ATM Header Classifier LFB and Functional API Implementation Agreement

April 11, 2005
Revision 1.0

Editor:

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

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

Table of Contents

Table of Figures

List of Tables

1 Revision History

Revision	Date	Reason for Changes
1.0	4/11/2005	Rev 1.0 of the ATM Header Classifier LFB and Functional API Implementation Agreement. Source: npf2004.151.14.

2 Introduction

This implementation agreement defines the ATM header classifier and lists the configurations required by the LFB.

2.1 Acronyms / Definitions

- **AAL:** ATM Adaptation Layer
- **ATM**: Asynchronous Transfer Mode
- **CLP:** Cell Loss Priority
- **FE:** Forwarding Element
- **IA:** Implementation Agreement
- **ID:** Identifier
- **NNI:** Network Node Interface
- **PTI:** Payload Type Indicator
- **PVC:** Permanent Virtual Connection
- **UNI:** User Network Interface
- **VC:** Virtual Connection
- **VCC:** Virtual Channel Connection
- **VCI:** Virtual Channel Identifier
- **VPC:** Virtual Path Connection
- **VPI:** Virtual Path Identifier

2.2 Assumptions

The ATM header classifier 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 VP/VC links and interfaces. 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 below type definitions
  - `NPF_error_t` – Refer section 5.2 of Software API Conventions IA Rev 2.0
  - `NPF_callbackHandle_t` - Refer section 5.2 of Software API Conventions IA Rev 2.0

  - `NPF_callbackType_t` - Refer section 5.2 of Software API Conventions IA Rev 2.0
  - `NPF_userContext_t` - Refer section 5.2 of Software API Conventions IA Rev 2.0
  - `NPF_eventMask_t` - Refer section 5.2 of Software API Conventions IA Rev 2.0
  - `NPF_errorReporting_t` - Refer section 5.2 of Software API Conventions IA Rev 2.0
- This document depends on Topology Manager Functional API Implementation Agreement Revision 1.0 for below type definitions
  - `NPF_BlockId_t` – Refer section 3.1.1 of Topology Manager Functional API IA Rev 1.0
  - `NPF_FE_Handle_t` – Refer section 3.1.1 of Topology Manager Functional API IA Rev 1.0
- ATM Software API Architecture Framework defines the architectural framework for the ATM FAPIs
- ATM Configuration Manager Functional API defines the functions to configure and manage ATM LFBs on a forwarding element

3 ATM Header Classifier Description

The ATM Header Classifier LFB receives ATM cells from the external ATM interface and does validation of the ATM header for errors. The ATM Header Classifier LFB uses the VPI from the ATM header along with the interface type identifying the interface as a UNI or NNI interface to determine the VP link on which the cell was received. If the VP is terminated at this node, the VCI of the ATM header is used to determine the VC link on which the cell was received. The ATM cell SDU is then extracted and sent to the next LFB in chain for processing.

The ATM Header Classifier LFB is modeled as shown in Figure 3.1:

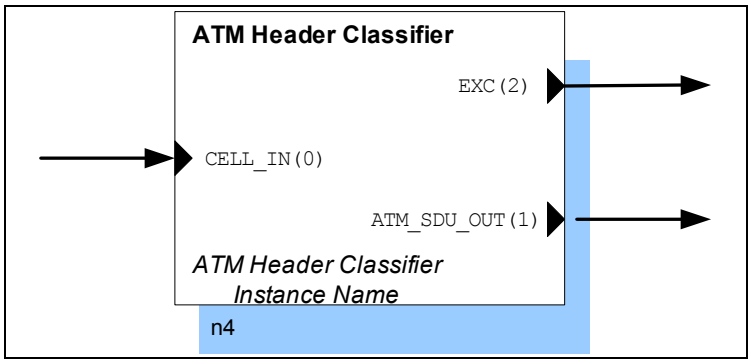

Figure 3.1: ATM Header Classifier LFB

The ATM Header Classifier LFB may contain multiple instances of interfaces that are identified by unique interface identifiers. Each interface instance has an attribute that identifies the interface as being a UNI or NNI interface. The incoming cells are associated with appropriate interface instance using the metadata received with the ATM cell. Such interface instances are depicted in Figure 3.2 below. The maximum number of such interfaces is an attribute of the ATM Header Classifier LFB and may be queried as such.

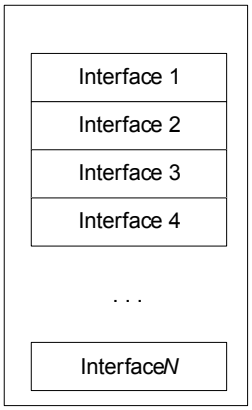

Figure 3.2: Interface Instances

The ATM Header Classifier LFB maintains the following statistics for each interface:

- Number of ATM cells received with CLP=0
- Number of ATM cells received with CLP=0+1
- Number of unexpected VPI/VCI cells (UNEX)
- Seconds with unexpected VPI/VCI cell (UNEX)
- Last recorded UNEX ATM cell header

The LFB may contain multiple instances of VP links associated with each interface that are identified by unique VP Link IDs. The incoming cells are classified using the VPI field of the ATM header together with the metadata identifying the interface on which the cells were received.

The LFB may contain multiple instances of VC links associated with each interface and is identified using the VPI/VCI from the ATM header along with the metadata identifying the interface on which the cell was received. ATM cells are associated with VC links only when the VP link carrying the cell is terminated at this node.

Such virtual link instances are depicted in Figure 3.3 below. The maximum number of VP links and VC links is an attribute of the ATM Header Classifier LFB and may be queried as such.

The ATM Header Classifier LFB maintains the following statistics for each virtual link:

- Number of ATM cells received with CLP=0
- Number of ATM cells received with CLP=0+1

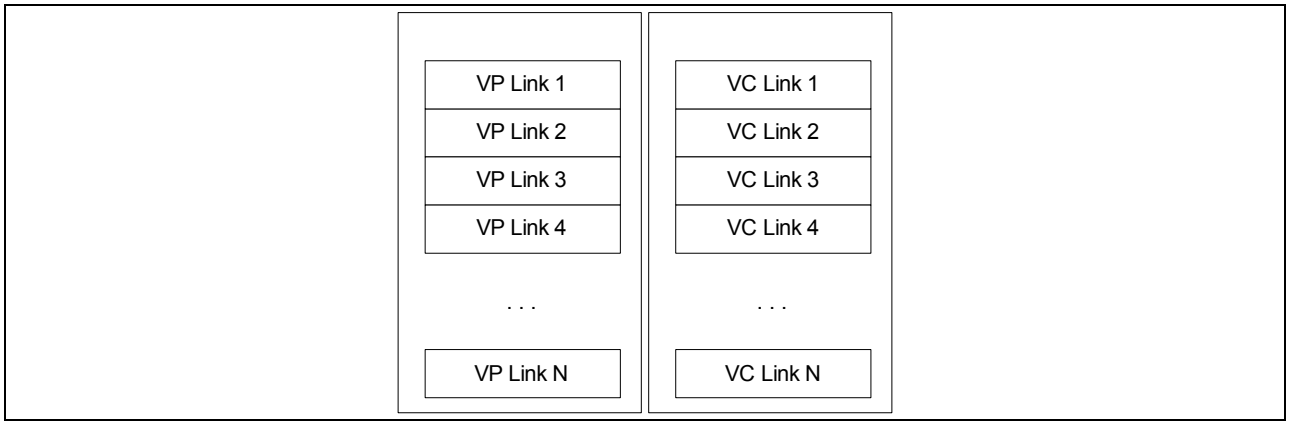

Figure 3.3: Virtual Link Instances

3.1 ATM Header Classifier Inputs

Table 3.1: ATM Header Classifier LFB Inputs

Symbolic Name	Input ID	Description
CELL_IN	0	This is the only input for the ATM Header Classifier LFB and is used to receive ATM cells from the LFB providing the ATM TC functions.

3.1.1 Metadata Required

Table 3.2: Input Metadata for ATM Header Classifier LFB

Metadata tag	Access method	Description
META_IF_ID	Read-And-Consume	Metadata to associate received ATM cell with the interface on which the cell was received.

3.2 ATM Header Classifier Outputs

Table 3.3: ATM Header Classifier LFB Outputs

Symbolic Name	Output ID	Description
ATM_SDU_OUT	1	This is the normal output for the ATM Header Classifier LFB through which the ATM SDU is passed to the next LFB in the chain.
EXC	2	The cell is sent to this output if 1. An unexpected VPI/VCI value received in the ATM header.

3.2.1.1 Metadata Produced

Table 3.4: Output Metadata for ATM Header Classifier LFB

Metadata tag	Access method	Description
META_VPL_ID	Write	Metadata identifying the VP link on which the ATM cell was received.
META_VCL_ID	Write	Metadata identifying the VC link on which the ATM cell was received. This metadata is produced only if the corresponding VP link is terminated.
META_AAL_TYPE	Write	ATM adaptation layer associated with this VC link. This metadata is produced only if the VC link is terminated. Could be one of AAL0, AAL1, AAL2, AAL5, AAL_UNKNOWN

META_ATM_PTI	Write	Payload Type of received ATM cell
META_ATM_LP	Write	Loss Priority of the received ATM cell
META_ATM_VCI	Write	The VCI of the received ATM cell.

3.3 Accepted Cell Types

The ATM Header Classifier LFB can accept ATM cells received over UNI or NNI.

3.4 Cell Modifications

The ATM Header Classifier LFB extracts the ATM SDU's from the received ATM cells and passes them to the next LFB. The ATM cells are not consumed by this LFB i.e. an ATM cell entering the ATM Header Classifier LFB always exits through one of the outputs. The ATM Header Classifier LFB processes ATM cells entering the LFB input sequentially. That means that ATM Header Classifier LFB does not change the order of cells.

3.5 Relationship with Other LFBs

The ATM Header Classifier LFB is placed in the processing chain after the ATM TC Receive LFB or IMA Receive LFB. The ATM Header Classifier LFB receives primarily cells from ATM TC Receive LFB that in turn receives cells from the transmission media.

Depending on the system design, one the following LFBs could be a recipient of the cells at the ATM Header Classifier ATM_SDU_OUT output.

- ATM OAM Receive LFB, which does OAM processing on the incoming ATM cells.
- ATM Policer LFB, which does UPC/NPC policing on the received ATM cells to enforce the traffic contract.
- Redirector LFB, which uses the AAL type to determine the next LFB in chain to process the ATM cell.

The EXC output of the ATM Header Classifier LFB could be connected to an LFB that receives cells with unexpected VPI/VCI values. Depending on system design this may be either dropper, which drops cells that are unexpected, or other LFB that makes a decision how to utilize such cells.

The recipient and producers of the cells and metadata that are described in this section may be replaced by LFBs that are able to generate information that is required by the ATM Header Classifier LFB at its input, and are able to utilize information present at the output of the ATM Header Classifier LFB. The exact design and connections between the ATM Header Classifier LFB and cooperating blocks is specific to the vendor that provides Forwarding Element design and FAPI implementation.

The sequence of actions that configures ATM Header Classifier LFB and cooperating ATM TC Receive LFB instance, and cooperation between these two LFBs is schematically depicted in Figure 3.4.

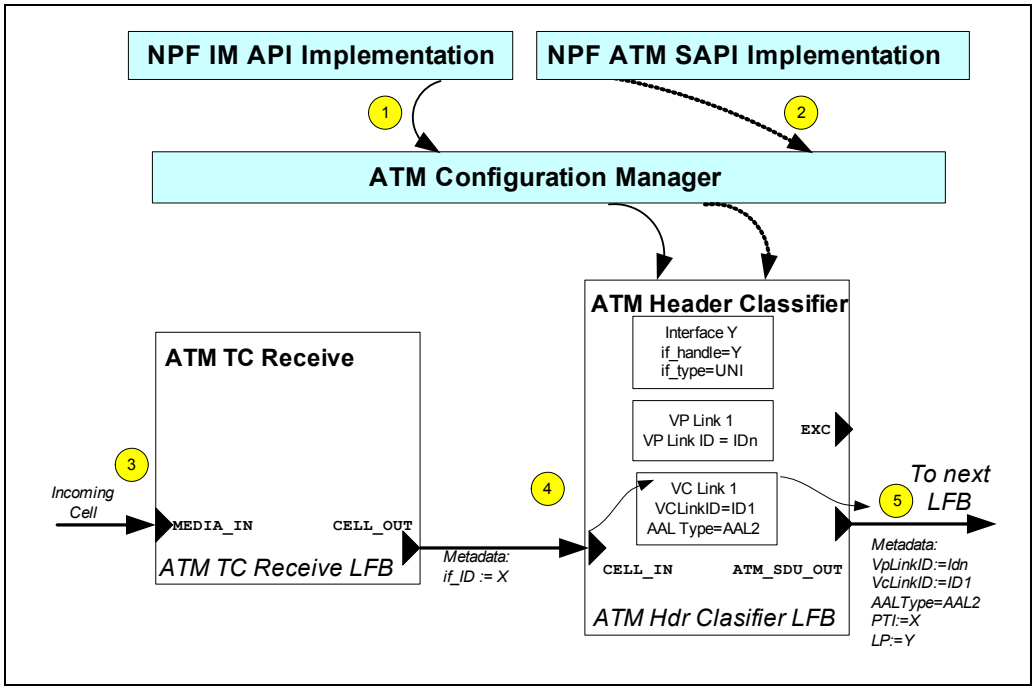

Figure 3.4: Cooperation between ATM TC Receive LFB and ATM Header Classifier LFB

This figure shows part of an example Forwarding Element that contains ATM configuration manager, ATM TC Receive and ATM Header Classifier LFBs. The ATM TC Receive LFB and the ATM Header Classifier LFB are connected in chain and configured by the ATM configuration manager LFB. The sequence of actions that configure interface and a VC link on the interface may be defined as follows (see corresponding numbers in circles in the figure):

1. The NPF IM API is invoked to create an UNI interface. The system software below the NPF IM API assigns an interface ID X to the interface and invokes the ATM configuration manager API to configure the interface. The ATM configuration manager FAPI call leads to creation of a UNI interface instance in the ATM Header Classifier LFB.
2. The NPF ATM SAPI API is invoked to create an ATM VC link instance. The system software below the NPF ATM SAPI assigns a VC link ID (‘ID1’) to the VC link and invokes the ATM configuration manager FAPI to create the VC link. The ATM configuration manager FAPI call leads to creation of a VP link instance in the ATM Header Classifier LFB.
3. Cell is received by the ATM TC Receive LFB on interface identified by interface ID ‘X’. ATM TC Receive LFB passes the cell to the `ATM_SDU_OUT` output along with the metadata specifying the interface on which this cell was received.
4. Cell is forwarded to the ATM Header Classifier input together with a metadata created by ATM TC Receive. The ATM Header Classifier LFB uses the interface ID to identify the interface instance associated with the interface ID specified in the input metadata. The interface type i.e. UNI or NNI is determined using the configuration of the interface instance. The ATM Header Classifier LFB reads the ATM header of the received ATM cell and uses the VPI, the interface ID and the configured interface type (UNI or NNI) of the interface on which the cell was received to determine the associated VP link on which the cell was received. The cell is determined to be received on a terminated VP link in this example. The VCI of the ATM header is further analyzed to determine the VC link on which this cell is received.

The ATM Header Classifier LFB passes the ATM cell SDU to the `ATM_SDU_OUT` output along with the metadata associated with the ATM SDU. The SDU is then processed by next LFB in chain.

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 Header Classifier LFB in a forwarding element using a block type value for the ATM Header Classifier LFB.

```
#define NPF_F_ATMHDRCLASSIFY_LFB_TYPE     30
```

4.1.2 ATM Header Classifier Configurations

4.1.2.1 ATM Header Classifier Virtual Channel Link Characteristics

The ATM Header classifier requires below configurations for each VC link.

- VPI – Virtual Path Identifier
- VCI – Virtual Circuit Identifier
- Interface ID
- VC link ID
- AAL associated with the VC link
- Adminstrative status – Up/Down/Testing

4.1.2.2 ATM Header Classifier Virtual Path Link Characteristics

The ATM Header classifier maintains below configurations for each VP link.

- VPI – Virtual Path Identifier
- Interface ID
- VP link ID
- Adminstrative status – Up/Down/Testing
- Virtual path link type – switched/terminated

4.1.2.3 ATM Header Classifier Interface Characteristics

The ATM Header classifier maintains below configurations for each configured interface:

- Interface type (UNI/NNI)

4.2 Data Structures for Completion Callbacks

4.2.1 ATM Header Classifier LFB Attributes query response

The attributes of an ATM Header Classifier LFB are the following:

```
typedef struct {
   NPF_uint32_t   maxVpl;               /* Maximum possible VP links     */
   NPF_uint32_t   curNumVpl;            /* Current number of VP links    */
   NPF_uint32_t   maxVcl;               /* Maximum possible VC links     */
   NPF_uint32_t   curNumVcl;            /* Current number of VC links    */
   NPF_uint32_t   maxInterfaces;        /* Maximum possible interfaces   */
   NPF_uint32_t   curNumIfs;            /* Current number of interfaces  */
}  NPF_F_ATMHdrClassifierLFB_AttrQueryResponse_t;
```

The `maxVpl`, `maxVcl` and `maxInterfaces` fields contains the maximum number of VP links, VC links and interfaces supported in this ATM Header Classifier LFB respectively. The `curNumVpl`, `curNumVcl` and `curNumIfs` field contains the number of VP links, VC link and interfaces configured in the ATM Header Classifier 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_ATMHdrClassifier_ErrorType_t error; /* Error code for response  */
    union {
        /* NPF_F_ATMHdrClassifier_LFB_AttributesQuery() */
        NPF_F_ATMHdrClassifierLFB_AttrQueryResponse_t   lfbAttrQueryResponse;
    } u;
} NPF_F_ATMHdrClassifier_AsyncResponse_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_ATMHdrClassifier_CallbackData_t.
*/
typedef enum NPF_F_ATMHdrClassifier_CallbackType {
    NPF_F_ATMHDRCLASSIFY_LFB_ATTR_QUERY = 1,
} NPF_F_ATMHdrClassifier_CallbackType_t;
```

4.2.4 Callback Data

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

```
/*
* The callback function receives the following structure containing
* of a asynchronous responses from a function call.
* For the completed request, the error code is specified in the
* NPF_ATM_AsyncResponse_t structure, along with any other information
*/
typedef struct {
    NPF_F_ATMHdrClassifier_CallbackType_t type; /* Which function called?  */
    NPF_IN NPF_BlockId_t          blockId;/* ID of LFB generating callback */
    NPF_F_ATMHdrClassifier_AsyncResponse_t  resp;  /* response structure   */
} NPF_F_ATMHdrClassifier_CallbackData_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_ATMHDRCLASSIFY_ATTR_QUERY	NPF_F_ATMHdrClassifierLFB_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 Header Classifier 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 ATM Header Classifier configuration and management APIs error codes. These codes are used in callbacks to deliver results of the requested operations. The base for the error codes used in ATM LFBs is derived as `LFB_TYPE_CODE * 100`.

```
/* Asynchronous error codes (returned in function callbacks) */
typedef NPF_uint32_t NPF_F_ATMHdrClassifier_ErrorType_t;

#define NPF_ATMHDRCLASSIFY_BASE_ERR (NPF_F_ATMHDRCLASSIFY_LFB_TYPE * 100)
#define NPF_E_ATMHDRCLASSIFY_INVALID_ATM_HDR_CLASSIFY_BLOCK_ID
                                              (NPF_ATMHDRCLASSIFY_BASE_ERR + 0)
```

5 Functional API (FAPI)

5.1 Required Functions

None

5.2 Conditional Functions

The ATM Header Classifier LFB does not have any mandatory functions. The conditional functions are required only if the LFB implements any of the optional functions for this LFB.

5.2.1 Completion Callback Function

```
typedef void (*NPF_F_ATMHdrClassifier_CallbackFunc_t) (
   NPF_IN NPF_userContext_t               userContext,
   NPF_IN NPF_correlator_t                correlator,
   NPF_IN NPF_F_ATMHdrClassifier_CallbackData_t data);
```

5.2.1.1 Description

This callback function is for the application to register an asynchronous response handling routine to the ATM Header Classifier API implementation. This callback function is intended to be implemented by the application, and be registered to the ATM Header Classifier API implementation through the `NPF_F_ATMHdrClassifier_Register` function. This function is a routine to handle ATM Header Classifier asynchronous responses. This is a required 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 Header Classifier 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_ATMHdrClassifier_Register(
   NPF_IN NPF_userContext_t               userContext,
   NPF_IN NPF_F_ATMHdrClassifier_CallbackFunc_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 Header Classifier 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 Header Classifier 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 Header Classifier 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_ATMHdrClassifier_Deregister(
   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 Header Classifier 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_ATMHdrClassifier_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);
```

5.3.1.1 Description

This function call is used to query ONLY one ATM Header Classifier LFB's attributes at a time. If the ATM Header Classifier 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 Header Classifier 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 Header Classifier 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

Possible error codes are:

- `NPF_NO_ERROR` – Operation completed successfully.
- `NPF_E_ATMHDRCLASSIFY_INVALID_ATM_HDR_CLASSIFY_BLOCK_ID` – LFB ID is not an ID of LFB that has ATM Header Classifier functionality.
- The `lfbAttrQueryResponse` field of the union in the `NPF_F_ATMHdrClassifier_AsyncResponse_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] "FAPI Topology Manager API", work in progress, Network Processing Forum SWAPI Functional API TG, 2004.

[SWAPICON] "Software API Conventions Revision 2", http://www.npforum.org/techinfo/APIConventions2_IA.pdf, Network Processing Forum SWAPI Foundations TG, September 2003

[ATMLFBARC] ATM Software API Architecture Framework Implementation Agreement

[ATMMGR] ATM Configuration Manager Functional API Implementation Agreement

APPENDIX A HEADER FILE INFORMATION

```
/*
 * This header file defines typedefs, constants and structures
 * for the NP Forum ATM Header Classifier Functional API
 */
#ifndef __NPF_F_ATM_HEADER_CLASSIFIER_H__
#define __NPF_F_ATM_HEADER_CLASSIFIER_H__

#ifdef __cplusplus
extern "C" {
#endif

/* ATM Header Classifier LFB Type ID */
#define NPF_F_ATMHDRCLASSIFY_LFB_TYPE     30

/* Asynchronous error codes (returned in function callbacks) */
typedef NPF_uint32_t NPF_F_ATMHdrClassifier_ErrorType_t;

#define NPF_ATMHDRCLASSIFY_BASE_ERR (NPF_F_ATMHDRCLASSIFY_LFB_TYPE * 100)
#define NPF_E_ATMHDRCLASSIFY_INVALID_ATM_HDR_CLASSIFY_BLOCK_ID\
                                    (NPF_ATMHDRCLASSIFY_BASE_ERR + 0)


/*************************************************************
 * Enumerations and types for ATM Header Classifier LFB      *
 *************************************************************/
/* Attributes of an ATM Header Classifier LFB */
typedef struct {
   NPF_uint32_t   maxVpl;               /* Maximum possible VP links     */
   NPF_uint32_t   curNumVpl;            /* Current number of VP links    */
   NPF_uint32_t   maxVcl;               /* Maximum possible VC links     */
   NPF_uint32_t   curNumVcl;            /* Current number of VC links    */
   NPF_uint32_t   maxInterfaces;        /* Maximum possible interfaces   */
   NPF_uint32_t   curNumIfs;            /* Current number of interfaces  */
}  NPF_F_ATMHdrClassifierLFB_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_ATMHdrClassifier_ErrorType_t error; /* Error code for response  */
    union {
        /* NPF_F_ATMHdrClassifier_LFB_AttributesQuery() */
        NPF_F_ATMHdrClassifierLFB_AttrQueryResponse_t   lfbAttrQueryResponse;
    } u;
} NPF_F_ATMHdrClassifier_AsyncResponse_t;

/*
* Completion Callback Types, to be found in the callback
* data structure, NPF_F_ATMHdrClassifier_CallbackData_t.
*/
typedef enum NPF_F_ATMHdrClassifier_CallbackType {
    NPF_F_ATMHDRCLASSIFY_LFB_ATTR_QUERY = 1,
} NPF_F_ATMHdrClassifier_CallbackType_t;
/*
* The callback function receives the following structure containing
* of a asynchronous responses from a function call.
```

```
* For the completed request, the error code is specified in the
* NPF_ATM_AsyncResponse_t structure, along with any other information
*/
typedef struct {
    NPF_F_ATMHdrClassifier_CallbackType_t type; /* Which function called?  */
    NPF_IN NPF_BlockId_t          blockId;/* ID of LFB generating callback */
    NPF_F_ATMHdrClassifier_AsyncResponse_t  resp;  /* response structure   */
} NPF_F_ATMHdrClassifier_CallbackData_t;

typedef void (*NPF_F_ATMHdrClassifier_CallbackFunc_t) (
      NPF_IN NPF_userContext_t userContext,
      NPF_IN NPF_correlator_t correlator,
      NPF_IN NPF_F_ATMHdrClassifier_CallbackData_t data);

/*********************************************************************
 * ATM Header Classifier LFB Registration/De-registration Functions *
 *********************************************************************/
/* Completion Callback Registration Function */
NPF_error_t NPF_F_ATMHdrClassifier_Register(
      NPF_IN NPF_userContext_t userContext,
      NPF_IN NPF_F_ATMHdrClassifier_CallbackFunc_t callbackFunc,
      NPF_OUT NPF_callbackHandle_t *callbackHandle);

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

/*********************************************************************
 * ATM Header Classifier LFB optional functions                      *
 *********************************************************************/

/* LFB Attributes Query Function */
NPF_error_t NPF_F_ATMHdrClassifier_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);

#ifdef __cplusplus
}
#endif
#endif /* __NPF_F_ATM_HEADER_CLASSIFIER_H__ */
```

APPENDIX B ACKNOWLEDGEMENTS

Working Group Chair: Alex Conta

Task Group Chair: Per Wollbrand

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

The list is in alphabetical order of last names:

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

APPENDIX C LIST OF COMPANIES BELONGING TO NPF DURING APPROVAL PROCESS

Agere Systems	Hifn	NTT Electronics
Altera	IBM	PMC Sierra
AMCC	IDT	Seaway Networks
Analog Devices	Infineon Technologies AG	Sensory Networks
Avici Systems	Intel	Sun Microsystems
Cypress Semiconductor	IP Fabrics	Teja Technologies
Enigma Semiconductor	IP Infusion	TranSwitch
Ericsson	Kawasaki LSI	U4EA Group
Erlang Technologies	Motorola	Wintegra
EZChip	NetLogic	Xelerated
Flextronics	Nokia	Xilinx
HCL Technologies	Nortel Networks	ZNYX Networks