ATM Header Generator LFB and Functional API Implementation Agreement

April 11, 2005
Revision 1.0

Editor:

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

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

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 Generator LFB and Functional API Implementation Agreement. Source: npf2004.152.14.

2 Introduction

This contribution defines the ATM header generator and lists configurations that are required in 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
- **SDU:** Service Data Unit
- **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 generator 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 – NPF work in progress
- ATM Configuration Manager Functional API defines the functions to configure and manage ATM LFBs on a forwarding element – NPF work in progress

3 ATM Header Generator Description

The ATM Header Generator LFB receives ATM SDU from the previous LFB in the pipeline, applies the ATM cell header on the ATM cell and hands it off to the ATM TC Transmit LFB. The ATM Header Generator LFB uses the VP/VC Link ID, payload type and loss priority signaled in the metadata to determine the ATM header to form the ATM cell. The ATM Header Generator LFB is modeled as shown in **Error! Reference source not found.**:

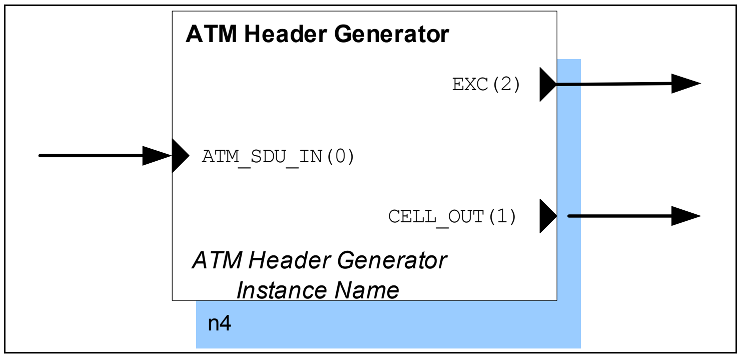

Figure 3.1: ATM Header Generator LFB

The ATM Header Generator LFB may contain multiple instances of interfaces that are identified by unique interface IDs. The incoming cells are associated with appropriate interface instance using the metadata received with the ATM cell. Such interface instances are depicted in Figure 3.1 below. The maximum number of such interfaces is an attribute of the ATM Header Generator LFB and may be queried as such.

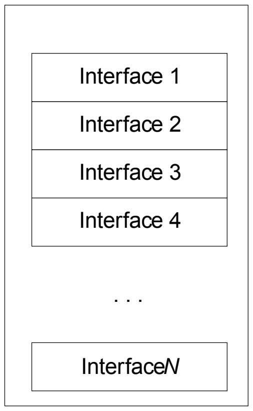

Figure 3.2: Interface Instances

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

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

The LFB may contain multiple instances of VP links and VC links that are identified by unique VP link ID and VC link ID respectively. The metadata received with the ATM SDU identifies the VP link or the VC link on which the ATM SDU is to be transmitted. Such VP and VC link instances are depicted in Figure 3.3 below. The maximum number of VP and VC links is an attribute of the ATM Header Generator LFB and may be queried as such.

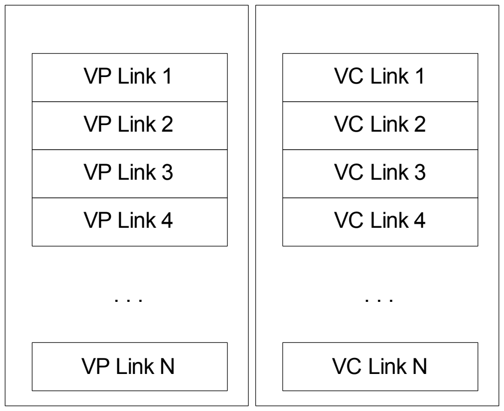

Figure 3.3: Virtual Link Instances

The ATM Header Generator LFB maintains the following counters for each VP and VC link:

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

3.1 ATM Header Generator Inputs

Table 3.1: ATM Header Generator LFB Inputs

Symbolic Name	Input ID	Description
`ATM_SDU_IN`	0	This is the only input for the ATM Header Generator LFB and is used to receive ATM SDU's to be sent on the VP/VC link signaled in the metadata.

3.1.1 Metadata Required

Table 3.2: Input Metadata for ATM Header Generator LFB

Metadata tag	Access method	Description
`META_VPL_ID`	Read-and-consumed	Metadata identifying the VP link on which the ATM cell is to be transmitted. This metadata is expected only for VP switched links.
`META_VCL_ID`	Read-and-consumed	Metadata identifying the VC link on which the ATM cell is to be transmitted. This metadata is expected for VC switched links or for originating VC links. When the ATM SDU is to be transmitted over a VP switched link, this metadata is not expected.
`META_ATM_PTI`	Read-and-consumed	Payload Type of ATM SDU
`META_ATM_LP`	Read-and-consumed	Loss Priority of the ATM SDU
`META_ATM_VCI`	Read-and-consumed	This metadata is expected only for VP switched links. The VP link ID identifies the VPI for the ATM cell. This metadata identifies the VCI for the cell.

3.2 ATM Header Generator Outputs

Table 3.3: ATM Header Generator LFB Outputs

Symbolic Name	Output ID	Description
`CELL_OUT`	1	This is the normal output for the ATM Header Generator LFB. ATM cell that could be identified with an established VP/VC link are sent on this output for further processing.
`EXC`	2	The cell is sent to this output if an error is encountered

3.2.1.1 Metadata Produced

Table 3.4: Output Metadata for ATM Header Generator LFB

Metadata tag	Access method	Description
`META_IF_ID`	Write	Metadata to associate interface on which the ATM cell has to be transmitted.

3.3 Accepted Cell Types

The ATM Header Generator LFB can accept ATM SDU's for transmission over UNI or NNI.

3.4 Cell Modifications

The ATM SDU's received by the ATM Header Generator LFB are placed in an ATM cell and encapsulated with the ATM header. If the ATM cell is to be transmitted on a VC link, the VPI and VCI configured corresponding to the VC link ID indicated in the metadata are used to build the ATM header

If the ATM cell is to be transmitted on a VP link, the VPI to build the ATM header is determined using the VP link ID indicated in the metadata. The VCI to build the ATM header is obtained from the metadata. The payload type and loss priority signaled in the metadata are used along with the determined VPI and VCI to form the ATM header.

The ATM SDU's received by the ATM Header Generator LFB are not consumed by the LFB and always exit through one of the outputs. The ATM Header Generator LFB processes ATM SDU's entering the LFB input sequentially. That means that ATM Header Generator LFB does not change the order of transmission of the ATM SDU's.

3.5 Relationship with Other LFBs

The ATM Header Generator LFB is placed in the processing chain before the ATM TC Transmit LFB or the IMA Transmit LFB. The ATM Header Generator LFB receives primarily ATM SDU's from previous LFB and passes ATM cells to the ATM TC Transmit LFB's for transmission on media.

The recipient and producers of the SDU's, 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 Generator LFB at its input, and are able to utilize information present at the output of the ATM Header Generator LFB. The exact design and connections between the ATM Header Generator LFB and cooperating blocks is specific to the vendor that provides Forwarding Element design and FAPI implementation.

The EXC output of the ATM Header Generator LFB could be connected an LFB that receives SDU's for which proper virtual link instance could not be found. Depending on system design this may be either dropper, which drops SDU's that could not be properly associated with a virtual link, or other LFB that makes a decision how to utilize such SDU's.

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

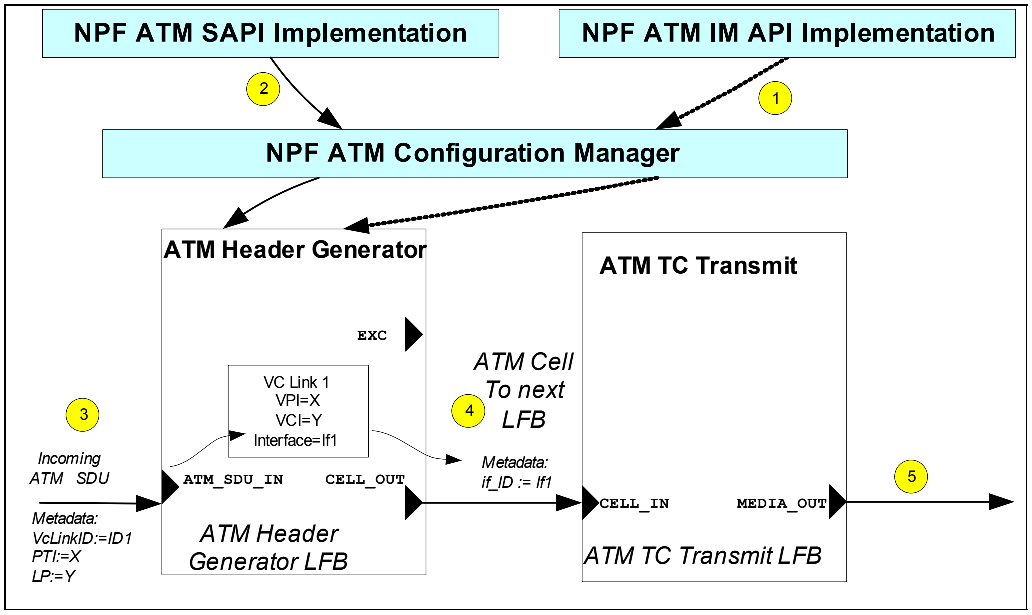

Figure 3.4: Cooperation between ATM TC Transmit LFB and ATM Header Generator LFB

This figure shows part of example Forwarding Element that contains ATM TC Transmit and ATM Header Generator LFBs. These two blocks are connected in chain and configured by a NPF SAPI implementation and Interface Management implementations. 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 Implementation creates an UNI interface. The system software below the NPF IM API assigns an interface ID ‘if1’ to the interface and invokes the ATM configuration manager API to create the interface. The ATM configuration manager FAPI call leads to creation of a UNI interface instance in the ATM Header Generator LFB.
2. The NPF ATM SAPI Implementation creates 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 link. The ATM configuration manager FAPI call leads to creation of a VC link instance in the ATM Header Generator LFB.
3. The ATM Header Generator LFB uses the VC Link ID signaled in metadata to find a matching VC link instance on receiving an ATM SDU. The VPI/VCI values corresponding to the VC link instance along with the PTI and LP signaled in the metadata are used to construct the ATM header to be used to create the ATM cell. The ATM cell is passed to the `CELL_OUT` output along with the interface ID of the interface on which the cell has to be transmitted.
4. Cell is forwarded to the ATM TC Transmit LFB input together with a metadata created by ATM Header Generator LFB.

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

```
#define NPF_F_ATMHDRGEN_LFB_TYPE    31
```

4.1.2 ATM Header Generator Configurations

4.1.2.1 ATM Virtual Channel Link Characteristics

The ATM Header Generator LFB requires below configurations for each virtual channel link.

- VPI – Virtual Path Identifier
- VCI – Virtual Circuit Identifier
- VC Link ID
- Interface ID
- Administrative Status – Up/Down/Testing

4.1.2.2 ATM Virtual Path Link Characteristics

The ATM Header Generator LFB requires below configurations for each virtual path link.

- VPI – Virtual Path Identifier
- VP Link ID
- Interface ID
- Administrative Status – Up/Down/Testing
- Virtual Path Link type – switched/terminated

4.2 Data Structures for Completion Callbacks

4.2.1 ATM Header Generator LFB Attributes query response

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

```
typedef struct {
   NPF_uint32_t   maxVpls;              /* Maximum possible VP links   */
   NPF_uint32_t   maxVcls;              /* Maximum possible VC links   */
   NPF_uint32_t   curNumVpls;           /* Current number of VP links  */
   NPF_uint32_t   curNumVcls;           /* Current number of VC links  */
   NPF_uint32_t   maxInterfaces;        /* Maximum possible interfaces */
   NPF_uint32_t   curNumIfs;            /* Current number of interfaces*/
}  NPF_F_ATMHdrGeneratorLFB_AttrQueryResponse_t;
```

The `maxVpls, maxVcls` and `maxInterfaces` fields contains the maximum number of VP links, VC links and interfaces supported in this ATM Header Generator LFB. The `curNumVpls, curNumVcls` and `curNumIfs` field contains the number of VP links, VC links and interfaces configured in the ATM Header Generator 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_ATMHdrGenerator_ErrorType_t error;/* Error code for response   */
    union {
        /* NPF_F_ATMHdrGenerator_LFB_AttributesQuery() */
        NPF_F_ATMHdrGeneratorLFB_AttrQueryResponse_t  lfbAttrQueryResponse;
    } u;
} NPF_F_ATMHdrGenerator_AsyncResp_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_ATMHdrGenerator_CallbackData_t.
*/
typedef enum NPF_F_ATMHdrGenerator_CallbackType {
    NPF_F_ATMHDRGEN_ATTR_QUERY = 1,
} NPF_F_ATMHdrGenerator_CallbackType_t;
```

4.2.3.1 Callback Data

An asynchronous response contains an error/success code and a function-specific structure embedded in a union in the `NPF_F_ATMHdrGenerator_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_ATMHdrGenerator_CallbackType_t type; /* Which function called?   */
    NPF_IN NPF_BlockId_t          blockId;/* ID of LFB generating callback */
    NPF_F_ATMHdrGenerator_AsyncResp_t  resp;/* response structure */
} NPF_F_ATMHdrGenerator_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_ATMHDRGEN_ATTR_QUERY	NPF_F_ATMHdrGeneratorLFB_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 Generator 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 Header Generator 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_ATMHdrGenerator_ErrorType_t;

#define NPF_ATMHDRGEN_BASE_ERR (NPF_F_ATMHDRGEN_LFB_TYPE * 100)
#define NPF_E_ATMHDRGEN_INVALID_ATMHDRGEN_BLOCK_ID
                                                  (NPF_ATMHDRGEN_BASE_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_ATMHdrGenerator_CallbackFunc_t) (
   NPF_IN NPF_userContext_t                userContext,
   NPF_IN NPF_correlator_t                 correlator,
   NPF_IN NPF_F_ATMHdrGenerator_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 Generator API implementation. This callback function is intended to be implemented by the application, and be registered to the ATM Header Generator API implementation through the `NPF_F_ATMHdrGenerator_Register` function. This function is a routine to handle ATM Header Generator asynchronous responses.

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 Generator 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_ATMHdrGenerator_Register(
   NPF_IN NPF_userContext_t                userContext,
   NPF_IN NPF_F_ATMHdrGenerator_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 Generator 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 Generator 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 Generator 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_ATMHdrGenerator_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 Generator 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_ATMHdrGenerator_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 Generator LFB's attributes at a time. If the ATM Header Generator 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 Generator 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 Generator 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_ATMHDRGEN_INVALID_ATMHDRGEN_BLOCK_ID` – LFB ID is not an ID of LFB that has ATM Header Generator functionality.

The `lfbAttrQueryResponse` field of the union in the `NPF_F_ATMHdrGenerator_AsyncResponse_t` structure is 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 Generator Functional API
 */
#ifndef __NPF_F_ATM_HEADER_GENERATOR_H__
#define __NPF_F_ATM_HEADER_GENERATOR_H__

#ifdef __cplusplus
extern "C" {
#endif

#define NPF_F_ATMHDRGEN_LFB_TYPE    31

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

#define NPF_ATMHDRGEN_BASE_ERR (NPF_F_ATMHDRGEN_LFB_TYPE * 100)
#define NPF_E_ATMHDRGEN_INVALID_ATMHDRGEN_BLOCK_ID\
                                (NPF_ATMHDRGEN_BASE_ERR + 0)

/****************************************************************
 * Enumerations and types for ATM Header Generator LFB         *
 ****************************************************************/
typedef struct {
    NPF_uint32_t   maxVpls;              /* Maximum possible VP links   */
    NPF_uint32_t   maxVcls;              /* Maximum possible VC links   */
    NPF_uint32_t   curNumVpls;           /* Current number of VP links  */
    NPF_uint32_t   curNumVcls;           /* Current number of VC links  */
    NPF_uint32_t   maxInterfaces;        /* Maximum possible interfaces */
    NPF_uint32_t   curNumIfs;            /* Current number of interfaces*/
}  NPF_F_ATMHdrGeneratorLFB_AttrQueryResponse_t;

/*
* Completion Callback Types, to be found in the callback
* data structure, NPF_F_ATMHdrGenerator_CallbackData_t.
*/
typedef enum NPF_F_ATMHdrGenerator_CallbackType {
    NPF_F_ATMHDRGEN_ATTR_QUERY = 1,
} NPF_F_ATMHdrGenerator_CallbackType_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_ATMHdrGenerator_ErrorType_t error;/* Error code for response   */
    union {
        /* NPF_F_ATMHdrGenerator_LFB_AttributesQuery() */
        NPF_F_ATMHdrGeneratorLFB_AttrQueryResponse_t  lfbAttrQueryResponse;
    } u;
} NPF_F_ATMHdrGenerator_AsyncResp_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_ATMHdrGenerator_CallbackType_t type; /* Which function called?   */
    NPF_IN NPF_BlockId_t          blockId;/* ID of LFB generating callback */
    NPF_F_ATMHdrGenerator_AsyncResp_t  resp;/* response structure */
} NPF_F_ATMHdrGenerator_CallbackData_t;

/********************************************************************
 * ATM Header Generator LFB Registration/De-registration Functions  *
 ********************************************************************/
typedef void (*NPF_F_ATMHdrGenerator_CallbackFunc_t) (
    NPF_IN NPF_userContext_t             userContext,
    NPF_IN NPF_correlator_t              correlator,
    NPF_IN NPF_F_ATMHdrGenerator_CallbackData_t    data);

NPF_error_t NPF_F_ATMHdrGenerator_Register(
    NPF_IN NPF_userContext_t             userContext,
    NPF_IN NPF_F_ATMHdrGenerator_CallbackFunc_t callbackFunc,
    NPF_OUT NPF_callbackHandle_t        *callbackHandle);

NPF_error_t NPF_F_ATMHdrGenerator_Deregister(
    NPF_IN NPF_callbackHandle_t    callbackHandle);

/********************************************************************
 * ATM Header Generator LFB optional functions                     *
 ********************************************************************/
NPF_error_t NPF_F_ATMHdrGenerator_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_GENERATOR_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