AAL2 CPS Transmit 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 AAL2 CPS Transmit LFB and Functional API Implementation Agreement. Source: npf2004.158.08.

2 Introduction

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

2.1 Acronyms

- **ATM**: Asynchronous Transfer Mode
- **API**: Application Programming Interface
- **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
- **SDU**: Service Data Unit
- **STF**: Start Field
- **UNI**: User Network Interface
- **UUI**: User to User Information
- **VC**: Virtual Channel

2.2 Assumptions

The AAL2 CPS Transmit 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 CPS SDUs received from the previous LFBs to create AAL2 CPS packets and to interleave these packets into ATM SDUs to be transferred on the AAL2 path. 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 Transmit Description

The AAL2 CPS Transmit LFB receives CPS SDUs to be transferred on AAL2 channels from the previous LFB over the `CPS_SDU_IN` input. The AAL2 CPS packets created by the AAL2 CPS Transmit LFB are then placed into ATM SDUs and passed to the next LFB (e.g., ATM Traffic Manager LFB) in the chain over the `ATM_SDU_OUT` output.

The AAL2 CPS Transmit LFB is modeled as shown in Figure 3.1

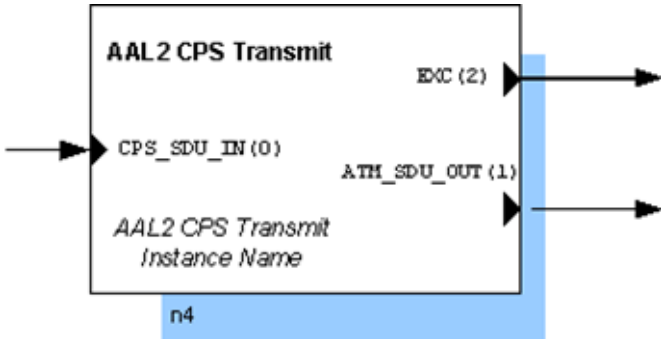

Figure 3.1: AAL2 CPS Transmit LFB

The LFB may contain multiple instances of AAL2 Paths that are identified by VC link IDs. 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 Transmit LFB and may be queried as such.

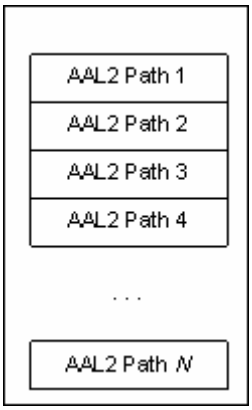

Figure 3.2: AAL2 Path Instances

Each AAL2 Path configured in the AAL2 CPS Transmit LFB may be associated with upto 255 AAL2 channels that are identified by a unique AAL2 Channel ID. Incoming packets are assigned to appropriate AAL2 channel instances according to the AAL2 Channel ID contained in the metadata received with the input packet. 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 Transmit LFB and may be queried as such.

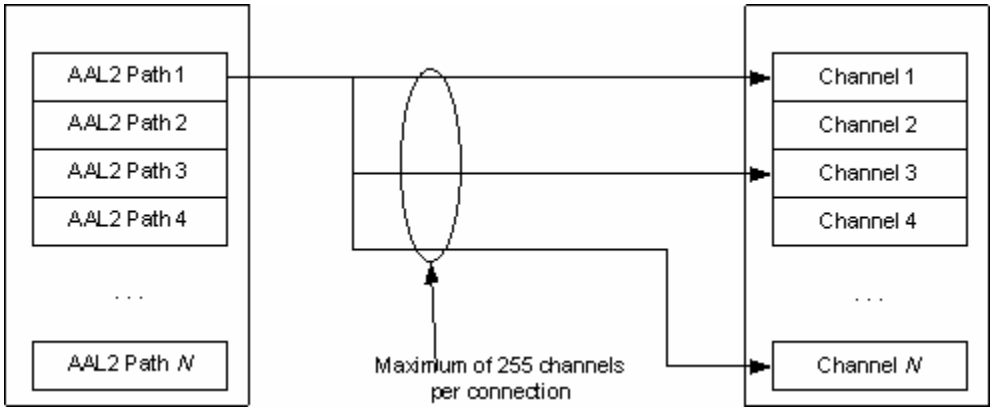

Figure 3.3: AAL2 channel instances and associated AAL2 path instances

The AAL2 channels created on an AAL2 path may be offered differential service with respect to the bandwidth usage and service delay. One or more AAL2 channels may be associated with a priority level. The bandwidth of the configured AAL2 path is shared between these priority levels proportionate to a weight configured for the priority level. The AAL2 CPS Transmit LFB enforces a threshold for the number of packets which may be queued in each priority queue for an AAL2 path.

The AAL2 CPS Transmit LFB maintains the following counters for each priority queue configured on an AAL2 Path:

- Number of CPS packets transmitted
- Number of bytes transmitted
- Number of CPS packets discarded due to various errors
- Number of bytes in the CPS packets discarded due to various errors

The AAL2 CPS Transmit LFB maintains the following counters for each AAL2 Channel:

- Number of CPS packets transmitted
- Number of bytes transmitted

3.1 AAL2 CPS Transmit Inputs

Table 3.1: AAL2 CPS Transmit LFB Inputs

Symbolic Name	Input ID	Description
`CPS_SDU_IN`	0	This is the only input for the AAL2 CPS Transmit LFB and is used to receive the CPS SDU for transmission on the AAL2 channel specified in the metadata.

3.1.1 Metadata Required

Table 3.2: Input Metadata for AAL2 CPS Transmit LFB

Metadata tag	Access method	Description
`META_CHNL_ID`	Read-and-consume	Metadata identifying the AAL2 channel on which the CPS SDU is to be transmitted.

META_UUI	Read-and-consume	The UUI value to be signaled in the CPS packet header created for this CPS SDU.
META_LI	Read-and-consume	The length of the CPS SDU to be transmitted.

3.2 AAL2 CPS Transmit Outputs

Table 3.3: AAL2 CPS Transmit LFB Outputs

Symbolic Name	Output ID	Description
ATM_SDU_OUT	1	This is the normal output for the AAL2 CPS Transmit LFB. The ATM SDUs created from CPS packets is sent on this output to the next LFB in the chain.
EXC	2	The packet requested for transmission is sent to this output when the packet needs to be discarded if the processing failed due to errors.

3.2.1.1 Metadata Produced

Table 3.4: Output Metadata for AAL2 CPS Transmit LFB

Metadata tag	Access method	Description
META_VCL_ID	Write	Metadata identifying the VC link ID of the AAL2 Path on which the ATM SDU is to be transmitted.
META_ATM_PTI	Write	The payload type indicating the payload type of the ATM SDU.

3.3 Accepted Inputs

The AAL2 CPS Transmit LFB can accept CPS SDUs for transmission over UNI or NNI.

3.4 CPS SDU Modifications

The CPS SDU is used to create a CPS packet. The CPS packets are then placed in ATM SDUs. One or more CPS packet along with the STF may be placed in an ATM SDU based on the interleaving option selected in the AAL2 Path profile. The received CPS SDU is then discarded and the ATM SDUs formed are then sent to the next LFB in the chain.

3.5 Relationship with Other LFBs

The AAL2 CPS Transmit LFB is placed in the processing chain before the ATM Traffic Manager LFB. The AAL2 CPS Transmit LFB receives primarily CPS SDUs from the previous LFB for transmission over AAL2 channels and performs CPS sublayer processing to create ATM SDU for transmission in ATM cells on the specified AAL2 Path. The sequence of actions that configures AAL2 CPS Transmit LFB and cooperating ATM Traffic Manager LFB instance, and cooperation between these two LFBs is schematically depicted in Figure 3.4.

The AAL2 CPS Transmit LFB may be preceded in the topology by any LFB that can produce the information required by the AAL2 CPS Transmit LFB at its input. Downstream (not necessarily next) of

the AAL2 CPS Transmit LFB, there should be LFBs that can utilize the information generated at output by AAL2 CPS Transmit LFB. The exact design and AAL2 Paths between the AAL2 CPS Transmit LFB and cooperating blocks is specific to the vendor that provides Forwarding Element design and FAPI implementation.

The `EXC` output of the AAL2 CPS Transmit LFB could be connected to an LFB that receives SDU's which could not be associated with any AAL2 Path or AAL2 channel or which need to be discarded due to various reasons.

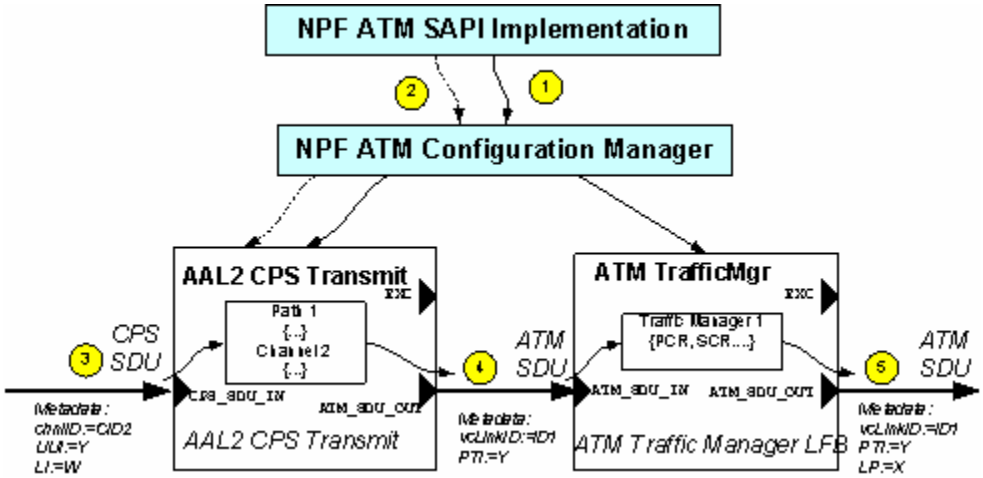

Figure 3.4: Cooperation between AAL2 CPS Transmit and ATM Traffic Manager

This figure shows part of example Forwarding Element that contains AAL2 CPS Transmit LFB and ATM Traffic Manager LFBs. These two blocks are connected in chain and configured by a NPF SAPI implementation. The sequence of actions that configure an AAL2 Path and channel 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 for the AAL2 path. The system software under the NPF ATM SAPI assigns a VC link ID 'ID1' to it and invokes the ATM Configuration manager FAPI to create the VC link. This causes an AAL2 Path instance to be created in the AAL2 CPS Transmit LFB. An ATM Traffic Manager instance is created in the ATM Traffic Manager LFB to process the traffic sent on the AAL2 Path with VC link ID 'ID1'.
2. The NPF ATM SAPI is invoked to create an AAL2 channel. The system software under the NPF ATM SAPI assigns a channel ID 'CID2' to the channel and invokes the ATM configuration manager FAPI to create the channel. This causes a channel instance to be created in the AAL2 CPS Transmit LFB and associated with the VC link 'ID1' associated with the AAL2 Path.
3. The AAL2 CPS Transmit LFB receives a CPS SDU from the CPS service user LFB. The channel ID specified in the input metadata is used to identify the priority level and the priority queue in which the received CPS SDU is to be placed. The AAL2 CPS Transmit LFB then determines the priority queue to service based on the priority level and weight associated with the priority queues and extracts a CPS SDU from that priority queue for transmission. The AAL2 CPS Transmit LFB creates CPS packets from the CPS SDU and interleaves them into ATM SDUs for transmission over the AAL2 path. The ATM SDU is sent to the next LFB in the chain over the `ATM_SDU_OUT` output.
4. The ATM SDU is forwarded to the ATM Traffic Manager LFB over the `ATM_SDU_OUT` output of the AAL2 CPS Transmit LFB. The ATM Traffic Manager LFB uses the VC link ID to determine the Traffic Manager instance associated with this AAL2 Path and performs the required traffic management actions like buffering, GCRA shaping, etc.
5. When the ATM Traffic Manager determines it is time to schedule transmission of an ATM cell on that AAL2 Path, the ATM SDU is forwarded to the next LFB in the chain for further processing.

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 Transmit LFB in a forwarding element using a block type value for the AAL2 CPS Transmit LFB.

```
#define NPF_F_AAL2CPSTX_LFB_TYPE     38
```

4.1.2 AAL2 Path Characteristics

The AAL2 CPS Transmit 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.
- Interleave control - Whether to interleave CPS packets in CPS PDU's. If this option selected as FALSE, TIMER CU is not applicable and the AAL2 payload cannot be greater than 44 bytes.
- Duration of the combined use timer (TIMER CU) for this AAL2 path. This configuration is used only if interleaving of CPS packets in CPS PDU's is enabled.
- The maximum number of priority queues configured for this AAL2 path.
- The weight associated with each configured priority queue. The weight is used to proportionately share the AAL2 path bandwidth among the configured priority queues.
- The discard threshold for each configured priority queue in terms of maximum number of CPS SDUs which may be queued in that queue waiting for a transmit opportunity. If the queue length exceeds this threshold then new CPS SDUs received for this priority queue are discarded.

4.1.3 AAL2 Channel Characteristics

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

- AAL2 channel identifier (CID) of the channel.
- The VC link ID of the associated AAL2 path.
- The maximum size CPS-SDU, in octets, which 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 priority level associated with this AAL2 channel. This should be one of the priority levels configured on the corresponding AAL2 path.

4.2 Data Structures for Completion Callbacks

4.2.1 AAL2 CPS Transmit LFB Attributes query response

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

```
typedef struct {
   NPF_uint32_t   maxPaths;             /* Maximum possible AAL2 Path's   */
   NPF_uint32_t   maxChnls;             /* Maximum possible AAL2 channels */
   NPF_uint32_t   curNumPaths;          /* Current number of AAL2 Path's  */
   NPF_uint32_t   curNumChnls;          /* Current number of AAL2 channels*/
   NPF_uint32_t   maxPathPrio;          /* Maximum priority levels/path   */
}  NPF_F_AAL2CpsTxAttrQueryResponse_t;
```

The `maxPaths` field contains the maximum number of AAL2 Paths supported in this AAL2 CPS Transmit LFB. The `curNumPaths` field contains the number of AAL2 Paths currently established in the AAL2 CPS Transmit 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. The `maxPathPrio` field contains the maximum number of priority levels which may be configured on any AAL2 path 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 interface handle,
* an error or success code, and in some cases a function-
* specific structure embedded in a union.
*/
typedef struct { /* Asynchronous Response Structure */
    NPF_F_AAL2CpsTxErrorType_t error;  /* Error code for this response */
    union {
        /* NPF_F_AAL2CpsTxLFB_AttributesQuery() */
        NPF_F_AAL2CpsTxAttrQueryResponse_t    lfbAttrQueryResponse;
    } u;
} NPF_F_AAL2CpsTxAsyncResp_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_AAL2CpsTxCallbackData_t.
*/
typedef enum NPF_F_AAL2CpsTxCallbackType {
    NPF_F_AAL2CPSTX_ATTR_QUERY = 1,
} NPF_F_AAL2CpsTxCallbackType_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_AAL2CpsTxCallbackData_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_AAL2CpsTxAsyncResponse_t structure, along with any other
* information
*/
typedef struct {
    NPF_F_AAL2CpsTxCallbackType_t type;  /* Which function called?      */
    NPF_IN NPF_BlockId_t          blockId;/* ID of LFB generating callback */
    NPF_F_AAL2CpsTxAsyncResp_t   resp;   /* Response struct  */
} NPF_F_AAL2CpsTxCallbackData_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_AAL2CPSTX_ATTR_QUERY`	`NPF_F_AAL2CpsTxAttrQueryResponse_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 Transmit 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 Transmit 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) */
#define NPF_AAL2CPSTX_BASE_ERR (NPF_F_AAL2CPSTX_LFB_TYPE * 100)

typedef NPF_uint32_t NPF_F_AAL2CpsTxErrorType_t;

#define AAL2CPSTX_ERR(n) ((NPF_F_AAL2CpsTxErrorType_t) \
                          (NPF_AAL2CPSTX_BASE_ERR+ (n))

#define NPF_E_AAL2CPSTX_INVALID_AAL2CPSTX_BLOCK_ID    AAL2CPSTX_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_AAL2CpsTxCallbackFunc_t) (
      NPF_IN NPF_userContext_t            userContext,
      NPF_IN NPF_correlator_t             correlator,
      NPF_IN NPF_F_AAL2CpsTxCallbackData_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 Transmit API implementation. This callback function is intended to be implemented by the application, and be registered to the AAL2 CPS Transmit API implementation through the `NPF_F_AAL2CpsTxRegister` 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 Transmit 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_AAL2CpsTxRegister(
      NPF_IN NPF_userContext_t                  userContext,
      NPF_IN NPF_F_AAL2CpsTxCallbackFunc_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 Transmit 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 Transmit 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 Transmit 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_AAL2CpsTxDeregister(
      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 Transmit 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_AAL2CpsTxLFB_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 Transmit LFB's attributes at a time. If the AAL2 CPS Transmit 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 Transmit 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 Transmit 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_AAL2CPSTX_INVALID_AAL2CPSTX_BLOCK_ID` – LFB ID is not an ID of LFB that has AAL2 CPS Transmit functionality

The `lfbAttrQueryResponse` field of the union in the `NPF_F_AAL2CpsTxAsyncResponse_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 Transmit Functional API
 */

#ifndef __NPF_F_AAL2_CPSTRANSMIT_H__
#define __NPF_F_AAL2_CPSTRANSMIT_H__

#ifdef __cplusplus
extern "C" {
#endif

/* It is possible to use the FAPI Topology Discovery
   APIs to discover an AAL2 CPS Transmit LFB
   in a forwarding element.  */
#define NPF_F_AAL2CPSTX_LFB_TYPE    38

/* Asynchronous error codes (returned in function callbacks) */
#define NPF_AAL2CPSTX_BASE_ERR (NPF_F_AAL2CPSTX_LFB_TYPE * 100)

typedef NPF_uint32_t NPF_F_AAL2CpsTxErrorType_t;

#define AAL2CPSTX_ERR(n) ((NPF_F_AAL2CpsTxErrorType_t) \
                            (NPF_AAL2CPSTX_BASE_ERR+ (n))

#define NPF_E_AAL2CPSTX_INVALID_AAL2CPSTX_BLOCK_ID    AAL2CPSTX_ERR(0)

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

/* The attributes of an AAL2 CPS Transmit */
typedef struct {
   NPF_uint32_t   maxPaths;             /* Maximum possible AAL2 Path's   */
   NPF_uint32_t   maxChnls;             /* Maximum possible AAL2 channels */
   NPF_uint32_t   curNumPaths;          /* Current number of AAL2 Path's  */
   NPF_uint32_t   curNumChnls;          /* Current number of AAL2 channels*/
}  NPF_F_AAL2CpsTxAttrQueryResponse_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_AAL2CpsTxErrorType_t error;  /* Error code for this response */
    union {
        /* NPF_F_AAL2CpsTxLFB_AttributesQuery() */
        NPF_F_AAL2CpsTxAttrQueryResponse_t    lfbAttrQueryResponse;
    } u;
} NPF_F_AAL2CpsTxAsyncResp_t;


/*
* Completion Callback Types, to be found in the callback
* data structure, NPF_F_AAL2CpsRx_CallbackData_t.
```

```
*/
typedef enum NPF_F_AAL2CpsTxCallbackType {
    NPF_F_AAL2CPSTX_ATTR_QUERY = 1,
} NPF_F_AAL2CpsTxCallbackType_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_AAL2CpsTxAsyncResponse_t structure, along with any other
* information
*/
typedef struct {
    NPF_F_AAL2CpsTxCallbackType_t type;  /* Which function called?      */
    NPF_IN NPF_BlockId_t          blockId;/* ID of LFB generating callback */
    NPF_F_AAL2CpsTxAsyncResp_t   resp;   /* Response struct  */
} NPF_F_AAL2CpsTxCallbackData_t;


/* Type for a callback function to be registered with LFB */
typedef void (*NPF_F_AAL2CpsTxCallbackFunc_t) (
      NPF_IN NPF_userContext_t userContext,
      NPF_IN NPF_correlator_t correlator,
      NPF_IN NPF_F_AAL2CpsTxCallbackData_t data);

/* Completion Callback Registration Function */
NPF_error_t NPF_F_AAL2CpsTxRegister(
      NPF_IN NPF_userContext_t userContext,
      NPF_IN NPF_F_AAL2CpsTxCallbackFunc_t callbackFunc,
      NPF_OUT NPF_callbackHandle_t *callbackHandle);

/* Completion Callback Deregistration Function */
NPF_error_t NPF_F_AAL2CpsTxDeregister(
      NPF_IN NPF_callbackHandle_t callbackHandle);


/* LFB Attributes Query Function */
NPF_error_t NPF_F_AAL2CpsTxLFB_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_CPSTRANSMIT_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