



AAL2 SAR SSCS Transmit LFB and Functional API Implementation Agreement

August 16, 2005
Revision 1.0

Editor:

Vedvyas Shanbhogue, Intel, vedvyas.shanbhogue@intel.com

Copyright © 2005 The Network Processing Forum (NPF). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction other than the following, (1) the above copyright notice and this paragraph must be included on all such copies and derivative works, and (2) this document itself may not be modified in any way, such as by removing the copyright notice or references to the NPF, except as needed for the purpose of developing NPF Implementation Agreements.

By downloading, copying, or using this document in any manner, the user consents to the terms and conditions of this notice. Unless the terms and conditions of this notice are breached by the user, the limited permissions granted above are perpetual and will not be revoked by the NPF or its successors or assigns.

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN "AS IS" BASIS WITHOUT ANY WARRANTY OF ANY KIND. THE INFORMATION, CONCLUSIONS AND OPINIONS CONTAINED IN THE DOCUMENT ARE THOSE OF THE AUTHORS, AND NOT THOSE OF NPF. THE NPF DOES NOT WARRANT THE INFORMATION IN THIS DOCUMENT IS ACCURATE OR CORRECT. THE NPF DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED THE IMPLIED LIMITED WARRANTIES OF MERCHANTABILITY, TITLE OR FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS.

The words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in the remainder of this document are to be interpreted as described in the NPF Software API Conventions Implementation Agreement revision 1.0.

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 SAR SSCS Transmit Description	6
	3.1 AAL2 SAR SSCS Transmit Inputs.....	6
	3.2 AAL2 SAR SSCS Transmit Outputs	7
	3.3 Accepted Inputs	7
	3.4 Packet Modifications	7
	3.5 Relationship with Other LFBs	8
4	Data Types	10
	4.1 Common LFB Data Types	10
	4.2 Data Structures for Completion Callbacks	10
	4.3 Data Structures for Event Notifications.....	11
	4.4 Error Codes	11
5	Functional API (FAPI).....	13
	5.1 Required Functions	13
	5.2 Conditional Functions.....	13
	5.3 Optional Functions.....	15
6	References.....	16
	Appendix A Header File Information.....	17
	Appendix B Acknowledgements.....	19
	Appendix C List of companies belonging to NPF during approval process.....	20

Table of Figures

Figure 3.1:	AAL2 SAR SSCS Transmit LFB	6
Figure 3.2:	AAL2 Channel Instances.....	6
Figure 3.3:	Cooperation between AAL2 SAR SSCS Transmit and AAL2 CPS Transmit LFB	8

List of Tables

Table 3.1:	AAL2 SAR SSCS Transmit LFB Inputs.....	6
Table 3.2:	Input Metadata for AAL2 SAR SSCS Transmit LFB	7
Table 3.3:	AAL2 SAR SSCS Transmit LFB Outputs	7
Table 3.4:	Output Metadata for AAL2 SAR SSCS Transmit LFB	7
Table 4.1:	Callback type to callback data mapping table.....	11

1 Revision History

Revision	Date	Reason for Changes
1.0	08/16/2005	Rev 1.0 of the AAL2 SAR SSCS Transmit LFB and Functional API Implementation Agreement. Source: npf2004.160.07.

2 Introduction

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

2.1 Acronyms

- **ATM:** Asynchronous Transfer Mode
- **AAL2:** ATM Adaptation Layer Type 2
- **API:** Application Programming Interface
- **CID:** Channel ID
- **CPS:** Common Part Sublayer
- **FAPI:** Functional API
- **IA:** Implementation Agreement
- **ID:** Identifier
- **LFB:** Logical Functional Block
- **NNI:** Network Node Interface
- **SAR:** Segmentation and Reassembly
- **SDU:** Service Data Unit
- **SSADT:** Service Specific Assured Data Transfer Sublayer
- **SSCS:** Service specific convergence sublayer
- **SSSAR:** Service Specific Segmentation and Reassembly Sublayer
- **SSTED:** Service Specific Transmission Error Detection Sublayer
- **UNI:** User Network Interface
- **UUI:** User to User Information
- **VC:** Virtual Channel

2.2 Assumptions

The AAL2 SAR SSCS 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 packets submitted for transmission over AAL2 channels. 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 AAL2 SAR SSCS Transmit Description

The AAL2 SAR SSCS Transmit LFB receives packets to be transferred on AAL2 channels from the previous LFB over the `AAL_SDU_IN` input. The AAL2 CPS SDUs created by the AAL2 SAR SSCS Transmit LFB are passed to the next LFB, AAL2 CPS Transmit LFB, over the `CPS_SDU_OUT` output. The AAL2 SAR SSCS Transmit LFB maintains the following counters for each configured AAL2 path and AAL2 channel:

- Number of CPS packets transmitted
- Number of bytes transmitted

The AAL2 SAR SSCS Transmit LFB is modeled as shown in Figure 3.1

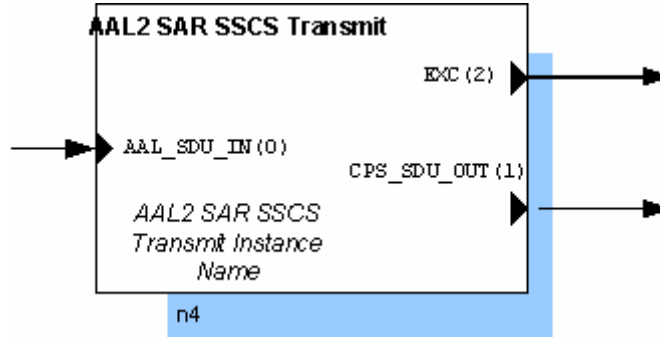


Figure 3.1: AAL2 SAR SSCS Transmit LFB

The LFB may contain multiple instances of AAL2 channels that are identified by unique AAL2 Channel IDs. Such instances are depicted in Figure 3.2. The maximum number of AAL2 channels is an attribute of the AAL2 SAR SSCS Transmit LFB and may be queried as such.

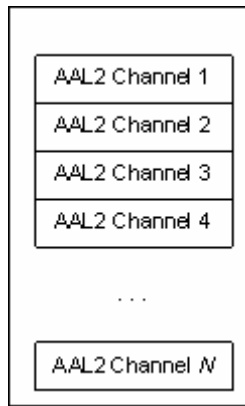


Figure 3.2: AAL2 Channel Instances

3.1 AAL2 SAR SSCS Transmit Inputs

Table 3.1: AAL2 SAR SSCS Transmit LFB Inputs

Symbolic Name	Input ID	Description
<code>AAL_SDU_IN</code>	0	This is the only input for the AAL2 SAR SSCS Transmit LFB and is used to receive packet payloads to be transmitted on the AAL2 channels.

3.1.1 Metadata Required

Table 3.2: Input Metadata for AAL2 SAR SSCS Transmit LFB

Metadata tag	Access method	Description
META_CHNL_ID	Read	Metadata identifying the AAL2 channel on which the packet is to be transmitted.
META_UUI	Read-and-consume	The UUI value to be signaled to the other end.
META_FRAME_LEN	Read-and-consume	The length of the packet to be transmitted. Does not include the CPS or any SSCS sublayer header/trailer.

3.2 AAL2 SAR SSCS Transmit Outputs

Table 3.3: AAL2 SAR SSCS Transmit LFB Outputs

Symbolic Name	Output ID	Description
CPS_SDU_OUT	1	This is the normal output for the AAL2 SAR SSCS Transmit LFB. The CPS SDUs created from input packets is sent on this output to the AAL2 CPS Transmit LFB.
EXC	2	The packet requested for transmission is sent to this output when the packet needs to be discarded due to errors.

3.2.1.1 Metadata Produced

Table 3.4: Output Metadata for AAL2 SAR SSCS Transmit LFB

Metadata tag	Access method	Description
META_UUI	Write	UUI to be used in the CPS packet header.
META_LI	Write	Length of the CPS SDU created after SAR SSCS processing.

3.3 Accepted Inputs

The AAL2 SAR SSCS Transmit LFB can accept packets for transmission over UNI or NNI.

3.4 Packet Modifications

If the AAL2 path on which the input packet is processed by the SSSAR SSCS, SSTED SSCS or the SSADT SSCS based on the configuration of the AAL2 channel on which the packet is to be transmitted. The CPS SDUs are created after segmentation. The received packet is then discarded and the CPS SDUs formed are then sent to the AAL2 CPS Transmit LFB in the chain to be sent as payload in ATM cells.

3.5 Relationship with Other LFBs

The AAL2 SAR SSCS Transmit LFB is placed in the processing chain before the AAL2 CPS Transmit LFB. The AAL2 SAR SSCS Transmit LFB receives primarily packets from the previous LFB for transmission over AAL2 channels and performs required SSCS sublayer processing to create CPS SDUs for transmission in ATM cells on the specified AAL2 path. The sequence of actions that configures AAL2 SAR SSCS Transmit LFB and cooperating AAL2 CPS Transmit LFB instance, and cooperation between these two LFBs is schematically depicted in Figure 3.3.

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

The EXC output of the AAL2 SAR SSCS Transmit LFB could be connected to an LFB that receives SDUs which could not be associated with any AAL2 channel or which need to be discarded due to various reasons.

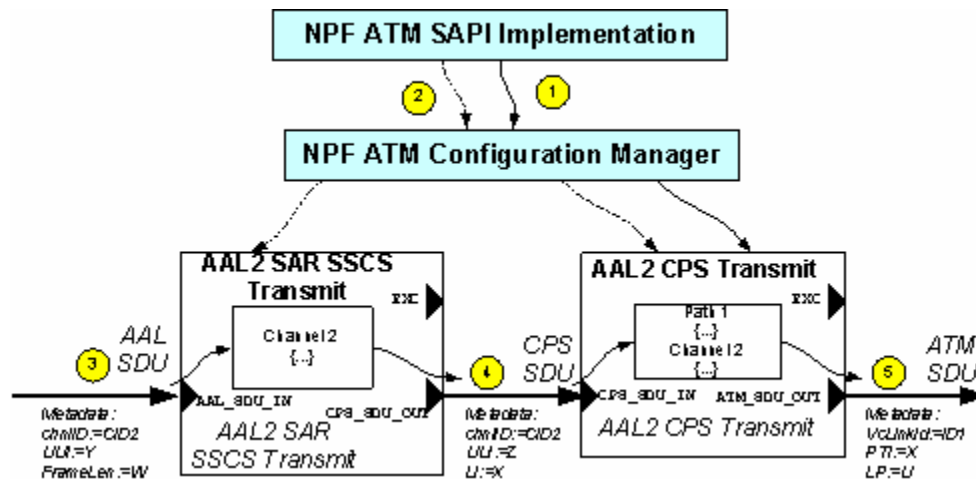


Figure 3.3: Cooperation between AAL2 SAR SSCS Transmit and AAL2 CPS Transmit LFB

This figure shows part of example Forwarding Element that contains AAL2 SAR SSCS Transmit LFB and AAL2 CPS Transmit 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 AAL2 channel may be defined as follows (see corresponding numbers in circles in the figure):

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.

1. 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 AAL2 SAR SSCS Transmit LFB.
2. The AAL2 SAR SSCS Transmit LFB receives a packet from an LFB using AAL2 services to send packets. The AAL2 SAR SSCS Transmit LFB creates CPS SDUs after required SSCS processing.
3. The CPS SDU is forwarded to the AAL2 CPS Transmit LFB along with the UUI and the length of the CPS SDU over the `CPS_SDU_OUT` output of the AAL2 SAR SSCS Transmit LFB.

4. The CPS SDU is used to create CPS packets that are interleaved in ATM SDUs. The ATM SDUs are sent over `ATM_SDU_OUT` output to the next LFB in the chain along with the PTI and the ID of the AAL2 path to the next LFB in the chain.

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

```
#define NPF_F_AAL2SARSSCSTX_LFB_TYPE 40
```

4.1.2 AAL2 Channel Characteristics

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

- AAL2 channel identifier (CID) of the channel
- The maximum SSSAR SDU length for this channel
- Sublayers enabled – SSTED, SSADT
- The maximum SSSAR segment length for this channel

4.2 Data Structures for Completion Callbacks

4.2.1 AAL2 SAR SSCS Transmit LFB Attributes query response

The attributes of an AAL2 SAR SSCS Transmit LFB are the following:

```
typedef struct {
    NPF_uint32_t    maxChnls;           /* Maximum possible AAL2 channels */
    NPF_uint32_t    curNumChnls;       /* Current number of AAL2 channels*/
} NPF_F_AAL2SarSscsTxAttrQueryResponse_t;
```

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_AAL2SarSscsTxErrorType_t error; /* Error code for this response*/
    union {
        /* NPF_F_AAL2SarSscsTxLFB_AttributesQuery() */
        NPF_F_AAL2SarSscsTxAttrQueryResponse_t    lfbAttrQueryResponse;
    } u;
} NPF_F_AAL2SarSscsTxAsyncResp_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_AAL2SarSscsTxCallbackData_t.
 */
typedef enum NPF_F_AAL2SarSscsTxCallbackType {
    NPF_F_AAL2SARSSCSTX_ATTR_QUERY = 1,
} NPF_F_AAL2SarSscsTxCallbackType_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_AAL2SarSscsTxCallbackData_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_AAL2SarSscsTxAsyncResp_t structure, along with any other
 * information
 */
typedef struct {
    NPF_F_AAL2SarSscsTxCallbackType_t type; /* Which function called? */
    NPF_IN NPF_BlockId_t blockId; /* ID of LFB generating callback */
    NPF_F_AAL2SarSscsTxAsyncResp_t resp; /* Response struct */
} NPF_F_AAL2SarSscsTxCallbackData_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
<code>NPF_F_AAL2SARSSCSTX_ATTR_QUERY</code>	<code>NPF_F_AAL2SarSscsTxAttrQueryResponse_t</code>

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 SAR SCS 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 SAR SSCS 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_AAL2SARSSCSTX_BASE_ERR (NPF_F_AAL2SARSSCSTX_LFB_TYPE * 100)
typedef NPF_uint32_t NPF_F_AAL2SarSscsTxErrorType_t;
#define AAL2SARSSCSTX_ERR(n) ((NPF_F_AAL2SarSscsTxErrorType_t) \
                             (NPF_AAL2SARSSCSTX_BASE_ERR+ (n)))

#define NPF_E_AAL2SARSSCSTX_INVALID_AAL2SARSSCSTX_BLOCK_ID \
        AAL2SARSSCSTX_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_AAL2SarSscsTxCallbackFunc_t) (
    NPF_IN NPF_userContext_t          userContext,
    NPF_IN NPF_correlator_t          correlator,
    NPF_IN NPF_F_AAL2SarSscsTxCallbackData_t data);
```

5.2.1.1 Description

This callback function is for the application to register an asynchronous response handling routine to the AAL2 SAR SSCS Transmit API implementation. This callback function is intended to be implemented by the application, and be registered to the AAL2 SAR SSCS Transmit API implementation through the `NPF_F_AAL2SarSscsTxRegister` 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 SAR SSCS 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_AAL2SarSscsTxRegister(
    NPF_IN NPF_userContext_t          userContext,
    NPF_IN NPF_F_AAL2SarSscsTxCallbackFunc_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 SAR SSCS 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 SAR SSCS 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 SAR SSCS 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_AAL2SarSscsTxDeregister(  
    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 SAR SSCS 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_AAL2SarSscsTxLFB_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 SAR SSCS Transmit LFB's attributes at a time. If the AAL2 SAR SSCS 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 SAR SSCS 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 SAR SSCS 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_AAL2SARSSCSTX_INVALID_AAL2SARSSCSTX_BLOCK_ID` - LFB ID is not an ID of LFB that has AAL2 SAR SSCS Transmit functionality

The `lfbAttrQueryResponse` field of the union in the

`NPF_F_AAL2SarSscsTxAsyncResponse_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_1A.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 SAR SSCS Transmit Functional API
 */

#ifndef __NPF_F_AAL2_SARSSCSTRANSMIT_H__
#define __NPF_F_AAL2_SARSSCSTRANSMIT_H__

#ifdef __cplusplus
extern "C" {
#endif

/* It is possible to use the FAPI Topology Discovery
   APIs to discover an AAL2 SAR SSCS Transmit LFB
   in a forwarding element. */
#define NPF_F_AAL2SARSSCSTX_LFB_TYPE 40

/* Asynchronous error codes (returned in function callbacks) */
#define NPF_AAL2SARSSCSTX_BASE_ERR (NPF_F_AAL2SARSSCSTX_LFB_TYPE * 100)
typedef NPF_uint32_t NPF_F_AAL2SarSscsTxErrorType_t;
#define AAL2SARSSCSTX_ERR(n) ((NPF_F_AAL2SarSscsTxErrorType_t) \
                             (NPF_AAL2SARSSCSTX_BASE_ERR+ (n)))

#define NPF_E_AAL2SARSSCSTX_INVALID_AAL2SARSSCSTX_BLOCK_ID \
        AAL2SARSSCSTX_ERR(0)
/*****
 * Enumerations and types for AAL2 SAR SSCS Transmit attributes and*
 * completion callback data types
 *****/

/* The attributes of an AAL2 SAR SSCS Transmit */
typedef struct {
    NPF_uint32_t    maxChnls;           /* Maximum possible AAL2 channels */
    NPF_uint32_t    curNumChnls;       /* Current number of AAL2 channels*/
} NPF_F_AAL2SarSscsTxAttrQueryResponse_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_AAL2SarSscsTxErrorType_t error; /* Error code for this response*/
    union {
        /* NPF_F_AAL2SarSscsTxLFB_AttributesQuery() */
        NPF_F_AAL2SarSscsTxAttrQueryResponse_t lfbAttrQueryResponse;
    } u;
} NPF_F_AAL2SarSscsTxAsyncResp_t;

/*
 * Completion Callback Types, to be found in the callback
 * data structure, NPF_F_AAL2SarSscsTxCallbackData_t.
 */
typedef enum NPF_F_AAL2SarSscsTxCallbackType {
    NPF_F_AAL2SARSSCSTX_ATTR_QUERY = 1,
} NPF_F_AAL2SarSscsTxCallbackType_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_AAL2SarSscsTxAsyncResp_t structure, along with any other
* information
*/
typedef struct {
    NPF_F_AAL2SarSscsTxCallbackType_t type; /* Which function called? */
    NPF_IN NPF_BlockId_t blockId; /* ID of LFB generating callback */
    NPF_F_AAL2SarSscsTxAsyncResp_t resp; /* Response struct */
} NPF_F_AAL2SarSscsTxCallbackData_t;

/* Type for a callback function to be registered with LFB */
typedef void (*NPF_F_AAL2SarSscsTxCallbackFunc_t) (
    NPF_IN NPF_userContext_t userContext,
    NPF_IN NPF_correlator_t correlator,
    NPF_IN NPF_F_AAL2SarSscsTxCallbackData_t data);

/* Completion Callback Registration Function */
NPF_error_t NPF_F_AAL2SarSscsTxRegister(
    NPF_IN NPF_userContext_t userContext,
    NPF_IN NPF_F_AAL2SarSscsTxCallbackFunc_t callbackFunc,
    NPF_OUT NPF_callbackHandle_t *callbackHandle);

/* Completion Callback Deregistration Function */
NPF_error_t NPF_F_AAL2SarSscsTxDeregister(
    NPF_IN NPF_callbackHandle_t callbackHandle);

/* LFB Attributes Query Function */
NPF_error_t NPF_F_AAL2SarSscsTxLFB_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_SARSSCSTRANSMIT_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 Damnvik, Ericsson
Patrik Herneld, Ericsson
Ajay Kamalvanshi, Nokia
Jaroslaw Kogut, Intel
Arthur Mackay, Freescale
Stephen Nadas, Ericsson
Michael Persson, Ericsson
John Renwick, Agere Systems
Vedvyas Shanbhogue (ed.), Intel
Keith Williamson, Motorola
Weislaw Wisniewski, Intel
Per Wollbrand, Ericsson

Appendix C List of companies belonging to NPF during approval process

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