



# ATM Policer LFB and Functional API Implementation Agreement

August 16, 2005  
Revision 1.0

**Editor:**

**Vedvyas Shanbhogue, Intel, [vedvyas.shanbhogue@intel.com](mailto:vedvyas.shanbhogue@intel.com)**

Copyright © 2005 The Network Processing Forum (NPF). All Rights Reserved.

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

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

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

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

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

## Table of Contents

1	Revision History .....	3
2	Introduction.....	4
	2.1 Acronyms.....	4
	2.2 Assumptions.....	4
	2.3 Scope .....	4
	2.4 External Requirements and Dependencies.....	4
3	ATM Policer Description.....	6
	3.1 ATM Policer Inputs .....	7
	3.2 ATM Policer Outputs.....	7
	3.3 Accepted Inputs .....	8
	3.4 Cell Modifications .....	8
	3.5 Relationship with Other LFBs .....	8
4	Data Types .....	10
	4.1 Common LFB Data Types .....	10
	4.2 Data Structures for Completion Callbacks .....	10
	4.3 Data Structures for Event Notifications.....	11
	4.4 Error Codes .....	11
5	Functional API (FAPI).....	13
	5.1 Required Functions .....	13
	5.2 Conditional Functions.....	13
	5.3 Optional Functions.....	15
6	References.....	16
	Appendix A Header File Information.....	17
	Appendix B Acknowledgements.....	19
	Appendix C List of companies belonging to NPF during approval process.....	20

## Table of Figures

Figure 3.1:	ATM Policer LFB .....	6
Figure 3.2:	Policer Instances.....	6
Figure 3.3:	Cooperation between ATM Policer and ATM Header Classifier LFB .....	8

## List of Tables

Table 3.1:	ATM Policer LFB Inputs.....	7
Table 3.2:	Input Metadata for ATM Policer LFB.....	7
Table 3.3:	ATM Policer LFB Outputs .....	7
Table 3.4:	Output Metadata for ATM Policer LFB.....	7
Table 4.1:	Callback type to callback data mapping table.....	11

## 1 Revision History

Revision	Date	Reason for Changes
1.0	08/16/2005	Rev 1.0 of the ATM Policer LFB and Functional API Implementation Agreement. Source: npf2004.155.15.

## 2 Introduction

This contribution defines the ATM Policer LFB and lists configurations that are required in the LFB.

### 2.1 Acronyms

- **ABR**: Available Bit Rate
- **ATM**: Asynchronous Transfer Mode
- **API**: Application Programming Interface
- **CLP**: Cell Loss Priority
- **CBR**: Constant Bit Rate
- **CDVT**: CDV Tolerance
- **FAPI**: Functional API
- **GFR**: Guaranteed Frame Rate
- **IA**: Implementation Agreement
- **ID**: Identifier
- **LFB**: Logical Functional Block
- **MBS**: Maximum Burst Size
- **MCR**: Minimum Cell Rate
- **MFS**: Maximum Frame Size
- **NPC**: Network Parameter Control
- **Nrt-VBR**: Non-Real-time VBR
- **NNI**: Network Node Interface
- **PCR**: Peak Cell Rate
- **PTI**: Payload Type Indicator
- **SCR**: Sustainable Cell Rate
- **SDU**: Service Data Unit
- **TM**: Traffic Management
- **UBR**: Unspecified Bit Rate
- **UNI**: User Network Interface
- **UPC**: Usage Parameter Control
- **VBR**: Variable Bit Rate

### 2.2 Assumptions

The ATM Policer LFB obtains its configurations from the ATM Configuration Manager Functional API implementation. The mechanism used to obtain this configuration is not in the scope of NPF.

### 2.3 Scope

This IA describes the configurations required by the LFB for policing cells received on ATM virtual links. 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
- ATM Software API Architecture Framework Implementation Agreement Revision 1.0 defines the architectural framework for the ATM FAPIs.
- ATM Configuration Manager Functional API Implementation Agreement Revision 1.0 defines the functions to configure and manage ATM LFBs on a forwarding element.

### 3 ATM Policer Description

The ATM Policer LFB monitors received ATM SDU from the previous LFB in the pipeline to ensure conformance to the traffic contract. The ATM policer LFB may be used to perform UPC or NPC functions on the received cell stream. The ATM policer uses the traffic parameters specified in the traffic descriptor (used by the source of the cell stream on this virtual link) to perform the UPC or NPC function. The ATM policer may perform following actions to ensure conformance of the virtual link to negotiated traffic contract:

- Cell passing
- Cell Tagging by changing loss priority to 1 from 0
- Cell discarding

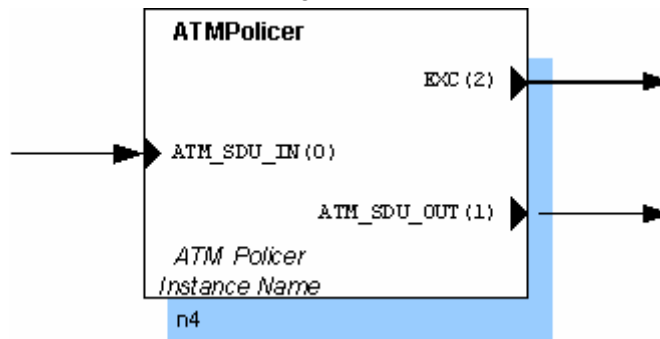
The ATM policer may also be configured to perform frame discard by examining the payload type of the ATM SDU using below schemes:

- Partial Packet Discard (PPD)
- Early Packet Discard (EPD)

The ATM Policer LFB maintains the following counters for each policer:

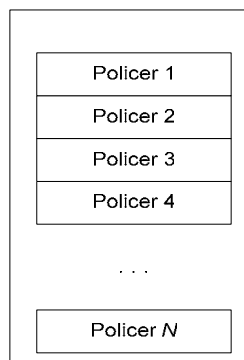
- Number of ATM cells tagged, i.e., loss priority changed from CLP=0 to CLP=1
- Number of ATM cells with CLP=0 discarded
- Number of ATM cells with CLP=0+1 discarded

The ATM Policer LFB is modeled as shown in Figure 3.1



**Figure 3.1: ATM Policer LFB**

The LFB may contain multiple instances of policers. A policer instance is associated with a virtual (VP or VC) link configured in the system. Incoming ATM SDU are assigned to appropriate policer instance according to metadata received with ATM SDU. Such instances are depicted below in Figure 3.2. The maximum number of policers is an attribute of the ATM Policer LFB and may be queried as such.



**Figure 3.2: Policer Instances**

### 3.1 ATM Policer Inputs

**Table 3.1: ATM Policer LFB Inputs**

Symbolic Name	Input ID	Description
ATM_SDU_IN	0	This is the only input for the ATM Policer LFB and is used to receive ATM SDUs to be monitored by the policer associated with the VP and/or the VC Link ID signaled in the metadata.

#### 3.1.1 Metadata Required

**Table 3.2: Input Metadata for ATM Policer LFB**

Metadata tag	Access method	Description
META_VPL_ID	Read	Metadata identifying the VP link on which the ATM cell was received.
META_VCL_ID	Read	Metadata identifying the VC link on which the ATM cell was received. This metadata is received only when the VP link is terminated.
META_ATM_PTI	Read	Payload Type of received ATM cell
META_ATM_LP	Read/Re-write	Loss Priority of the received ATM cells. The loss priority is re-written if required based on the policing actions.

### 3.2 ATM Policer Outputs

**Table 3.3: ATM Policer LFB Outputs**

Symbolic Name	Output ID	Description
ATM_SDU_OUT	1	This is the normal output for the ATM Policer LFB. ATM SDUs that are conformant to the traffic contract established are passed on this output.
EXC	2	The cell is sent to this output if the cell violates the established traffic contract for the VP/VC Link and is discarded.

#### 3.2.1.1 Metadata Produced

**Table 3.4: Output Metadata for ATM Policer LFB**

Metadata tag	Access method	Description
META_ATM_LP	Re-write	Loss Priority of the received ATM cell is rewritten if the cell is tagged.

### 3.3 Accepted Inputs

The ATM Policer LFB can accept any ATM SDUs received over UNI or NNI.

### 3.4 Cell Modifications

The ATM SDUs received by the ATM Policer LFB are not subject to any modification and are not consumed by this LFB and always exit through one of the outputs. The ATM Policer LFB processes ATM SDUs entering the LFB input sequentially. That means that ATM Policer LFB does not change the order of transmission of the ATM SDUs.

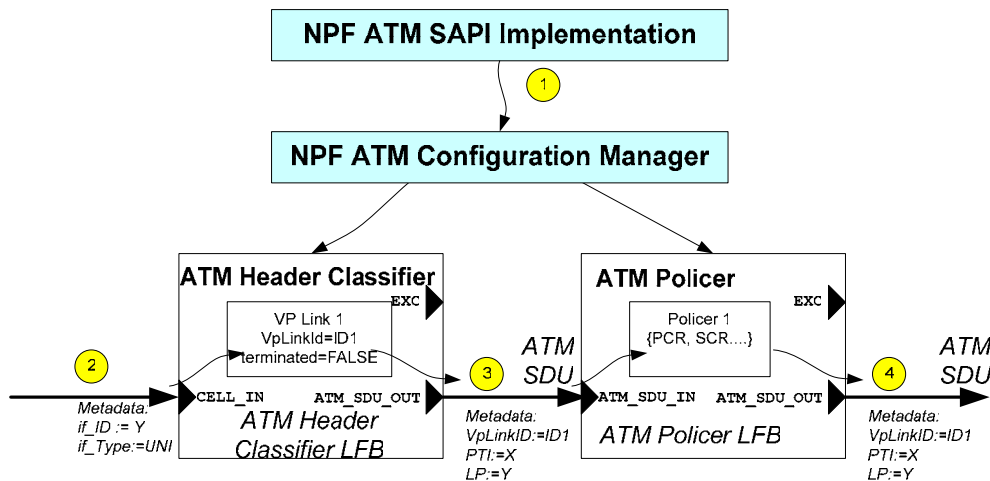
### 3.5 Relationship with Other LFBs

The ATM Policer LFB is placed in the processing chain after the ATM Header Classifier LFB. The ATM Policer LFB receives primarily ATM SDUs from previous LFB and passes them to the next LFB in chain after monitoring for traffic contract violations.

The ATM Policer LFB may be preceded in the topology by any LFB that can produce the information required by the ATM Policer LFB at its input. Downstream (not necessarily next) of the ATM Policer LFB, there should be LFBs that can utilize the information generated at output by ATM Policer LFB. The exact design and connections between the ATM Policer LFB and cooperating blocks is specific to the vendor that provides Forwarding Element design and FAPI implementation.

The EXC output of the ATM Policer LFB could be connected to an LFB that receives SDUs which violate the established traffic contract for the virtual link. Depending on system design this may be either dropper, which drops SDUs or other LFB that makes a decision how to utilize such SDUs.

The sequence of actions that configures ATM Policer LFB and cooperating ATM Header Classifier LFB instance, and cooperation between these two LFBs is schematically depicted in Figure 3.3.



**Figure 3.3: Cooperation between ATM Policer and ATM Header Classifier LFB**

This figure shows part of example Forwarding Element that contains ATM Header Classifier LFB and ATM Policer LFBs. These two blocks are connected in chain and configured by the ATM Configuration manager LFB. The sequence of actions that configure a virtual link instance in ATM Header Classifier and a policer may be defined as follows (see corresponding numbers in circles in the figure):

1. The NPF ATM SAPI is invoked to create a switched VP Link. The system software below the NPF ATM SAPI assigns a VP Link ID 'ID1' to this VP link and invokes the ATM configuration manager FAPI to create the VP Link. This causes a switched VP Link instance to be created in the ATM Header Classifier LFB to map the ATM header and incoming interface to VP Link ID 'ID1'. An ATM policer instance is created in the ATM Policer LFB to monitor the traffic received on the VP Link with ID 'ID1'.

2. The ATM Header Classifier LFB receives an ATM cell from the Ingress ATM TC LFB. The ATM Header Classifier LFB uses the VPI from the ATM header and the interface ID passed in the metadata to identify the VP link instance on which the cell as received as VP Link with ID 'ID1'. As this is a switched VP Link, the VCI is not analyzed further and the ATM SDU is sent to the ATM Policer LFB with the VP Link ID 'ID1' in the metadata.
3. The ATM SDU is forwarded to the ATM Policer LFB along with the PTI and LP extracted from the ATM header over the `ATM_SDU_OUT` output of the ATM Header Classifier LFB. The ATM Policer LFB uses the VP Link ID 'ID1' to determine the policer instance associated with this VP Link and performs the policing action.
4. The ATM SDU is forwarded by the ATM Policer LFB to the next LFB in the chain for further processing.

## 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 ATM Policer LFB in a forwarding element using a block type value for the ATM Policer LFB.

```
#define NPF_F_ATMPOLICER_LFB_TYPE 35
```

#### 4.1.2 ATM Policer Configurations

The ATM Policer LFB requires the below configurations for each virtual channel link.

- Virtual Link ID
- Virtual Link type – VP link / VC link
- Service category of the connection - CBR, rt-VBR, nrt-VBR, ABR, UBR, GFR, UBR with PCR, UBR without PCR, UBR with minimum desired cell rate (MDCR), UBR with MDCR and PCR, other
- Peak cell rate (PCR)
- Sustainable cell rate (SCR)
- Maximum burst size (MBS)
- Minimum cell rate (MCR)
- Maximum frame size (MFS)
- Cell delay variation tolerance (CDVT)
- Priority associated with UBR service
- UPC cell drop policy – Drop CLP=0 cells, Drop CLP=1 and CLP=0 cells, Drop frames using EPD, Drop frames using PPD
- Tagging option – Whether tagging of CLP=0 cells enabled

### 4.2 Data Structures for Completion Callbacks

#### 4.2.1 ATM Policer LFB Attributes query response

The attributes of an ATM Policer LFB are the following:

```
typedef struct {
    NPF_uint32_t    maxPolicers;           /* Maximum possible policers */
    NPF_uint32_t    curNumPolicers;       /* Current number of policers */
} NPF_F_ATMPolicerLFB_AttrQueryResponse_t;
```

The `maxPolicers` field contains the maximum number of policers supported in this ATM Policer LFB. The `curNumPolicers` field contains the number of policers currently established in the ATM Policer LFB.

#### 4.2.2 Asynchronous Response

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

```
/*
 * An asynchronous response contains an error or success code, and in some
 * cases a function specific structure embedded in a union.
 */
typedef struct { /* Asynchronous Response Structure */
    NPF_F_ATMPolicerErrorType_t error; /* Error code for this response */
    union {
        /* NPF_F_ATMPolicerLFB_AttributesQuery() */
    }
};
```

```

        NPF_F_ATMPolicerLFB_AttrQueryResponse_t      lfbAttrQueryResponse;
    } u;
} NPF_F_ATMPolicerAsyncResp_t;

```

### 4.2.3 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_ATMPolicerCallbackData_t.
 */
typedef enum NPF_F_ATMPolicerCallbackType {
    NPF_F_ATMPOLICER_ATTR_QUERY = 1,          /* ATM Policer LFB Attr Query */
} NPF_F_ATMPolicerCallbackType_t;

```

#### 4.2.3.1 Callback Data

An asynchronous response contains an error or success code and a function-specific structure embedded in a union in the NPF\_F\_ATMPolicerCallbackData\_t structure.

```

/*
 * The callback function receives the following structure containing
 * of an asynchronous responses from a function call.
 * For the completed request, the error code is specified in the
 * NPF_ATMPolicerAsyncResp_t structure, along with any other information
 */
typedef struct {
    NPF_F_ATMPolicerCallbackType_t type; /* Which function was called? */
    NPF_IN NPF_BlockId_t      blockId; /* ID of LFB generating callback */
    NPF_F_ATMPolicerAsyncResp_t resp; /* Response structures */
} NPF_ATMPolicerCallbackData_t;

```

The callback data that returned for different callback types is summarized in Table 4.1.

**Table 4.1: Callback type to callback data mapping table**

Callback Type	Callback Data
NPF_F_ATMPOLICER_ATTR_QUERY	NPF_F_ATMPolicerLFB_AttrQueryResponse_t

## 4.3 Data Structures for Event Notifications

### 4.3.1 Event Notification Types

None

### 4.3.2 Event Notification Structures

None

## 4.4 Error Codes

### 4.4.1 Common NPF Error Codes

The common error codes that are returned by ATM Policer 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).
- NPF\_E\_RESOURCE\_EXISTS - A duplicate request to create a resource was detected. No new resource was created.
- NPF\_E\_RESOURCE\_NONEXISTENT - A duplicate request to destroy or free a resource was detected. The resource was previously destroyed or never existed.

#### 4.4.2 LFB Specific Error Codes

This section defines ATM Policer configuration and management APIs error codes. These codes are used in callbacks to deliver results of the requested operations.

```

/* Asynchronous error codes (returned in function callbacks) */
typedef NPF_uint32_t NPF_F_ATMPolicerErrorType_t;
#define NPF_ATMPOLICER_BASE_ERR (NPF_F_ATMPOLICER_LFB_TYPE * 100)
#define ATMPOLICER_ERR(n) ((NPF_F_ATMPolicerErrorType_t)\
                           (NPF_ATMPOLICER_BASE_ERR+(n))

/* LFB ID is not an ID of LFB that has ATM Policer functionality */
#define NPF_E_ATMPOLICER_INVALID_ATMPOLICER_BLOCK_ID ATMPOLICER_ERR(0)

```

## 5 Functional API (FAPI)

### 5.1 Required Functions

None

### 5.2 Conditional Functions

The conditional API functions for registration and de-registration of the completion callback functions need to be implemented if any of the optional functions defined for this LFB are implemented.

#### 5.2.1 Completion Callback Function

```
typedef void (*NPF_F_ATMPolicerCallbackFunc_t) (
    NPF_IN NPF_userContext_t          userContext,
    NPF_IN NPF_correlator_t          correlator,
    NPF_IN NPF_F_ATMPolicerCallbackData_t data);
```

##### 5.2.1.1 Description

This callback function is for the application to register an asynchronous response handling routine to the ATM Policer API implementation. This callback function is intended to be implemented by the application, and be registered to the ATM Policer API implementation through the `NPF_F_ATMPolicerRegister` function.

##### 5.2.1.2 Input Parameters

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

##### 5.2.1.3 Output Parameters

None

##### 5.2.1.4 Return Values

None

#### 5.2.2 Completion Callback Registration Function

```
NPF_error_t NPF_F_ATMPolicerRegister(
    NPF_IN NPF_userContext_t          userContext,
    NPF_IN NPF_F_ATMPolicerCallbackFunc_t callbackFunc,
    NPF_OUT NPF_callbackHandle_t      *callbackHandle);
```

##### 5.2.2.1 Description

This function is used by an application to register its completion callback function for receiving asynchronous responses related to ATM Policer 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, and will output a handle that is already assigned to the pair, and will return `NPF_E_ALREADY_REGISTERED`.

### 5.2.2.2 Input Parameters

- `userContext` – A context item for uniquely identifying the context of the application registering the completion callback function. The exact value will be provided back to the registered completion callback function as its first parameter when it is called. 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.

### 5.2.2.3 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 ATM Policer API functions. It will also be used when deregistering the `userContext` and `callbackFunc` pair.

### 5.2.2.4 Return Values

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

### 5.2.2.5 Notes

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

## 5.2.3 Completion Callback Deregistration Function

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

### 5.2.3.1 Description

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

### 5.2.3.2 Input Parameters

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

### 5.2.3.3 Output Parameters

None

### 5.2.3.4 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.

### 5.2.3.5 Notes

- This API function MAY be invoked by any application no longer interested in receiving asynchronous responses for ATM Policer 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.

## 5.3 Optional Functions

### 5.3.1 LFB Attributes Query Function

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

#### 5.3.1.1 Description

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

#### 5.3.1.2 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 FE Handle returned by `NPF_F_topologyGetFEInfoList()` call.
- `blockId` - The unique identification of the ATM Policer LFB.

#### 5.3.1.3 Output Parameters

None

#### 5.3.1.4 Return Values

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

#### 5.3.1.5 Asynchronous Response

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

- `NPF_NO_ERROR` - Operation completed successfully.
- `NPF_E_ATMPOLICER_INVALID_ATMPOLICER_BLOCK_ID` - LFB ID is not an ID of LFB that has ATM Policer functionality.

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

## 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](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\\_1A.pdf](http://www.npforum.org/techinfo/APIConventions2_1A.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.

**Appendix A Header File Information**

```

/*
 * This header file defines typedefs, constants and structures
 * for the NP Forum ATM Policer Functional API
 */
#ifndef __NPF_F_ATM_POLICER_H__
#define __NPF_F_ATM_POLICER_H__

#ifdef __cplusplus
extern "C" {
#endif
/* It is possible to use the FAPI Topology Discovery
   APIs to discover an ATM Policer LFB
   in a forwarding element. */
#define NPF_F_ATMPOLICER_LFB_TYPE 35

/* Asynchronous error codes (returned in function callbacks) */
typedef NPF_uint32_t NPF_F_ATMPolicerErrorType_t;
#define NPF_ATMPOLICER_BASE_ERR (NPF_F_ATMPOLICER_LFB_TYPE * 100)
#define ATMPOLICER_ERR(n) ((NPF_F_ATMPolicerErrorType_t)\
                          (NPF_ATMPOLICER_BASE_ERR+(n)))

/* LFB ID is not an ID of LFB that has ATM Policer functionality */
#define NPF_E_ATMPOLICER_INVALID_ATMPOLICER_BLOCK_ID ATMPOLICER_ERR(0)

/*****
 * Enumerations and types for ATM Policer attributes and
 * completion callback data types
 *****/

/* The attributes of an ATM Policer LFB */
typedef struct {
    NPF_uint32_t    maxPolicers;           /* Maximum possible policers */
    NPF_uint32_t    curNumPolicers;       /* Current number of policers */
} NPF_F_ATMPolicerLFB_AttrQueryResponse_t;

/*
 * An asynchronous response contains an error or success code, and in some
 * cases a function specific structure embedded in a union.
 */
typedef struct { /* Asynchronous Response Structure */
    NPF_F_ATMPolicerErrorType_t error; /* Error code for this response */
    union {
        /* NPF_F_ATMPolicerLFB_AttributesQuery() */
        NPF_F_ATMPolicerLFB_AttrQueryResponse_t    lfbAttrQueryResponse;
    } u;
} NPF_F_ATMPolicerAsyncResp_t;

/*
 * Completion Callback Types, to be found in the callback
 * data structure, NPF_F_ATMPolicerCallbackData_t.
 */
typedef enum NPF_F_ATMPolicerCallbackType {
    NPF_F_ATMPOLICER_ATTR_QUERY = 1, /* ATM Policer LFB Attr Query */
} NPF_F_ATMPolicerCallbackType_t;

/*
 * The callback function receives the following structure containing

```

```

* of an asynchronous responses from a function call.
* For the completed request, the error code is specified in the
* NPF_ATMPolicerAsyncResponse_t structure, along with any other information
*/
typedef struct {
    NPF_F_ATMPolicerCallbackType_t type; /* Which function was called? */
    NPF_IN NPF_BlockId_t          blockId; /* ID of LFB generating callback */
    NPF_F_ATMPolicerAsyncResp_t resp; /* Pointer to response structures*/
} NPF_ATMPolicerCallbackData_t;

/* Type for a callback function to be registered with ATM policer */
typedef void (*NPF_F_ATMPolicerCallbackFunc_t) (
    NPF_IN NPF_userContext_t      userContext,
    NPF_IN NPF_correlator_t       correlator,
    NPF_IN NPF_F_ATMPolicerCallbackData_t data);

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

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

/* LFB Attributes Query Function */
NPF_error_t NPF_F_ATMPolicerLFB_AttributesQuery(
    NPF_IN NPF_callbackHandle_t  callbackHandle,
    NPF_IN NPF_correlator_t      correlator,
    NPF_IN NPF_errorReporting_t  errorReporting,
    NPF_IN NPF_FE_Handle_t       feHandle,
    NPF_IN NPF_BlockId_t         blockId);
#ifdef __cplusplus
}
#endif

#endif /* __NPF_F_ATM_POLICER_H__ */

```

## **Appendix B    Acknowledgements**

**Working Group Chair:** Alex Conta

**Task Group Chair:** Per Wollbrand

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

The list is in alphabetical order of last names:

Pål Damnvik, Ericsson  
Patrik Herneld, Ericsson  
Ajay Kamalvanshi, Nokia  
Jaroslaw Kogut, Intel  
Arthur Mackay, Freescale  
Stephen Nadas, Ericsson  
Michael Persson, Ericsson  
John Renwick, Agere Systems  
Vedvyas Shanbhogue (ed.), Intel  
Keith Williamson, Motorola  
Weislaw Wisniewski, Intel  
Per Wollbrand, Ericsson

## **Appendix C      List of companies belonging to NPF during approval process**

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