_ATM_CAS_MODE_DS1SF = 2,
    NPF_F_ATM_CAS_MODE_DS1ESF = 3,
    NPF_F_ATM_CAS_MODE_J2 = 4
} NPF_F_ATM_AAL1CASMode_t;
```

This structure is used to configure AAL specific parameters for VCC's carrying AAL1 traffic.

```
/*
* AAL1 Profile
*/
typedef struct {
    NPF_F_ATM_AAL1Subtype_t    subtype;        /* AAL1 sub-type           */
    NPF_F_ATM_CBR_t            cbrRate;        /* rate of CBR service     */
    NPF_uint32_t               rateMultiplier; /* Rate multiplier         */
    NPF_F_ATM_AAL1ClkRecType_t clkRecoveryType;/* Clock Recovery Type     */
    NPF_F_ATM_AAL1FEC_t        fecType;        /* Error Correction Method */
    NPF_F_ATM_AAL1CASMode_t    casMode;        /* CAS Transport mode      */

    /* Structured data transfer configuration. When sdtSupport is set
       to TRUE, it indicates structured data transfer. fecType should
       be configured as NPF_F_ATM_NO_FEC to select SDT                    */
    NPF_boolean_t          sdtSupport;         /* Whether SDT configured  */
    NPF_uint16_t           sdtBlockSize;       /* SDT Block Size          */

    /* partFilledCells set to TRUE causes the cell to be partially filled *
       before transmission in order to avoid excessive latency            */
    NPF_boolean_t          partFilledCells;  /*Enable partial cell method?*/

    /* Amount of user info in bytes that can be carried in partially filled*/
    /* cells. Valid only when partial filled cell method is selected       */
    NPF_uint32_t                partFilledCellsUserInfoSize;

    /* The maximum cell arrival jitter in 10 usec increments that the
       reassembly process will tolerate in the cell stream. Jitter beyond
       this value may lead to errors.                                     */
    NPF_F_ATM_Timers_t     cesCDVRxtolrnc;

    /* Define maximum size in 10 us increments for the reassembly buffer. */
    NPF_F_ATM_Timers_t     maxReasmBufSize;

    /* Time in milliseconds for the cell loss integration period. If cells
       are lost for this period of time, the Interworking VCC Termination
       Point entity will  generate a cell starvation alarm.           */
    NPF_F_ATM_Timers_t     cellLossIntegrPeriod;
} NPF_F_ATM_AAL1Profile_t;
```

4.3.3 ATM Adaptation Layer Type 2 (AAL2) Profile

```
/* AAL2 SSCS service category */
typedef enum {
    NPF_F_ATM_AUDIO = 1,               /* Audio SAP enabled               */
    NPF_F_ATM_MULTIRATE = 2,           /* Multirate SAP enabled           */
    NPF_F_ATM_AUDIO_AND_MULTIRATE =3   /* Audio and Multirate SAP enabled */
} NPF_F_ATM_AAL2SscsServiceCategory_t;

/* PCM encoding */
typedef enum {
```

```
   NPF_F_ATM_ALAW  = 1,                /* Companding as per A-Law         */
   NPF_F_ATM_ULAW = 2                  /* Companding as per u-Law         */
} NPF_F_ATM_AAL2SscsPcmEncoding_t;

typedef enum {
    NPF_F_ATM_ITUT  = 1,               /* Profile defined by ITU-T         */
    NPF_F_ATM_OTHER = 2                /* Profile defined by other entities*/
} NPF_F_ATM_AAL2SscsProfileSource_t;

/*
 * The SSCS configued for this AAL2 connection.
 */
typedef enum {
    NPF_F_ATM_AAL2_SSCS_NONE = 0,      /* No SSCS                          */
    NPF_F_ATM_AAL2_SSCS_SAR  = 1,      /* SSSAR SSCS (I.366.1)             */
    NPF_F_ATM_AAL2_SSCS_TRUNK= 2       /* Trunking SSCS (I.366.2)          */
} NPF_F_ATM_AAL2SscsType_t;

/*
 * Maximum possible AAL2- CPS SDU length for an AAL2 channel
 */
typedef enum {
    NPF_F_ATM_AAL2_CPS_SDU_LEN_45 = 0,   /* Maximum SDU length is 45 */
    NPF_F_ATM_AAL2_CPS_SDU_LEN_64 = 1    /* Maximum SDU length is 64 */
} NPF_F_ATM_AAL2_CpsSduLen_t;
```

The AAL2 channels created on an AAL2 path may be offered differential service with respect to the bandwidth usage and service delay. One or more AAL2 channels may be associated with a priority level. The bandwidth of the configured AAL2 path is shared between these priority levels proportionate to the weight configured for the priority level. The structure detailed below is used to specify the weight associated with each priority level. Setting `maxPrio` to 0 is used to indicate that there is no prioritization for traffic on different AAL2 channels within the AAL2 path i.e. all AAL2 channels are treated equal.

```
/*
 * AAL2 QoS Profile
 */
typedef struct {
    NPF_uint32_t           maxPrio; /* Number of priority levels           */
    NPF_uint32_t          *weight;  /* Array specifying Weight used to     *
                                    * share path bandwidth among AAL2     *
                                    * groups on this AAL2 path            */
    NPF_uint32_t         *discThrsh;/* Threshold specified in number of    *
                                    * packets pending in a priority queue *
                                    * waiting for transmit opportunity    *
                                    * If queue length exceeds this figure *
                                    * then new packets are received on    *
                                    * this queue are discarded            */
} NPF_F_ATM_AAL2QosProfile_t;

/*
* AAL2 Trunking SSCS Profile
*/
typedef struct{
    /* Common SSCS Parameters(SSCS type = Trunking)as specified in I.366.2 */
    NPF_F_ATM_AAL2SscsServiceCategory_t srvCategory;/*Service category     */
    NPF_boolean_t audioServiceTransport;       /* Audio Transport enabled ?*/
    NPF_F_ATM_AAL2SscsProfileSource_t profileSource; /* profile source     */
    NPF_uint32_t predefinedProfileIdentifier;    /* predefined profile id  */
```

```
    NPF_F_ATM_AAL2SscsPcmEncoding_t pcmEncoding; /* PCM encoding type      */
    NPF_boolean_t faxDemodTransport;          /* demod. fax data support ? */
    NPF_boolean_t casTransport;               /* CAS support ?             */
    NPF_boolean_t dtmfDigitPacketTransport;   /* DTMF dialed digit support?*/
    NPF_boolean_t mfR1DigitPacketTransport;   /* MF-R1 dial digit support ?*/
    NPF_boolean_t mfR2DigitPacketTransport;   /* MF-R2 dial digit support ?*/
    NPF_boolean_t circuitModeDataTransport;   /* Circuit mode data support?*/
    NPF_uint32_t  circuitModeDataNumChannels; /* Multiplier N in N*64kbit/s
                                                 circuit mode data?      */
    NPF_boolean_t loopbackEnabled;           /* I.366.2 loopback enabled  */
} NPF_F_ATM_AAL2TrunkSscsProfile_t;


/*
* AAL2 CPS Profile
*/
typedef struct{

    /* This parameter indicates the maximum size CPS-SDU, in octets, that
       is transported on any AAL type 2 channel of an ATM connection. This
       parameter can take on the values "45" or "64" and is set by the
       signaling or management procedures.(See Max_CPS-SDU_Length;I.363.2)*/
    NPF_F_ATM_AAL2_CpsSduLen_t  cpsMaxSduLength;/*Maximum CPS-SDU size     */

    /* If the singleCpsPacketPerCpsPduNoOverlap option is selected         */
    /* then the TIMER CU is nor applicable and the AAL2 payload cannot be  */
    /* greater than 44 bytes.                                              */
    NPF_boolean_t   singleCpsPerPduNoOverlap; /* CPS interleave control    */

    /* The Combined use timer value configured for this connection. This is*
     * valid only when singleCpsPerPduNoOverlap is set as FALSE           */
    NPF_F_ATM_Timers_t    cpsTimer_CU;         /* Combined Use Timer_CU    */

} NPF_F_ATM_AAL2CpsProfile_t;

/*
* AAL2 Profile
*/
typedef struct{
    /* AAl2 QoS Profile     */
    NPF_F_ATM_AAL2QosProfile_t        aal2QosProfile;

    /* AAL2 CPS Profile */
    NPF_F_ATM_AAL2CpsProfile_t        aal2CpsProfile;

    /* AAL2 service specific convergence sub layer configured for this VC  */
    NPF_F_ATM_AAL2SscsType_t          sscsType;

    /* I.366.2 Trunking SSCS Profile; Used if SSCS Type is set to
     * NPF_F_ATM_AAL2_SSCS_TRUNK                                           */
    NPF_F_ATM_AAL2TrunkSscsProfile_t  aal2TrkSscsProf;
} NPF_F_ATM_AAL2Profile_t;
```

4.3.4 ATM Adaptation Layer Type 5 (AAL 5) Profile

```
/*
*  This attribute indicates whether the AAL for the supporting VCC
*  operating in message mode or streaming mode, assured or non assured
*/
```

```
typedef enum {
    NPF_F_ATM_MESSAGE = 0,
    NPF_F_ATM_STREAMING = 1,
} NPF_F_ATM_AAL5_Mode_t;

/*
* SSCS type
*/
typedef enum{
    NPF_F_ATM_NULL_SSCS        = 0,/* NULL SSCS                        */
    NPF_F_ATM_DATA_ASSURED     = 1,/* Data SSCS on SSCOP(non assured) */
    NPF_F_ATM_DATA_NON_ASSURED = 2,/* Data SSCS on SSCOP(assured)     */
    NPF_F_ATM_FRAME_RELAY      = 3,/* Frame relay SSCS                 */
} NPF_F_ATM_AAL5_SscsType_t;
```

When the `deliverCorruptSdu` option in the AAL5 profile is set to `TRUE`, any partially reassembled SDU will be delivered to the AAL service user if the reassembly process is aborted due to errors. The maximum length of the SDU delivered to service user in this case is specified by `maxCorruptSduDeliverLen`.

The reassembly process may be be guared by a reassembly timer specified in the `rasTimer` field. Setting the timer value to '0' indicates that the reassembly timer is disabled.

```
/*
* AAL5 Profile
*/
typedef struct{
    NPF_uint32_t            maxCpcsSduSizeForward; /* Max o/g CPCS_SDU sz  */
    NPF_uint32_t            maxCpcsSduSizeBackward;/* Max i/c CPCS_SDU sz  */
    NPF_F_ATM_AAL5_Mode_t   aalMode;               /* AAL Mode             */
    NPF_F_ATM_AAL5_SscsType_t   sscsType;          /* SSCS Type            */
    NPF_boolean_t           deliverCorruptSdu;     /*If delivery of corrupt
                                                   * SDU is enabled        */
    NPF_uint32_t            maxCorruptSduDeliverLen;  /* Maximum size of
                                                   * delivered corrupt SDU */
    /* Timer configured to guard re-assembly process                       */
    NPF_F_ATM_Timers_t      rasTimer;             /* Reassembly Timer      */

}NPF_F_ATM_AAL5Profile_t;
```

4.4 ATM Virtual Link Data Types

4.4.1 ATM Virtual Channel Characteristics

This structure is used to configure a virtual channel on an interface in the system. The `aalType` is used to select the appropriate AAL provide from the `aalProfile` union.

The VC configuration structure allows mapping of the VC being provisioned to another set of VCs on the same FE or to the backplane by mapping to a backplane link. Each cross connection established in this manner is identified by a unique VC channel cross connect identifier. This identifier is assigned by the FAPI client and is transparently used by the FAPI implementation. The identifier may be used by the FAPI client to delete one or all the cross connections established at the time of VC provisioning by using the `NPF_F_ATM_ConfigMgr_VcLinkXcDelete` function. The FAPI client may also add more cross connection if required using the `NPF_F_ATM_ConfigMgr_VcLinkXcSet` function. The 'B' links to be cross connected should be established ahead of the link being configured. The maximum number of such cross connections allowed is defined by the FAPI implementation and is out of the scope of NPF.

```
/*
```

```
* ATM VC attributes
*/
typedef struct {
    /* The ATM virtual link identifier that is used to uniquely identify   *
     * a virtual channel link. The FAPI client can assign any value to the *
     * ATM virtual link identifier and the value is completely opaque to   *
     * the implementation.                                                 */
    NPF_F_ATM_VirtLinkID_t  vcLinkId;   /* Unique ID of this VC link       */

    /* The below field is used to specify the VPI and VCI of this VC       */
    NPF_F_ATM_VcAddr_t      vc;         /* VPI/VCI of virtual channel      */

    /* The scope of VC i.e. whether it is an external VC or an internal VC *
     * Internal VCs are used to carry data within the system and have no   *
     * external network terminations.                                      */

    NPF_F_ATM_VirtLinkScope_t  vcScope; /* External or internal VC         */

    /* This field defines the ATM interface identifier that is used        *
     * to uniquely identify an ATM interface in the system. The FAPI       *
     * client SHOULD assign the interface index defined by the             *
     * Interface Management APIs to this field.                            */
    NPF_F_ATM_IfID_t           ifId;       /* ATM Interface ID             */

    /* The ATM adaptation layers to support multiple protocols to fit      *
     * the needs of different service users enhance the services provided  *
     * by the ATM layer. This data type is used to identify the various    *
     * adaptation layers used in ATM networks. The value                   *
     * NPF_F_ATM_AAL_UNKNOWN indicates that the AAL cannot be determined   *
     * on this connection.                                                 */
    NPF_F_ATM_AAL_t         aalType;    /* AAL type                        */

    /* AAL specific configurations. The members of below union selected    *
     * based on the configured aalType.                                    */
    NPF_F_ATM_AAL_Profile_t aalProfile; /* AAL Profile                     */

    /* The below field is used to specify the direction of data flow on    *
     * the configured VC link with respect to FE being configured          */
    NPF_F_ATM_Direction_t   direction;  /* receive/transmit/duplex         */

    /* The NPF_F_ATM_QoS_t structure is used to configure the traffic      *
     * management parameters for this virtual channel. If the direction of *
     * of the VC is configured as received then only policing parameters   *
     * are valid for the VC. If the direction is configured as transmit    *
     * the queuing parameters are valid. When the direction is configured *
     * as duplex, both policing and queuing params need to be configured   */
    NPF_F_ATM_QoS_t         qos;        /* QoS profile                     */

    /* The below field provides the administratively configured status of  *
     * the VC. The actual status i.e operational status assumed by the     *
     * link may be different based on the actual physical status. The      *
     * actual operational status queried may be using function provided    *
     * to query the operational status                                     */
    /* Administrative status can be NPF_STATUS_UP, NPF_STATUS_DOWN         *
     * or NPF_STATUS_TESTING                                               */
    NPF_ObjStatus_t         admStatus;  /* Status of VC - UP/DOWN/TESTING  */

    /* The below field provides the status of statistics collections on the*
     * VC. When set to NPF_TRUE, statistics collection is enabled on this  *
```

```
     * VC. The statistics collection may also be enabled or disabled at a  *
     * future point in time by issueing the statistics enable/disable      *
     * function call. The current statistics collection state may be       *
     * queried using the function provided to query the VC information     */
    NPF_boolean_t         statsEnabled;       /* Statistic collecting state*/

    /* The below field is used to cross connect the VC link to another     *
     * VC link on the same FE or to a backplane switch link. When the      *
     * numLink_B field is set 0, it indicates there is no cross connection *
     * established for this link. For a unicast mapping, the VC link being *
     * provisioned may be connected to another VC link or backplane link   *
     * and the numLink_B is set to 1. For multicast mapping, the numLink_B *
     * is set to the number of cross connections to be made.               */
    NPF_uint32_t  numLink_B;      /* Number of links connected to the link *
                                     being provisioned                    */
    NPF_F_ATM_ConfigMgr_VcLinkXcInfo_t   *link_B; /* Mapped link 'B'       */
} NPF_F_ATM_ConfigMgr_Vc_t;
```

4.4.2 ATM Virtual Channel Information

This structure is used to return below information in response to virtual channel information query by the FAPI client.

- Virtual Channel Configuration Information
- OAM procedures and status
- Virtual Channel Bindings
- Statistics collection state

```
/*
* ATM VC query response
*/
typedef struct {
     NPF_F_ATM_ConfigMgr_Vc_t     vcConfig;  /* VC configuration info.    */

     /* vcBound indicates if the VC is bound to a child interface. If set *
      * to TRUE, the interface handle of the child interface is returned  *
      * in the ifChildHandle field                                        */
     NPF_boolean_t                 vcBound;  /* Whether bound to child I/F*/
     NPF_IfHandle_t           ifChildHandle; /* Bound interface           */

     NPF_ObjStatus_t              operStatus;/* Operational status of VC  */

     /* OAM configuration and status information. Depending on the conn.  *
      * point type either eteFlowInfo, segFlowInfo or both may be valid   */
     NPF_F_ATM_OAM_CP_Type_t       connPtType;  /* Connection Pt. type    */
     NPF_F_ATM_ConfigMgr_OamInfo_t eteFlowInfo; /* E-T-E Flow Information */
     NPF_F_ATM_ConfigMgr_OamInfo_t segFlowInfo; /* SEG Flow Information   */
} NPF_F_ATM_ConfigMgr_VcInfo_t;
```

4.4.3 ATM Virtual Path Characteristics

This structure is used to configure a virtual path on an interface in the system.

The VP configuration structure allows mapping of the VP being provisioned to another set of VPs on the same FE or to the backplane by mapping to a backplane link. Each cross connection established in this manner is identified by a unique VP channel cross connect identifier. This identifier is assigned by the FAPI client and is transparently used by the FAPI implementation. The identifier may be used by the FAPI client to delete one or all cross connections established at the time of VP provisioning by using the `NPF_F_ATM_ConfigMgr_VpLinkXcDelete` function. The FAPI client may also add more cross connections if required using the `NPF_F_ATM_ConfigMgr_VpLinkXcSet` function. The 'B'

links to be cross connected should be established ahead of the link being configured. The maximum number of such cross connections allowed is defined by the FAPI implementation and is out of the scope of NPF.

```
/*
* ATM VP link attributes
*/
typedef struct {
    /* The ATM virtual link identifier that is used to uniquely identify   *
     * a virtual path link. The FAPI client can assign any value to the    *
     * ATM virtual link identifier and the value is completely opaque to   *
     * the implementation.                                                 */
    NPF_F_ATM_VirtLinkID_t  vpLinkId;   /* ID assigned to this VP link     */

    /* The VPI of the VP link being configured                             */
    NPF_F_ATM_VPI_t         vpi;        /* VPI of virtual connection       */

    /* The scope of VP i.e. whether it is an external VP or an internal VP *
     * Internal VPs are used to carry data within the system and have no   *
     * external network terminations.                                      */

    NPF_F_ATM_VirtLinkScope_t  vpScope; /* External or internal VP         */

    /* This section defines the ATM interface identifier that is used      *
     * to uniquely identify an ATM interface in the system. The FAPI       *
     * client SHOULD assign the interface index defined by the             *
     * Interface Management APIs to this field.                            */
    NPF_F_ATM_IfID_t           ifId;       /* ATM Interface ID             */

    /* The below field is used to specify the direction of data flow on    *
     * the configured VP link with respect to FE being configured          */
    NPF_F_ATM_Direction_t   direction;  /* receive/transmit/duplex         */

    /* This field specifies if the specified VP is terminated in this FE.  *
     * If the VP is terminated then the ATM header is analysed further to  *
     * identify the VC link using the VCI in the ATM header                */
    NPF_boolean_t           terminated; /* Switched/Terminated VP          */

    /* The NPF_F_ATM_QoS_t structure is used to configure the traffic      *
     * management parameters for this virtual path   .                     */
    NPF_F_ATM_QoS_t         qos;        /* QoS profile                     */

    /* The below field provides the administratively configured status of  *
     * the VP. The actual status assumed by the link may be different based*
     * on the actual physical status. The operation status may be queried*
     * using the function provided to query the operational status         */
    /* Administrative status can be NPF_STATUS_UP, NPF_STATUS_DOWN         *
     * or NPF_STATUS_TESTING                                               */
    NPF_ObjStatus_t         admStatus;  /* Status of VP - UP/DOWN/TESTING  */

    /* The below field provides the status of statistics collections on the*
     * VP. When set to NPF_TRUE, statistics collection is enabled on this  *
     * VP. The statistics collection may also be enabled or disabled at a  *
     * future point in time by issueing the statistics enable/disable      *
     * function call. The current statistics collection state may be       *
     * queried using the function provided to query the VP information     */
    NPF_boolean_t         statsEnabled;       /* Statistic collecting state*/

    /* The below field is used to cross connect the VP link to another     *
     * VP link on the same FE or to a backplane switch link. When the      *
```

```
     * numLink_B field is set 0, it indicates there is no cross connection *
     * established for this link. For a unicast mapping, the VP link being *
     * provisioned may be connected to another VP link or backplane link   *
     * and the numLink_B is set to 1. For multicast mapping, the numLink_B *
     * is set to the number of cross connections to be made.               */
    NPF_uint32_t  numLink_B;      /* Number of links connected to the link *
                                     being provisioned                    */
    NPF_F_ATM_ConfigMgr_VpLinkXcInfo_t   *link_B; /* Mapped link 'B'      */
} NPF_F_ATM_ConfigMgr_Vp_t;
```

4.4.4 ATM Virtual Path Information

This structure is used to return below information in response to virtual path information query by the FAPI client.

- Virtual Path Configuration Information
- OAM procedures enabled and current status
- Virtual Channel Bindings
- Statistics collection state

```
/*
* ATM VP query response
*/
typedef struct {
     NPF_F_ATM_ConfigMgr_Vp_t     vpConfig;  /* VP configuration info.    */
     NPF_ObjStatus_t            operStatus;  /* Operational status of VP  */

     /* OAM configuration and status information. Depending on the conn.  *
      * point type either eteFlowInfo, segFlowInfo or both may be valid   */
     NPF_F_ATM_OAM_CP_Type_t       connPtType;  /* Connection Pt. type    */
     NPF_F_ATM_ConfigMgr_OamInfo_t eteFlowInfo; /* E-T-E Flow Information */
     NPF_F_ATM_ConfigMgr_OamInfo_t segFlowInfo; /* SEG Flow Information   */
} NPF_F_ATM_ConfigMgr_VpInfo_t;
```

4.4.5 Cross Connection Type

This enumeration specifies the type of cross connection for a VP/VC link or an AAL2 channel. The VP/VC link may be connected to either another VP/VC link or an AAL2 channel on the same FE or they may be connected to a backplane destination over a backplane switch link. An external VP/VC links or AAL2 channels may be cross connected to an internal VP/VC links or AAL2 channels respectively on the same FE.

```
typedef enum {
    NPF_F_ATM_EXT_TO_EXT = 0,   /* Connect to an external VP/VC         *
                                 * link or AAL2 channel                */
    NPF_F_ATM_EXT_TO_BACK = 1,  /* Connect to a backplane switch link   */
    NPF_F_ATM_BACK_TO_INT = 2,  /* Connect to an internal VP/VC         *
                                 * link or AAL2 channel                */
    NPF_F_ATM_EXT_TO_INT = 3    /* Connect to a external VP/VC  link    *
                                 * or AAL2 channel or a                *
                                 * corresponding internal VP/VC link   *
                                 * or AAL2 channel                     */
} NPF_F_ATM_XcType_t;
```

4.4.6 Cross Connection of ATM virtual channel links

This structure is used to configure a VC link cross connect to connect link 'A' to one or more link 'B'. The number of cross connections to be made is specified by the `numLink_B` field. The link 'B' for cross connection of the VC link 'A' is one of the below:

- A external or internal VC link configured on the same FE as the VC link 'A'
- A backplane link

When the VC link is connected to a VC link located on the same FE, the ID of the Virtual link to which the cross connection is required is specified using the virtual link ID of the link 'B'. When the VC link is connected to a backplane link, the cross connection is specified using the backplane switch headers of the backplane switch link and the backplane interface on which the backplane link is carried. Multicast connections may be created by cross connecting link 'A' to multiple destinations.

The link 'B' to connect to the link 'A' is determined using the `xcType` field as below:

Table 4.2: VC Link cross-connection types

xcType	Description
NPF_F_ATM_EXT_TO_EXT	The external VC link 'A' is connected to an external VC link 'B' on the same FE. The `mapVcLink` in the union of the structure below specifies the VC link ID of VC link 'B' connected to the VC link A.
NPF_F_ATM_EXT_TO_BACK	The external VC link 'A' is connected to a backplane link. The backplane link is specified using the `mapSwLink` field of the union.
NPF_F_ATM_BACK_TO_INT	The internal VC link 'A' is connected to a backplane link. The backplane link is specified using the `mapSwLink` field of the union.
NPF_F_ATM_EXT_TO_INT	The external/internal VC link 'A' is connected to an external/internal VC link 'B' on the same FE. The `mapVcLink` in the union of the structure below specifies the VC link ID of VC link 'B' connected to the VC link A.

```
/*
* ATM VC Link Cross connect Information
 */
typedef struct {
    NPF_F_VcXcId_t           vcXcId;        /* ID of this cross connection */
    NPF_F_ATM_XcType_t       xcType;        /* Type of cross connection    */
    union {
        NPF_F_ATM_VirtLinkID_t  mapVcLink;  /* Mapped to a VC Link         */
        NPF_F_SwitchAddress_t   mapSwLink;  /* Mapped to backplane         */
    }u;
} NPF_F_ATM_ConfigMgr_VcLinkXcInfo_t;

/*
* ATM VC link cross connect
*/
typedef struct {
    NPF_F_ATM_VirtLinkID_t                link_A;  /* VC Link 'A'           */
    NPF_uint32_t                       numLink_B;  /* Number of 'B' links   *
                                                    * connected to link 'A' */
    NPF_F_ATM_ConfigMgr_VcLinkXcInfo_t  *link_B;  /* Mapped link 'B'       */
} NPF_F_ATM_ConfigMgr_VcLinkXc_t;
```

The direction of data flow between the link 'A' and link 'B' is determined by the direction configured for the link 'A'. When the VC link 'A' direction is configured as `NPF_F_ATM_RECEIVE`, the frames/cells received over the link 'A' are transmitted over the link 'B' to the connected VC link or to the backplane. When the VC link 'A' direction is configured as `NPF_F_ATM_TRANSMIT`, the frames/cells received over the link 'B' are transmitted over the VC link specified as link 'A'. When the VC link 'A' is configured as `NPF_F_ATM_DUPLEX`, the frames/cells received over the link 'A' are transmitted over the link 'B' to the connected VC or to the backplane and vice-versa.

4.4.7 Cross Connection of ATM virtual path links

This structure is used to configure a VP link cross connect to connect link 'A' to one or more link 'B'. The number of cross connections to be made is specified by the `numLink_B` field. The link 'B' for cross connection of the VP link 'A' is one of the below:

- An external or internal VP link configured on the same FE as the VP link 'A'
- A backplane link

When the VP link is connected to a VP link located on the same FE, the ID of the Virtual link to which the cross connection is required is specified using the virtual link ID of the link 'B'. When the VP link is connected to a backplane link, the cross connection is specified using the backplane switch headers of the backplane switch link and the backplane interface on which the backplane link is carried. Multicast connections may be created by cross connecting link 'A' to multiple destinations.

The link 'B' to connect to the link 'A' is determined using the `xcType` field as below:

Table 4.3: VP Link cross-connection types

xcType	Description
NPF_F_ATM_EXT_TO_EXT	The external VP link 'A' is connected to an external VP link 'B' on the same FE. The `mapVpLink` in the union of the structure below specifies the VP link ID of VP link 'B' connected to the VP link A.
NPF_F_ATM_EXT_TO_BACK	The external VP link 'A' is connected to a backplane link. The backplane link is specified using the `mapSwLink` field of the union.
NPF_F_ATM_BACK_TO_INT	The external VP link 'A' is connected to a backplane link. The backplane link is specified using the `mapSwLink` field of the union.
NPF_F_ATM_EXT_TO_INT	The external/internal VP link 'A' is connected to an external/internal VP link 'B' on the same FE. The `mapVpLink` in the union of the structure below specifies the VP link ID of VP link 'B' connected to the VP link A.

```
/*
 * ATM VP Link Cross connect Information
 */
typedef struct {
    NPF_F_VcXcId_t           vpXcId;        /* ID of this cross connection */
    NPF_F_ATM_XcType_t       xcType;        /* Type of cross connection    */
    union {
        NPF_F_ATM_VirtLinkID_t  mapVpLink;  /* Mapped to a VP Link         */
        NPF_F_SwitchAddress_t   mapSwLink;  /* Mapped to backplane         */
    }u;
} NPF_F_ATM_ConfigMgr_VpLinkXcInfo_t;

/*
* ATM VC link cross connect - Connect link 'A' to link 'B'
*/
typedef struct {
    NPF_F_ATM_VirtLinkID_t            link_A;  /* VP Link 'A'              */
    NPF_uint32_t                   numLink_B;  /* Number of 'B' links      *
                                                * connected to link 'A'    */
    NPF_F_ATM_ConfigMgr_VpLinkXcInfo_t *link_B;/* Array of Mapped link 'B' */
} NPF_F_ATM_ConfigMgr_VpLinkXc_t;
```

The direction of data flow between the link 'A' and link 'B' is determined by the direction configured for the link 'A'. When the VP link 'A' direction is configured as `NPF_F_ATM_RECEIVE`, the frames/cells received over the link 'A' are transmitted over the link 'B' to the connected VP link or to the backplane. When the VP link 'A' direction is configured as `NPF_F_ATM_TRANSMIT`, the frames/cells received over the link 'B' are transmitted over the VP link specified as link 'A'. When the VP link 'A' is configured as `NPF_F_ATM_DUPLEX`, the frames/cells received over the link 'A' are transmitted over the link 'B' to the connected VP or to the backplane and vice-versa.

4.4.8 Virtual Channel statistics

The below structure is used to return the statistics information for the queried virtual channel. Statistics counters are non negative integers which wrap around on reaching maximum value using twos complement arithmetic. When the AAL type specified by `aal` field is set to `NPF_F_ATM_AAL_UNKNOWN` the AAL associated with this VC cannot be determined and AAL level statistics are not valid.

```
/*
* ATM Per-Vc Statistics, returned in asynchronous
*/
typedef struct {
    NPF_F_ATM_Stats_t          atmStats;     /* ATM Level statistics     */
    NPF_F_ATM_AAL_t            aal;          /* AAL type                 */
    union {
        NPF_F_ATM_AAL1Stats_t  aal1_stats;   /* AAL1 Statistics          */
        NPF_F_ATM_AAL2Stats_t  aal2_stats;   /* AAL2 Statistics          */
        NPF_F_ATM_AAL5Stats_t  aal5_stats;   /* AAL5 Statistics          */
    }u;
} NPF_F_ATM_ConfigMgr_VcStats_t;
```

4.4.9 ATM Layer Statistics

Statistics counters are non negative integers which wrap around on reaching maximum value using twos complement arithmetic. These statistics cover all cells including OAM cells received on the ATM connection.

```
/*
 * ATM Traffic Management Statistics
 */
typedef struct {
    /* Count of received cells tagged/discarded due to policing actions  */
    NPF_uint64_t cellsTaggedRx;    /* Receive Cells changed CLP0 to CLP1 */
    NPF_uint64_t cellsClp01DiscRx; /* Receive Cells discarded (CLP0+1)   */
    NPF_uint64_t cellsClp0DiscRx;  /* Receive Cells discarded (CLP0)     */

    /* Count of cells tagged/discarded due to congestion in outgoing queue */
    NPF_uint64_t cellsTaggedTx;    /* Transmit Cells changed CLP0 to CLP1*/
    NPF_uint64_t cellsClp01DiscTx; /* Transmit Cells discarded (CLP0+1)  */
    NPF_uint64_t cellsClp0DiscTx;  /* Transmit Cells discarded (CLP0)    */

    /* Queuing statistics in the transmit direction                     */
    NPF_uint32_t curQueueLenTx;    /* Transmit connection queue lengths  */
    NPF_uint32_t maxQueueLenTx;    /* Maximum queue length seen so far   */
} NPF_F_ATM_TMStats_t;

/*
 * ATM Level Statistics
 */
typedef struct {
    /* Count of total number of cells received on this connection       */
```

```
    NPF_uint64_t cellsClp01Rx;     /* Receive Total Cells (CLP0 + CLP1)  */
    NPF_uint64_t cellsClp0Rx;      /* Receive High Priority Cells (CLP0) */

    /* Count of received cells dropped due to resource unavailability    *
     * like buffers to hold/reassemble the cells etc.                    */
    NPF_uint32_t cellsDiscResErrRx;

    /* Count of total number of cells transmitted on this connection     */
    NPF_uint64_t cellsClp01Tx;     /* Transmit Total Cells (CLP0 + CLP1) */
    NPF_uint64_t cellsClp0Tx;      /* Transmit High Priority Cells (CLP0)*/

    /* Statistics counters for policing and queuing (TM) functions       */
    NPF_F_ATM_TMStats_t atmTrafficManagementStats;
} NPF_F_ATM_Stats_t;
```

4.4.10 ATM Adaptation Layer type 1 statistics

Statistics counters are non negative integers which wrap around on reaching maximum value using twos complement arithmetic.

```
typedef struct {
    /* Number of AAL1 header errors detected, including those corrected.
       Header errors include correctable and uncorrectable CRC plus bad
       parity. */
    NPF_uint32_t  errSnp;           /* No. of AAL1 cells with SNP errors */

    /* Sequence Count total violations: i.e., the count of incoming AAL Type1
       SAR-PDUs where the sequence count in the PDU header causes a
       transition from the SYNC state to the OUT OF SEQUENCE state as defined
       by ITU-T Recommendation I.363.1. (optional)
         -  lost cell: i.e., the number of lost cells, as detected by the
            AAL1 sequence number processing, for example. This count records
            the number of cells detected as lost in the network prior to the
            destination interworking function AAL1 layer processing.
            (optional)
         -  misinserted cells: i.e., the number of sequence violation events
            which the AAL CS interprets as misinserted of cells as defined by
            ITU-T Recommendation I.363.1. (optional)*/
    NPF_uint32_t  errSeqNoRx;       /* No. of AAL1 cells with SN errors  */

    /* Number of times the reassembly buffer overflows. If the interworking
       function is implemented with multiple buffers, such as a cell level
       buffer and a bit level buffer, then either buffer overflow will cause
       this count to be incremented).  */
    NPF_uint32_t  errBfrOverflowRx; /* No. of times buffer overflow at CS*/

    /* Number of times the reassembly buffer underflows. In the case of a
       continuous underflow caused by a loss of ATM cell flow, a single
       buffer underflow should be counted. If the interworking function is
       implemented with multiple buffers, such as a cell level buffer and a
       bit level buffer, then either buffer underflow will cause this count
       to be incremented. */
    NPF_uint32_t  errBfrUndrflowRx; /* No. of times buffer undrflow at CS*/

    /* Number of events in which the AAL1 reassembler found that a structured
       data pointer is not where it is expected, and the pointer must be re
       acquired. This count is only meaningful for structured data transfer
       modes as unstructured modes do not use pointers. (mandatory for
       structured data transfer) */
    NPF_uint32_t  errSdtPtrReframesRx;
```

```
    /* Number of times the AAL reassembler detects a parity check failure at
       the point where a structured data pointer is expected. This count is
       only meaningful for structured data transfer modes as unstructured
       modes do not use pointers.)*/
    NPF_uint32_t  errSdtPtrParityRx;
} NPF_F_ATM_AAL1Stats_t;
```

4.4.11 ATM Adaptation Layer type 2 statistics

Statistics counters are non negative integers which wrap around on reaching maximum value using twos complement arithmetic. The statistics counters are grouped into CPS level statistics and SAR SSCS level statistics. The packet and byte counters for CPS packets transmitted and received on an AAL2 path are reported for each priority level configured for the AAL2 path.

```
/* CPS Packet and Byte counters */
typedef struct {
    NPF_uint64_t cpsPktRx;      /* No. of AAL2 CPS packets received   */
    NPF_uint64_t cpsBytesRx;    /* No. CPS packet bytes received      */

    NPF_uint64_t cpsPktTx;      /* No. of AAL2 CPS packets transmitted*/
    NPF_uint64_t cpsBytesTx;    /* No. CPS packet bytes transmitted   */

    /* The below counters specify the number CPS packets discarded in *
     * the transmit direction due to various errors and the           *
     * corresponding byte counts for the discarded packets            */
    NPF_uint32_t cpsPktDisc;    /* No. of AAL2 CPS packets discarded  */
    NPF_uint32_t cpsBytesDisc;  /* No. CPS packet bytes discarded     */

} NPF_F_ATM_AAL2CpsPktByteCtrs_t ;
```

This structure specifies the statistics counters accumulated by the CPS sub-layer.

```
typedef struct {
    /* errno = 0; I.363.2 */
    /* The parity of the STF indicates transmission errors              */
    NPF_uint32_t errCpsParityRx;   /* CPS PDU with parity error         */

    /* errno = 1; I.363.2 */
    /* The sequence number of the STF is wrong                          */
    NPF_uint32_t errCpsSeqNoRx;    /* CPS PDU with sequence no. error   */

    /* errno = 2; I.363.2 */
    /* The number of octets expected for a CPS-Packet overlapping into
       this CPS-PDU does not match the information contained in the STF  */
    NPF_uint32_t errCpsOsfUnex;    /* CPS PDU with unexpected offset    */

    /* errno = 3; I.363.2 */
    /* The OSF of the STF contains a value 48 or greater                */
    NPF_uint32_t errCpsOffsetRx;   /* CPS PDU with offset field error   */

    /* errno = 4; I.363.2 */
    /* The Header Error Control (HEC) Code of a CPS-Packet header
       indicates transmission errors in the CPS-Packet header           */
    NPF_uint32_t errCpsHecRx;      /* CPS packets with CRC error        */

    /* errno = 5; I.363.2 */
    /* The length of the received CPS-Packet Payload (CPS-SDU) exceeds
```

```
       the maximum length indicated in "Max_SDU_Deliver_Length".         */
    NPF_uint32_t errCpsLenRx;

    /* errno = 6; I.363.2 */
    /* Number of times reassembly cancelled due to errno = 0, 1 or 2     */
    NPF_uint32_t numReasmCancel;

    /* errno = 7; I.363.2 */
    /* The Header Error Control (HEC) Code of a CPS-Packet header that
     * was overlapping a CPS-PDU boundary indicates transmission errors in
     * the CPS-Packet header; if the value of the OSF is less than 47,
     * processing starts at the octet pointed to by the OSF.             */
    NPF_uint32_t numPhBfrResetErrHec;

    /* errno = 8; I.363.2 */
    /* The UUI field in the received CPS-Packet header contains a value
       ("28" or "29") that is reserved for future standardization.       */
    NPF_uint32_t errBadUUIRx;      /* Reserved UUI; unexpected UUI Rx    */

    /* errno = 9; I.363.2 */
    /* The CID value in the received CPS-Packet header is not associated
       with a SAP.                                                       */
    NPF_uint32_t errBadCIDRx;      /* Reserved CID;Unknown CID value Rx  */

    /* The packet and byte counters for the number of CPS packets received*
     * and transmitted are maintained per priority level and are returned *
     * as an array. The number of elements in the array is equal to number*
     * of priority levels configured in the QoS profile for the AAL2 path */
    NPF_F_ATM_AAL2CpsPktByteCtrs_t  *cpsPktByteCtrArr;
} NPF_F_ATM_AAL2CpsStats_t;
```

This structure specifies the statistics counters accumulated by the SAR SSCS sub-layer.

```
typedef struct {
    /* errno = 10; I.366.1 */
    /* The maximum permissible size for a reassembled SSSAR-SDU
      ("Max_SDU_Length") has been exceeded.                              */
    NPF_uint32_t errSscsOversizedSssarSduRx;      /* Oversized SSSAR SDU */

    /* errno = 11; I.366.1 */
    /* The reassembly timer RAS_Timer has expired.                       */
    NPF_uint32_t errSscsSssarRasTimerExpiryRx;    /* Reassembly timeout  */

    /* errno = 20; I.366.1 */
    /* An SSTED-PDU of length 8 or less has been received.               */
    NPF_uint32_t errSscsUndersizedSstedPduRx;     /* PDU<8 bytes received*/

    /* errno = 21; I.366.1 */
    /* The value of the Length field in the SSTED-PDU does not match the
       length of the received SSTED-PDU.                                 */
    NPF_uint32_t errSscsSstedPduLengthMismatchRx; /* Length mismatch     */

    /* errno = 22; I.366.1 */
    /* The value of the CRC field is not equal to the CRC calculated over
       the received information.                                         */
    NPF_uint32_t errSscsSstedCrcMismatchRx;       /* SSTED CRC mismatch  */

} NPF_F_ATM_AAL2SarSscsStats_t;
```

This structure specifies the statistics counters accumulated for AAL2 paths. The SAR SSCS statistics are collected only when the SAR SSCS is configured for the path.

```
typedef struct {
    /* CPS sub layer statistics */
    NPF_F_ATM_AAL2CpsStats_t        cpsStats;

    /* SAR SSCS sub layer statistics. Accumulated if the SAR SSCS is
     * associated with the AAL2 channels                                */
    NPF_F_ATM_AAL2SarSscsStats_t    sarSscsStats;
} NPF_F_ATM_AAL2Stats_t;
```

4.4.12 ATM Adaptation Layer type 5 statistics

Statistics counters are non negative integers which wrap around on reaching maximum value using twos complement arithmetic.

```
typedef struct {
    NPF_uint64_t framesRx;         /* Receive AAL5 Frames                 */
    NPF_uint32_t errBadCrcRx;      /* Receive AAL5 Frames with CRC error  */
    NPF_uint32_t errBadLenRx;      /* Receive AAL5 frames with length err */
    NPF_uint64_t bytesRx;          /* Receive Bytes                       */

    NPF_uint64_t framesTx;         /* Transmit AAL5 Frames                */
    NPF_uint64_t bytesTx;          /* Transmit Bytes                      */
} NPF_F_ATM_AAL5Stats_t;
```

4.4.13 Virtual Path statistics

The below structure is used to return the statistics information for the queried virtual path. Statistics counters are non negative integers which wrap around on reaching maximum value using twos complement arithmetic. These statistics cover all cells including OAM cells received on the virtual path.

```
/*
* ATM Per-Vp Statistics, returned in asynchronous response
*/
typedef struct {
    NPF_F_ATM_Stats_t          atmStats;     /* ATM Level statistics     */
} NPF_F_ATM_ConfigMgr_VpStats_t;
```

4.5 ATM AAL2 Channel Data Types

4.5.1 AAL2 Channel Configuration

The below structure describes the configuration parameters for an AAL2 channel. The connection ID and the CID are used to uniquely identify an AAL2 channel. Each channel is assigned a handle called the AAL2 channel ID which identifies an AAL2 channel uniquely in the system.

```
/*
 * AAL2 channel CPS configuration
 */
typedef struct {
    NPF_F_AAL2ChanId_t      aal2ChanId; /* ID assigned to this AAL2 chnl.  */
    NPF_F_ATM_VirtLinkID_t  aal2path;   /* ID of VC constituting AAL2 path */
    NPF_uint8_t             aal2Cid;    /* The CID for this channel        */

    NPF_uint32_t            chnlPrio;   /* Priority of this AAL2 channel   */
                                        /* must be < maxPrio configured on *
                                         * the corresponding AAL2 path     */
```

```
    /* This parameter indicates the maximum size CPS-SDU, in octets, that is
       transported on a particular AAL type 2 channel. It also indicates the
       maximum size CPS-SDU that may be delivered to the corresponding CPS
       user. This  parameter  can take on the values "45" or "64" and is set
       by signaling or management procedures. The following inequality must
       be maintained -    maxSduDeliverLen <=cpsMaxSduLen                  */
    NPF_F_ATM_AAL2_CpsSduLen_t  maxSduDeliverLength;
} NPF_F_ATM_AAL2_Chnl_CpsCfg_t;

/*
 * AAL2 channel SAR SSCS configuration – filled when SSCS configured for the
 * the AAL2 path is SAR SSCS
 */
typedef struct {
    /* Common SSCS Parameters (SSCS type = SAR)as specified in I.366.1     */
    /* Selection of the transmission error detection  mechanisms (SSTED)   */
    NPF_boolean_t       sstedStatus;          /* SSTED selected?           */

    /* Selection of the assured data transfer mechanism (SSADT)            */
    /* When ssadtStatus is set to TRUE, the sstedStatus MUST be set to TRUE*/
    NPF_boolean_t       ssadtStatus;          /* SSADT selected?           */

    /* Maximum SSSAR SDU length in  bytes                                  */
    /* This parameter indicates the maximum size SSSAR-SDU, in octets, that
     * is allowed to be reassembled. Valid values are between 1 and 65568  */
    NPF_uint32_t        maxSssarSduLength;   /* Max SSSAR-SDU size?        */

    /* Maximum size of segments used to create SSSAR PDU                   */
    /* This must bet set between 1 and maxSduDeliverLength specified in the*/
    /* CPS configuration portion of the AAL2 channel configuration struct  */
    /* This parameter can assume a value between 1 to 45 or 1 to 64 based  *
     * on the configuration of the maxSduDeliverLength field               */
    NPF_uint8_t         maxSssarSegLength;

    /* Timer configured to guard re-assembly process for SSSAR segments    */
    NPF_F_ATM_Timers_t  rasTimer;       /* RAS Timer                       */

} NPF_F_ATM_AAL2_Chnl_SarSscsCfg_t;
```

The AAL2 channels may be bound to a higher level interface. The AAL2 channel is then an endpoint of the connection terminated in this VC. This mechanism allows other applications like PPP over AAL2 and so on.

The AAL2 channel configuration structure also allows mapping of the AAL2 channel being provisioned to another set of AAL2 channels on the same FE or to the backplane by mapping to a backplane link. Each cross connection established in this manner is identified by a unique AAL2 channel cross connect identifier. This identifier is assigned by the FAPI client and is transparently used by the FAPI implementation. The identifier may be used by the FAPI client to delete one or all the cross connections established at the time of AAL2 channel provisioning by using the `NPF_F_ATM_ConfigMgr_AAL2_ChnlXcDelete` function. The FAPI client may also add more cross connection if required using the `NPF_F_ATM_ConfigMgr_AAL2_ChnlXcSet` function. The 'B' links to be cross connected should be established ahead of the link being configured. The maximum number of such cross connections allowed is defined by the FAPI implementation and is out of the scope of NPF.

```
/*
*   AAL2 Channel Config Info.
*/
typedef struct {
```

```
    /* AAL2 channel CPS sub layer configuration parameters */
    NPF_F_ATM_AAL2_Chnl_CpsCfg_t        cpsConfig;

    /* AAL2 channel SAR SSCS sub layer configuration parameters */
    NPF_F_ATM_AAL2_Chnl_SarSscsCfg_t    sarSscsConfig;

    /* The below field provides the administratively configured status of  *
     * the channel. The actual status assumed by the channel may be        *
     * different based on the actual physical status. The operation       *
     * status may be queried using the function provided to query         *
     * the operational status.                                             *
     * Administrative status can be NPF_STATUS_UP, NPF_STATUS_DOWN         *
     * or NPF_STATUS_TESTING                                               */
    NPF_ObjStatus_t        admStatus; /*Status of channel-UP/DOWN/TESTING  */

    /* The below field provides the status of statistics collections on the*
     * channel. When set to NPF_TRUE, statistics collection is enabled.    *
     * The statistics collection may also be enabled or disabled at a      *
     * future point in time by issueing the statistics enable/disable      *
     * function call. The current statistics collection state may be       *
     * queried using the function provided to query the channel information*/
    NPF_boolean_t         statsEnabled;       /* Statistic collecting state*/

    /* The below field is used to cross connect AAL2 channel to another    *
     * AAL2 channel on the same FE or to a backplane switch link. When     *
     * the numLink_B field is set to 0, it indicates there is no cross     *
     * connection established for this AAL2 channel. For a unicast mapping,*
     * the AAL2 channel being provisioned may be connected to another      *
     * AAL2 channel or backplane link and the numLink_B is set to 1.       *
     * For multicast mapping, the numLink_B is set to the number of        *
     * cross connections to be made.                                       */
    NPF_uint32_t  numLink_B;      /* Number of links connected to the AAL2 *
                                     Channel being provisioned            */
    NPF_F_ATM_AAL2_ChnlXcInfo_t           *link_B;   /* Mapped link 'B'    */

} NPF_F_ATM_ConfigMgr_AAL2_Channel_t;
```

4.5.2 AAL2 Channel Information

This structure is used to return below information in response to AAL2 channel information query by the FAPI client.

- AAL2 Channel Configuration Information
- AAL2 Channel Bindings
- Statistics collection state

```
/*
* AAL2 Channel query response
*/
typedef struct {
     /* AAL2 channel configuration information */
     NPF_F_ATM_ConfigMgr_AAL2_Channel_t  chnlCfg;

     /* chnlBound indicates if the channel is bound to a child interface. *
      * If TRUE, the interface handle of the child interface is returned  *
      * in the ifChildHandle field                                        */
     NPF_boolean_t              chnlBound;  /* Whether bound to child I/F*/
     NPF_IfHandle_t           ifChildHandle; /* Bound interface          */

     NPF_ObjStatus_t            operStatus; /* Operational status of chnl*
```

```
                                                    * i.e UP/DOWN/TESTING       */
} NPF_F_ATM_ConfigMgr_AAL2_ChannelInfo_t;
```

4.5.3 Cross Connection of AAL2 channels

This structure is used to configure a AAL2 channel cross connect to connect AAL2 channel 'A' to link 'B'. The link 'B' could be one below type:

- An AAL2 channel configured on the same FE as the AAL2 channel 'A'
- A backplane link

When the AAL2 channel is connected to a AAL2 channel located on the same FE, the ID of the AAL2 channel to which the cross connection is required is specified for the cross connection. When the AAL2 channel is connected to a backplane link, the link 'B' for the cross connection is specified using the backplane switch headers of the backplane switch link and the backplane interface on which the backplane link is carried. Multicast connections may be created by cross connecting AAL2 channel 'A' to multiple destinations.

The channel 'B' to connect to the channel 'A' is determined using the `xcType` field as below:

Table 4.4: AAL2 Channel cross-connection types

xcType	Description
NPF_F_ATM_EXT_TO_EXT	The external AAL2 channel 'A' is connected to an external AAL2 channel 'B' on the same FE. The `mapAal2Chnl` in the union of the structure below specifies the AAL2 channel ID of channel 'B' connected to the AAL2 channel A.
NPF_F_ATM_EXT_TO_BACK	The external AAL2 channel 'A' is connected to a backplane link. The backplane link is specified using the `mapSwLink` field of the union.
NPF_F_ATM_BACK_TO_INT	The internal AAL2 channel 'A' is connected to a backplane link. The backplane link is specified using the `mapSwLink` field of the union.
NPF_F_ATM_EXT_TO_INT	The external/internal AAL2 channel 'A' is connected to an external/internal AAL2 channel 'B' on the same FE. The `mapAal2Chnl` in the union of the structure below specifies the AAL2 channel ID of channel 'B' connected to the AAL2 channel A.

```
/*
 * ATM AAL2 channel Link Cross connect Information
 */
typedef struct {
    NPF_F_AAL2ChnlXcId_t aal2ChnlXcId;  /* ID of AAL2 chnl. Cross cnct   */
    NPF_F_ATM_XcType_t   xcType;        /* Type of cross connection      */
    union {
        NPF_F_AAL2ChanId_t     mapAal2ChanId; /* Mapped AAL2 channel ID  */
        NPF_F_SwitchAddress_t  mapSwLink;     /* Mapped to backplane     */
    }u;
} NPF_F_ATM_AAL2_ChnlXcInfo_t;

/*
* ATM AAL2 channel cross connect - Connect channel 'A' to channel 'B'
*/
typedef struct {
    /* AAL2 channel 'A' to connect to second AAL2 channel or backplane     */
```

```
    NPF_F_AAL2ChanId_t                      aal2ChanId_A;

    /* Number of 'B' links connected to channel 'A'                        */
    NPF_uint32_t                            numLink_B;

    /* Mapped link 'B' - Backplane link or another AAL2 channel on same FE */
    NPF_F_ATM_AAL2_ChnlXcInfo_t             *link_B;
} NPF_F_ATM_ConfigMgr_AAL2_ChnlXc_t;
```

4.5.4 ATM Adaptation Layer type 2 Channel statistics

Statistics counters are non negative integers which wrap around on reaching maximum value using twos complement arithmetic. The statistics counters are grouped into CPS level statistics and SAR SSCS level statistics.

This structure specifies the statistics counters accumulated by the CPS sub-layer.

```
typedef struct {
    /* errno = 5; I.363.2 */
    /* The length of the received CPS-Packet Payload (CPS-SDU) exceeds
       the maximum length indicated in "Max_SDU_Deliver_Length".         */
    NPF_uint32_t errCpsLenRx;

    /* errno = 6; I.363.2 */
    /* Number of times reassembly cancelled due to errno = 0, 1 or 2     */
    NPF_uint32_t numReasmCancel;

    /* errno = 7; I.363.2 */
    /* The Header Error Control (HEC) Code of a CPS-Packet header that
     * was overlapping a CPS-PDU boundary indicates transmission errors in
     * The CPS-Packet header; if the value of the OSF is less than 47,
     * Processing starts at the octet pointed to by the OSF.             */
    NPF_uint32_t numPhBfrResetErrHec;

    /* errno = 8; I.363.2 */
    /* The UUI field in the received CPS-Packet header contains a value
       ("28" or "29") that is reserved for future standardization.       */
    NPF_uint32_t errBadUUIRx;      /* Reserved UUI; unexpected UUI Rx    */

    NPF_uint64_t cpsPktRx;         /* No. of AAL2 CPS packets received   */
    NPF_uint64_t cpsBytesRx;       /* No. CPS packet bytes received      */
    NPF_uint64_t cpsPktTx;         /* No. of AAL2 CPS packets transmitted*/
    NPF_uint64_t cpsBytesTx;       /* No. CPS packet bytes transmitted   */
} NPF_F_ATM_AAL2ChnlCpsStats_t;
```

This structure specifies the statistics counters accumulated by the SAR SSCS sub-layer.

```
typedef struct {
    /* errno = 10; I.366.1 */
    /* The maximum permissible size for a reassembled SSSAR-SDU
      ("Max_SDU_Length") has been exceeded.                              */
    NPF_uint32_t errSscsOversizedSssarSduRx;      /* Oversized SSSAR SDU */

    /* errno = 11; I.366.1 */
    /* The reassembly timer RAS_Timer has expired.                        */
    NPF_uint32_t errSscsSssarRasTimerExpiryRx;    /* Reassembly timeout  */

    /* errno = 20; I.366.1 */
    /* An SSTED-PDU of length 8 or less has been received.                */
    NPF_uint32_t errSscsUndersizedSstedPduRx;     /* PDU<8 bytes received*/
```

```
    /* errno = 21; I.366.1 */
    /* The value of the Length field in the SSTED-PDU does not match the
       length of the received SSTED-PDU.                                  */
    NPF_uint32_t errSscsSstedPduLengthMismatchRx; /* Length mismatch     */

    /* errno = 22; I.366.1 */
    /* The value of the CRC field is not equal to the CRC calculated over
       the received information.                                          */
    NPF_uint32_t errSscsSstedCrcMismatchRx;       /* SSTED CRC mismatch  */
} NPF_F_ATM_AAL2ChnlSarSscsStats_t;
```

This structure specifies the statistics counters for the AAl2 channels.

```
typedef struct {

    /* CPS sub layer statistics */
    NPF_F_ATM_AAL2ChnlCpsStats_t     cpsStats;

    /* SAR SSCS sub layer statistics. Accumulated if the SAR SSCS is
     * associated with the AAL2 channels                                */
    NPF_F_ATM_AAL2ChnlSarSscsStats_t sarSscsStats;
} NPF_F_ATM_ConfigMgr_AAL2ChannelStats_t;
```

4.6 ATM Interface Data Types

4.6.1 ATM Interface Type

This enumeration defines the different ATM interface types. The type of interface is used to determine the number of bits to parse in the ATM header when classifying ATM cells.

```
typedef enum {
    NPF_F_ATM_IF_UNI = 0,  /* UNI Interface */
    NPF_F_ATM_IF_NNI       /* NNI Interface */
} NPF_F_ATM_IfType_t;
```

4.6.2 ATM Interface Characteristics

This structure is used to configure the attributes of an interface.

The cpId field is set to the identifier of the reference points defined at the ATM layer along a VPC (or VCC). The connection points are located at the ingress and egress of an ATM network element, where VP/VC link termination functions operate. Different values are intended to be used for identifying the ingress and egress of a given ATM network element.

```
typedef struct {
    NPF_F_ATM_IfID_t       ifID;      /* Interface handle               */
    NPF_F_ATM_IfType_t     ifType;    /* Interface UNI/NNI              */
    NPF_F_ATM_OAM_CPID_t   cpId;      /* Connect point ID               */
} NPF_F_ATM_ConfigMgr_IfCfg_t;
```

4.6.3 ATM Interface Statistics

This structure is used to return the statistics counters accumulated for each configured interface.

```
typedef struct {
    NPF_uint64_t      cellsClp01Rx;   /* Receive Total Cells (CLP0 + CLP1) */
    NPF_uint64_t      cellsClp0Rx;    /* Receive High Priority Cells (CLP0)*/
    NPF_uint64_t      cellsClp01Tx;   /* Transmit Total Cells (CLP0 + CLP1)*/
    NPF_uint64_t      cellsClp0Tx;    /* Transmit High Priority Cells(CLP0)*/
    NPF_uint32_t      unexCellsRx;    /* Receive cells w/unexpected VPI/VCI*/
    NPF_uint32_t      unexSecsRx;     /* Rx Seconds of Unexpected VPI/VCI  */
    NPF_F_ATM_VcAddr_t lastUnexATMHdr;/* ATM header of last unexpected cell*/
} NPF_F_ATM_ConfigMgr_IfStats_t;
```

4.7 ATM OAM Data Types

4.7.1 Connection Point type

This structure defines the different configurations in which a connection point can exist with respect to an OAM flow.

```
/*
* OAM Connection Point Type
*/
typedef enum{
   NPF_ATM_OAM_ETE_ENDPOINT         = 0, /* ETE End point                  */
   NPF_ATM_OAM_SEG_ENDPOINT         = 1, /* Segment end point              */
   NPF_ATM_OAM_ETE_AND_SEG_ENDPOINT = 2, /* ETE & segment endpoint         */
   NPF_ATM_OAM_ETE_INTERMEDIATE     = 3, /* Intermediate point - ETE flow  */
   NPF_ATM_OAM_ETE_AND_SEG_INTERMEDIATE = 4 /* Intermediate point - ETE
                                               And segment flow           */
} NPF_F_ATM_OAM_CP_Type_t;
```

The ATM layer contains the two highest OAM levels, F4 and F5, in B-ISDN protocol reference model. The two OAM levels are as below:

- F4 level : Virtual path level
- F5 level : Virtual channel level

OAM flows are related to bidirectional Maintenance Entities (MEs) corresponding to either 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.

The F4 flow is further classified into two kinds:

- End-to-end F4 flows: This flow, identified by a standardized VCI is used for end-to-end VPC operations communications.
- Segment F4 flow: This flow, identified by a standardized VCI is used for communicating operations information within the boundaries of the VPC segment.

End-to-end F4 flows must be terminated at the endpoints of a VPC. Segment F4 flows must be terminated at the CPs terminating a VPC segment. The terminating CPs can be intermediate CPs along the VPC or they can coincide with a VPC endpoint.

The F5 flow is further classified into two kinds:

- End-to-end F5 flows: This flow, identified by a standardized PTI, is used for end-to-end VCC operations communications.
- Segment F5 flow: This flow, identified by a standardized PTI, is used for communicating operations information within the boundaries of the VCC segment

End-to-end F5 flows must be terminated at the endpoints of a VCC. Segment F5 flows must be terminated at CPs terminating a VCC segment. The terminating CPs can be intermediate CPs along the VCC or they can coincide with a VCC endpoint.

Configuring the OAM CP as an ETE and/or segment end point enables the AIS/RDI detection/generation. The other OAM procedures like performance monitoring etc. may be enabled after the connection point is configured.

4.7.2 Connection Point ID

The connection point ID identifies a connection point along a VPC or a VCC. Different values are intended to be used for identifying the ingress and egress of a given ATM network element.

```
typedef struct {
      NPF_uchar8_t    cpId[16];     /* Connection point ID */
} NPF_F_ATM_OAM_CPID_t;
```

4.7.3 Response to CC activation request

This structure contains the possible responses to a Continuity Check activate request received by a CP from a remote end. The activate/de-active requested by the remote end is indicated to the registered event handlers. The FAPI client is expected to accept or reject the request received from the remote end for the continuity check procedure.

```
/*
* OAM CC Response Type
*/
typedef enum {
   NPF_F_ATM_OAM_CC_RSP_ACCEPT = 0, /* Accept requested procedure */
   NPF_F_ATM_OAM_CC_RSP_REJECT = 1, /* Reject requested procedure */
} NPF_F_ATM_OAM_CC_RspType_t;
```

4.7.4 Response to PM activation request

This structure contains the possible responses to a Performance Monitoring activate request received by a CP from a remote end. The activate/de-active requested by the remote end is indicated to the registered event handlers. The FAPI client is expected to accept or reject the request received from the remote end for the performance monitoring procedure.

```
/*
* OAM PM Response
*/
typedef enum {
    NPF_IF_ATM_OAM_PM_RSP_ACCEPT = 0, /* Accept requested procedure */
    NPF_IF_ATM_OAM_PM_RSP_REJECT = 1, /* Reject requested procedure */
} NPF_F_ATM_OAM_PM_RspType_t;
```

4.7.5 OAM Flow Level

The ATM layer contains the two highest OAM levels, F4 and F5, in B-ISDN protocol reference model. The two OAM levels are as below:

- F4 level : Virtual path level
- F5 level : Virtual channel level

OAM flows are related to bidirectional Maintenance Entities (MEs) corresponding to either 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.

```
/*
* OAM Flow Level
*/
typedef enum{
      NPF_F_ATM_OAM_FLOW_LEVEL_F4 = 0,      /* F4 level flow */
      NPF_F_ATM_OAM_FLOW_LEVEL_F5 = 1,      /* F5 Level flow */
} NPF_F_ATM_OAM_FlowLevel_t;
```

4.7.6 OAM Flow type

The ATM layer besides providing the vertical subdivision into F4 and F5 levels for OAM flows also provides a horizontal partition i.e. both flows can either cover the entire virtual connection (end-to-end flow) or only parts of the virtual connection (segment flow). The below data type is used to enumerate the two horizontal partitions for the OAM flows.

```
/*
* OAM Flow Type
*/
typedef enum{
   NPF_F_SEGMENT = 0,               /* Flow type SEGMENT */
   NPF_F_END_TO_END = 1,            /* Flow type ETE     */
} NPF_F_ATM_OAM_FlowType_t;
```

4.7.7 Operation Type for OAM procedures

The continuity check and performance monitoring procedures between two connection points may be activated using either activation/de-activation procedures specified in recommendation I.610 or via other means like TMN. The operation type NPF_F_ACTIVATE and NPF_F_DEACTIVATE are used by the FAPI client to request start or stop the CC and PM procedures using the activation/de-activation cell exchange.

When the activation/de-activation of the procedure have been done through other means like TMN, the FAPI client can request the initiation/suspension of the CC and PM procedures using the NPF_F_START and NPF_F_STOP operations. When the NPF_F_START and NPF_F_STOP operations are invoked, no activation/de-activation cell exchange is carried out prior to initiating/suspending the requested procedure.

Intermediate connections points may non-intrusively perform performance monitoring by accumulating statistics for the user traffic on the monitored flows. The non intrusive performance monitoring may be enabled/disabled using the NPF_F_START/NPF_F_STOP operations at connection points marked as being intermediate connection points. These connection points will not generate any FPM/BR cells as part of this performance monitoring.

```
/*
* OAM Oper Type
*/
typedef enum{
   NPF_F_ACTIVATE    = 0,  /* Activate procedure */
   NPF_F_DEACTIVATE  = 1,  /* Deactive procedure */
   NPF_F_START       = 2,  /* Start procedure    */
   NPF_F_STOP        = 3,  /* Stop Procedure     */
} NPF_F_ATM_OAM_OperType_t;
```

4.7.8 OAM Cell Type

The OAM cells are classified based on the OAM type and the function type in the OAM cells. The below enumeration provides the various types of OAM cells.

```
/*
* OAM Fault Management         (AIS/RDI/CC/LB)
* OAM Performance Management   (FPM/BR)
* OAM Activation/Deactivation (FPM_BR/CC/FPM)
*/
typedef enum{
   NPF_F_AIS_CELL = 0,          /* Alarm Indication Signal Cell            */
   NPF_F_RDI_CELL = 1,          /* Remote Defect Indicator Cell            */
   NPF_F_CC_CELL  = 2,          /* Continuity Check Cell                   */
   NPF_F_LB_CELL  = 3,          /* Loopback Cell                           */
   NPF_F_FPM_CELL = 4,          /* Forward Performance Monitoring Cell     */
   NPF_F_BR_CELL  = 5,          /* Backward Reporting Cell                 */
   NPF_F_ACT_DEACT_FPM_BR = 6, /* Activate/Deactive FPM and associated BR */
   NPF_F_ACT_DEACT_CC   = 7,    /* Activate/Deactive Continuity check      */
   NPF_F_ACT_DEACT_FPM  = 8,    /* Activate/Deactive FPM                   */
} NPF_F_ATM_OAM_CellType_t;
```

4.7.9 OAM Performance Monitoring Block Size

These are nominal block size values, and the actual size of the monitored cell block may vary. The cell block size may vary up to a maximum margin of 50% of the value of N for end-to-end performance monitoring. However, for end-to-end performance monitoring, the monitoring cell must be inserted into the user cells stream no more than N/2 user cells after an insertion request has been initiated. The actual monitoring block size averages out to approximately N cells. Recommendation I.356 shows how to choose the block size N.

```
/*
```

```
* OAM PM Block Size
*/
typedef enum{
   NPF_F_SIZE_32768 = 0,   /* Block size 32768 cells */
   NPF_F_SIZE_16384 = 1,   /* Block size 16384 cells */
   NPF_F_SIZE_8192  = 2,   /* Block size  8192 cells */
   NPF_F_SIZE_4096  = 3,   /* Block size  4096 cells */
   NPF_F_SIZE_2048  = 4,   /* Block size  2048 cells */
   NPF_F_SIZE_1024  = 5,   /* Block size  1024 cells */
   NPF_F_SIZE_512   = 6,   /* Block size   512 cells */
   NPF_F_SIZE_256   = 7,   /* Block size   256 cells */
   NPF_F_SIZE_128   = 8    /* Block size   128 cells */
} NPF_F_ATM_OAM_BlkSize_t;
```

4.7.10 OAM Performance Monitoring Function Type

When performance monitoring and backward reporting are both activated it is specified using the enumeration `NPF_F_PM_FUNC_TYPE_FPM_BR`. When only performance monitoring is activated, it is specified using the enumeration `NPF_F_PM_FUNC_TYPE_FPM`.

```
/* OAM PM Function Type */
/* The data type defines PM function types can be set */
typedef enum {
    NPF_F_PM_FUNC_TYPE_FPM_BR = 0,
    NPF_F_PM_FUNC_TYPE_FPM
} NPF_F_ATM_OAM_PM_FuncType_t;
```

4.7.11 Direction of operation of OAM functions

This enumeration identifies the direction(s) of transmission to activate/deactivate OAM function. The A-B and B-A notation is used to differentiate between the direction of transmission away or towards the activator/deactivator, respectively.

```
/*
* OAM Direction
*/
typedef enum{
   NPF_F_FORWARD        = 0,  /* A-B direction                              */
   NPF_F_BACKWARD       = 1,  /* B-A direction                              */
   NPF_F_TWO_WAY        = 2,  /* Both A-B and B-A direction                 */
   NPF_F_NOT_APPLICABLE = 3   /* Direction not applicable to this function */
} NPF_F_ATM_OAM_Direction_t;
```

4.7.12 Continuity Check cell insertion method

Two alternative mechanisms exist for the insertion of CC cells after the activation of the CC function:

- Option 1: CC cells are sent in the forward direction by a VPC/VCC source point or a VPC/VCC segment source point when no user cells have been sent for a period of nominally 1 second.
- Option 2: CC cells are sent repetitively with a periodicity of nominally 1 cell per second independent of the user cells flow.

The below enumeration is used to describe these two options.

```
/*
* OAM Continuity Check Method
*/
typedef enum {
    /* Send CC when no user cells only if no user cells have been sent in
     * CC Duration on link for that flow                                   */
    NPF_F_ATM_OAM_CC_SEND_CC_WHEN_NO_USER_CELLS = 0,
```

```
    /* Send CC cells periodically Irrespective of user cells flowing on    *
     * link for that flow                                                  */
    NPF_F_ATM_OAM_CC_SEND_CC_ALWAYS = 1
} NPF_F_ATM_OAM_CC_Method_t;
```

4.7.13 Connection Point Configuration Data Type

This data type is used to configure the connection point type for a network connection. The connection points are reference points defined along a network connection defined at a given network layer. Connection points defined at the ATM layer along a VPC or a VCC are located at the ingress and egress of an ATM network element where VP or VC link termination functions operate.

If the LLID option is enabled at a CP the CP shall analyze the incoming segment LB cells. Intermediate points shall return segment LB cells if the LLID matches the CPID or if the LLID is set to all "0"s value.

By default all connection points shall be considered to be intermediate connection points for segment and ETE flows. The LLID option shall be considered to be enabled by default. The connection point type and the state of the LLID option can be reconfigured by issuing the `NPF_F_ATM_ConfigMgr_CP_Set` function.

```
/*
* OAM CP Configuration information
*/
typedef struct {
    NPF_F_ATM_VirtLink_t    linkId;     /* VP or VC Link    */
    NPF_F_ATM_OAM_CP_Type_t cpType;     /* Connection point type           */
    NPF_boolean_t              llidOption; /* Is LLID option enabled ?
*/
} NPF_F_ATM_ConfigMgr_OAM_CP_t;
```

4.7.14 Continuity Check Configuration Data Type

The continuity check procedure for F4 and F5 flows as specified in recommendation I.610 is configured using the below data type.

```
/*
* OAM CC Activation/Deactivation
*/
typedef struct {
    NPF_F_ATM_VirtLink_t         linkId;   /* VP or VC Link */
    NPF_F_ATM_OAM_FlowType_t     flowType; /* segment/end-to-end           */
    NPF_F_ATM_OAM_OperType_t     operType; /*Activate/Deactivate/Start/Stop*/
    NPF_F_ATM_OAM_Direction_t    direction;/* Away/towards/both            */
    NPF_F_ATM_OAM_CC_Method_t    ccMethod; /* Options to send CC Cell
                                              0-Send when no user cell
                                              1-Send periodically          */
} NPF_F_ATM_ConfigMgr_OAM_CC_t;
```

Continuity check procedure may be activated/de-activated as a result of activation/de-activation cell exchange between the two end points or out of band by other means like TMN.

4.7.15 Continuity Check Response Data Type

This data structure is used by the FAPI client to respond to an activation/de-activation request for continuity check procedure as specified in recommendation I.610.

```
/*
* OAM CC Response
*/
typedef struct {
    NPF_F_ATM_VirtLink_t          linkId;   /* VP or VC Link */
    NPF_F_ATM_OAM_FlowType_t      flowType; /* segment/end-to-end     */
```

```
    NPF_F_ATM_OAM_CC_RspType_t     ccRsp;
    NPF_F_ATM_OAM_CC_Method_t   ccMethod;   /* For CC source and ACCEPT only.
                                               Option to send cc cell
                                               0 – to send when no user cell
                                               1 – to force send */
} NPF_F_ATM_ConfigMgr_OAM_CC_Rsp_t;
```

This ccMethod field is valid in the response only when this node will be the source for the continuity check procedure and the response from the management application indicates ACCEPT action for the CC activate request from the requesting CP.

4.7.16 Performance Monitoring Data Type

This data structure is used to activate/de-active/start/stop performance monitoring procedure on the specified OAM flow as described in recommendation I.610.

```
/*
* OAM PM Activation/Deactivation – FPM-BR/FPM
*/
typedef struct {
    NPF_F_ATM_VirtLink_t         linkId;   /* VP or VC Link*/
    NPF_F_ATM_OAM_FlowType_t     flowType; /* segment/end-to-end          */
    NPF_F_ATM_OAM_PM_FuncType_t  funcType; /* FPM-BR/FPM                  */
    NPF_F_ATM_OAM_Direction_t    dir;      /* Direction of operation      */
    NPF_F_ATM_OAM_BlkSize_t      fwdSize;  /* A-B block size              */
    NPF_F_ATM_OAM_BlkSize_t      bwdSize;  /* B-A block size              */
    NPF_F_ATM_OAM_OperType_t     operType; /* Act/Deactivate/Start/Stop   */
} NPF_F_ATM_ConfigMgr_OAM_PM_t;
```

The possible combinations of `funcType, dir` and `operType` are as below:

Table 4.5: Performance Monitoring Configurations

funcType	dir	operType	Description
FPM	A-B	Start	The performance monitoring is activated in this case by means like the TMN and no activation OAM cells are exchanged by the connection point A and B. The source process for the PM function is activated at connection point ‘A’ for FPM only performance monitoring in the A-B direction. The connection point ‘A’ counts the user cells transmitted in the A-B direction for the specified OAM flow and generates the FPM cells in the A-B direction when the configured block size number of cells has been transmitted.
FPM	B-A	Start	The performance monitoring is activated in this case by means like the TMN and no activation OAM cells are exchanged by the connection point A and B. The sink process for the PM function is activated at connection point ‘A’ for FPM only performance monitoring in the B-A direction. The connection point ‘A’ counts the user cells received in the B-A direction for the specified OAM flow and updates the performance monitoring statistics for the B-A direction on reception of FPM cell from connection point B.
FPM	Both	Start	The performance monitoring is activated in this case by means like

			the TMN and no activation OAM cells are exchanged by the connection point A and B. The source process for PM is activated at the connection point 'A' for FPM only performance monitoring in the A-B direction and the sink process for PM is activated at connection point 'A' for FPM only performance monitoring in the B-A direction. The connection point 'A' counts the user cells transmitted in the A-B direction for the specified OAM flow and generates the FPM cells in the A-B direction when the configured block size number of cells has been transmitted. The connection point 'A' counts the user cells received in the B-A direction for the specified OAM flow and updates the performance monitoring statistics for the B-A direction when FPM cell is received from connection point B.
FPM	A-B	Activate	The performance monitoring is activated in this case by an exchange of activation OAM cells between the initiator i.e. connection point 'A' and the connection point 'B' to activate performance monitoring in the A-B direction. The connection point 'A' generates an activation cell to the connection point B. If the activation request is accepted by the connection point B, the source process for the PM is activated at connection point 'A' for FPM only performance monitoring in the A-B direction. The connection point 'A' starts counting the user cells transmitted in the A-B direction for the specified OAM flow and generates the FPM cells in the A-B direction when the configured block size number of cells has been transmitted.
FPM	B-A	Activate	The performance monitoring is activated in this case by an exchange of activation OAM cells between the initiator i.e. connection point 'A' and the connection point 'B' to activate performance monitoring in the B-A direction. The connection point 'A' generates an activation cell to the connection point B. If the activation request is accepted by the connection point B, the sink process for the PM is activated at connection point 'A' for FPM only performance monitoring in the B-A direction. The connection point 'A' counts the user cells received in the B-A direction for the specified OAM flow and updates the performance monitoring statistics for the B-A direction on reception of FPM cell from connection point B.
FPM	Both	Activate	The performance monitoring is activated in this case by an exchange of activation OAM cells between the initiator i.e. connection point 'A' and the connection point 'B' to activate performance monitoring in both A-B and B-A direction. The connection point 'A' generates an activation cell to the connection

			point B. If the activation request is accepted by the connection point B, the sink process for the PM is activated at connection point 'A' for FPM only performance monitoring in B-A direction and the source process for the PM is activated at the connection point 'A' for FPM only performance monitoring in the A-B direction. The connection point 'A' counts the user cells transmitted in the A-B direction for the specified OAM flow and generates the FPM cells in the A-B direction when the configured block size number of cells has been transmitted. The connection point 'A' counts the user cells received in the B-A direction for the specified OAM flow and updates the performance monitoring statistics for the B-A direction on reception of FPM cell from connection point B.
FPM	A-B	Stop	The performance monitoring is de-activated in this case by means like the TMN and no de-activation OAM cells are exchanged by the connection point A and B. The connection point 'A' stops counting of the user cells transmitted in the A-B direction for the specified OAM flow.
FPM	B-A	Stop	The performance monitoring is de-activated in this case by means like the TMN and no de-activation OAM cells are exchanged by the connection point A and B. The connection point 'A' stops counting the user cells received in the B-A direction for the specified OAM flow.
FPM	Both	Stop	The performance monitoring is de-activated in this case by means like the TMN and no de-activation OAM cells are exchanged by the connection point 'A' and 'B'. The connection point 'A' stops counting of the user cells transmitted in the A-B direction for the specified OAM flow. The connection point 'A' stops counting the user cells received in the B-A direction for the specified OAM flow.
FPM	A-B	Deactivate	The performance monitoring is de-activated in this case by an exchange of de-activation OAM cells between the initiator i.e. connection point 'A' and the connection point 'B' to de-activate performance monitoring in the A-B direction. The connection point 'A' generates a de-activation cell to the connection point B. If the de-activation request is accepted by the connection point 'B', the connection Point 'A' stops counting of the user cells transmitted in the A-B direction for the specified OAM flow.

FPM	B-A	Deactivate	The performance monitoring is de-activated in this case by an exchange of de-activation OAM cells between the initiator i.e. connection point ‘A’ and the connection point ‘B’ to de-activate performance monitoring in the B-A direction. The connection point ‘A’ generates a de-activation cell to the connection point B. If the de-activation request is accepted by the connection point ‘B’, the connection point ‘A’ stops counting the user cells received in the B-A direction for the specified OAM flow.
FPM	Both	Deactivate	The performance monitoring is de-activated in this case by an exchange of de-activation OAM cells between the initiator i.e. connection point ‘A’ and the connection point ‘B’ to de-activate performance monitoring in the B-A and B-A direction. The connection point ‘A’ generates a de-activation cell to the connection point B. If the de-activation request is accepted by the connection point ‘B’, the connection point ‘A’ stops counting of the user cells transmitted in the A-B direction and the user cells received in the B-A direction for the specified OAM flow.
FPM-BR	A-B	Start	The performance monitoring is activated in this case by means like the TMN and no activation OAM cells are exchanged by the connection point A and B. The source process for the PM function is activated at connection point ‘A’ for performance monitoring in the A-B direction using FPM and associated BR. The connection point ‘A’ counts the user cells transmitted in the A-B direction for the specified OAM flow and generates the FPM cells in the A-B direction when the configured block size number of cells has been transmitted. On reception of BR cell from connection point B, the performance monitoring statistics for A-B direction are updated.
FPM-BR	B-A	Start	The performance monitoring is activated in this case by means like the TMN and no activation OAM cells are exchanged by the connection point A and B. The sink process for the PM function is activated at connection point ‘A’ for PM in the B-A direction using FPM and associated BR. The connection point ‘A’ counts the user cells received in the B-A direction for the specified OAM flow. When the connection point ‘A’ receives the FPM cell generated by connection point ‘B’, a BR cells is generated by the connection point ‘A’ in the A-B direction and the performance monitoring statistics for B-A direction are updated.
FPM-BR	Both	Start	The performance monitoring is activated in this case by means like the TMN and no activation OAM cells are exchanged by the connection point A and B. The sink process for the PM function is activated at connection point ‘A’ for PM in the B-A direction using FPM and associated BR. The source process for PM function is also

			activated at the connection point ‘A’ for PM in the A-B direction using FPM and associated BR. The connection point ‘A’ counts the user cells transmitted in the A-B direction for the specified OAM flow and generates the FPM cells in the A-B direction when the configured block size number of cells has been transmitted. On reception of BR cell from connection point B, the performance monitoring statistics for A-B direction are updated. The connection point ‘A’ counts the user cells received in the B-A direction for the specified OAM flow. When the connection point ‘A’ receives the FPM cell generated by connection point ‘B’, a BR cells is generated by the connection point ‘A’ in the A-B direction and the performance monitoring statistics for B-A direction are updated.
FPM-BR	A-B	Activate	The performance monitoring is activated in this case by an exchange of activation OAM cells between the initiator i.e. connection point ‘A’ and the connection point ‘B’ to activate performance monitoring in the A-B direction using FPM and associated BR. The connection point ‘A’ generates an activation cell to the connection point B. If the activation request is accepted by the connection point B, the connection point ‘A’ counts the user cells transmitted in the A-B direction for the specified OAM flow and generates the FPM cells in the A-B direction when the configured block size number of cells has been transmitted. On reception of BR cell from connection point B, the performance monitoring statistics for A-B direction are updated.
FPM-BR	B-A	Activate	The performance monitoring is activated in this case by an exchange of activation OAM cells between the initiator i.e. connection point ‘A’ and the connection point ‘B’ to activate performance monitoring in the B-A direction using FPM and associated BR. The connection point ‘A’ generates an activation cell to the connection point B. If the activation request is accepted by the connection point B, the connection point ‘A’ counts the user cells received in the B-A direction for the specified OAM flow. When the connection point ‘A’ receives the FPM cell generated by connection point ‘B’, a BR cell is generated by the connection point ‘A’ in the A-B direction and the performance monitoring statistics for the B-A direction are updated.
FPM-BR	Both	Activate	The performance monitoring is activated in this case by an exchange of activation OAM cells between the initiator i.e. connection point ‘A’ and the connection point ‘B’ to activate performance monitoring in both A-B and B-A direction using FPM and associated BR. The connection point ‘A’ generates an

			activation cell to the connection point B. If the activation request is accepted by the connection point B, the connection point 'A' counts the user cells transmitted in the A-B direction for the specified OAM flow and generates the FPM cells in the A-B direction when the configured block size number of cells has been transmitted. On reception of BR cell from connection point B, the performance monitoring statistics for A-B direction are updated. If the activation request is accepted by the connection point B, the connection point 'A' counts the user cells received in the B-A direction for the specified OAM flow. When the connection point 'A' receives the FPM cell generated by connection point 'B', a BR cells is generated by the connection point 'A' in the A-B direction and the performance monitoring statistics for B-A direction are updated.
FPM-BR	A-B	Stop	The performance monitoring is de-activated in this case by means like the TMN and no de-activation OAM cells are exchanged by the connection point A and B. Connection points 'A' stops counting user cells transmitted in the A-B direction on the specified OAM flow.
FPM-BR	B-A	Stop	The performance monitoring is de-activated in this case by means like the TMN and no de-activation OAM cells are exchanged by the connection point A and B. Connection points 'A' stops counting user cells received in the B-A direction on the specified OAM flow.
FPM-BR	Both	Stop	The performance monitoring is de-activated in this case by means like the TMN and no de-activation OAM cells are exchanged by the connection point A and B. Connection points 'A' stops counting user cells received in the B-A direction and user cells transmitted in the A-B direction on the specified OAM flow.
FPM-BR	A-B	Deactivate	The performance monitoring is de-activated in this case by an exchange of de-activation OAM cells between the initiator i.e. connection point 'A' and the connection point 'B' to de-activate performance monitoring in the A-B direction. The connection point 'A' generates a de-activation cell to the connection point B. If the de-activation request is accepted by the connection point 'B', the connection points 'A' stops counting user cells transmitted in the A-B direction on the specified OAM flow.
FPM-BR	B-A	Deactivate	The performance monitoring is de-activated in this case by an exchange of de-activation OAM cells between the initiator i.e. connection point 'A' and the connection point 'B' to de-activate

			performance monitoring in the B-A direction. The connection point 'A' generates a de-activation cell to the connection point B. If the de-activation request is accepted by the connection point 'B', the connection points 'A' stops counting user cells received in the B-A direction on the specified OAM flow.
FPM-BR	Both	Deactivate	The performance monitoring is de-activated in this case by an exchange of de-activation OAM cells between the initiator i.e. connection point 'A' and the connection point 'B' to de-activate performance monitoring in the B-A and A-B direction. The connection point 'A' generates a de-activation cell to the connection point B. If the de-activation request is accepted by the connection point 'B', the connection points 'A' stops counting user cells received in the B-A direction and the user cells transmitted in the A-B direction on the specified OAM flow.

4.7.17 Performance Monitoring Statistics Request Data Type

This section defines the data type used by the FAPI client to request the accumulated performance monitoring counters. As part of the request the FAPI client may request the accumulated statistics to be zeroed. Zeroing out the counters does not stop/disable the ongoing performance monitoring procedure.

The ATM configuration manager provides two mechanisms to retrieve performance monitoring statistics. One is via a registered event handler in which case the FAPI implementation generates performance monitoring statistics events whenever a FPM or BR cell is received to deliver the collected statistics. The other mechanism is via the `NPF_F_ATM_ConfigMgr_OAM_PM_StatsGet` function call to retrieve the statistics.

```
/* Performance monitoring statistics request data type */
typedef struct {
    NPF_F_ATM_VirtLink_t      linkId;             /* VP or VC Link       */
    NPF_F_ATM_OAM_FlowType_t  flowType;           /* segment/end-to-end  */
    NPF_boolean_t             zeroStats;          /* clear stats if TRUE */
} NPF_F_ATM_ConfigMgr_OAM_PM_StatsReq_t;
```

4.7.18 Performance Monitoring Statistics Data Type

The ATM configuration manager allows the FAPI client to obtain the performance monitoring statistics using two methods. The first method is to register an event handler to receive the performance monitoring statistics. The FAPI implementation will deliver the accumulated statistics for each monitored cell block on reception of a FPM or BR cell.

The second method is for the FAPI client to query the ATM configuration manager for the statistics information. The FAPI implementation will accumulate and integrate the statistics and provide accumulated statistics in response to the query. This section defines the data type used to report the accumulated statistics in response to the query from the FAPI client.

```
/* Errored cell block statistics; Does not include counts for severely  *
 * errored cell blocks which are provided as part of SECB stats         */
typedef struct {
    /*Number of misinserted CLP0+1 cells in errored cell blocks         */
    NPF_uint32_t      numMisinsertedCLP01cells;
    /*Number of misinserted CLP0 cells in errored cell blocks           */
    NPF_uint32_t      numMisinsertedCLP0cells;
    /* Number of lost CLP0+1 users cells in errored cell blocks         */
```

```
    NPF_uint32_t      numLostCLP01cells;
    /* Number of lost CLP0 users cells in errored cell blocks          */
    NPF_uint32_t      numLostCLP0cells;
    /* Number of errored cell bits in errored cell blocks              */
    NPF_uint32_t      numErroredCLP01Bits;
    /* Number of cell blocks with CLP0+1 losses                        */
    NPF_uint32_t      numBlocksClp01Loss;
    /* Number of cell blocks with CLP0 losses                          */
    NPF_uint32_t      numBlocksClp0Loss;
    /* Number of cell blocks with CLP0+1 Misinsertions                 */
    NPF_uint32_t      numBlocksClp01Misinsertion;
    /* Number of cell blocks with CLP0 Misinsertions                   */
    NPF_uint32_t      numBlocksClp0Misinsertion;
    /* Number of cell blocks with errors i.e cell loss, misinsertion or *
     * bit errors                                                      */
    NPF_uint32_t      totalErroredCellBlocks;
} NPF_F_ATM_OAM_ErrCB_Stats_t;
/* Severely errored cell block (SECB) statistics. Please see I.610 and I.356
 * For definition of severely errored cell blocks.
 * The counts here do not include the non severely errored cell blocks
 */
typedef struct {
    /*Number of misinserted CLP0+1 cells in severely errored cell blocks*/
    NPF_uint32_t      numMisinsertedCLP01cells;
    /*Number of misinserted CLP0 cells in severely errored cell blocks  */
    NPF_uint32_t      numMisinsertedCLP0cells;
    /* Number of lost CLP0+1 users cells in severely errored cell blocks*/
    NPF_uint32_t      numLostCLP01cells;
    /* Number of lost CLP0 users cells in severely errored cell blocks  */
    NPF_uint32_t      numLostCLP0cells;
    /* Number of errored cell bits in severely errored cell blocks     */
    NPF_uint32_t      numErroredCLP01Bits;
    /* Number of severely errored cell blocks with CLP0+1 losses       */
    NPF_uint32_t      numSECBClp01Loss;
    /* Number of severely errored cell blocks with CLP0 losses         */
    NPF_uint32_t      numSECBClp0Loss;
    /* Number of severely errored cell blocks with CLP0+1 Misinsertions */
    NPF_uint32_t      numSECBClp01Misinsertion;
    /* Number of severely errored cell blocks with CLP0 Misinsertions   */
    NPF_uint32_t      numSECBClp0Misinsertion;
    /* Total Number of severely errored cell blocks with errors i.e     *
     * cell loss, misinsetion or bit errors                            */
    NPF_uint32_t      totalSECB;
} NPF_F_ATM_OAM_SevErrCB_Stats_t;

/* This structure contains the PM stats counters for one direction      */
typedef struct {
    NPF_uint32_t                  averageBlockSize; /* Avg. block size */
    NPF_uint32_t                  minBlockSize;     /* Min block size  */
    NPF_uint32_t                  maxBlockSize;     /* Max block size  */

    /* Lost PM cells is the count of FPM cell lost in the B->A direction *
     * and the lost of BR cells in the A->B direction                  */
    NPF_uint32_t                  lostPMcells;      /*  PM cells lost  */

    /* Errored cell block statistics */
    NPF_F_ATM_OAM_ErrCB_Stats_t     erroredCBStats;   /* Errored CB stats*/
    /* Severely errored cell block statistics */
    NPF_F_ATM_OAM_SevErrCB_Stats_t  secbStats;        /* SECB stats      */
```

```
} NPF_F_ATM_OAM_PM_Stats_Info_t;

/* Performance monitoring statistics returned in response to FAPI client *
 * query. The counts are accumulated since the last time the counters    *
 * were zeroed by the FAPI client. When performance monitoring is enabled*
 * the counters start from 0.                                            */

typedef struct {
    /* Direction in which performance management procedure is activated *
     * is either Forward (A->B), Backward (B->A) or both A->B and B->A  */
    NPF_F_ATM_OAM_Direction_t    dir;

    /* Performance monitoring statistics for A->B direction. The counter*
     * in this structure are valid if PM is enabled in forward direction*
     * i.e. A->B direction or in both directions. These counters are    *
     * updated on reception of a BR from the B connection point         *
     * The direction field indicates direction in which PM is enabled   */
    NPF_F_ATM_OAM_PM_Stats_Info_t   oamPMStatsAtoB;

    /* Performance monitoring statistics for B->A direction. The counter*
     * in this structure are valid if PM is enabled in forward direction*
     * i.e. B->A direction or in both directions. These counters are    *
     * updated on reception of a FPM from the B connection point        *
     * The direction field indicates direction in which PM is enabled   */
    NPF_F_ATM_OAM_PM_Stats_Info_t   oamPMStatsBtoA;
} NPF_F_ATM_ConfigMgr_OAM_PM_Stats_t;
```

4.7.19 Loopback Operation Data Type

This data structure is used to initiate a loopback procedure as described in recommendation I.610. VP and VC loopback cells may be inserted by any connection point along a VPC or VCC respectively.

The `llId` field identifies the CP along the virtual connection or connection segment where the loopback is to occur. The `llId` is set to one of the below depending on the desired loopback operation.

- All 1s – Represents the endpoint. This is the segment endpoint for segment loopback and the connection endpoint for end-to-end loopback.
- All 0s – Represents all CPs for which the LLID option is enabled. This includes the segment endpoint. It is only applicable to segment LB cells.
- All other values – Indicates the specific CP where the loopback is to occur.

When the `includeSrcId` field is set to `NPF_TRUE`, the CPID configured for the interface on which the link is created is included in the LB cell. When set to `NPF_FALSE`, the default all 1's value shall be used to form the LB cell.

The `remCell` field indicates if the CP which inserted the LB cell should remove the returned LB cells. The CP shall remove the LB cell if the `remCell` is set to `NPF_TRUE` and the correlation tag and source ID in the returned cell match those in the parent cell.

```
/*
* OAM Loopback
*/
typedef struct {
    NPF_F_ATM_VirtLink_t     linkId;     /* VP or VC Link                */
    NPF_F_ATM_OAM_FlowType_t flowType;   /* segment/end-to-end           */
    NPF_F_ATM_OAM_CPID_t     llId;       /* Loopback location Id         */
    NPF_boolean_t            remCell;    /* Remove returned cells ?      */
    NPF_boolean_t          includeSrcId; /* If source CP ID be included  */
                                         /* in LB cell                   */
} NPF_F_ATM_ConfigMgr_OAM_LB_t;
```

4.7.20 Loopback Procedure Result Data Type

In response to the loopback cell the CP sending the loopback request could receive multiple loopback responses. The `NPF_F_ATM_ConfigMgr_OAM_LB_Result_t` structure is used to return the responses received to the loopback procedure. The `numLbResp` may be set to 1 when the results are returned one by one. Alternately all loopback responses may be collected in a single result structure and the `numLbResp` set to the received number of responses. It is an implementation decision to return the results one by one or all at once.

If no loopback cells are received and the loopback procedure times out, the `numLbResp` shall be set to 0 in the `NPF_F_ATM_ConfigMgr_OAM_LB_Result_t` structure used to return the response to the FAPI client indicating a failure of the loopback procedure.

```
/*
*   Loopback Procedure Result
*/
typedef struct {
    NPF_uint8_t           numLbResp;       /* Number of Loopback responses */
    NPF_F_ATM_OAM_CPID_t  *llId;           /* Loopback Location Id         */
} NPF_F_ATM_ConfigMgr_OAM_LB_Result_t;
```

4.7.21 Non Intrusive Monitoring Data Type

Non-intrusive monitoring of any type of end-to-end or segment fault and performance management VP/VC OAM flows may be performed at any intermediate point along a VPC/VCC. This includes intermediate points within a VPC/VCC segment as well as the VPC/VCC segment endpoints. The purpose of the non-intrusive monitoring function is to provide to network providers additional OAM information which cannot be derived from the content of segment OAM flows.

Non-intrusive monitoring of VP/VC OAM flows consists of detecting and processing the content of VP/VC OAM cells received on a VP/VC link in passing at an intermediate point without modifying the characteristics (e.g. cell content, cell sequence) of the aggregated (OAM & monitored cells) flow observed.

Enabling non intrusive monitoring of fault management cells (AIS/RDI/CC/LB) at intermediate points allows the connection point to detect faults and declare AIS state when AIS cells are received, transmission path AIS-defects are detected or defects like loss of continuity is detected on the monitored flow (segment/end-to-end) on the monitored link (VP/VC link). The AIS (segment_VP-AIS, e-t-e_VP-AIS, segment_VC-AIS, e-t-e_VC-AIS) state shall be released when a user cell or a CC cell is received on the monitored flow. The AIS state shall also be released if no AIS cells are seen on the monitored flow for 2.5 +/- 0.5 seconds. When non-intrusive monitoring is stopped for a specified flow, the AIS states declared for that flow shall be released. The declaration/release of AIS state and detection/clearing of the LOC defect on monitored flows shall be indicated to the FAPI clients using the fault management events listed below:

- `NPF_F_ATM_AIS_RAISED`
- `NPF_F_ATM_RDI_RAISED`
- `NPF_F_ATM_AIS_CLEARED`
- `NPF_F_ATM_RDI_CLEARED`
- `NPF_F_ATM_LOC_RAISED`
- `NPF_F_ATM_LOC_CLEARED`

When starting the non-intrusive monitoring of CC flows the FAPI client should by means outside the scope of NPF determine whether the CC procedure is activated for the monitored flow and link to avoid generation of spurious LOC defects. Also the FAPI client should by means outside the scope of NPF stop monitoring of CC flows when the CC procedure is deactivated at the end points terminating the monitored flow.

The FAPI client may additionally register for events delivering the contents of the monitored cells types.

```
/*
* OAM Non-Intrusive Monitoring
*/
typedef struct {
    NPF_F_ATM_VirtLink_t     linkId;      /* VP or VC Link               */
    NPF_F_ATM_OAM_OperType_t operType;    /* Start/Stop Monitoring       */
    NPF_F_ATM_OAM_FlowType_t flowType;    /* segment/end-to-end          */
    NPF_F_ATM_OAM_CellType_t cellType;    /* Type of OAM cell to monitor */
} NPF_F_ATM_ConfigMgr_OAM_Mon_t;
```

4.7.22 Declare/Release AIS Alarm State Data Type

This structure is used to declare or release AIS alarm state for the specified link at the connection point. The defect type specified in the below structure is not fully specified as the specific enumerations for defect type are for further study as specified in recommendation I.610.

When AIS state is declared for a VP, the AIS state is automatically declared for all VCs included within the VP as specified in recommendation I.610.

```
/*
* OAM Alarms - Declare and Release AIS alarms
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t  flowType;           /* segment/end-to-end  */
    NPF_uint8_t               defectType;         /* Defect type         */
    NPF_uint8_t               defectLocation[16]; /* Defect Location     */
} NPF_F_ATM_OAM_Alarm_Info_t;

typedef struct {
    NPF_F_ATM_VirtLink_t        linkId;             /* VP or VC Link       */
    NPF_F_ATM_OAM_Alarm_Info_t  alarmInfo;          /* Alarm information   */
} NPF_F_ATM_ConfigMgr_OAM_Alarm_t;
```

4.7.23 Response to PM activation/De-activation request

This structure contains the possible responses to a Performance Monitoring activate request received by a CP from a remote end. The activate/de-active requested by the remote end is indicated to the registered event handlers. The FAPI client is expected to accept or reject the request received from the remote end for the performance monitoring procedure using the `NPF_F_ATM_ConfigMgr_OAM_PM_Rsp ()` function.

```
/*
* OAM PM Response
*/
typedef struct {
    NPF_F_ATM_VirtLink_t           linkId;     /* VP or VC Link          */
    NPF_F_ATM_OAM_FlowType_t       flowType;   /* segment/end-to-end     */
    NPF_F_ATM_OAM_PM_RspType_t     pmRsp;      /* Accept/Reject response */
} NPF_F_ATM_ConfigMgr_OAM_PM_Rsp_t;
```

4.7.24 AIS Cell Information Data Types

This structure is used to pass alarm information received in AIS cells to registered event handlers.

```
/*
*   AIS Alarm event info.
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t   flowType;           /* Segment/ETE         */
    NPF_uint16_t               defectType;         /* Defect type         */
    NPF_char8_t                defectLocation[16]; /* Defect Location     */
```

```
} NPF_F_ATM_OAM_AIS_Event_t;
```

4.7.25 RDI Cell Information Data Types

This structure is used to pass alarm information received in RDI cells to registered event handlers.

```
/*
*   RDI Alarm event info.
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t  flowType;           /* Segment/ETE          */
    NPF_uint16_t              defectType;         /* Defect type          */
    NPF_char8_t               defectLocation[16]; /* Defect Location      */
} NPF_F_ATM_OAM_RDI_Event_t;
```

4.7.26 FPM Cell Information Data Type

This structure is used to pass information received in Forward performance monitoring (FPM) cells to registered event handlers.

```
/*
*   FPM Cell Info
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t  flowType;       /* Segment/ETE               */
    NPF_uchar8_t              mscn;           /* FPM MCSN                  */
    NPF_uint16_t              totUsrCell01;   /* Total User Cell(CLP-0+1)  */
    NPF_uint16_t              totUsrCell0;    /* Total User Cell(CLP-0)    */
    NPF_uint16_t              blkErrDetCode;  /* Block Error Detection Code*/
    NPF_uint32_t              timeStamp;      /* Time Stamp */
} NPF_F_ATM_OAM_FPM_Event_t;
```

4.7.27 BR Cell Information Data Type

This structure is used to pass information received in Back Reporting (BR) cells to registered event handlers.

```
/*
*   BR Cell Info
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t  flowType;     /* Segment/ETE                */
    NPF_uchar8_t              mscn;         /* BR Monitoring Cell Seq. No. */
    NPF_uint16_t              totUsrCell01; /* 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 MCSN               */
    NPF_uchar8_t              secbc;        /* Severely Err. Cell block    */
    NPF_uint16_t              totRcvdUsrCell0;/* Total Rx User Cell(CLP-0) */
    NPF_uint8_t               blkErr;       /* Block Error Result(CLP-0+1) */
    NPF_uint16_t              totRcvdUsrCell1;/* Total Rx User Cell(CLP0+1)*/
} NPF_F_ATM_OAM_BR_Event_t;
```

4.7.28 LB Cell Information Data Type

This structure is used to pass the received loopback cell information to registered event handlers.

```
/*
*   Loopback Cell Info
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t    flowType;        /* Segment/ETE          */
    NPF_boolean_t               sourceIdValid;   /* If sourceID in Ind   */
    NPF_uint8_t                 sourceId[16];    /* source ID            */
    NPF_uint8_t                 llid[16];        /* loopback location ID */
```

```
} NPF_F_ATM_OAM_LB_Event_t;
```

4.7.29 CC Cell Information Data Type

This structure is used to pass the received continuity check cell information to registered event handlers.

```
typedef struct
{
    NPF_F_ATM_OAM_FlowType_t    flowType;      /* Segment/ETE          */
} NPF_F_ATM_OAM_CC_Event_t;
```

4.7.30 Performance Monitoring Statistics Event Data Type

This structure is used to pass the performance monitoring statistics to registered event handlers. The delivery of the statistics is triggered by reception of either FPM or BR cell and provides the statistics for one cell block i.e. the cell block for which the FPM or BR was received.

The ATM configuration manager provides two mechanisms to retrieve performance monitoring statistics. One via a registered event handler in which case the FAPI implementation generates performance monitoring statistics events whenever a FPM or BR cell is received to deliver the collected statistics. The other mechanism is the `NPF_F_ATM_ConfigMgr_OAM_PM_StatsGet` function call to retrieve the statistics.

```
typedef enum
{
   NPF_F_ATM_OAM_PM_STATS_TRIGGER_FPM_RECEIVED=0, /* Indication due to FPM */
   NPF_F_ATM_OAM_OAM_PM_STATS_TRIGGER_BR_RECEIVED /* Indication due to BR  */
} NPF_F_ATM_OAM_PmStatsTrigger_t;

typedef struct
{
    NPF_F_ATM_OAM_FlowType_t     flowType;         /* Segment/ETE          */
    NPF_F_ATM_OAM_PmStatsTrigger_t pmStatsTrigger; /* FPM or BR received   */
    NPF_uint8_t              mcsn;   /* BR Monitoring cell seq no          */
    NPF_uint16_t             tuc01;  /* Total user cell (CLP 0+1)          */
    NPF_uint16_t             tuc0;   /* Total user cell (CLP 0)            */
    NPF_uint32_t             tstp;   /* Time stamp                         */
    NPF_uint8_t              rMcsn;  /* Reported Monitoring Cell seq No.   */
    NPF_uint8_t              secbc;  /* Severely Errored Cell Block Count  */
    NPF_uint16_t             trcc01; /* Total received user cell(CLP 0+1)  */
    NPF_uint16_t             trcc0;  /* Total received user cell(CLP 0)    */
    NPF_uint8_t              bler01; /* Block error result (CLP 0+1)       */
} NPF_F_ATM_OAM_PM_Stats_Event_t;
```

4.7.31 PM procedure activation and deactivation event

This structure contains the information received in a PM activate/deactivate request from a peer network element.

```
typedef struct {
    NPF_F_ATM_OAM_FlowType_t      flowType;       /* Segment/ETE          */
    NPF_F_ATM_OAM_PM_FuncType_t   functionType;   /* FPM-BR/FPM           */
    NPF_F_ATM_OAM_Direction_t     oamDirection;   /* A->B/B->A/Bothway    */
    NPF_F_ATM_OAM_BlkSize_t       fwdBlkSize;     /* Forward block size   */
    NPF_F_ATM_OAM_BlkSize_t       backBlkSize;    /* Backward block size  */
} NPF_F_ATM_OAM_PM_ActDeact_Event_t;
```

4.7.32 CC procedure activation and deactivation event

This structure contains the information received in a CC activate/deactivate request from a peer network element.

```
typedef struct {
```

```
    NPF_F_ATM_OAM_FlowType_t      flowType;    /* Segment/ETE          */
    NPF_F_ATM_OAM_Direction_t  oamDirection;   /* A->B/B->A/Bothway    */
} NPF_F_ATM_OAM_CC_ActDeact_Event_t;
```

4.7.32.1 OAM Information

This structure contains the state of various OAM procedures and the current OAM status of the queried virtual link.

```
/*
* OAM Configuration and Status Information
*/
typedef struct {
    /* Continuity Check state and configurations                          */
    NPF_boolean_t                ccActive;   /* Whether CC procedure ON */
    /* Below two fields valid if ccActive set to TRUE                     */
    NPF_F_ATM_OAM_Direction_t    ccDir;      /* Away/towards/both       */
    NPF_F_ATM_OAM_CC_Method_t    ccMethod;   /* Options to send CC Cell
                                               0-Send when no user cell
                                               1-Send periodically     */

    /* Performance Monitoring state and configurations                    */
       NPF_boolean_t             pmActive;   /* Whether PM procedure ON */
    /* Below four fields valid if pmActive set to TRUE                    */
    NPF_F_ATM_OAM_PM_FuncType_t  pmFuncType; /* FPM-BR/FPM              */
    NPF_F_ATM_OAM_Direction_t    pmDir;      /* Direction of operation  */
    NPF_F_ATM_OAM_BlkSize_t      pmFwdSize;  /* A-B block size          */
    NPF_F_ATM_OAM_BlkSize_t      pmBwdSize;  /* B-A block size          */

    /* Alarms states                                                      */
    NPF_boolean_t                inAISState; /* Whether in AIS state    */
    NPF_boolean_t                inRDIState; /* Whether in RDI state    */

    /* Loopback procedure configuration                                   */
    NPF_boolean_t                llidOption; /* Is LLID option enabled ?*/

    /* Non intrusive monitoring of OAM flows. When number of monitored  *
     * cell types indicated using numMonCellTypes is set to non zero,   *
     * the list of cell types monitored is provided by *cellType array  */
    NPF_F_uint32_t             numMonCellTypes;
    NPF_F_ATM_OAM_CellType_t  *monCellType; /* OAM cell types monitored */
} NPF_F_ATM_ConfigMgr_OamInfo_t;
```

4.8 ATM Configuration Manager Specific Data Types

4.8.1 LFB Specific Error Codes

This section defines ATM Configuration Manager's configuration and management APIs error codes. These codes are used in callbacks to deliver results of the requested operations. The base for the error codes used in ATM LFBs is derived as `LFB_TYPE_CODE * 100`.

```
/*
* Asynchronous error codes (returned in function callbacks)
*/
    #define NPF_F_ATMCMGR_BASE_ERR    (NPF_F_ATMCONFIGMGR_LFB_TYPE * 100)
    #define NPF_ATM_F_E_INVALID_VC_ADDRESS      (NPF_F_ATMCMGR_BASE_ERR + 0)
    #define NPF_ATM_F_E_INVALID_ATM_AAL         (NPF_F_ATMCMGR_BASE_ERR + 1)
    #define NPF_ATM_F_E_INVALID_CHILD_HANDLE    (NPF_F_ATMCMGR_BASE_ERR + 2)
    #define NPF_ATM_F_E_VC_NOT_BOUND            (NPF_F_ATMCMGR_BASE_ERR + 3)
    #define NPF_ATM_F_E_INVALID_ATM_QOS         (NPF_F_ATMCMGR_BASE_ERR + 4)
    #define NPF_ATM_F_E_INVALID_ATTRIBUTE       (NPF_F_ATMCMGR_BASE_ERR + 5)
```

```
    #define NPF_ATM_F_E_INVALID_SUBTYPE         (NPF_F_ATMCMGR_BASE_ERR + 6)
    #define NPF_ATM_F_E_INVALID_CBR             (NPF_F_ATMCMGR_BASE_ERR + 7)
    #define NPF_ATM_F_E_INVALID_CLK_REC_TYPE    (NPF_F_ATMCMGR_BASE_ERR + 8)
    #define NPF_ATM_F_E_INVALID_FEC             (NPF_F_ATMCMGR_BASE_ERR + 9)
    #define NPF_ATM_F_E_INVALID_SDT             (NPF_F_ATMCMGR_BASE_ERR + 10)
    #define NPF_ATM_F_E_INVALID_PAR_FILL_CELL   (NPF_F_ATMCMGR_BASE_ERR + 11)
    #define NPF_ATM_F_E_INVALID_CELL_LOSS_PER   (NPF_F_ATMCMGR_BASE_ERR + 12)
    #define NPF_ATM_F_E_INVALID_CPS_ATTR        (NPF_F_ATMCMGR_BASE_ERR + 13)
    #define NPF_ATM_F_E_INVALID_SSCS_I3661_ATTR (NPF_F_ATMCMGR_BASE_ERR + 14)
    #define NPF_ATM_F_E_INVALID_SSCS_I3662_ATTR (NPF_F_ATMCMGR_BASE_ERR + 15)
    #define NPF_ATM_F_E_INVALID_ATM_TRUNK_ATTR  (NPF_F_ATMCMGR_BASE_ERR + 16)
    #define NPF_ATM_F_E_INVALID_MAX_SIZE_FWD    (NPF_F_ATMCMGR_BASE_ERR + 17)
    #define NPF_ATM_F_E_INVALID_MAX_SIZE_BWD    (NPF_F_ATMCMGR_BASE_ERR + 18)
    #define NPF_ATM_F_E_INVALID_AAL_MODE        (NPF_F_ATMCMGR_BASE_ERR + 19)
    #define NPF_ATM_F_E_INVALID_SSCS_TYPE       (NPF_F_ATMCMGR_BASE_ERR + 20)
    #define NPF_ATM_F_E_AAL2_QOS_PRIO_INVALID   (NPF_F_ATMCMGR_BASE_ERR + 21)
    #define NPF_ATM_F_E_AAL2_QOS_WEIGHT_INVALID (NPF_F_ATMCMGR_BASE_ERR + 22)
    #define NPF_ATM_F_E_PERF_MONITORING_OFF     (NPF_F_ATMCMGR_BASE_ERR + 23)
    #define NPF_ATM_F_E_CONT_OBJS_EXIST         (NPF_F_ATMCMGR_BASE_ERR + 24)
    #define NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP   (NPF_F_ATMCMGR_BASE_ERR + 25)
```

4.8.2 Completion Callback Data Type

```
/*
* Completion Callback Types, to be found in the callback
* data structure, NPF_F_ATM_ConfigMgr_CallbackData_t.
*/
typedef enum NPF_F_ATM_ConfigMgr_CallbackType {
    NPF_F_ATM_CONFIGMGR_IF_SET = 1,
    NPF_F_ATM_CONFIGMGR_IF_DELETE = 2,
    NPF_F_ATM_CONFIGMGR_IF_STATS_GET = 3,
    NPF_F_ATM_CONFIGMGR_IF_STATS_ENABLE = 4,
    NPF_F_ATM_CONFIGMGR_IF_STATS_DISABLE = 5,
    NPF_F_ATM_CONFIGMGR_IF_QUERY = 6,
    NPF_F_ATM_CONFIGMGR_IF_AIS_STATE_SET = 7,
    NPF_F_ATM_CONFIGMGR_IF_AIS_STATE_CLEAR = 8,

    NPF_F_ATM_CONFIGMGR_VC_SET = 9,
    NPF_F_ATM_CONFIGMGR_VP_SET = 10,
    NPF_F_ATM_CONFIGMGR_VC_BIND = 11,
    NPF_F_ATM_CONFIGMGR_VC_UNBIND = 12,
    NPF_F_ATM_CONFIGMGR_VL_DELETE = 13,
    NPF_F_ATM_CONFIGMGR_VL_ENABLE = 14,
    NPF_F_ATM_CONFIGMGR_VL_DISABLE = 15,
    NPF_F_ATM_CONFIGMGR_VL_OPER_STATUS_GET = 16,
    NPF_F_ATM_CONFIGMGR_VL_STATS_ENABLE = 17,
    NPF_F_ATM_CONFIGMGR_VL_STATS_DISABLE = 18,
    NPF_F_ATM_CONFIGMGR_VC_STATS_GET = 19,
    NPF_F_ATM_CONFIGMGR_VC_QUERY = 20,
    NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_SET = 21,
    NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_DELETE = 22,
    NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_QUERY = 23,
    NPF_F_ATM_CONFIGMGR_VP_STATS_GET = 24,
    NPF_F_ATM_CONFIGMGR_VP_QUERY = 25,
    NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_SET = 26,
    NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_DELETE = 27,
    NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_QUERY = 28,

    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_SET = 29,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_BIND = 30,
```

```
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_UNBIND = 31,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_DELETE = 32,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_GET = 33,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_QUERY = 34,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_ENABLE = 35,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_DISABLE = 36,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_OPER_STATUS_GET = 37,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_ENABLE = 38,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_DISABLE = 39,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_SET = 40,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_DELETE = 41,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_QUERY = 42,

    NPF_F_ATM_CONFIGMGR_OAM_CP_SET = 43,
    NPF_F_ATM_CONFIGMGR_OAM_CC_SET = 44,
    NPF_F_ATM_CONFIGMGR_OAM_CC_RSP = 45,
    NPF_F_ATM_CONFIGMGR_OAM_PM_SET = 46,
    NPF_F_ATM_CONFIGMGR_OAM_PM_RSP = 47,
    NPF_F_ATM_CONFIGMGR_OAM_PM_STATS_GET = 48,
    NPF_F_ATM_CONFIGMGR_OAM_LB_SET = 49,
    NPF_F_ATM_CONFIGMGR_OAM_MONITOR = 50,
    NPF_F_ATM_CONFIGMGR_OAM_ALARM_SET = 51,
    NPF_F_ATM_CONFIGMGR_OAM_ALARM_CLEAR = 52,
} NPF_F_ATM_ConfigMgr_CallbackType_t;
```

4.8.3 Asynchronous Response Callback Data Type

```
/*
* This union is a handy way of representing the various object identifiers
* used by the APIs.
*/
typedef union {
    NPF_F_ATM_IfID_t          ifID;       /* Interface ID                */
    NPF_F_ATM_VirtLink_t      linkId;     /* VP or VC Link               */
    NPF_F_AAL2ChanId_t        aal2ChanId; /* AAL2 channel ID             */
    NPF_F_VcXcId_t            vcXcId;     /* VC cross connect ID         */
    NPF_F_VpXcId_t            vpXcId;     /* VP cross connect ID         */
    NPF_F_AAL2ChnlXcId_t      chnlXcId;   /* AAL2 Chnl cross connect ID  */
} NPF_F_ATM_ConfigMgr_Id_t;
/*
* An asynchronous response contains an configuration object ID,
* a error or success code, and in some cases a function-
* specific structure embedded in a union. One or more of
* these is passed to the callback function as an array
* within the NPF_F_ATM_ConfigManager_CallbackData_t structure (below)
*/

typedef struct {
    NPF_error_t                 error;      /* Error code for this resp    */
    NPF_F_ATM_ConfigMgr_Id_t    objId;      /* Object Identifier           */
    union {
        /* Function-specific structures: */
        NPF_uint32_t                            unused;

        /* Queried VC statistics.
         * Completion callback - NPF_F_ATM_CONFIGMGR_VC_STATS_GET */
        NPF_F_ATM_ConfigMgr_VcStats_t           vcStats;

        /* Queried virtual path statistics
         * Completion callback - NPF_F_ATM_CONFIGMGR_VP_STATS_GET */
```

```
        NPF_F_ATM_ConfigMgr_VpStats_t           vpStats;

        /* Queried AAL2 channel statistics
         *Completion callback - NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_GET */
        NPF_F_ATM_ConfigMgr_AAL2ChannelStats_t  aal2ChanStats;

        /* Result of get operational status; status of link
         * Completion callback - NPF_F_ATM_CONFIGMGR_VL_OPER_STATUS_GET */
        NPF_ObjStatus_t                         operStatus;

        /* Result of initiated loopback procedure
         * Completion callback - NPF_F_ATM_CONFIGMGR_OAM_CC_RSP */
        NPF_F_ATM_ConfigMgr_OAM_LB_Result_t     lbResult;

             /* ATM virtual channel configuration and status
         * Completion callback - NPF_F_ATM_CONFIGMGR_VC_QUERY */
             NPF_F_ATM_ConfigMgr_VcInfo_t            vcConfigInfo;

             /* ATM virtual path configuration and status
         * Completion callback - NPF_F_ATM_CONFIGMGR_VP_QUERY */
             NPF_F_ATM_ConfigMgr_VpInfo_t            vpConfigInfo;

        /* AAL2 channel configuration and status
         * Completion callback - NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_QUERY */
             NPF_F_ATM_ConfigMgr_AAL2_ChannelInfo_t  chnlConfigInfo;

        /* VC cross connect attributes
         * Completion callback - NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_QUERY */
             NPF_F_ATM_ConfigMgr_VcLinkXc_t          vcXcConfig;

        /* VP link cross connect attributes
         * Completion callback - NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_QUERY */
             NPF_F_ATM_ConfigMgr_VpLinkXc_t          vpXcConfig;

        /* AAL2 channel cross connect attributes
         * Completion callback -
         *       NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_QUERY */
             NPF_F_ATM_ConfigMgr_AAL2_ChnlXc_t       chnlXcConfig;

             /* Queried Interface statistics
         * Completion callback - NPF_F_ATM_CONFIGMGR_IF_STATS_GET */
        NPF_F_ATM_ConfigMgr_IfStats_t           ifStats;

        /* ATM interface attributes
         * Completion callback - NPF_F_ATM_CONFIGMGR_IF_QUERY */
             NPF_F_ATM_ConfigMgr_IfCfg_t             ifConfig;

        /* OAM performance monitoring stats for requested flow
         * Completion callback - NPF_F_ATM_CONFIGMGR_OAM_PM_STATS_GET */
        NPF_F_ATM_ConfigMgr_OAM_PM_Stats_t      pmStats;

    } u;
} NPF_F_ATM_ConfigMgr_AsyncResponse_t;
/*
* The callback function receives the following structure containing
* one or more asynchronous responses from a single function call.
* There are several possibilities:
* 1. The called function does a single request
* - n_resp = 1, and the resp array has just one element.
```

```
* - allOK = TRUE if the request completed without error
* and the only return value is the response code.
* - if allOK = FALSE, the "resp" structure has the error code.
* 2. the called function supports an array of requests
* a. All completed successfully, at the same time, and the
* only returned value is the response code:
* - allOK = TRUE, n_resp = 0.
* b. Some completed, but not all, or there are values besides
* the response code to return:
* - allOK = FALSE, n_resp = the number completed
* - the "resp" array will contain one element for
* each completed request, with the error code
* in the NPF_F_ATM_ConfigManager_AsyncResponse_t structure, along
* with any other information needed to identify
* which request element the response belongs to.
* - Callback function invocations are repeated in
* this fashion until all requests are complete.
* Responses are not repeated for request elements
* already indicated as complete in earlier callback function invocations.
*/
typedef struct {
    NPF_F_ATM_ConfigMgr_CallbackType_t  type;        /* Function called */
    NPF_boolean_t           allOK;      /* TRUE if all completed OK     */
    NPF_uint32_t            n_resp;     /* Number of responses in array */
    NPF_F_ATM_ConfigMgr_AsyncResponse_t *resp;    /* response structures*/
} NPF_F_ATM_ConfigMgr_CallbackData_t;
The following table summarizes the functions in this API and their type code.
```

Table 4.6: ATM Configuration Manager API Function to Type Code Mapping

Function Name	Type Code
NPF_F_ATM_ConfigMgr_VcSet	NPF_F_ATM_CONFIGMGR_VC_SET
NPF_F_ATM_ConfigMgr_VpSet	NPF_F_ATM_CONFIGMGR_VP_SET
NPF_F_ATM_ConfigMgr_VcBind	NPF_F_ATM_CONFIGMGR_VC_BIND
NPF_F_ATM_ConfigMgr_VcUnbind	NPF_F_ATM_CONFIGMGR_VC_UNBIND
NPF_F_ATM_ConfigMgr_VirtLinkDelete	NPF_F_ATM_CONFIGMGR_VL_DELETE
NPF_F_ATM_ConfigMgr_VirtLinkEnable	NPF_F_ATM_CONFIGMGR_VL_ENABLE
NPF_F_ATM_ConfigMgr_VirtLinkDisable	NPF_F_ATM_CONFIGMGR_VL_DISABLE
NPF_F_ATM_ConfigMgr_VirtLinkStatsEnable	NPF_F_ATM_CONFIGMGR_VL_STATS_ENABLE
NPF_F_ATM_ConfigMgr_VirtLinkStatsDisable	NPF_F_ATM_CONFIGMGR_VL_STATS_DISABLE
NPF_F_ATM_ConfigMgr_VcOperStatusGet	NPF_F_ATM_CONFIGMGR_VL_OPER_STATUS_GET
NPF_F_ATM_ConfigMgr_VcStatsGet	NPF_F_ATM_CONFIGMGR_VC_STATS_GET
NPF_F_ATM_ConfigMgr_VcQuery	NPF_F_ATM_CONFIGMGR_VC_QUERY
NPF_F_ATM_ConfigMgr_VcLinkXcSet	NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_SET
NPF_F_ATM_ConfigMgr_VcLinkXcDelete	NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_DELETE
NPF_F_ATM_ConfigMgr_VcLinkXcQuery	NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_QUERY
NPF_F_ATM_ConfigMgr_VpStatsGet	NPF_F_ATM_CONFIGMGR_VP_STATS_GET
NPF_F_ATM_ConfigMgr_VpQuery	NPF_F_ATM_CONFIGMGR_VP_QUERY
NPF_F_ATM_ConfigMgr_VpLinkXcSet	NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_SET
NPF_F_ATM_ConfigMgr_VpLinkXcDelete	NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_DELETE
NPF_F_ATM_ConfigMgr_VpLinkXcQuery	NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_QUERY

NPF_F_ATM_ConfigMgr_AAL2_ChannelSet	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_SET
NPF_F_ATM_ConfigMgr_AAL2_ChannelDelete	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_DELETE
NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsGet	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_GET
NPF_F_ATM_ConfigMgr_AAL2_ChannelQuery	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_QUERY
NPF_F_ATM_ConfigMgr_AAL2_ChnlXcSet	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_SET
NPF_F_ATM_ConfigMgr_AAL2_ChnlXcDelete	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_DELETE
NPF_F_ATM_ConfigMgr_AAL2_ChnlXcQuery	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_QUERY
NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsEnable	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_ENABLE
NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsDisable	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_DISABLE
NPF_F_ATM_ConfigMgr_AAL2_ChannelEnable	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_ENABLE
NPF_F_ATM_ConfigMgr_AAL2_ChannelDisable	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_DISABLE
NPF_F_ATM_ConfigMgr_AAL2_ChannelBind	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_BIND
NPF_F_ATM_ConfigMgr_AAL2_ChannelUnbind	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_UNBIND
NPF_F_ATM_ConfigMgr_AAL2_ChannelOperStatusGet	NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_OPER_STATUS_GET
NPF_F_ATM_ConfigMgr_OAM_CP_Set	NPF_F_ATM_CONFIGMGR_OAM_CP_SET
NPF_F_ATM_ConfigMgr_OAM_CC_Set	NPF_F_ATM_CONFIGMGR_OAM_CC_SET
NPF_F_ATM_ConfigMgr_OAM_CC_Rsp	NPF_F_ATM_CONFIGMGR_OAM_CC_RSP
NPF_F_ATM_ConfigMgr_OAM_PM_Set	NPF_F_ATM_CONFIGMGR_OAM_PM_SET
NPF_F_ATM_ConfigMgr_OAM_PM_Rsp	NPF_F_ATM_CONFIGMGR_OAM_PM_RSP
NPF_F_ATM_ConfigMgr_OAM_PM_StatsGet	NPF_F_ATM_CONFIGMGR_OAM_PM_STATS_GET
NPF_F_ATM_ConfigMgr_OAM_LB_Set	NPF_F_ATM_CONFIGMGR_OAM_LB_SET
NPF_F_ATM_ConfigMgr_OAM_Mon_Set	NPF_F_ATM_CONFIGMGR_OAM_MONITOR
NPF_F_ATM_ConfigMgr_OAM_Alarm_Set	NPF_F_ATM_CONFIGMGR_OAM_ALARM_SET
NPF_F_ATM_ConfigMgr_OAM_Alarm_Clear	NPF_F_ATM_CONFIGMGR_OAM_ALARM_CLEAR
NPF_F_ATM_ConfigMgr_IfSet	NPF_F_ATM_CONFIGMGR_IF_SET
NPF_F_ATM_ConfigMgr_IfDelete	NPF_F_ATM_CONFIGMGR_IF_DELETE
NPF_F_ATM_ConfigMgr_IfStatsGet	NPF_F_ATM_CONFIGMGR_IF_STATS_GET
NPF_F_ATM_ConfigMgr_IfStatsEnable	NPF_F_ATM_CONFIGMGR_IF_STATS_ENABLE
NPF_F_ATM_ConfigMgr_IfStatsDisable	NPF_F_ATM_CONFIGMGR_IF_STATS_DISABLE
NPF_F_ATM_ConfigMgr_IfQuery	NPF_F_ATM_CONFIGMGR_IF_QUERY
NPF_F_ATM_ConfigMgr_IfAISStateSet	NPF_F_ATM_CONFIGMGR_IF_AIS_STATE_SET
NPF_F_ATM_ConfigMgr_IfAISStateClear	NPF_F_ATM_CONFIGMGR_IF_AIS_STATE_CLEAR

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

Table 4.7: ATM Configuration Manager API Function to Return Type mapping

Function Name	Structure Returned
NPF_F_ATM_ConfigMgr_VcSet	None
NPF_F_ATM_ConfigMgr_VpSet	None
NPF_F_ATM_ConfigMgr_VcBind	None
NPF_F_ATM_ConfigMgr_VcUnbind	None
NPF_F_ATM_ConfigMgr_VirtLinkDelete	None
NPF_F_ATM_ConfigMgr_VirtLinkEnable	None
NPF_F_ATM_ConfigMgr_VirtLinkDisable	None
NPF_F_ATM_ConfigMgr_VirtLinkStatsEnable	None
NPF_F_ATM_ConfigMgr_VirtLinkStatsDisable	None

NPF_F_ATM_ConfigMgr_VcOperStatusGet	operStatus
NPF_F_ATM_ConfigMgr_VcStatsGet	vcStats
NPF_F_ATM_ConfigMgr_VcQuery	vcConfigInfo
NPF_F_ATM_ConfigMgr_VcLinkXcSet	None
NPF_F_ATM_ConfigMgr_VcLinkXcDelete	None
NPF_F_ATM_ConfigMgr_VcLinkXcQuery	vcXcConfig
NPF_F_ATM_ConfigMgr_VpStatsGet	vpStats
NPF_F_ATM_ConfigMgr_VpQuery	vpConfigInfo
NPF_F_ATM_ConfigMgr_VpLinkXcSet	None
NPF_F_ATM_ConfigMgr_VpLinkXcDelete	None
NPF_F_ATM_ConfigMgr_VpLinkXcQuery	vpXcConfig
NPF_F_ATM_ConfigMgr_AAL2_ChannelSet	None
NPF_F_ATM_ConfigMgr_AAL2_ChannelDelete	None
NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsGet	aal2ChanStats
NPF_F_ATM_ConfigMgr_AAL2_ChannelQuery	chnlConfigInfo
NPF_F_ATM_ConfigMgr_AAL2_ChnlXcSet	None
NPF_F_ATM_ConfigMgr_AAL2_ChnlXcDelete	None
NPF_F_ATM_ConfigMgr_AAL2_ChnlXcQuery	chnlXcConfig
NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsEnable	None
NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsDisable	None
NPF_F_ATM_ConfigMgr_AAL2_ChannelEnable	None
NPF_F_ATM_ConfigMgr_AAL2_ChannelDisable	None
NPF_F_ATM_ConfigMgr_AAL2_ChannelBind	None
NPF_F_ATM_ConfigMgr_AAL2_ChannelUnbind	None
NPF_F_ATM_ConfigMgr_AAL2_ChannelOperStatusGet	None
NPF_F_ATM_ConfigMgr_OAM_CP_Set	None
NPF_F_ATM_ConfigMgr_OAM_CC_Set	None
NPF_F_ATM_ConfigMgr_OAM_CC_Rsp	None
NPF_F_ATM_ConfigMgr_OAM_PM_Set	None
NPF_F_ATM_ConfigMgr_OAM_PM_Rsp	None
NPF_F_ATM_ConfigMgr_OAM_PM_StatsGet	pmStats
NPF_F_ATM_ConfigMgr_OAM_LB_Set	lbResult
NPF_F_ATM_ConfigMgr_OAM_Mon_Set	None
NPF_F_ATM_ConfigMgr_OAM_Alarm_Set	None
NPF_F_ATM_ConfigMgr_OAM_Alarm_Clear	None
NPF_F_ATM_ConfigMgr_IfSet	None
NPF_F_ATM_ConfigMgr_IfDelete	None
NPF_F_ATM_ConfigMgr_IfStatsGet	ifStats
NPF_F_ATM_ConfigMgr_IfStatsEnable	None
NPF_F_ATM_ConfigMgr_IfStatsDisable	None
NPF_F_ATM_ConfigMgr_IfQuery	ifConfig
NPF_F_ATM_ConfigMgr_IfAISStateSet	None
NPF_F_ATM_ConfigMgr_IfAISStateClear	None

4.8.4 Event Notification Data Types

This following sections detail the information related to ATM configuration manager LFB events. When an event notification routine is invoked, one of the parameters will be a structure of information related to one or more events.

4.8.4.1 Event Notification Types

The event type indicates the type of event data in the union of event structures returned in `NPF_F_ATM_ConfigMgr_Event_t`.

```
/*
*   ATM configuration manager Event Types
*/
typedef enum {
  /* ATM OAM Fault Management Event Types */
  NPF_F_ATM_AIS_RAISED = 1,     /* AIS defect detected. AIS received       */
```

```
  NPF_F_ATM_RDI_RAISED = 2,     /* RDI defect detected. RDI received       */
  NPF_F_ATM_AIS_CLEARED = 3,    /* AIS defect cleared.                     */
  NPF_F_ATM_RDI_CLEARED = 4,    /* RDI defect cleared                      */
  NPF_F_ATM_LOC_RAISED = 5,     /* LOC defect detected                     */
  NPF_F_ATM_LOC_CLEARED = 6,    /* LOC defect cleared                      */

  /* ATM OAM Performance Management Event Types */
  NPF_F_ATM_PM_STATISTICS = 7,  /* Perf. Monitoring Stats                  */

  /* CC/PM activate/deactivate indication */
  NPF_F_ATM_CC_ACT_DEACT_REQ_INDICATION = 8, /* CC activate/deactivare ind */
  NPF_F_ATM_PM_ACT_DEACT_REQ_INDICATION = 9, /* PM activate/deactivate ind */

  /* Passive Monitoring events */
  NPF_F_ATM_CC_RECEIVED  = 10,    /* CC cell received  ;Passive monitoring */
  NPF_F_ATM_FPM_RECEIVED = 11,    /* FPM cell received ;Passive monitoring */
  NPF_F_ATM_BR_RECEIVED  = 12,    /* BR cell received  ;passive monitoring */
  NPF_F_ATM_LB_RECEIVED  = 13,    /* LB cell received  ;passive monitoring */
  NPF_F_ATM_AIS_RECEIVED = 14,    /* AIS cell received ;passive monitoring */
  NPF_F_ATM_RDI_RECEIVED = 15,    /* RDI cell received ;passive monitoring */
  NPF_F_ATM_PM_ACT_DEACT_RX = 16, /* PM Act/Deact cell ;passive monitoring */
  NPF_F_ATM_CC_ACT_DEACT_RX = 17, /* CC Act/Deact cell ;passive monitoring */

  /* Traffic Management threshold crossing event */
  NPF_F_ATM_TM_HIT_WARN_THRESH = 18,/*Link Queue length hit warn threshold */
  NPF_F_ATM_TM_HIT_DROP_THRESH = 19,/*Link Queue length hit drop threshold */
  NPF_F_ATM_TM_HIT_DROP_THRESH = 20,/*AAL2 TM Queue length hit drop thresh */
} NPF_F_ATM_ConfigMgr_Event_t;
```

4.8.4.2 Event Mask bit definitions

```
/*
 *   Definitions for selectively enabling ATM Configuration Manager Events
 */
#define NPF_F_ATM_AIS_RAISED_ENABLE                  (1 << 0)
#define NPF_F_ATM_RDI_RAISED_ENABLE                  (1 << 1)
#define NPF_F_ATM_AIS_CLEARED_ENABLE                 (1 << 2)
#define NPF_F_ATM_RDI_CLEARED_ENABLE                 (1 << 3)
#define NPF_F_ATM_LOC_RAISED_ENABLE                  (1 << 4)
#define NPF_F_ATM_LOC_CLEARED_ENABLE                 (1 << 5)
#define NPF_F_ATM_PM_STATISTICS_ENABLE               (1 << 6)
#define NPF_F_ATM_CC_ACT_DEACT_REQ_INDICATION_ENABLE (1 << 7)
#define NPF_F_ATM_PM_ACT_DEACT_REQ_INDICATION_ENABLE (1 << 8)
#define NPF_F_ATM_CC_RECEIVED_ENABLE                 (1 << 9)
#define NPF_F_ATM_FPM_RECEIVED_ENABLE                (1 << 10)
#define NPF_F_ATM_BR_RECEIVED_ENABLE                 (1 << 11)
#define NPF_F_ATM_LB_RECEIVED_ENABLE                 (1 << 12)
#define NPF_F_ATM_AIS_RECEIVED_ENABLE                (1 << 13)
#define NPF_F_ATM_RDI_RECEIVED_ENABLE                (1 << 14)
#define NPF_F_ATM_PM_ACT_DEACT_RX_ENABLE             (1 << 15)
#define NPF_F_ATM_CC_ACT_DEACT_RX_ENABLE             (1 << 16)
#define NPF_F_ATM_TM_HIT_WARN_THRESH_ENABLE          (1 << 17)
#define NPF_F_ATM_TM_HIT_DROP_THRESH_ENABLE          (1 << 18)
#define NPF_F_AAL2_TM_HIT_DROP_THRESH_ENABLE         (1 << 19)
#define NPF_F_ATM_CMGR_EV_LAST                       (1 << 20)
```

The FAPI client may register for all events using `NPF_EV_ALL_EVENTS_ENABLE`.

4.8.4.3 Event Notification Structures

This section describes the various events which MAY be implemented. It is important to note that even if an implementation does not support any of these events, the implementation still needs to provide the register and deregister event function to enable interoperability.

This structure defines all the possible event definitions for the ATM configuration manager. An event type field indicates which member of the union is relevant in the specific structure.

```
/*
 *   ATM Configuration Manager Event reporting data type
 *   This structure represents a single event in an event array. The type
 *   field indicates the specific event in the union.
 */
typedef struct {
    NPF_F_ATM_ConfigMgr_Event_t     eventType;   /* Type of event reported */
    union {
        /* Fault mgmt events */
        NPF_F_ATM_ConfigMgr_OAM_FM_EventData_t       fm;
        /* Perf. Mgmt events */
        NPF_F_ATM_ConfigMgr_OAM_PM_EventData_t       pm;
        /* Act/deact events  */
        NPF_F_ATM_ConfigMgr_OAM_ActDeact_EventData_t actDeact;
        /* Passive Monitoring events */
        NPF_F_ATM_ConfigMgr_OAM_Mon_EventData_t      mon;
        /* Traffic Management events */
        NPF_F_ATM_ConfigMgr_TrafMgmt_EventData_t     traf;
    } u;
} NPF_F_ATM_ConfigMgr_EventData_t;
```

The below event is triggered due to ATM OAM fault management procedures.

```
/*
 *   ATM OAM Fault Management Event Notification
 */
typedef struct {
    NPF_F_ATM_VirtLink_t                linkId;    /* VP or VC Link        */
    /* Detailed information for the Alarm being raised/clear */
    NPF_F_ATM_OAM_Alarm_Info_t          oamAlarmInfo;
} NPF_F_ATM_ConfigMgr_OAM_FM_EventData_t;
```

The below event is triggered due to ATM OAM performance management procedures.

```
/*
*   ATM OAM Performance Management Event Notification
*/
typedef struct {
    NPF_F_ATM_VirtLink_t            linkId;       /* VP or VC Link     */
    NPF_F_ATM_OAM_PM_Stats_Event_t  oamPmStats;   /* Perf. Mon. stats  */
} NPF_F_ATM_ConfigMgr_OAM_PM_EventData_t;
```

The below event is triggered due to ATM OAM performance management or continuity check activation/deactivation request being received on the specified link.

```
/*
*   ATM OAM activation/deactivation Event Notification
*/
typedef struct {
    NPF_F_ATM_VirtLink_t                        linkId;   /* VP or VC Link   */
    union {
        /* PM activate/deactivation Request indication*/
```

```
        NPF_F_ATM_OAM_PM_ActDeact_Event_t oamPmActDeact;
        /* CC activate/deact Request indication       */
        NPF_F_ATM_OAM_CC_ActDeact_Event_t oamCcActDeact;
    }u;
} NPF_F_ATM_ConfigMgr_OAM_ActDeact_EventData_t;
```

The below event is triggered due to reception of a monitored OAM cell on specified link.

```
/*
*   ATM OAM Passive Monitoring Event Notification
*/
typedef struct {
    NPF_F_ATM_VirtLink_t                    linkId;   /* VP or VC Link     */
    union {
       NPF_F_ATM_OAM_AIS_Event_t         oamAisInfo; /* AIS cell info     */
       NPF_F_ATM_OAM_RDI_Event_t         oamRdiInfo; /* RDI cell info     */
       NPF_F_ATM_OAM_FPM_Event_t         oamFpmInfo; /* FPM cell info     */
       NPF_F_ATM_OAM_BR_Event_t          oamBrInfo;  /* BR cell info      */
       NPF_F_ATM_OAM_LB_Event_t          oamLBInfo;  /* LB cell info      */
       NPF_F_ATM_OAM_CC_Event_t          oamCCInfo;  /* CC cell info      */
       NPF_F_ATM_OAM_PM_ActDeact_Event_t pmADInfo;   /* PM Act/Deact cell */
       NPF_F_ATM_OAM_CC_ActDeact_Event_t ccADInfo;   /* CC Act/Deact cell */
    }u;
} NPF_F_ATM_ConfigMgr_OAM_Mon_EventData_t;
```

The below event is triggered due to the cells/packets queued at the traffic manager for the specified link crossing the early warning threshold or the drop threshold. The FAPI/LFB implementation may implement certain hysteresis around the warning or drop threshold for event notification to avoid a flurry of events being reported when the queue length is hovering around the threshold. If the link is an AAL2 path and prioritization of the AAL2 traffic is configured on the AAL2 path, threshold crossing notifications may be generated for packets queued at a configured priority queue crossing the discard threshold. The AAL2 path for such notifications is identified by the `virtLinkId` field in the event structure.

```
/*
 *    Traffic management threshold crossing notifications
 */
typedef struct {
    NPF_F_ATM_VirtLink_t    virtLinkId;  /* VP or VC Link                 */
    union {
       NPF_uint32_t           chnlPrio;  /* Priority of AAL2 path queue    *
                                          * that hit the discard threshold */
    } u;
} NPF_F_ATM_ConfigMgr_TrafMgmt_EventData_t;
```

The below table summarizes the various events which may be reported by the ATM configuration manager and the field in the union of the event data structure carrying data for the event.

Table 4.8: Event types and data

Event Type	Description	Union Member
NPF_F_ATM_AIS_RAISED	AIS defect detected. AIS received	fm
NPF_F_ATM_RDI_RAISED	RDI defect detected. RDI received	fm
NPF_F_ATM_AIS_CLEARED	AIS defect cleared	fm
NPF_F_ATM_RDI_CLEARED	RDI defect cleared	fm

NPF_F_ATM_LOC_RAISED	LOC defect detected	fm
NPF_F_ATM_LOC_CLEARED	LOC defect cleared	fm
NPF_F_ATM_CC_ACT_DEACT_REQ_INDICATION	Activation/Deactivation cell received for continuity check activation/deactivation. The registered event handler must issue function NPF_F_ATM_ConfigMgr_OAM_CC_Rsp to accept/reject the activation/deactivation request	actDeact.oamCcActDeact
NPF_F_ATM_PM_ACT_DEACT_REQ_INDICATION	Activation/Deactivation cell received for performance monitoring activation/ deactivation. The registered event handler must issue NPF_F_ATM_ConfigMgr_OAM_PM_Rsp function to accept/reject the activation/deactivation request	actDeact.oamPmActDeact
NPF_F_ATM_PM_STATS	To report performance monitoring statistics	pmStats
NPF_F_ATM_CC_RECEIVED	To notify CC cell reception	mon.oamCCInfo
NPF_F_ATM_FPM_RECEIVED	To notify FPM cell reception	mon.oamFpmInfo
NPF_F_ATM_BR_RECEIVED	To notify BR cell reception	mon.oamBrInfo
NPF_F_ATM_LB_RECEIVED	To notify LB cell reception	mon.oamLBInfo
NPF_F_ATM_AIS_RECEIVED	To notify AIS cell reception	mon.oamAisInfo
NPF_F_ATM_RDI_RECEIVED	To notify RDI cell reception	mon.oamRdiInfo
NPF_F_ATM_TM_HIT_WARN_THRESH	Link Queue length hit warn threshold	traf
NPF_F_ATM_TM_HIT_DROP_THRESH	Link Queue length hit drop threshold	traf

5 Functional API (FAPI)

5.1 Registration/De-Registration Functions

5.1.1 Completion Callback Function

```
typedef void (*NPF_F_ATM_ConfigMgr_CallBackFunc_t)(
                NPF_IN NPF_userContext_t                  userContext,
                NPF_IN NPF_correlator_t                   correlator,
                NPF_IN NPF_F_ATM_ConfigMgr_CallbackData_t  data);
```

5.1.1.1 Description

The callback function is implemented by the application, and is registered with the ATM connection manager through the NPF_F_ATM_ConfigMgr_Register () function. The callback data structure contains an array of responses, so that callbacks for multiple interfaces or ATM Vc's referenced in a single function call can be aggregated into fewer (perhaps just one) callback function invocations. The application can expect to receive exactly the same number of responses (callback array elements) as the multiplicity of the request, but the responses may be spread over multiple callback function invocations. How the function implementation allocates responses to callback invocations is up to the implementer. As an optimization: if the implementation is able to return success indications (NPF_NO_ERROR) for all responses from a single request in a single invocation of the callback function, and there is no information to return besides the success/failure code: instead of returning an array of responses, the implementation SHALL return a simple code indicating that all requested actions completed without error.

5.1.1.2 Input Parameters

- `userContext` – The context item that was supplied by the application when the completion callback function was registered.
- `correlator` – The correlator item that was supplied by the application when the function call was made. The correlator is used by the application mainly to distinguish between multiple invocations of the same function.
- `data` – Response information related to the function call. Contains information that is common among all functions, as well as information that is specific to a particular function.

5.1.1.3 Output Parameters

None

5.1.1.4 Return Value

None

5.1.2 Completion Callback Registration Function

```
NPF_error_t NPF_F_ATM_ConfigMgr_Register(
            NPF_IN  NPF_userContext_t                   userContext,
            NPF_IN  NPF_F_ATM_ConfigMgr_CallBackFunc_t callbackFunc,
            NPF_OUT NPF_callbackHandle_t                *callbackHandle);
```

5.1.2.1 Description

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

`NPF_F_ATM_ConfigMgr_Register()` is a synchronous function and has no completion callback associated with it.

5.1.2.2 Input Parameters

- `userContext` – A context item used for uniquely identifying the context of the application registering the completion callback function. The exact value will be provided back to the registered completion callback function as its 1st parameter when it is called. Application can assign any value to the `atmUserContext` and the value is completely opaque to the implementation.
- `callbackFunc` – The pointer to the completion callback function to be registered.

5.1.2.3 Output Parameters

- `callbackHandle` – A unique identifier assigned for the registered `userContext` and `callbackFunc` pair. This handle will be used by the application to specify which callback function to be called when invoking asynchronous the asynchronous functions. It will also be used when de-registering the `userContext` and `callbackFunc` pair.

5.1.2.4 Return Values

- `NPF_NO_ERROR` – The registration completed successfully.
- `NPF_E_BAD_CALLBACK_FUNCTION` - The callback function is NULL.
- `NPF_E_UNKNOWN` – An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative

- NPF_E_CALLBACK_ALREADY_REGISTERED - No new registration was made since the userContext and callbackFunc pair was already registered. Whether this should be treated as an error or not is dependent on the application.

5.1.3 Completion Callback Deregistration Function

```
NPF_error_t NPF_F_ATM_ConfigMgr_Deregister(
                  NPF_IN NPF_callbackHandle_t callbackHandle);
```

5.1.3.1 Description

This function is used by the application to de-register a pair of user context and callback function. If there are any outstanding calls related to the de-registered callback function, the callback function might be called for those outstanding calls even after de-registration.
NPF_F_ATM_ConfigMgr_Deregister() is a synchronous function and has no completion callback associated with it.

5.1.3.2 Input Parameters

- callbackHandle – The unique identifier representing the pair of user context and callback function to be de-registered.

5.1.3.3 Output Parameters

None

5.1.3.4 Return Values

- NPF_NO_ERROR - De-registration completed successfully.
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_BAD_CALLBACK_HANDLE - The function does not recognize the callback handle. There is no effect to the registered callback functions.

5.1.4 Event Handler Function

```
typedef void (*NPF_F_ATM_ConfigMgr_EventCallFunc_t) (
    NPF_IN  NPF_userContext_t                    atmUserContext,
    NPF_IN  NPF_uint32_t                         nEvent,
    NPF_IN  NPF_F_ATM_ConfigMgr_EventData_t      *atmEventArray);
```

5.1.4.1 Description

This handler function is for the FAPI client to register an event handling routine to the ATM configuration manager. One or more events can be notified to the application through a single invocation of this event handler function. Information on each event is represented in an array in the atmEventArray structure, where a client can traverse through the array and process each of the events. The registered event handler function is intended to be implemented by the FAPI client, and be registered to the ATM configuration manager implementation through NPF_F_ATM_ConfigMgr_EventHandler_Register() function. This function is invoked when the related event happens. The ATM configuration manager may invoke the registered event handler function any time after the NPF_F_ATM_ConfigMgr_EventHandler_Register() is invoked by the FAPI client.

5.1.4.2 Input Parameters

- atmUserContext – A context item used for uniquely identifying the context of the application registering the completion callback function. The exact value will be provided back to the registered completion callback function as its 1st parameter when it is called. The application can assign any value to the atmUserContext and the value is completely opaque to the implementation.
- nEvent – Number of events reported.

- atmEventArray – A structure containing an array of event information structures.

5.1.4.3 Output Parameters

None

5.1.4.4 Return Values

None

5.1.5 Event Registration Function

```
NPF_error_t NPF_F_ATM_ConfigMgr_EventHandler_Register(
            NPF_IN  NPF_userContext_t                    atmUserContext,
            NPF_IN  NPF_F_ATM_ConfigMgr_EventCallFunc_t atmEvtCallFn,
            NPF_IN  NPF_eventMask_t                      atmEvtMask,
            NPF_OUT NPF_callbackHandle_t                 *atmEvtCallHdl);
```

5.1.5.1 Description

A FAPI client to register its event handling routine for receiving notifications of LFB Events uses this function. The FAPI client may register multiple event handling routines using this function. The pair of atmUserContext and atmEvtCallFn identifies the event handling routine. For each individual pair, a unique atmEvtCallHdl will be assigned for future reference. Since the event handling routine is identified by both atmUserContext and atmEventCallFunc, duplicate registration of same event handling routine with different atmUserContext is allowed. Also, the same atmUserContext can be shared among different event handling routines. Duplicate registration of the same atmUserContext and atmEventCallFunc pair has no effect, and will output a handle that is already assigned to the pair, and will return an error that indicates that the callback has already been registered.

5.1.5.2 Input Parameters

- atmUserContext – A context item used for uniquely identifying the context of the application registering the completion callback function. The exact value will be provided back to the registered completion callback function as its 1st parameter when it is called. Application can assign any value to the atmUserContext and the value is completely opaque to the implementation.
- atmEvtCallFn – Contains the class of event for which handler is being registered and a pointer to the event handling routine to be registered.
- atmEvtMask – Indicates which events the FAPI client wishes to receive. An application can register a handler to receive selected events by setting a bit in the NPF_eventMask_t parameter for each event it wishes to receive, when it calls the event registration function. A mask value set to NPF_EV_ALL_EVENTS_ENABLE selects all events. If the FAPI client wishes to change the selection of events for a particular handler function, it may call the event registration function again with the same handler function address and context, but with a different event selection mask.

5.1.5.3 Output Parameters

- atmEvtCallHdl – A unique identifier assigned for the registered atmUserContext and atmEventCallFunc pair. The FAPI client to specify which event handling routine to be called when invoking asynchronous functions will use this handle. It will also be used when de-registering the atmUserContext and atmEventCallFunc pair.

5.1.5.4 Return Values

- NPF_NO_ERROR - The registration completed successfully.
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_BAD_CALLBACK_FUNCTION - atmEventCallFunc is NULL.

- NPF_E_CALLBACK_ALREADY_REGISTERED - No new registration was made since the atmUserContext and atmEventCallFunc pair was already registered.

5.1.6 Event Handler Deregistration Function

```
NPF_error_t NPF_F_ATM_ConfigMgr_EventHandler_Deregister(
            NPF_IN NPF_callbackHandle_t    atmEventCallHandle);
```

5.1.6.1 Description

This function is used by an application to de-register a pair of user context and event handler. If there are any outstanding calls related to the de-registered callback function, the callback function might be called for those outstanding calls even after de-registration. This is a synchronous function and has no associated completion callback.

5.1.6.2 Input Parameters

- atmEventCallHandle – The unique identifier representing the pair of user context and event Handler to be de-registered.

5.1.6.3 Output Parameters

None

5.1.6.4 Return Values

- NPF_NO_ERROR - De-registration was completed successfully.
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_BAD_CALLBACK_HANDLE - The function does not recognize the event callback handle. There is no effect to the registered event Handler.

5.2 ATM Interface Configuration Functions

5.2.1 Add or Modify an ATM Interface

```
NPF_error_t NPF_F_ATM_ConfigMgr_IfSet(
       NPF_IN NPF_callbackHandle_t       cbHandle,
       NPF_IN NPF_correlator_t           cbCorrelator,
       NPF_IN NPF_errorReporting_t       errorReporting,
       NPF_IN NPF_FE_Handle_t            feHandle,
       NPF_IN NPF_BlockId_t              blockId,
       NPF_IN NPF_uint32_t               numEntries,
       NPF_IN NPF_F_ATM_ConfigMgr_IfCfg_t   *cfgArray);
```

5.2.1.1 Description

This function adds/creates one or more ATM interfaces, or modifies the attributes of an existing ATM interface.

5.2.1.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of interfaces to set
- cfgArray - Pointer to an array of ATM interface attribute structures

5.2.1.3 Output Parameters

None

5.2.1.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN - The interface configurations did not complete successfully due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - The interfaces were not configured because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.2.1.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an interface ID in the objId member of the response structure and a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR.

The below error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_ATM_F_E_INVALID_ATTRIBUTE - Invalid parameters for interface configuration
- NPF_E_RESOURCE_EXISTS - Specified interface already exist.

5.2.2 Delete an ATM Interface

```
NPF_error_t NPF_F_ATM_ConfigMgr_IfDelete(
    NPF_IN NPF_callbackHandle_t     cbHandle,
    NPF_IN NPF_correlator_t         cbCorrelator,
    NPF_IN NPF_errorReporting_t     errorReporting,
    NPF_IN NPF_FE_Handle_t          feHandle,
    NPF_IN NPF_BlockId_t            blockId,
    NPF_IN NPF_boolean_t            delContainedObjs,
    NPF_IN NPF_uint32_t             numEntries,
    NPF_IN NPF_F_ATM_IfID_t         *delArray);
```

This function deletes one or more interfaces. All connections established on this interface should be deleted before the interface is deleted.

5.2.2.1 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- delContainedObjs – When set to NPF_TRUE indicates that all objects contained within the deleted interface should also be deleted. Examples of contained objects are VC links and VP links. The effect of this parameter propagates hierarchically to all contained objects. For example if an interface is deleted, all VP links on that interface are deleted along with any VC links contained in those VP links and if any of those VC links contained any AAL2 channels the AAL2 channels will also be deleted. If this parameter is set to NPF_FALSE, the function will return an error if there are objects contained within the interface being deleted.
- numEntries - Number of interfaces to delete

- delArray – Pointer to an array of interface handles of interfaces to delete

5.2.2.2 Output Parameters

None

5.2.2.3 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN - Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.2.2.4 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an interface ID in the objId member of the response structure and a success code or a possible error code for that interface. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_ATM_F_E_CONT_OBJS_EXIST - Specified interface could not be deleted as it has other objects like VP/VC links configured in it and the delContainedObjs parameter was set to NPF_FALSE.
- NPF_E_RESOURCE_NONEXISTENT - Specified interface does not exist

5.2.3 Read ATM Interface Statistics

```
NPF_error_t NPF_F_ATM_ConfigMgr_IfStatsGet(
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
     NPF_IN NPF_errorReporting_t          errorReporting,
     NPF_IN NPF_FE_Handle_t               feHandle,
     NPF_IN NPF_BlockId_t                 blockId,
     NPF_IN NPF_uint32_t                  numEntries,
     NPF_IN NPF_F_ATM_IfID_t              *ifArray);
```

5.2.3.1 Description of function

This function returns, via a callback, the current counter values for each of one or more indicated ATM interfaces. The statistics counters are collected from the set of LFB instances specified in each ifArray entry for each interface.

5.2.3.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of interfaces statistics to query
- ifArray – Pointer to an array of interfaces to query

5.2.3.3 Output Parameters

None

5.2.3.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.2.3.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an interface ID in the objId field of the response structure and a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned and the statistics accumulated on that interface are returned in the ifStats field of the union of the response structure.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_RESOURCE_NONEXISTENT - Specified interface does not exist

5.2.4 Enable statistics collection on an ATM Interface

```
NPF_error_t NPF_F_ATM_ConfigMgr_IfStatsEnable(
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
     NPF_IN NPF_errorReporting_t          errorReporting,
     NPF_IN NPF_FE_Handle_t               feHandle,
     NPF_IN NPF_BlockId_t                 blockId,
     NPF_IN NPF_uint32_t                  numEntries,
     NPF_IN NPF_F_ATM_IfID_t              *ifIDArr);
```

5.2.4.1 Description of function

This function enables statistics collection one more ATM interfaces identified by the ATM Interface ID. When statistics counting changes state to ‘enabled’, counters for the specified interface are reset (set to all 0s). Statistics counters are non negative integers which wrap around on reaching maximum value using twos complement arithmetic.

5.2.4.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of interfaces on which statistics is enabled.
- ifIDArr – Array of Interface ID of ATM interface on which statistics is enabled

5.2.4.3 Output Parameters

None

5.2.4.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.2.4.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an interface ID in the objId field of the response structure and a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_RESOURCE_NONEXISTENT - Specified interface does not exist

5.2.5 Disable statistics collection on an ATM Interface

```
NPF_error_t NPF_F_ATM_ConfigMgr_IfStatsDisable(
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
     NPF_IN NPF_errorReporting_t          errorReporting,
     NPF_IN NPF_FE_Handle_t               feHandle,
     NPF_IN NPF_BlockId_t                 blockId,
     NPF_IN NPF_uint32_t                  numEntries,
     NPF_IN NPF_F_ATM_IfID_t              *ifIDArr);
```

5.2.5.1 Description of function

This function disables statistics collection one more ATM interfaces identified by the ATM Interface ID.

5.2.5.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of interfaces on which statistics is disabled.
- ifIDArr – Array of Interface ID of ATM interface on which statistics is disabled

5.2.5.3 Output Parameters

None

5.2.5.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.2.5.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an interface ID in the objId field of the response structure and a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful

- NPF_E_RESOURCE_NONEXISTENT - Specified interface does not exist

5.2.6 Query ATM Interface

```
NPF_error_t NPF_F_ATM_ConfigMgr_IfQuery(
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
     NPF_IN NPF_errorReporting_t          errorReporting,
     NPF_IN NPF_FE_Handle_t               feHandle,
     NPF_IN NPF_BlockId_t                 blockId,
     NPF_IN NPF_uint32_t                  numEntries,
     NPF_IN NPF_F_ATM_IfID_t              *ifArray);
```

5.2.6.1 Description of function

This function returns, via a callback, the current configuration for each of one or more indicated ATM interfaces. If the numEntries parameter passed to the function is set to 0, the configurations for all configured interfaces are returned.

5.2.6.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of interfaces to query. When set to 0 configurations for all configured interfaces are returned.
- ifArray – Pointer to an array of interfaces to query

5.2.6.3 Output Parameters

None

5.2.6.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.2.6.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an interface ID in the objId field of the response structure and a success code or a possible error code. The configuration of the queried interface is returned in the union of the asynchronous response structure. If the function invocation was successful an error code NPF_NO_ERROR is returned and the current configuration of the interface is returned in the ifConfig field of the union in the response structure.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_RESOURCE_NONEXISTENT - Specified interface does not exist

5.2.7 Set AIS state for an ATM Interface

```
NPF_error_t NPF_F_ATM_ConfigMgr_IfAISStateSet(
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
```

```
    NPF_IN NPF_errorReporting_t          errorReporting,
    NPF_IN NPF_FE_Handle_t               feHandle,
    NPF_IN NPF_BlockId_t                 blockId,
    NPF_IN NPF_F_ATM_IfID_t              ifID,
    NPF_IN NPF_uint8_t                   defectType,
    NPF_IN NPF_uint8_t                   defectLocation[16]);
```

5.2.7.1 Description of function

This function is used to declare AIS state for an ATM interface. Declaring AIS state for an interface leads to AIS state being declared on all VP and VC links on that interface.

5.2.7.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `ifID` – Interface ID of ATM interface for which AIS state is declared
- `defectType` – Type of defect leading to AIS state being declared
- `defectLocation` – Defect location for the defect

5.2.7.3 Output Parameters

None

5.2.7.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.2.7.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains an interface ID in the `objId` field of the response structure and a success code or a possible error code. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_RESOURCE_NONEXISTENT` - Specified interface does not exist

5.2.8 Clear AIS state for an ATM Interface

```
NPF_error_t NPF_F_ATM_ConfigMgr_IfAISStateClear(
     NPF_IN NPF_callbackHandle_t         cbHandle,
     NPF_IN NPF_correlator_t             cbCorrelator,
     NPF_IN NPF_errorReporting_t         errorReporting,
     NPF_IN NPF_FE_Handle_t              feHandle,
     NPF_IN NPF_BlockId_t                blockId,
     NPF_IN NPF_F_ATM_IfID_t             ifID);
```

5.2.8.1 Description of function

This function is used to clear the AIS state declared for an interface. Clearing AIS state for an interface leads to AIS state being cleared for all VP and VC links on that interface.

5.2.8.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- ifID – Interface ID of ATM interface for which AIS state is cleared

5.2.8.3 Output Parameters

None

5.2.8.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.2.8.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an interface ID in the objId field of the response structure and a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_RESOURCE_NONEXISTENT - Specified interface does not exist

5.3 ATM Virtual Link Management Functions

5.3.1 Add or Modify an ATM Virtual Channel

```
NPF_error_t NPF_F_ATM_ConfigMgr_VcSet(
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FE_Handle_t            feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_Vc_t  *cfgArray);
```

5.3.1.1 Description

This function adds/creates one or more VCs, or modifies the existing ATM attributes of VCs in an FE.

5.3.1.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.

- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of VCs to set
- cfgArray - Pointer to an array of ATM VC attribute structures

5.3.1.3 Output Parameters

None

5.3.1.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN - The VC link configurations did not complete successfully due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - The VC links were not configured because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.1.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains a VC link ID in the objId field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following errors could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_ATM_F_E_INVALID_VC_ADDRESS: ATM VC address is invalid
- NPF_ATM_F_E_INVALID_ATM_AAL: ATM AAL type code is invalid
- NPF_ATM_F_E_INVALID_ATM_QOS: Invalid ATM QoS specification for a VC
- NPF_ATM_F_E_INVALID_ATTRIBUTE: Invalid attribute

AAL1 Error Codes:

- NPF_ATM_F_E_INVALID_SUBTYPE: Invalid subtype
- NPF_ATM_F_E_INVALID_CBR: Invalid CBR or beyond the QoS of the VC
- NPF_ATM_F_E_INVALID_CLK_REC_TYPE: Invalid clock recovery type
- NPF_ATM_F_E_INVALID_FEC: Invalid FEC
- NPF_ATM_F_E_INVALID_SDT: Invalid SDT
- NPF_ATM_F_E_INVALID_PAR_FILL_CELL: Invalid partially filled cells
- NPF_ATM_F_E_INVALID_CELL_LOSS_PER: Invalid cell loss integration period

AAL2 Error Codes:

- NPF_ATM_F_E_INVALID_CPS_ATTR: Invalid CPS attributes
- NPF_ATM_F_E_INVALID_SSCS_I3661_ATTR: Invalid I.366.1 SSCS attribute
- NPF_ATM_F_E_INVALID_SSCS_I3662_ATTR: Invalid I.366.2 SSCS attribute
- NPF_ATM_F_E_INVALID_ATM_TRUNK_ATTR: Invalid ATM trunking attribute
- NPF_ATM_F_E_AAL2_QOS_PRIO INVALID: Invalid priority for AAL2 QoS
- NPF_ATM_F_E_AAL2_QOS_WEIGHT_INVALID: Invalid weight for AAL2 QoS

AAL5 Error Codes:

- NPF_ATM_F_E_INVALID_MAX_SIZE_FWD: Invalid forward max CPCS SDU Size
- NPF_ATM_F_E_INVALID_MAX_SIZE_BWD: Invalid backward max CPCS SDU Size

- NPF_ATM_F_E_INVALID_AAL_MODE: Invalid AAL mode
- NPF_ATM_F_E_INVALID_SSCS_TYPE: Invalid SSCS type

5.3.2 Add or Modify an ATM Virtual Path

```
NPF_error_t NPF_F_ATM_ConfigMgr_VpSet(
    NPF_IN NPF_callbackHandle_t      cbHandle,
    NPF_IN NPF_correlator_t          cbCorrelator,
    NPF_IN NPF_errorReporting_t      errorReporting,
    NPF_IN NPF_FE_Handle_t           feHandle,
    NPF_IN NPF_BlockId_t             blockId,
    NPF_IN NPF_uint32_t              numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_Vp_t *cfgArray);
```

5.3.2.1 Description

This function adds/creates one or more VPs, or modifies the existing ATM attributes of VPs in an FE.

5.3.2.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of VPs to set
- cfgArray - Pointer to an array of ATM VP attribute structures

5.3.2.3 Output Parameters

None

5.3.2.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN - The VP link configurations did not complete successfully due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - The VP link were not configured because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.2.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains a VP link ID in the objId field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following errors could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_ATM_F_E_INVALID_ATM_QOS: Invalid ATM QoS specification for a VP
- NPF_ATM_F_E_INVALID_ATTRIBUTE: Invalid attribute

5.3.3 Bind a higher layer Interface to an ATM Virtual Connection

```
NPF_error_t NPF_F_ATM_ConfigMgr_VcBind(
```

```
    NPF_IN NPF_callbackHandle_t            cbHandle,
    NPF_IN NPF_correlator_t                cbCorrelator,
    NPF_IN NPF_errorReporting_t            errorReporting,
    NPF_IN NPF_FE_Handle_t                 feHandle,
    NPF_IN NPF_BlockId_t                   blockId,
    NPF_IN NPF_IfHandle_t                  ifChildHandle,
    NPF_IN NPF_uint32_t                    numEntries,
    NPF_IN NPF_F_ATM_VirtLinkID_t          *bindArray);
```

5.3.3.1 Description of function

This function associates a single higher-layer interface with the one or more Virtual Channel Connections (VCC). The VCC is terminated in this FE and the VC link specified in the function parameters is an endpoint of the connection. The VC links must be created before making the call. This function binds other applications over ATM like IP over ATM and so on.

5.3.3.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `ifChildHandle` - Handle of the higher layer interface to bind.
- `numEntries` - Number of VCs to bind
- `bindArray` – Pointer to an array of VC link IDs to bind to the specified interface.

5.3.3.3 Output Parameters

None

5.3.3.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.3.3.5 Asynchronous response

A total of `numEntries` asynchronous (`NPF_F_ATM_ConfigMgr_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a VC link ID in the `objId` field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.
The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_ATM_F_E_INVALID_CHILD_HANDLE`: Invalid child interface handle
- `NPF_ATM_F_E_INVALID_ATTRIBUTE`: Invalid attributes
- `NPF_E_RESOURCE_NONEXIST`: Specified connection doesn't exist

5.3.4 Unbind a higher layer Interface from an ATM Virtual Connection

```
NPF_error_t NPF_F_ATM_ConfigMgr_VcUnbind(
    NPF_IN NPF_callbackHandle_t            cbHandle,
    NPF_IN NPF_correlator_t                cbCorrelator,
```

```
    NPF_IN NPF_errorReporting_t            errorReporting,
    NPF_IN NPF_FE_Handle_t                 feHandle,
    NPF_IN NPF_BlockId_t                   blockId,
    NPF_IN NPF_uint32_t                    numEntries,
    NPF_IN NPF_F_ATM_VirtLinkID_t          *unbindArray);
```

5.3.4.1 Description of function

This function removes the association between the higher-layer interface and one or more Virtual Channel Connections (VCC). The VCC is terminated in this FE and the VC link specified in the function parameters is an endpoint of the connection.

5.3.4.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of VCs to unbind
- `unbindArray` – Pointer to an array of VC link IDs.

5.3.4.3 Output Parameters

None

5.3.4.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.3.4.5 Asynchronous response

A total of `numEntries` asynchronous (`NPF_F_ATM_ConfigMgr_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a VC link ID in the `objId` field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_ATM_F_E_VC_NOT_BOUND:` The specified VC is not bound to any interface.
- `NPF_ATM_F_E_INVALID_ATTRIBUTE:` Invalid attributes
- `NPF_E_RESOURCE_NONEXIST:` Specified connection doesn't exist

5.3.5 Delete an ATM Virtual Link

```
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkDelete(
    NPF_IN NPF_callbackHandle_t     cbHandle,
    NPF_IN NPF_correlator_t         cbCorrelator,
    NPF_IN NPF_errorReporting_t     errorReporting,
    NPF_IN NPF_FE_Handle_t          feHandle,
    NPF_IN NPF_BlockId_t            blockId,
    NPF_IN NPF_boolean_t            delXc,
```

```
NPF_IN NPF_boolean_t            delContainedObjs,
NPF_IN NPF_uint32_t             numEntries,
NPF_IN NPF_F_ATM_VirtLink_t     *delArray);
```

5.3.5.1 Description of function

This function deletes one or more ATM virtual links. The virtual links may be either VP or VC links as specified in the input parameters.

5.3.5.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `delXc` – When set to `NPF_TRUE` indicates that any cross connection(s) made to this link should also be deleted on deletion of this link. The cross connected link(s) will not be deleted unless explicitly listed in the `delArray`
- `delContainedObjs` – When set to `NPF_TRUE` indicates that all objects contained within the deleted virtual link should be deleted. Examples of contained objects are VC links when the virtual link being deleted is a VP link, AAL2 channels present in a VC link etc. The effect of this parameter propagates hierarchically to all contained objects. For example if a VP links is deleted then all contained VC links will be deleted and if any of those VC links contained any AAL2 channels the AAL2 channels will be deleted. If this parameter is set to `NPF_FALSE,` the function will return an error if there are objects contained within the virtual link being deleted.
- `numEntries` - Number of virtual links to delete
- `delArray` – Pointer to an array of virtual links to delete

5.3.5.3 Output Parameters

None

5.3.5.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.

5.3.5.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the ID and type of the virtual link in the `objId` field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_ATM_F_E_CONT_OBJS_EXIST` – The specified virtual link could not be deleted as there are objects configured within this virtual link and the parameter `delContainedObjs` was set to `NPF_FALSE`.

- NPF_E_RESOURCE_NONEXIST: Specified virtual link doesn't exist

5.3.6 Enable a Virtual Link

```
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkEnable(
    NPF_IN NPF_callbackHandle_t     cbHandle,
    NPF_IN NPF_correlator_t         cbCorrelator,
    NPF_IN NPF_errorReporting_t     errorReporting,
    NPF_IN NPF_FE_Handle_t          feHandle,
    NPF_IN NPF_BlockId_t            blockId,
    NPF_IN NPF_uint32_t             numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t     *enaArray);
```

5.3.6.1 Description of function

This function administratively enables one more virtual links (VP or VC) identified by the virtual link ID. If the interface is ready for operation, then the ATM virtual links can receive and transmit cells. If the virtual link being enabled is an AAL2 path, the AAL2 channels belonging to that AAL2 path are also enabled unless they are administratively disabled.

5.3.6.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of virtual links to enable
- enaArray – Pointer to an array of virtual links to enable

5.3.6.3 Output Parameters

None

5.3.6.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.6.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains the ID and type of the virtual link in the objId field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST: Specified connection doesn't exist

5.3.7 Disable a Virtual Link

```
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkDisable(
```

```
    NPF_IN NPF_callbackHandle_t          cbHandle,
    NPF_IN NPF_correlator_t              cbCorrelator,
    NPF_IN NPF_errorReporting_t          errorReporting,
    NPF_IN NPF_FE_Handle_t               feHandle,
    NPF_IN NPF_BlockId_t                 blockId,
    NPF_IN NPF_uint32_t                  numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t          *disArray);
```

5.3.7.1 Description of function

This function administratively disables one more virtual links specified in the input parameters. Disabling a virtual link stops the reception and transmission of ATM cells on that virtual link. If the disabled virtual link is an AAL2 path, the AAL2 channels belonging to that AAL2 path are also disabled.

5.3.7.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of virtual links to disable
- `disArray` – Pointer to an array of virtual links to disable

5.3.7.3 Output Parameters

None

5.3.7.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.3.7.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the type and ID of the virtual link ID in the `objId` field of the response structure along with a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST`: Specified virtual link doesn't exist

5.3.8 Get Operational Status of a Virtual Link

```
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkOperStatusGet(
    NPF_IN NPF_callbackHandle_t          cbHandle,
    NPF_IN NPF_correlator_t              cbCorrelator,
    NPF_IN NPF_errorReporting_t          errorReporting,
    NPF_IN NPF_FE_Handle_t               feHandle,
    NPF_IN NPF_BlockId_t                 blockId,
```

```
    NPF_IN NPF_uint32_t                    numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t            *linkArray);
```

5.3.8.1 Description of function

This function is used to query the operational status of one or more virtual links.

5.3.8.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of virtual links to query for operational status
- `linkArray` – Pointer to an array of virtual links to query

5.3.8.3 Output Parameters

None

5.3.8.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.3.8.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the ID and type of the virtual link in the `objId` field of the response structure along with a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned and the current operation status of the virtual link is returned in the `operStatus` field of the union in the response structure.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST:` Specified virtual link doesn't exist

5.3.9 Enable Statistics collection on a Virtual Link

```
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkStatsEnable(
    NPF_IN NPF_callbackHandle_t     cbHandle,
    NPF_IN NPF_correlator_t         cbCorrelator,
    NPF_IN NPF_errorReporting_t     errorReporting,
    NPF_IN NPF_FE_Handle_t          feHandle,
    NPF_IN NPF_BlockId_t            blockId,
    NPF_IN NPF_uint32_t             numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t     *enaArray);
```

5.3.9.1 Description of function

This function enables statistics collection one more virtual links (VP or VC) identified by the virtual link ID. When statistics counting changes state to 'enabled', counters for the specified link are reset

(set to all 0s). Statistics counters are non negative integers which wrap around on reaching maximum value using twos complement arithmetic.

5.3.9.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of virtual links on which to enable statistics collection
- `enaArray` – Pointer to an array of virtual links on which to enable statistics collection

5.3.9.3 Output Parameters

None

5.3.9.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.3.9.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the ID and type of the virtual link in the `objId` field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST`: Specified connection doesn't exist

5.3.10 Disable Statistics collection on a Virtual Link

```
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkStatsDisable(
    NPF_IN NPF_callbackHandle_t     cbHandle,
    NPF_IN NPF_correlator_t         cbCorrelator,
    NPF_IN NPF_errorReporting_t     errorReporting,
    NPF_IN NPF_FE_Handle_t          feHandle,
    NPF_IN NPF_BlockId_t            blockId,
    NPF_IN NPF_uint32_t             numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t     *disArray);
```

5.3.10.1 Description of function

This function disables statistics collection one more virtual links (VP or VC) identified by the virtual link ID.

5.3.10.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`

- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of virtual links on which to disable statistics collection
- disArray – Pointer to an array of virtual links on which to disable statistics collection

5.3.10.3 Output Parameters

None

5.3.10.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.10.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains the ID and type of the virtual link in the objId field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST: Specified connection doesn't exist

5.3.11 Read ATM Virtual Channel Statistics

```
NPF_error_t NPF_F_ATM_ConfigMgr_VcStatsGet(
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
     NPF_IN NPF_errorReporting_t          errorReporting,
     NPF_IN NPF_FE_Handle_t               feHandle,
     NPF_IN NPF_BlockId_t                 blockId,
     NPF_IN NPF_uint32_t                  numEntries,
     NPF_IN NPF_F_ATM_VirtLinkID_t        *linkArray);
```

5.3.11.1 Description of function

This function returns, via a callback, the current counter values for each of one or more indicated ATM VCs.

5.3.11.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.

- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of VCs to query for statistics
- linkArray – Pointer to an array of VCs to query

5.3.11.3 Output Parameters

None

5.3.11.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.11.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an ID and type of the VC in the objId field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned and the statistics are returned in the vcStats field of the union in the response structure.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST - Specified VC doesn't exist

5.3.12 Query ATM Virtual Channel Configuration

```
NPF_error_t NPF_F_ATM_ConfigMgr_VcQuery(
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
     NPF_IN NPF_errorReporting_t          errorReporting,
     NPF_IN NPF_FE_Handle_t               feHandle,
     NPF_IN NPF_BlockId_t                 blockId,
     NPF_IN NPF_uint32_t                  numEntries,
     NPF_IN NPF_F_ATM_VirtLinkID_t        *linkArray);
```

5.3.12.1 Description of function

This function queries the configurations of one or more specified VCs. If the numEntries parameter passed to the function is set to 0, the configurations for all configured VCs are returned.

5.3.12.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of VCs to query. When set to 0 the configurations for all configured VCs is returned
- linkArray – Pointer to an array of VCs to query

5.3.12.3 Output Parameters

None

5.3.12.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.12.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an ID and type of the VC in the objId field of the response structure and a success code or a possible error code for that connection. If the numEntries parameter passed to the function is set to 0, the configurations for all configured VCs are returned. The union of the response structure contains the current configurations for the VC. If the function invocation was successful an error code NPF_NO_ERROR is returned and the current configuration of the VC is returned in the vcConfigInfo field of union in the response structure.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST - Specified VC doesn't exist

5.3.13 Add VC link Cross connect

```
NPF_error_t NPF_F_ATM_ConfigMgr_VcLinkXcSet (
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
     NPF_IN NPF_errorReporting_t          errorReporting,
     NPF_IN NPF_FE_Handle_t               feHandle,
     NPF_IN NPF_BlockId_t                 blockId,
     NPF_IN NPF_uint32_t                  numEntries,
     NPF_IN NPF_F_ATM_ConfigMgr_VcLinkXc_t *vcLinkXc);
```

5.3.13.1 Description of function

This function allows a client to cross connect ATM VC links. Both the VCs must exist before invoking the function.

5.3.13.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of VC link cross connects to configure
- vcLinkXc – Pointer to an array of VC link cross connection configurations

5.3.13.3 Output Parameters

None

5.3.13.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.13.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains VC link cross connect ID in the objId field of the response structure and a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_RESOURCE_EXIST: Specified cross connect exists
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_F_E_INVALID_ATTRIBUTE – Invalid configuration parameters

5.3.14 Delete VC link Cross connect

```
NPF_error_t NPF_F_ATM_ConfigMgr_VcLinkXcDelete (
   NPF_IN NPF_callbackHandle_t                cbHandle,
   NPF_IN NPF_correlator_t                    cbCorrelator,
   NPF_IN NPF_errorReporting_t                errorReporting,
   NPF_IN NPF_FE_Handle_t                     feHandle,
   NPF_IN NPF_BlockId_t                       blockId,
   NPF_IN NPF_uint32_t                        numEntries,
   NPF_IN NPF_F_VcXcId_t                      *vcLinkXc);
```

5.3.14.1 Description

This function allows a client to disassociate one or more cross connects of ATM VC link connections.

5.3.14.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of cross connections to delete
- vcLinkXc – Pointer to an array of cross connections to delete

5.3.14.3 Output Parameters

None

5.3.14.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.14.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains VC cross connect ID in the objId field of the response structure and a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST: Specified cross connect doesn't exist

5.3.15 Query Virtual Channel Cross connect

```
NPF_error_t NPF_F_ATM_ConfigMgr_VcLinkXcQuery (
   NPF_IN NPF_callbackHandle_t                cbHandle,
   NPF_IN NPF_correlator_t                    cbCorrelator,
   NPF_IN NPF_errorReporting_t                errorReporting,
   NPF_IN NPF_FE_Handle_t                     feHandle,
   NPF_IN NPF_BlockId_t                       blockId,
   NPF_IN NPF_uint32_t                        numEntries,
   NPF_IN NPF_F_VcXcId_t                      *linkXc);
```

5.3.15.1 Description

This function allows a client to query the configuration of one or more cross connects of ATM VC links. If the numEntries parameter passed to the function is set to 0, the configuration for all configured cross connects is returned.

5.3.15.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of cross connections to query. When set to 0 the configurations for all configured cross connections is returned.
- linkXc – Pointer to an array of cross connections to query

5.3.15.3 Output Parameters

None

5.3.15.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.15.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains VC cross connect ID in the objId field of the response structure and a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned and the

vcXcConfig field in union of the response structure contains the current configurations of the VC link cross connect.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST - Specified cross connect doesn't exist

5.3.16 Read ATM Virtual Path Statistics

```
NPF_error_t NPF_F_ATM_ConfigMgr_VpStatsGet(
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
     NPF_IN NPF_errorReporting_t          errorReporting,
     NPF_IN NPF_FE_Handle_t               feHandle,
     NPF_IN NPF_BlockId_t                 blockId,
     NPF_IN NPF_uint32_t                  numEntries,
     NPF_IN NPF_F_ATM_VirtLinkID_t        *linkArray);
```

5.3.16.1 Description of function

This function returns, via a callback, the current counter values for each of one or more indicated ATM VPs.

5.3.16.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of VPs to query for statistics
- linkArray – Pointer to an array of VPs to query

5.3.16.3 Output Parameters

None

5.3.16.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.16.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains the ID and type of the VP link in the objId field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned and the statistics are returned in the vpStats field of the union in the response structure.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful

- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST - Specified VP link doesn't exist

5.3.17 Query ATM Virtual Path Link Configuration

```
NPF_error_t NPF_F_ATM_ConfigMgr_VpQuery(
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
     NPF_IN NPF_errorReporting_t          errorReporting,
     NPF_IN NPF_FE_Handle_t               feHandle,
     NPF_IN NPF_BlockId_t                 blockId,
     NPF_IN NPF_uint32_t                  numEntries,
     NPF_IN NPF_F_ATM_VirtLinkID_t        *linkArray);
```

5.3.17.1 Description of function

This function queries the configurations of one or more specified virtual paths. If the numEntries parameter passed to the function is set to 0, the configurations for all configured virtual paths are returned.

5.3.17.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of VPs to query. When set to 0 the configurations for all configured VPs is returned
- linkArray – Pointer to an array of VPs to query

5.3.17.3 Output Parameters

None

5.3.17.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.17.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an ID and type of the virtual path in the objId field of the response structure and a success code or a possible error code for that connection. If the numEntries parameter passed to the function is set to 0, the configurations for all configured virtual paths are returned. The union of the response structure contains the current configurations for the virtual path. If the function invocation was successful an error code NPF_NO_ERROR is returned and the current configuration of the VP link is returned in the vpConfigInfo field of union in the response structure.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful

- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST: Specified virtual path doesn't exist

5.3.18 Add Virtual Path Link Cross connect

```
NPF_error_t NPF_F_ATM_ConfigMgr_VpLinkXcSet (
     NPF_IN NPF_callbackHandle_t          cbHandle,
     NPF_IN NPF_correlator_t              cbCorrelator,
     NPF_IN NPF_errorReporting_t          errorReporting,
     NPF_IN NPF_FE_Handle_t               feHandle,
     NPF_IN NPF_BlockId_t                 blockId,
     NPF_IN NPF_uint32_t                  numEntries,
     NPF_IN NPF_F_ATM_ConfigMgr_VpLinkXc_t *vpLinkXc);
```

5.3.18.1 Description of function

This function allows a client to cross connect ATM VP links. Both the virtual paths must exist before invoking the function.

5.3.18.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of VP link cross connects to configure
- vpLinkXc – Pointer to an array of VP link cross connection configurations

5.3.18.3 Output Parameters

None

5.3.18.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.3.18.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains VP link cross connect ID in the objId field of the response structure and a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_RESOURCE_EXIST: Specified cross connect exists
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_F_E_INVALID_ATTRIBUTE: Invalid configuration parameters

5.3.19 Delete Virtual Path Link Cross connect

```
NPF_error_t NPF_F_ATM_ConfigMgr_VpLinkXcDelete (
```

```
   NPF_IN NPF_callbackHandle_t                cbHandle,
   NPF_IN NPF_correlator_t                    cbCorrelator,
   NPF_IN NPF_errorReporting_t                errorReporting,
   NPF_IN NPF_FE_Handle_t                     feHandle,
   NPF_IN NPF_BlockId_t                       blockId,
   NPF_IN NPF_uint32_t                        numEntries,
   NPF_IN NPF_F_VpXcId_t                      *vpLinkXc);
```

5.3.19.1 Description

This function allows a client to disassociate one or more cross connects of ATM VP link connections.

5.3.19.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of cross connections to delete
- `vpLinkXc` – Pointer to an array of cross connections to delete

5.3.19.3 Output Parameters

None

5.3.19.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.3.19.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains VP link cross connect ID in the `objId` field of the response structure and a success code or a possible error code. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST`: Specified cross connect doesn't exist

5.3.20 Query Virtual Path Cross connect

```
NPF_error_t NPF_F_ATM_ConfigMgr_VpLinkXcQuery (
   NPF_IN NPF_callbackHandle_t                cbHandle,
   NPF_IN NPF_correlator_t                    cbCorrelator,
   NPF_IN NPF_errorReporting_t                errorReporting,
   NPF_IN NPF_FE_Handle_t                     feHandle,
   NPF_IN NPF_BlockId_t                       blockId,
   NPF_IN NPF_uint32_t                        numEntries,
   NPF_IN NPF_F_VpXcId_t                      *linkXc);
```

5.3.20.1 Description

This function allows a client to query the configuration of one or more cross connects of ATM VP links. If the `numEntries` parameter passed to the function is set to 0, the configuration for all configured cross connects is returned.

5.3.20.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of cross connections to query. When set to 0 the configurations for all configured cross connections is returned.
- `linkXc` – Pointer to an array of cross connections to query

5.3.20.3 Output Parameters

None

5.3.20.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.3.20.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains VP link cross connect ID in the `objId` field of the response structure and a success code or a possible error code. If the function invocation was successful an error code `NPF_NO_ERROR` is returned and the `vpXcConfig` field in union of the response structure contains the current configurations of the VP link cross connect.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST`: Specified cross connect doesn't exist

5.4 AAL2 channel management functions

5.4.1 Add or Modify an AAL2 Channel

5.4.1.1 Syntax

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelSet(
    NPF_IN NPF_callbackHandle_t                 cbHandle,
    NPF_IN NPF_correlator_t                     cbCorrelator,
    NPF_IN NPF_errorReporting_t                 errorReporting,
    NPF_IN NPF_FE_Handle_t                      feHandle,
    NPF_IN NPF_BlockId_t                        blockId,
    NPF_IN NPF_uint32_t                         numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_AAL2_Channel_t   *chnlArray);
```

5.4.1.2 Description of function

This function is used to add or modify an AAL2 channel on the specified AAL2 path. The profile of the AAL2 channels is passed in the `chnlArray` field of the function parameters.

5.4.1.3 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of AAL2 channels to create
- `chnlArray` – Pointer to an array of AAL2 channels to create

5.4.1.4 Output Parameters

None

5.4.1.5 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.4.1.6 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains an AAL2 channel ID in the `objId` field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTE` - Invalid attribute
- `NPF_ATM_F_E_INVALID_CPS_ATTR`: Invalid CPS attribute
- `NPF_ATM_F_E_INVALID_SSCS_I3661_ATTR`: Invalid I.366.1 SSCS attribute
- `NPF_ATM_F_E_INVALID_SSCS_I3662_ATTR`: Invalid I.366.2 SSCS attribute
- `NPF_ATM_F_E_INVALID_ATM_TRUNK_ATTR`: Invalid ATM trunking attribute
- `NPF_E_RESOURCE_EXIST`: Specified AAL2 channel exists
- `NPF_E_UNKNOWN` – An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_ATM_F_E_AAL2_QOS_PRIO INVALID`: Invalid priority for AAL2 QoS

5.4.2 Bind a higher layer Interface to an AAL2 Channel

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelBind(
    NPF_IN NPF_callbackHandle_t            cbHandle,
    NPF_IN NPF_correlator_t                cbCorrelator,
    NPF_IN NPF_errorReporting_t            errorReporting,
    NPF_IN NPF_FE_Handle_t                 feHandle,
    NPF_IN NPF_BlockId_t                   blockId,
    NPF_IN NPF_IfHandle_t                  ifChildHandle,
    NPF_IN NPF_uint32_t                    numEntries,
```

```
    NPF_IN NPF_F_AAL2ChanId_t              *bindArray);
```

5.4.2.1 Description of function

This function associates a single higher-layer interface with the one or more AAL2 channels. The AAL2 channel is terminated in this FE. The AAL2 channel must be created before making the call.

5.4.2.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `ifChildHandle` - Handle of the higher layer interface to bind.
- `numEntries` - Number of AAL2 channels to bind
- `bindArray` – Pointer to an array of AAL2 channel IDs to bind to the specified interface.

5.4.2.3 Output Parameters

None

5.4.2.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.4.2.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains an AAL2 channel ID in the `objId` field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_ATM_F_E_INVALID_CHILD_HANDLE`: Invalid child interface handle
- `NPF_ATM_F_E_INVALID_ATTRIBUTE`: Invalid attributes
- `NPF_E_RESOURCE_NONEXIST`: Specified AAL2 channel doesn't exist

5.4.3 Unbind a higher layer Interface from an AAL2 channel

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelUnbind(
    NPF_IN NPF_callbackHandle_t            cbHandle,
    NPF_IN NPF_correlator_t                cbCorrelator,
    NPF_IN NPF_errorReporting_t            errorReporting,
    NPF_IN NPF_FE_Handle_t                 feHandle,
    NPF_IN NPF_BlockId_t                   blockId,
    NPF_IN NPF_uint32_t                    numEntries,
    NPF_IN NPF_F_ATM_VirtLinkID_t          *unbindArray);
```

5.4.3.1 Description of function

This function removes the association between the higher-layer interface and one or more AAL2 channels.

5.4.3.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of AAL2 channels to unbind
- `unbindArray` – Pointer to an array of AAL2 channel IDs.

5.4.3.3 Output Parameters

None

5.4.3.4 Return Values

- `NPF_NO_ERROR` – The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.4.3.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains an AAL2 channel ID in the `objId` field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_ATM_F_E_CHNL_NOT_BOUND`: The specified AAL2 channel is not bound to any interface.
- `NPF_ATM_F_E_INVALID_ATTRIBUTE`: Invalid attributes
- `NPF_E_RESOURCE_NONEXIST`: Specified AAL2 channel doesn't exist

5.4.4 Delete an AAL2 Channel

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelDelete(
    NPF_IN NPF_callbackHandle_t                    cbHandle,
    NPF_IN NPF_correlator_t                        cbCorrelator,
    NPF_IN NPF_errorReporting_t                    errorReporting,
    NPF_IN NPF_FE_Handle_t                         feHandle,
    NPF_IN NPF_BlockId_t                           blockId,
    NPF_IN NPF_boolean_t                           delXc,
    NPF_IN NPF_uint32_t                            numEntries,
    NPF_IN NPF_F_AAL2ChanId_t                      *chnlArray);
```

5.4.4.1 Description of function

This function deletes the specified channel from the AAL2 path.

5.4.4.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `delXc` – When set to `NPF_TRUE` indicates that any cross connections made to this channel should also be deleted on deletion of this channel. The cross connected channel(s) will not deleted unless explicitly listed in the `chnlArray`
- `numEntries` - Number of AAL2 channels to delete
- `chnlArray` – Pointer to an array of AAL2 channels to delete

5.4.4.3 Output Parameters

None

5.4.4.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.4.4.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the AAL2 channel ID in the `objId` field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST:` Specified AAL2 channel doesn't exist

5.4.5 Query AAL2 Channel Statistics

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsGet(
     NPF_IN NPF_callbackHandle_t                    cbHandle,
     NPF_IN NPF_correlator_t                        cbCorrelator,
     NPF_IN NPF_errorReporting_t                    errorReporting,
     NPF_IN NPF_FE_Handle_t                         feHandle,
     NPF_IN NPF_BlockId_t                           blockId,
     NPF_IN NPF_uint32_t                            numEntries,
     NPF_IN NPF_F_AAL2ChanId_t                      *chnlArray);
```

5.4.5.1 Description of function

This function queries the statistics counters accumulated for a specified AAL2 channel

5.4.5.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`

- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of AAL2 channels to query for statistics
- chnlArray – Pointer to an array of AAL2 channels to query for statistics

5.4.5.3 Output Parameters

None

5.4.5.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.4.5.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an AAL2 channel ID in the objId field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code NPF_NO_ERROR is returned and the statistics accumulated for the channel are returned in the aal2ChanStats of field of the union in the response structure.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST: Specified AAL2 channel doesn't exist

5.4.6 Query AAL2 Channel

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelQuery(
     NPF_IN NPF_callbackHandle_t                    cbHandle,
     NPF_IN NPF_correlator_t                        cbCorrelator,
     NPF_IN NPF_errorReporting_t                    errorReporting,
     NPF_IN NPF_FE_Handle_t                         feHandle,
     NPF_IN NPF_BlockId_t                           blockId,
     NPF_IN NPF_uint32_t                            numEntries,
     NPF_IN NPF_F_AAL2ChanId_t                      *chnlArray);
```

5.4.6.1 Description of function

This function queries the configurations of one or more specified AAL2 channels. If the numEntries parameter passed to the function is set to 0, the configurations for all configured AAL2 channels is returned.

5.4.6.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.

- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of AAL2 channels to query. When set to 0, the configurations for all configured AAL2 channels is returned.
- chnlArray – Pointer to an array of AAL2 channels to query

5.4.6.3 Output Parameters

None

5.4.6.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.4.6.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains an AAL2 channel ID in the objId field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code NPF_NO_ERROR is returned and the configuration of the AAL2 channel is returned in the chnlConfigInfo field of the union of the response structure.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST: Specified AAL2 channel doesn't exist

5.4.7 Enable an AAL2 channel

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelEnable(
    NPF_IN NPF_callbackHandle_t     cbHandle,
    NPF_IN NPF_correlator_t         cbCorrelator,
    NPF_IN NPF_errorReporting_t     errorReporting,
    NPF_IN NPF_FE_Handle_t          feHandle,
    NPF_IN NPF_BlockId_t            blockId,
    NPF_IN NPF_uint32_t             numEntries,
    NPF_IN NPF_F_AAL2ChanId_t       *enaArray);
```

5.4.7.1 Description of function

This function administratively enables one more AAL2 channels identified by the AAL2 channel ID. If the AAL2 path is ready for operation, the AAL2 channels can receive and transmit packets.

5.4.7.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of AAL2 channels to enable

- enaArray – Pointer to an array of AAL2 channels to enable

5.4.7.3 Output Parameters

None

5.4.7.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.4.7.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains the ID of the AAL2 channel in the objId field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST: Specified AAL2 channel doesn't exist

5.4.8 Disable an AAL2 channel

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelDisable(
   NPF_IN NPF_callbackHandle_t        cbHandle,
   NPF_IN NPF_correlator_t            cbCorrelator,
   NPF_IN NPF_errorReporting_t        errorReporting,
   NPF_IN NPF_FE_Handle_t             feHandle,
   NPF_IN NPF_BlockId_t               blockId,
   NPF_IN NPF_uint32_t                numEntries,
   NPF_IN NPF_F_AAL2ChanId_t          *disArray);
```

5.4.8.1 Description of function

This function administratively disables one more AAL2 channels specified in the input parameters. Disabling an AAL2 channel stops reception and transmission of AAL2 packets on that AAL2 channel.

5.4.8.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of AAL2 channels to disable
- disArray – Pointer to an array of AAL2 channels to disable

5.4.8.3 Output Parameters

None

5.4.8.4 Return Values

- NPF_NO_ERROR - The operation is in progress.

- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.4.8.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains the ID of the AAL2 channel ID in the objId field of the response structure along with a success code or a possible error code for that AAL2 channel. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST: Specified AAL2 channel doesn't exist

5.4.9 Get Operational Status of a AAL2 channel

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelOperStatusGet(
    NPF_IN NPF_callbackHandle_t          cbHandle,
    NPF_IN NPF_correlator_t              cbCorrelator,
    NPF_IN NPF_errorReporting_t          errorReporting,
    NPF_IN NPF_FE_Handle_t               feHandle,
    NPF_IN NPF_BlockId_t                 blockId,
    NPF_IN NPF_uint32_t                  numEntries,
    NPF_IN NPF_F_AAL2ChanId_t            *chnlArray);
```

5.4.9.1 Description of function

This function is used to query the operational status of one or more AAL2 channels.

5.4.9.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of AAL2 channels to query for operational status
- chnlArray – Pointer to an array of AAL2 channels to query

5.4.9.3 Output Parameters

None

5.4.9.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.4.9.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains the ID of the

AAL2 channel in the objId field of the response structure along with a success code or a possible error code for that channel. If the function invocation was successful an error code NPF_NO_ERROR is returned and the current operation status of the AAL2 channel is returned in the operStatus field of the union in the response structure.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST - Specified AAl2 channel doesn't exist

5.4.10 Enable Statistics collection on a AAL2 channel

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsEnable(
    NPF_IN NPF_callbackHandle_t          cbHandle,
    NPF_IN NPF_correlator_t              cbCorrelator,
    NPF_IN NPF_errorReporting_t          errorReporting,
    NPF_IN NPF_FE_Handle_t               feHandle,
    NPF_IN NPF_BlockId_t                 blockId,
    NPF_IN NPF_uint32_t                  numEntries,
    NPF_IN NPF_F_AAL2ChanId_t            *chnlArray);
```

5.4.10.1 Description of function

This function enables statistics collection one more AAL2 channels identified by the AAL2 channel ID. When statistics counting changes state to 'enabled', counters for the specified channel are reset (set to all 0s). Statistics counters are non negative integers which wrap around on reaching maximum value using twos complement arithmetic.

5.4.10.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of AAL2 channels to enable statistics
- chnlArray – Pointer to an array of AAL2 channels on which statistics is enabled

5.4.10.3 Output Parameters

None

5.4.10.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.4.10.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains the ID of the AAL2 channel in the objId field of the response structure along with a success code or a possible error code for that channel. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST: Specified AAL2 channel doesn't exist

5.4.11 Disable Statistics collection on a AAL2 channel

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsDisable(
    NPF_IN NPF_callbackHandle_t          cbHandle,
    NPF_IN NPF_correlator_t              cbCorrelator,
    NPF_IN NPF_errorReporting_t          errorReporting,
    NPF_IN NPF_FE_Handle_t               feHandle,
    NPF_IN NPF_BlockId_t                 blockId,
    NPF_IN NPF_uint32_t                  numEntries,
    NPF_IN NPF_F_AAL2ChanId_t            *chnlArray);
```

5.4.11.1 Description of function

This function disables statistics collection one more AAL2 channels identified by the AAL2 channel ID.

5.4.11.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_ATM_ConfigMgr_Register()
- cbCorrelator - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- errorReporting - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- feHandle - The FE Handle returned by NPF_F_ATM_topologyGetFEInfoList () call.
- blockId – The unique identification of the ATM Configuration Manager.
- numEntries - Number of AAL2 channels to disable statistics
- chnlArray – Pointer to an array of AAL2 channels on which statistics is disabled

5.4.11.3 Output Parameters

None

5.4.11.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN – Failure due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - Failed because the callback handle was invalid.
- NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP – The requested feature is not supported.

5.4.11.5 Asynchronous response

A total of numEntries asynchronous (NPF_F_ATM_ConfigMgr_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains the ID of the AAL2 channel in the objId field of the response structure along with a success code or a possible error code for that channel. If the function invocation was successful an error code NPF_NO_ERROR is returned.

The following error codes could be returned:

- NPF_NO_ERROR - Operation successful
- NPF_E_UNKNOWN - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- NPF_E_RESOURCE_NONEXIST: Specified AAL2 channel doesn't exist

5.4.12 Add AAL2 Channel Cross connect

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChnlXcSet (
    NPF_IN NPF_callbackHandle_t                cbHandle,
    NPF_IN NPF_correlator_t                    cbCorrelator,
    NPF_IN NPF_errorReporting_t                errorReporting,
    NPF_IN NPF_FE_Handle_t                     feHandle,
    NPF_IN NPF_BlockId_t                       blockId,
    NPF_IN NPF_uint32_t                        numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_AAL2_ChnlXc_t   *chnlXc);
```

5.4.12.1 Description of function

This function is used to cross connect two AAL2 channels.

5.4.12.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Connection Manager.
- `numEntries` - Number of cross connects to set
- `chnlXc` – Pointer to an array of AAL2 channel cross connections configurations

5.4.12.3 Output Parameters

None

5.4.12.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.4.12.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains an AAL2 channel cross connect ID in the `objId` field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_EXISTS`: Specified AAL2 channel cross connect exists

5.4.13 Delete AAL2 Channel Cross connect

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChnlXcDelete (
   NPF_IN NPF_callbackHandle_t                cbHandle,
   NPF_IN NPF_correlator_t                    cbCorrelator,
   NPF_IN NPF_errorReporting_t                errorReporting,
   NPF_IN NPF_FE_Handle_t                     feHandle,
   NPF_IN NPF_BlockId_t                       blockId,
```

```
   NPF_IN NPF_uint32_t                          numEntries,
   NPF_IN NPF_F_AAL2ChnlXcId_t                  *chnlXc);
```

5.4.13.1 Description

This function is used to delete AAL2 channel cross connections.

5.4.13.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of cross connections to delete
- `chnlXc` – Pointer to an array of cross connections to delete

5.4.13.3 Output Parameters

None

5.4.13.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.4.13.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains an AAL2 channel cross connect ID in the `objId` field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST`: Specified AAL2 channel cross connect does not exist

5.4.14 Query AAL2 Channel Cross connect

```
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChnlXcQuery (
   NPF_IN NPF_callbackHandle_t                  cbHandle,
   NPF_IN NPF_correlator_t                      cbCorrelator,
   NPF_IN NPF_errorReporting_t                  errorReporting,
   NPF_IN NPF_FE_Handle_t                       feHandle,
   NPF_IN NPF_BlockId_t                         blockId,
   NPF_IN NPF_uint32_t                          numEntries,
   NPF_IN NPF_F_AAL2ChnlXcId_t                  *chnlXc);
```

5.4.14.1 Description

This function is used to query the current configuration of one or more specified AAL2 channel cross connects. When `numEntries` in the input parameters is set to 0, the configurations of all configured AAL2 channel cross connects is returned.

5.4.14.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of cross connections to query. When set to 0, the configuration of all configured AAL2 channel cross connects is returned.
- `chnlXc` – Pointer to an array of AAL2 channel cross connections to query

5.4.14.3 Output Parameters

None

5.4.14.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.4.14.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains an AAL2 channel cross connect ID in the `objId` field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned and the configurations of the queried AAL2 channel cross connect is returned in the `chnlXcConfig` field of the union in the response structure.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST`: Specified AAL2 channel cross connect does not exist

5.5 ATM OAM services functions

5.5.1 Configure/Modify OAM attributes of a Connection Point

```
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_CP_Set(
     NPF_IN NPF_callbackHandle_t           cbHandle,
     NPF_IN NPF_correlator_t               cbCorrelator,
     NPF_IN NPF_errorReporting_t           errorReporting,
     NPF_IN NPF_FE_Handle_t                feHandle,
     NPF_IN NPF_BlockId_t                  blockId,
     NPF_IN NPF_uint32_t                   numEntries,
     NPF_IN NPF_F_ATM_ConfigMgr_OAM_CP_t   *oamCPArray);
```

5.5.1.1 Description

This function configures/modifies the connection points for the F4/F5 OAM services using the OAM profile parameters, from the contents of an array passed by the caller, as defined in `NPF_F_ATM_ConfigMgr_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. The connection points are considered as intermediate points for ETE and segment flows by default. A connection point configured as an end point for an OAM flow may be reconfigured to be an intermediate point and vice versa by invoking the `NPF_F_ATM_ConfigMgr_OAM_CP_Set` function.

5.5.1.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of OAM connection points to configure
- `oamCPArray` - Pointer to an array structures containing the OAM profile parameters for the specified connection point

5.5.1.3 Output Parameters

None

5.5.1.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.5.1.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains a type and ID of the ATM virtual link carrying the OAM flow in the `objId` field of the response structure along with a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTES`: Invalid/erroneous configuration parameters
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST`: Specified connection point does not exist

5.5.2 Activate, Deactivate, Start and Stop OAM Continuity Check

```
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_CC_Set(
    NPF_IN NPF_callbackHandle_t           cbHandle,
    NPF_IN NPF_correlator_t               cbCorrelator,
    NPF_IN NPF_errorReporting_t           errorReporting,
    NPF_IN NPF_FE_Handle_t                feHandle,
    NPF_IN NPF_BlockId_t                  blockId,
```

```
        NPF_IN NPF_uint32_t                    numEntries,
        NPF_IN NPF_F_ATM_ConfigMgr_OAM_CC_t    *oamCCArray);
```

5.5.2.1 Description of function

This function activates and de-activates the CC procedure on a certain number of active connections in each direction (forward/backward). CC can be established during connection establishment or at anytime after the connection has been established. The connection creation must precede this operation.

5.5.2.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of connection to be operated on
- `oamCCArray` - Pointer to an array structure containing continuity check parameters

5.5.2.3 Output Parameters

None

5.5.2.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.5.2.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the type and ID of the ATM virtual link carrying the OAM flow in the `objId` field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTES`: Invalid/erroneous parameters
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST`: Specified connection point does not exist

5.5.3 Response to OAM Continuity Check Indication

```
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_CC_Rsp(
     NPF_IN NPF_callbackHandle_t           cbHandle,
     NPF_IN NPF_correlator_t               cbCorrelator,
     NPF_IN NPF_errorReporting_t           errorReporting,
     NPF_IN NPF_FE_Handle_t                feHandle,
     NPF_IN NPF_BlockId_t                  blockId,
     NPF_IN NPF_uint32_t                   numEntries,
     NPF_IN NPF_F_ATM_ConfigMgr_OAM_CC_Rsp_t       *oamCCArray);
```

5.5.3.1 Description of function

This function sends the response of CC activate/de-active request Indication. On receiving an activate/de-activate request from the remote end, the ATM configuration manager will generate an indication to the registered event handlers. In response to the activate/de-activate request received, the registered handlers should invoke the `NPF_F_ATM_ConfigMgr_OAM_CC_Rsp` function to accept or reject the activation/de-activation request.

5.5.3.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of responses
- `oamCCArray` - Pointer to an array structure containing response to activation/deactivation request.

5.5.3.3 Output Parameters

None

5.5.3.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.5.3.5 Asynchronous response

A total of `numEntries` asynchronous (`NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the type and ID of the ATM virtual link carrying the OAM flow in the `objId` field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTES`: Invalid/erroneous parameters
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST`: Specified connection point does not exist

5.5.4 Activate, Deactivate, Start and Stop OAM Performance Management

```
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_PM_Set(
    NPF_IN NPF_callbackHandle_t          cbHandle,
    NPF_IN NPF_correlator_t              cbCorrelator,
    NPF_IN NPF_errorReporting_t          errorReporting,
    NPF_IN NPF_FE_Handle_t               feHandle,
    NPF_IN NPF_BlockId_t                 blockId,
    NPF_IN NPF_uint32_t                  numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_PM_t  *oamPMArray);
```

5.5.4.1 Description

This function activates and de-activates the PM (FPM and associated BR, FPM only) on specified connection 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.

5.5.4.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of virtual links to configure for PM procedure
- `oamPMArray` - Pointer to an array structures containing the attributes for PM

5.5.4.3 Output Parameters

None

5.5.4.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.5.4.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the type and ID of the ATM virtual link carrying the OAM flow in the `objId` field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTES` - Invalid/erroneous parameters
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST` - Specified connection point does not exist

5.5.5 Response to OAM Performance Management Indication

```
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_PM_Rsp(
     NPF_IN NPF_callbackHandle_t              cbHandle,
     NPF_IN NPF_correlator_t                  cbCorrelator,
     NPF_IN NPF_errorReporting_t              errorReporting,
     NPF_IN NPF_FE_Handle_t                   feHandle,
     NPF_IN NPF_BlockId_t                     blockId,
     NPF_IN NPF_uint32_t                      numEntries,
     NPF_IN NPF_F_ATM_ConfigMgr_OAM_PM_Rsp_t  *oamPMArray);
```

5.5.5.1 Description

This function sends the response of PM activate/de-active request Indication. On receiving an activate/de-activate request from the remote end, the ATM configuration manager will generate an indication to the registered event handlers. In response to the activate/de-activate request received, the registered handlers should invoke the `NPF_F_ATM_ConfigMgr_OAM_PM_Rsp` function to accept or reject the activation/de-activation request.

5.5.5.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of responses
- `oamPMArray` - Pointer to an array structure containing PM activate/de-activate response

5.5.5.3 Output Parameters

None

5.5.5.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.5.5.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the type and ID of the ATM virtual link carrying the OAM flow in the `objId` field of the response structure along with a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTES` - Invalid/erroneous parameters
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST` - Specified connection point does not exist

5.5.6 Query Performance Monitoring Statistics for an OAM flow

```
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_PM_StatsGet(
   NPF_IN NPF_callbackHandle_t      cbHandle,
   NPF_IN NPF_correlator_t          cbCorrelator,
   NPF_IN NPF_errorReporting_t      errorReporting,
   NPF_IN NPF_FE_Handle_t           feHandle,
   NPF_IN NPF_BlockId_t             blockId,
   NPF_IN NPF_uint32_t              numEntries,
   NPF_IN NPF_F_ATM_ConfigMgr_OAM_PM_StatsReq_t   *oamPMStatsReqArray);
```

5.5.6.1 Description

This function is used to query the accumulated performance monitoring statistics for an OAM flow. As part of the request, the FAPI client may also request the accumulated statistics to be zeroed.

5.5.6.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of OAM flows to query for performance monitoring statistics
- `oamPMStatsReqArray` - Pointer to an array structures containing the queried flows

5.5.6.3 Output Parameters

None

5.5.6.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.5.6.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the type and ID of the ATM virtual link carrying the OAM flow in the `objId` field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code `NPF_NO_ERROR` is returned and the requested statistics is returned in the `pmStats` field of the union in the asynchronous response structure.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTES` - Invalid/erroneous parameters
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST` - Specified connection point does not exist
- `NPF_ATM_F_E_PERF_MONITORING_OFF` - Performance monitoring not enabled for requested flow

5.5.7 Initiate an OAM Loopback Procedure

```
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_LB_Set(
     NPF_IN NPF_callbackHandle_t           cbHandle,
     NPF_IN NPF_correlator_t               cbCorrelator,
     NPF_IN NPF_errorReporting_t           errorReporting,
     NPF_IN NPF_FE_Handle_t                feHandle,
     NPF_IN NPF_BlockId_t                  blockId,
     NPF_IN NPF_uint32_t                   numEntries,
     NPF_IN NPF_F_ATM_ConfigMgr_OAM_LB_t   *oamLBArray);
```

5.5.7.1 Description of function

This function initiates the loopback procedure for specified flow

5.5.7.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of connections to send loopback requests
- `oamLBArray` - Pointer to an array structures containing attributes for the loopback procedure

5.5.7.3 Output Parameters

None

5.5.7.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.5.7.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations to notify the result of the loopback procedure. Each response contains the type and ID of the ATM virtual link carrying the OAM flow in the `objId` field of the response structure along with a success code or a possible error code for that procedure. The result of the loopback procedure is specified in the `lbResult` field of the union of the response structure. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTES`: Invalid/erroneous parameters
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST`: Specified connection point does not exist

5.5.8 Monitor OAM Cells

```
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_Mon_Set(
     NPF_IN NPF_callbackHandle_t            cbHandle,
     NPF_IN NPF_correlator_t                cbCorrelator,
     NPF_IN NPF_errorReporting_t            errorReporting,
     NPF_IN NPF_FE_Handle_t                 feHandle,
     NPF_IN NPF_BlockId_t                   blockId,
     NPF_IN NPF_uint32_t                    numEntries,
     NPF_IN NPF_F_ATM_ConfigMgr_OAM_Mon_t  *oamMonArray);
```

5.5.8.1 Description of function

The function activates/de-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.

5.5.8.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of flows to monitor
- `oamMonArray` - Pointer to an array structures containing the OAM cell types to be monitored

5.5.8.3 Output Parameters

None

5.5.8.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.5.8.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains a type and ID of the ATM virtual link carrying the OAM flow in the `objId` field of the response structure along with a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTES` - Invalid/erroneous parameters
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST` - Specified connection point does not exist

5.5.9 Set AIS Alarm State

```
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_Alarm_Set (
     NPF_IN NPF_callbackHandle_t              cbHandle,
```

```
    NPF_IN NPF_correlator_t                 cbCorrelator,
    NPF_IN NPF_errorReporting_t             errorReporting,
    NPF_IN NPF_FE_Handle_t                  feHandle,
    NPF_IN NPF_BlockId_t                    blockId,
    NPF_IN NPF_uint32_t                     numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_Alarm_t  *oamAlarmArray);
```

5.5.9.1 Description

This function sets the AIS alarm state for the specified link(s) at the CP.

5.5.9.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of alarms to set
- `oamAlarmArray` - Pointer to an array structures containing OAM alarms to raise at CP

5.5.9.3 Output Parameters

None

5.5.9.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.5.9.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the type and ID of the virtual link carrying the OAM flow in the `objId` field of the response structure and a success code or a possible error code for that connection. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTES` - Invalid/erroneous parameters
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST` - Specified connection point does not exist

5.5.10 Clear AIS Alarm State

```
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_Alarm_Clear(
     NPF_IN NPF_callbackHandle_t            cbHandle,
     NPF_IN NPF_correlator_t                cbCorrelator,
     NPF_IN NPF_errorReporting_t            errorReporting,
     NPF_IN NPF_FE_Handle_t                 feHandle,
     NPF_IN NPF_BlockId_t                   blockId,
     NPF_IN NPF_uint32_t                    numEntries,
     NPF_IN NPF_F_ATM_ConfigMgr_OAM_Alarm_t *oamAlarmArray);
```

5.5.10.1 Description

This function clears the AIS alarm state for specified link(s) at the CP.

5.5.10.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_ATM_ConfigMgr_Register()`
- `cbCorrelator` - A unique application invocation value that will be supplied to the asynchronous completion callback routine.
- `errorReporting` - An indication of whether the application desires to receive an asynchronous completion callback for this function call.
- `feHandle` - The FE Handle returned by `NPF_F_ATM_topologyGetFEInfoList ()` call.
- `blockId` – The unique identification of the ATM Configuration Manager.
- `numEntries` - Number of alarms to clear
- `oamAlarmArray` - Pointer to an array structures containing OAM alarms to clear at CP

5.5.10.3 Output Parameters

None

5.5.10.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` – Failure due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - Failed because the callback handle was invalid.
- `NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP` – The requested feature is not supported.

5.5.10.5 Asynchronous response

A total of `numEntries` asynchronous `(NPF_F_ATM_ConfigMgr_AsyncResponse_t)` responses are passed to the callback function, in one or more invocations. Each response contains the type and ID of the ATM virtual link carrying the OAM flow in the `objId` field of the response structure along with a success code or a possible error code. If the function invocation was successful an error code `NPF_NO_ERROR` is returned.

The following error codes could be returned:

- `NPF_NO_ERROR` - Operation successful
- `NPF_ATM_F_E_INVALID_ATTRIBUTES` - Invalid/erroneous parameters
- `NPF_E_UNKNOWN` - An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative
- `NPF_E_RESOURCE_NONEXIST` - Specified connection point does not exist

6 References

The following documents contain provisions, which through reference in this text constitute provisions of this specification. At the time of publication, the editions indicated were valid. All referenced documents are subject to revision, and parties to agreements based on this specification are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below.

[FORCESREQ] "Requirement for separation of IP control and forwarding", H.Khosravi, T.Anderson et al, November, 2003 (RFC 3654)

[FAPITOPO] "FAPI Topology Manager API", work in progress, Network Processing Forum SWAPI Functional API TG, 2004.

[SWAPICON] "Software API Conventions Revision 2", http://www.npforum.org/techinfo/APIConventions2_IA.pdf, Network Processing Forum SWAPI Foundations TG, September 2003

[ITU-ATM] "B-ISDN ATM Layer Specifications – I.361)"

[ITU-AAL1] "B-ISDN ATM Adaptation Layer Specification: Type 1 AAL – I.363.1"

[ITU-AAL2] "B-ISDN ATM Adaptation Layer Specification: Type 2 AAL – I.363.2"

[ITU-AAL2SAR] "B-ISDN ATM Segmentation and Reassembly Service specific Convergence sublayer for AAL Type 2 – I.366.1"

[ITU-NBAAL2] "AAL Type 2 service specific convergence sublayer for narrow-band services – I.366.2"

[ITU-AAL5] "B-ISDN ATM Adaptation Layer Specification: Type 5 AAL – I.363.5"

[ITU-OAM] "B-ISDN Operation and Maintenance Principles and Functions – I.610"

[ATMF-TM41] "Traffic Management Specification version 4.1 – af-tm-0121.000"

[ATMF-TMABR] "Traffic Management ABR Addendum – af-tm-0077.000"

[ATMF-TMUBR] "Addendum to Traffic Management v4.1 optional minimum desired cell rate indication for UBR – af-tm-0150.000"

[ATMF-TMDUBR] "Addendum to Traffic Management v4.1: Differentiated UBR – af-tm-0149.000"

[ATMF-CES] "Circuit Emulation Service 2.0 – af-vtoa-0078.000"

[ATMF-CES-SIS] "Circuit Emulation Service – Interoperability specification – af-saa-0032.000"

APPENDIX A HEADER FILE INFORMATION

```
/*
 * This header file defines typedefs, constants and structures
 * for the NP Forum ATM Configuration Manager Functional API
 */
#ifndef __NPF_F_ATM_CONFIGURATION_MANAGER_H__
#define __NPF_F_ATM_CONFIGURATION_MANAGER_H__

#ifdef __cplusplus
extern "C" {
#endif

/* ATM Configuration Manager LFB Type ID */
#define NPF_F_ATMCONFIGMGR_LFB_TYPE   40   /* ATM Config. Mgr. LFB type code
*/

/*
* Asynchronous error codes (returned in function callbacks)
*/
#define NPF_F_ATMCMGR_BASE_ERR       (NPF_F_ATMCONFIGMGR_LFB_TYPE * 100)
#define NPF_ATM_F_E_INVALID_VC_ADDRESS      (NPF_F_ATMCMGR_BASE_ERR + 0)
#define NPF_ATM_F_E_INVALID_ATM_AAL         (NPF_F_ATMCMGR_BASE_ERR + 1)
#define NPF_ATM_F_E_INVALID_CHILD_HANDLE    (NPF_F_ATMCMGR_BASE_ERR + 2)
#define NPF_ATM_F_E_VC_NOT_BOUND            (NPF_F_ATMCMGR_BASE_ERR + 3)
#define NPF_ATM_F_E_INVALID_ATM_QOS         (NPF_F_ATMCMGR_BASE_ERR + 4)
#define NPF_ATM_F_E_INVALID_ATTRIBUTE       (NPF_F_ATMCMGR_BASE_ERR + 5)
#define NPF_ATM_F_E_INVALID_SUBTYPE         (NPF_F_ATMCMGR_BASE_ERR + 6)
#define NPF_ATM_F_E_INVALID_CBR             (NPF_F_ATMCMGR_BASE_ERR + 7)
#define NPF_ATM_F_E_INVALID_CLK_REC_TYPE    (NPF_F_ATMCMGR_BASE_ERR + 8)
#define NPF_ATM_F_E_INVALID_FEC             (NPF_F_ATMCMGR_BASE_ERR + 9)
#define NPF_ATM_F_E_INVALID_SDT             (NPF_F_ATMCMGR_BASE_ERR + 10)
#define NPF_ATM_F_E_INVALID_PAR_FILL_CELL   (NPF_F_ATMCMGR_BASE_ERR + 11)
#define NPF_ATM_F_E_INVALID_CELL_LOSS_PER   (NPF_F_ATMCMGR_BASE_ERR + 12)
#define NPF_ATM_F_E_INVALID_CPS_ATTR        (NPF_F_ATMCMGR_BASE_ERR + 13)
#define NPF_ATM_F_E_INVALID_SSCS_I3661_ATTR (NPF_F_ATMCMGR_BASE_ERR + 14)
#define NPF_ATM_F_E_INVALID_SSCS_I3662_ATTR (NPF_F_ATMCMGR_BASE_ERR + 15)
#define NPF_ATM_F_E_INVALID_ATM_TRUNK_ATTR  (NPF_F_ATMCMGR_BASE_ERR + 16)
#define NPF_ATM_F_E_INVALID_MAX_SIZE_FWD    (NPF_F_ATMCMGR_BASE_ERR + 17)
#define NPF_ATM_F_E_INVALID_MAX_SIZE_BWD    (NPF_F_ATMCMGR_BASE_ERR + 18)
#define NPF_ATM_F_E_INVALID_AAL_MODE        (NPF_F_ATMCMGR_BASE_ERR + 19)
#define NPF_ATM_F_E_INVALID_SSCS_TYPE       (NPF_F_ATMCMGR_BASE_ERR + 20)
#define NPF_ATM_F_E_AAL2_QOS_PRIO_INVALID   (NPF_F_ATMCMGR_BASE_ERR + 21)
#define NPF_ATM_F_E_AAL2_QOS_WEIGHT_INVALID (NPF_F_ATMCMGR_BASE_ERR + 22)
#define NPF_ATM_F_E_PERF_MONITORING_OFF     (NPF_F_ATMCMGR_BASE_ERR + 23)
#define NPF_ATM_F_E_CONT_OBJS_EXIST         (NPF_F_ATMCMGR_BASE_ERR + 24)
#define NPF_ATM_F_E_CMGR_FEATURE_NOT_SUPP   (NPF_F_ATMCMGR_BASE_ERR + 25)

/* Timer Units */
typedef enum {
    NPF_F_ATM_TIME_UNIT_NS = 0,       /* Timers specified in nanoseconds  */
    NPF_F_ATM_TIME_UNIT_US = 1,       /* Timers specified in microseconds */
    NPF_F_ATM_TIME_UNIT_MS = 2,       /* Timers specified in milliseconds */
    NPF_F_ATM_TIME_UNIT_1S = 3        /* Timers specified in seconds      */
} NPF_F_ATM_TimerUnit_t;

/* Type of ATM link */
typedef enum {
    NPF_F_ATM_VP_LINK = 0,       /* Virtual Path Link    */
```

```
    NPF_F_ATM_VC_LINK = 1        /* Virtual channel link */
} NPF_F_ATM_VirtLinkType_t;

/* Scope of an ATM virtual link. Internal virtual links are not visible *
 * to the network and are used for transporting data within the system  */
typedef enum {
    NPF_F_ATM_VIRT_LINK_INTERNAL = 0,       /* Internal VP/VC link    */
    NPF_F_ATM_VIRT_LINK_EXTERNAL = 1        /* External VP/VC link    */
} NPF_F_ATM_VirtLinkScope_t;

/* Object Status */
typedef enum {
    /* The below status can be used to denote operational or admin status
     * of an object, depending on the context in which they are used
     */
    NPF_STATUS_UP = 1,         /* Status marked UP                        */
    NPF_STATUS_DOWN = 2,       /* Status marked DOWN                      */
    NPF_STATUS_TESTING = 3,    /* Object in some test mode                */

    /* The below status can be used to denote operational status of object */
    NPF_STATUS_UNKNOWN = 4,    /* Cannot be determined due to some reason  */
    NPF_STATUS_DORMANT = 5,    /* Ready but waiting for external action    */
    NPF_STATUS_NOT_PRESENT = 6,/* Object has missing components like h/w   */
    NPF_STATUS_LOWER_LYR_DOWN = 7 /* Lower layer down                     */
} NPF_ObjStatus_t;
/*
* OAM Connection Point Type
*/
typedef enum{
    NPF_ATM_OAM_ETE_ENDPOINT         = 0, /* ETE End point                */
    NPF_ATM_OAM_SEG_ENDPOINT         = 1, /* Segment end point            */
    NPF_ATM_OAM_ETE_AND_SEG_ENDPOINT = 2, /* ETE & segment endpoint       */
    NPF_ATM_OAM_ETE_INTERMEDIATE     = 3, /* Intermediate point - ETE flow */
    NPF_ATM_OAM_ETE_AND_SEG_INTERMEDIATE = 4 /* Intermediate point - ETE
                                              And segment flow           */
} NPF_F_ATM_OAM_CP_Type_t;
/*
* Link direction
*/
typedef enum{
    NPF_F_ATM_RECEIVE  = 0,        /* link is receive  only */
    NPF_F_ATM_TRANSMIT = 1,        /* link is transmit only */
    NPF_F_ATM_DUPLEX = 2           /* link is bidirectional */
} NPF_F_ATM_Direction_t;

/*
* ATM VPI types
*/
typedef NPF_uint16_t NPF_F_ATM_VPI_t; /* VPI is 8 or 12 bits */
/*
* ATM VCI types
*/
typedef NPF_uint16_t NPF_F_ATM_VCI_t; /* VCI is 16 bits      */
typedef struct {         /* ATM Vc Address (VPI/VCI) structure */
    NPF_F_ATM_VPI_t vpi; /* VPI number */
    NPF_F_ATM_VCI_t vci; /* VCI number */
} NPF_F_ATM_VcAddr_t;

/* ATM Virtual Link Identifier */
```

```
typedef NPF_uint32_t NPF_F_ATM_VirtLinkID_t;    /* Virtual Link ID      */

/* ATM virtual link generic identifer; for VP or VC link               */
typedef struct {
    NPF_F_ATM_VirtLinkType_t atm_linkType;    /* VP link or VC link   */
    NPF_F_ATM_VirtLinkID_t   atm_linkID;      /* Virtual Link ID      */
} NPF_F_ATM_VirtLink_t;

/* ATM Interface Identifier */
typedef NPF_uint32_t NPF_F_ATM_IfID_t;         /* Interface ID      */


/* AAL2 Channel Identifier */
typedef NPF_uint32_t NPF_F_AAL2ChanId_t;        /* AAL2 channel ID   */

/* Virtual Channel Cross Connect Identifier */
typedef NPF_uint32_t NPF_F_VcXcId_t;        /* ATM VC Cross Connect ID   */

/* Virtual Path Cross Connect Identifier */
typedef NPF_uint32_t NPF_F_VpXcId_t;        /* ATM VP Cross Connect ID   */

/* AAL2 Channel Cross Connect Identifier */
typedef NPF_uint32_t NPF_F_AAL2ChnlXcId_t;   /* AAL2 channel Xconnect ID */

/* Timer Configuration data type */
typedef struct {
    NPF_F_ATM_TimerUnit_t  timeUnit;  /* Unit in which timer specified    */
    NPF_uint32_t           timeValue; /* Timer value in units specified   */
} NPF_F_ATM_Timers_t;

/*************************************************************
 * Enumerations and types for ITU I.610 OAM attributes/operations
 *************************************************************/
/* Connection Point ID */
typedef struct {
    NPF_uchar8_t    cpId[16];     /* Connection point ID */
} NPF_F_ATM_OAM_CPID_t;

/*
* OAM CC Response Type
*/
typedef enum {
    NPF_F_ATM_OAM_CC_RSP_ACCEPT = 0, /* Accept requested procedure */
    NPF_F_ATM_OAM_CC_RSP_REJECT = 1, /* Reject requested procedure */
} NPF_F_ATM_OAM_CC_RspType_t;
/*
* OAM PM Response
*/
typedef enum {
    NPF_IF_ATM_OAM_PM_RSP_ACCEPT = 0, /* Accept requested procedure */
    NPF_IF_ATM_OAM_PM_RSP_REJECT = 1, /* Reject requested procedure */
} NPF_F_ATM_OAM_PM_RspType_t;
/*
* OAM Flow Level
*/
typedef enum{
    NPF_F_ATM_OAM_FLOW_LEVEL_F4 = 0,      /* F4 level flow */
    NPF_F_ATM_OAM_FLOW_LEVEL_F5 = 1,      /* F5 Level flow */
} NPF_F_ATM_OAM_FlowLevel_t;
```

```
/*
* OAM Flow Type
*/
typedef enum{
    NPF_F_SEGMENT = 0,              /* Flow type SEGMENT */
    NPF_F_END_TO_END = 1,           /* Flow type ETE     */
} NPF_F_ATM_OAM_FlowType_t;
/*
* OAM Oper Type
*/
typedef enum{
    NPF_F_ACTIVATE    = 0,  /* Activate procedure */
    NPF_F_DEACTIVATE  = 1,  /* Deactive procedure */
    NPF_F_START       = 2,  /* Start procedure    */
    NPF_F_STOP        = 3,  /* Stop Procedure     */
} NPF_F_ATM_OAM_OperType_t;
/*
* OAM Fault Management        (AIS/RDI/CC/LB)
* OAM Performance Management  (FPM/BR)
* OAM Activation/Deactivation (FPM_BR/CC/FPM)
*/
typedef enum{
    NPF_F_AIS_CELL = 0,         /* Alarm Indication Signal Cell            */
    NPF_F_RDI_CELL = 1,         /* Remote Defect Indicator Cell            */
    NPF_F_CC_CELL  = 2,         /* Continuity Check Cell                   */
    NPF_F_LB_CELL  = 3,         /* Loopback Cell                           */
    NPF_F_FPM_CELL = 4,         /* Forward Performance Monitoring Cell     */
    NPF_F_BR_CELL  = 5,         /* Backward Reporting Cell                 */
    NPF_F_ACT_DEACT_FPM_BR = 6, /* Activate/Deactive FPM and associated BR */
    NPF_F_ACT_DEACT_CC   = 7,   /* Activate/Deactive Continuity check      */
    NPF_F_ACT_DEACT_FPM  = 8,   /* Activate/Deactive FPM                   */
} NPF_F_ATM_OAM_CellType_t;
/*
* OAM PM Block Size
*/
typedef enum{
    NPF_F_SIZE_32768 = 0,   /* Block size 32768 cells */
    NPF_F_SIZE_16384 = 1,   /* Block size 16384 cells */
    NPF_F_SIZE_8192  = 2,   /* Block size  8192 cells */
    NPF_F_SIZE_4096  = 3,   /* Block size  4096 cells */
    NPF_F_SIZE_2048  = 4,   /* Block size  2048 cells */
    NPF_F_SIZE_1024  = 5,   /* Block size  1024 cells */
    NPF_F_SIZE_512   = 6,   /* Block size   512 cells */
    NPF_F_SIZE_256   = 7,   /* Block size   256 cells */
    NPF_F_SIZE_128   = 8    /* Block size   128 cells */
} NPF_F_ATM_OAM_BlkSize_t;

/* OAM PM Function Type */
/* The data type defines PM function types can be set */
typedef enum {
    NPF_F_PM_FUNC_TYPE_FPM_BR = 0,
    NPF_F_PM_FUNC_TYPE_FPM
} NPF_F_ATM_OAM_PM_FuncType_t;

/*
* OAM Direction
*/
typedef enum{
    NPF_F_FORWARD        = 0,  /* A-B direction                          */
```

```
    NPF_F_BACKWARD       = 1,  /* B-A direction                              */
    NPF_F_TWO_WAY        = 2,  /* Both A-B and B-A direction                 */
    NPF_F_NOT_APPLICABLE = 3   /*Direction not applicable to this function */
} NPF_F_ATM_OAM_Direction_t;
/*
* OAM Continuity Check Method
*/
typedef enum {
    /* Send CC when no user cells only if no user cells have been sent in
     * CC Duration on link for that flow                                     */
    NPF_F_ATM_OAM_CC_SEND_CC_WHEN_NO_USER_CELLS = 0,

    /* Send CC cells periodically Irrespective of user cells flowing on    *
     * link for that flow                                                    */
    NPF_F_ATM_OAM_CC_SEND_CC_ALWAYS = 1
} NPF_F_ATM_OAM_CC_Method_t;
/*************************************************************
 * Enumerations and types for Backplane addressing and mapping  *
 *************************************************************/
/* Identifies a backplane interface to a FE */
typedef NPF_uint32_t NPF_F_ATM_BpIfID_t;      /* Backplane Interface ID    */

/* Identifies a backplane switch link */
typedef NPF_uint32_t NPF_F_ATM_BpLinkID_t;   /* Backplane Link ID    */

/* Backplane Switch Address */
/* Structure is used to pass the headers to be used to tunnel data         *
 * received over AAL2 channels or VP/VC links to backplane                 */
typedef struct {                          /* Switch Address structure     */
    NPF_F_ATM_BpLinkID_t  bpLinkID;       /* Backplane switch Link ID     */
    NPF_F_ATM_Direction_t direction;      /* Direction of data flow       */
    NPF_F_ATM_BpIfID_t    rxBpIfId;       /* Receive Backplane I/F ID     */
    NPF_uint32_t          rxaddressLength; /* Length of the Switch Address */
    NPF_uint8_t           *rxAddress;     /* Array of octets containing   */
                                          /* the Switch Address.          */
    NPF_F_ATM_BpIfID_t    txBpIfId;       /* Transmit Backplane I/F ID    */
    NPF_uint32_t          txaddressLength; /* Length of the Switch Address */
    NPF_uint8_t           *txAddress;     /* Array of octets containing   */
                                          /* the Switch Address.          */
} NPF_F_SwitchAddress_t;

/*************************************************************
 * Enumerations and types for ATM Quality of Service           *
 * Policing and Queuing characteristics for VPs and VCs       *
 *************************************************************/
/*
* Service Category
* The service category of this virtual connection.
*/
typedef enum {
    NPF_F_ATM_OTHER_SRV_CAT = 0,       /* Unspecified service cat         */
    NPF_F_ATM_CBR = 1,                 /* Constant bit rate               */
    NPF_F_ATM_rtVBR = 2,               /* Variable bit rate - real time   */
    NPF_F_ATM_nrtVBR = 3,              /* Variable bit rate - non real time*/
    NPF_F_ATM_ABR = 4,                 /* Available bit rate              */
    NPF_F_ATM_UBR_WITHOUT_PCR = 5,     /* Unspecified bit rate-no peak rate*/
    NPF_F_ATM_UBR_WITH_PCR = 6,        /* Unspecified bit rate w/ peak rate*/
    NPF_F_ATM_UBR_WITH_MDCR = 7,       /* Unspecified bit rate w/ minimum  */
                                       /* Desired cell rate               */
```

```
    NPF_F_ATM_UBR_WITH_MDCR_PCR = 8,   /* Unspecified bit rate w/ minimum  */
                                       /* Desired cell rate and peak rate  */
    NPF_F_ATM_GFR = 9                  /* Guaranteed frame rate            */
} NPF_F_ATM_ServiceCategory_t;

typedef NPF_uint16_t NPF_F_ATM_DropPolicy_t;

#define NPF_F_ATM_CELL_CLP0_DROP       0x0001 /* Drop CLP=0 cells    */
#define NPF_F_ATM_CELL_CLP0_CLP1_DROP  0x0002 /* Drop CLP=1 cells    */
#define NPF_F_ATM_PACKET_EP_DROP       0x0004 /* Drop frames;        *
                                               * Early pkt discard   */
#define NPF_F_ATM_PACKET_PP_DROP       0x0008 /* Drop frames;        *
                                               * Partial pkt discard */

/*
* ATM QOS profile,
* used in the ATM virtual link attribute structure
*/
typedef struct {
    NPF_F_ATM_ServiceCategory_t rxSrvCat;  /* ATM Service Category         */
    NPF_uint32_t     pcr;          /* Ingress Peak Cell Rate               */
    NPF_uint32_t     scr;          /* Ingress Sustainable Cell Rate        */
    NPF_uint32_t     mbs;          /* Ingress Maximum Burst Size           */
    NPF_uint32_t     mcr;          /* Ingress Minimum Cell Rate            */
    NPF_uint32_t     maxFrmSize;   /* Ingress Maximum Frame Size           */
    NPF_uint32_t     cdvt;         /* Ingress Cell Delay Var. Tolerance    */
    NPF_uint16_t     ubrPrio;      /* Ingress Priority for UBR with prio.  */
    NPF_F_ATM_DropPolicy_t upcPolicy; /* Usage Parameter control policy    */
    NPF_boolean_t    tagging;      /* Tag CLP=0 cells as per UPC           */
} NPF_F_ATM_Policing_Char_t;

typedef struct {
    NPF_F_ATM_ServiceCategory_t srvCat;   /* ATM Service Category         */
    NPF_uint32_t     pcr;          /* Egress Peak Cell Rate                */
    NPF_uint32_t     scr;          /* Egress Sustainable Cell Rate         */
    NPF_uint32_t     mbs;          /* Egress Maximum Burst Size            */
    NPF_uint32_t     mcr;          /* Egress Minimum Guaranteed Cell Rate  */
    NPF_uint32_t     maxFrmSize;   /* Egress Maximum Frame Size            */
    NPF_uint32_t     cdvt;         /* Egress Cell Delay Var. Tolerance     */
    NPF_uint16_t     ubrPrio;      /* Egress Priority for UBR with prio.   */
    NPF_uint32_t     bfrThresh;    /* Buffer congestion threshold for VC   */
    NPF_uint32_t     warnThresh;   /* Overflow early warn threshold for VC */
    NPF_F_ATM_DropPolicy_t dropPolicy;  /* Drop policy based on buffer use */
} NPF_F_ATM_Queuing_Char_t;
typedef struct {
    NPF_F_ATM_Policing_Char_t policeParams; /* Policing parameters         */
    NPF_F_ATM_Queuing_Char_t  queueParams;  /* Shaping/queuing parameters  */
} NPF_F_ATM_QoS_t;



/*****************************/
/* ATM Adaptation Layer Types */
/*****************************/
/*************************************************************
 * Enumerations and types for ATM Adaptation layer type - 1     *
 *************************************************************/
/* ATM Adaptation Layer Type 1 (AAL1) Profile */
/*
```

```
*   AAL type 1 subtype used by the CBR service application (e.g. 64
*   KBPS voice band signal transport, circuit transport)
*/
typedef enum {
    NPF_F_ATM_NULL = 0,
    NPF_F_ATM_VOICEBAND = 1,
    NPF_F_ATM_CIRCUIT_EMULATION_SYNCHRONOUS = 2,
    NPF_F_ATM_CIRCUIT_EMULATION_ASYNCHRONOUS =3,
    NPF_F_ATM_HIGH_QUALITY_AUDIO = 4,
    NPF_F_ATM_VIDEO = 5
} NPF_F_ATM_AAL1Subtype_t;

/*
*   Rate of CBR service supported by the AAL
*/
typedef enum {
    NPF_F_ATM_64_KBPS     = 0,
    NPF_F_ATM_1544_KBPS   = 1,
    NPF_F_ATM_6312_KBPS   = 2,
    NPF_F_ATM_32064_KBPS  = 3,
    NPF_F_ATM_44736_KBPS  = 4,
    NPF_F_ATM_97728_KBPS  = 5,
    NPF_F_ATM_2048_KBPS   = 6,
    NPF_F_ATM_8448_KBPS   = 7,
    NPF_F_ATM_34368_KBPS  = 8,
    NPF_F_ATM_139264_KBPS = 9,
    NPF_F_ATM_N64_KBPS    = 10,
    NPF_F_ATM_N8_KBPS     = 11
} NPF_F_ATM_CBR_t;

/*  Clock recovery type :
*   Synchronous,
*   Asynchronous-SRTS(Synchronous Residual Time Stamp) or
*   Asynchronous-Adaptive Clock Recovery.
*/
typedef enum {
    NPF_F_ATM_SYNCHRONOUS = 0,          /* Synchronous clock recovery     */
    NPF_F_ATM_ASYNCHRONOUS_SRTS = 1,    /* Synchronous Residual Time Stamp*/
    NPF_F_ATM_SYNCHRONOUS_ADAPTIVE = 2 /* Adaptive clock recovery        */
} NPF_F_ATM_AAL1ClkRecType_t;

/*   FEC method:
*       no FEC,
*       FEC for Loss Sensitive Signal Transport or
*       FEC for Delay Sensitive
*/
typedef enum {
    NPF_F_ATM_NO_FEC = 0,
    NPF_F_ATM_LOSS_SENSITIVE_SIGNAL_FEC = 1,
    NPF_F_ATM_DELAY_SENSITIVE_SIGNAL_FEC = 2
} NPF_F_ATM_AAL1FEC_t;

/*   CAS Mode: Valid only for structured interfaces
*        No CAS bits carried or
*     Carry CAS Bits in E1 multiframe structure or
*       Carry CAS bits in DS1 SF multiframe structure or
*     Carry CAS bits in DS1 ESF multiframe structure
*/
typedef enum {
```

```
    NPF_F_ATM_BASIC_MODE = 0,
    NPF_F_ATM_CAS_MODE_E1 = 1,
    NPF_F_ATM_CAS_MODE_DS1SF = 2,
    NPF_F_ATM_CAS_MODE_DS1ESF = 3,
    NPF_F_ATM_CAS_MODE_J2 = 4
} NPF_F_ATM_AAL1CASMode_t;


/*
* AAL1 Profile
*/
typedef struct {
    NPF_F_ATM_AAL1Subtype_t    subtype;        /* AAL1 sub-type           */
    NPF_F_ATM_CBR_t            cbrRate;        /* rate of CBR service     */
    NPF_uint32_t               rateMultiplier; /* Rate multiplier         */
    NPF_F_ATM_AAL1ClkRecType_t clkRecoveryType;/* Clock Recovery Type     */
    NPF_F_ATM_AAL1FEC_t        fecType;        /* Error Correction Method */
    NPF_F_ATM_AAL1CASMode_t    casMode;        /* CAS Transport mode      */

    /* Structured data transfer configuration. When sdtSupport is set
       to TRUE, it indicates structured data transfer. fecType should
    NPF_F_ATM_NO_FEC to select SDT                    */
    NPF_boolean_t          sdtSupport;         /* Whether SDT configured  */
    NPF_uint16_t           sdtBlockSize;       /* SDT Block Size          */

    /* partFilledCells set to TRUE causes the cell to be partially filled *
       before transmission in order to avoid excessive latency           */
    NPF_boolean_t          partFilledCells;  /*Enable partial cell method?*/

    /* Amount of user info in bytes that can be carried in partially filled*/
    /* cells. Valid only when partial filled cell method is selected       */
    NPF_uint32_t           partFilledCellsUserInfoSize;

    /* The maximum cell arrival jitter in 10 usec increments that the
       reassembly process will tolerate in the cell stream. Jitter beyond
       this value may lead to errors.                                    */
    NPF_F_ATM_Timers_t     cesCDVRxtolrnc;

    /* Define maximum size in 10 us increments for the reassembly buffer.  */
    NPF_F_ATM_Timers_t     maxReasmBufSize;

    /* Time in milliseconds for the cell loss integration period. If cells
       are lost for this period of time, the Interworking VCC Termination
       Point entity will  generate a cell starvation alarm.              */
    NPF_F_ATM_Timers_t     cellLossIntegrPeriod;
} NPF_F_ATM_AAL1Profile_t;




/*************************************************************
 * Enumerations and types for ATM Adaptation layer type - 2     *
 *************************************************************/
/* ATM Adaptation Layer Type 2 (AAL2) Profile */
/* AAL2 SSCS service category */
typedef enum {
```

```
    NPF_F_ATM_AUDIO = 1,                /* Audio SAP enabled                */
    NPF_F_ATM_MULTIRATE = 2,            /* Multirate SAP enabled            */
    NPF_F_ATM_AUDIO_AND_MULTIRATE =3    /* Audio and Multirate SAP enabled */
} NPF_F_ATM_AAL2SscsServiceCategory_t;

/* PCM encoding */
typedef enum {
    NPF_F_ATM_ALAW  = 1,                 /* Companding as per A-Law         */
    NPF_F_ATM_ULAW = 2                   /* Companding as per u-Law         */
} NPF_F_ATM_AAL2SscsPcmEncoding_t;

typedef enum {
    NPF_F_ATM_ITUT  = 1,                /* Profile defined by ITU-T         */
    NPF_F_ATM_OTHER = 2                 /* Profile defined by other entities*/
} NPF_F_ATM_AAL2SscsProfileSource_t;

/*
 * The SSCS configued for this AAL2 connection.
 */
typedef enum {
    NPF_F_ATM_AAL2_SSCS_NONE = 0,       /* No SSCS                          */
    NPF_F_ATM_AAL2_SSCS_SAR  = 1,       /* SSSAR SSCS (I.366.1)             */
    NPF_F_ATM_AAL2_SSCS_TRUNK= 2        /* Trunking SSCS (I.366.2)          */
} NPF_F_ATM_AAL2SscsType_t;

/*
 * Maximum possible AAL2- CPS SDU length for an AAL2 channel
 */
typedef enum {
    NPF_F_ATM_AAL2_CPS_SDU_LEN_45 = 0,   /* Maximum SDU length is 45 */
    NPF_F_ATM_AAL2_CPS_SDU_LEN_64 = 1    /* Maximum SDU length is 64 */
} NPF_F_ATM_AAL2_CpsSduLen_t;

/*
 * AAL2 QoS Profile
 */
typedef struct {
    NPF_uint32_t           maxPrio; /* Number of priority levels           */
    NPF_uint32_t          *weight;  /* Array specifying Weight used to     *
                                    * share path bandwidth among AAL2     *
                                    * groups on this AAL2 path            */
    NPF_uint32_t         *discThrsh;/* Threshold specified in number of    *
                                    * packets pending in a priority queue *
                                    * waiting for transmit opportunity    *
                                    * If queue length exceeds this figure *
                                    * then new packets are received on    *
                                    * this queue are discarded            */
} NPF_F_ATM_AAL2QosProfile_t;

/*
* AAL2 Trunking SSCS Profile
*/
typedef struct{
   /* Common SSCS Parameters(SSCS type = Trunking)as specified in I.366.2 */
    NPF_F_ATM_AAL2SscsServiceCategory_t srvCategory;/*Service category     */
    NPF_boolean_t audioServiceTransport;       /* Audio Transport enabled ?*/
    NPF_F_ATM_AAL2SscsProfileSource_t profileSource; /* profile source     */
    NPF_uint32_t predefinedProfileIdentifier;    /* predefined profile id  */
    NPF_F_ATM_AAL2SscsPcmEncoding_t pcmEncoding; /* PCM encoding type      */
```

```
    NPF_boolean_t faxDemodTransport;          /* demod. fax data support ? */
    NPF_boolean_t casTransport;               /* CAS support ?             */
    NPF_boolean_t dtmfDigitPacketTransport;   /* DTMF dialed digit support?*/
    NPF_boolean_t mfR1DigitPacketTransport;   /* MF-R1 dial digit support ?*/
    NPF_boolean_t mfR2DigitPacketTransport;   /* MF-R2 dial digit support ?*/
    NPF_boolean_t circuitModeDataTransport;   /* Circuit mode data support?*/
    NPF_uint32_t  circuitModeDataNumChannels; /* Multiplier N in N*64kbit/s
                                                 circuit mode data?       */
    NPF_boolean_t loopbackEnabled;            /* I.366.2 loopback enabled  */
} NPF_F_ATM_AAL2TrunkSscsProfile_t;


/*
* AAL2 CPS Profile
*/
typedef struct{

    /* This parameter indicates the maximum size CPS-SDU, in octets, that
       is transported on any AAL type 2 channel of an ATM connection. This
       parameter can take on the values "45" or "64" and is set by the
       signaling or management procedures.(See Max_CPS-SDU_Length;I.363.2) */
    NPF_F_ATM_AAL2_CpsSduLen_t  cpsMaxSduLength;/*Maximum CPS-SDU size     */

    /* If the singleCpsPacketPerCpsPduNoOverlap option is selected         */
    /* then the TIMER CU is nor applicable and the AAL2 payload cannot be  */
    /* greater than 44 bytes.                                              */
    NPF_boolean_t   singleCpsPerPduNoOverlap; /* CPS interleave control    */

    /* The Combined use timer value configured for this connection. This is*
     * valid only when singleCpsPerPduNoOverlap is set as FALSE            */
    NPF_F_ATM_Timers_t    cpsTimer_CU;         /* Combined Use Timer_CU    */

} NPF_F_ATM_AAL2CpsProfile_t;

/*
* AAL2 Profile
*/
typedef struct{
    /* AAl2 QoS Profile     */
    NPF_F_ATM_AAL2QosProfile_t        aal2QosProfile;

    /* AAL2 CPS Profile */
    NPF_F_ATM_AAL2CpsProfile_t        aal2CpsProfile;

    /* AAL2 service specific convergence sub layer configured for this VC  */
    NPF_F_ATM_AAL2SscsType_t          sscsType;

    /* I.366.2 Trunking SSCS Profile; Used if SSCS Type is set to
    NPF_F_ATM_AAL2_SSCS_TRUNK                                               */
    NPF_F_ATM_AAL2TrunkSscsProfile_t  aal2TrkSscsProf;
} NPF_F_ATM_AAL2Profile_t;

/*************************************************************
 * Enumerations and types for ATM Adaptation layer type - 5     *
 *************************************************************/
/*
*  This attribute indicates whether the AAL for the supporting VCC
*  operating in message mode or streaming mode, assured or non assured
*/
```

```
typedef enum {
    NPF_F_ATM_MESSAGE = 0,
    NPF_F_ATM_STREAMING = 1,
} NPF_F_ATM_AAL5_Mode_t;

/*
* SSCS type
*/
typedef enum{
    NPF_F_ATM_NULL_SSCS        = 0,/* NULL SSCS                       */
    NPF_F_ATM_DATA_ASSURED     = 1,/* Data SSCS on SSCOP(non assured) */
    NPF_F_ATM_DATA_NON_ASSURED = 2,/* Data SSCS on SSCOP(assured)     */
    NPF_F_ATM_FRAME_RELAY      = 3,/* Frame relay SSCS                */
} NPF_F_ATM_AAL5_SscsType_t;

/*
* AAL5 Profile
*/
typedef struct{
    NPF_uint32_t            maxCpcsSduSizeForward; /* Max o/g CPCS_SDU sz  */
    NPF_uint32_t            maxCpcsSduSizeBackward;/* Max i/c CPCS_SDU sz  */
    NPF_F_ATM_AAL5_Mode_t   aalMode;               /* AAL Mode             */
    NPF_F_ATM_AAL5_SscsType_t   sscsType;          /* SSCS Type            */
    NPF_boolean_t           deliverCorruptSdu;     /*If delivery of corrupt
                                                   * SDU is enabled        */
    NPF_uint32_t            maxCorruptSduDeliverLen;  /* Maximum size of
                                                   * delivered corrupt SDU */
    /* Timer configured to guard re-assembly process                       */
    NPF_F_ATM_Timers_t      rasTimer;             /* Reassembly Timer      */

}NPF_F_ATM_AAL5Profile_t;

/*************************************************************
 * Enumerations and types for ATM Adaptation layer profiles     *
 *************************************************************/
/*
 * ATM Vc AAL type code, used in the ATM Vc
 * attribute structure
 */
typedef enum {
    NPF_F_ATM_AAL0       = 0, /* RAW cell transfer. No AAL     */
    NPF_F_ATM_AAL1       = 1, /* AAL 1                         */
    NPF_F_ATM_AAL2       = 2, /* AAL 2                         */
    NPF_F_ATM_AAL5       = 3, /* AAL 5                         */
    NPF_F_ATM_AAL_UNKNOWN= 4, /* AAL cannot be determined      */
} NPF_F_ATM_AAL_t;
/*
* AAL Profile
*/
typedef union {
    NPF_F_ATM_AAL1Profile_t   aal1Profile;  /* ATM Adaptation Layer Type 1 */
    NPF_F_ATM_AAL2Profile_t   aal2Profile;  /* ATM Adaptation Layer Type 2 */
    NPF_F_ATM_AAL5Profile_t   aal5Profile;  /* ATM Adaptation Layer Type 5 */
} NPF_F_ATM_AAL_Profile_t;

/*************************************************************
 * Enumerations and types for cross connection of VC links, VP  *
 * links and AAL2 channels to external links or backplane links *
 *************************************************************/
```

```
/* Cross Connection Type */
typedef enum {
    NPF_F_ATM_EXT_TO_EXT = 0,   /* Connect to an external VP/VC         *
                                 * link or AAL2 channel                 */
    NPF_F_ATM_EXT_TO_BACK = 1,  /* Connect to a backplane switch link   */
    NPF_F_ATM_BACK_TO_INT = 2,  /* Connect to an internal VP/VC         *
                                 * link or AAL2 channel                 */
    NPF_F_ATM_EXT_TO_INT = 3    /* Connect to a external VP/VC  link    *
                                 * or AAL2 channel or a                 *
                                 * corresponding internal VP/VC link    *
                                 * or AAL2 channel                      */
} NPF_F_ATM_XcType_t;

/*
* ATM VC Link Cross connect Information
 */
typedef struct {
    NPF_F_VcXcId_t           vcXcId;        /* ID of this cross connection */
    NPF_F_ATM_XcType_t       xcType;        /* Type of cross connection    */
    union {
       NPF_F_ATM_VirtLinkID_t  mapVcLink;  /* Mapped to a VC Link         */
       NPF_F_SwitchAddress_t   mapSwLink;  /* Mapped to backplane         */
    }u;
} NPF_F_ATM_ConfigMgr_VcLinkXcInfo_t;

/*
* ATM VC link cross connect
*/
typedef struct {
    NPF_F_ATM_VirtLinkID_t               link_A;  /* VC Link 'A'           */
    NPF_uint32_t                       numLink_B;  /* Number of 'B' links   *
                                                    * connected to link 'A' */
    NPF_F_ATM_ConfigMgr_VcLinkXcInfo_t  *link_B;  /* Mapped link 'B'       */
} NPF_F_ATM_ConfigMgr_VcLinkXc_t;

/*
 * ATM VP Link Cross connect Information
 */
typedef struct {
    NPF_F_VcXcId_t           vpXcId;        /* ID of this cross connection */
    NPF_F_ATM_XcType_t       xcType;        /* Type of cross connection    */
    union {
       NPF_F_ATM_VirtLinkID_t  mapVpLink;  /* Mapped to a VP Link         */
       NPF_F_SwitchAddress_t   mapSwLink;  /* Mapped to backplane         */
    }u;
} NPF_F_ATM_ConfigMgr_VpLinkXcInfo_t;

/*
* ATM VC link cross connect - Connect link 'A' to link 'B'
*/
typedef struct {
    NPF_F_ATM_VirtLinkID_t            link_A;  /* VP Link 'A'              */
    NPF_uint32_t                   numLink_B;  /* Number of 'B' links      *
                                                * connected to link 'A'    */
    NPF_F_ATM_ConfigMgr_VpLinkXcInfo_t *link_B;/* Array of Mapped link 'B' */
} NPF_F_ATM_ConfigMgr_VpLinkXc_t;

/*
 * ATM AAL2 channel Link Cross connect Information
```

```
 */
typedef struct {
    NPF_F_AAL2ChnlXcId_t aal2ChnlXcId;  /* ID of AAL2 chnl. Cross cnct   */
    NPF_F_ATM_XcType_t   xcType;        /* Type of cross connection      */
    union {
       NPF_F_AAL2ChanId_t     mapAal2ChanId; /* Mapped AAL2 channel ID  */
       NPF_F_SwitchAddress_t  mapSwLink;     /* Mapped to backplane     */
    }u;
} NPF_F_ATM_AAL2_ChnlXcInfo_t;

/*
* ATM AAL2 channel cross connect - Connect channel 'A' to channel 'B'
*/
typedef struct {
    /* AAL2 channel 'A' to connect to second AAL2 channel or backplane     */
    NPF_F_AAL2ChanId_t                    aal2ChanId_A;

    /* Number of 'B' links connected to channel 'A'                        */
    NPF_uint32_t                          numLink_B;

    /* Mapped link 'B' - Backplane link or another AAL2 channel on same FE */
    NPF_F_ATM_AAL2_ChnlXcInfo_t           *link_B;
} NPF_F_ATM_ConfigMgr_AAL2_ChnlXc_t;

/*************************************************************
 * Enumerations and types to define attributes of VC and VP links
 *************************************************************/
/*
* ATM VC attributes
*/
typedef struct {
    /* The ATM virtual link identifier that is used to uniquely identify   *
     * a virtual channel link. The FAPI client can assign any value to the *
     * ATM virtual link identifier and the value is completely opaque to   *
     * the implementation.                                                 */
    NPF_F_ATM_VirtLinkID_t  vcLinkId;   /* Unique ID of this VC link       */

    /* The below field is used to specify the VPI and VCI of this VC       */
    NPF_F_ATM_VcAddr_t      vc;         /* VPI/VCI of virtual channel      */

    /* The scope of VC i.e. whether it is an external VC or an internal VC *
     * Internal VCs are used to carry data within the system and have no   *
     * external network terminations.                                      */

    NPF_F_ATM_VirtLinkScope_t  vcScope; /* External or internal VC         */

    /* This field defines the ATM interface identifier that is used        *
     * to uniquely identify an ATM interface in the system. The FAPI       *
     * client SHOULD assign the interface index defined by the             *
     * Interface Management APIs to this field.                            */
    NPF_F_ATM_IfID_t          ifId;       /* ATM Interface ID
*/

    /* The ATM adaptation layers to support multiple protocols to fit      *
     * the needs of different service users enhance the services provided  *
     * by the ATM layer. This data type is used to identify the various    *
     * adaptation layers used in ATM networks. The value                   *
    NPF_F_ATM_AAL_UNKNOWN indicates that the AAL cannot be determined   *
     * on this connection.                                                 */
```

```
    NPF_F_ATM_AAL_t         aalType;    /* AAL type                          */

    /* AAL specific configurations. The members of below union selected    *
     * based on the configured aalType.                                    */
    NPF_F_ATM_AAL_Profile_t aalProfile; /* AAL Profile                       */

    /* The below field is used to specify the direction of data flow on    *
     * the configured VC link with respect to FE being configured          */
    NPF_F_ATM_Direction_t   direction;  /* receive/transmit/duplex           */

    /* NPF_F_ATM_QoS_t structure is used to configure the traffic      *
     * management parameters for this virtual channel. If the direction of *
     * of the VC is configured as received then only policing parameters   *
     * are valid for the VC. If the direction is configured as transmit    *
     * the queuing parameters are valid. When the direction is configured *
     * as duplex, both policing and queuing params need to be configured   */
    NPF_F_ATM_QoS_t         qos;        /* QoS profile                       */

    /* The below field provides the administratively configured status of  *
     * the VC. The actual status i.e operational status assumed by the     *
     * link may be different based on the actual physical status. The      *
     * actual operational status queried may be using function provided    *
     * to query the operational status                                     */
    NPF_ObjStatus_t         admStatus;  /* Status of VC - UP/DOWN/TESTING  */

    /* The below field provides the status of statistics collections on the*
     * NPF_TRUE, statistics collection is enabled on this                  *
     * VC. The statistics collection may also be enabled or disabled at a  *
     * future point in time by issueing the statistics enable or disable   *
     * function call. The current statistics collection state may be       *
     * queried using the function provided to query the VC information     */
    NPF_boolean_t         statsEnabled;       /* Statistic collecting state*/

    /* The below field is used to cross connect the VC link to another     *
     * VC link on the same FE or to a backplane switch link. When the      *
     * numLink_B field is set 0, it indicates there is no cross connection *
     * established for this link. For a unicast mapping, the VC link being *
     * provisioned may be connected to another VC link or backplane link   *
     * and the numLink_B is set to 1. For multicast mapping, the numLink_B *
     * is set to the number of cross connections to be made.               */
    NPF_uint32_t  numLink_B;      /* Number of links connected to the link *
                                     being provisioned                     */
    NPF_F_ATM_ConfigMgr_VcLinkXcInfo_t   *link_B; /* Mapped link 'B'       */
} NPF_F_ATM_ConfigMgr_Vc_t;
/*
* OAM Configuration and Status Information
*/
typedef struct {
    /* Continuity Check state and configurations                          */
    NPF_boolean_t               ccActive;    /* Whether CC procedure ON */
    /* Below two fields valid if ccActive set to TRUE                     */
    NPF_F_ATM_OAM_Direction_t    ccDir;      /* Away/towards/both       */
    NPF_F_ATM_OAM_CC_Method_t    ccMethod;   /* Options to send CC Cell
                                               0-Send when no user cell
                                               1-Send periodically     */

    /* Performance Monitoring state and configurations                    */
     NPF_boolean_t               pmActive;   /* Whether PM procedure ON */
    /* Below four fields valid if pmActive set to TRUE                    */
```

```
    NPF_F_ATM_OAM_PM_FuncType_t  pmFuncType; /* FPM-BR/FPM              */
    NPF_F_ATM_OAM_Direction_t    pmDir;      /* Direction of operation  */
    NPF_F_ATM_OAM_BlkSize_t      pmFwdSize;  /* A-B block size          */
    NPF_F_ATM_OAM_BlkSize_t      pmBwdSize;  /* B-A block size          */

    /* Alarms states                                                    */
    NPF_boolean_t                inAISState; /* Whether in AIS state    */
    NPF_boolean_t                inRDIState; /* Whether in RDI state    */

    /* Loopback procedure configuration                                 */
    NPF_boolean_t                 llidOption;/*Is LLID option enabled ?*/

    /* Non intrusive monitoring of OAM flows. When number of monitored  *
     * cell types indicated using numMonCellTypes is set to non zero,   *
     * the list of cell types monitored is provided by *cellType array  */
    NPF_uint32_t            numMonCellTypes;
    NPF_F_ATM_OAM_CellType_t  *monCellType; /* OAM cell types monitored */
} NPF_F_ATM_ConfigMgr_OamInfo_t;
/*
* ATM VC query response
*/
typedef struct {
    NPF_F_ATM_ConfigMgr_Vc_t     vcConfig;  /* VC configuration info.     */

     /* vcBound indicates if the VC is bound to a child interface. If set *
      * to TRUE, the interface handle of the child interface is returned  *
      * in the ifChildHandle field                                        */
    NPF_boolean_t                 vcBound; /* Whether bound to child I/F  */
    NPF_IfHandle_t          ifChildHandle; /* Bound interface             */

    NPF_ObjStatus_t              operStatus;/* Operational status of VC   */

     /* OAM configuration and status information. Depending on the conn.  *
      * point type either eteFlowInfo, segFlowInfo or both may be valid   */
    NPF_F_ATM_OAM_CP_Type_t       connPtType;  /* Connection Pt. type     */
    NPF_F_ATM_ConfigMgr_OamInfo_t eteFlowInfo; /* E-T-E Flow Information  */
    NPF_F_ATM_ConfigMgr_OamInfo_t segFlowInfo; /* SEG Flow Information    */
} NPF_F_ATM_ConfigMgr_VcInfo_t;
/*
* ATM VP link attributes
*/
typedef struct {
    /* The ATM virtual link identifier that is used to uniquely identify   *
     * a virtual path link. The FAPI client can assign any value to the    *
     * ATM virtual link identifier and the value is completely opaque to   *
     * the implementation.                                                 */
    NPF_F_ATM_VirtLinkID_t  vpLinkId;   /* ID assigned to this VP link     */

    /* The VPI of the VP link being configured                             */
    NPF_F_ATM_VPI_t         vpi;        /* VPI of virtual connection       */

    /* The scope of VP i.e. whether it is an external VP or an internal VP *
     * Internal VPs are used to carry data within the system and have no   *
     * external network terminations.                                      */

    NPF_F_ATM_VirtLinkScope_t  vpScope; /* External or internal VP         */

    /* This section defines the ATM interface identifier that is used      *
     * to uniquely identify an ATM interface in the system. The FAPI       *
```

```
     * client SHOULD assign the interface index defined by the             *
     * Interface Management APIs to this field.                            */
    NPF_F_ATM_IfID_t          ifId;       /* ATM Interface ID              */

    /* The below field is used to specify the direction of data flow on    *
     * the configured VP link with respect to FE being configured          */
    NPF_F_ATM_Direction_t   direction;  /* receive/transmit/duplex         */

    /* This field specifies if the specified VP is terminated in this FE.  *
     * If the VP is terminated then the ATM header is analysed further to  *
     * identify the VC link using the VCI in the ATM header                */
    NPF_boolean_t           terminated; /* Switched/Terminated VP          */

    /* NPF_F_ATM_QoS_t structure is used to configure the traffic          *
     * management parameters for this virtual path   .                     */
    NPF_F_ATM_QoS_t         qos;        /* QoS profile                     */

    /* The below field provides the administratively configured status of  *
     * the VP. The actual status assumed by the link may be different based*
     * on the actual physical status. The operation status may be queried*
     * using the function provided to query the operational status         */
    NPF_ObjStatus_t         admStatus;  /* Status of VP - UP/DOWN/TESTING  */

    /* The below field provides the status of statistics collections on the*
    NPF_TRUE, statistics collection is enabled on this  *
     * VP. The statistics collection may also be enabled or disabled at a  *
     * future point in time by issueing the statistics enable/disable      *
     * function call. The current statistics collection state may be       *
     * queried using the function provided to query the VP information     */
    NPF_boolean_t         statsEnabled;       /* Statistic collecting state*/

    /* The below field is used to cross connect the VP link to another     *
     * VP link on the same FE or to a backplane switch link. When the      *
     * numLink_B field is set 0, it indicates there is no cross connection *
     * established for this link. For a unicast mapping, the VP link being *
     * provisioned may be connected to another VP link or backplane link   *
     * and the numLink_B is set to 1. For multicast mapping, the numLink_B *
     * is set to the number of cross connections to be made.               */
    NPF_uint32_t  numLink_B;      /* Number of links connected to the link *
                                     being provisioned                     */
    NPF_F_ATM_ConfigMgr_VpLinkXcInfo_t   *link_B; /* Mapped link 'B'       */
} NPF_F_ATM_ConfigMgr_Vp_t;
/*
* ATM VP query response
*/
typedef struct {
    NPF_F_ATM_ConfigMgr_Vp_t     vpConfig;  /* VP configuration info.    */
    NPF_ObjStatus_t            operStatus;  /* Operational status of VP  */

     /* OAM configuration and status information. Depending on the conn. *
      * point type either eteFlowInfo, segFlowInfo or both may be valid  */
    NPF_F_ATM_OAM_CP_Type_t       connPtType;  /* Connection Pt. type    */
    NPF_F_ATM_ConfigMgr_OamInfo_t eteFlowInfo; /* E-T-E Flow Information */
    NPF_F_ATM_ConfigMgr_OamInfo_t segFlowInfo; /* SEG Flow Information   */
} NPF_F_ATM_ConfigMgr_VpInfo_t;

/*************************************************************
 * Enumerations and types for ATM virtual link statistics
 *************************************************************/
```

```
/*
 * ATM Traffic Management Statistics
 */
typedef struct {
    /* Count of received cells tagged/discarded due to policing actions  */
    NPF_uint64_t cellsTaggedRx;    /* Receive Cells changed CLP0 to CLP1 */
    NPF_uint64_t cellsClp01DiscRx; /* Receive Cells discarded (CLP0+1)   */
    NPF_uint64_t cellsClp0DiscRx;  /* Receive Cells discarded (CLP0)     */

    /* Count of cells tagged/discarded due to congestion in o/g queue    */
    NPF_uint64_t cellsTaggedTx;    /* Transmit Cells changed CLP0 to CLP1*/
    NPF_uint64_t cellsClp01DiscTx; /* Transmit Cells discarded (CLP0+1)  */
    NPF_uint64_t cellsClp0DiscTx;  /* Transmit Cells discarded (CLP0)    */

    /* Queuing statistics in the transmit direction                      */
    NPF_uint32_t curQueueLenTx;    /* Transmit connection queue lengths  */
    NPF_uint32_t maxQueueLenTx;    /* Maximum queue length seen so far   */
} NPF_F_ATM_TMStats_t;

/*
 * ATM Level Statistics
 */
typedef struct {
    /* Count of total number of cells received on this connection        */
    NPF_uint64_t cellsClp01Rx;     /* Receive Total Cells (CLP0 + CLP1)  */
    NPF_uint64_t cellsClp0Rx;      /* Receive High Priority Cells (CLP0) */

    /* Count of received cells dropped due to resource unavailability    *
     * like buffers to hold/reassemble the cells etc.                    */
    NPF_uint32_t cellsDiscResErrRx;

    /* Count of total number of cells transmitted on this connection     */
    NPF_uint64_t cellsClp01Tx;     /* Transmit Total Cells (CLP0 + CLP1) */
    NPF_uint64_t cellsClp0Tx;      /* Transmit High Priority Cells (CLP0)*/

    /* Statistics counters for policing and queuing (TM) functions       */
    NPF_F_ATM_TMStats_t atmTrafficManagementStats;
} NPF_F_ATM_Stats_t;


/* ATM Adaptation Layer type 1 statistics */

typedef struct {
    /* Number of AAL1 header errors detected, including those corrected.
       Header errors include correctable and uncorrectable CRC plus bad
       parity. */
    NPF_uint32_t  errSnp;           /* No. of AAL1 cells with SNP errors */

    /* Sequence Count total violations: i.e., the count of incoming AAL Type1
       SAR-PDUs where the sequence count in the PDU header causes a
       transition from the SYNC state to the OUT OF SEQUENCE state as defined
       by ITU-T Recommendation I.363.1. (optional)
       -   lost cell: i.e., the number of lost cells, as detected by the
       AAL1 sequence number processing, for example. This count records
       the number of cells detected as lost in the network prior to the
       destination interworking function AAL1 layer processing. (optional)
       -   misinserted cells: i.e., the number of sequence violation events
       which the AAL CS interprets as misinserted of cells as defined by
       ITU-T Recommendation I.363.1. (optional)*/
```

```
    NPF_uint32_t  errSeqNoRx;       /* No. of AAL1 cells with SN errors  */

    /* Number of times the reassembly buffer overflows. If the interworking
       function is implemented with multiple buffers, such as a cell level
       buffer and a bit level buffer, then either buffer overflow will cause
       this count to be incremented).  */
    NPF_uint32_t  errBfrOverflowRx; /* No. of times buffer overflow at CS*/

    /* Number of times the reassembly buffer underflows. In the case of a
       continuous underflow caused by a loss of ATM cell flow, a single
       buffer underflow should be counted. If the interworking function is
       implemented with multiple buffers, such as a cell level buffer and a
       bit level buffer, then either buffer underflow will cause this count
       to be incremented. */
    NPF_uint32_t  errBfrUndrflowRx; /* No. of times buffer undrflow at CS*/

    /* Number of events in which the AAL1 reassembler found that a structured
       data pointer is not where it is expected, and the pointer must be re
       acquired. This count is only meaningful for structured data transfer
       modes as unstructured modes do not use pointers. (mandatory for
       structured data transfer) */
    NPF_uint32_t  errSdtPtrReframesRx;

    /* Number of times the AAL reassembler detects a parity check failure at
       the point where a structured data pointer is expected. This count is
       only meaningful for structured data transfer modes as unstructured
       modes do not use pointers.)*/
    NPF_uint32_t  errSdtPtrParityRx;
} NPF_F_ATM_AAL1Stats_t;

/* ATM Adaptation Layer type 2 statistics */
/* CPS Packet and Byte counters */
typedef struct {
    NPF_uint64_t cpsPktRx;      /* No. of AAL2 CPS packets received   */
    NPF_uint64_t cpsBytesRx;    /* No. CPS packet bytes received      */

    NPF_uint64_t cpsPktTx;      /* No. of AAL2 CPS packets transmitted*/
    NPF_uint64_t cpsBytesTx;    /* No. CPS packet bytes transmitted   */

    /* The below counters specify the number CPS packets discarded in *
     * the transmit direction due to various errors and the           *
     * corresponding byte counts for the discarded packets            */
    NPF_uint32_t cpsPktDisc;    /* No. of AAL2 CPS packets discarded  */
    NPF_uint32_t cpsBytesDisc;  /* No. CPS packet bytes discarded     */

} NPF_F_ATM_AAL2CpsPktByteCtrs_t ;

/* statistics counters accumulated by the CPS sub-layer */
typedef struct {
    /* errno = 0; I.363.2 */
    /* The parity of the STF indicates transmission errors              */
    NPF_uint32_t errCpsParityRx;   /* CPS PDU with parity error         */

    /* errno = 1; I.363.2 */
    /* The sequence number of the STF is wrong                          */
    NPF_uint32_t errCpsSeqNoRx;    /* CPS PDU with sequence no. error    */

    /* errno = 2; I.363.2 */
    /* The number of octets expected for a CPS-Packet overlapping into
```

```
       this CPS-PDU does not match the information contained in the STF  */
    NPF_uint32_t errCpsOsfUnex;    /* CPS PDU with unexpected offset     */

    /* errno = 3; I.363.2 */
    /* The OSF of the STF contains a value 48 or greater                 */
    NPF_uint32_t errCpsOffsetRx;   /* CPS PDU with offset field error    */

    /* errno = 4; I.363.2 */
    /* The Header Error Control (HEC) Code of a CPS-Packet header
       indicates transmission errors in the CPS-Packet header            */
    NPF_uint32_t errCpsHecRx;      /* CPS packets with CRC error         */

    /* errno = 5; I.363.2 */
    /* The length of the received CPS-Packet Payload (CPS-SDU) exceeds
       the maximum length indicated in "Max_SDU_Deliver_Length".         */
    NPF_uint32_t errCpsLenRx;

    /* errno = 6; I.363.2 */
    /* Number of times reassembly cancelled due to errno = 0, 1 or 2     */
    NPF_uint32_t numReasmCancel;

    /* errno = 7; I.363.2 */
    /* The Header Error Control (HEC) Code of a CPS-Packet header that
     * was overlapping a CPS-PDU boundary indicates transmission errors in
     * the CPS-Packet header; if the value of the OSF is less than 47,
     * processing starts at the octet pointed to by the OSF.             */
    NPF_uint32_t numPhBfrResetErrHec;

    /* errno = 8; I.363.2 */
    /* The UUI field in the received CPS-Packet header contains a value
       ("28" or "29") that is reserved for future standardization.       */
    NPF_uint32_t errBadUUIRx;      /* Reserved UUI; unexpected UUI Rx    */

    /* errno = 9; I.363.2 */
    /* The CID value in the received CPS-Packet header is not associated
       with a SAP.                                                       */
    NPF_uint32_t errBadCIDRx;      /* Reserved CID;Unknown CID value Rx  */

    /* The packet and byte counters for the number of CPS packets received*
     * and transmitted are maintained per priority level and are returned *
     * as an array. The number of elements in the array is equal to number*
     * of priority levels configured in the QoS profile for the AAL2 path */
    NPF_F_ATM_AAL2CpsPktByteCtrs_t  *cpsPktByteCtrArr;
} NPF_F_ATM_AAL2CpsStats_t;

/* statistics counters accumulated by the SAR SSCS sub-layer */
typedef struct {
    /* errno = 10; I.366.1 */
    /* The maximum permissible size for a reassembled SSSAR-SDU
      ("Max_SDU_Length") has been exceeded.                              */
    NPF_uint32_t errSscsOversizedSssarSduRx;      /* Oversized SSSAR SDU */

    /* errno = 11; I.366.1 */
    /* The reassembly timer RAS_Timer has expired.                       */
    NPF_uint32_t errSscsSssarRasTimerExpiryRx;    /* Reassembly timeout  */

    /* errno = 20; I.366.1 */
    /* An SSTED-PDU of length 8 or less has been received.               */
    NPF_uint32_t errSscsUndersizedSstedPduRx;     /* PDU<8 bytes received*/
```

```
    /* errno = 21; I.366.1 */
    /* The value of the Length field in the SSTED-PDU does not match the
       length of the received SSTED-PDU.                                  */
    NPF_uint32_t errSscsSstedPduLengthMismatchRx; /* Length mismatch     */

    /* errno = 22; I.366.1 */
    /* The value of the CRC field is not equal to the CRC calculated over
       the received information.                                          */
    NPF_uint32_t errSscsSstedCrcMismatchRx;       /* SSTED CRC mismatch  */

} NPF_F_ATM_AAL2SarSscsStats_t;

/* statistics counters accumulated for AAL2 paths.*/
typedef struct {
    /* CPS sub layer statistics */
    NPF_F_ATM_AAL2CpsStats_t        cpsStats;

    /* SAR SSCS sub layer statistics. Accumulated if the SAR SSCS is
     * associated with the AAL2 channels                                */
    NPF_F_ATM_AAL2SarSscsStats_t    sarSscsStats;
} NPF_F_ATM_AAL2Stats_t;

/* ATM Adaptation Layer type 5 statistics */
typedef struct {
    NPF_uint64_t framesRx;         /* Receive AAL5 Frames                 */
    NPF_uint32_t errBadCrcRx;      /* Receive AAL5 Frames with CRC error  */
    NPF_uint32_t errBadLenRx;      /* Receive AAL5 frames with length err */
    NPF_uint64_t bytesRx;          /* Receive Bytes                       */

    NPF_uint64_t framesTx;         /* Transmit AAL5 Frames                */
    NPF_uint64_t bytesTx;          /* Transmit Bytes                      */
} NPF_F_ATM_AAL5Stats_t;

/*
* ATM Per-Vp Statistics, returned in asynchronous response
*/
typedef struct {
    NPF_F_ATM_Stats_t          atmStats;     /* ATM Level statistics     */
} NPF_F_ATM_ConfigMgr_VpStats_t;

/*
* ATM Per-Vc Statistics, returned in asynchronous
*/
typedef struct {
    NPF_F_ATM_Stats_t          atmStats;     /* ATM Level statistics     */
    NPF_F_ATM_AAL_t            aal;          /* AAL type                 */
    union {
       NPF_F_ATM_AAL1Stats_t  aal1_stats;   /* AAL1 Statistics          */
       NPF_F_ATM_AAL2Stats_t  aal2_stats;   /* AAL2 Statistics          */
       NPF_F_ATM_AAL5Stats_t  aal5_stats;   /* AAL5 Statistics          */
    }u;
} NPF_F_ATM_ConfigMgr_VcStats_t;

/************************************************************
 * Enumerations and types for ATM AAL2 channel attributes
 ************************************************************/
/*
 * AAL2 channel CPS configuration
```

```
 */
typedef struct {
    NPF_F_AAL2ChanId_t      aal2ChanId; /* ID assigned to this AAL2 chnl.  */
    NPF_F_ATM_VirtLinkID_t  aal2path;   /* ID of VC constituting AAL2 path */
    NPF_uint8_t             aal2Cid;    /* The CID for this channel        */

     NPF_uint32_t            chnlPrio;  /* Priority of this AAL2 channel   */
                                        /* must be < maxPrio configured on *
                                         * the corresponding AAL2 path     */

    /* This parameter indicates the maximum size CPS-SDU, in octets, that is
       transported on a particular AAL type 2 channel. It also indicates the
       maximum size CPS-SDU that may be delivered to the corresponding CPS
       user. This  parameter  can take on the values "45" or "64" and is set
       by signaling or management procedures. The following inequality must
       be maintained -    maxSduDeliverLen <=cpsMaxSduLen                   */
    NPF_F_ATM_AAL2_CpsSduLen_t  maxSduDeliverLength;
} NPF_F_ATM_AAL2_Chnl_CpsCfg_t;

/*
 * AAL2 channel SAR SSCS configuration - filled when SSCS configured for the
 * the AAL2 path is SAR SSCS
 */
typedef struct {
    /* Common SSCS Parameters (SSCS type = SAR)as specified in I.366.1     */
    /* Selection of the transmission error detection  mechanisms (SSTED)   */
    NPF_boolean_t       sstedStatus;          /* SSTED selected?           */

    /* Selection of the assured data transfer mechanism (SSADT)            */
    /* When ssadtStatus is set to TRUE, the sstedStatus MUST be set to TRUE*/
    NPF_boolean_t       ssadtStatus;          /* SSADT selected?           */

    /* Maximum SSSAR SDU length in  bytes                                  */
    /* This parameter indicates the maximum size SSSAR-SDU, in octets, that
     * is allowed to be reassembled. Valid values are between 1 and 65568  */
    NPF_uint32_t        maxSssarSduLength;   /* Max SSSAR-SDU size?        */

    /* Maximum size of segments used to create SSSAR PDU                   */
    /* This must bet set between 1 and maxSduDeliverLength specified in the*/
    /* CPS configuration portion of the AAL2 channel configuration struct  */
    /* This parameter can assume a value between 1 to 45 or 1 to 64 based  *
     * on the configuration of the maxSduDeliverLength field               */
    NPF_uint8_t         maxSssarSegLength;

    /* Timer configured to guard re-assembly process for SSSAR segments    */
    NPF_F_ATM_Timers_t  rasTimer;       /* RAS Timer                       */

} NPF_F_ATM_AAL2_Chnl_SarSscsCfg_t;


/*
*   AAL2 Channel Config Info.
*/
typedef struct {
    /* AAL2 channel CPS sub layer configuration parameters */
    NPF_F_ATM_AAL2_Chnl_CpsCfg_t        cpsConfig;

    /* AAL2 channel SAR SSCS sub layer configuration parameters */
    NPF_F_ATM_AAL2_Chnl_SarSscsCfg_t    sarSscsConfig;
```

```
    /* The below field provides the administratively configured status of  *
     * the channel. The actual status assumed by the channel may be        *
     * different based on the actual physical status. The operation        *
     * status may be queried using the function provided to query          *
     * the operational status.                                             */
    NPF_ObjStatus_t        admStatus; /*Status of channel-UP/DOWN/TESTING  */

    /* The below field provides the status of statistics collections on the*
    NPF_TRUE, statistics collection is enabled.    *
     * The statistics collection may also be enabled or disabled at a      *
     * future point in time by issueing the statistics enable/disable      *
     * function call. The current statistics collection state may be       *
     * queried using the function provided to query the channel information*/
    NPF_boolean_t         statsEnabled;       /* Statistic collecting state*/

    /* The below field is used to cross connect AAL2 channel to another    *
     * AAL2 channel on the same FE or to a backplane switch link. When     *
     * the numLink_B field is set to 0, it indicates there is no cross     *
     * connection established for this AAL2 channel. For a unicast mapping,*
     * the AAL2 channel being provisioned may be connected to another      *
     * AAL2 channel or backplane link and the numLink_B is set to 1.       *
     * For multicast mapping, the numLink_B is set to the number of        *
     * cross connections to be made.                                       */
    NPF_uint32_t  numLink_B;      /* Number of links connected to the AAL2 *
                                     Channel being provisioned             */
    NPF_F_ATM_AAL2_ChnlXcInfo_t           *link_B;   /* Mapped link 'B'    */

} NPF_F_ATM_ConfigMgr_AAL2_Channel_t;

/*
* AAL2 Channel query response
*/
typedef struct {
     /* AAL2 channel configuration information */
    NPF_F_ATM_ConfigMgr_AAL2_Channel_t  chnlCfg;

     /* chnlBound indicates if the channel is bound to a child interface. *
      * If TRUE, the interface handle of the child interface is returned  *
      * in the ifChildHandle field                                        */
    NPF_boolean_t               chnlBound;   /* Whether bound to child I/F*/
    NPF_IfHandle_t          ifChildHandle;   /* Bound interface           */

    NPF_ObjStatus_t             operStatus;  /* Operational status of chnl*
                                              * i.e UP/DOWN/TESTING       */
} NPF_F_ATM_ConfigMgr_AAL2_ChannelInfo_t;

/*************************************************************
 * Enumerations and types for ATM AAL2 channel Statistics
 *************************************************************/
/* statistics counters accumulated by the CPS sub-layer. */
typedef struct {
    /* errno = 5; I.363.2 */
    /* The length of the received CPS-Packet Payload (CPS-SDU) exceeds
       the maximum length indicated in "Max_SDU_Deliver_Length".         */
    NPF_uint32_t errCpsLenRx;

    /* errno = 6; I.363.2 */
    /* Number of times reassembly cancelled due to errno = 0, 1 or 2     */
```

```
    NPF_uint32_t numReasmCancel;

    /* errno = 7; I.363.2 */
    /* The Header Error Control (HEC) Code of a CPS-Packet header that
     * was overlapping a CPS-PDU boundary indicates transmission errors in
     * The CPS-Packet header; if the value of the OSF is less than 47,
     * Processing starts at the octet pointed to by the OSF.            */
    NPF_uint32_t numPhBfrResetErrHec;

    /* errno = 8; I.363.2 */
    /* The UUI field in the received CPS-Packet header contains a value
       ("28" or "29") that is reserved for future standardization.      */
    NPF_uint32_t errBadUUIRx;      /* Reserved UUI; unexpected UUI Rx    */

    NPF_uint64_t cpsPktRx;         /* No. of AAL2 CPS packets received   */
    NPF_uint64_t cpsBytesRx;       /* No. CPS packet bytes received      */
    NPF_uint64_t cpsPktTx;         /* No. of AAL2 CPS packets transmitted*/
    NPF_uint64_t cpsBytesTx;       /* No. CPS packet bytes transmitted   */
} NPF_F_ATM_AAL2ChnlCpsStats_t;

/* statistics counters accumulated by the SAR SSCS sub-layer */
typedef struct {
    /* errno = 10; I.366.1 */
    /* The maximum permissible size for a reassembled SSSAR-SDU
      ("Max_SDU_Length") has been exceeded.                              */
    NPF_uint32_t errSscsOversizedSssarSduRx;      /* Oversized SSSAR SDU */

    /* errno = 11; I.366.1 */
    /* The reassembly timer RAS_Timer has expired.                      */
    NPF_uint32_t errSscsSssarRasTimerExpiryRx;    /* Reassembly timeout  */

    /* errno = 20; I.366.1 */
    /* An SSTED-PDU of length 8 or less has been received.              */
    NPF_uint32_t errSscsUndersizedSstedPduRx;     /* PDU<8 bytes received*/

    /* errno = 21; I.366.1 */
    /* The value of the Length field in the SSTED-PDU does not match the
       length of the received SSTED-PDU.                                */
    NPF_uint32_t errSscsSstedPduLengthMismatchRx; /* Length mismatch     */

    /* errno = 22; I.366.1 */
    /* The value of the CRC field is not equal to the CRC calculated over
       the received information.                                        */
    NPF_uint32_t errSscsSstedCrcMismatchRx;       /* SSTED CRC mismatch  */
} NPF_F_ATM_AAL2ChnlSarSscsStats_t;

/* statistics counters for the AAl2 channels. */
typedef struct {
    /* CPS sub layer statistics */
    NPF_F_ATM_AAL2ChnlCpsStats_t     cpsStats;

    /* SAR SSCS sub layer statistics. Accumulated if the SAR SSCS is
     * associated with the AAL2 channels                                */
    NPF_F_ATM_AAL2ChnlSarSscsStats_t sarSscsStats;
} NPF_F_ATM_ConfigMgr_AAL2ChannelStats_t;

/*************************************************************
 * Enumerations and types for ATM Interface attributes and stats
 *************************************************************/
```

```
/* ATM Interface Type */
typedef enum {
    NPF_F_ATM_IF_UNI = 0,  /* UNI Interface */
    NPF_F_ATM_IF_NNI       /* NNI Interface */
} NPF_F_ATM_IfType_t;
/* ATM Interface Characteristics */
typedef struct {
    NPF_F_ATM_IfID_t       ifID;      /* Interface handle                */
    NPF_F_ATM_IfType_t     ifType;    /* Interface UNI/NNI               */
    NPF_F_ATM_OAM_CPID_t   cpId;      /* Connect point ID                */
} NPF_F_ATM_ConfigMgr_IfCfg_t;
/* ATM Interface Statistics */
typedef struct {
    NPF_uint64_t      cellsClp01Rx;   /* Receive Total Cells (CLP0 + CLP1) */
    NPF_uint64_t      cellsClp0Rx;    /* Receive High Priority Cells (CLP0)*/
    NPF_uint64_t      cellsClp01Tx;   /* Transmit Total Cells (CLP0 + CLP1)*/
    NPF_uint64_t      cellsClp0Tx;    /* Transmit High Priority Cells(CLP0)*/
    NPF_uint32_t      unexCellsRx;    /* Receive cells w/unexpected VPI/VCI*/
    NPF_uint32_t      unexSecsRx;     /* Rx Seconds of Unexpected VPI/VCI  */
    NPF_F_ATM_VcAddr_t lastUnexATMHdr;/* ATM header of last unexpected cell*/
} NPF_F_ATM_ConfigMgr_IfStats_t;



/*
* OAM CP Configuration information
*/
typedef struct {
    NPF_F_ATM_VirtLink_t    linkId;     /* VP or VC Link                   */
    NPF_F_ATM_OAM_CP_Type_t cpType;     /* Connection point type           */
    NPF_boolean_t           llidOption; /* Is LLID option enabled ?        */
} NPF_F_ATM_ConfigMgr_OAM_CP_t;
/*
* OAM CC Activation/Deactivation
*/
typedef struct {
    NPF_F_ATM_VirtLink_t         linkId;   /* VP or VC Link */
    NPF_F_ATM_OAM_FlowType_t     flowType; /* segment/end-to-end           */
    NPF_F_ATM_OAM_OperType_t     operType; /*Activate/Deactivate/Start/Stop*/
    NPF_F_ATM_OAM_Direction_t    direction;/* Away/towards/both            */
    NPF_F_ATM_OAM_CC_Method_t    ccMethod; /* Options to send CC Cell
                                              0-Send when no user cell
                                              1-Send periodically          */
} NPF_F_ATM_ConfigMgr_OAM_CC_t;

/*
* OAM CC Response
*/
typedef struct {
    NPF_F_ATM_VirtLink_t          linkId;   /* VP or VC Link */
    NPF_F_ATM_OAM_FlowType_t       flowType;/* segment/end-to-end     */
    NPF_F_ATM_OAM_CC_RspType_t     ccRsp;
    NPF_F_ATM_OAM_CC_Method_t     ccMethod;   /* For CC source and ACCEPT
only.
                                              Option to send cc cell
                                              0 - to send when no user cell
                                              1 - to force send */
} NPF_F_ATM_ConfigMgr_OAM_CC_Rsp_t;
/*
```

```
* OAM PM Activation/Deactivation - FPM-BR/FPM
*/
typedef struct {
    NPF_F_ATM_VirtLink_t         linkId;   /* VP or VC Link*/
    NPF_F_ATM_OAM_FlowType_t     flowType; /* segment/end-to-end          */
    NPF_F_ATM_OAM_PM_FuncType_t  funcType; /* FPM-BR/FPM                  */
    NPF_F_ATM_OAM_Direction_t    dir;      /* Direction of operation      */
    NPF_F_ATM_OAM_BlkSize_t      fwdSize;  /* A-B block size              */
    NPF_F_ATM_OAM_BlkSize_t      bwdSize;  /* B-A block size              */
    NPF_F_ATM_OAM_OperType_t     operType; /* Act/Deactivate/Start/Stop   */
} NPF_F_ATM_ConfigMgr_OAM_PM_t;

/* Performance monitoring statistics request data type */
typedef struct {
    NPF_F_ATM_VirtLink_t      linkId;             /* VP or VC Link       */
    NPF_F_ATM_OAM_FlowType_t  flowType;           /* segment/end-to-end  */
    NPF_boolean_t             zeroStats;          /* clear stats if TRUE */
} NPF_F_ATM_ConfigMgr_OAM_PM_StatsReq_t;
/* Performance Monitoring Statistics Data Type */
/* Errored cell block statistics; Does not include counts for severely  *
 * errored cell blocks which are provided as part of SECB stats         */
typedef struct {
    /*Number of misinserted CLP0+1 cells in errored cell blocks         */
    NPF_uint32_t      numMisinsertedCLP01cells;
    /*Number of misinserted CLP0 cells in errored cell blocks           */
    NPF_uint32_t      numMisinsertedCLP0cells;
    /* Number of lost CLP0+1 users cells in errored cell blocks         */
    NPF_uint32_t      numLostCLP01cells;
    /* Number of lost CLP0 users cells in errored cell blocks           */
    NPF_uint32_t      numLostCLP0cells;
    /* Number of errored cell bits in errored cell blocks               */
    NPF_uint32_t      numErroredCLP01Bits;
    /* Number of cell blocks with CLP0+1 losses                         */
    NPF_uint32_t      numBlocksClp01Loss;
    /* Number of cell blocks with CLP0 losses                           */
    NPF_uint32_t      numBlocksClp0Loss;
    /* Number of cell blocks with CLP0+1 Misinsertions                  */
    NPF_uint32_t      numBlocksClp01Misinsertion;
    /* Number of cell blocks with CLP0 Misinsertions                    */
    NPF_uint32_t      numBlocksClp0Misinsertion;
    /* Number of cell blocks with errors i.e cell loss, misinsertion or *
     * bit errors                                                       */
    NPF_uint32_t      totalErroredCellBlocks;
} NPF_F_ATM_OAM_ErrCB_Stats_t;
/* Severely errored cell block (SECB) statistics. Please see I.610 and I.356
 * For definition of severely errored cell blocks.
 * The counts here do not include the non severely errored cell blocks
 */
typedef struct {
    /*Number of misinserted CLP0+1 cells in severely errored cell blocks*/
    NPF_uint32_t      numMisinsertedCLP01cells;
    /*Number of misinserted CLP0 cells in severely errored cell blocks  */
    NPF_uint32_t      numMisinsertedCLP0cells;
    /* Number of lost CLP0+1 users cells in severely errored cell blocks*/
    NPF_uint32_t      numLostCLP01cells;
    /* Number of lost CLP0 users cells in severely errored cell blocks  */
    NPF_uint32_t      numLostCLP0cells;
    /* Number of errored cell bits in severely errored cell blocks      */
    NPF_uint32_t      numErroredCLP01Bits;
```

```
    /* Number of severely errored cell blocks with CLP0+1 losses        */
    NPF_uint32_t      numSECBClp01Loss;
    /* Number of severely errored cell blocks with CLP0 losses          */
    NPF_uint32_t      numSECBClp0Loss;
    /* Number of severely errored cell blocks with CLP0+1 Misinsertions */
    NPF_uint32_t      numSECBClp01Misinsertion;
    /* Number of severely errored cell blocks with CLP0 Misinsertions   */
    NPF_uint32_t      numSECBClp0Misinsertion;
    /* Total Number of severely errored cell blocks with errors i.e     *
     * cell loss, misinsetion or bit errors                             */
    NPF_uint32_t      totalSECB;
} NPF_F_ATM_OAM_SevErrCB_Stats_t;

/* This structure contains the PM stats counters for one direction      */
typedef struct {
    NPF_uint32_t                    averageBlockSize; /* Avg. block size */
    NPF_uint32_t                    minBlockSize;     /* Min block size  */
    NPF_uint32_t                    maxBlockSize;     /* Max block size  */

    /* Lost PM cells is the count of FPM cell lost in the B->A direction *
     * and the lost of BR cells in the A->B direction                    */
    NPF_uint32_t                    lostPMcells;      /*  PM cells lost  */

    /* Errored cell block statistics */
    NPF_F_ATM_OAM_ErrCB_Stats_t     erroredCBStats;   /* Errored CB stats*/
    /* Severely errored cell block statistics */
    NPF_F_ATM_OAM_SevErrCB_Stats_t  secbStats;        /* SECB stats      */
} NPF_F_ATM_OAM_PM_Stats_Info_t;

/* Performance monitoring statistics returned in response to FAPI client *
 * query. The counts are accumulated since the last time the counters    *
 * were zeroed by the FAPI client. When performance monitoring is enabled*
 * the counters start from 0.                                            */

typedef struct {
    /* Direction in which performance management procedure is activated *
     * is either Forward (A->B), Backward (B->A) or both A->B and B->A  */
    NPF_F_ATM_OAM_Direction_t    dir;

    /* Performance monitoring statistics for A->B direction. The counter*
     * in this structure are valid if PM is enabled in forward direction*
     * i.e. A->B direction or in both directions. These counters are    *
     * updated on reception of a BR from the B connection point         *
     * The direction field indicates direction in which PM is enabled   */
    NPF_F_ATM_OAM_PM_Stats_Info_t   oamPMStatsAtoB;

    /* Performance monitoring statistics for B->A direction. The counter*
     * in this structure are valid if PM is enabled in forward direction*
     * i.e. B->A direction or in both directions. These counters are    *
     * updated on reception of a FPM from the B connection point        *
     * The direction field indicates direction in which PM is enabled   */
    NPF_F_ATM_OAM_PM_Stats_Info_t   oamPMStatsBtoA;
} NPF_F_ATM_ConfigMgr_OAM_PM_Stats_t;
/*
* OAM Loopback
*/
typedef struct {
    NPF_F_ATM_VirtLink_t     linkId;     /* VP or VC Link                */
    NPF_F_ATM_OAM_FlowType_t flowType;   /* segment/end-to-end           */
```

```
    NPF_F_ATM_OAM_CPID_t     llId;       /* Loopback location Id          */
    NPF_boolean_t            remCell;    /* Remove returned cells ?       */
    NPF_boolean_t          includeSrcId; /* If source CP ID be included   */
                                         /* in LB cell                    */
} NPF_F_ATM_ConfigMgr_OAM_LB_t;

/*
*   Loopback Procedure Result
*/
typedef struct {
    NPF_uint8_t           numLbResp;       /* Number of Loopback responses */
    NPF_F_ATM_OAM_CPID_t  *llId;           /* Loopback Location Id         */
} NPF_F_ATM_ConfigMgr_OAM_LB_Result_t;
/*
* OAM Non-Intrusive Monitoring
*/
typedef struct {
    NPF_F_ATM_VirtLink_t     linkId;      /* VP or VC Link                */
    NPF_F_ATM_OAM_OperType_t operType;    /* Start/Stop Monitoring        */
    NPF_F_ATM_OAM_FlowType_t flowType;    /* segment/end-to-end           */
    NPF_F_ATM_OAM_CellType_t cellType;    /* Type of OAM cell to monitor */
} NPF_F_ATM_ConfigMgr_OAM_Mon_t;
/*
* OAM Alarms - Declare and Release AIS alarms
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t  flowType;           /* segment/end-to-end  */
    NPF_uint8_t               defectType;         /* Defect type         */
    NPF_uint8_t               defectLocation[16]; /* Defect Location     */
} NPF_F_ATM_OAM_Alarm_Info_t;

typedef struct {
    NPF_F_ATM_VirtLink_t        linkId;             /* VP or VC Link       */
    NPF_F_ATM_OAM_Alarm_Info_t  alarmInfo;          /* Alarm information   */
} NPF_F_ATM_ConfigMgr_OAM_Alarm_t;
/*
* OAM PM Response
*/
typedef struct {
    NPF_F_ATM_VirtLink_t           linkId;     /* VP or VC Link          */
    NPF_F_ATM_OAM_FlowType_t       flowType;   /* segment/end-to-end     */
    NPF_F_ATM_OAM_PM_RspType_t     pmRsp;      /* Accept/Reject response */
} NPF_F_ATM_ConfigMgr_OAM_PM_Rsp_t;
/*
*   AIS Alarm event info.
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t   flowType;           /* Segment/ETE         */
    NPF_uint16_t               defectType;         /* Defect type         */
    NPF_char8_t                defectLocation[16]; /* Defect Location     */
} NPF_F_ATM_OAM_AIS_Event_t;
/*
*   RDI Alarm event info.
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t  flowType;           /* Segment/ETE         */
    NPF_uint16_t              defectType;         /* Defect type         */
    NPF_char8_t               defectLocation[16]; /* Defect Location     */
} NPF_F_ATM_OAM_RDI_Event_t;
```

```
/*
*   FPM Cell Info
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t  flowType;       /* Segment/ETE               */
    NPF_uchar8_t              mscn;           /* FPM MCSN                  */
    NPF_uint16_t              totUsrCell01;   /* Total User Cell(CLP-0+1)  */
    NPF_uint16_t              totUsrCell0;    /* Total User Cell(CLP-0)    */
    NPF_uint16_t              blkErrDetCode;  /* Block Error Detection Code*/
    NPF_uint32_t              timeStamp;      /* Time Stamp */
} NPF_F_ATM_OAM_FPM_Event_t;
/*
*   BR Cell Info
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t  flowType;     /* Segment/ETE                 */
    NPF_uchar8_t              mscn;         /* BR Monitoring Cell Seq. No. */
    NPF_uint16_t              totUsrCell01; /* 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 MCSN               */
    NPF_uchar8_t              secbc;        /* Severely Err. Cell block    */
    NPF_uint16_t              totRcvdUsrCell0;/* Total Rx User Cell(CLP-0) */
    NPF_uint8_t               blkErr;       /* Block Error Result(CLP-0+1) */
    NPF_uint16_t              totRcvdUsrCell1;/* Total Rx User Cell(CLP0+1)*/
} NPF_F_ATM_OAM_BR_Event_t;
/*
*   Loopback Cell Info
*/
typedef struct {
    NPF_F_ATM_OAM_FlowType_t    flowType;        /* Segment/ETE          */
    NPF_boolean_t               sourceIdValid;   /* If sourceID in Ind   */
    NPF_uint8_t                 sourceId[16];    /* source ID            */
    NPF_uint8_t                 llid[16];        /* loopback location ID */
} NPF_F_ATM_OAM_LB_Event_t;
/* CC Cell Information Data Type */
typedef struct
{
    NPF_F_ATM_OAM_FlowType_t    flowType;      /* Segment/ETE          */
} NPF_F_ATM_OAM_CC_Event_t;
/* Performance Monitoring Statistics Event Data Type */
typedef enum
{
    NPF_F_ATM_OAM_PM_STATS_TRIGGER_FPM_RECEIVED=0, /* Indication due to FPM
*/
    NPF_F_ATM_OAM_OAM_PM_STATS_TRIGGER_BR_RECEIVED /* Indication due to BR
*/
} NPF_F_ATM_OAM_PmStatsTrigger_t;

typedef struct
{
    NPF_F_ATM_OAM_FlowType_t     flowType;         /* Segment/ETE          */
    NPF_F_ATM_OAM_PmStatsTrigger_t pmStatsTrigger; /* FPM or BR received   */
    NPF_uint8_t              mcsn;   /* BR Monitoring cell seq no          */
    NPF_uint16_t             tuc01;  /* Total user cell (CLP 0+1)          */
    NPF_uint16_t             tuc0;   /* Total user cell (CLP 0)            */
    NPF_uint32_t             tstp;   /* Time stamp                         */
    NPF_uint8_t              rMcsn;  /* Reported Monitoring Cell seq No.   */
    NPF_uint8_t              secbc;  /* Severely Errored Cell Block Count  */
```

```
    NPF_uint16_t            trcc01; /* Total received user cell(CLP 0+1)  */
    NPF_uint16_t            trcc0;  /* Total received user cell(CLP 0)    */
    NPF_uint8_t             bler01; /* Block error result (CLP 0+1)       */
} NPF_F_ATM_OAM_PM_Stats_Event_t;
/* PM procedure activation and deactivation event */
typedef struct {
    NPF_F_ATM_OAM_FlowType_t      flowType;       /* Segment/ETE          */
    NPF_F_ATM_OAM_PM_FuncType_t   functionType;   /* FPM-BR/FPM           */
    NPF_F_ATM_OAM_Direction_t     oamDirection;   /* A->B/B->A/Bothway    */
    NPF_F_ATM_OAM_BlkSize_t       fwdBlkSize;     /* Forward block size   */
    NPF_F_ATM_OAM_BlkSize_t       backBlkSize;    /* Backward block size  */
} NPF_F_ATM_OAM_PM_ActDeact_Event_t;
/* CC procedure activation and deactivation event */
typedef struct {
    NPF_F_ATM_OAM_FlowType_t      flowType;    /* Segment/ETE          */
    NPF_F_ATM_OAM_Direction_t  oamDirection;   /* A->B/B->A/Bothway    */
} NPF_F_ATM_OAM_CC_ActDeact_Event_t;


/*****************************************************************************
*
 * Completion callbacks and event registration types and
 * function prototypes for the config. manager LFB

*****************************************************************************
/

/*
* Completion Callback Types, to be found in the callback
* data structure, NPF_F_ATM_ConfigMgr_CallbackData_t.
*/
typedef enum NPF_F_ATM_ConfigMgr_CallbackType {
    NPF_F_ATM_CONFIGMGR_IF_SET = 1,
    NPF_F_ATM_CONFIGMGR_IF_DELETE = 2,
    NPF_F_ATM_CONFIGMGR_IF_STATS_GET = 3,
    NPF_F_ATM_CONFIGMGR_IF_STATS_ENABLE = 4,
    NPF_F_ATM_CONFIGMGR_IF_STATS_DISABLE = 5,
    NPF_F_ATM_CONFIGMGR_IF_QUERY = 6,
    NPF_F_ATM_CONFIGMGR_IF_AIS_STATE_SET = 7,
    NPF_F_ATM_CONFIGMGR_IF_AIS_STATE_CLEAR = 8,

    NPF_F_ATM_CONFIGMGR_VC_SET = 9,
    NPF_F_ATM_CONFIGMGR_VP_SET = 10,
    NPF_F_ATM_CONFIGMGR_VC_BIND = 11,
    NPF_F_ATM_CONFIGMGR_VC_UNBIND = 12,
    NPF_F_ATM_CONFIGMGR_VL_DELETE = 13,
    NPF_F_ATM_CONFIGMGR_VL_ENABLE = 14,
    NPF_F_ATM_CONFIGMGR_VL_DISABLE = 15,
    NPF_F_ATM_CONFIGMGR_VL_OPER_STATUS_GET = 16,
    NPF_F_ATM_CONFIGMGR_VL_STATS_ENABLE = 17,
    NPF_F_ATM_CONFIGMGR_VL_STATS_DISABLE = 18,
    NPF_F_ATM_CONFIGMGR_VC_STATS_GET = 19,
    NPF_F_ATM_CONFIGMGR_VC_QUERY = 20,
    NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_SET = 21,
    NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_DELETE = 22,
    NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_QUERY = 23,
    NPF_F_ATM_CONFIGMGR_VP_STATS_GET = 24,
    NPF_F_ATM_CONFIGMGR_VP_QUERY = 25,
    NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_SET = 26,
```

```
    NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_DELETE = 27,
    NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_QUERY = 28,

    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_SET = 29,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_BIND = 30,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_UNBIND = 31,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_DELETE = 32,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_GET = 33,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_QUERY = 34,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_ENABLE = 35,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_DISABLE = 36,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_OPER_STATUS_GET = 37,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_ENABLE = 38,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_DISABLE = 39,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_SET = 40,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_DELETE = 41,
    NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_QUERY = 42,

    NPF_F_ATM_CONFIGMGR_OAM_CP_SET = 43,
    NPF_F_ATM_CONFIGMGR_OAM_CC_SET = 44,
    NPF_F_ATM_CONFIGMGR_OAM_CC_RSP = 45,
    NPF_F_ATM_CONFIGMGR_OAM_PM_SET = 46,
    NPF_F_ATM_CONFIGMGR_OAM_PM_RSP = 47,
    NPF_F_ATM_CONFIGMGR_OAM_PM_STATS_GET = 48,
    NPF_F_ATM_CONFIGMGR_OAM_LB_SET = 49,
    NPF_F_ATM_CONFIGMGR_OAM_MONITOR = 50,
    NPF_F_ATM_CONFIGMGR_OAM_ALARM_SET = 51,
    NPF_F_ATM_CONFIGMGR_OAM_ALARM_CLEAR = 52,
} NPF_F_ATM_ConfigMgr_CallbackType_t;

/* Asynchronous Response Callback Data Type */

/*
* This union is a handy way of representing the various object identifiers
* used by the APIs.
*/
typedef union {
    NPF_F_ATM_IfID_t          ifID;       /* Interface ID                */
    NPF_F_ATM_VirtLink_t      linkId;     /* VP or VC Link               */
    NPF_F_AAL2ChanId_t        aal2ChanId; /* AAL2 channel ID             */
    NPF_F_VcXcId_t            vcXcId;     /* VC cross connect ID         */
    NPF_F_VpXcId_t            vpXcId;     /* VP cross connect ID         */
    NPF_F_AAL2ChnlXcId_t      chnlXcId;   /* AAL2 Chnl cross connect ID  */
} NPF_F_ATM_ConfigMgr_Id_t;
/*
* An asynchronous response contains an configuration object ID,
* a error or success code, and in some cases a function-
* specific structure embedded in a union. One or more of
* these is passed to the callback function as an array
* within the NPF_F_ATM_ConfigManager_CallbackData_t structure (below)
*/

typedef struct {
    NPF_error_t                 error;      /* Error code for this resp    */
    NPF_F_ATM_ConfigMgr_Id_t    objId;      /* Object Identifier           */
    union {
        /* Function-specific structures: */
        NPF_uint32_t                            unused;
```

```
        /* Queried VC statistics.
        NPF_F_ATM_CONFIGMGR_VC_STATS_GET */
        NPF_F_ATM_ConfigMgr_VcStats_t           vcStats;

        /* Queried virtual path statistics
        NPF_F_ATM_CONFIGMGR_VP_STATS_GET */
        NPF_F_ATM_ConfigMgr_VpStats_t           vpStats;

        /* Queried AAL2 channel statistics
        NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_STATS_GET */
        NPF_F_ATM_ConfigMgr_AAL2ChannelStats_t  aal2ChanStats;

        /* Result of get operational status; status of link
        NPF_F_ATM_CONFIGMGR_VL_OPER_STATUS_GET */
        NPF_ObjStatus_t                         operStatus;

        /* Result of initiated loopback procedure
        NPF_F_ATM_CONFIGMGR_OAM_CC_RSP */
        NPF_F_ATM_ConfigMgr_OAM_LB_Result_t     lbResult;

        /* ATM virtual channel configuration and status
        NPF_F_ATM_CONFIGMGR_VC_QUERY */
        NPF_F_ATM_ConfigMgr_VcInfo_t      vcConfigInfo;

        /* ATM virtual path configuration and status
        NPF_F_ATM_CONFIGMGR_VP_QUERY */
        NPF_F_ATM_ConfigMgr_VpInfo_t      vpConfigInfo;

        /* AAL2 channel configuration and status
        NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_QUERY */
        NPF_F_ATM_ConfigMgr_AAL2_ChannelInfo_t  chnlConfigInfo;

        /* VC cross connect attributes
        NPF_F_ATM_CONFIGMGR_VC_CROSSCONNECT_QUERY */
        NPF_F_ATM_ConfigMgr_VcLinkXc_t          vcXcConfig;

        /* VP link cross connect attributes
        NPF_F_ATM_CONFIGMGR_VP_CROSSCONNECT_QUERY */
        NPF_F_ATM_ConfigMgr_VpLinkXc_t          vpXcConfig;

        /* AAL2 channel cross connect attributes
         * Completion callback -
        NPF_F_ATM_CONFIGMGR_AAL2_CHANNEL_CROSSCONNECT_QUERY */
        NPF_F_ATM_ConfigMgr_AAL2_ChnlXc_t       chnlXcConfig;

        /* Queried Interface statistics
        NPF_F_ATM_CONFIGMGR_IF_STATS_GET */
        NPF_F_ATM_ConfigMgr_IfStats_t           ifStats;

        /* ATM interface attributes
        NPF_F_ATM_CONFIGMGR_IF_QUERY */
        NPF_F_ATM_ConfigMgr_IfCfg_t         ifConfig;

        /* OAM performance monitoring stats for requested flow
        NPF_F_ATM_CONFIGMGR_OAM_PM_STATS_GET */
        NPF_F_ATM_ConfigMgr_OAM_PM_Stats_t      pmStats;

    } u;
} NPF_F_ATM_ConfigMgr_AsyncResponse_t;
```

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

/*
 *   ATM OAM Fault Management Event Notification
 */
typedef struct {
    NPF_F_ATM_VirtLink_t                linkId;    /* VP or VC Link        */
    /* Detailed information for the Alarm being raised/clear */
    NPF_F_ATM_OAM_Alarm_Info_t          oamAlarmInfo;
} NPF_F_ATM_ConfigMgr_OAM_FM_EventData_t;

/*
*   ATM OAM Performance Management Event Notification
*/
typedef struct {
    NPF_F_ATM_VirtLink_t            linkId;       /* VP or VC Link     */
    NPF_F_ATM_OAM_PM_Stats_Event_t  oamPmStats;   /* Perf. Mon. stats  */
} NPF_F_ATM_ConfigMgr_OAM_PM_EventData_t;

/*
*   ATM OAM activation/deactivation Event Notification
*/
typedef struct {
    NPF_F_ATM_VirtLink_t                      linkId;   /* VP or VC Link   */
    union {
         /* PM activate/deactivation Request indication*/
         NPF_F_ATM_OAM_PM_ActDeact_Event_t oamPmActDeact;
```

```
        /* CC activate/deact Request indication       */
        NPF_F_ATM_OAM_CC_ActDeact_Event_t oamCcActDeact;
    }u;
} NPF_F_ATM_ConfigMgr_OAM_ActDeact_EventData_t;

/*
*   ATM OAM Passive Monitoring Event Notification
*/
typedef struct {
    NPF_F_ATM_VirtLink_t                    linkId;   /* VP or VC Link     */
      union {
          NPF_F_ATM_OAM_AIS_Event_t         oamAisInfo; /* AIS cell info   */
          NPF_F_ATM_OAM_RDI_Event_t         oamRdiInfo; /* RDI cell info   */
          NPF_F_ATM_OAM_FPM_Event_t         oamFpmInfo; /* FPM cell info   */
          NPF_F_ATM_OAM_BR_Event_t          oamBrInfo;  /* BR cell info    */
          NPF_F_ATM_OAM_LB_Event_t          oamLBInfo;  /* LB cell info    */
          NPF_F_ATM_OAM_CC_Event_t          oamCCInfo;  /* CC cell info    */
          NPF_F_ATM_OAM_PM_ActDeact_Event_t pmADInfo;   /*PM Act/Deact cell*/
          NPF_F_ATM_OAM_CC_ActDeact_Event_t ccADInfo;   /*CC Act/Deact cell*/
    }u;
} NPF_F_ATM_ConfigMgr_OAM_Mon_EventData_t;

/*
 *    Traffic management threshold crossing notifications
 */
typedef struct {
    NPF_F_ATM_VirtLink_t    virtLinkId;  /* VP or VC Link                  */
    union {
       NPF_uint32_t           chnlPrio;  /* Priority of AAL2 path queue    *
                                          * that hit the discard threshold */
    } u;
} NPF_F_ATM_ConfigMgr_TrafMgmt_EventData_t;

/*
*   ATM configuration manager Event Types
*/
typedef enum {
    /* ATM OAM Fault Management Event Types */
    NPF_F_ATM_AIS_RAISED = 1,     /* AIS defect detected. AIS received     */
    NPF_F_ATM_RDI_RAISED = 2,     /* RDI defect detected. RDI received     */
    NPF_F_ATM_AIS_CLEARED = 3,    /* AIS defect cleared.                   */
    NPF_F_ATM_RDI_CLEARED = 4,    /* RDI defect cleared                    */
    NPF_F_ATM_LOC_RAISED = 5,     /* LOC defect detected                   */
    NPF_F_ATM_LOC_CLEARED = 6,    /* LOC defect cleared                    */

    /* ATM OAM Performance Management Event Types */
    NPF_F_ATM_PM_STATISTICS = 7,  /* Perf. Monitoring Stats                */

    /* CC/PM activate/deactivate indication */
    NPF_F_ATM_CC_ACT_DEACT_REQ_INDICATION = 8, /*CC activate/deactivare ind*/
    NPF_F_ATM_PM_ACT_DEACT_REQ_INDICATION = 9, /*PM activate/deactivate ind*/

    /* Passive Monitoring events */
    NPF_F_ATM_CC_RECEIVED  = 10,    /* CC cell received;Passive monitoring */
    NPF_F_ATM_FPM_RECEIVED = 11,    /* FPM cell received;Passive monitoring*/
    NPF_F_ATM_BR_RECEIVED  = 12,    /* BR cell received;passive monitoring */
    NPF_F_ATM_LB_RECEIVED  = 13,    /* LB cell received;passive monitoring */
    NPF_F_ATM_AIS_RECEIVED = 14,    /* AIS cell received;passive monitoring*/
    NPF_F_ATM_RDI_RECEIVED = 15,    /* RDI cell received;passive monitoring*/
```

```
    NPF_F_ATM_PM_ACT_DEACT_RX = 16, /* PM Act/Deact cell;passive monitoring*/
    NPF_F_ATM_CC_ACT_DEACT_RX = 17, /* CC Act/Deact cell;passive monitoring*/

    /* Traffic Management threshold crossing event */
    NPF_F_ATM_TM_HIT_WARN_THRESH = 18,/*Link Queue len hit warn threshold */
    NPF_F_ATM_TM_HIT_DROP_THRESH = 19,/*Link Queue len hit drop threshold */
    NPF_F_AAL2_TM_HIT_DROP_THRESH = 20,/*AAL2 Queue len hit drop threshold*/
} NPF_F_ATM_ConfigMgr_Event_t;

/*
 *   Definitions for selectively enabling ATM Configuration Manager Events
 */
#define NPF_F_ATM_AIS_RAISED_ENABLE                  (1 << 0)
#define NPF_F_ATM_RDI_RAISED_ENABLE                  (1 << 1)
#define NPF_F_ATM_AIS_CLEARED_ENABLE                 (1 << 2)
#define NPF_F_ATM_RDI_CLEARED_ENABLE                 (1 << 3)
#define NPF_F_ATM_LOC_RAISED_ENABLE                  (1 << 4)
#define NPF_F_ATM_LOC_CLEARED_ENABLE                 (1 << 5)
#define NPF_F_ATM_PM_STATISTICS_ENABLE               (1 << 6)
#define NPF_F_ATM_CC_ACT_DEACT_REQ_INDICATION_ENABLE (1 << 7)
#define NPF_F_ATM_PM_ACT_DEACT_REQ_INDICATION_ENABLE (1 << 8)
#define NPF_F_ATM_CC_RECEIVED_ENABLE                 (1 << 9)
#define NPF_F_ATM_FPM_RECEIVED_ENABLE                (1 << 10)
#define NPF_F_ATM_BR_RECEIVED_ENABLE                 (1 << 11)
#define NPF_F_ATM_LB_RECEIVED_ENABLE                 (1 << 12)
#define NPF_F_ATM_AIS_RECEIVED_ENABLE                (1 << 13)
#define NPF_F_ATM_RDI_RECEIVED_ENABLE                (1 << 14)
#define NPF_F_ATM_PM_ACT_DEACT_RX_ENABLE             (1 << 15)
#define NPF_F_ATM_CC_ACT_DEACT_RX_ENABLE             (1 << 16)
#define NPF_F_ATM_TM_HIT_WARN_THRESH_ENABLE          (1 << 17)
#define NPF_F_ATM_TM_HIT_DROP_THRESH_ENABLE          (1 << 18)
#define NPF_F_AAL2_TM_HIT_DROP_THRESH_ENABLE         (1 << 19)
#define NPF_F_ATM_CMGR_EV_LAST                       (1 << 20)
/*
 *   ATM Configuration Manager Event reporting data type
 *   This structure represents a single event in an event array. The type
 *   field indicates the specific event in the union.
 */
typedef struct {
    NPF_F_ATM_ConfigMgr_Event_t     eventType;   /* Type of event reported */
      union {
          /* Fault mgmt events */
          NPF_F_ATM_ConfigMgr_OAM_FM_EventData_t       fm;
          /* Perf. Mgmt events */
          NPF_F_ATM_ConfigMgr_OAM_PM_EventData_t       pm;
          /* Act/deact events  */
          NPF_F_ATM_ConfigMgr_OAM_ActDeact_EventData_t actDeact;
          /* Passive Monitoring events */
          NPF_F_ATM_ConfigMgr_OAM_Mon_EventData_t      mon;
          /* Traffic Management events */
          NPF_F_ATM_ConfigMgr_TrafMgmt_EventData_t     traf;
    } u;
} NPF_F_ATM_ConfigMgr_EventData_t;


/* Functional API (FAPI) */
/*******************************************************
 * Registration/De-Registration Functions               *
 *******************************************************/
```

```
/* Completion Callback Function */
typedef void (*NPF_F_ATM_ConfigMgr_CallBackFunc_t)(
    NPF_IN NPF_userContext_t userContext,
    NPF_IN NPF_correlator_t  correlator,
    NPF_IN NPF_F_ATM_ConfigMgr_CallbackData_t  data);

/* Completion Callback Registration Function */
NPF_error_t NPF_F_ATM_ConfigMgr_Register(
    NPF_IN  NPF_userContext_t userContext,
    NPF_IN  NPF_F_ATM_ConfigMgr_CallBackFunc_t callbackFunc,
    NPF_OUT NPF_callbackHandle_t *callbackHandle);

/* Completion Callback Deregistration Function */
NPF_error_t NPF_F_ATM_ConfigMgr_Deregister(
    NPF_IN NPF_callbackHandle_t callbackHandle);

/* Event Handler Function */
typedef void (*NPF_F_ATM_ConfigMgr_EventCallFunc_t) (
    NPF_IN NPF_userContext_t atmUserContext,
    NPF_IN NPF_uint32_t nEvent,
    NPF_IN NPF_F_ATM_ConfigMgr_EventData_t *atmEventArray);

/* Event Registration Function */
NPF_error_t NPF_F_ATM_ConfigMgr_EventHandler_Register(
    NPF_IN NPF_userContext_t atmUserContext,
    NPF_IN NPF_F_ATM_ConfigMgr_EventCallFunc_t atmEvtCallFn,
    NPF_IN NPF_eventMask_t atmEvtMask,
    NPF_OUT NPF_callbackHandle_t *atmEvtCallHdl);

/* Event Handler Deregistration Function */
NPF_error_t NPF_F_ATM_ConfigMgr_EventHandler_Deregister(
    NPF_IN NPF_callbackHandle_t atmEventCallHandle);

/*******************************************************
 * ATM Interface configuration functions
 *******************************************************/

/* Add or Modify an ATM Interface */
NPF_error_t NPF_F_ATM_ConfigMgr_IfSet(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_uint32_t         numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_IfCfg_t   *cfgArray);

/* Delete an ATM Interface */
NPF_error_t NPF_F_ATM_ConfigMgr_IfDelete(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_boolean_t        delContainedObjs,
    NPF_IN NPF_uint32_t         numEntries,
    NPF_IN NPF_F_ATM_IfID_t     *delArray);

/* Read ATM Interface Statistics */
```

```
NPF_error_t NPF_F_ATM_ConfigMgr_IfStatsGet(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_uint32_t         numEntries,
    NPF_IN NPF_F_ATM_IfID_t     *ifArray);

/* Enable statistics collection on an ATM Interface */
NPF_error_t NPF_F_ATM_ConfigMgr_IfStatsEnable(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_uint32_t         numEntries,
    NPF_IN NPF_F_ATM_IfID_t     *ifIDArr);

/* Disable statistics collection on an ATM Interface */
NPF_error_t NPF_F_ATM_ConfigMgr_IfStatsDisable(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_uint32_t         numEntries,
    NPF_IN NPF_F_ATM_IfID_t     *ifIDArr);

/* Query ATM Interface */
NPF_error_t NPF_F_ATM_ConfigMgr_IfQuery(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_uint32_t         numEntries,
    NPF_IN NPF_F_ATM_IfID_t     *ifArray);

/* Set AIS state for an ATM Interface */
NPF_error_t NPF_F_ATM_ConfigMgr_IfAISStateSet(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_F_ATM_IfID_t     ifID,
    NPF_IN NPF_uint8_t          defectType,
    NPF_IN NPF_uint8_t          defectLocation[16]);

/* Clear AIS state for an ATM Interface */
NPF_error_t NPF_F_ATM_ConfigMgr_IfAISStateClear(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_F_ATM_IfID_t     ifID);
```

```
/*******************************************************
 * Functions to operate on ATM VP/VC link
 *******************************************************/

/* add or Modify an ATM Virtual Channel */
NPF_error_t NPF_F_ATM_ConfigMgr_VcSet(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_uint32_t         numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_Vc_t  *cfgArray);

/* Add or Modify an ATM Virtual Path */
NPF_error_t NPF_F_ATM_ConfigMgr_VpSet(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_uint32_t         numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_Vp_t *cfgArray);

/* Bind a higher layer Interface to an ATM Virtual Connection */
NPF_error_t NPF_F_ATM_ConfigMgr_VcBind(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_IfHandle_t       ifChildHandle,
    NPF_IN NPF_uint32_t         numEntries,
    NPF_IN NPF_F_ATM_VirtLinkID_t *bindArray);

/* Unbind a higher layer Interface from an ATM Virtual Connection */
NPF_error_t NPF_F_ATM_ConfigMgr_VcUnbind(
    NPF_IN NPF_callbackHandle_t cbHandle,
    NPF_IN NPF_correlator_t     cbCorrelator,
    NPF_IN NPF_errorReporting_t errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t        blockId,
    NPF_IN NPF_uint32_t         numEntries,
    NPF_IN NPF_F_ATM_VirtLinkID_t *unbindArray);

/* Delete an ATM Virtual Link */
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkDelete(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_boolean_t         delXc,
    NPF_IN NPF_boolean_t         delContainedObjs,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t  *delArray);

/* Enable a Virtual Link */
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkEnable(
```

```
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t  *enaArray);

/* Disable a Virtual Link */
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkDisable(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t  *disArray);

/* Get Operational Status of a Virtual Link */
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkOperStatusGet(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t  *linkArray);

/* Enable Statistics collection on a Virtual Link */
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkStatsEnable(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t  *enaArray);

/* Disable Statistics collection on a Virtual Link */
NPF_error_t NPF_F_ATM_ConfigMgr_VirtLinkStatsDisable(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLink_t  *disArray);

/* Read ATM Virtual Channel Statistics */
NPF_error_t NPF_F_ATM_ConfigMgr_VcStatsGet(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLinkID_t *linkArray);

/* Query ATM Virtual Channel Configuration */
```

```
NPF_error_t NPF_F_ATM_ConfigMgr_VcQuery(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLinkID_t *linkArray);

/* Add VC link Cross connect */
NPF_error_t NPF_F_ATM_ConfigMgr_VcLinkXcSet (
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_VcLinkXc_t *vcLinkXc);

/* Delete VC link Cross connect */
NPF_error_t NPF_F_ATM_ConfigMgr_VcLinkXcDelete (
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_VcXcId_t        *vcLinkXc);

/* Query Virtual Channel Cross connect */
NPF_error_t NPF_F_ATM_ConfigMgr_VcLinkXcQuery (
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_VcXcId_t        *linkXc);

/* Read ATM Virtual Path Statistics */
NPF_error_t NPF_F_ATM_ConfigMgr_VpStatsGet(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLinkID_t *linkArray);

/* Query ATM Virtual Path Link Configuration */
NPF_error_t NPF_F_ATM_ConfigMgr_VpQuery(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLinkID_t *linkArray);
```

```
/* Add Virtual Path Link Cross connect */
NPF_error_t NPF_F_ATM_ConfigMgr_VpLinkXcSet (
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_VpLinkXc_t *vpLinkXc);

/* Delete Virtual Path Link Cross connect */
NPF_error_t NPF_F_ATM_ConfigMgr_VpLinkXcDelete (
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_VpXcId_t        *vpLinkXc);

/* Query Virtual Path Cross connect */
NPF_error_t NPF_F_ATM_ConfigMgr_VpLinkXcQuery (
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_VpXcId_t        *linkXc);


/*******************************************************
 * Functions to operate on ATM AAL2 channels
 *******************************************************/
/* Add or Modify an AAL2 Channel */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelSet(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_AAL2_Channel_t    *chnlArray);

/* Bind a higher layer Interface to an AAL2 Channel Syntax */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelBind(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_IfHandle_t        ifChildHandle,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChanId_t    *bindArray);

/* Unbind a higher layer Interface from an AAL2 channel */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelUnbind(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
```

```
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_VirtLinkID_t *unbindArray);

/* Delete an AAL2 Channel */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelDelete(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_boolean_t         delXc,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChanId_t    *chnlArray);

/* Query AAL2 Channel Statistics */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsGet(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChanId_t    *chnlArray);

/* Query AAL2 Channel */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelQuery(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChanId_t    *chnlArray);

/* Enable an AAL2 channel */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelEnable(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChanId_t    *enaArray);

/* Disable an AAL2 channel */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelDisable(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChanId_t    *disArray);

/* Get Operational Status of a AAL2 channel */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelOperStatusGet(
```

```
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChanId_t    *chnlArray);

/* Enable Statistics collection on a AAL2 channel */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsEnable(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChanId_t    *chnlArray);

/* Disable Statistics collection on a AAL2 channel */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChannelStatsDisable(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChanId_t    *chnlArray);

/* Add AAL2 Channel Cross connect */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChnlXcSet (
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_AAL2_ChnlXc_t    *chnlXc);

/* Delete AAL2 Channel Cross connect */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChnlXcDelete (
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChnlXcId_t  *chnlXc);

/* Query AAL2 Channel Cross connect */
NPF_error_t NPF_F_ATM_ConfigMgr_AAL2_ChnlXcQuery (
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_AAL2ChnlXcId_t  *chnlXc);

/*******************************************************
```

```
 * Functions for OAM operations on VP/VC links
 ****************************************************/
/* Configure/Modify OAM attributes of a Connection Point */
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_CP_Set(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_CP_t *oamCPArray);

/* Activate, Deactivate, Start and Stop OAM Continuity Check */
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_CC_Set(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_CC_t *oamCCArray);

/* Response to OAM Continuity Check Indication */
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_CC_Rsp(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_CC_Rsp_t *oamCCArray);

/* Activate, Deactivate, Start and Stop OAM Performance Management */
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_PM_Set(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_PM_t *oamPMArray);

/* Response to OAM Performance Management Indication */
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_PM_Rsp(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_PM_Rsp_t *oamPMArray);

/* Query Performance Monitoring Statistics for an OAM flow */
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_PM_StatsGet(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
```

```
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_PM_StatsReq_t *oamPMStatsReqArray);

/* Initiate an OAM Loopback Procedure */
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_LB_Set(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_LB_t *oamLBArray);

/* Monitor OAM Cells */
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_Mon_Set(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_Mon_t *oamMonArray);

/* Set AIS Alarm State */
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_Alarm_Set (
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_Alarm_t *oamAlarmArray);

/* Clear AIS Alarm State */
NPF_error_t NPF_F_ATM_ConfigMgr_OAM_Alarm_Clear(
    NPF_IN NPF_callbackHandle_t  cbHandle,
    NPF_IN NPF_correlator_t      cbCorrelator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t        feHandle,
    NPF_IN NPF_BlockId_t         blockId,
    NPF_IN NPF_uint32_t          numEntries,
    NPF_IN NPF_F_ATM_ConfigMgr_OAM_Alarm_t *oamAlarmArray);

#ifdef __cplusplus
}
#endif

#endif /* __NPF_F_ATM_CONFIGURATION_MANAGER_H__ */
```

APPENDIX B ACKNOWLEDGEMENTS

Working Group Chair: Alex Conta

Task Group Chair: Per Wollbrand

The following individuals are acknowledged for their participation to ATM Task Group 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:

Pål Dammvik, Ericsson
Patrik Herneld, Ericsson
Jaroslaw Kogut, Intel
Arthur Mackay, Freescale
Stephen Nadas, Ericsson
Michael Persson, Ericsson
John Renwick, Agere Systems
Vedvyas Shanbhogue (ed.), Intel
Michael Speer, Sun Microsystems
Keith Williamson, Motorola
Weislaw Wisniewski, Intel
Per Wollbrand, Ericsson

APPENDIX C LIST OF COMPANIES BELONGING TO NPF DURING APPROVAL PROCESS

Agere Systems	Hifn	NTT Electronics
Altera	IBM	PMC Sierra
AMCC	IDT	Seaway Networks
Analog Devices	Infineon Technologies AG	Sensory Networks
Avici Systems	Intel	Sun Microsystems
Cypress Semiconductor	IP Fabrics	Teja Technologies
Enigma Semiconductor	IP Infusion	TranSwitch
Ericsson	Kawasaki LSI	U4EA Group
Erlang Technologies	Motorola	Wintegra
EZChip	NetLogic	Xelerated
Flextronics	Nokia	Xilinx
HCL Technologies	Nortel Networks	ZNYX Networks