ATM Adaptation Layer 1 (AAL1) Receive LFB and Functional API Implementation Agreement

January 23, 2006
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	01/23/06	Rev 1.0 of the AAL1 Receive LFB and Functional API Implementation Agreement. Source: npf2006.020.00.

2 Introduction

This implementation agreement defines the ATM Adaptation Layer 1 (AAL1) Receive LFB and lists the configurations required by the LFB. The AAL1 provides a mechanism to its service users to transfer and deliver service data units with a constant bit rate across an ATM network.

The AAL1 receive LFB receives ATM SDUs from the ATM layer over VCs configured for AAL1 service. The AAL1 Receive LFB performs the SAR and CS sublayer functions required for re-assembly of the user information, de-blocking of the reassembled information, handling of lost/misinserted/delayed cells, determination of the source data structure and recovery of the source clock frequency as per ITU recommendation I.363.1.

The AAL1 VCs are bound to PDH interfaces for interworking between the TDM and ATM domains. Interworking between the ATM and PDH interfaces is performed by binding AAL1 VCs to PDH interfaces using a parent-child relationship as shown below. A given AAL1 VC may be bound to one and only one PDH interface.

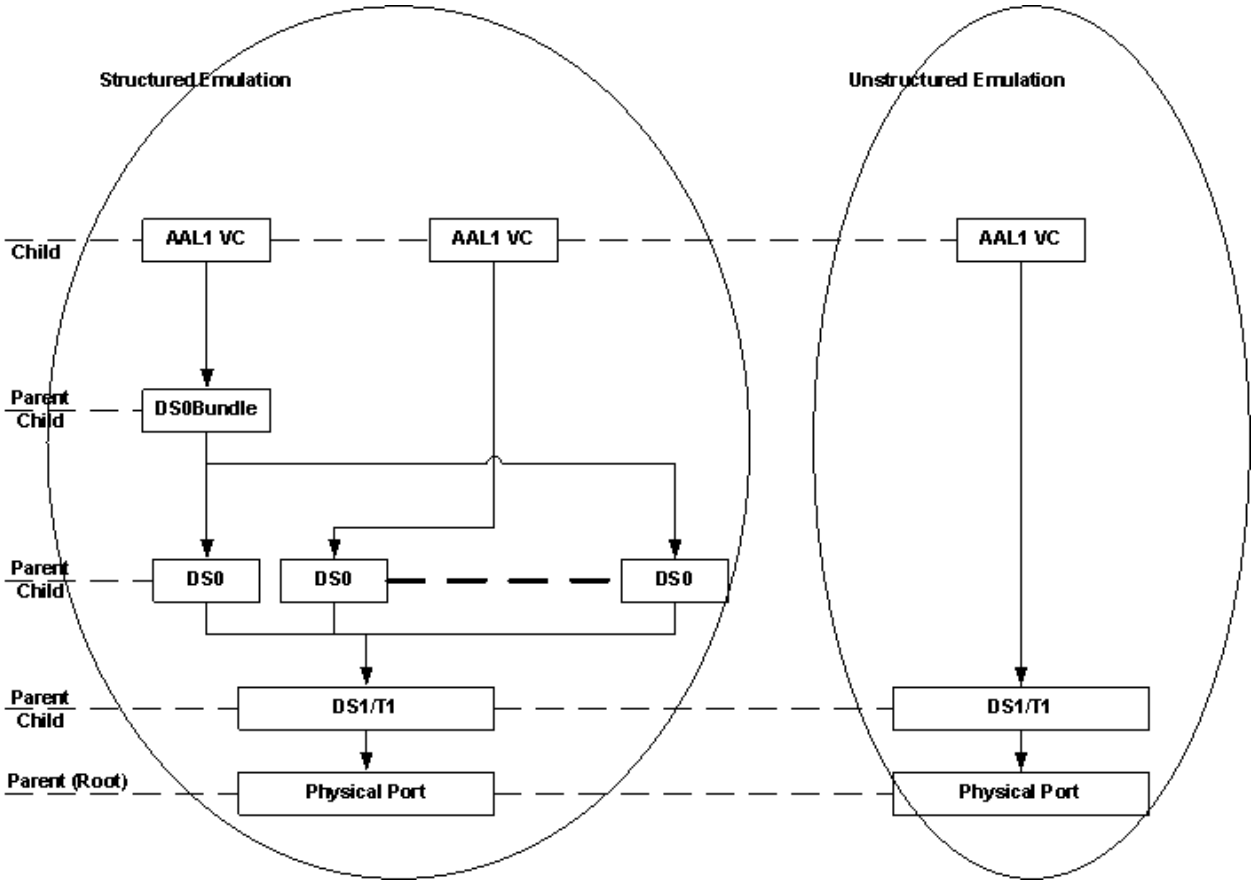

Figure 2.1: Example of PDH interface to AAL1 VC interoworking to provide structured and unstructure circuit emulation services

2.1 Acronyms

- **ATM**: Asynchronous Transfer Mode
- **AAL**: ATM Adaptation Layer
- **AAL1**: ATM Adaptation Layer Type 1
- **API**: Application Programming Interface
- **CS**: Convergence Sublayer

- **FEC**: Forward error correction
- **FAPI:** Functional API
- **IA**: Implementation Agreement
- **ID**: Identifier
- **LFB**: Logical Functional Block
- **NNI**: Network Node Interface
- **PTI**: Payload Type Indicator
- **PVC**: Permanent Virtual Connection
- **RTS**: Residual Time Stamp
- **SDT**: Structured Data Transfer
- **SRTS**: Synchronous Residual Time Stamp
- **SAR**: Segmentation and Reassembly
- **SDU**: Service Data Unit
- **SSCS:** Service specific convergence sublayer
- **UNI**: User Network Interface
- **VC**: Virtual Channel

2.2 Assumptions

The AAL1 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 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 the 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 AAL1 Receive LFB Description

The AAL1 Receive LFB receives ATM SDUs from the ATM Header Classifier LFB and performs the receive side AAL1 SAR and CS functions. The TDM payload extracted from the received cell stream is sent to the next LFB in the processing chain along with interface ID of the PDH interface on which the TDM data is to played out.

The LFB may contain multiple instances of VC links identified by their VC Link IDs. ATM SDUs received from the ATM header classifier are associated with the corresponding VC link instance using the VC Link ID signaled as metadata by the ATM header classifier LFB. A VC link instance may be bound to the PDH interface that is emulated by this VC link.

Such virtual link instances are depicted in Figure 3.2 below. The maximum number of VC links is an attribute of the AAL1 Receive LFB and may be queried as such.

The AAL1 Receive LFB is modeled as shown in Figure 3.1:

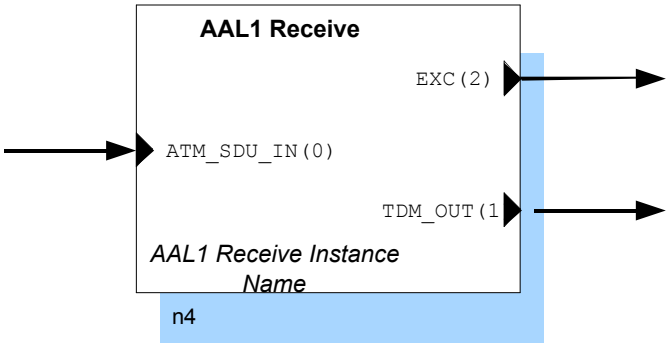

Figure 3.1: AAL1 Receive LFB

The AAL1 Receive LFB maintains the following statistics for each virtual link:

- Number of cells received with header error.
- Number of cells received with sequence number errors.
- Number of times reassembly buffer overflow occurred.
- Number of times reassembly buffer underflow occurred.
- Number of times AAL1 received LFB encountered unexpected structure pointers.
- Number of times AAL1 receive LFB encountered parity error when structure pointer was as expected.

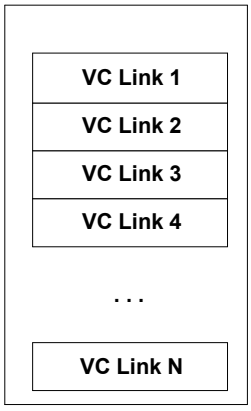

Figure 3.2: Virtual Link Instances

The AAL1 Receive LFB may contain a buffer associated with each VC link to handle the cell delay variation on the VC link. The size of the buffer associated with each VC link may be configured using the functions provided by the ATM Configuration Manager FAPI. The AAL1 Receive LFB may be required to insert dummy bits into the buffer in order to maintain bit count integrity in the event of an underflow. In the event of an overflow of such a buffer, the AAL1 receive LFB may need to drop an appropriate number of bits.

The AAL1 receive LFB may perform source clock recovery using the configured clock recovery mechanism like SRTS, adaptive clock recovery etc. The recovered clocking information may be used to drive the clocking of the local PDH interface. The mechanisms used to deliver the recovered clock to the PDH interface framers is outside the scope of NPF. The format in which the TDM payload is generated on the TDM_OUT output of the LFB is outside the scope of NPF.

3.1 AAL1 Receive LFB Inputs

Table 3.1: AAL1 Receive LFB Inputs

Symbolic Name	Input ID	Description
ATM_SDU_IN	0	This is the only input for the AAL1 Receive LFB and is used to receive ATM SDUs from the ATM Header classifier.

3.1.1 Metadata Required

Table 3.2: Input Metadata for AAL1 Receive LFB

Metadata tag	Access method	Description
META_VCL_ID	Read-and-consume	Metadata identifying the VC link on which the ATM cell was received.

3.2 AAL1 Receive Outputs

Table 3.3: AAL1 Receive LFB Outputs

Symbolic Name	Output ID	Description
TDM_OUT	1	This is the normal output for the AAL1 Receive LFB through with the TDM payload is passed to the next LFB in the processing chain.
EXC	2	The ATM SDU is sent to this output if processing failed due to errors.

3.2.1.1 Metadata Produced

Table 3.4: Output Metadata for AAL1 Receive LFB

Metadata tag	Access method	Description
META_IF_ID	Write	Metadata identifying the TDM interface on which the TDM payload has to be played out.

3.3 Accepted Inputs

The AAL1 Receive LFB can accept ATM cells received over UNI or NNI.

3.4 Cell Modifications

The AAL1 Receive LFB extracts the 47-octet data from ATM SDU's by extracting the SAR PDU Header. The SAR PDU payload is then subject to CS processing. The CS sublayer processing de-blocks the SAR PDU payload to regenerate the TDM payload.

3.5 Relationship with Other LFBs

The AAL1 receive LFB is placed in the processing chain after the ATM Header Classifier LFB and receives ATM SDUs received over VC links carrying AAL1 traffic.

The EXC output of the AAL1 receive LFB could be connected to an LFB that receives ATM SDUs which are discarded by the AAL1 receive LFB due to various errors like SAR PDU header errors etc. Depending on system design this may be either dropper, or other LFB that makes a decision how to utilize such SDUs.

The AAL1 Receive LFB may be preceded in the topology by any LFB that can produce the information required by the AAL1 Receive LFB at its input. Downstream (not necessarily next) of the AAL1 Receive LFB, there should be LFBs that can utilize the information generated at output by AAL1 Receive LFB. The exact design and connections between the AAL1 receive LFB and cooperating blocks is specific to the vendor that provides Forwarding Element design and FAPI implementation.

The sequences of actions that configure AAL1 receive LFB and TDM Transmit LFB instance, and cooperation between these two LFBs is schematically depicted in Figure 3.3.

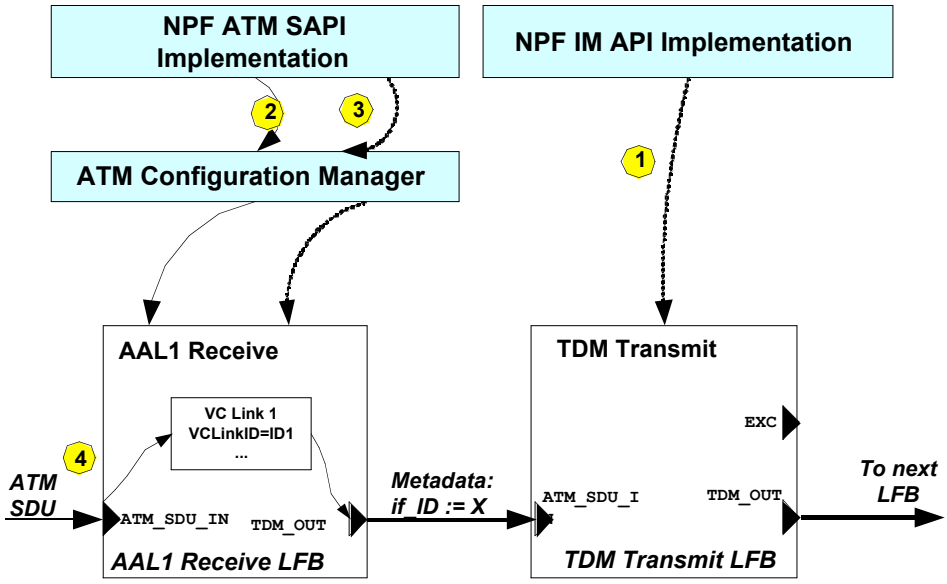

Figure 3.3: Cooperation between AAL1 Receive and TDM Transmit LFB

This figure shows part of an example Forwarding Element that contains ATM configuration manager, AAL1 Receive LFB and TDM Transmit LFBs. The AAL1 receive LFB and the TDM Transmit LFB is connected in chain. The sequence of actions that configure a TDM interface and bind a VC link to the TDM interface may be defined as follows (see corresponding numbers in circles in the figure):

1. The NPF IM API is invoked to create a PDH interface. The system software below the NPF IM API assigns an interface ID X to the interface and invokes the TDM Transmit LFB FAPI to configure the interface.
2. The NPF ATM SAPI API is invoked to create an ATM VC link instance. The system software below the NPF ATM SAPI assigns a VC link ID ('ID1') to the VC link and invokes the ATM configuration manager FAPI to create the VC link. The ATM configuration manager FAPI call leads to creation of a VC link instance in the AAL1 Receive LFB.

3. The NPF ATM SAPI API is invoked to bind the ATM VC link to the PDH interface. The system software below the NPF ATM SAPI invokes the ATM configuration manager FAPI to bind the VC link to the PDH interface.
4. An ATM SDU is received by the AAL1 received LFB over VC link ID1. The AAL1 receive LFB performs the AAL1 SAR and CS functions on the received ATM SDU and passes the TDM payload along with interface ID of the bound TDM interface to the TDM Transmit 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 AAL1 Receive LFB in a forwarding element using a block type value for the AAL1 Receive LFB.

```
#define NPF_F_AAL1RECEIVE_LFB_TYPE  44
```

4.1.2 AAL1 Virtual Channel Link Characteristics

The AAL1 Receive LFB requires below configurations for each VC link.

- VC link ID
- Bound PDH Interface ID
- AAL1 sub type – NULL, voice band, synchronous circuit emulation, asynchronous circuit emulation, high quality audio, video
- Rate of the CBR service and rate multiplier
- Clock recovery type – synchronous, asychronous using SRTS, adaptive clock recovery
- Forward error correction type – none, loss sensitive FEC, delay sensitive FEC
- CAS transport mode – basic mode, E1 mode, DS1 superframe, DS1 extended super frame, J2 mode
- Whether partial cell fill mode enabled and user information size in partially filled cells if configured.
- CDV tolerance and reassembly buffer size
- Cell loss integration period
- AAL associated with the VC link
- Adminstrative status – Up/Down/Testing

4.2 Data Structures for Completion Callbacks

4.2.1 AAL1 Receive LFB Attributes query response

The attributes of an AAL1 Receive LFB are the following:

```
typedef struct {
   NPF_uint32_t   maxVcl;               /* Maximum possible VC links     */
   NPF_uint32_t   curNumVcl;            /* Current number of VC links    */
   NPF_uint32_t   maxReasmBfrSize;      /* Max size of reassembly buffer */
   NPF_uint32_t   totReasmBfrSize;      /* Max size of reassembly buffer */
}  NPF_F_AAL1ReceiveLFB_AttrQueryResponse_t;
```

The `maxVcl` field contains the maximum number of VC links supported in this AAL1 Receive LFB. The `curNumVcl` field contains the number of VC link configured in the AAL1 Receive LFB. The `maxReasmBfrSize` indicates the maximum reassembly buffer size that may be configured for an emulated circuit and the `totReasmBfrSize` indicates the total memory available for buffering in this 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_AAL1ReceiveErrorType_t error; /* Error code for response  */
    union {
        /* NPF_F_AAL1ReceiveLFB_AttributesQuery() */
        NPF_F_AAL1ReceiveLFB_AttrQueryResponse_t   lfbAttrQueryResponse;
    } u;
} NPF_F_AAL1ReceiveAsyncResponse_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_AAL1ReceiveCallbackData_t.
*/
typedef enum NPF_F_AAL1ReceiveCallbackType {
    NPF_F_AAL1RECEIVE_LFB_ATTR_QUERY = 1,
} NPF_F_AAL1ReceiveCallbackType_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_AAL1ReceiveCallbackData_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_F_AAL1ReceiveAsyncResponse_t structure, along with any other
* information
*/
typedef struct {
    NPF_F_AAL1ReceiveCallbackType_t type; /* Which function called?  */
    NPF_BlockId_t          blockId;/* ID of LFB generating callback */
    NPF_F_AAL1ReceiveAsyncResponse_t  resp;  /* response structure   */
} NPF_F_AAL1ReceiveCallbackData_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_AAL1RECEIVE_ATTR_QUERY`	`NPF_F_AAL1ReceiveLFB_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 AAL1 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).

4.4.2 LFB Specific Error Codes

This section defines AAL1 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_AAL1ReceiveErrorType_t;

   #define NPF_AAL1RECEIVE_BASE_ERR (NPF_F_AAL1RECEIVE_LFB_TYPE * 100)
   #define NPF_E_AAL1RECEIVE_INVALID_AAL1RECEIVE_BLOCK_ID
                                          (NPF_AAL1RECEIVE_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_AAL1ReceiveCallbackFunc_t) (
   NPF_IN NPF_userContext_t               userContext,
   NPF_IN NPF_correlator_t                correlator,
   NPF_IN NPF_F_AAL1ReceiveCallbackData_t data);
```

5.2.1.1 Description

This callback function is for the application to register an asynchronous response handling routine to the AAL1 Receive API implementation. This callback function is intended to be implemented by the application, and be registered to the AAL1 Receive API implementation through the `NPF_F_AAL1ReceiveRegister` 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 AAL1 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_AAL1ReceiveRegister(
      NPF_IN NPF_userContext_t                   userContext,
      NPF_IN NPF_F_AAL1ReceiveCallbackFunc_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 AAL1 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 AAL1 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 AAL1 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_AAL1ReceiveDeregister(
      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 AAL1 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_AAL1ReceiveLFB_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 AAL1 Receive LFB's attributes at a time. If the AAL1 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 AAL1 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 AAL1 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_AAL1RECEIVE_INVALID_AAL1RECEIVE_BLOCK_ID` – LFB ID is not an ID of LFB that has AAL1 Receive functionality

The `lfbAttrQueryResponse` field of the union in the `NPF_F_AAL1ReceiveAsyncResponse_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 AAL1 Receive Functional API
 */
#ifndef __NPF_F_AAL1_RECEIVE_H__
#define __NPF_F_AAL1_RECEIVE_H__

#ifdef __cplusplus
extern "C" {
#endif

/* AAL1 Receive LFB Type ID */
#define NPF_F_AAL1RECEIVE_LFB_TYPE       44

/* Asynchronous error codes (returned in function callbacks) */
   typedef NPF_uint32_t NPF_F_AAL1Receive_ErrorType_t;

   #define NPF_AAL1RECEIVE_BASE_ERR (NPF_F_AAL1RECEIVE_LFB_TYPE * 100)
   #define NPF_E_AAL1RECEIVE_INVALID_AAL1RECEIVE_BLOCK_ID
                                          (NPF_AAL1RECEIVE_BASE_ERR + 0)

/*************************************************************
 * Enumerations and types for AAL1 Receive LFB                *
 *************************************************************/
typedef struct {
    NPF_uint32_t   maxVcl;              /* Maximum possible VC links     */
    NPF_uint32_t   curNumVcl;           /* Current number of VC links    */
    NPF_uint32_t   maxReasmBfrSize;     /* Max size of reassembly buffer */
    NPF_uint32_t   totReasmBfrSize;     /* Max size of reassembly buffer */
}  NPF_F_AAL1ReceiveLFB_AttrQueryResponse_t;

/*
 * Completion Callback Types, to be found in the callback
 * data structure, NPF_F_AAL1ReceiveCallbackData_t.
 */
typedef enum NPF_F_AAL1ReceiveCallbackType {
    NPF_F_AAL1RECEIVE_LFB_ATTR_QUERY    = 1,
} NPF_F_AAL1ReceiveCallbackType_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_AAL1ReceiveErrorType_t error;
    union {
        /* NPF_F_AAL1ReceiveLFB_AttributesQuery() */
        NPF_F_AAL1ReceiveLFB_AttrQueryResponse_t   lfbAttrQueryResponse;
    } u;
} NPF_F_AAL1ReceiveAsyncResponse_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_AAL1ReceiveAsyncResponse_t structure, along with any other information
*/
typedef struct {
```

```
    NPF_F_AAL1ReceiveCallbackType_t type; /* Which function called?  */
    NPF_IN NPF_BlockId_t          blockId;/* ID of LFB generating callback */
    NPF_F_AAL1ReceiveAsyncResponse_t  resp;  /* response structure */
} NPF_F_AAL1ReceiveCallbackData_t;

typedef void (*NPF_F_AAL1ReceiveCallbackFunc_t) (
    NPF_IN NPF_userContext_t             userContext,
    NPF_IN NPF_correlator_t              correlator,
    NPF_IN NPF_F_AAL1ReceiveCallbackData_t    data);

/*****************************************************************
 * AAL1 Receive LFB Registration/De-registration Functions       *
 *****************************************************************/

NPF_error_t NPF_F_AAL1ReceiveRegister(
    NPF_IN NPF_userContext_t             userContext,
    NPF_IN NPF_F_AAL1ReceiveCallbackFunc_t callbackFunc,
    NPF_OUT NPF_callbackHandle_t        *callbackHandle);

NPF_error_t NPF_F_AAL1ReceiveDeregister(
    NPF_IN NPF_callbackHandle_t    callbackHandle);

/*****************************************************************
 * AAL1 Receive LFB optional functions                           *
 *****************************************************************/
NPF_error_t NPF_F_AAL1ReceiveLFB_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_AAL1_RECEIVE_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:
Pat Carr, Wintegra
Pål Dammvik, Ericsson
Patrik Herneld, Ericsson
Ajay Kamalvanshi, Nokia
Arthur Mackay, Freescale
Michael Persson, Ericsson
Tiberu Petrica, Freescale
John Renwick, Agere Systems
Vedvyas Shanbhogue (ed.), Intel
Roger Smith, Wintegra
Keith Williamson, Motorola
Paul Wilson, Freescale
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