Inverse Multiplexing for ATM (IMA) LFB and Functional API

September 7, 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	09/06/2005	Rev 1.0 of the Inverse Multiplexing for ATM (IMA) LFB and Functional API Implementation Agreement. Source : npf2004.325.13

2 Introduction

This IA defines the IMA LFB and its functional API. The IA also defines the inputs and outputs for the IMA LFB and the metadata generated and consumed by the IMA LFB.

2.1 Acronyms

- **ATM:** Asynchronous Transfer Mode
- **API:** Application Program Interface
- **CTC:** Common Transmit Clock configuration
- **FE:** Far End
- **ICP cell:** IMA Control Protocol Cell
- **ID:** Identifier
- **IMA:** Inverse Multiplexing for ATM
- **ITC:** Independent Transmit Clock configuration
- **LCD:** Loss of cell delineation defect
- **LDS:** Link Delay Synchronization
- **LFB:** Logical Functional Block
- **LID:** Link Identifier
- **LIF:** Loss of IMA frame defect
- **LODS:** Link out of delay synchronization defect
- **LOF:** Loss of frame
- **LOS:** Loss of Signal
- **LSM:** Link State Machine
- **NE:** Near End
- **NNI:** Network Node Interface
- **OIF:** Out of IMA frame anomaly
- **PDH:** Plesiochronous Digital Hierarchy
- **PMD:** Physical Media Dependent
- **RDI:** Remote Defect Indication
- **RFI:** Remote Failure Indicator
- **SES:** Severely errored seconds
- **TC:** Transmission Convergence
- **UAS:** Unavailable seconds
- **UNI:** User Network Interface

2.2 Assumptions

The ATM TC LFB shall provide suitable configurations to cater to the requirements (R-3) and (R-4) on the Transmission Convergence sublayer specified in af-phy-0086.001 for the links to be used in an IMA group

2.3 Scope

This IA describes the functional API provided by the IMA LFB for configuring IMA interfaces in the forwarding element. The IA also specifies the metadata generated and consumed by this LFB.

2.4 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).
- This document depends on Software API Conventions Implementation agreement Revision 2.0 for the 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_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 the 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
- This document depends on the Interface Management API Implementation Agreement (ATM Interfaces) for the below data types:
    - `NPF_IfATM_IMA_Symmetry_t`
    - `NPF_IfATM_IMA_Tclock_t`
    - `NPF_IfATM_IMA_FrameLength_t`
    - `NPF_IfATM_IMA_Ver_t`
- ATM Software API Architecture Framework Implementation Agreement Revision 1.0 defines the architectural framework for the ATM FAPIs.
- This document depends on the ATM Configuration Manager Functional API Implementation Agreement Revision 1.0 for the below type definitions:
    - `NPF_F_ATM_Timers_t`
    - `NPF_ObjStatus_t`

3 IMA LFB Description

The IMA LFB performs multiplexing and de-multiplexing of ATM cells in a cyclical fashion among links of an IMA group to form a higher bandwidth logical link whose rate is approximately the sum of the link rates. In the transmit direction, the ATM cell stream received from the ATM layer is distributed on a cell by cell basis, across the multiple links within the IMA group. In the receive direction, the IMA LFB recombines the cells from each link, on a cell by cell basis, recreating the original ATM cell stream. The aggregate cell stream is then passed to the ATM layer.

The IMA LFB periodically transmits ICP cells that contain information that permit reconstruction of the ATM cell stream at the receiving end of the IMA virtual link. At the receive end, the IMA LFB reconstructs the ATM cell stream after accounting for the link differential delays, smoothing CDV introduced by the control cells, etc. The IMA LFB also transmits filler cells to maintain a continuous stream of cells at the physical layer when there are no ATM layer cells to be sent. The filler cells received by the IMA LFB are discarded. The IMA LFB is modeled as shown in Figure 3.1:

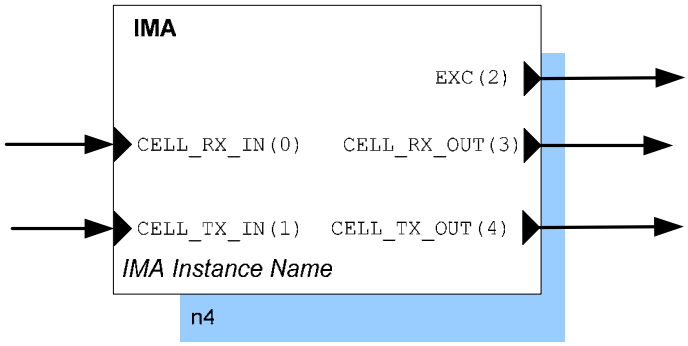

Figure 3.1: ATM IMA LFB

The IMA LFB receives ATM cells received over the physical interface from the ATM TC LFB through the `CELL_RX_IN` input. The IMA LFB reconstructs the ATM cell stream and sends the ATM cells over the `CELL_RX_OUT` output to the ATM Header Classifier LFB.

The IMA LFB receives ATM cells for transmission over the IMA group from the ATM Header Generator LFB over the `CELL_TX_IN` input. The IMA LFB distributes the ATM cells over the links constituting the IMA group and sends the cells over the `CELL_TX_OUT` output to the ATM TC LFB. The IMA LFB also sends ICP and filler cells for transmission on the IMA links through the `CELL_TX_OUT` output.

The LFB may contain multiple instances of IMA links identified by unique interface Ids the corresponding PDH links. The LFB may contain multiple instances of IMA groups identified by unique interface Ids of the IMA interface or group. The term IMA interface and IMA group are used interchangeably in this IA. One or more (upto 32) PDH interfaces form the parent interfaces for the IMA interface. The transmission convergence function for the IMA links are performed by the interface specific ATM TC LFB and the associated PMD sublayer functions.

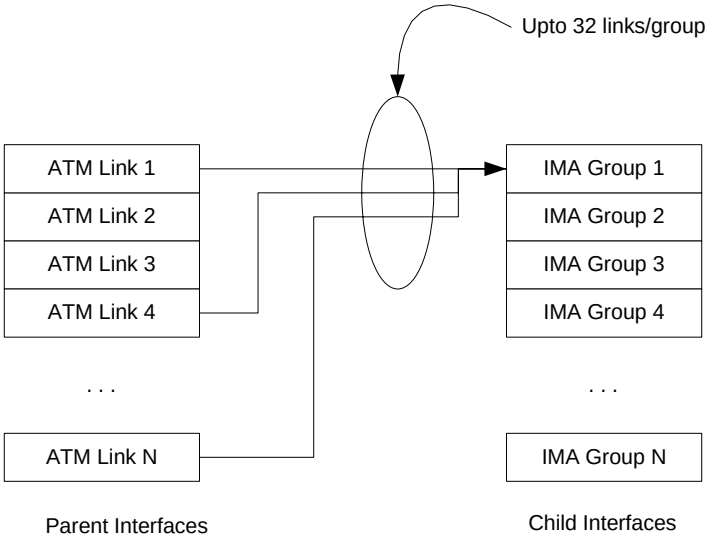

Figure 3.2: IMA Links and Groups

The IMA links associated with an IMA group may be used by the IMA LFB for receive, transmit, or both based on the group symmetry mode configured for the IMA group.

3.1 IMA LFB Inputs

Table 3-1 IMA LFB Inputs

Symbolic Name	Input ID	Description
`CELL_RX_IN`	0	This input is used to receive the ATM cells from the physical layer. Cells received over the IMA links from the physical layer are used to construct the ATM cell stream to be sent to the ATM Header Classifier LFB.
`CELL_TX_IN`	1	This input is used to receive ATM cells for transmission over an IMA group from the ATM Header Generator LFB. The ATM cell stream received for transmission over the IMA group is distributed in a cyclic manner among the constituent IMA links of the group.

3.1.1 Metadata Required

The IMA LFB expects the below metadata on the `CELL_RX_IN` input.

Table 3-2 Input Metadata for CELL_RX_IN input of IMA LFB

Metadata tag	Access method	Description
`META_IF_ID`	Read-And-Consume	Metadata identifying the interface ID of the parent PDH interface on which the ATM cell was received.

The IMA LFB expects the below metadata on the `CELL_TX_IN` input.

Table 3-3 Input Metadata for CELL_TX_IN input of IMA LFB

Metadata tag	Access method	Description
META_IF_ID	Read-And-Consume	Metadata identifying the interface ID of the IMA interface on which the ATM cell is to be transmitted.

3.2 IMA LFB Outputs

Table 3-4 IMA LFB Outputs

Symbolic Name	Output ID	Description
CELL_RX_OUT	1	This is the output on which the ATM cell stream extracted over from links forming the IMA group is sent to the ATM Header Classifier LFB
CELL_TX_OUT	2	This output is used to send the ATM cells to the ATM TC LFBs for transmission over the IMA links.
EXC	3	This output is used to send ATM cells which need to be discarded due to errors.

3.2.1 Metadata Produced on CELL_RX_OUT output

The metadata produced on this output is as below

Table 3-5 Output Metadata IMA LFB on CELL_RX_OUT output

Metadata tag	Access method	Description
META_IF_ID	Write	Metadata identifying the interface ID of the IMA group on which the cell was received.

3.2.2 Metadata Produced on CELL_TX_OUT output

Table 3-6 Output Metadata IMA LFB on CELL_TX_OUT output

Metadata tag	Access method	Description
META_IF_ID	Write	Metadata identifying the interface ID of the ATM link on which the cell is to be transmitted

3.3 Accepted Cell Types

The IMA LFB can be used on send and receive ATM cells over either UNI or NNI interfaces.

3.4 Cell Modifications

- The ICP and filler cells received over the IMA links are extracted by the IMA LFB in the receive direction.
- The IMA LFB will generate ICP and filler cells as required on the IMA links in the transmit direction.
- The ATM layer cells received from ATM TC Receive LFB are passed without any modification or re-ordering to the ATM Header Classifier LFB.
- The ATM layer cells received from the ATM Header generator LFB are passed without any modification or re-ordering to the ATM TC Transmit LFB for transmission on the IMA links.

3.5 Relationship with Other LFBs

The IMA LFB interacts with the ATM TC Receive LFB, ATM TC Transmit LFB, ATM Header Classifier LFB and the ATM Header Generator LFB as shown in Figure 3.3.

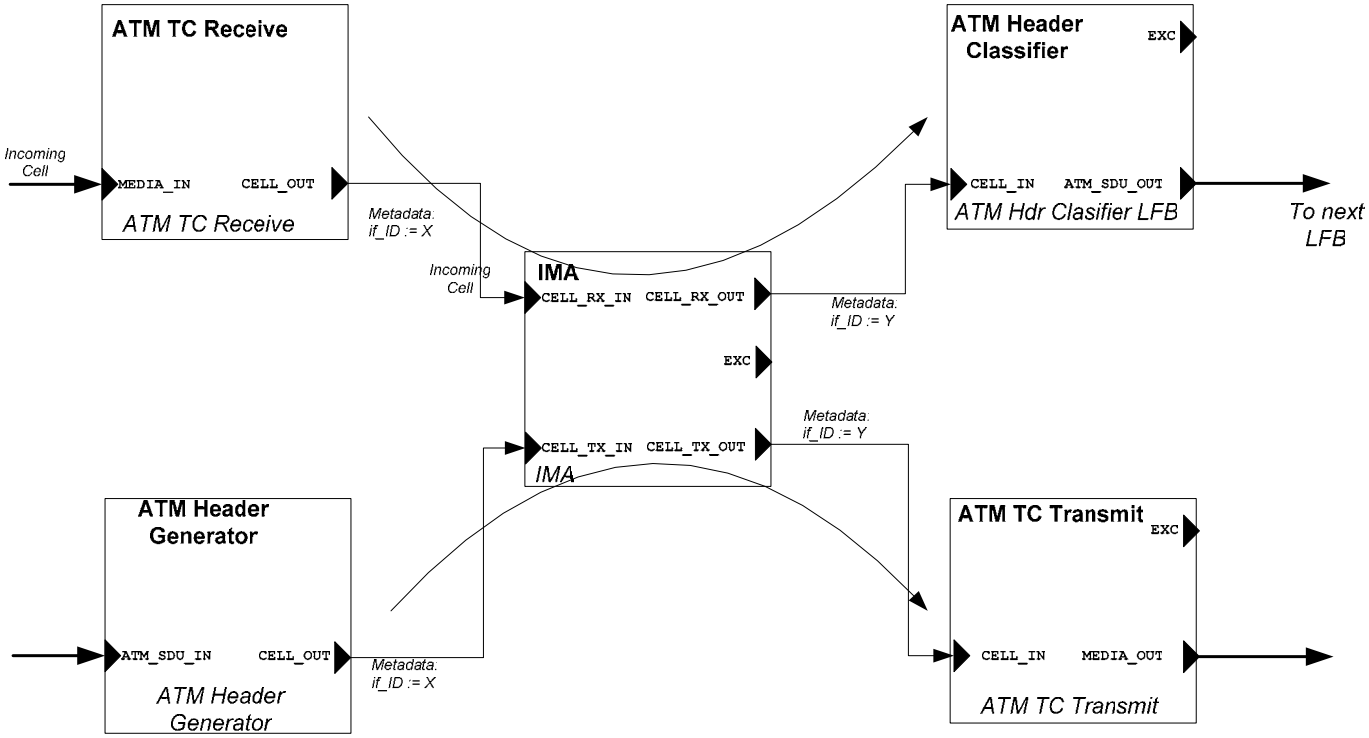

Figure 3.3: IMA LFB Interfaces

The EXC output of the IMA LFB could be connected an LFB that receives cells for which could not be processed due to errors. Depending on system design this may be either the dropper LFB or any other LFB that makes a decision on how to utilize such cells.

4 Data Types

4.1 Common LFB Data Types

4.1.1 LFB Type Code

It is possible to use the FAPI Topology Discovery APIs to discover an IMA LFB in a forwarding element using a block type value for the IMA LFB.

```
/* LFB type for IMA LFB */
#define NPF_F_IMA_LFB_TYPE    46
```

4.1.2 IMA Configurations

4.1.2.1 IMA Group ID

This section defines the IMA group identifier that is used to uniquely identify an IMA group. Any restrictions placed on the range or values that can be assigned to the IMA group ID are outside the scope of NPF.

```
typedef NPF_uint32_t NPF_F_IMA_Group_ID_t;        /* IMA group ID   */
```

4.1.2.2 IMA Link ID

This section defines the IMA link identifier that is used to uniquely identify an IMA link. Any restrictions placed on the range or values that can be assigned to the IMA link ID are outside the scope of NPF.

```
typedef NPF_uint32_t NPF_F_IMA_Link_ID_t;        /* IMA link ID   */
```

4.1.2.3 IMA Link Status

This structure defines the near end or far end states for receive and transmit Link State Machine.

```
typedef enum {
   /* Link not configured */
   NPF_F_IMA_LSM_STATE_NOT_IN_GROUP = 1,

   /* Link configured but cannot be used */
   NPF_F_IMA_LSM_STATE_UNUSABLE_UNKNOWN = 2,
   NPF_F_IMA_LSM_STATE_UNUSABLE_FAULT_LINK_DEFECT = 3,
   NPF_F_IMA_LSM_STATE_UNUSABLE_FAULT_LIF = 3,
   NPF_F_IMA_LSM_STATE_UNUSABLE_FAULT_LODS = 3,

   NPF_F_IMA_LSM_STATE_UNUSABLE_MISCONNECTED = 4,
   NPF_F_IMA_LSM_STATE_UNUSABLE_INHIBITED = 5,
   NPF_F_IMA_LSM_STATE_UNUSABLE_FAILED = 6,

   /* Link is ready to use */
   NPF_F_IMA_LSM_STATE_USABLE = 7,

   /* Link is active and capable of passing cells to/from ATM Layer */
   NPF_F_IMA_LSM_STATE_ACTIVE = 8
} NPF_F_IMA_LSM_State_t;
```

4.1.2.4 IMA Group State Machine states

This enumeration defines Group State Machine State.

```
typedef enum {
   NPF_F_IMA_GSM_NOT_CONFIGURED = 1,
   NPF_F_IMA_GSM_START_UP = 2,
   NPF_F_IMA_GSM_START_UP_ACK = 3,
   NPF_F_IMA_GSM_CONFIG_ABORTED_UNSUPPORTED_FRAME_LEN = 4,
```

```
   NPF_F_IMA_GSM_CONFIG_ABORTED_INCOMPATIBLE_SYMMETRY = 5,
   NPF_F_IMA_GSM_CONFIG_ABORTED_UNSUPPORTED_IMA_VERSION = 6,
   NPF_F_IMA_GSM_CONFIG_ABORTED_OTHER = 7,
   NPF_F_IMA_GSM_INSUFFICIENT_LINKS = 8,
   NPF_F_IMA_GSM_BLOCKED = 9,
   NPF_F_IMA_GSM_OPERATIONAL = 10
} NPF_F_IMA_GSM_State_t;
```

4.1.2.5 IMA Group Traffic Machine State

This enumerator defines Group Traffic State Machine State.

```
typedef enum {
   NPF_F_IMA_GTSM_DOWN = 0,
   NPF_F_IMA_GTSM_UP
} NPF_F_IMA_GTSM_State_t;
```

4.1.2.6 IMA Group Testing Mode

This structure is used to configure the testing link in the group. When the test link ID is configured as -1 or the test pattern is set to -1, the LFB is free to choose a link for testing and the test pattern to be used. On configuration of the group, the test procedure is disabled by default and should be enabled by the FAPI client if required by specifying the test link ID to be used for the procedure. The algorithm used to select such a link and the test pattern to be used is outside the scope of NPF.

```
typedef enum {
    NPF_F_TEST_PROC_DISABLED = 1,
    NPF_F_TEST_PROC_OPERATIONAL = 2,
} NPF_F_IMA_Test_Proc_Status_t;

typedef struct {
   /* Testing link ID */
   NPF_int8_t  testLID;

   /* Test pattern */
   NPF_int32_t  testPattern;

   /* Test Procedure Status */
   NPF_F_IMA_Test_Proc_Status_t testStatus;

   /* Test verification Duration. The far end is expected to loop back the
    * test pattern on all links in the group within this duration. Failing
    * which the end initiating the test procedure will declare a test
    * procedure failure on the links on which the test pattern was not
    * loopback.
    */
   NPF_F_ATM_Timers_t expRespDuration;
} NPF_F_IMA_Group_Test_Mode_t;

typedef struct {
   /* A unique ID to identify the group */
   NPF_F_IMA_Group_ID_t        groupID;

   /* Start/Stop/Change pattern */
   NPF_F_IMA_Group_Test_Mode_t groupTestMode;
} NPF_F_IMA_Group_Test_Proc_Config_t;
```

4.1.2.7 IMA Group Configuration

The below structure contains the configuration parameters for an IMA group.

```
typedef struct {
```

```
/* The Interface handle of the IMA group */
NPF_IfHandle_t              imaIfID;

/* A unique ID to identify the group. The interface handle for the group
 * is an arbitrary value assigned by the IM APIs. The groupID may be used
 * to provide a fast way to lookup the group information. The FAPI
 * implementations may restrict the range of values assigned to this field
 * or may impose restrictions on the way this number is constructed and
 * any such restrictions are outside the scope of NPF. This field is not
 * the IMA ID sent in the ICP cells.
 */
NPF_F_IMA_Group_ID_t        groupID;

/* IMA protocol version – 1_0 or 1_1. Refer section 2.1.2.2 of Interface
 * Management API Implementation Agreement (ATM Interfaces) Revision 3.0
 * for type definition
 */
NPF_IfATM_IMA_Ver_t         imaVer;

/* Minimum number of active receive links to make group operational */
NPF_uint8_t                 minNumRxLinks;

/* Minimum number of active transmit links to make group operational */
NPF_uint8_t                 minNumTxLinks;

/* Expected bandwidth in bits per second of the links which may be
 * added to this group. If configured as 0, it indicates that the FAPI
 * implementation may derive this from the first link that is added to the
 * group
 */
NPF_uint32_t                expLinkRate;

/* IMA Group Symmetry Mode. Refer section 2.1.2.2 of Interface Management
 * API Implementation Agreement (ATM Interfaces) Revision 3.0
 * for type definition */
NPF_IfATM_IMA_Symmetry_t    symmetry;

/* Transmit clocking mode – CTC/ITC. Refer section 2.1.2.4 of Interface
 * Management API Implementation Agreement (ATM Interfaces) Revision 3.0
 * for type definition
 */
NPF_IfATM_IMA_Tclock_t      neTxClockMode;

/* Link ID of the default transmit timing reference link
 * The Tx reference link ID specified below is used as a hint by the
 * FAPI implementation to choose the TX timing reference link. If the link
 * link corresponding to the LID hinted below is available, it is selected
 * as the timing reference link. A value of -1 specifies that no hint is
 * being provided by the FAPI client to the FAPI implementation and the
 * LFB/FAPI are free to choose a suitable link as the timing reference
 * link */
NPF_int8_t                  defTxTimingRefLinkLID;


/* IMA ID configured for the near end */
NPF_uint8_t                 txImaID;

/* Frame length to use in transmit direction. Refer section 2.1.2.3 of
 * Interface Management API Implementation Agreement (ATM Interfaces)
```

```
    * Revision 3.0 for type definition */
   NPF_IfATM_IMA_FrameLength_t   txFrameLength;

   /* Maximum tolerated differential delay in milliseconds. Refer section
    * 4.1.16 of ATM Configuration Manager Functional API (Work in progress)
    * for type definition
    */
   NPF_F_ATM_Timers_t          diffDelayMax;

   /* Alpha value to be used by IFSM */
   NPF_F_IMA_AlphaValue_t      alphaValue;

   /* Beta value to be used by the IFSM */
   NPF_F_IMA_BetaValue_t       betaValue;

   /* Gamma value to be used by the IFSM */
   NPF_F_IMA_GammaValue_t      gammaValue;

   /* Administrative status of the group – UP/DOWN. Refer section
    * 4.1.17 of ATM Configuration Manager Functional API (Work in progress)
    * for type definition
    */
   NPF_ObjStatus_t             adminStatus;

   /* Configuration for test procedure */
   NPF_F_IMA_Group_Test_Mode_t testMode;

} NPF_F_IMA_Group_Config_t;
```

4.1.2.8 IMA Group States

The structure returns the state of the group and group traffic state machines for this group.

```
typedef struct {
   /* Status of the group state machines for this group */
   NPF_F_IMA_GSM_State_t       neGroupState;
   NPF_F_IMA_GSM_State_t       feGroupState;

   /* Status of the group traffic state machine for this group */
   NPF_F_IMA_GTSM_State_t      gtsmState;

   /* Whether test procedure disabled or operational */
   NPF_F_IMA_Test_Proc_Status_t testProcStatus;

   /* Status of the test (if operational) – failed/passed */
   NPF_boolean_t              testProcFailed;   /* TRUE/FALSE */

} NPF_F_IMA_Group_State_t;
```

4.1.2.9 IMA Group Query Information

This structure defines the information returned when an IMA group is queried. The structure returns the group configuration information as well as the status of the various state machines for this group.

```
typedef struct {
   /* IMA group configuration */
   NPF_F_IMA_Group_Config_t    neGroupConfig;

   /* Status of the state machines for this group */
   NPF_F_IMA_Group_State_t     gsmGtsmState;

   /* FE Transmit clocking mode – CTC/ITC. Refer section 2.1.2.4 of Interface
```

```
 * Management API Implementation Agreement (ATM Interfaces) Revision 3.0
 * for type definition
 */
NPF_IfATM_IMA_Tclock_t      feTxClockMode;

/* IMA ID configured for the far end */
NPF_uint8_t                 rxImaID;

/* Frame length used in receive direction. Refer section 2.1.2.2 of
 * Interface Management  API Implementation Agreement (ATM Interfaces)
 * Revision 3.0 for type definition
 */
NPF_IfATM_IMA_FrameLength_t   rxFrameLength;

/* ID of the link in group with least delay */
NPF_F_IMA_Link_ID_t         leastDelayLinkID;

/* Link ID of the current transmit timing reference link */
NPF_F_IMA_Link_ID_t         curTxTimingRefLinkLID;

/* Link ID of the current receive timing reference link */
NPF_F_IMA_Link_ID_t         curRxTimingRefLinkLID;

/* OAM label being Tx – identifies version negotiated/configured */
 NPF_uint8_t               txOamLabel;

/* OAM label being Rx – identifies version negotiated/configured */
 NPF_uint8_t               rxOamLabel;

/* Available cell rate (cells per second) in transmit direction */
 NPF_uint32_t              txAvailCellRate;

/* Available cell rate (cells per second)in receive direction */
 NPF_uint32_t              rxAvailCellRate;

/* Test procedure status. This field if valid only if the test procedure
 * is operation on link in this group. When set to NPF_TRUE it indicates
 * that the test procedure failed and the bit map of links on which the
 * test pattern failed to loop back is specified in the testResultBitMap
 * field.
 */
 NPF_boolean_t             testProcFailed;

/* Bit map indicating the links on which the test pattern failed to loop
 * back. Valid only if the test procedure is operation on this group
 */
 NPF_uint32_t              testResultBitMap;

/* Number of configured RX links */
NPF_uint8_t                 numRxCfgLinks;

/* Array of link Ids of Rx links configured for this group */
NPF_F_IMA_Link_ID_t        *rxCfgLinkArr;

/* Number of configured TX links */
NPF_uint8_t                 numTxCfgLinks;

/* Array of link Ids of Tx links configured for this group */
NPF_F_IMA_Link_ID_t        *txCfgLinkArr;
```

```
   /* Number of active RX links */
   NPF_uint8_t                numRxActLinks;

   /* Array of link Ids of active Rx links for this group */
   NPF_F_IMA_Link_ID_t        *rxActLinkArr;

   /* Number of active TX links */
   NPF_uint8_t                numTxActLinks;

   /* Array of link Ids of active Tx links for this group */
   NPF_F_IMA_Link_ID_t        *txActLinkArr;
} NPF_F_IMA_Group_Info_t;
```

4.1.2.10 IMA Link Configuration

The below structure contains the configuration parameters for a link in an IMA group.

```
typedef struct {
   /* The Interface handle of the PDH Link */
   NPF_IfHandle_t             imaIfID;

   /* A unique ID to identify the link. The interface handle for the link
    * is an arbitrary value assigned by the IM APIs. The linkID may be used
    * to provide a fast way to lookup the link information. The FAPI
    * implementations say restrict the range of values assigned to this field
    * or restrictions on the manner in which this number is constructed and
    * any such restrictions are outside the scope of NPF.
    * This number is not the logical link ID of the link.
    */
   NPF_F_IMA_Link_ID_t        linkID;

   /* Group to which the link is assigned. Value 0 indicate not in a group */
   NPF_F_IMA_Group_ID_t       groupID;

   /* Administrative status of the link – UP/DOWN. Refer section
    * 4.1.17 of ATM Configuration Manager Functional API (Work in progress)
    * for type definition
    */
   NPF_ObjStatus_t            adminStatus;

   /* Logical Link ID (LID) used in Transmit direction. A value of -1
    * assigned to the txLinkId indicates that the FAPI implementation is
    * to choose the LID to be assigned to this link */
   NPF_int8_t                 txLinkId;

   /* ICP cell offset for frames sent on this link. The FAPI client may
    * assign a value of -1 to the icpCellOffset indicating that the
    * FAPI implementation is free to choose the ICP cell offset
    * When configured as -1, the FAPI implementation may choose to distribute
    * ICP cells from link to link withing an IMA group in an uniform fashion
    * across the IMA frame. The mechanism used to select the ICP cell offset
    * by FAPI implementation when the icpCellOffset is set to -1 is outside
    * the scope of NPF
    */
   NPF_uint16_t                icpCellOffset;
} NPF_F_IMA_Link_Config_t;
```

4.1.2.11 IMA Link States

The structure returns receive and transmit link state machines states for this link.

```
typedef struct {
   /* near end IMA Rx LSM State */
   NPF_F_IMA_LSM_State_t       neRxLinkState;

   /* near end IMA Tx LSM State */
   NPF_F_IMA_LSM_State_t       neTxLinkState;

   /* far end IMA Rx LSM State */
   NPF_F_IMA_LSM_State_t       feRxLinkState;

   /* far end IMA Tx LSM State */
   NPF_F_IMA_LSM_State_t       feTxLinkState;
} NPF_F_IMA_Link_State_t;
```

4.1.2.12 IMA Link Query Information

This structure defines the information returned when an IMA link is queried. The structure returns the link configuration information as well as the status of the various state machines for this link.

```
typedef struct {
   /* near end IMA link configuration */
   NPF_F_IMA_Link_Config_t     neLinkConfig;

   /* NE/FE Rx and Tx LSM states */
   NPF_F_IMA_Link_State_t      linkStates;

   /* Logical Link ID (LID) in Receive direction. A value of -1 indicates
    * that the LID is not known */
   NPF_int8_t                  rxLinkId;

   /* Differential delay measured between this link and the link within the
    * IMA group with the least delay. Refer section 4.1.16 of ATM
    * Configuration Manager Functional API (Work in progress) for type
    * definition
    */
   NPF_F_ATM_Timers_t       relativeDelay;

} NPF_F_IMA_Link_Info_t;
```

4.1.2.13 IMA Group Statistics

This structure defines the IMA group related statistics information.

```
typedef struct {

   /* Time in seconds for which this group has been in operation state */
    NPF_uint32_t              groupRunningSecs;

   /* Count of one second intervals where the GTSM was unavailable (R136)*/
   NPF_uint32_t        groupUnavailSecs;

   /* Count of near end group failures (R137)*/
   NPF_uint32_t        neNumFailures;

   /* Count of far end group failures (O25)*/
   NPF_uint32_t        feNumFailures;
} NPF_F_IMA_Group_Stats_t;
```

4.1.2.14 IMA Link Statistics

This structure defines the IMA link related statistics information.

```
typedef struct {
   /* Count of errored, missing, invalid ICP except during
      SES-IMA/UAS-IMA (R125)                                */
   NPF_uint32_t imaViolations;

   /* Number of OIF anomalies at near end except during
      SES-IMA/UAS-IMA (O20)                                 */
   NPF_uint32_t oifAnomalies;

   /* Count of 1 sec intervals containing > 30% invalid
      IMA, link defects, LIF, or LODS except during UAS-IMA (R126) */
   NPF_uint32_t neSevErroredSecs;

   /* Count of 1 sec intervals containing RDI-IMA defects
      Except during UAS-IMA-FE condition (R127) */
   NPF_uint32_t feSevErroredSecs;

   /* Count of unavailable seconds at near end (R128) */
   NPF_uint32_t neUnavailSecs;

   /* Count of unavailable seconds at far end (R129) */
   NPF_uint32_t feUnavailSecs;

   /* Count of unusable seconds at near end LSM (R130) */
   NPF_uint32_t neTxUnusableSecs;

   /* Count of unusable seconds at near end LSM (R131) */
   NPF_uint32_t neRxUnusableSecs;

   /* Count of seconds with Tx unusable indications from
     far end Tx LSM (R132) */
   NPF_uint32_t feTxUnusableSecs;

   /* Count of seconds with Rx unusable indications from
     far end Rx LSM (R133) */
   NPF_uint32_t feRxUnusableSecs;

   /* Number of times near end transmit failure alarm
      condition entered (R134)*/
   NPF_uint32_t neTxNumFailures;

   /* Number of times near end receive failure alarm
      condition entered (R135)*/
   NPF_uint32_t neRxNumFailures;

   /* Number of times far end transmit failure alarm
      condition entered (O21)*/
   NPF_uint32_t feTxNumFailures;

   /* Number of times far end receive failure alarm
      condition entered (O22)*/
   NPF_uint32_t feRxNumFailures;

   /* Number of stuff events inserted in tx direction (O-23) */
   NPF_uint32_t txStuffs;
```

```
   /* Number of stuff events detected in rx direction (O-24) */
   NPF_uint32_t rxStuffs;

   /* Flag helping the FAPI user to simplify and make the reporting of
      Unavailable Seconds more efficient at 15 minutes PM intervals. The
      Flag indicates the following.

      1) Link is in Available state and did count SES in the last second
         before the statistic query.

      2) Link is in Unavailability state and did not count SES in the last
         Second before the statistic query.

      3) None of 1 or 2.

      The flag set to 1 indicates that Unavailability state is about to be
      Entered and the flag set to 2 indicates that Unavailability state is
      about to be left.

      In both these cases, the FAPI user must do a new query 10 seconds later
      To secure reporting the correct SES and UAS values. When the flag is
      set to 3, the FAPI user can use the SES and UAS counter values directly
      and does not need to make another query 10 seconds later. */

   NPF_uint32_t uasInfoFlag;
} NPF_F_IMA_Link_Stats_t;
```

4.1.2.15 ICP Query Response Structure

The FAPI client may request the FAPI implementation to provide the last ICP cell seen on any link in an IMA group. The below structure is used to return the ATM SDU of the ATM cell containing the last seen ICP cell on the queried IMA link. If no valid ICP cells have been received on the queried link, the `icpValid` field shall be set to `FALSE`.

```
typedef struct {
   NPF_boolean_t icpValid;     /* Whether ICP cell information valid */
   NPF_uint8_t icp_bytes[48];      /* ICP Cell payload                  */
} NPF_F_IMA_Icp_Cell_t;
```

4.1.2.16 IMA LFB Attributes query response

The attributes of an IMA are the following:

```
typedef struct {
   NPF_uint32_t   maxNumGroups;          /* Maximum possible IMA groups   */
   NPF_uint32_t   curNumGroups;          /* Current number of IMA groups  */
   NPF_uint32_t   maxNumLinks;           /* Maximum possible IMA links    */
   NPF_uint32_t   curNumLinks;           /* Current number of IMA links   */
} NPF_F_IMA_LFB_AttrQueryResponse_t;
```

The `maxNumGroups` field contains the maximum number of IMA groups supported in this IMA LFB. The `curNumGroups` field contains the number of IMA groups currently configured in the LFB. The `maxNumLinks` field contains the maximum number of IMA links supported in this IMA LFB. The `curNumLinks` field contains the number of IMA links currently configured in the LFB.

4.2 Data Structures for Completion Callbacks

4.2.1 Asynchronous Response

The Asynchronous Response data structure is used during callbacks in response to API invocations.

```
/*
* This union is a handy way of representing the various object identifiers
* used by the APIs.
*/
typedef union {
   /* IMA Group ID */
   NPF_F_IMA_Group_ID_t        groupID;

   /* IMA Link ID */
   NPF_F_IMA_Link_ID_t        linkID;
} NPF_F_IMA_Id_t;

/*
* An asynchronous response contains a configuration object ID,
* an error or success code, and in some cases a function-
* specific structure embedded in a union. One or more of
* these is passed to the callback function as an array
* within the callback data structure (below)
*/
typedef struct {/* Asynchronous Response Structure */
    NPF_error_t                error;      /* Error code for this resp    */
    NPF_F_IMA_Id_t             objId;      /* Object Indetifier           */
    union {
        /* NPF_F_IMA_LFB_AttributesQuery() */
        NPF_F_IMA_LFB_AttrQueryResponse_t lfbAttrQueryResponse;

        /* NPF_F_IMA_Link_StatsGet()       */
        NPF_F_IMA_Link_Stats_t           linkStats;

        /* NPF_F_IMA_Link_StateGet()       */
        NPF_F_IMA_Link_State_t           linkState;

        /* NPF_F_IMA_Link_Query()          */
        NPF_F_IMA_Link_Info_t            linkInfo;

        /* NPF_F_IMA_Link_LastICPInfoGet() */
        NPF_F_IMA_Icp_Cell_t             icpCell;

        /* NPF_F_IMA_Group_StatsGet()      */
        NPF_F_IMA_Group_Stats_t          groupStats;

        /* NPF_F_IMA_Group_StateGet()      */
        NPF_F_IMA_Group_State_t          groupState;

        /* NPF_F_IMA_Group_Query()         */
        NPF_F_IMA_Group_Info_t           groupInfo;

       /* NPF_F_IMA_Group_TestSet()       */
        NPF_uint32_t                     testResultBitMap;
    } u;
} NPF_F_IMA_AsyncResponse_t;
```

4.2.2 Callback Type

This enumeration is used to indicate reason for invoking the callback function.

```
/*
* Completion Callback Types, to be found in the callback
* data structure, NPF_F_IMA_CallbackData_t.
*/
typedef enum NPF_F_IMA_CallbackType {
```

```
    /* Function to query IMA LFB attributes */
    NPF_F_IMA_ATTR_QUERY = 1,

    /* Functions for IMA group configuration and management */
    NPF_F_IMA_GROUP_SET = 2,         /* Add or Modify an IMA group       */
    NPF_F_IMA_GROUP_DELETE = 3,      /* Delete an IMA group              */
    NPF_F_IMA_GROUP_ENABLE = 4,      /* Put an IMA group in service      */
    NPF_F_IMA_GROUP_DISABLE = 5,     /* Take an IMA group out of service */
    NPF_F_IMA_GROUP_QUERY = 6,       /* Query config. And states of group*/
    NPF_F_IMA_GROUP_STATS_GET = 7,   /* Query statistics of an IMA group */
    NPF_F_IMA_GROUP_STATE_GET = 8,   /* Query state m/c states of a group*/
    NPF_F_IMA_GROUP_TEST_SET = 9,    /* Start/Stop Test pattern procedure*/

    /* Functions for IMA link configuration and management */
    NPF_F_IMA_LINK_SET = 10,          /* Add or Modify an IMA link       */
    NPF_F_IMA_LINK_DELETE = 11,      /* Delete an IMA link               */
    NPF_F_IMA_LINK_ENABLE = 12,      /* Put an IMA link in service       */
    NPF_F_IMA_LINK_DISABLE = 13,     /* Put an IMA link out of service   */
    NPF_F_IMA_LINK_QUERY = 14,       /* Query config and states of a link*/
    NPF_F_IMA_LINK_STATS_GET = 15,   /* Query statistics of an IMA link  */
    NPF_F_IMA_LINK_STATE_GET = 16,   /* Query state m/c states of a link */
    NPF_F_IMA_LINK_LAST_ICP_GET = 17,/* Get the payload of last ICP
                                     * cell received on queried link  */
} NPF_F_IMA_CallbackType_t;
```

4.2.3 Callback Data

An asynchronous response contains an error/success code and a function-specific structure embedded in a union in the `NPF_F_IMA_CallbackData_t` structure.

```
/*
* 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_IMA_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_IMA_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_IMA_AsyncResponse_t resp;       /* Response struct              */
} NPF_F_IMA_CallbackData_t;
```

The callback data that returned for different callback types is summarized in Table 4-1 Callback type to Callback data mapping table.

Table 4-1 Callback type to Callback data mapping table

Callback Type	Callback Data
NPF_F_IMA_ATTR_QUERY	lfbAttrQueryResponse
NPF_F_IMA_GROUP_SET	None
NPF_F_IMA_GROUP_DELETE	None
NPF_F_IMA_GROUP_ENABLE	None
NPF_F_IMA_GROUP_DISABLE	None
NPF_F_IMA_GROUP_QUERY	groupInfo
NPF_F_IMA_GROUP_STATS_GET	groupStats
NPF_F_IMA_GROUP_STATE_GET	groupState
NPF_F_IMA_GROUP_TEST_SET	None
NPF_F_IMA_LINK_SET	None
NPF_F_IMA_LINK_DELETE	None
NPF_F_IMA_LINK_ENABLE	None
NPF_F_IMA_LINK_DISABLE	None
NPF_F_IMA_LINK_QUERY	linkInfo
NPF_F_IMA_LINK_STATS_GET	linkStats
NPF_F_IMA_LINK_STATE_GET	linkState
NPF_F_IMA_LINK_LAST_ICP_GET	icpCell

The IMA LFB API functions and their type codes are summarized in Table 4-2.

Table 4-2 Callback type to function mapping

Callback Type	Function
NPF_F_IMA_ATTR_QUERY	NPF_F_IMA_LFB_AttributesQuery()
NPF_F_IMA_GROUP_SET	NPF_F_IMA_GroupSet()
NPF_F_IMA_GROUP_DELETE	NPF_F_IMA_GroupDelete()
NPF_F_IMA_GROUP_ENABLE	NPF_F_IMA_GroupEnable()
NPF_F_IMA_GROUP_DISABLE	NPF_F_IMA_GroupDisable()
NPF_F_IMA_GROUP_QUERY	NPF_F_IMA_GroupQuery()
NPF_F_IMA_GROUP_STATS_GET	NPF_F_IMA_GroupStatsGet()
NPF_F_IMA_GROUP_STATE_GET	NPF_F_IMA_GroupStateGet()
NPF_F_IMA_GROUP_TEST_SET	NPF_F_IMA_GroupTestSet()
NPF_F_IMA_LINK_SET	NPF_F_IMA_LinkSet()
NPF_F_IMA_LINK_DELETE	NPF_F_IMA_LinkDelete()
NPF_F_IMA_LINK_ENABLE	NPF_F_IMA_LinkEnable()
NPF_F_IMA_LINK_DISABLE	NPF_F_IMA_LinkDisable()
NPF_F_IMA_LINK_QUERY	NPF_F_IMA_LinkQuery()
NPF_F_IMA_LINK_STATS_GET	NPF_F_IMA_LinkStatsGet()
NPF_F_IMA_LINK_STATE_GET	NPF_F_IMA_LinkStateGet()
NPF_F_IMA_LINK_LAST_ICP_GET	NPF_F_IMA_Link_LastICPInfoGet()

4.3 Data Structures for Event Notifications

4.3.1 Event Notification Types

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

```
/*
*   IMA LFB Event Types
*/
typedef enum {
   /* LIF defect detected at NE for the link */
   NPF_F_IMA_EVENT_LINK_LIF_RAISED = 1,

   /* LIF defect cleared at NE for the link */
   NPF_F_IMA_EVENT_LINK_LIF_CLEARED = 2,

   /* LODS defect detected at NE for the link */
   NPF_F_IMA_EVENT_LINK_LODS_RAISED = 3,

   /* LODS defect cleared at NE for the link */
   NPF_F_IMA_EVENT_LINK_LODS_CLEARED = 4,

   /* RDI-IMA defect detected at NE for the link */
   NPF_F_IMA_EVENT_LINK_RFI_RAISED = 5,

   /* RDI-IMA defect detected at NE for the link */
   NPF_F_IMA_EVENT_LINK_RFI_CLEARED = 6,

   /* Tx link found to be not connected to matching IMA unit at FE */
   NPF_F_IMA_EVENT_LINK_TX_MISCONNECT_RAISED = 7,

   /* Tx link misconnection cleared  */
   NPF_F_IMA_EVENT_LINK_TX_MISCONNECT_CLEARED = 8,

   /* Rx link found to be not connected to matching IMA unit at FE */
   NPF_F_IMA_EVENT_LINK_RX_MISCONNECT_RAISED = 9,

   /* Rx link misconnection cleared  */
   NPF_F_IMA_EVENT_LINK_RX_MISCONNECT_CLEARED = 10,

   /* Implementation specific Tx fault raised */
   NPF_F_IMA_EVENT_LINK_TX_FAULT_RAISED = 11,

   /* Implementation specific Tx fault cleared */
   NPF_F_IMA_EVENT_LINK_TX_FAULT_CLEARED = 12,

   /* Implementation specific Rx fault raised */
   NPF_F_IMA_EVENT_LINK_RX_FAULT_RAISED = 13,

   /* Implementation specific Rx fault cleared */
   NPF_F_IMA_EVENT_LINK_RX_FAULT_CLEARED = 14,

   /* FE reports Tx link unusable */
   NPF_F_IMA_EVENT_LINK_TX_UNUSABLE_FE_RAISED = 15,

   /* FE reports Tx link usable/active */
   NPF_F_IMA_EVENT_LINK_TX_UNUSABLE_FE_CLEARED = 16,

   /* FE reports Rx link unusable */
   NPF_F_IMA_EVENT_LINK_RX_UNUSABLE_FE_RAISED = 17,

   /* FE reports Rx link usable/active */
   NPF_F_IMA_EVENT_LINK_RX_UNUSABLE_FE_CLEARED = 18,

   /* Test pattern failed to loop on specified link */
```

```
NPF_F_IMA_EVENT_LINK_TEST_LINK_FAIL_RAISED = 19,

/* Test link failure condition on specified link cleared */
NPF_F_IMA_EVENT_LINK_TEST_LINK_FAIL_CLEARED = 20,

/* Event to notify change in near end link state machine transition */
NPF_F_IMA_EVENT_LINK_STATE_MACHINE_TRANSITION = 21,

/* Far end group in startup state */
NPF_F_IMA_EVENT_GROUP_STARTUP_FE_RAISED_RAISED = 22,

/* Far end tried to use unacceptable configuration params */
NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED_RAISED = 23,

/* Far end uses new acceptable configuration params */
NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED_CLEARED = 24,

/* Far end reports unacceptable configuration params */
NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED_FE_RAISED = 25,

/* Far end accepts new configuration params */
NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED_FE_CLEARED = 26,

/* Less than P(tx) or P(rx) links are active */
NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS_RAISED = 27,

/* Condition where less than P(tx) or P(rx) links are active cleared */
NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS_CLEARED = 28,

/* Far end reports less than P(rx) or P(tx) links are active */
NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS_FE_RAISED = 29,

/* Condition where Far end reports less than P(rx) or P(tx)
  links are active cleared */
NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS_FE_CLEARED = 30,

/* Far end reports that it is blocked */
NPF_F_IMA_EVENT_GROUP_BLOCKED_FE_RAISED = 31,

/* Far end reports that blocking no longer exists */
NPF_F_IMA_EVENT_GROUP_BLOCKED_FE_CLEARED = 32,

/* Far end transmit clock mode is different than NE transmit clock mode */
NPF_F_IMA_EVENT_GROUP_TIMING_MISMATCH_RAISED = 33,

/* Mismatch of far end transmit clock mode and NE transmit clock mode
 * cleared                                                          */
NPF_F_IMA_EVENT_GROUP_TIMING_MISMATCH_CLEARED = 34,

/* Test pattern failed to loop on some links */
NPF_F_IMA_EVENT_GROUP_TEST_LINK_FAIL_RAISED = 35,

/* Test link failure condition cleared on all link */
NPF_F_IMA_EVENT_GROUP_TEST_LINK_FAIL_CLEARED = 36,

/* Event to notify change in near end group state machine transition */
NPF_F_IMA_EVENT_GROUP_STATE_MACHINE_TRANSITION = 37,

/* Event to notify change in near end group traffic state machine
```

```
      transition */
   NPF_F_IMA_EVENT_GROUP_TRAFFIC_STATE_MACHINE_TRANSITION = 38,

} NPF_F_IMA_Event_t;
```

4.3.1.1 Event Mask bit definitions

```
/*
 *   Definitions for selectively enabling IMA LFB Events
 */
/* Link specific alarms */
#define NPF_F_IMA_EVENT_LINK_LIF                              (1 << 0)
#define NPF_F_IMA_EVENT_LINK_LODS                             (1 << 1)
#define NPF_F_IMA_EVENT_LINK_RFI                              (1 << 2)
#define NPF_F_IMA_EVENT_LINK_TX_MISCONNECT                    (1 << 3)
#define NPF_F_IMA_EVENT_LINK_RX_MISCONNECT                    (1 << 4)
#define NPF_F_IMA_EVENT_LINK_TX_FAULT                         (1 << 5)
#define NPF_F_IMA_EVENT_LINK_RX_FAULT                         (1 << 6)
#define NPF_F_IMA_EVENT_LINK_TX_UNUSABLE_FE                   (1 << 7)
#define NPF_F_IMA_EVENT_LINK_RX_UNUSABLE_FE                   (1 << 8)
#define NPF_F_IMA_EVENT_LINK_TEST_LINK_STATUS                 (1 << 9)
#define NPF_F_IMA_EVENT_LINK_STATE_MACHINE_TRANSITION         (1 << 10)

/* Group specific alarms */
#define NPF_F_IMA_EVENT_GROUP_STARTUP_FE                      (1 << 16)
#define NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED                  (1 << 17)
#define NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED_FE               (1 << 18)
#define NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS              (1 << 19)
#define NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS_FE           (1 << 20)
#define NPF_F_IMA_EVENT_GROUP_BLOCKED_FE                      (1 << 21)
#define NPF_F_IMA_EVENT_GROUP_TIMING_MISMATCH                 (1 << 22)
#define NPF_F_IMA_EVENT_GROUP_TEST_LINK_STATUS                (1 << 23)
#define NPF_F_IMA_EVENT_GROUP_STATE_MACHINE_TRANSITION        (1 << 24)
#define NPF_F_IMA_EVENT_GROUP_TRAFFIC_STATE_MACHINE_TRANSITION (1 << 25)
#define NPF_F_IMA_EVENT_LAST                                  (1 << 25)
```

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

4.3.2 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 IMA LFB FAPI. An event type field indicates which member of the union is relevant in the specific structure.

```
/*
 *   IMA LFB 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_IMA_Event_t     eventType;   /* Type of event reported        */
    NPF_F_IMA_Id_t        objId;       /* Object for which event raised */

    union {
        /* Link states – filled for link specific events       */
        NPF_F_IMA_Link_State_t           linkState;

        /* Group states – filled for group specific events     */
        NPF_F_IMA_Group_State_t          groupState;
```

```
    } u;
} NPF_F_IMA_EventData_t;
```

4.4 Error Codes

4.4.1 Common NPF Error Codes

The common error codes that are returned by IMA LFB are listed below:

- `NPF_NO_ERROR` - This value MUST be returned when a function was successfully invoked. This value is also used in completion callbacks where it MUST be the only value used to signify success.
- `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` - A function was invoked with a callback handle that did not correspond to a valid NPF callback handle as returned by a registration function, or a callback handle was registered with a registration function belonging to a different API than the function call where the handle was passed in.
- `NPF_E_BAD_CALLBACK_FUNCTION` - A callback registration was invoked with a function pointer parameter that was invalid.
- `NPF_E_CALLBACK_ALREADY_REGISTERED` - A callback or event registration was invoked with a pair composed of a function pointer and a user context that was previously used for an identical registration.
- `NPF_E_FUNCTION_NOT_SUPPORTED` - This error value MUST be returned when an optional function call is not implemented by an implementation. This error value MUST NOT be returned by any required function call. This error value MUST be returned as the function return value (i.e., synchronously).

4.4.2 LFB Specific Error Codes

This section defines IMA LFB 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) */
typedef NPF_uint32_t NPF_F_Ima_ErrorType_t;

#define NPF_IMA_BASE_ERR (NPF_F_IMA_LFB_TYPE * 100)
#define IMA_ERR(n) ((NPF_F_IMA_ErrorType_t) (NPF_IMA_BASE_ERR+ (n))

/* LFB ID is not an ID of LFB that has IMA functionality*/
#define NPF_IMA_F_E_INVALID_IMA_BLOCK_ID IMA_ERR (0)

/* Invalid configuration attributes */
#define NPF_IMA_F_E_INVALID_ATTRIBUTE    IMA_ERR (1)

/* Test procedure failed on one or more receive links */
#define NPF_IMA_F_E_TEST_PROC_FAILED     IMA_ERR (2)

/* Group specified in link configuration not recognized */
#define NPF_IMA_F_E_UNKNOWN_GROUP        IMA_ERR (3)

/* Group cannot be deleted as it has associated links and FAPI client has not
 * requested deletion of contained links  */
#define NPF_IMA_F_E_CONT_LINKS_EXIST        IMA_ERR (4)
```

5 Functional API (FAPI)

5.1 Required Functions

5.1.1 Completion Callback Function

This callback function is for the application to register an asynchronous response handling routine to the IMA API implementation. This callback function is intended to be implemented by the application. The application should register this function with the IMA API implementation using the `NPF_F_IMA_Register` function.

```
typedef void (*NPF_F_IMA_CallbackFunc_t) (
      NPF_IN NPF_userContext_t            userContext,
      NPF_IN NPF_correlator_t             correlator,
      NPF_IN NPF_F_IMA_CallbackData_t     data);
```

5.1.1.1 Description

This function is a routine to handle to IMA asynchronous responses.

5.1.1.2 Input Parameters

- `userContext` - The context item supplied by the application when the completion callback routine was registered.
- `correlator` - The correlator item that was supplied by the application when the IMA API function call was invoked.
- `data` - The response information related to the particular callback type

5.1.1.3 Output Parameters

None

5.1.1.4 Return Values

None

5.1.2 Completion Callback Registration Function

```
NPF_error_t NPF_F_IMA_Register (
      NPF_IN NPF_userContext_t                  userContext,
      NPF_IN NPF_F_IMA_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 related to IMA API function calls. Applications 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 a 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. On attempting to register a duplicate callback function the handle of the callback previously registered will be returned in `callBackHandle` and the return code will indicate `NPF_E_ALREADY_REGISTERED.`

5.1.2.2 Input Parameters

- `userContext` – A parameter for uniquely identifying the context of the application registering the completion callback function. The exact value will be provided back to the registered completion callback function as its first parameter when it is called. Applications can assign any value to the `userContext` and the value is completely opaque to the API 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 NPF IMA API functions. It will also be used when deregistering the `userContext` and `callbackFunc` pair.

5.1.2.4 Return Values

- `NPF_NO_ERROR` - The registration completed successfully.
- `NPF_E_BAD_CALLBACK_FUNCTION` – The callbackFunc is NULL or invalid.
- `NPF_E_ALREADY_REGISTERED` – No new registration was made since the userContext and callbackFunc pair was already registered.

5.1.2.5 Notes

- This API function may be invoked by any application interested in receiving asynchronous responses for IMA API function calls.
- This function operates in a synchronous manner, providing a return value as listed above.

5.1.3 Completion Callback Deregistration Function

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

5.1.3.1 Description

This function is used by an application to deregister a user context and callback function pair.

5.1.3.2 Input Parameters

- `callbackHandle` - The unique identifier returned to the application when the completion callback routine was registered.

5.1.3.3 Output Parameters

None

5.1.3.4 Return Values

- `NPF_NO_ERROR` - De-registration was completed successfully.
- `NPF_E_BAD_CALLBACK_HANDLE` – De-registration did not complete successfully due to problems with the callback handle provided.

5.1.3.5 Notes

- This API function MAY be invoked by any application no longer interested in receiving asynchronous responses for IMA API function calls.
- This function operates in a synchronous manner, providing a return value as listed above.
- There may be a timing window where outstanding callbacks continue to be delivered to the callback routine after de-registration function has been invoked. It is the implementation's responsibility to guarantee that the callback function is not called after the deregister function has returned.

5.1.4 Event Handler Function

```
typedef void (*NPF_F_IMA_EventCallFunc_t) (
    NPF_IN  NPF_userContext_t      userContext,
    NPF_IN  NPF_uint32_t           nEvent,
    NPF_IN  NPF_F_IMA_EventData_t  *imaEventArray);
```

5.1.4.1 Description

This handler function is for the FAPI client to register an event handling routine to the IMA LFB. 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 `imaEventArray` 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 IMA LFB implementation through `NPF_F_IMA_EventHandler_Register()` function.

This function is invoked when the related event happens. The IMA LFB may invoke the registered event handler function any time after the `NPF_F_IMA_EventHandler_Register` () is invoked by the FAPI client.

5.1.4.2 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 first parameter when it is called. The application can assign any value to the userContext and the value is completely opaque to the implementation.
- `nEvent` – Number of events reported.
- `imaEventArray` – 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_IMA_EventHandler_Register(
            NPF_IN  NPF_userContext_t         userContext,
            NPF_IN  NPF_F_IMA_EventCallFunc_t imaEvtCallFn,
            NPF_IN  NPF_eventMask_t           imaEvtMask,
            NPF_OUT NPF_callbackHandle_t      *imaEvtCallHdl);
```

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 userContext and `imaEvtCallFn` identifies the event handling routine. For each individual pair, a unique `imaEvtCallHdl` will be assigned for future reference. Since the event handling routine is identified by both `userContext` and `imaEventCallFunc`, duplicate registration of same event handling routine with different userContext is allowed. Also, the same `userContext` can be shared among different event handling routines. Duplicate registration of the same `userContext` and `imaEventCallFunc` 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

- `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 userContext and the value is completely opaque to the implementation.
- `imaEvtCallFn` – Contains the class of event for which handler is being registered and a pointer to the event handling routine to be registered.

- `imaEvtMask` – 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

- `imaEvtCallHdl` – A unique identifier assigned for the registered `userContext` and `imaEventCallFunc` 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 `userContext` and `imaEventCallFunc` 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`: `imaEventCallFunc` is NULL.
- `NPF_E_CALLBACK_ALREADY_REGISTERED`: No new registration was made since the userContext and `imaEventCallFunc` pair was already registered.

5.1.6 Event Handler Deregistration Function

```
NPF_error_t NPF_F_IMA_EventHandler_Deregister(
              NPF_IN NPF_callbackHandle_t    imaEventCallHandle);
```

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

- `imaEventCallHandle` – 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.

5.1.7 LFB Attributes Query Function

```
NPF_error_t NPF_F_IMA_LFB_AttributesQuery (
      NPF_IN NPF_callbackHandle_t   callbackHandle,
      NPF_IN NPF_correlator_t       correlator,
      NPF_IN NPF_errorReporting_t   errorReporting,
      NPF_IN NPF_FEHandle_t         feHandle,
      NPF_IN NPF_BlockId_t          blockId);
```

5.1.7.1 Description

This function call is used to query ONLY one IMA LFB's attributes at a time. If the IMA LFB exists, the attributes of this LFB are returned in the completion callback.

5.1.7.2 Input Parameters

- `callbackHandle` - The unique identifier provided to the application when the completion callback routine was registered.
- `correlator` - A unique application invocation context 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 API invocation.
- `feHandle` - The forwarding element Handle returned by `NPF_F_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.

5.1.7.3 Output Parameters

None

5.1.7.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The LFB attributes were not queried due to invalid IMA block ID passed in input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The LFB attributes were not queried because the callback handle was invalid.
- `NPF_E_FUNCTION_NOT_SUPPORTED` - The function call is not supported.

5.1.7.5 Asynchronous Response

There may be multiple asynchronous callbacks to this request. Possible error codes are:

- `NPF_NO_ERROR` – Operation completed successfully.
- `NPF_E_IMA_INVALID_IMA_BLOCK_ID` – LFB ID is not an ID of LFB that has IMA functionality.

The `lfbAttrQueryResponse` field of the union in the `NPF_F_IMA_AsyncResponse_t` structure returned in callback contains response data. The error code is returned in the error field.

5.1.8 Add or Modify an IMA group

```
NPF_error_t NPF_F_IMA_GroupSet (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Group_Config_t   *cfgArray);
```

5.1.8.1 Description

This function adds/creates one or more IMA groups, or modifies the attributes of an existing group. If the administrative status of the group is set as NPF_STATUS_DOWN, the group will not transition further from down state.

5.1.8.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA groups to set
- `cfgArray` - Pointer to an array of IMA group attribute structures

5.1.8.3 Output Parameters

None

5.1.8.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.8.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a group 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_IMA_F_E_INVALID_ATTRIBUTE` - Invalid attribute

5.1.9 Delete an IMA group

```
NPF_error_t NPF_F_IMA_GroupDelete (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_boolean_t              delContainedLnks,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Group_ID_t       *delArray);
```

5.1.9.1 Description

This function is used to delete a previously configured group.

5.1.9.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.

- `delContainedLnks` – When set to `NPF_TRUE` indicates that all associated links should be deleted. If this parameter is set to `NPF_FALSE`, the function will return an error if there are links contained within the group being deleted.
- `numEntries` - Number of IMA groups to delete
- `delArray` - Pointer to an array of IMA group IDs of IMA groups to delete

5.1.9.3 Output Parameters

None

5.1.9.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.9.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a group 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_IMA_F_E_CONT_LINKS_EXIST` – The specified group could not be deleted as there are links associated with this group and the parameter `delContainedLnks` was set to `NPF_FALSE`.
- `NPF_E_RESOURCE_NONEXIST` - Specified IMA group doesn't exist

5.1.10 Put an IMA group in service

```
NPF_error_t NPF_F_IMA_GroupEnable (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Group_ID_t       *enaArray);
```

5.1.10.1 Description

This function is used to mark the administrative status of an IMA group as enabled. Enabling the IMA group cause the inhibition of the group state to be removed allowing the group state machine to transition if allowed from the BLOCKED state to OPERATIONAL state.

5.1.10.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA groups to enable
- `enaArray` - Pointer to an array of IMA group IDs of IMA groups to enable

5.1.10.3 Output Parameters

None

5.1.10.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.10.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a group 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_E_RESOURCE_NONEXIST` - Specified IMA group doesn't exist

5.1.11 Put an IMA group out of service

```
NPF_error_t NPF_F_IMA_GroupDisable (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Group_ID_t       *enaArray);
```

5.1.11.1 Description

This function is used to mark the administrative status of an IMA group as disabled. Disabling the IMA group cause the inhibition of the group and causes the group state machine to transition to the BLOCKED state.

5.1.11.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA groups to disable

- `enaArray` - Pointer to an array of IMA group IDs of IMA groups to disable

5.1.11.3 Output Parameters

None

5.1.11.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.11.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a group 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_E_RESOURCE_NONEXIST` - Specified IMA group doesn't exist

5.1.12 Query an IMA Group

```
NPF_error_t NPF_F_IMA_GroupQuery (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Group_ID_t       *grpIdArr);
```

5.1.12.1 Description

This function is used to query the configuration and current state of one or more IMA groups. If the `numEntries` is set to 0, information for all IMA groups configured in the LFB is returned in the response.

5.1.12.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA groups to query
- `grpIdArr` - Pointer to an array of IMA group IDs of IMA groups to query

5.1.12.3 Output Parameters

None

5.1.12.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.12.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a group 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 group information is returned in the `groupInfo` field of the union in the response structure. 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_E_RESOURCE_NONEXIST` - Specified IMA group doesn't exist

5.1.13 Get statistics accumulated for an IMA Group

```
NPF_error_t NPF_F_IMA_GroupStatsGet (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_boolean_t              resetStats,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Group_ID_t       *grpIdArr);
```

5.1.13.1 Description

This function is used to get via a callback the current counter values for one or more IMA groups.

5.1.13.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `resetStats` – If set to TRUE, the statistics counters being read are reset to 0
- `numEntries` - Number of IMA groups to query
- `grpIdArr` - Pointer to an array of IMA group IDs of IMA groups to query

5.1.13.3 Output Parameters

None

5.1.13.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.

- NPF_E_UNKNOWN - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - The configuration did not complete successfully as the callback handle was invalid.

5.1.13.5 Asynchronous Response

A total of numEntries asynchronous (NPF_F_IMA_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains a group 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 counters are returned in the groupStats field of the union in the response structure.

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_E_RESOURCE_NONEXIST - Specified IMA group doesn't exist

5.1.14 Get state information for an IMA Group

```
NPF_error_t NPF_F_IMA_GroupStateGet (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Group_ID_t       *grpIdArr);
```

5.1.14.1 Description

This function is used to get via a callback the current group state machine and group traffic state machine states for the queried IMA group.

5.1.14.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_IMA_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 forwarding element Handle returned by NPF_F_ATM_topologyGetFEInfoList() call.
- blockId – The unique identification of the IMA LFB.
- numEntries - Number of IMA groups to query
- grpIdArr - Pointer to an array of IMA group IDs of IMA groups to query

5.1.14.3 Output Parameters

None

5.1.14.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - The configuration did not complete successfully as the callback handle was invalid.

5.1.14.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a group 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 counters are returned in the `groupState` field of the union in the response structure.

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_E_RESOURCE_NONEXIST` - Specified IMA group doesn't exist

5.1.15 Configure Test Pattern procedure for an IMA group

```
NPF_error_t NPF_F_IMA_GroupTestSet (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Group_Test_Proc_Config_t  *grpIdArr);
```

5.1.15.1 Description

This function is used to configure test pattern procedure for an IMA group. The FAPI client may either choose the test link ID and test pattern for generating the test pattern or let the FAPI implementation select the test link ID and the test pattern. The status of the test pattern procedure is indicated back to the FAPI client in the asynchronous response. If the test pattern procedure is not disabled, the LFB continues to send the test pattern on the specified test link. Any subsequent change in the test procedure status for the recognized links and the group is indicated to the FAPI client via corresponding events.

5.1.15.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA groups to test
- `grpIdArr` - Pointer to an array of IMA group IDs of IMA groups to test

5.1.15.3 Output Parameters

None

5.1.15.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.

- NPF_E_BAD_CALLBACK_HANDLE - The configuration did not complete successfully as the callback handle was invalid.

5.1.15.5 Asynchronous Response

A total of numEntries asynchronous (NPF_F_IMA_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains a group 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. If the test link procedure failed, a bitmap testResultBitMap indicating the receive links on which the test pattern failed to loop back is indicated to the FAPI client. Each bit in the bit map corresponds to the logical link ID of a link in the group and if set indicates that the test pattern failed to loop back on that link. The least significant bit corresponds to the lowest numbered link in the group.

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_E_RESOURCE_NONEXIST - Specified IMA group doesn't exist
- NPF_IMA_F_E_TEST_PROC_FAILED - Test pattern failed to loop back on one or more links in the group.

5.1.16 Add or Modify an IMA link

```
NPF_error_t NPF_F_IMA_LinkSet (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Link_Config_t   *cfgArray);
```

5.1.16.1 Description

This function adds/creates one or more IMA link, or modifies the attributes of an existing link. If the administrative status of the link is set as NPF_STATUS_DOWN, the link will not transition further from unusable state. If the administrative status is set as NPF_STATUS_UP, the link will transition to usable status.

5.1.16.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_IMA_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 forwarding element Handle returned by NPF_F_ATM_topologyGetFEInfoList() call.
- blockId – The unique identification of the IMA LFB.
- numEntries - Number of IMA links to set
- cfgArray - Pointer to an array of IMA link attribute structures

5.1.16.3 Output Parameters

None

5.1.16.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.16.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a 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_IMA_F_E_INVALID_ATTRIBUTE` - Invalid attribute
- `NPF_IMA_F_E_UNKNOWN_GROUP` – Group specified in link configuration is not recognized

5.1.17 Delete an IMA Link

```
NPF_error_t NPF_F_IMA_LinkDelete (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Link_ID_t       *delArray);
```

5.1.17.1 Description

This function is used to delete a previously configured link.

5.1.17.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA links to delete
- `delArray` - Pointer to an array of IMA link IDs of IMA links to delete

5.1.17.3 Output Parameters

None

5.1.17.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.

- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.17.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a 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_E_RESOURCE_NONEXIST` -Specified IMA link doesn't exist

5.1.18 Put an IMA link in service

```
NPF_error_t NPF_F_IMA_LinkEnable (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Link_ID_t       *enaArray);
```

5.1.18.1 Description

This function is used to mark the administrative status of an IMA link as enabled. Enabling the IMA link cause the inhibition of the link state to be removed allowing the link state machine to transition if allowed from the UNUSABLE state to USABLE state.

5.1.18.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA links to enable
- `enaArray` - Pointer to an array of IMA link IDs of IMA links to enable

5.1.18.3 Output Parameters

None

5.1.18.4 Return Value*s*

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.18.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a 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_E_RESOURCE_NONEXIST` - Specified IMA group doesn't exist

5.1.19 Put an IMA link out of service

```
NPF_error_t NPF_F_IMA_LinkDisable (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Link_ID_t       *enaArray);
```

5.1.19.1 Description

This function is used to mark the administrative status of an IMA link as disabled. Disabling the IMA link cause the inhibition of the link and leading to the link being marked BLOCKED.

5.1.19.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA links to disable
- `enaArray` - Pointer to an array of IMA link IDs of IMA links to disable

5.1.19.3 Output Parameters

None

5.1.19.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.19.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a group 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_E_RESOURCE_NONEXIST` - Specified IMA link doesn't exist

5.1.20 Query an IMA Link

```
NPF_error_t NPF_F_IMA_LinkQuery (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Link_ID_t       *lnkIdArr);
```

5.1.20.1 Description

This function is used to query the configuration and current state of one or more IMA links. If the `numEntries` is set to 0, information for all IMA links configured in the LFB is returned in the response.

5.1.20.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA links to query
- `lnkIdArr` - Pointer to an array of IMA link IDs of IMA links to query

5.1.20.3 Output Parameters

None

5.1.20.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.20.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a 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 and the group information is returned in the `linkInfo` field of the union in the response structure.

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_E_RESOURCE_NONEXIST - Specified IMA link doesn't exist

5.1.21 Get statistics accumulated for an IMA Link

```
NPF_error_t NPF_F_IMA_LinkStatsGet (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_boolean_t              resetStats,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Link_ID_t       *lnkIdArr);
```

5.1.21.1 Description

This function is used to get via a callback the current counter values for one or more IMA links.

5.1.21.2 Input Parameters

- cbHandle – The callback handle returned by NPF_F_IMA_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 forwarding element Handle returned by NPF_F_ATM_topologyGetFEInfoList() call.
- resetStats – If set to TRUE, the statistics counters being read are reset to 0
- blockId – The unique identification of the IMA LFB.
- numEntries - Number of IMA groups to query
- lnkIdArr - Pointer to an array of IMA link IDs of IMA links to query

5.1.21.3 Output Parameters

None

5.1.21.4 Return Values

- NPF_NO_ERROR - The operation is in progress.
- NPF_E_UNKNOWN - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE - The configuration did not complete successfully as the callback handle was invalid.

5.1.21.5 Asynchronous Response

A total of numEntries asynchronous (NPF_F_IMA_AsyncResponse_t) responses are passed to the callback function, in one or more invocations. Each response contains a 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 and the counters are returned in the linkStats field of the union in the response structure.

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_E_RESOURCE_NONEXIST - Specified IMA link doesn't exist

5.1.22 Get state information for an IMA Link

```
NPF_error_t NPF_F_IMA_LinkStateGet (
    NPF_IN NPF_callbackHandle_t       cbHandle,
    NPF_IN NPF_correlator_t           cbCorrelator,
    NPF_IN NPF_errorReporting_t       errorReporting,
    NPF_IN NPF_FEHandle_t             feHandle,
    NPF_IN NPF_BlockId_t              blockId,
    NPF_IN NPF_uint32_t               numEntries,
    NPF_IN NPF_F_IMA_Link_ID_t       *lnkIdArr);
```

5.1.22.1 Description

This function is used to get via a callback the current link state machine states for the queried IMA links.

5.1.22.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA links to query
- `lnkIdArr` - Pointer to an array of IMA link IDs of IMA links to query

5.1.22.3 Output Parameters

None

5.1.22.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.22.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a 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 and the counters are returned in the `linkState` field of the union in the response structure.

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_E_RESOURCE_NONEXIST` - Specified IMA link doesn't exist

5.1.23 Get last received ICP cell for an IMA Link

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

```
NPF_IN NPF_errorReporting_t       errorReporting,
NPF_IN NPF_FEHandle_t             feHandle,
NPF_IN NPF_BlockId_t              blockId,
NPF_IN NPF_uint32_t               numEntries,
NPF_IN NPF_F_IMA_Link_ID_t       *lnkIdArr);
```

5.1.23.1 Description

This function is used to get via a callback the contents of the last ICP cell received on the queried IMA links. This is an optional function.

5.1.23.2 Input Parameters

- `cbHandle` – The callback handle returned by `NPF_F_IMA_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 forwarding element Handle returned by `NPF_F_ATM_topologyGetFEInfoList()` call.
- `blockId` – The unique identification of the IMA LFB.
- `numEntries` - Number of IMA links to query
- `lnkIdArr` - Pointer to an array of IMA link IDs of IMA links to query

5.1.23.3 Output Parameters

None

5.1.23.4 Return Values

- `NPF_NO_ERROR` - The operation is in progress.
- `NPF_E_UNKNOWN` - The configurations did not complete successfully due to problems encountered when handling the input parameters.
- `NPF_E_BAD_CALLBACK_HANDLE` - The configuration did not complete successfully as the callback handle was invalid.

5.1.23.5 Asynchronous Response

A total of `numEntries` asynchronous (`NPF_F_IMA_AsyncResponse_t`) responses are passed to the callback function, in one or more invocations. Each response contains a 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 and the counters are returned in the `icpCell` field of the union in the response structure.

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_E_RESOURCE_NONEXIST` - Specified IMA link doesn't 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] "Topology Manager Functional API", http://www.npforum.org/techinfo/topology_fapi_npf2002%20438%2023.pdf, Network Processing Forum.

[SWAPICON] "Software API Conventions Revision 2", http://www.npforum.org/techinfo/APIConventions2_IA.pdf, Network Processing Forum

[ATMLFBARC] "ATM Software API Architecture Framework", http://www.npforum.org/techinfo/npf2004.088.12.pdf, Network Processing Forum.

[ATMMGR] "ATM Configuration Manager Functional API", http://www.npforum.org/techinfo/npf2004.165.31.pdf, Network Processing Forum

[ATMIMAPI] "Interface Management API Implementation Agreement (ATM Interfaces) revision 3.0", http://www.npforum.org/techinfo/IM_API_IA_npf2004.218.12.pdf, Network Processing Forum.

Appendix A Header File Information

```
/*
 * This header file defines typedef, constants and structures
 * for the NP Forum ATM Policer Functional API
 */

#ifndef __NPF_F_ATM_IMA_H__
#define __NPF_F_ATM_IMA_H__

#ifdef __cplusplus
extern "C" {
#endif

/* It is possible to use the FAPI Topology Discovery
   APIs [npf2002.438] to discover an ATM IMA LFB
   in a forwarding element.  */

/* LFB type for IMA LFB */
#define NPF_F_IMA_LFB_TYPE    46

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

#define NPF_IMA_BASE_ERR (NPF_F_IMA_LFB_TYPE * 100)
#define IMA_ERR(n) ((NPF_F_IMA_ErrorType_t) (NPF_IMA_BASE_ERR+ (n))

/* LFB ID is not an ID of LFB that has IMA functionality*/
#define NPF_IMA_F_E_INVALID_IMA_BLOCK_ID IMA_ERR (0)

/* Invalid configuration attributes */
#define NPF_IMA_F_E_INVALID_ATTRIBUTE    IMA_ERR (1)

/* Test procedure failed on one or more receive links */
#define NPF_IMA_F_E_TEST_PROC_FAILED     IMA_ERR (2)

/* Group specified in link configuration not recognized */
#define NPF_IMA_F_E_UNKNOWN_GROUP        IMA_ERR (3)

/* Group cannot be deleted as it has associated links and FAPI client has not
 * requested deletion of contained links */
#define NPF_IMA_F_E_CONT_LINKS_EXIST        IMA_ERR (4)

/*
 *   Definitions for selectively enabling IMA LFB Events
 */
/* Link specific alarms */
#define NPF_F_IMA_EVENT_LINK_LIF                              (1 << 0)
#define NPF_F_IMA_EVENT_LINK_LODS                             (1 << 1)
#define NPF_F_IMA_EVENT_LINK_RFI                              (1 << 2)
#define NPF_F_IMA_EVENT_LINK_TX_MISCONNECT                    (1 << 3)
#define NPF_F_IMA_EVENT_LINK_RX_MISCONNECT                    (1 << 4)
#define NPF_F_IMA_EVENT_LINK_TX_FAULT                         (1 << 5)
#define NPF_F_IMA_EVENT_LINK_RX_FAULT                         (1 << 6)
#define NPF_F_IMA_EVENT_LINK_TX_UNUSABLE_FE                   (1 << 7)
#define NPF_F_IMA_EVENT_LINK_RX_UNUSABLE_FE                   (1 << 8)
#define NPF_F_IMA_EVENT_LINK_TEST_LINK_STATUS                 (1 << 9)
#define NPF_F_IMA_EVENT_LINK_STATE_MACHINE_TRANSITION         (1 << 10)

/* Group specific alarms */
```

```
#define NPF_F_IMA_EVENT_GROUP_STARTUP_FE                    (1 << 16)
#define NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED                (1 << 17)
#define NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED_FE             (1 << 18)
#define NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS            (1 << 19)
#define NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS_FE         (1 << 20)
#define NPF_F_IMA_EVENT_GROUP_BLOCKED_FE                    (1 << 21)
#define NPF_F_IMA_EVENT_GROUP_TIMING_MISMATCH               (1 << 22)
#define NPF_F_IMA_EVENT_GROUP_TEST_LINK_STATUS              (1 << 23)
#define NPF_F_IMA_EVENT_GROUP_STATE_MACHINE_TRANSITION      (1 << 24)
#define NPF_F_IMA_EVENT_GROUP_TRAFFIC_STATE_MACHINE_TRANSITION (1 << 25)
#define NPF_F_IMA_EVENT_LAST                                (1 << 25)
/*************************************************************
 * Enumerations and types for ATM IMA attributes and          *
 * completion callback data types                             *
 *************************************************************/
typedef NPF_uint32_t NPF_F_Ima_ErrorType_t;    /* Error type    */
typedef NPF_uint32_t NPF_F_IMA_Group_ID_t;     /* IMA group ID  */
typedef NPF_uint32_t NPF_F_IMA_Link_ID_t;      /* IMA link ID   */

/* Link state machine states */
typedef enum {
   /* Link not configured */
   NPF_F_IMA_LSM_STATE_NOT_IN_GROUP = 1,

   /* Link configured but cannot be used */
   NPF_F_IMA_LSM_STATE_UNUSABLE_UNKNOWN = 2,
   NPF_F_IMA_LSM_STATE_UNUSABLE_FAULT_LINK_DEFECT = 3,
   NPF_F_IMA_LSM_STATE_UNUSABLE_FAULT_LIF = 3,
   NPF_F_IMA_LSM_STATE_UNUSABLE_FAULT_LODS = 3,

   NPF_F_IMA_LSM_STATE_UNUSABLE_MISCONNECTED = 4,
   NPF_F_IMA_LSM_STATE_UNUSABLE_INHIBITED = 5,
   NPF_F_IMA_LSM_STATE_UNUSABLE_FAILED = 6,

   /* Link is ready to use */
   NPF_F_IMA_LSM_STATE_USABLE = 7,

   /* Link is active and capable of passing cells to/from ATM Layer */
   NPF_F_IMA_LSM_STATE_ACTIVE = 8
} NPF_F_IMA_LSM_State_t;

/* Group State Machine State. */
typedef enum {
   NPF_F_IMA_GSM_NOT_CONFIGURED = 1,
   NPF_F_IMA_GSM_START_UP = 2,
   NPF_F_IMA_GSM_START_UP_ACK = 3,
   NPF_F_IMA_GSM_CONFIG_ABORTED_UNSUPPORTED_FRAME_LEN = 4,
   NPF_F_IMA_GSM_CONFIG_ABORTED_INCOMPATIBLE_SYMMETRY = 5,
   NPF_F_IMA_GSM_CONFIG_ABORTED_UNSUPPORTED_IMA_VERSION = 6,
   NPF_F_IMA_GSM_CONFIG_ABORTED_OTHER = 7,
   NPF_F_IMA_GSM_INSUFFICIENT_LINKS = 8,
   NPF_F_IMA_GSM_BLOCKED = 9,
   NPF_F_IMA_GSM_OPERATIONAL = 10
} NPF_F_IMA_GSM_State_t;

/* IMA Group Traffic Machine State */
typedef enum {
   NPF_F_IMA_GTSM_DOWN = 0,
   NPF_F_IMA_GTSM_UP
```

```
} NPF_F_IMA_GTSM_State_t;

/* IMA Group Testing Mode */
typedef enum {
    NPF_F_TEST_PROC_DISABLED = 1,
    NPF_F_TEST_PROC_OPERATIONAL = 2,
} NPF_F_IMA_Test_Proc_Status_t;
/* Group Test Mode */
typedef struct {
   /* Testing link ID */
   NPF_int8_t  testLID;

   /* Test pattern */
   NPF_int32_t  testPattern;

   /* Test Procedure Status */
   NPF_F_IMA_Test_Proc_Status_t testStatus;

   /* Test verification Duration. The far end is expected to loop back the
    * test pattern on all links in the group within this duration. Failing
    * which the end initiating the test procedure will declare a test
    * procedure failure on the links on which the test pattern was not
    * loopback.
    */
   NPF_F_ATM_Timers_t expRespDuration;
} NPF_F_IMA_Group_Test_Mode_t;

/* Group test configuration */
typedef struct {
   /* A unique ID to identify the group */
   NPF_F_IMA_Group_ID_t        groupID;

   /* Start/Stop/Change pattern */
   NPF_F_IMA_Group_Test_Mode_t groupTestMode;
} NPF_F_IMA_Group_Test_Proc_Config_t;

typedef struct {
   /* The Interface handle of the IMA group */
   NPF_IfHandle_t              imaIfID;

   /* A unique ID to identify the group. The interface handle for the group
    * is an arbitrary value assigned by the IM APIs. The groupID may be used
    * to provide a fast way to lookup the group information. The FAPI
    * implementations may restrict the range of values assigned to this field
    * or may impose restrictions on the way this number is constructed and
    * any such restrictions are outside the scope of NPF. This field is not
    * the IMA ID sent in the ICP cells.
    */
   NPF_F_IMA_Group_ID_t        groupID;

   /* IMA protocol version – 1_0 or 1_1. Refer section 2.1.2.2 of Interface
    * Management API Implementation Agreement (ATM Interfaces) Revision 3.0
    * for type definition
    */
   NPF_IfATM_IMA_Ver_t         imaVer;

   /* Minimum number of active receive links to make group operational */
   NPF_uint8_t                 minNumRxLinks;
```

```
/* Minimum number of active transmit links to make group operational */
NPF_uint8_t                 minNumTxLinks;

/* Expected bandwidth in bits per second of the links which may be
 * added to this group. If configured as 0, it indicates that the FAPI
 * implementation may derive this from the first link that is added to the
 * group
 */
NPF_uint32_t                expLinkRate;

/* IMA Group Symmetry Mode. Refer section 2.1.2.2 of Interface Management
 * API Implementation Agreement (ATM Interfaces) Revision 3.0
 * for type definition */
NPF_IfATM_IMA_Symmetry_t    symmetry;

/* Transmit clocking mode – CTC/ITC. Refer section 2.1.2.4 of Interface
 * Management API Implementation Agreement (ATM Interfaces) Revision 3.0
 * for type definition
 */
NPF_IfATM_IMA_Tclock_t      neTxClockMode;

/* Link ID of the default transmit timing reference link
 * The Tx reference link ID specified below is used as a hint by the
 * FAPI implementation to choose the TX timing reference link. If the link
 * link corresponding to the LID hinted below is available, it is selected
 * as the timing reference link. A value of -1 specifies that no hint is
 * being provided by the FAPI client to the FAPI implementation and the
 * LFB/FAPI are free to choose a suitable link as the timing reference
 * link */
NPF_int8_t                  defTxTimingRefLinkLID;


/* IMA ID configured for the near end */
NPF_uint8_t                 txImaID;

/* Frame length to use in transmit direction. Refer section 2.1.2.3 of
 * Interface Management API Implementation Agreement (ATM Interfaces)
 * Revision 3.0 for type definition */
NPF_IfATM_IMA_FrameLength_t   txFrameLength;

/* Maximum tolerated differential delay in milliseconds. Refer section
 * 4.1.16 of ATM Configuration Manager Functional API (Work in progress)
 * for type definition
 */
NPF_F_ATM_Timers_t          diffDelayMax;

/* Alpha value to be used by IFSM */
NPF_F_IMA_AlphaValue_t      alphaValue;

/* Beta value to be used by the IFSM */
NPF_F_IMA_BetaValue_t       betaValue;

/* Gamma value to be used by the IFSM */
NPF_F_IMA_GammaValue_t      gammaValue;

/* Administrative status of the group – UP/DOWN. Refer section
 * 4.1.17 of ATM Configuration Manager Functional API (Work in progress)
 * for type definition
 */
```

```
   NPF_ObjStatus_t             adminStatus;

   /* Configuration for test procedure */
   NPF_F_IMA_Group_Test_Mode_t testMode;

} NPF_F_IMA_Group_Config_t;

/* IMA Group States */
typedef struct {
   /* Status of the group state machines for this group */
   NPF_F_IMA_GSM_State_t       neGroupState;
   NPF_F_IMA_GSM_State_t       feGroupState;

   /* Status of the group traffic state machine for this group */
   NPF_F_IMA_GTSM_State_t      gtsmState;

   /* Whether test procedure disabled or operational */
   NPF_F_IMA_Test_Proc_Status_t testProcStatus;

   /* Status of the test (if operational) – failed/passed */
   NPF_boolean_t              testProcFailed;   /* TRUE/FALSE */

} NPF_F_IMA_Group_State_t;


/* IMA Group Query Information */
typedef struct {
   /* IMA group configuration */
   NPF_F_IMA_Group_Config_t    neGroupConfig;

   /* Status of the state machines for this group */
   NPF_F_IMA_Group_State_t     gsmGtsmState;

   /* FE Transmit clocking mode – CTC/ITC. Refer section 2.1.2.4 of Interface
    * Management API Implementation Agreement (ATM Interfaces) Revision 3.0
    * for type definition
    */
   NPF_IfATM_IMA_Tclock_t      feTxClockMode;

   /* IMA ID configured for the far end */
   NPF_uint8_t                 rxImaID;

   /* Frame length used in receive direction. Refer section 2.1.2.2 of
    * Interface Management  API Implementation Agreement (ATM Interfaces)
    * Revision 3.0 for type definition
    */
   NPF_IfATM_IMA_FrameLength_t   rxFrameLength;

   /* ID of the link in group with least delay */
   NPF_F_IMA_Link_ID_t         leastDelayLinkID;

   /* Link ID of the current transmit timing reference link */
   NPF_F_IMA_Link_ID_t         curTxTimingRefLinkLID;

   /* Link ID of the current receive timing reference link */
   NPF_F_IMA_Link_ID_t         curRxTimingRefLinkLID;

   /* OAM label being Tx – identifies version negotiated/configured */
    NPF_uint8_t               txOamLabel;
```

```
   /* OAM label being Rx – identifies version negotiated/configured */
    NPF_uint8_t              rxOamLabel;

   /* Available cell rate (cells per second) in transmit direction */
    NPF_uint32_t             txAvailCellRate;

   /* Available cell rate (cells per second)in receive direction */
    NPF_uint32_t             rxAvailCellRate;

   /* Test procedure status. This field if valid only if the test procedure
    * is operation on link in this group. When set to NPF_TRUE it indicates
    * that the test procedure failed and the bit map of links on which the
    * test pattern failed to loop back is specified in the testResultBitMap
    * field.
    */
    NPF_boolean_t            testProcFailed;

   /* Bit map indicating the links on which the test pattern failed to loop
    * back. Valid only if the test procedure is operation on this group
    */
    NPF_uint32_t             testResultBitMap;

   /* Number of configured RX links */
   NPF_uint8_t                numRxCfgLinks;

   /* Array of link Ids of Rx links configured for this group */
   NPF_F_IMA_Link_ID_t        *rxCfgLinkArr;

   /* Number of configured TX links */
   NPF_uint8_t                numTxCfgLinks;

   /* Array of link Ids of Tx links configured for this group */
   NPF_F_IMA_Link_ID_t        *txCfgLinkArr;

   /* Number of active RX links */
   NPF_uint8_t                numRxActLinks;

   /* Array of link Ids of active Rx links for this group */
   NPF_F_IMA_Link_ID_t        *rxActLinkArr;

   /* Number of active TX links */
   NPF_uint8_t                numTxActLinks;

   /* Array of link Ids of active Tx links for this group */
   NPF_F_IMA_Link_ID_t        *txActLinkArr;
} NPF_F_IMA_Group_Info_t;

/* IMA Link Configuration */
typedef struct {
   /* The Interface handle of the PDH Link */
   NPF_IfHandle_t             imaIfID;

   /* A unique ID to identify the link. The interface handle for the link
    * is an arbitrary value assigned by the IM APIs. The linkID may be used
    * to provide a fast way to lookup the link information. The FAPI
    * implementations say restrict the range of values assigned to this field
    * or restrictions on the manner in which this number is constructed and
    * any such restrictions are outside the scope of NPF.
```

```
     * This number is not the logical link ID of the link.
     */
    NPF_F_IMA_Link_ID_t         linkID;

    /* Group to which the link is assigned. Value 0 indicate not in a group */
    NPF_F_IMA_Group_ID_t        groupID;

    /* Administrative status of the link – UP/DOWN. Refer section
     * 4.1.17 of ATM Configuration Manager Functional API (Work in progress)
     * for type definition
     */
    NPF_ObjStatus_t             adminStatus;

    /* Logical Link ID (LID) used in Transmit direction. A value of -1
     * assigned to the txLinkId indicates that the FAPI implementation is
     * to choose the LID to be assigned to this link */
    NPF_int8_t                 txLinkId;

    /* ICP cell offset for frames sent on this link. The FAPI client may
     * assign a value of -1 to the icpCellOffset indicating that the
     * FAPI implementation is free to choose the ICP cell offset
     * When configured as -1, the FAPI implementation may choose to distribute
     * ICP cells from link to link withing an IMA group in an uniform fashion
     * across the IMA frame. The mechanism used to select the ICP cell offset
     * by FAPI implementation when the icpCellOffset is set to -1 is outside
     * the scope of NPF
     */
    NPF_uint16_t                 icpCellOffset;
} NPF_F_IMA_Link_Config_t;

/* IMA Link States */
typedef struct {
    /* near end IMA Rx LSM State */
    NPF_F_IMA_LSM_State_t       neRxLinkState;

    /* near end IMA Tx LSM State */
    NPF_F_IMA_LSM_State_t       neTxLinkState;

    /* far end IMA Rx LSM State */
    NPF_F_IMA_LSM_State_t       feRxLinkState;

    /* far end IMA Tx LSM State */
    NPF_F_IMA_LSM_State_t       feTxLinkState;
} NPF_F_IMA_Link_State_t;

/* IMA Link Query Information */
typedef struct {
    /* near end IMA link configuration */
    NPF_F_IMA_Link_Config_t     neLinkConfig;

    /* NE/FE Rx and Tx LSM states */
    NPF_F_IMA_Link_State_t      linkStates;

    /* Logical Link ID (LID) in Receive direction. A value of -1 indicates
     * that the LID is not known */
    NPF_int8_t                  rxLinkId;

    /* Differential delay measured between this link and the link within the
     * IMA group with the least delay. Refer section 4.1.16 of ATM
```

```
    * Configuration Manager Functional API (Work in progress) for type
    * definition
    */
   NPF_F_ATM_Timers_t       relativeDelay;

} NPF_F_IMA_Link_Info_t;

/* IMA Group Statistics */
typedef struct {

   /* Time in seconds for which this group has been in operation state */
    NPF_uint32_t              groupRunningSecs;

   /* Count of one second intervals where the GTSM was unavailable (R136)*/
   NPF_uint32_t        groupUnavailSecs;

   /* Count of near end group failures (R137)*/
   NPF_uint32_t        neNumFailures;

   /* Count of far end group failures (O25)*/
   NPF_uint32_t        feNumFailures;
} NPF_F_IMA_Group_Stats_t;


/* IMA Link Statistics */
typedef struct {
   /* Count of errored, missing, invalid ICP except during
      SES-IMA/UAS-IMA (R125)                                  */
   NPF_uint32_t imaViolations;

   /* Number of OIF anomalies at near end except during
      SES-IMA/UAS-IMA (O20)                                   */
   NPF_uint32_t oifAnomalies;

   /* Count of 1 sec intervals containing > 30% invalid
      IMA, link defects, LIF, or LODS except during UAS-IMA (R126) */
   NPF_uint32_t neSevErroredSecs;

   /* Count of 1 sec intervals containing RDI-IMA defects
      Except during UAS-IMA-FE condition (R127) */
   NPF_uint32_t feSevErroredSecs;

   /* Count of unavailable seconds at near end (R128) */
   NPF_uint32_t neUnavailSecs;

   /* Count of unavailable seconds at far end (R129) */
   NPF_uint32_t feUnavailSecs;

   /* Count of unusable seconds at near end LSM (R130) */
   NPF_uint32_t neTxUnusableSecs;

   /* Count of unusable seconds at near end LSM (R131) */
   NPF_uint32_t neRxUnusableSecs;

   /* Count of seconds with Tx unusable indications from
     far end Tx LSM (R132) */
   NPF_uint32_t feTxUnusableSecs;

   /* Count of seconds with Rx unusable indications from
```

```
     far end Rx LSM (R133) */
   NPF_uint32_t feRxUnusableSecs;

   /* Number of times near end transmit failure alarm
      condition entered (R134)*/
   NPF_uint32_t neTxNumFailures;

   /* Number of times near end receive failure alarm
      condition entered (R135)*/
   NPF_uint32_t neRxNumFailures;

   /* Number of times far end transmit failure alarm
      condition entered (O21)*/
   NPF_uint32_t feTxNumFailures;

   /* Number of times far end receive failure alarm
      condition entered (O22)*/
   NPF_uint32_t feRxNumFailures;

   /* Number of stuff events inserted in tx direction (O-23) */
   NPF_uint32_t txStuffs;

   /* Number of stuff events detected in rx direction (O-24) */
   NPF_uint32_t rxStuffs;

   /* Flag helping the FAPI user to simplify and make the reporting of
      Unavailable Seconds more efficient at 15 minutes PM intervals. The
      Flag indicates the following.

      1) Link is in Available state and did count SES in the last second
         before the statistic query.

      2) Link is in Unavailability state and did not count SES in the last
         Second before the statistic query.

      3) None of 1 or 2.

      The flag set to 1 indicates that Unavailability state is about to be
      Entered and the flag set to 2 indicates that Unavailability state is
      about to be left.

      In both these cases, the FAPI user must do a new query 10 seconds later
      To secure reporting the correct SES and UAS values. When the flag is
      set to 3, the FAPI user can use the SES and UAS counter values directly
      and does not need to make another query 10 seconds later. */

   NPF_uint32_t uasInfoFlag;
} NPF_F_IMA_Link_Stats_t;

/* ICP Query Response Structure */
typedef struct {
   NPF_boolean_t icpValid;     /* Whether ICP cell information valid */
   NPF_uint8_t icp_bytes[48];  /* ICP Cell payload                   */
} NPF_F_IMA_Icp_Cell_t;

/* IMA LFB Attributes query response */
typedef struct {
   NPF_uint32_t   maxNumGroups;          /* Maximum possible IMA groups   */
   NPF_uint32_t   curNumGroups;          /* Current number of IMA groups  */
```

```
   NPF_uint32_t   maxNumLinks;          /* Maximum possible IMA links    */
   NPF_uint32_t   curNumLinks;          /* Current number of IMA links   */
} NPF_F_IMA_LFB_AttrQueryResponse_t;

/* Structures for Completion Callbacks */
/*
* This union is a handy way of representing the various object identifiers
* used by the APIs.
*/
typedef union {
   /* IMA Group ID */
   NPF_F_IMA_Group_ID_t        groupID;

   /* IMA Link ID */
   NPF_F_IMA_Link_ID_t        linkID;
} NPF_F_IMA_Id_t;

/*
* An asynchronous response contains a configuration object ID,
* an error or success code, and in some cases a function-
* specific structure embedded in a union. One or more of
* these is passed to the callback function as an array
* within the callback data structure (below)
*/
typedef struct {/* Asynchronous Response Structure */
    NPF_error_t                error;      /* Error code for this resp    */
    NPF_F_IMA_Id_t             objId;      /* Object Indetifier           */
    union {
        /* NPF_F_IMA_LFB_AttributesQuery() */
        NPF_F_IMA_LFB_AttrQueryResponse_t lfbAttrQueryResponse;

        /* NPF_F_IMA_Link_StatsGet()       */
        NPF_F_IMA_Link_Stats_t           linkStats;

        /* NPF_F_IMA_Link_StateGet()       */
        NPF_F_IMA_Link_State_t           linkState;

        /* NPF_F_IMA_Link_Query()          */
        NPF_F_IMA_Link_Info_t            linkInfo;

        /* NPF_F_IMA_Link_LastICPInfoGet() */
        NPF_F_IMA_Icp_Cell_t             icpCell;

        /* NPF_F_IMA_Group_StatsGet()      */
        NPF_F_IMA_Group_Stats_t          groupStats;

        /* NPF_F_IMA_Group_StateGet()      */
        NPF_F_IMA_Group_State_t          groupState;

        /* NPF_F_IMA_Group_Query()         */
        NPF_F_IMA_Group_Info_t           groupInfo;

       /* NPF_F_IMA_Group_TestSet()       */
        NPF_uint32_t                     testResultBitMap;
    } u;
} NPF_F_IMA_AsyncResponse_t;

/*
* Completion Callback Types, to be found in the callback
```

```
* data structure, NPF_F_IMA_CallbackData_t.
*/
typedef enum NPF_F_IMA_CallbackType {
    /* Function to query IMA LFB attributes */
    NPF_F_IMA_ATTR_QUERY = 1,

    /* Functions for IMA group configuration and management */
    NPF_F_IMA_GROUP_SET = 2,         /* Add or Modify an IMA group       */
    NPF_F_IMA_GROUP_DELETE = 3,      /* Delete an IMA group              */
    NPF_F_IMA_GROUP_ENABLE = 4,      /* Put an IMA group in service      */
    NPF_F_IMA_GROUP_DISABLE = 5,     /* Take an IMA group out of service */
    NPF_F_IMA_GROUP_QUERY = 6,       /* Query config. And states of group*/
    NPF_F_IMA_GROUP_STATS_GET = 7,   /* Query statistics of an IMA group */
    NPF_F_IMA_GROUP_STATE_GET = 8,   /* Query state m/c states of a group*/
    NPF_F_IMA_GROUP_TEST_SET = 9,    /* Start/Stop Test pattern procedure*/

    /* Functions for IMA link configuration and management */
    NPF_F_IMA_LINK_SET = 10,          /* Add or Modify an IMA link        */
    NPF_F_IMA_LINK_DELETE = 11,      /* Delete an IMA link               */
    NPF_F_IMA_LINK_ENABLE = 12,      /* Put an IMA link in service       */
    NPF_F_IMA_LINK_DISABLE = 13,     /* Put an IMA link out of service   */
    NPF_F_IMA_LINK_QUERY = 14,       /* Query config and states of a link*/
    NPF_F_IMA_LINK_STATS_GET = 15,   /* Query statistics of an IMA link  */
    NPF_F_IMA_LINK_STATE_GET = 16,   /* Query state m/c states of a link */
    NPF_F_IMA_LINK_LAST_ICP_GET = 17,/* Get the payload of last ICP
                                      * cell received on queried link  */
} NPF_F_IMA_CallbackType_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_IMA_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_IMA_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_IMA_AsyncResponse_t resp;       /* Response struct              */
```

```
} NPF_F_IMA_CallbackData_t;

/*
*   IMA LFB Event Types
*/
typedef enum {
   /* LIF defect detected at NE for the link */
   NPF_F_IMA_EVENT_LINK_LIF_RAISED = 1,

   /* LIF defect cleared at NE for the link */
   NPF_F_IMA_EVENT_LINK_LIF_CLEARED = 2,

   /* LODS defect detected at NE for the link */
   NPF_F_IMA_EVENT_LINK_LODS_RAISED = 3,

   /* LODS defect cleared at NE for the link */
   NPF_F_IMA_EVENT_LINK_LODS_CLEARED = 4,

     /* RDI-IMA defect detected at NE for the link */
   NPF_F_IMA_EVENT_LINK_RFI_RAISED = 5,

   /* RDI-IMA defect detected at NE for the link */
   NPF_F_IMA_EVENT_LINK_RFI_CLEARED = 6,

   /* Tx link found to be not connected to matching IMA unit at FE */
   NPF_F_IMA_EVENT_LINK_TX_MISCONNECT_RAISED = 7,

   /* Tx link misconnection cleared  */
   NPF_F_IMA_EVENT_LINK_TX_MISCONNECT_CLEARED = 8,

   /* Rx link found to be not connected to matching IMA unit at FE */
   NPF_F_IMA_EVENT_LINK_RX_MISCONNECT_RAISED = 9,

   /* Rx link misconnection cleared  */
   NPF_F_IMA_EVENT_LINK_RX_MISCONNECT_CLEARED = 10,

   /* Implementation specific Tx fault raised */
   NPF_F_IMA_EVENT_LINK_TX_FAULT_RAISED = 11,

   /* Implementation specific Tx fault cleared */
   NPF_F_IMA_EVENT_LINK_TX_FAULT_CLEARED = 12,

   /* Implementation specific Rx fault raised */
   NPF_F_IMA_EVENT_LINK_RX_FAULT_RAISED = 13,

   /* Implementation specific Rx fault cleared */
   NPF_F_IMA_EVENT_LINK_RX_FAULT_CLEARED = 14,

   /* FE reports Tx link unusable */
   NPF_F_IMA_EVENT_LINK_TX_UNUSABLE_FE_RAISED = 15,

   /* FE reports Tx link usable/active */
   NPF_F_IMA_EVENT_LINK_TX_UNUSABLE_FE_CLEARED = 16,

   /* FE reports Rx link unusable */
   NPF_F_IMA_EVENT_LINK_RX_UNUSABLE_FE_RAISED = 17,

   /* FE reports Rx link usable/active */
   NPF_F_IMA_EVENT_LINK_RX_UNUSABLE_FE_CLEARED = 18,
```

```
/* Test pattern failed to loop on specified link */
NPF_F_IMA_EVENT_LINK_TEST_LINK_FAIL_RAISED = 19,

/* Test link failure condition on specified link cleared */
NPF_F_IMA_EVENT_LINK_TEST_LINK_FAIL_CLEARED = 20,

/* Event to notify change in near end link state machine transition */
NPF_F_IMA_EVENT_LINK_STATE_MACHINE_TRANSITION = 21,

/* Far end group in startup state */
NPF_F_IMA_EVENT_GROUP_STARTUP_FE_RAISED_RAISED = 22,

/* Far end tried to use unacceptable configuration params */
NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED_RAISED = 23,

/* Far end uses new acceptable configuration params */
NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED_CLEARED = 24,

/* Far end reports unacceptable configuration params */
NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED_FE_RAISED = 25,

/* Far end accepts new configuration params */
NPF_F_IMA_EVENT_GROUP_CONFIG_ABORTED_FE_CLEARED = 26,

/* Less than P(tx) or P(rx) links are active */
NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS_RAISED = 27,

/* Condition where less than P(tx) or P(rx) links are active cleared */
NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS_CLEARED = 28,

/* Far end reports less than P(rx) or P(tx) links are active */
NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS_FE_RAISED = 29,

/* Condition where Far end reports less than P(rx) or P(tx)
  links are active cleared */
NPF_F_IMA_EVENT_GROUP_INSUFFICIENT_LINKS_FE_CLEARED = 30,

/* Far end reports that it is blocked */
NPF_F_IMA_EVENT_GROUP_BLOCKED_FE_RAISED = 31,

/* Far end reports that blocking no longer exists */
NPF_F_IMA_EVENT_GROUP_BLOCKED_FE_CLEARED = 32,

/* Far end transmit clock mode is different than NE transmit clock mode */
NPF_F_IMA_EVENT_GROUP_TIMING_MISMATCH_RAISED = 33,

/* Mismatch of far end transmit clock mode and NE transmit clock mode
 * cleared                                                          */
NPF_F_IMA_EVENT_GROUP_TIMING_MISMATCH_CLEARED = 34,

/* Test pattern failed to loop on some links */
NPF_F_IMA_EVENT_GROUP_TEST_LINK_FAIL_RAISED = 35,

/* Test link failure condition cleared on all link */
NPF_F_IMA_EVENT_GROUP_TEST_LINK_FAIL_CLEARED = 36,

/* Event to notify change in near end group state machine transition */
NPF_F_IMA_EVENT_GROUP_STATE_MACHINE_TRANSITION = 37,
```

```
   /* Event to notify change in near end group traffic state machine
      transition */
   NPF_F_IMA_EVENT_GROUP_TRAFFIC_STATE_MACHINE_TRANSITION = 38,

} NPF_F_IMA_Event_t;
/*
 *   IMA LFB 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_IMA_Event_t     eventType;   /* Type of event reported        */
    NPF_F_IMA_Id_t        objId;       /* Object for which event raised */

    union {
        /* Link states – filled for link specific events       */
        NPF_F_IMA_Link_State_t           linkState;

        /* Group states – filled for group specific events     */
        NPF_F_IMA_Group_State_t          groupState;
    } u;
} NPF_F_IMA_EventData_t;

/* Completion Callback Function */
typedef void (*NPF_F_IMA_CallbackFunc_t) (
      NPF_IN NPF_userContext_t            userContext,
      NPF_IN NPF_correlator_t             correlator,
      NPF_IN NPF_F_IMA_CallbackData_t     data);

/* Completion Callback Registration Function */
NPF_error_t NPF_F_IMA_Register (
      NPF_IN NPF_userContext_t                  userContext,
      NPF_IN NPF_F_IMA_CallbackFunc_t callbackFunc,
      NPF_OUT NPF_callbackHandle_t        *callbackHandle);

/* Completion Callback Deregistration Function */
NPF_error_t NPF_F_IMA_Deregister (
      NPF_IN NPF_callbackHandle_t   callbackHandle);

/* Event Handler Function */
typedef void (*NPF_F_IMA_EventCallFunc_t) (
    NPF_IN  NPF_userContext_t      userContext,
    NPF_IN  NPF_uint32_t           nEvent,
    NPF_IN  NPF_F_IMA_EventData_t  *imaEventArray);

/* Event Registration Function */
NPF_error_t NPF_F_IMA_EventHandler_Register(
          NPF_IN  NPF_userContext_t         userContext,
          NPF_IN  NPF_F_IMA_EventCallFunc_t imaEvtCallFn,
            NPF_IN  NPF_eventMask_t           imaEvtMask,
          NPF_OUT NPF_callbackHandle_t      *imaEvtCallHdl);
/* Event Handler Deregistration Function */
NPF_error_t NPF_F_IMA_EventHandler_Deregister(
          NPF_IN NPF_callbackHandle_t     imaEventCallHandle);

/* LFB Attributes Query Function */
NPF_error_t NPF_F_IMA_LFB_AttributesQuery (
      NPF_IN NPF_callbackHandle_t   callbackHandle,
```

```
      NPF_IN NPF_correlator_t       correlator,
      NPF_IN NPF_errorReporting_t   errorReporting,
      NPF_IN NPF_FEHandle_t         feHandle,
      NPF_IN NPF_BlockId_t          blockId);

/* Add or Modify an IMA group */
NPF_error_t NPF_F_IMA_GroupSet (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t           feHandle,
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Group_Config_t   *cfgArray);
/* Delete an IMA group */
NPF_error_t NPF_F_IMA_GroupDelete (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_boolean_t              delContainedLnks,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Group_ID_t       *delArray);

/* Put an IMA group in service */
NPF_error_t NPF_F_IMA_GroupEnable (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Group_ID_t       *enaArray);
/* Put an IMA group out of service */
NPF_error_t NPF_F_IMA_GroupDisable (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Group_ID_t       *enaArray);
/* Query an IMA Group */
NPF_error_t NPF_F_IMA_GroupQuery (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Group_ID_t       *grpIdArr);

/* Get statistics accumulated for an IMA Group */
NPF_error_t NPF_F_IMA_GroupStatsGet (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
```

```
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_boolean_t          resetStats,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Group_ID_t       *grpIdArr);

/* Get state information for an IMA Group */

NPF_error_t NPF_F_IMA_GroupStateGet (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Group_ID_t       *grpIdArr);

/* Configure Test Pattern procedure for an IMA group */
NPF_error_t NPF_F_IMA_GroupTestSet (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Group_Test_Proc_Config_t  *grpIdArr);

/* Add or Modify an IMA link */
NPF_error_t NPF_F_IMA_LinkSet (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Link_Config_t   *cfgArray);
/* Delete an IMA Link */
NPF_error_t NPF_F_IMA_LinkDelete (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Link_ID_t       *delArray);
/* Put an IMA link in service */
NPF_error_t NPF_F_IMA_LinkEnable (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
      NPF_IN NPF_BlockId_t            blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Link_ID_t       *enaArray);
/* Put an IMA link out of service */
NPF_error_t NPF_F_IMA_LinkDisable (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
      NPF_IN NPF_FEHandle_t          feHandle,
```

```
        NPF_IN NPF_BlockId_t          blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Link_ID_t       *enaArray);
/* Query an IMA Link */
NPF_error_t NPF_F_IMA_LinkQuery (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
        NPF_IN NPF_FEHandle_t        feHandle,
        NPF_IN NPF_BlockId_t          blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Link_ID_t       *lnkIdArr);

/* Get statistics accumulated for an IMA Link */
NPF_error_t NPF_F_IMA_LinkStatsGet (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
        NPF_IN NPF_FEHandle_t        feHandle,
        NPF_IN NPF_boolean_t              resetStats,
        NPF_IN NPF_BlockId_t          blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Link_ID_t       *lnkIdArr);

/* Get state information for an IMA Link */
NPF_error_t NPF_F_IMA_LinkStateGet (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
        NPF_IN NPF_FEHandle_t        feHandle,
        NPF_IN NPF_BlockId_t          blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Link_ID_t       *lnkIdArr);

/* Get last received ICP cell for an IMA Link */
NPF_error_t NPF_F_IMA_LinkLastICPInfoGet (
        NPF_IN NPF_callbackHandle_t       cbHandle,
        NPF_IN NPF_correlator_t           cbCorrelator,
        NPF_IN NPF_errorReporting_t       errorReporting,
        NPF_IN NPF_FEHandle_t        feHandle,
        NPF_IN NPF_BlockId_t          blockId,
        NPF_IN NPF_uint32_t           numEntries,
        NPF_IN NPF_F_IMA_Link_ID_t       *lnkIdArr);

#ifdef __cplusplus
}
#endif

#endif /* __NPF_F_ATM_IMA_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:

Pat Carr, Wintegra
Pål Dammvik, Ericsson
Stephen Doyle, Intel
Conor Ferguson, Intel
Patrik Herneld, Ericsson
Ajay Kamalvanshi, Nokia
Jaroslaw Kogut, Intel
Arthur Mackay, Freescale
Michael Persson, Ericsson
Tiberu Petrica, Freescale
John Renwick, Agere Systems
Vedvyas Shanbhogue (ed.), Intel
Roger Smith, Wintegra
Keith Williamson, Motorola
Paul Wilson, Freescale
Weislaw Wisniewski, Intel
Per Wollbrand, Ericsson

Appendix C List of companies belonging to NPF during approval process

Agere Systems
AMCC
Analog Devices
Cypress Semiconductor
Enigma Semiconductor
Ericsson
Flextronics
Freescale Semiconductor
HCL Technologies
Hifn
IDT
Infineon Technologies AG
Intel
IP Fabrics
IP Infusion
Motorola
Mercury Computer Systems
Nokia
NTT Electronics
PMC-Sierra
Sensory Networks
Sun Microsystems
Teja Technologies
TranSwitch
U4EA Group
Wintegra
Xelerated
Xilinx