AAL5 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 AAL5 Receive LFB and Functional API Implementation Agreement. Source: npf2004.153.09.

2 Introduction

This contribution defines the AAL5 Receive LFB and lists configurations that are required in the LFB.

2.1 Acronyms

- **AAL:** ATM Adaptation Layer
- **AAL5:** ATM Adaptation Layer – Type 5
- **API:** Application Programming Interface
- **ATM**: Asynchronous Transfer Mode
- **CLP:** Cell Loss Priority
- **CPCS:** Common Part Convergence Sublayer
- **FAPI:** Functional API
- **IA:** Implementation Agreement
- **ID:** Identifier
- **LFB –** Logical Functional Block
- **NNI:** Network Node Interface
- **PTI:** Payload Type Indicator
- **PVC:** Permanent Virtual Connection
- **SDU:** Service Data Unit
- **SSCS:** Service Specific Convergence Sublayer
- **UNI:** User Network Interface
- **UUI:** User to User Information
- **VC:** Virtual Connection
- **VCC:** Virtual Channel Connection
- **VCI:** Virtual Channel Identifier

2.2 Assumptions

The AAL5 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 handling VC links carrying AAL5 traffic. 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 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 AAL5 Receive Description

The AAL5 Receive LFB receives ATM SDU's from the previous LFB and performs re-assembly of the ATM SDU's to create AAL5 packets. The AAL5 Receive LFB maintains the following counters for each VC link:

- Number of AAL5 frames received
- Number of AAL5 frames received with CRC error
- Number of AAL5 frames received with length error
- Number of bytes received on this VC link

The AAL5 Receive LFB is modeled as shown in Figure 3.1

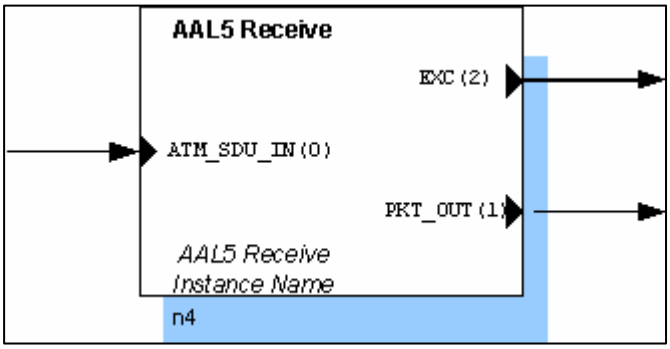

Figure 3.1: AAL5 Receive LFB

The LFB may contain multiple instances of VC links that are identified by unique VC link IDs. Incoming packets are assigned to appropriate VC link instance according to metadata received with ATM cell. Such instances are depicted in Figure 3.2 below. The maximum number of VC links that can be configured is an attribute of the AAL5 Receive LFB and may be queried as such.

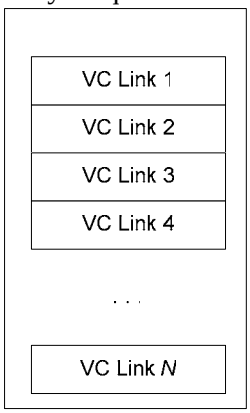

Figure 3.2: VC Link Instances

3.1 AAL5 Receive Inputs

Table 3.1: AAL5 Receive LFB Inputs

Symbolic Name	Input ID	Description
ATM_SDU_IN	0	This is the only input for the AAL5 Receive LFB and is used to receive ATM SDU's for re-assembly.

3.1.1 Metadata Required

Table 3.2: Input Metadata for AAL5 Receive LFB

Metadata tag	Access method	Description
META_VCL_ID	Read	Metadata identifying the VC link on which the ATM cell was received.
META_ATM_PTI	Read-and-consume	Payload Type of received ATM SDU.
META_ATM_LP	Read-and-consume	Loss Priority of the received ATM SDU.

3.2 AAL5 Receive Outputs

Table 3.3: AAL5 Receive LFB Outputs

Symbolic Name	Output ID	Description
CPCS_IF_DATA_OUT	1	This is the normal output for the AAL5 Receive LFB. Error free reassembled AAL5 packets are sent over this output to the next LFB in the chain.
EXC	2	The SDU or a partially re-assembled AAL5 packet is sent to this output if the reassembly of the AAL5 packet failed due to reasons like CRC error, length error, etc.

3.2.1.1 Metadata Produced

Table 3.4: Output Metadata for AAL5 Receive LFB

Metadata tag	Access method	Description
META_CPCS_LP	Write	CPCS Loss Priority of the reassembled AAL5 frame computed as specified in section 10.2.3 of ITU specification I.363.5.
META_UUI	Write	The UUI received in the re-assembled AAL5 frame. The UUI is transparently transported by the CPCS between peer CPCS users.
META_FRAME_LEN	Write	The length of the re-assembled AAL5 packet. Does not include the AAL5 trailer that is discarded by the AAL5 Receive LFB after re-assembly is completed.
META_BOUND_IF	Write	The handle of the child interface bound to this VC link.
META_RCV_STATUS	Write	This metadata element is generated only if the corrupted SDU delivery option is enabled. This field is a bit map indicating the type of error

		encountered in the reassembly process. The definition of the bit map is as specified in Annex E of I.363.5.

3.3 Accepted Cell Types

The AAL5 Receive LFB can accept ATM SDU's received over UNI or NNI.

3.4 Cell Processing

The AAL5 Receive LFB reassembles the received ATM SDUs to create the AAL5 frame. The CRC computed over the reassembled payload is compared with that received in the trailer along with the length of the reassembled payload. If there is a mismatch in either of these parameters, the reassembled frame is discarded by sending over the EXC output.

If the corrupted SDU delivery option is enabled, the partially reassembled SDU would be passed to the next LFB in the chain along with the metadata indicating the reception status (`META_RCV_STATUS`) to describe the error detected in the reassembly process.

If the reassembly was successful, the AAL5 packet in the reassembled payload is passed to the next LFB in the chain for further processing.

The higher layer interface handle that is bound to this VC link is also provided along with the UUI received with this frame to the next LFB. The higher layer interface handle may be used to identify the next LFB to process the packet. For example if the higher layer interface handle is that used by the packet handler client, the packet will be delivered to the packet handler for sending to the control plane. This would be true for VC links carrying signaling traffic.

3.5 Relationship with Other LFBs

The AAL5 Receive LFB is placed in the processing chain after the ATM Policer or the ATM Header Classifier LFB. The AAL5 Receive LFB receives primarily ATM SDU from the previous LFB and performs reassembly of SDU to form AAL5 frames. The sequence of actions that configures AAL5 Receive LFB and cooperating ATM Policer LFB instance, and cooperation between these two LFBs is schematically depicted in Figure 3.3.

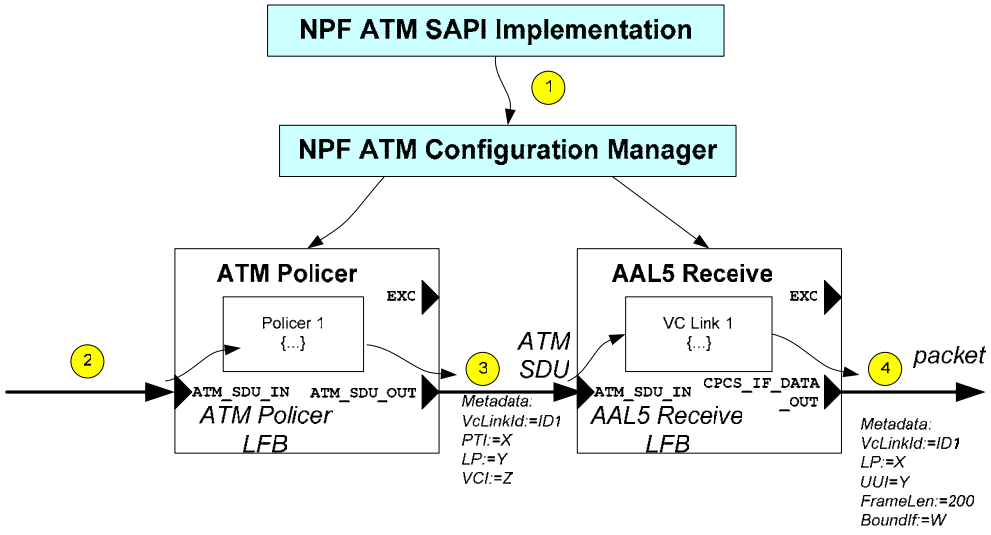

Figure 3.3: Cooperation between ATM Policer LFB and AAL5 Receive LFB

This figure shows part of example Forwarding Element that contains AAL5 Receive LFB and ATM Policer LFBs. These two blocks are connected in chain and configured by an ATM configuration manager LFB. The sequence of actions that configure a VC link may be defined as follows (see corresponding numbers in circles in the figure):

1. The NPF ATM SAPI is invoked to create a VC link. The system software below the NPF ATM SAPI assigns a VC link ID ‘ID1’ to the VC link and invokes the ATM configuration manager API to create an ATM VC link. This causes a VC link instance to be created in the AAL5 Receive LFB. An ATM Policer instance is created in the ATM Policer LFB to police the traffic sent on the VC link with VC link ID ‘ID1’.
2. The ATM Policer receives an ATM SDU from the ATM Header Classifier LFB. The ATM Policer uses the VC link ID to determine the policer instance to use to police the received ATM SDU. The ATM SDUs that are compliant to the traffic contract are passed to the AAL5 Receive LFB.
3. The AAL5 Receive LFB receives the ATM SDU from the ATM Policer and performs re-assembly to create the AAL5 frame.
4. The AAL5 packet reassembled by the AAL5 Receive LFB is passed to the next LFB in the chain over the `CPCS_IF_DATA_OUT` output. The higher layer interface bound to this VC link is also passed in the metadata along with the packet.

The EXC output of the AAL5 Receive LFB could be connected to an LFB that receives SDUs which could not be associated with any VC link or AAL5 frames which need to be discarded due to various reasons like CRC error, length mismatch, etc.

The AAL5 Receive LFB may be preceded in the topology by any LFB that can produce the information required by the AAL5 Receive LFB at its input. Downstream (not necessarily next) of the AAL5 Receive LFB, there should be LFBs that can utilize the information generated at output by AAL5 Receive LFB. The exact design and connections between the AAL5 Receive LFB and cooperating blocks is specific to the vendor that provides Forwarding Element design and FAPI implementation.

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 AAL5 Receive LFB in a forwarding element using a block type value for the AAL5 Receive LFB.

```
#define NPF_F_AAL5RECEIVE_LFB_TYPE  33
```

4.1.2 AAL5 Receive Configurations

4.1.2.1 ATM Virtual Channel Link Characteristics

The AAL5 Receive LFB requires below configurations for each virtual channel link.

- ATM VC link ID
- Maximum incoming CPCS PDU size
- AAL mode – messaging/streaming
- Whether corrupt SDU delivery is enabled and the maximum length of corrupt SDU to deliver.
- Reassembly timeout value
- SSCS configured for this VC link – NULL, assured data transfer, non assured data transfer, frame relay

4.2 Data Structures for Completion Callbacks

4.2.1 AAL5 Receive LFB Attributes query response

The attributes of an AAL5 Receive LFB are the following:

```
typedef struct {
   NPF_uint32_t   maxVcls;             /* Maximum possible VC links  */
   NPF_uint32_t   curNumVcls;          /* Current number of VC links */
}  NPF_F_AAL5ReceiveLFB_AttrQueryResponse_t;
```

The `maxVcls` field contains the maximum number of VC links supported in this AAL5 Receive LFB.

The `curNumVcls` field contains the number of VC links currently established in the AAL5 Receive 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_AAL5ReceiveErrorType_t error;/*Error code for this response*/
    union {
        /* NPF_F_AAL5ReceiveLFB_AttributesQuery() */
        NPF_F_AAL5ReceiveLFB_AttrQueryResponse_t lfbAttrQueryResponse;
    } u;
} NPF_F_AAL5ReceiveAsyncResponse_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_AAL5ReceiveCallbackData_t.
```

```
*/
typedef enum NPF_F_AAL5ReceiveCallbackType {
    NPF_F_AAL5RECEIVE_ATTR_QUERY = 1,
} NPF_F_AAL5ReceiveCallbackType_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_AAL5ReceiveCallbackData_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_AAL5ReceiveAsyncResponse_t structure, along with any other information
*/
typedef struct {
    NPF_F_AAL5ReceiveCallbackType_t type; /* Which function called? */
    /* ID of LFB generating callback*/
    NPF_IN NPF_BlockId_t          blockId;
    NPF_F_AAL5ReceiveAsyncResponse_t resp;/* response structures    */
} NPF_F_AAL5ReceiveCallbackData_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_AAL5RECEIVE_ATTR_QUERY`	`NPF_F_AAL5ReceiveLFB_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 AAL5 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 AAL5 Receive configuration and management APIs error codes. These codes are used in callbacks to deliver results of the requested operations.

```
#define NPF_AAL5RECEIVE_BASE_ERR (NPF_F_AAL5RECEIVE_LFB_TYPE * 100)
/* Asynchronous error codes (returned in function callbacks) */
typedef NPF_uint32_t NPF_F_AAL5ReceiveErrorType_t;

#define AAL5RECEIVE_ERR(n) ((NPF_F_AAL5Receive_ErrorType_t) \
                            (NPF_AAL5RECEIVE_BASE_ERR+ (n))

#define NPF_E_AAL5RECEIVE_INVALID_AAL5RECEIVE_BLOCK_ID\
                             AAL5RECEIVE_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_AAL5ReceiveCallbackFunc_t) (
   NPF_IN NPF_userContext_t               userContext,
   NPF_IN NPF_correlator_t                correlator,
   NPF_IN NPF_F_AAL5ReceiveCallbackData_t data);
```

5.2.1.1 Description

This callback function is for the application to register an asynchronous response handling routine to the AAL5 Receive API implementation. This callback function is intended to be implemented by the application, and be registered to the AAL5 Receive API implementation through the `NPF_F_AAL5ReceiveRegister` 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 AAL5 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_AAL5ReceiveRegister(
   NPF_IN NPF_userContext_t               userContext,
   NPF_IN NPF_F_AAL5ReceiveCallbackFunc_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 AAL5 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 AAL5 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 AAL5 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_AAL5ReceiveDeregister(
   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 AAL5 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_AAL5ReceiveLFB_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 AAL5 Receive LFB's attributes at a time. If the AAL5 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 AAL5 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 AAL5 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_AAL5RECEIVE_INVALID_AAL5RECEIVE_BLOCK_ID` – LFB ID is not an ID of LFB that has AAL5 Receive functionality.

The `lfbAttrQueryResponse` field of the union in the `NPF_F_AAL5ReceiveAsyncResponse_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.

[ITUI.363.5] "B-ISDN ATM Adaptation Layer Specification: Type 5 AAL", International Telecommunication Union (ITU-T).

Appendix A Header File Information

```
/*
 * This header file defines typedefs, constants and structures
 * for the NP Forum AAL5 Receive Functional API
 */

#ifndef __NPF_F_AAL5_RX_H__
#define __NPF_F_AAL5_RX_H__

#ifdef __cplusplus
extern "C" {
#endif

/* It is possible to use the FAPI Topology Discovery
   APIs to discover an AAL5 Receive LFB
   in a forwarding element.  */
#define NPF_F_AAL5RECEIVE_LFB_TYPE  33

#define NPF_AAL5RECEIVE_BASE_ERR (NPF_F_AAL5RECEIVE_LFB_TYPE * 100)
/* Asynchronous error codes (returned in function callbacks) */
typedef NPF_uint32_t NPF_F_AAL5ReceiveErrorType_t;
#define AAL5RECEIVE_ERR(n) ((NPF_F_AAL5ReceiveErrorType_t) \
                            (NPF_AAL5RECEIVE_BASE_ERR+ (n))
#define NPF_E_AAL5RECEIVE_INVALID_AAL5RECEIVE_BLOCK_ID      AAL5RECEIVE_ERR
(0)

/*************************************************************
 * Enumerations and types for AAL5 Rx attributes and          *
 * completion callback data types                             *
 *************************************************************/

/* The attributes of an AAL5 Receive LFB */
typedef struct {
      NPF_uint32_t   maxVcls;        /* Maximum possible VC links  */
      NPF_uint32_t   curNumVcls;     /* Current number of VC links */
}  NPF_F_AAL5ReceiveLFB_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_AAL5ReceiveErrorType_t error; /*Error code for response*/
    union {
        /* NPF_F_AAL5ReceiveLFB_AttributesQuery() */
        NPF_F_AAL5ReceiveLFB_AttrQueryResponse_t lfbAttrQueryResponse;
    } u;
} NPF_F_AAL5ReceiveAsyncResponse_t;

/*
* Completion Callback Types, to be found in the callback
* data structure, NPF_F_AAL5ReceiveCallbackData_t.
*/
typedef enum NPF_F_AAL5ReceiveCallbackType {
    NPF_F_AAL5RECEIVE_ATTR_QUERY = 1,
} NPF_F_AAL5ReceiveCallbackType_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_AAL5ReceiveAsyncResponse_t structure, along with any other information
*/
typedef struct {
    NPF_F_AAL5ReceiveCallbackType_t type; /* Which function called? */
    /* ID of LFB generating callback*/
    NPF_IN NPF_BlockId_t          blockId;
    NPF_F_AAL5ReceiveAsyncResponse_t resp; /* response structures   */
} NPF_F_AAL5ReceiveCallbackData_t;

/* Type for a callback function to be registered with AAL5 Rx */
typedef void (*NPF_F_AAL5ReceiveCallbackFunc_t) (
      NPF_IN NPF_userContext_t userContext,
      NPF_IN NPF_correlator_t correlator,
      NPF_IN NPF_F_AAL5ReceiveCallbackData_t data);


/* Completion Callback Registration Function */
NPF_error_t NPF_F_AAL5ReceiveRegister (
      NPF_IN NPF_userContext_t userContext,
      NPF_IN NPF_F_AAL5ReceiveCallbackFunc_t callbackFunc,
      NPF_OUT NPF_callbackHandle_t *callbackHandle);

/* Completion Callback Deregistration Function */
NPF_error_t NPF_F_AAL5ReceiveDeregister (
      NPF_IN NPF_callbackHandle_t callbackHandle);

/* LFB Attributes Query Function */
NPF_error_t NPF_F_AAL5ReceiveLFB_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_AAL5_RX_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	IDT	Sensory Networks
AMCC	Infineon Technologies AG	Sun Microsystems
Analog Devices	Intel	Teja Technologies
Cypress Semiconductor	IP Fabrics	TranSwitch
Enigma Semiconductor	IP Infusion	U4EA Group
Ericsson	Motorola	Wintegra
Flextronics	Mercury Computer Systems	Xelerated
Freescale Semiconductor	Nokia	Xilinx
HCL Technologies	NTT Electronics	
Hifn	PMC-Sierra	