AAL2 CPS 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

1 Revision History 3
2 Introduction 4
2.1 Acronyms 4
2.2 Assumptions 4
2.3 Scope 4
2.4 External Requirements and Dependencies 4
3 AAL2 CPS Receive Description 6
3.1 AAL2 CPS Receive Inputs 7
3.2 AAL2 CPS Receive Outputs 8
3.3 Accepted Inputs 8
3.4 Cell Modifications 9
3.5 Relationship with Other LFBs 9
4 Data Types 11
4.1 Common LFB Data Types 11
4.2 Data Structures for Completion Callbacks 11
4.3 Data Structures for Event Notifications 12
4.4 Error Codes 12
5 Functional API (FAPI) 14
5.1 Required Functions 14
5.2 Conditional Functions 14
5.3 Optional Functions 16
6 References 17
Appendix A Header File Information 18
Appendix B Acknowledgements 20
Appendix C List of companies belonging to NPF during approval process 21

Table of Figures

Figure 3.1: AAL2 CPS Receive LFB 6
Figure 3.2: AAL2 Path Instances 6
Figure 3.3: AAL2 Channel Instances and associated AAL2 path instances 7
Figure 3.4: Cooperation between AAL2 CPS Receive and ATM Policer LFB 9

List of Tables

Table 3.1: AAL2 CPS Receive LFB Inputs 7
Table 3.2: Input Metadata for AAL2 CPS Receive LFB 8
Table 3.3: AAL2 CPS Receive LFB Outputs 8
Table 3.4: Output Metadata for CPS_SDU_OUT output 8
Table 3.5: Output Metadata for EXC output 8
Table 4.1: Callback type to callback data mapping table 12

1 Revision History

Revision	Date	Reason for Changes
1.0	08/16/2005	Rev 1.0 of the AAL2 CPS Receive LFB and Functional API Implementation Agreement. Source: npf2004.157.09.

2 Introduction

This contribution defines the AAL2 CPS Receive LFB and lists configurations that are required in the LFB.

2.1 Acronyms

- **ATM**: Asynchronous Transfer Mode
- **API**: Application Programming Interface
- **AAL2**: ATM Adaptation Layer Type 2
- **CPS:** Common Part Sublayer
- **CID:** Channel ID
- **FAPI:** Functional API
- **IA**: Implementation Agreement
- **ID**: Identifier
- **LFB**: Logical Functional Block
- **LI:** Length Indicator
- **NNI**: Network Node Interface
- **OSF**: Offset Field
- **SDU**: Service Data Unit
- **SSCS:** Service specific convergence sublayer
- **STF:** Start Field
- **UNI**: User Network Interface
- **UUI**: User to User Information
- **VC**: Virtual Channel

2.2 Assumptions

The AAL2 CPS 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 processing ATM SDU to extract and process CPS packets interleaved on the ATM cell stream. 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 AAL2 CPS Receive Description

The AAL2 CPS Receive LFB performs receives ATM SDUs from the previous LFB and extracts CPS packets interleaved on the ATM cell stream. The AAL2 CPS Receive LFB is modeled as shown in Figure 3.1

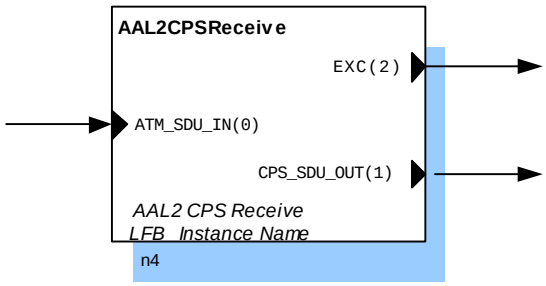

Figure 3.1: AAL2 CPS Receive LFB

The LFB may contain multiple instances of AAL2 paths that are identified by unique VC link IDs identifying the AAL2 path. Incoming packets are assigned to appropriate AAL2 path instance according to metadata received with input packet. Such instances are depicted in Figure 3.2. The maximum number of AAL2 paths is an attribute of the AAL2 CPS Receive LFB and may be queried as such.

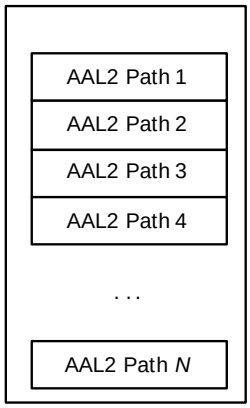

Figure 3.2: AAL2 Path Instances

The AAL2 CPS Receive LFB maintains the following counters for each AAL2 path:

- Number of ATM cells with the errored parity in STF.
- Number of ATM cells with wrong sequence number in STF.
- Number of ATM cells received where octets expected for a CPS-Packet overlapping into this CPS-PDU does not match the information in the STF.
- Number of ATM cells received with the OSF of the STF contains a value 48 or greater.
- Number of CPS packets with Header Error Control (HEC) error.
- Number of CPS packets where the length of the received CPS-Packet Payload (CPS-SDU) exceeds the maximum length indicated in "Max_SDU_Deliver_Length".
- Number of times reassembly cancelled.
- Number of timer the HEC Code of a CPS-Packet header that was overlapping a CPS-PDU boundary indicates transmission errors in the CPS-Packet header.
- Number of CPS packets received with UUI field containing a value ("28" or "29").

- Number of CPS packet discarded due to unknown CID.
- Number of CPS packets received.
- Number of CPS packet bytes received.

Each AAL2 path configured in the AAL2 CPS Receive LFB may be associated with upto 255 AAL2 channels that are identified by unique VC Link ID (identifying the AAL2 path) and a channel identifier, CID. Incoming packets are assigned to appropriate AAL2 channel instances according to the VC Link ID contained in the metadata received with the input ATM SDU and the CID in the CPS packets extracted from the ATM SDU. Such instances are depicted in Figure 3.3 below. The maximum number of channels supported across all AAL2 paths is an attribute of the AAL2 CPS Receive LFB and may be queried as such.

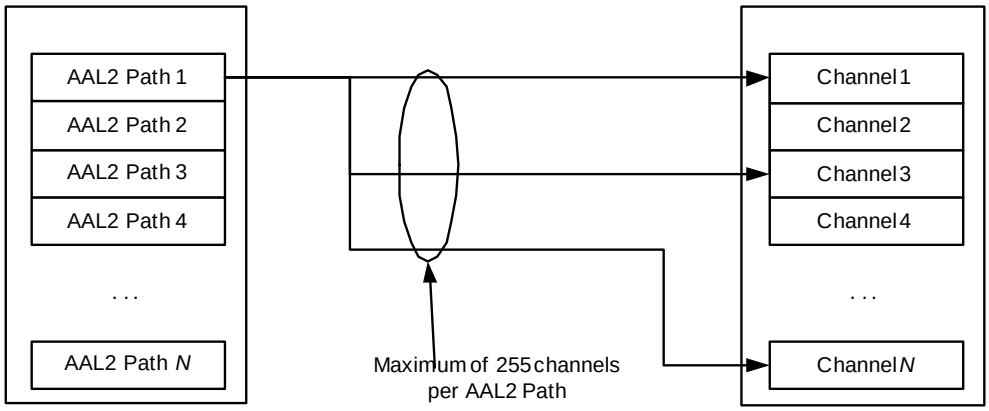

Figure 3.3: AAL2 Channel Instances and associated AAL2 path instances

The AAL2 CPS Receive LFB maintains the following counters for each AAL2 channel:

- Number of CPS packets where the length of the received CPS-Packet Payload (CPS-SDU) exceeds the maximum length indicated in "Max_SDU_Deliver_Length".
- Number of times reassembly cancelled.
- Number of timer the HEC Code of a CPS-Packet header that was overlapping a CPS-PDU boundary indicates transmission errors in the CPS-Packet header.
- Number of CPS packets received with UUI field containing a value ("28" or "29").
- Number of CPS packets received.
- Number of CPS packet bytes received.

3.1 AAL2 CPS Receive Inputs

Table 3.1: AAL2 CPS Receive LFB Inputs

Symbolic Name	Input ID	Description
`ATM_SDU_IN`	0	This is the only input for the AAL2 CPS Receive LFB and is used to receive ATM SDUs for extraction of CPS packets from the ATM cell stream.

3.1.1 Metadata Required

Table 3.2: Input Metadata for AAL2 CPS Receive LFB

Metadata tag	Access method	Description
`META_VCL_ID`	Read-and-consumed	Metadata identifying the VC link of AAL2 path on which the ATM cell was received.

3.2 AAL2 CPS Receive Outputs

Table 3.3: AAL2 CPS Receive LFB Outputs

Symbolic Name	Output ID	Description
`CPS_SDU_OUT`	1	This is the normal output for the AAL2 CPS Receive LFB. CPS SDU extracted from the received CPS packets are sent over this output to the next LFB in the chain.
`EXC`	2	The ATM SDU, CPS packet or a partially CPS packet is sent to this output if the processing failed due to errors.

3.2.1.1 Metadata Produced on CPS_SDU_OUT output

Table 3.4: Output Metadata for CPS_SDU_OUT output

Metadata tag	Access method	Description
`META_CHNL_ID`	Write	Metadata identifying the AAL2 channel on which the packet was received.
`META_CPS_UUI`	Write	The UUI received of the CPS packet.
`META_LI`	Write	The length of the extracted CPS SDU.
`META_SSCS_TYPE`	Write	The SSCS to process this CPS SDU.
`META_BOUND_IF`	Write	The handle of the higher layer interface bound to this AAL2 channel.

3.2.1.2 Metadata Produced on EXC output

Table 3.5: Output Metadata for EXC output

Metadata tag	Access method	Description
`META_EXC_CAUSE`	Write	Metadata specifying the reason for sending to the `EXC` output.
`META_EXC_OUT_TYPE`	Write	The type of object sent to the `EXC` output.

3.3 Accepted Inputs

The AAL2 CPS Receive LFB can accept ATM cell SDUs received over UNI or NNI.

3.4 Cell Modifications

The AAL2 CPS Receive LFB extracts AAL2 CPS packets from the received ATM SDUs. The CPS packet header is verified for errors. If there is an error the CPS packet and/or the ATM SDU is discarded by sending over the EXC output. If the CPS packet was extracted successful, the AAL2 CPS SDU is passed to the next LFB in the chain for further processing.

3.5 Relationship with Other LFBs

The AAL2 CPS Receive LFB is placed in the processing chain after the ATM Policer or the ATM Header Classifier LFB. The AAL2 CPS Receive LFB receives primarily ATM SDU from the previous LFB and extracts AAL2 CPS packets from the received SDUs. The sequence of actions that configures AAL2 CPS Receive LFB and cooperating ATM Policer LFB instance, and cooperation between these two LFBs is schematically depicted in Figure 3.4.

The AAL2 CPS Receive LFB may be preceded in the topology by any LFB that can produce the information required by the AAL2 CPS Receive LFB at its input. Downstream (not necessarily next) of the AAL2 CPS Receive LFB, there should be LFBs that can utilize the information generated at output by AAL2 CPS Receive LFB. The exact design and connections between the AAL2 CPS Receive LFB and cooperating blocks is specific to the vendor that provides Forwarding Element design and FAPI implementation.

The EXC output of the AAL2 CPS Receive LFB could be connected to an LFB that receives SDUs which could not be associated with any AAL2 path instance or partially/fully extracted AAL2 CPS packets which need to be discarded due to various reasons like CRC error, length mismatch, etc.

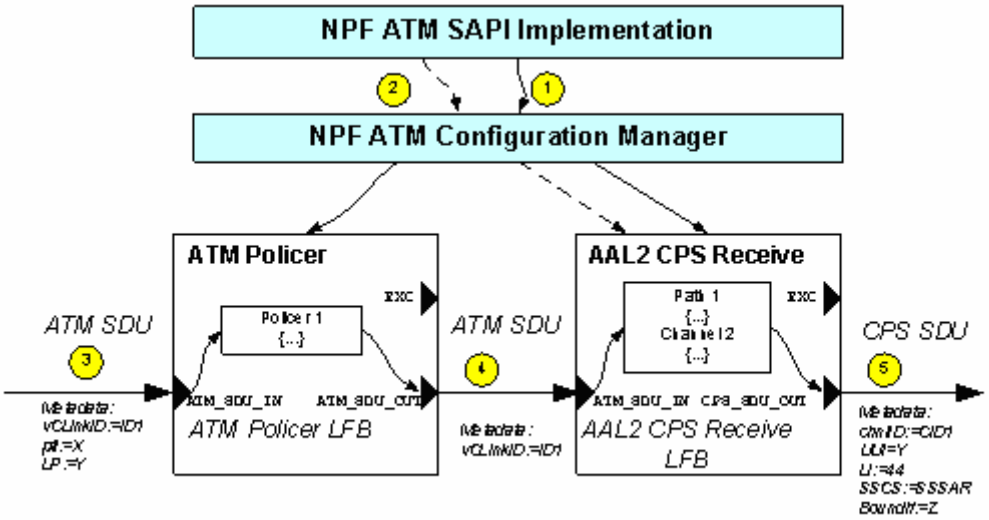

Figure 3.4: Cooperation between AAL2 CPS Receive and ATM Policer LFB

This figure shows part of example Forwarding Element that contains AAL2 CPS Receive LFB and ATM Policer LFBs. These two blocks are connected in chain and configured by the ATM configuration manager LFB. The sequence of actions that configures an AAL2 path and AAL2 channel on the interface 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 to form the AAL2 path. The system software below the NPF ATM SAPI assigns a VC Link ID ‘ID1’ to the AAL2 path and invokes the ATM configuration manager LFB FAPI to create the AAL2 path. This causes an AAL2 path instance to be created in the AAL2 CPS Receive LFB. An ATM Policer instance is created in the ATM Policer LFB to policer the traffic received on the VC Link ‘ID1’.
2. The NPF ATM SAPI is invoked to create an AAL2 channel. The system software below the NPF ATM SAPI assigns a channel ID ‘CID1’ to this AAL2 channel and invokes the ATM

configuration manager FAPI to create the channel. This causes a channel instance to be created in the AAL2 CPS Receive LFB and associated with the AAL2 path instance with VC Link ID ‘ID1’.

3. 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 AAL2 CPS Receive LFB.
4. The AAL2 CPS Receive LFB receives the ATM SDU from the ATM Policer and extracts AAL2 CPS packet from the ATM SDU (the CPS packet may be contained entirely in on ATM SDU or may be spread across more than one ATM SDU).
5. The AAL2 CPS SDU extracted from the AAL2 packets by the AAL2 CPS Receive LFB is passed to the next LFB in the chain over the `CPS_SDU_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 AAL2 CPS Receive LFB in a forwarding element using a block type value for the AAL2 CPS Receive LFB.

```
#define NPF_F_AAL2CPSRX_LFB_TYPE     37
```

4.1.2 AAL2 Path Characteristics

The AAL2 CPS Receive LFB requires below configurations for each configured AAL2 path.

- VC Link ID identifying the AAL2 path.
- The maximum size CPS-SDU, in octets, that is transported on any AAL2 channel of this AAL2 path. This can take on the values 45 or 64.

4.1.3 AAL2 Channel Characteristics

The AAL2 CPS Receive LFB requires below configurations for each configured AAL2 channel.

- AAL2 channel identifier (CID) of the channel.
- The VC Link ID identifying the associated AAL2 path.
- The maximum size CPS-SDU, in octets that can be transported on this AAL2 channel. Must be set to a value less than or equal to the corresponding configuration provided for the associated AAL2 path.
- The SSCS configured to process CPS packets received on this AAL2 channel – None, SAR SSCS, Trunking SSCS.
- Interface ID of higher layer interface bound to this AAL2 channel.
- Administrative status – Up/Down/Testing.

4.2 Data Structures for Completion Callbacks

4.2.1 AAL2 CPS Receive LFB Attributes query response

The attributes of an AAL2 CPS Receive LFB are the following:

```
typedef struct {
   NPF_uint32_t   maxPaths;             /* Maximum possible AAL2 paths     */
   NPF_uint32_t   maxChnls;             /* Maximum possible AAL2 channels  */
   NPF_uint32_t   curNumPaths;          /* Current number of AAL2 paths    */
   NPF_uint32_t   curNumChnls;          /* Current number of AAL2 channels */
}  NPF_F_AAL2CpsRxLFB_AttrQueryResponse_t;
```

The `maxPaths` field contains the maximum number of AAL2 paths supported in this AAL2 CPS Receive LFB. The `curNumPaths` field contains the number of AAL2 paths currently established in the AAL2 CPS Receive LFB. The `maxChnls` field contains the maximum number of AAL2 channels supported in this LFB. The `curNumChnls` field contains the number of AAL2 channels currently established 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_AAL2CpsRxErrorType_t error;  /* Error code for this response */
    union {
        /* NPF_F_AAL2CpsRxLFB_AttributesQuery() */
        NPF_F_AAL2CpsRxLFB_AttrQueryResponse_t    lfbAttrQueryResponse;
    } u;
} NPF_F_AAL2CpsRxAsyncResp_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_AAL2CpsRxCallbackData_t.
*/
typedef enum NPF_F_AAL2CpsRxCallbackType {
    NPF_F_AAL2CPSRX_ATTR_QUERY = 1,
} NPF_F_AAL2CpsRxCallbackType_t;
```

4.2.3.1 Callback Data

An asynchronous response contains an error/success code and a function-specific structure embedded in a union in the `NPF_F_AAL2CpsRxCallbackData_t` structure.

```
/*
* The callback function receives the following structure containing
* of an asynchronous responses from a function call. For the completed
* request, the error code is specified in the
* NPF_F_AAL2CpsRxAsyncResponse_t structure, along with any other
* information
*/
typedef struct {
    NPF_F_AAL2CpsRxCallbackType_t type;  /* Which function called?        */
    NPF_IN NPF_BlockId_t          blockId;/* ID of LFB generating callback */
    NPF_F_AAL2CpsRxAsyncResp_t     resp; /* Response structure            */
} NPF_F_AAL2CpsRxCallbackData_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_AAL2CPSRX_ATTR_QUERY`	`NPF_F_AAL2CpsRxLFB_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 AAL2 CPS 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 AAL2 CPS Receive Configuration and management APIs error codes. These codes are used in callbacks to deliver results of the requested operations.

```
#define NPF_AAL2CPSRX_BASE_ERR (NPF_F_AAL2CPSRX_LFB_TYPE * 100)
/* Asynchronous error codes (returned in function callbacks) */
typedef NPF_uint32_t NPF_F_AAL2CpsRx_ErrorType_t;
#define AAL2CPSRX_ERR(n) ((NPF_F_AAL2CpsRxErrorType_t) \
                                        (NPF_AAL2CPSRX_BASE_ERR+ (n))
#define NPF_E_AAL2CPSRX_INVALID_AAL2CPSRX_BLOCK_ID    AAL2CPSRX_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_AAL2CpsRxCallbackFunc_t) (
      NPF_IN NPF_userContext_t            userContext,
      NPF_IN NPF_correlator_t             correlator,
      NPF_IN NPF_F_AAL2CpsRxCallbackData_t      data);
```

5.2.1.1 Description

This callback function is for the application to register an asynchronous response handling routine to the AAL2 CPS Receive API implementation. This callback function is intended to be implemented by the application, and be registered to the AAL2 CPS Receive API implementation through the `NPF_F_AAL2CpsRxRegister` 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 AAL2 CPS 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_AAL2CpsRxRegister(
      NPF_IN NPF_userContext_t                  userContext,
      NPF_IN NPF_F_AAL2CpsRxCallbackFunc_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 AAL2 CPS 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 AAL2 CPS 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 AAL2 CPS 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_AAL2CpsRxDeregister(
      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 AAL2 CPS 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_AAL2CpsRxLFB_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 AAL2 CPS Receive LFB's attributes at a time. If the AAL2 CPS 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 AAL2 CPS 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 AAL2 CPS 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_AAL2CPSRX_INVALID_AAL2CPSRX_BLOCK_ID` – LFB ID is not an ID of LFB that has AAL2 CPS Receive functionality

The `lfbAttrQueryResponse` field of the union in the `NPF_F_AAL2CpsRxAsyncResponse_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 AAL2 CPS Receive Functional API
 */

#ifndef __NPF_F_AAL2_CPSRECEIVE_H__
#define __NPF_F_AAL2_CPSRECEIVE_H__

#ifdef __cplusplus
extern "C" {
#endif

/* It is possible to use the FAPI Topology Discovery
   APIs to discover an AAL2 CPS Receive LFB
   in a forwarding element.  */
#define NPF_F_AAL2CPSRX_LFB_TYPE    37

#define NPF_AAL2CPSRX_BASE_ERR (NPF_F_AAL2CPSRX_LFB_TYPE * 100)
/* Asynchronous error codes (returned in function callbacks) */
typedef NPF_uint32_t NPF_F_AAL2CpsRxErrorType_t;
#define AAL2CPSRX_ERR(n) ((NPF_F_AAL2CpsRxErrorType_t) \
                  (NPF_AAL2CPSRX_BASE_ERR+ (n))
#define NPF_E_AAL2CPSRX_INVALID_AAL2CPSRX_BLOCK_ID    AAL2CPSRX_ERR(0)

/*************************************************************
 * Enumerations and types for AAL2 CPS Receive attributes and   *
 * completion callback data types                               *
 *************************************************************/

/* The attributes of an AAL2 CPS Receive */
typedef struct {
   NPF_uint32_t   maxPaths;     /* Maximum possible AAL2 paths     */
   NPF_uint32_t   maxChnls;     /* Maximum possible AAL2 channels  */
   NPF_uint32_t   curNumPaths;  /* Current number of AAL2 paths    */
   NPF_uint32_t   curNumChnls;  /* Current number of AAL2 channels */
}  NPF_F_AAL2CpsRxLFB_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_AAL2CpsRxErrorType_t error;  /* Error code for this response */
    union {
        /* NPF_F_AAL2CpsRxLFB_AttributesQuery() */
        NPF_F_AAL2CpsRxLFB_AttrQueryResponse_t    lfbAttrQueryResponse;
    } u;
} NPF_F_AAL2CpsRxAsyncResp_t;


/*
* Completion Callback Types, to be found in the callback
* data structure, NPF_F_AAL2CpsRxCallbackData_t.
*/
typedef enum NPF_F_AAL2CpsRxCallbackType {
    NPF_F_AAL2CPSRX_ATTR_QUERY = 1,
} NPF_F_AAL2CpsRxCallbackType_t;
```

```
/*
* The callback function receives the following structure containing
* of an asynchronous responses from a function call. For the completed
* request, the error code is specified in the
* NPF_F_AAL2CpsRxAsyncResponse_t structure, along with any other
* information
*/
typedef struct {
    NPF_F_AAL2CpsRxCallbackType_t type;  /* Which function called?        */
    NPF_IN NPF_BlockId_t          blockId;/* ID of LFB generating callback */
    NPF_F_AAL2CpsRxAsyncResp_t    resp; /* Response structure             */
} NPF_F_AAL2CpsRxCallbackData_t;

/* Type for a callback function to be registered with LFB */
typedef void (*NPF_F_AAL2CpsRxCallbackFunc_t) (
      NPF_IN NPF_userContext_t userContext,
      NPF_IN NPF_correlator_t  correlator,
        NPF_IN NPF_F_AAL2CpsRxCallbackData_t data);

/* Completion Callback Registration Function */
NPF_error_t NPF_F_AAL2CpsRxRegister(
      NPF_IN NPF_userContext_t userContext,
      NPF_IN NPF_F_AAL2CpsRxCallbackFunc_t callbackFunc,
      NPF_OUT NPF_callbackHandle_t *callbackHandle);

/* Completion Callback Deregistration Function */
NPF_error_t NPF_F_AAL2CpsRxDeregister(
      NPF_IN NPF_callbackHandle_t callbackHandle);

/* LFB Attributes Query Function */
NPF_error_t NPF_F_AAL2CpsRxLFB_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_AAL2_CPSRECEIVE_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