ATM OAM Receive LFB and Functional API Implementation Agreement

August 16, 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	08/16/2005	Rev 1.0 of the ATM OAM Receive LFB and Functional API Implementation Agreement. Source: npf2004.161.06.

2 Introduction

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

2.1 Acronyms

- **AIS**: Alarm Indication Signal
- **ATM**: Asynchronous Transfer Mode
- **API**: Application Programming Interface
- **BR**: Backward Reporting
- **CC**: Continuity Check
- **F4:** OAM flow on virtual path level
- **F5:** OAM flow on virtual channel level
- **FAPI:** Functional API
- **FPM:** Forward Performance Monitoring
- **ID:** Identifier
- **LB:** Loopback
- **LFB:** Logical Functional Block
- **LP:** Loss Priority
- **NNI:** Network Node Interface
- **OAM:** Operation and Maintenance
- **PM:** Performance Monitoring
- **PTI:** Payload Type Indicator
- **PVC:** Permanent Virtual Connection
- **RDI:** Remote Defect Indication
- **VC:** Virtual Connection
- **VCC:** Virtual Circuit Connections
- **VCI:** Virtual Channel Identifier
- **VPC:** Virtual Path Connections
- **VPI:** Virtual Path Identifier

2.2 Assumptions

The ATM OAM Receive 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 ATM OAM processing. 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 OAM Receive Description

The ATM OAM Receive LFB does ATM OAM processing on the ATM SDU received from the previous LFB over the `ATM_SDU_IN` input. Depending on the connection on which the cells are received the cells may be classified into user cells and OAM cells for the OAM flow associated with that connection. The cells received on a VP link may be either user cells or OAM cells for F4 flow. The cells received on a VC link may be either user cells or OAM cells for F5 flow. Additionally, F5 OAM cells received on VC links are considered as user cells for the F4 flow on the associated VP.

OAM flows are related to bi-directional Maintenance Entities (MEs) corresponding either to the entire ATM VPC/VCC, referred to as the VPC/VCC ME, or to a portion of this connection referred to as a VPC/VCC segment ME.

Before the start of any OAM operation, the boundary needs to be drawn for the paired endpoints. The MEs terminating the ATM links are configured before as an endpoint of the VPC/VCC or endpoint of the VPC/VCC segment. End-to-end F5 flows terminate at the endpoints of a VCC, while the segment F5 flows terminate at the VCC segment endpoints. Similarly, the end-to-end F4 flows terminate at the endpoints of a VPC, while the segment F4 flows terminate at the VPC segment endpoints. The ATM OAM receive LFB performs the OAM functions configured for the OAM flow terminations. The ATM OAM receive LFB may also be configured to perform passive monitoring of OAM flows.

The ATM OAM Receive LFB is modeled as shown in Figure 3.1

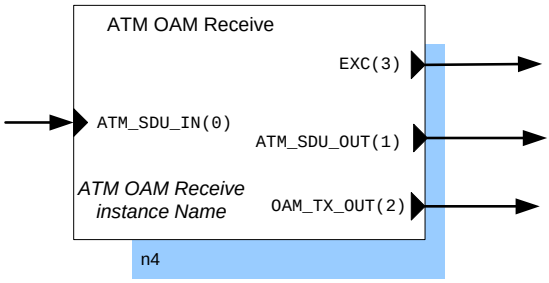

Figure 3.1: ATM OAM Receive LFB

The LFB may contain multiple instances of F4 flows identified by unique VP Link IDs. The LFB may contain multiple instances of F5 flows identified by unique VC Link IDs. ATM cells are associated with VC links only when the VP link carrying the cell is terminated at this node.

Such instances are depicted in Figure 3.2.

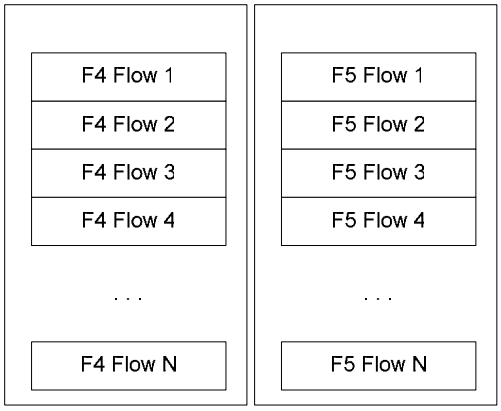

Figure 3.2: F4/F5 Flow Instances

The ATM OAM Receive LFB may generate OAM cells in response to the OAM cells received on a given link. The F4 and F5 flows are bidirectional and the OAM cells for both directions of the flow must follow the same physical route so that it is possible for any CP on that connection to correlate the fault and performance information from both directions. The `OAM_TX_OUT` output is used to send OAM cells created in response to the received OAM cells for F4/F5 flows, fault conditions or due to performance monitoring functions, etc. on the F4/F5 flows.

When non intrusive monitoring of fault management cells (AIS/RDI/CC/LB) is enabled at intermediate points the ATM OAM Receive LFB may detect faults and declare AIS state when AIS cells are received, transmission path AIS-defects are detected or defects like loss of continuity is detected on the monitored flow (segment/end-to-end) on the monitored link (VP/VC link). The AIS (segment_VP-AIS, e-t-e_VP-AIS, segment_VC-AIS, e-t-e_VC-AIS) state shall be released when a user cell or a CC cell is received on the monitored flow. The AIS state shall also be released if no AIS cells are seen on the monitored flow for 2.5 +/- 0.5 seconds. When non-intrusive monitoring is stopped for a specified flow, the AIS states declared for that flow shall be released. Declaration of the AIS condition may cause the ATM OAM Receive LFB to generate AIS cells on the `ATM_SDU_OUT` output.

3.1 ATM OAM Receive Inputs

Table 3.1: ATM OAM Receive LFB Inputs

Symbolic Name	Input ID	Description
`ATM_SDU_IN`	0	This is the only input for the ATM OAM Receive LFB and is used to receive ATM SDUs from the previous LFB.

3.1.1 Metadata Required

Table 3.2: Input Metadata for ATM OAM Receive 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. The PTI is used to identify F5 OAM cells.
`META_ATM_LP`	Read	Loss Priority of the received ATM cells.
`META_ATM_VCI`	Read	The VCI of the received ATM cell. The VCI is used to identify F4 OAM cells.

3.2 ATM OAM Receive Outputs

Table 3.3: ATM OAM Receive LFB Outputs

Symbolic Name	Output ID	Description
`ATM_SDU_OUT`	1	This is the normal output for the ATM OAM

		Receive LFB. The user cells for OAM flows are passed unmodified to this output. The ATM OAM Receive LFB may also generate OAM cells on this output when non intrusive monitoring of fault management cells is enabled at intermediate connection points.
OAM_TX_OUT	2	This output is used to send OAM cells created in response to the received OAM cells for F4/F5 flows, fault conditions or due to performance monitoring functions, etc. on the F4/F5 flows. The F4 and F5 flows are bidirectional and the OAM cells for both directions of the flow must follow the same physical route so that it is possible for any CP on that connection to correlate the fault and performance information from both directions.
EXC	3	The packet requested for transmission is sent to this output when the ATM SDU needs to be discarded due to errors.

3.2.1 Metadata Produced on ATM_SDU_OUT output

The ATM OAM Receive LFB does not modify or generate any metadata for ATM SDU of user cells sent to this output.

The ATM OAM Receive LFB may generate OAM cells on this output when non intrusive monitoring of fault management cells is enabled at intermediate connection points. For such OAM cells the ATM OAM Receive LFB generates the below metadata:

Table 3-4 Output Metadata for OAM cells generated on ATM_SDU_OUT

Metadata tag	Access method	Description
META_VPL_ID	Write	Metadata identifying the VP link on which the ATM OAM cell is to be transmitted. This metadata is generated when the ATM OAM cell is a F4 OAM cell.
META_VCL_ID	Write	Metadata identifying the VC link on which the ATM OAM cell is to be transmitted. This metadata is generated when the ATM OAM cell is a F5 OAM cell.
META_ATM_VCI	Write	Metadata identifying the VCI to be used for the ATM OAM cell to be transmitted. Only generated when sending F4 OAM cells.
META_ATM_PTI	Write	Payload Type of ATM cell.
META_ATM_LP	Write	Loss priority of the ATM cell.

3.2.2 Metadata Produced on OAM_TX_OUT output

This output is used to send OAM cells created in response to the received OAM cells for F4/F5 flows, fault conditions or due to performance monitoring functions, etc. on the F4/F5 flows. The F4 and F5 flows are bidirectional and the OAM cells for both directions of the flow must follow the same

physical route so that it is possible for any CP on that connection to correlate the fault and performance information from both directions.

Table 3-5 Output Metadata for OAM cells generated on OAM_TX_OUT

Metadata tag	Access method	Description
`META_VPL_ID`	Write	Metadata identifying the VP link on which the ATM OAM cell is to be transmitted. This metadata is generated when the ATM OAM cell is a F4 OAM cell.
`META_VCL_ID`	Write	Metadata identifying the VC link on which the ATM OAM cell is to be transmitted. This metadata is generated when the ATM OAM cell is a F5 OAM cell.
`META_ATM_VCI`	Write	Metadata identifying the VCI to be used for the ATM OAM cell to be transmitted. Only generated when sending F4 OAM cells.
`META_ATM_PTI`	Write	Payload Type of ATM cell.
`META_ATM_LP`	Write	Loss priority of the ATM cell.

3.3 Accepted Inputs

The ATM OAM Receive LFB accepts ATM SDU received over UNI or NNI interface.

3.4 Cell Modifications

The user cells received on the connection are not modified by the ATM OAM receive LFB. The ATM OAM Receive LFB will not change the order of the cells input to the LFB. The below modifications are affected to the incoming cell stream by the ATM OAM Receive LFB.

- The endpoints of a VPC will extract all end-to-end F4 OAM cells.
- The sink point of a VPC segment will extract all segment F4 OAM cells. The segment sink point may coincide with a VPC endpoint or may be intermediate connection points designated as the sink for the segment F4 OAM flow.
- The endpoints of a VCC will extract all end-to-end F5 OAM cells.
- The sink point of a VCC segment will extract all segment F5 OAM cells. The segment sink point may coincide with a VCC endpoint or may be intermediate connection points designated as the sink for the segment F5 OAM flow.

3.5 Relationship with Other LFBs

The ATM OAM receive LFB may be placed in the processing chain after the ATM Header Classifier LFB or the ATM Policer LFB. The ATM OAM Receive LFB receives primarily ATM SDUs from the previous LFB and performs ATM OAM processing on the received cell stream. The sequence of actions that configures ATM OAM Receive LFB and cooperating ATM Header Classifier LFB instance, and cooperation between these two LFBs is schematically depicted in Figure 3.3.

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

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

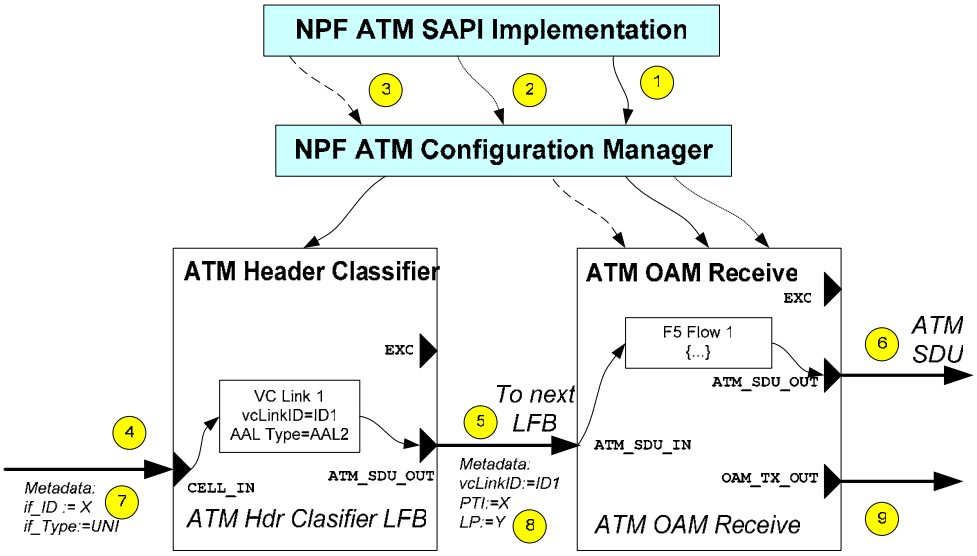

Figure 3.3: Cooperation between ATM OAM Receive and ATM Header Classifier

This figure shows part of example Forwarding Element that contains ATM Header Classifier LFB and ATM OAM Receive LFB. These two blocks are connected in chain and configured by a NPF SAPI implementation.

The sequence of actions that configure a VC links and enables performance monitoring functions on the VC link leading to generation of a BR cell may be defined as follows (see corresponding numbers in circles in the figure):

1. The NPF ATM SAPI is invoked to create an ATM VC link. The system software under the NPF ATM SAPI assigns a VC link ID ‘ID1’ to it and invokes the ATM Configuration manager FAPI to create the ATM VC link. This causes an ATM VC link instance to be created in the ATM header classifier LFB and a F5 Flow instance in the ATM OAM Receive LFB.
2. The NPF ATM SAPI is invoked to configure the ATM VC link as a segment endpoint for F5 flow. This system software under the NPF ATM SAPI invokes the ATM configuration manager FAPI to configure the ATM VC link as a segment end point for F5 flow.
3. The NPF ATM SAPI is invoked to configure performance monitoring procedures on the VC link with link ID ‘ID1’. In this example, the FPM-BR procedure is activated in the A-B direction. The system software under the NPF ATM SAPI invokes the ATM configuration manager FAPI to configure performance monitoring procedure on the ATM VC link.
4. An ATM cell is received by the ATM header classifier LFB on the interface identified by interface ID ‘X’. The ATM header classifier LFB reads the ATM header of the received ATM cell and uses the VPI, VCI and the interface ID and the interface type (UNI or NNI) of the interface on which the cell was received to determine the associated VC link instance.
5. The ATM SDU is passed along with the metadata to the ATM OAM Receive LFB. The ATM OAM receive LFB uses the VC link ID received in the input metadata to identify the associated F5 flow instance. The ATM OAM receives LFB uses the PTI received in the input metadata to identify the cell as a user cell or an OAM cell. The current received cell is identified as a user cell for F5 flow and as performance monitoring is enabled for the connection on which the cell was received, the performance monitoring statistics are updated.
6. The receive ATM SDU is passed to the next LFB in the processing chain over the ATM_SDU_OUT output. The input metadata is passed without any modifications.

7. An ATM cell is received by the ATM header classifier LFB on the interface identified by interface ID ‘X’. The ATM header classifier LFB reads the ATM header of the received ATM cell and uses the VPI, VCI and the interface ID and the interface type (UNI or NNI) of the interface on which the cell was received to determine the associated VC link instance.
8. The ATM SDU is passed along with the metadata to the ATM OAM Receive LFB. The ATM OAM receive LFB uses the VC link ID received in the input metadata to identify the F5 flow instance. The ATM OAM receives LFB uses the PTI receive in the input metadata to classify the cell as a user cell or an OAM cell. The current received cell is classified as an OAM cell for F5 flow. The ATM SDU is further examined to determine the type of OAM cell received. In this example, the received OAM cell is determined to be an FPM cell. The ATM OAM Receive LFB uses the accumulated statistics on the ATM connection and the data in the received FPM to generate a BR cell.
9. The FPM is consumed by the OAM receive LFB and the BR cell generated in response is sent to the `OAM_TX_OUT` output.

.

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 OAM Receive LFB in a forwarding element using a block type value for the ATM OAM Receive LFB.

```
#define NPF_F_ATMOAMRX_LFB_TYPE      41
```

4.1.2 ATM OAM Receive ATM link characteristics

The ATM OAM Receive LFB requires below configurations for each configured F4/F5 flow:

VP/VC link ID

- Connection point type – ETE endpoint, segment end point, ETE and segment endpoint, intermediate point for ETE flow, intermediate point for ETE and segment flows
- Whether LLID option is enabled
- The connection point ID
- If performance monitoring functions are enabled, then the following configurations are required
    - Performance monitoring function – FPM-BR or FPM
    - Direction – Forward, backward, two way
    - Forward direction block size (A-B direction)
    - Backward direction block size (B-A direction)
- If continuity check functions are enabled, then the following configurations are required
    - Direction – Forward, backward, two way
    - Continuity check method – whether continuity check sent periodically or sent only in the absence of user cells
- If loopback operation is to be carried out the following configurations are required
    - Loopback location ID
    - Whether source connection point ID to be included in the loopback cell
- AIS alarm states for the specified link. The following configurations are needed
    - Defect type
    - Defect location
- If non intrusive monitoring of the OAM flows is to be performed, then the following configurations are needed
    - Flow level to monitor
    - OAM cell types to monitor

4.2 Data Structures for Completion Callbacks

4.2.1 ATM OAM Receive LFB Attributes query response

The attributes of an ATM OAM Receive LFB are the following:

```
typedef struct {
   NPF_uint32_t   maxF4Flows;           /* Maximum possible F4 flows     */
   NPF_uint32_t   curNumF4Flows;        /* Current number of F4 flows    */
   NPF_uint32_t   maxF5Flows;           /* Maximum possible F5 flows     */
   NPF_uint32_t   curNumF5Flows;        /* Current number of F5 flows    */
   NPF_uint32_t   maxF4PMProcess;       /* Maximum F4 PM processes       */
```

```
   NPF_uint32_t   curNumF4PMProcess;    /* Current number of F4 PM procs */
   NPF_uint32_t   maxF5PMProcess;       /* Maximum F5 PM processes       */
   NPF_uint32_t   curNumF5PMProcess;    /* Current number of F5 PM procs */
} NPF_F_ATMOamRxLFB_AttrQueryResponse_t;
```

The `maxF4Flows` field contains the maximum number of ATM VP links supported in this ATM OAM Receive LFB. The `curNumF4Flows` field contains the number of ATM VP links established in the LFB. The `maxF5Flows` field contains the maximum number of ATM VC links supported in this ATM OAM Receive LFB. The `curNumF5Flows` field contains the number of ATM VC links established in the LFB. The `maxF4PMProcesses` and `maxF5PMProcesses` fields indicate the maximum number of F4 and F5 flows on which performance monitoring processes may be activated in this LFB. The `curNumF4PMProcesses` and `curNumF5PMProcesses` fields indicate the current number of F4 and F5 flows on which performance monitoring processes are activated.

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_ATMOamRxErrorType_t error;  /* Error code for this response*/
    union {
        /* NPF_F_ATMOamRxLFB_AttributesQuery() */
        NPF_F_ATMOamRxLFB_AttrQueryResponse_t    lfbAttrQueryResponse;
    } u;
} NPF_F_ATMOamRxAsyncResponse_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_ATMOamRxCallbackData_t.
*/
typedef enum NPF_F_ATMOamRxCallbackType {
    NPF_F_ATMOAMRX_ATTR_QUERY = 1,
} NPF_F_ATMOamRxCallbackType_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_ATMOamRxCallbackData_t` structure.

```
/*
* The callback function receives the following structure containing
* an asynchronous responses from a function call.
* For the completed request, the error code is specified in the
* NPF_ATMOamRxAsyncResponse_t structure, along with any other information
*/
typedef struct {
    NPF_F_ATMOamRxCallbackType_t type;    /* Which function called?       */
    NPF_IN NPF_BlockId_t          blockId; /*ID of LFB generating callback */
    NPF_F_ATMOamRxAsyncResponse_t resp;   /* Response struct              */
} NPF_F_ATMOamRxCallbackData_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_ATMOAMRX_ATTR_QUERY	NPF_F_ATMOamRxLFB_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 OAM Receive 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 OAM Receive 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_ATMOamRxErrorType_t;
#define NPF_ATMOAMRX_BASE_ERR (NPF_F_ATMOAMRX_LFB_TYPE * 100)
#define ATMOAMRX_ERR(n) ((NPF_F_ATMOamRxErrorType_t) \
                                        (NPF_ATMOAMRX_BASE_ERR+ (n))
/* LFB ID is not an ID of LFB that has ATM OAM Receive functionality*/
#define NPF_E_ATMOAMRX_INVALID_ATMOAMRX_BLOCK_ID ATMOAMRX_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_ATMOamRxCallbackFunc_t) (
      NPF_IN NPF_userContext_t            userContext,
      NPF_IN NPF_correlator_t             correlator,
      NPF_IN NPF_F_ATMOamRxCallbackData_t data);
```

5.2.1.1 Description

This callback function is for the application to register an asynchronous response handling routine to the ATM OAM Receive API implementation. This callback function is intended to be implemented by the application, and be registered to the ATM OAM Receive API implementation through the `NPF_F_ATMOamRxRegister` 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 OAM Receive 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_ATMOamRxRegister(
      NPF_IN NPF_userContext_t                  userContext,
      NPF_IN NPF_F_ATMOamRxCallbackFunc_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 OAM Receive 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 OAM Receive 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 OAM Receive 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_ATMOamRxDeregister(
      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 OAM Receive 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_ATMOamRxLFB_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 OAM Receive LFB's attributes at a time. If the ATM OAM Receive 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 OAM Receive 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 OAM Receive 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_ATMOAMRX_INVALID_ATMOAMRX_BLOCK_ID` – LFB ID is not an ID of LFB that has ATM OAM Receive functionality

The `lfbAttrQueryResponse` field of the union in the `NPF_F_ATMOamRxAsyncResponse_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, Network Processing Forum.

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

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

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

Appendix A Header File Information

```
/*
 * This header file defines typedefs, constants and structures
 * for the NP Forum ATM OAM Receive Functional API
 */

#ifndef __NPF_F_ATM_OAMRX_H__
#define __NPF_F_ATM_OAMRX_H__

#ifdef __cplusplus
extern "C" {
#endif

/* It is possible to use the FAPI Topology Discovery
   APIs to discover an ATM OAM Receive LFB
   in a forwarding element.  */
#define NPF_F_ATMOAMRX_LFB_TYPE     41

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

#define NPF_ATMOAMRX_BASE_ERR (NPF_F_ATMOAMRX_LFB_TYPE * 100)
#define ATMOAMRX_ERR(n) ((NPF_F_ATMOamRxErrorType_t) \
                         (NPF_ATMOAMRX_BASE_ERR+ (n))
/* LFB ID is not an ID of LFB that has ATM OAM Receive functionality*/
#define NPF_E_ATMOAMRX_INVALID_ATMOAMRX_BLOCK_ID ATMOAMRX_ERR (0)

/*************************************************************
 * Enumerations and types for ATM OAM Rx attributes and      *
 * completion callback data types                            *
 *************************************************************/

/* The attributes of an ATM OAM Receive LFB */
typedef struct {
   NPF_uint32_t   maxF4Flows;          /* Maximum possible F4 flows     */
   NPF_uint32_t   curNumF4Flows;       /* Current number of F4 flows    */
   NPF_uint32_t   maxF5Flows;          /* Maximum possible F5 flows     */
   NPF_uint32_t   curNumF5Flows;       /* Current number of F5 flows    */
   NPF_uint32_t   maxF4PMProcess;      /* Maximum F4 PM processes       */
   NPF_uint32_t   curNumF4PMProcess;   /* Current number of F4 PM procs */
   NPF_uint32_t   maxF5PMProcess;      /* Maximum F5 PM processes       */
   NPF_uint32_t   curNumF5PMProcess;   /* Current number of F5 PM procs */
} NPF_F_ATMOamRxLFB_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_ATMOamRxErrorType_t error;  /* Error code for this response*/
    union {
        /* NPF_F_ATMOamRxLFB_AttributesQuery() */
        NPF_F_ATMOamRxLFB_AttrQueryResponse_t    lfbAttrQueryResponse;
    } u;
} NPF_F_ATMOamRxAsyncResponse_t;

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

```
* data structure, NPF_F_ATMOamRxCallbackData_t.
*/
typedef enum NPF_F_ATMOamRxCallbackType {
    NPF_F_ATMOAMRX_ATTR_QUERY = 1,
} NPF_F_ATMOamRxCallbackType_t;


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


/* Type for a callback function to be registered with ATM OAM Rx */
typedef void (*NPF_F_ATMOamRxCallbackFunc_t) (
      NPF_IN NPF_userContext_t userContext,
      NPF_IN NPF_correlator_t correlator,
        NPF_IN NPF_F_ATMOamRxCallbackData_t data);


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

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

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