LLD Interface PSI - ACI

Project	TCS 3.x
Document Type	Technical Documentation
Title	LLD Interface PSI - ACI
Author	Liyi Yu
Creation Date	09. 02. 2004
Last Modified	
ID and Version	To Be Assigned
Status	Being Processed

0 Document Control

Every effort has been made to ensure that the information contained in this document is accurate at the time of printing. However, the software described in this document is subject to continuous development and improvement. Texas Instruments reserves the right to change the specification of the software. Information in this document is subject to change without notice and does not represent a commitment on the part of Texas Instruments. Texas Instruments accepts no liability for any loss or damage arising from the use of any information contained in this document.

The software described in this document is furnished under a license agreement and may be used or copied only in accordance with the terms of the agreement. It is an offence to copy the software in any way except as specifically set out in the agreement. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Texas Instruments.

0.1 Document History

ID	Author	Date	Status
	Liyi Yu	09.02.2004	Being processed

0.2 References

0.3 Abbreviations, Terms

Table of Contents

Table of Figures

1 Introduction

G23 is a software package implementing Layers 2 and 3 of the ETSI-defined GSM air interface signalling protocol, and as such represents that part of a GSM mobile station's protocol software which is both, platform and manufacturer independent. Therefore, G23 can be viewed as a building block providing standardised functionality through generic interfaces for easy integration.

The G23 suite of products consists of the following items:

- Layers 2 and 3 for speech & short message services,
- Layers 2 and 3 for fax & data services,
- Application Control Interface,
- Slim MMI [02.30] and
- Test and integration support tools.

2 Overview

2.1 General

This document is a Low Level Design (LLD) document, which describes the implementation of the interface between entity PSI – Protocol Stack Entity and ACI.

Entity PSI is a new entity, which will replace the current entities UART, PKTIO and AAA, and includes a new driver USB.

The implementation here will handle the communication between PSI and ACI. It is assumed that the user already has the basic knowledge of PSI and DTI lib. Since the implementation is based on the existing module UART and PKTIO, the understanding of them might be helpful.

The communication between PSI and ACI includes primitive communications for data flow and for control flow and some configuration primitives. Flow controls (data flow and control flow) are managed by the DTI manager, which is located in ACI. Other than that ACI is able to handle all the configuration primitive and handle a very special type of data: AT commands. The handling of AT commands is done in module SAP_DTI.

According to the HLD (refer ???) the implementation of PSI is devided into 2 phases. At the end of the 1st phase, the PSI should be implemented with the USB functionality. Which means that the UART and PSI will exist at the same time. This document only describes the PSI implementation with USB functionality.

The following figure shows the overview of the involved modules for phase 1. The changed or new added modules for the implementation here are marked in yellow.

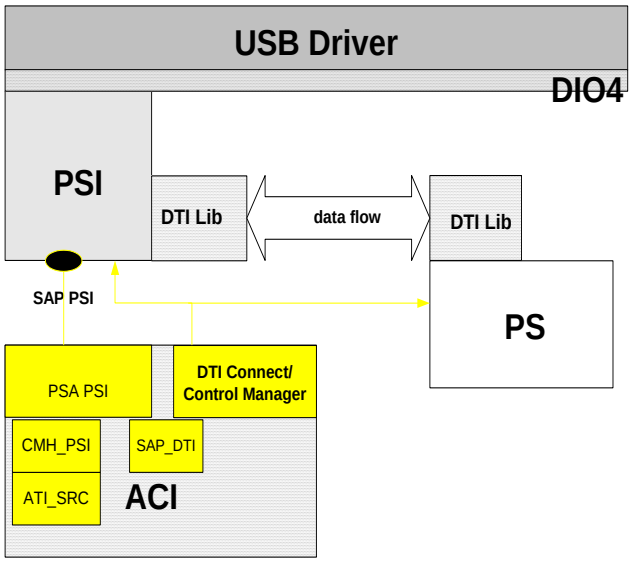


Figure 1 Module overview

From the above figure we can see that the implementation will take into account of modules PSA_PAI, CMH_PSI, ATI_SRC, DTI Manager and SAP_DTI.

2.2 Detailed Description

As shown in the above figure, the modules should be modified or implemented in ACI are PSA module, CMH module, ATI module and DTI manager module. The following paragraphs describe briefly the modules that will be implemented or changed. More details please refer to chapter 3 and chapter 4.

PSA_PSI:

This is a new module in ACI, which handles the incoming and outgoing primitives from and to PSI entity.

CMH_PSI:

This module implements the CMH related functions for PSA_PSI.

ATI_SRC_PSI:

This module is responsible for creating new ATI sources for PSI.

SAP_DTI:

In this module ACI acts as an entity that can receive data primitives from DTI. This is the handling for DTI callbacks and the data primitives (AT commands) sent by DTI.

3 Scenarios with MSCs

The roll of ACI in this case is a bit complicated. ACI can receive AT commands from PSI (from DTI channel), so ACI is a data receiver; Since PSI has the capability of sending AT commands. So ACI is also an ATI source manager; and last but not least, ACI includes the DTI control manager.

Since PSI is a new entity added to communicate with ACI, the above-mentioned aspects should all be taken into account in the implementation.

3.1 Connection Primitives Handling

3.1.1 PSI_CONN_IND: Creation of Connection

Description:

If the DIO driver wants to establish a connection with ACI, PSI will first send out a primitive PSI_CONN_IND to ACI. When ACI receives this primitive, it registers the device in the PSA module, registers the new device in the DTI control manager, creates ATI source for the driver and builds a DTI channel between PSI and ACI for AT commands if the data mode has the capability of sending AT commands. If everything has gone well ACI will reply with PSI_CONN_RES, otherwise PSI_CONN_REJ.

This primitive starts the first communication between PSI and ACI.

Parameters:

devId (U32):

Contains values of device numbers. The parameter is a combination of an driver identifier (bit 24-31), type of device (bit 8-23) and the actual number of devices (bit 0-7).

dio_cap:

The parameter contains all the driver capability parameter.

psi_data_mode:
This parameter describes the used data mode of the device.

Implementation:

When ACI receives the indication primitive, it will do the following:

1. A new ATI source will be created according to < psi_data_mode> include command capability;
2. Register the device in DTI manager and create a DTI connection using the functions provided by DTI control manager;
3. Send out a response primitive (see 3.1.2) or a reject primitive (see 3.1.3).

MSC (see below)

3.1.2 PSI_CONN_RES

Description:

This primitive is the positive respond to PSI_CONN_IND. It confirms the successful device registration in ACI.

Parameters:

devId (U32):

Contains values of device numbers. The parameter is a combination of an driver identifier (bit 24-31), type of device (bit 8-23) and the actual number of devices (bit 0-7).

Implementation:

Send out the primitive with the input parameter <devId>.

MSC (see below)

3.1.3 PSI_CONN_REJ

Description:

This primitive rejects the driver registration request. If something in 3.1.1 went wrong, ACI will send back this primitive instead of a positive one to the primitive PSI_CONN_IND.

Parameters:

devId (U32):

Contains values of device numbers. The parameter is a combination of an driver identifier (bit 24-31), type of device (bit 8-23) and the actual number of devices (bit 0-7).

Implementation:

Send out the primitive with the input parameter <devId>.

MSC (PSI_CONN_IND, PSI_CONN_RES and PSI_CONN_REJ):

Scenario: Creation of connection

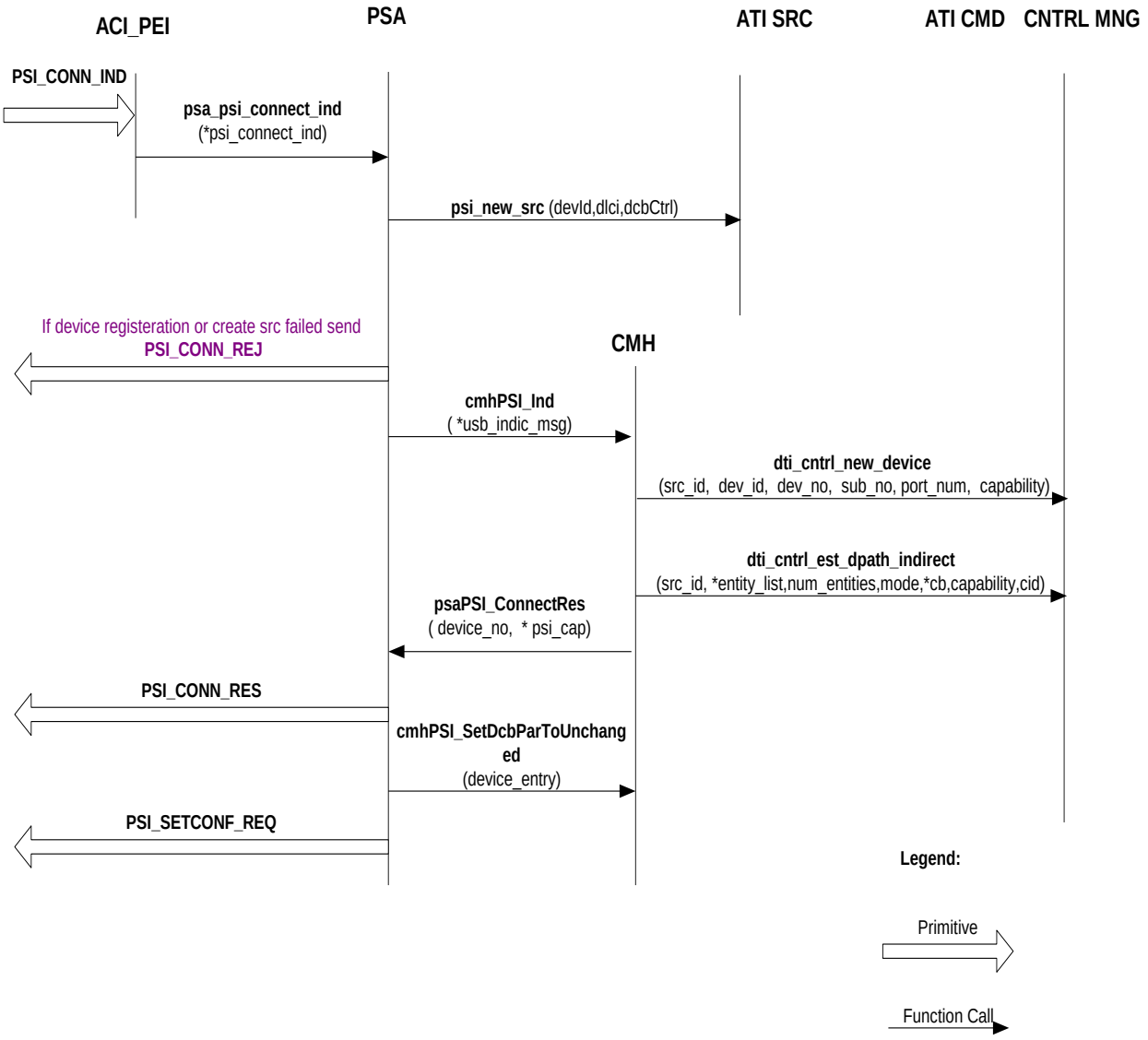


Figure 2 MSC for connection creation with PSI

3.1.4 PSI_DTI_CLOSE_IND

Description:

This primitive indicates ACI that the DTI connection was closed. ACI will inform the DTI manager that the DTI connection has been closed.

Parameters:

devId (U32):

Contains values of device numbers. The parameter is a combination of an driver identifier (bit 24-31), type of device (bit 8-23) and the actual number of devices (bit 0-7).

link_id:

The link identifier for DTI. This parameter is used to identify the affected channel. This is used in target environment.

Implementation:

ACI will look for the device number in the device list, if the device does not exist, ACI will ignore the primitive. If the device exists, ACI will inform DTI manager that the DTI connection between PSI and the peer entity has been closed. But the ATI source and entries in ACI will not be erased until the disconnect indication primitive is received.

MSC (see below)

3.1.5 PSI_DISCONN_IND: Erasing a Connection

Description:

If the DIO driver wants to close a connection with ACI, PSI will send out a primitive PSI_DTI_CLOSE_IND to ACI to close the DTI connection. And then send the primitive PSI_DISCONN_IND. The handling of primitive PSI_DTI_CLOSE_IND will be described in section 3.1.4 . The handling of PSI_DISCONN_IND is exactly the opposite of creating a connection for PSI, which means to erase everything created during the connect process.

Parameters:

DevId: device Id

cause: The parameter indicates the outcome of any operation. The cause values given here are error codes indicating internally recognized problems. This parameter is not used in ACI now, may be used in the future.

Implementation:

When the primitive PSI_DISCONN_IND is received, ACI will erase the ATI source, erase all the necessary registrations and entries in DTI and finally erase the device entry in PSA.

MSC (PSI_DTI_CLOSE_IND and PSI_DISCONN_IND):

Scenario: Disconnect Primitive Received

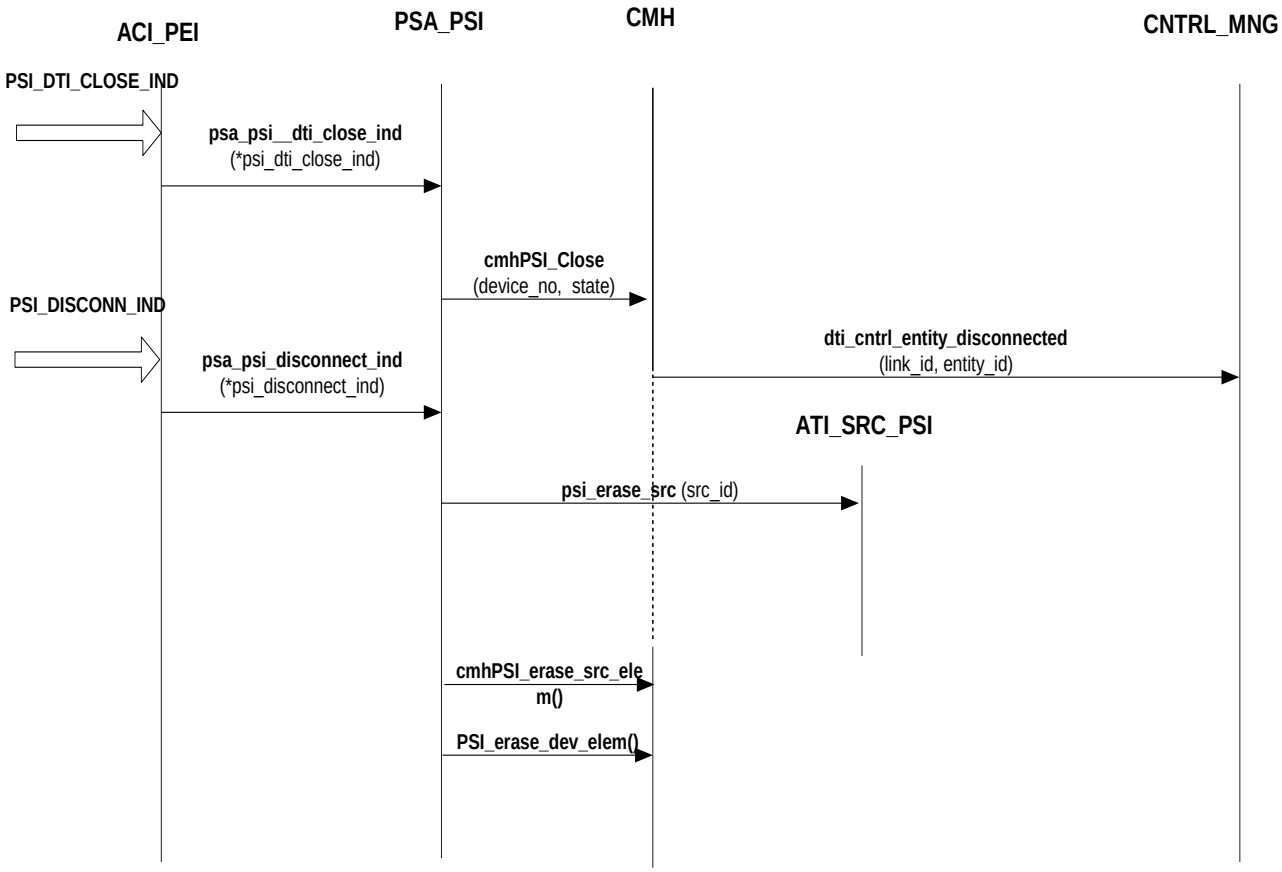


Figure 3 MSC for erasing of connection with PSI

3.2 Connection Line State Primitives Handling

3.2.1 PSI_LINE_STAT_IND

Description:

The driver can indicate detected escape sequences and DTR line drops. ACI receives the line states parameter in primitive PSI_LINE_STAT_IND. After ACI receives the primitive from PSI, it finds out the peer of PSI. Basing on the peer and the different settings of the PSI share parameter, different behaviours will be performed. The handling for PSI is very similar to UART.

Parameters:

DevId: device Id;

line_state: The parameter line_state contains information both about set/reset of serial line states and about the escape sequence detection.

Handling:

If escape sequence is detected, or if the line drop is detected and the PSI should turn back to command mode or the call should be cleared, PSI will connect back to ACI again with a command capability. In case line drop is detected but this should be ignored, then ACI will do nothing.

MSC:

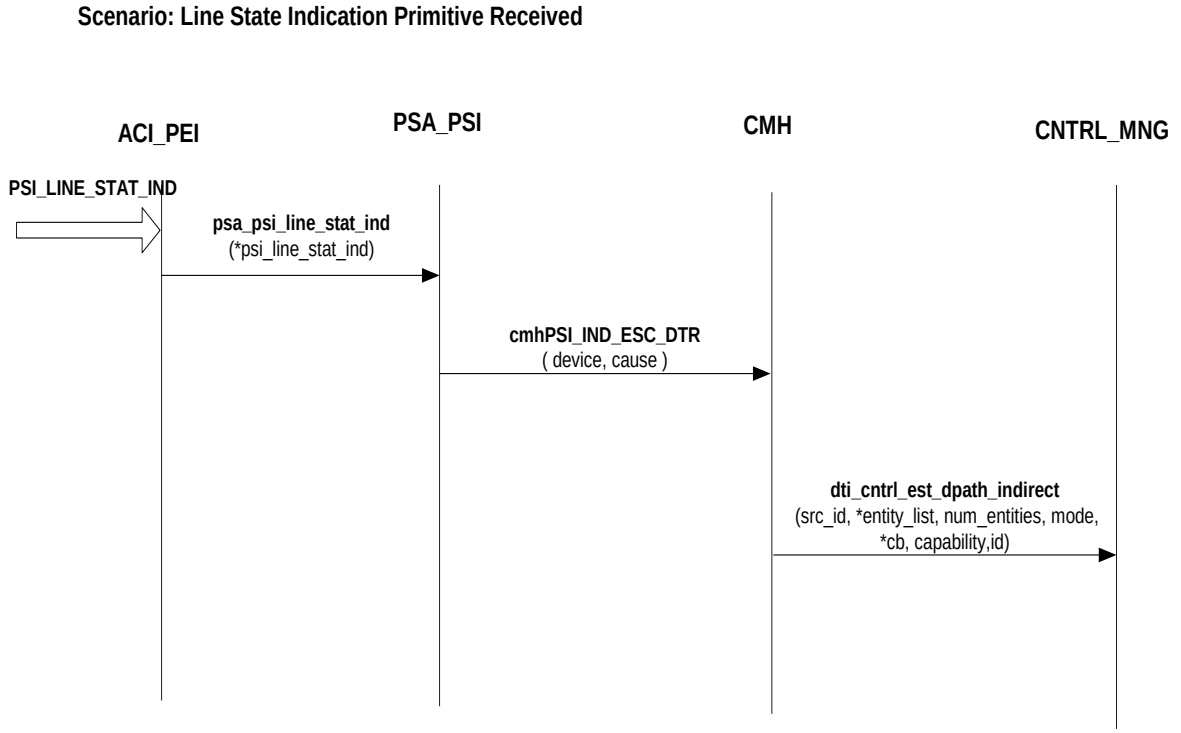


Figure 4 MSC for processing of line state indication primitive

3.2.2 PSI_SETCONF _REQ

Description:

The user can set driver configuration parameter anytime. The parameter dio_dcb is defined as union because there are three different configuration structures for serial. packet and multiplexer devices.

Parameters:

DevId: devId informs about the common driver characteristic. It contains the kind of device; the kind of transported data and the device number chosen by driver.

DIO_DCB: The parameter defines the structure for T_DIO_DCB containing all of driver configuration parameter. The union contains three different configuration structures referred to device_type (serial, packet, mux).

3.2.3 PSI_SETCONF_CNF

Description:

This primitive confirms ACI the changed driver configuration.

Parameters:

DevId: devId informs about the common driver characteristic. It contains the kind of device; the kind of transported data and the device number chosen by driver.

cause: The parameter indicates the outcome of any operation. The cause values given here are error codes indicating internally recognized problems.

3.3 DTI Primitives Handling

PSA sends out different control primitives to PSI if any of the entities wants to communicate with PSI or to change some of the parameters.

3.3.1 PSI_DTI_OPEN_REQ and PSI_DTI_OPEN_CON

3.3.1.1 PSI_DTI_OPEN_REQ

Description:

This primitive PSI_DTI_OPEN_REQ initiate the data flow starting the DTI connection between PSI and the peer entity. PSI opens the port to DTI.

Parameters:

DevId:

devId informs about the common driver characteristic. It contains the kind of device, the kind of transported data and the device number chosen by driver.

Peer:

Peer defines the peer name as a C-String.

Linkid:

The link identifier for DTI. This parameter is used to identify the affected channel.

Dti_direction:

This parameter controls if PSI uses DTI in the normal way or if the primitives are inverted. In the normal way data is sent as DTI_DATA_IND and received as DTI_DATA_REQ. In the inverted mode data is send as DTI_DATA_REQ and received as DTI_DATA_IND (relay functionality). The appropriate constants are are already defined in dti.h.

Handling:

PSA will just provide the interface for sending out the primitive to PSI. Since the sending of this primitive is by the DTI control manager functions dti_cntrl_maintain_entity_connect() and dti_cntrl_maintain_entity_disconnect, adding of a switch case for DTI_ENTITY_PSI inside these functions is

necessary. Additionally the new entity Id DTI_ENTITY_PSI should be inserted in the list of T_DTI_ENTITY_ID in dti_conn_mng.h.

MSC:

Scenario: Sending of PSI_DTI_OPEN_REQ

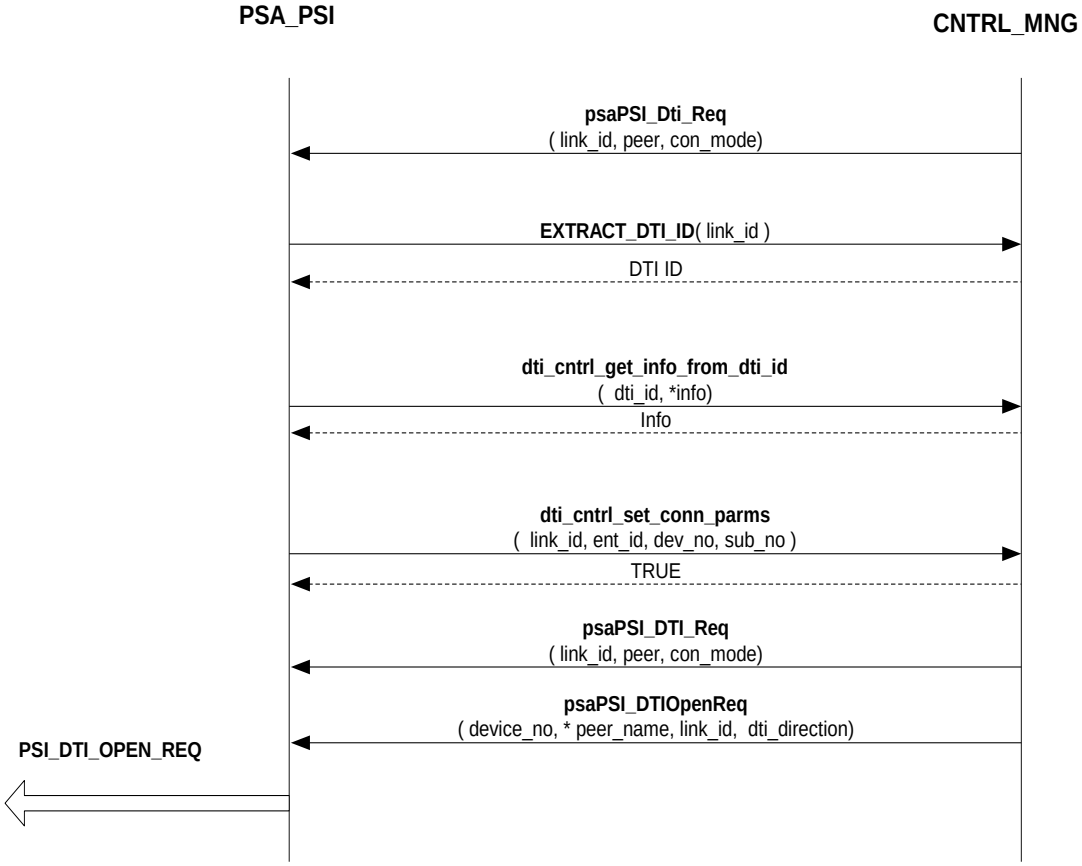


Figure 5 MSC for sending of PSI_DTI_OPEN_REQ

3.3.1.2 PSI_DTI_OPEN_CNF

Description:

This primitive confirms ACI the successful opened DTI connection.

Parameters:

DevId:

devId informs about the common driver characteristic. It contains the kind of device; the kind of transported data and the device number chosen by driver.

Cause:

The parameter indicates the outcome of any operation. The cause values given here are error codes indicating internally recognized problems.

Linkid:

The link identifier for DTI. This parameter is used to identify the affected channel.

Handling:

Inform the DTI control manager about the state change of the entity.

MSC:

Scenario: Receiving of PSI_DTI_OPEN_CNF

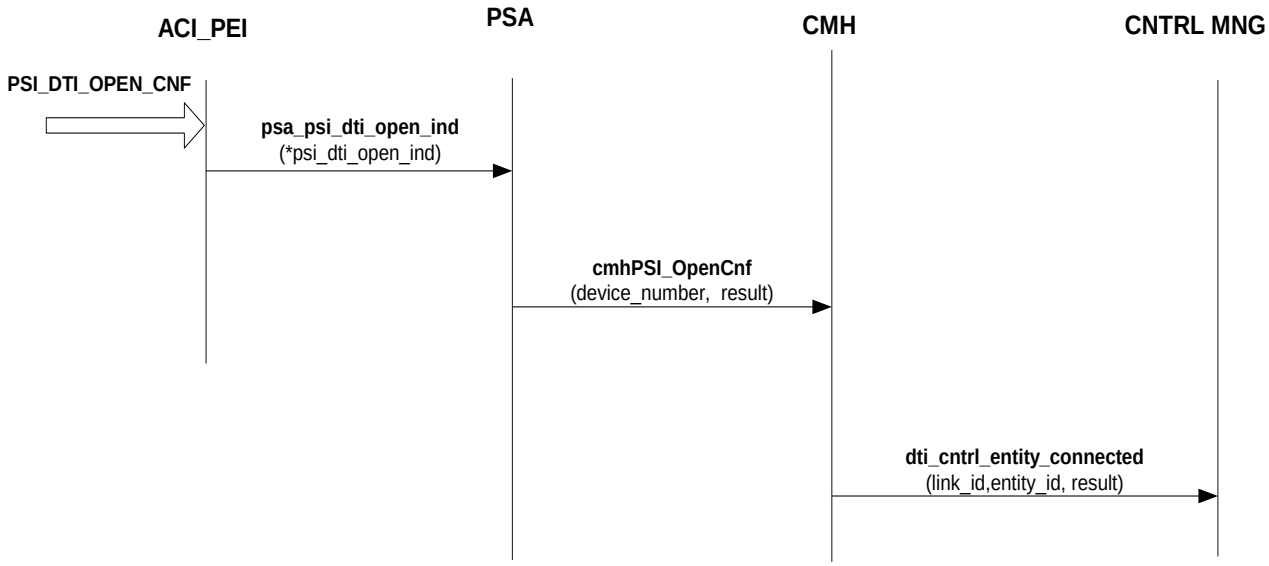


Figure 6 MSC for processing PSI_DTI_OPEN_CNF

3.3.2 PSI_DTI_CLOSE_REQ and PSI_DTI_CLOSE_CNF

3.3.2.1 PSI_DTI_CLOSE_REQ

Description:

If the user is not be able to use the device any longer ACI requests PSI via primitive PSI_DTI_CLOSE_REQ closing a existing DTI connection. PSI confirms the request with the primitive PSI_DTI_CLOSE_CNF. After that ACI requests closing of the device via the primitive PSI_CLOSE_REQ. In the last step PSI sends the confirmation primitive PSI_CLOSE_CNF to ACI.

Parameters:

DevId:

devId informs about the common driver characteristic. It contains the kind of device; the kind of transported data and the device number chosen by driver.

Linkid:

The link identifier for DTI. This parameter is used to identify the affected channel.

Handling:

PSA will just provide the interface to DTI control manager for sending out the primitive to PSI. Since the sending of this primitive is by the DTI control manager functions dti_cntrl_maintain_entity_connect() and dti_cntrl_maintain_entity_disconnect(), adding of a switch case for DTI_ENTITY_USB inside these functions is necessary. Additionally the new entity Id DTI_ENTITY_PSI should be inserted in the list of T_DTI_ENTITY_ID in dti_conn_mng.h.

MSC:

Scenario: Sending of PSI_DTI_CLOSE_REQ

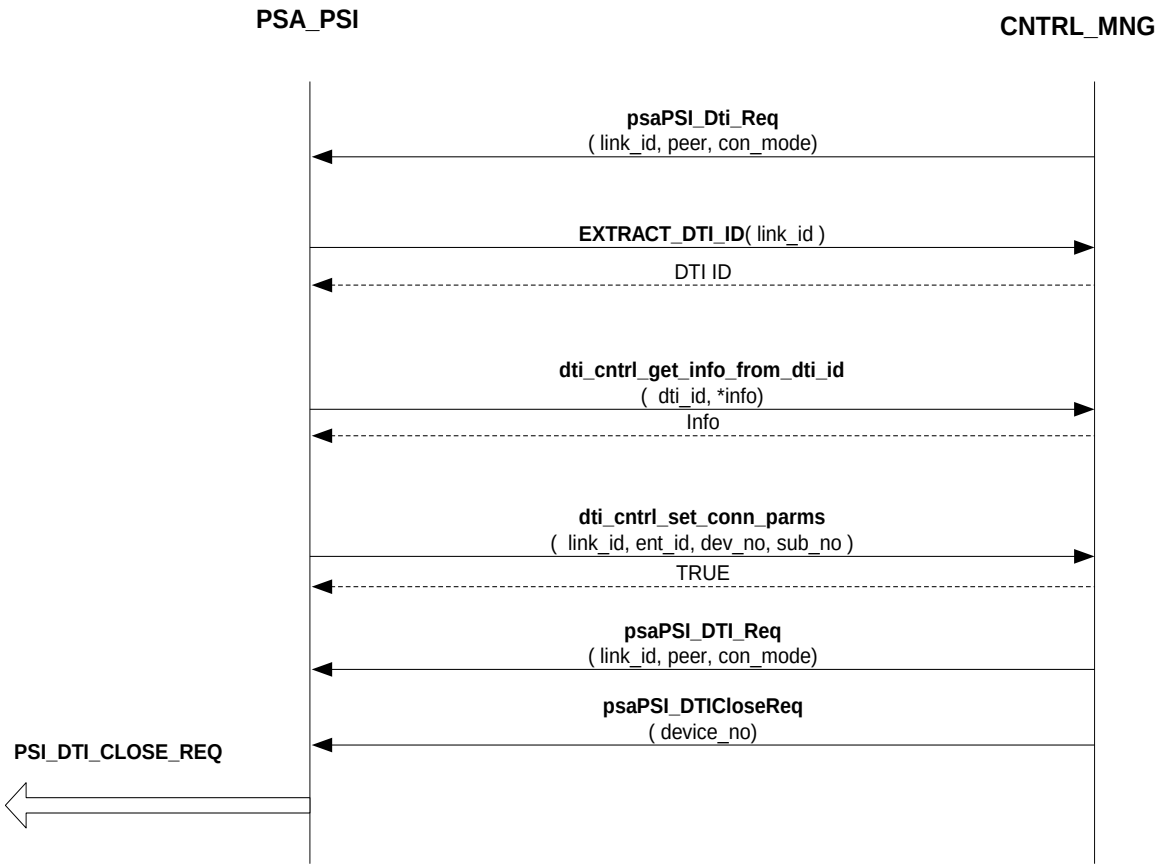


Figure 7 MSC for sending of PSI_DTI_CLOSE_REQ

3.3.2.2 PSI_DTI_CLOSE_CNF

Description:

This primitive confirms ACI the successful opened DTI connection.

Parameters:

DevId:

devId informs about the common driver characteristic. It contains the kind of device, the kind of transported data and the device number chosen by driver.

Linkid:

The link identifier for DTI. This parameter is used to identify the affected channel.

Handling:

Inform the DTI control manager about the state change of the entity.

Scenario: Receiving of PSI_DTI_CLOSE_CNF

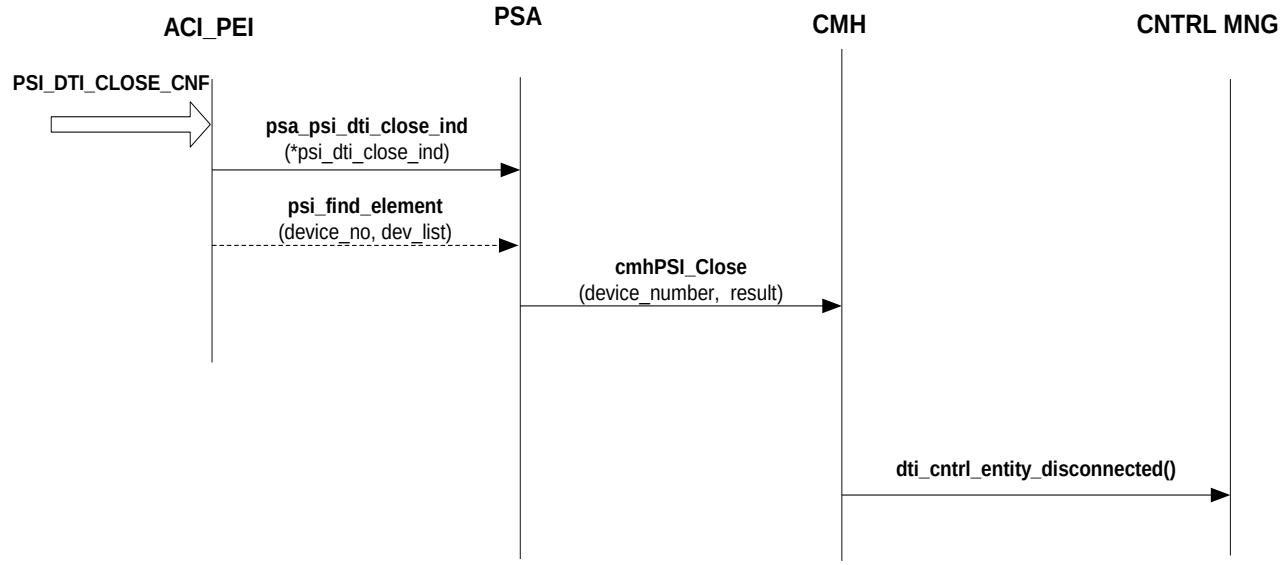


Figure 8 MSC for processing of PSI_DTI_CLOSE_CNF

3.3.3 PSI_CLOSE_REQ and PSI_CLOSE_CNF

3.3.3.1 PSI_CLOSE_REQ

Description:

This primitive informs PSI that the protocol stack is not be able the use the device any longer.

Parameters:

devId (U32):

Contains values of device numbers. The parameter is a combination of an driver identifier (bit 24-31), type of device (bit 8-23) and the actual number of devices (bit 0-7). The part " type of device" is the same as described for parameter device_type.

Implementation:

Send out the primitive with the devId after the primitive PSI_DTI_CLOSE_CNF is received.

3.3.3.2 PSI_CLOSE_CNF

Description:

This primitive confirms ACI that PSI has close the device.

Parameters:

devId (U32):

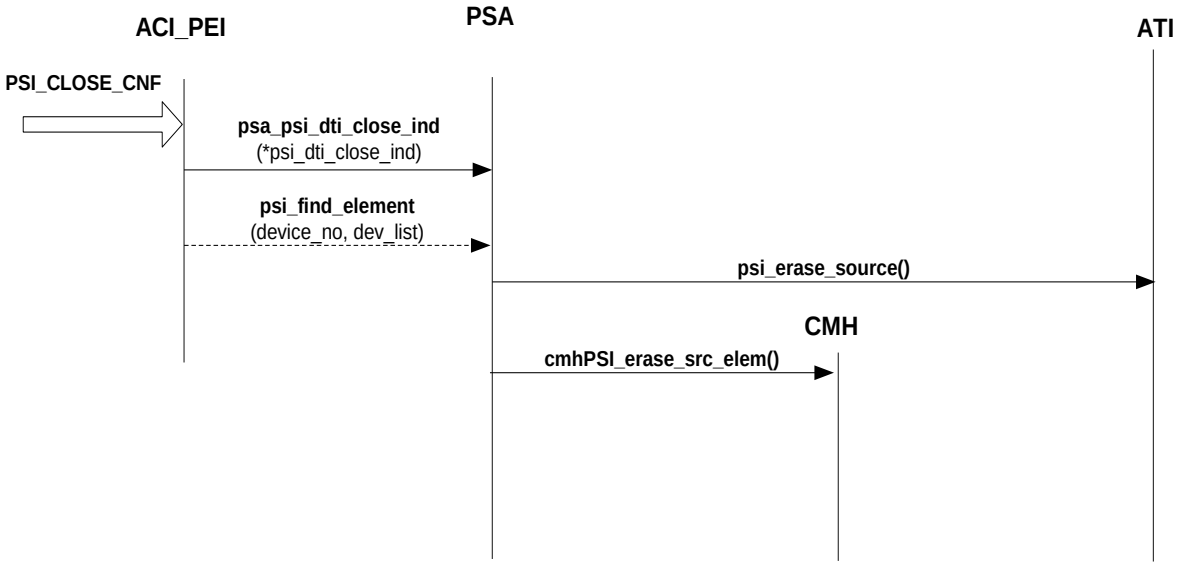
Contains values of device numbers. The parameter is a combination of an driver identifier (bit 24-31), type of device (bit 8-23) and the actual number of devices (bit 0-7). The part " type of device" is the same as described for parameter device_type.

Implementation:

See 3.1.5 .

MSC:

Scenario: Receiving of PSI_CLOSE_CNF



3.4 AT Commands Handling

This part of the code is stored in file sap_dti.c. This file handles the scenarios where ACI acts as a normal DTI data receiver, which handles DTI primitives and handles the DTI callbacks. This is an existing module in ACI. Since in earlier times only UART is able to send data to ACI, the existing handling is too specific to UART.

Now that we will add new device PSI that can also send AT commands to ACI, a small change in this module is necessary.

3.4.1 Handling of DTI Callbacks – AT Commands

Description:

After the communication channel is established, ACI as an entity connected with PSI can receive data via DTI connections. The data can only be AT commands. If some data are sent to ACI in the DTI channel, a callback function will be called to inform ACI that reasonable data are received and ACI should do something with the data. The possible parameters (instance, interface, channel, reason and *dti_data_ind) are included. But only the parameter <instance> and <*dti_data_ind > are considered here.

Parameters:

Instance: Source Id

Interface: peer entity. No need to consider this because the only peer entity for ACI is PSI.

Channel: DTI channel. No need to consider this.

Reason: why the callback function has been called, for this scenario the reason is reasonable data received.

*Dti_data_ind: Pointer to the received data.

Implementation:

If this situation is met, ACI checks the connection situation. If everything is fine, ACI will call the function psi_src_proc_char() to check if there are more data required. If the received data are already a complete AT command, the ATI processing functions e.g. ati_execute() will be called to process the AT command.

MSC:

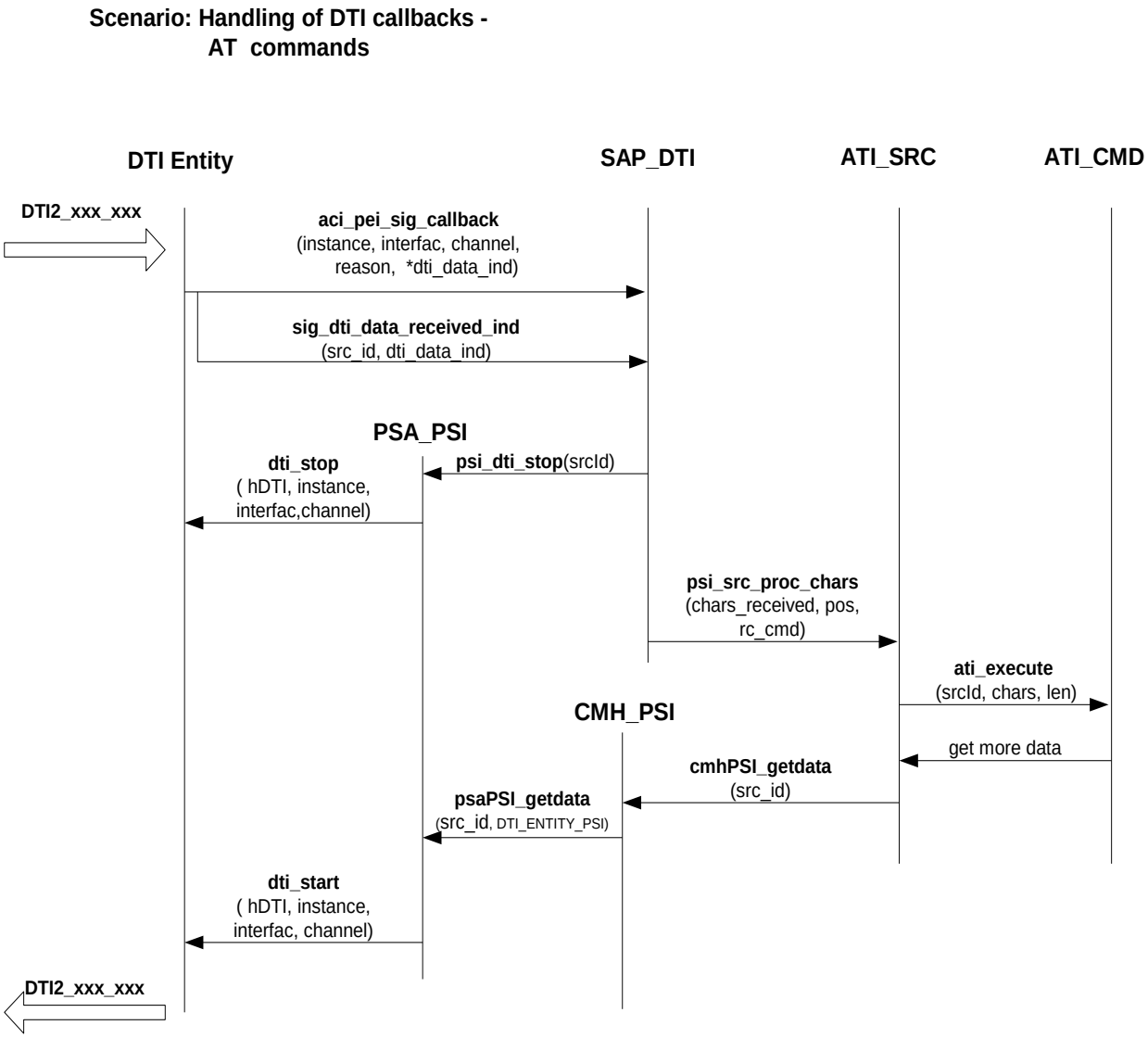


Figure 9 MSC for AT command handling from PSI

3.4.2 Handling of DTI Callbacks – Connection Opened

Description:

This is a scenario of opening a DTI connection for ACI. This happens when DTI wants to open a channel for any reason. For instance PSI sends the PSI_CONN_IND and DTI control manager creates the DTI channel for PSI and ACI. ACI as an entity connected with PSI can receive and send data via a DTI channel. If a DTI channel is opened for ACI, a callback function is called to inform ACI. The possible parameters (instance, interface, channel, reason and *dti_data_ind) are included. But only the parameter <instance> is used here.

Parameters:

Instance: Source Id

Interface: peer entity. No need to consider this because the only peer entity for ACI is PSI.

Channel: DTI channel. No need to consider this.

Reason: why the callback function has been called, for this scenario the reason is DTI connection has been opened for ACI.

*Dti_data_ind: Pointer to the received data.

Implementation:

If this situation is met, ACI should change the DTI connection status for aci_src_dti_params->isDtiConnected and inform the DTI manager. ACI also has to send out the buffered data to PSI by calling the function psi_send_buffer_data() based on the source Id. The buffered data comes from the time when DTI channel was closed.

MSC:

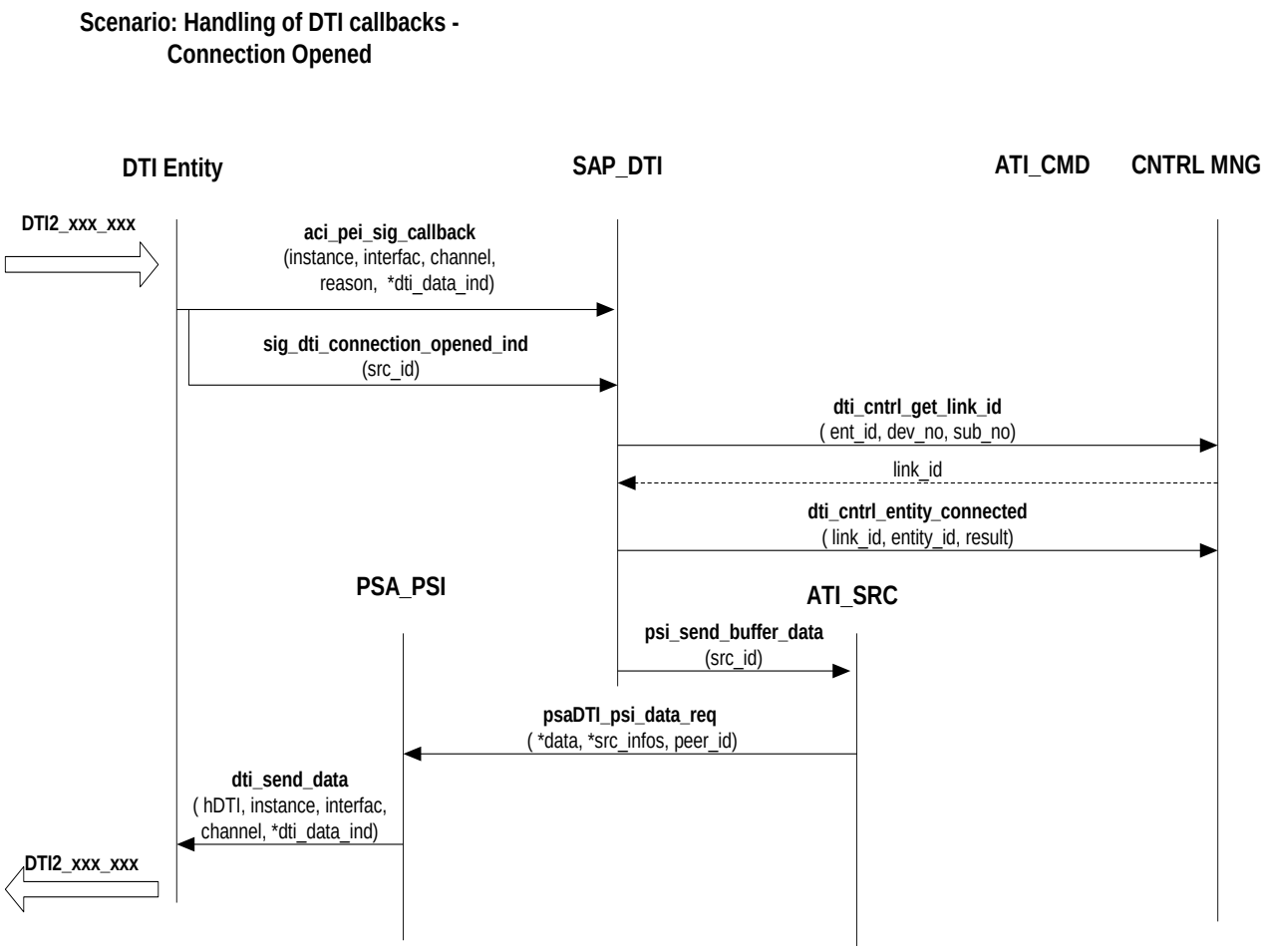


Figure 10 Handling of connection open indication

3.4.3 Handling of DTI Callbacks – Buffer Ready Indication

Description:

DTI informs ACI that the buffer is ready to store more data for the transfer. In this case, ACI will change the sending state from "not ready" to "ready" and send out the buffered data to PSI. The possible parameters (instance, interface, channel, reason and *dti_data_ind) are included.

Parameters:

Instance: Source Id

Interface: peer entity. No need to consider this because the only peer entity for ACI is PSI.

Channel: DTI channel. No need to consider this.

Reason: why the callback function has been called, for this scenario the reason is buffer ready.

*Dti_data_ind: Pointer to the received data if there is.

Implementation:

If this situation is met, ACI changes its own sending status to ready and sends out the buffered data.

MSC:

Scenario: Handling of DTI callbacks - Buffer Ready

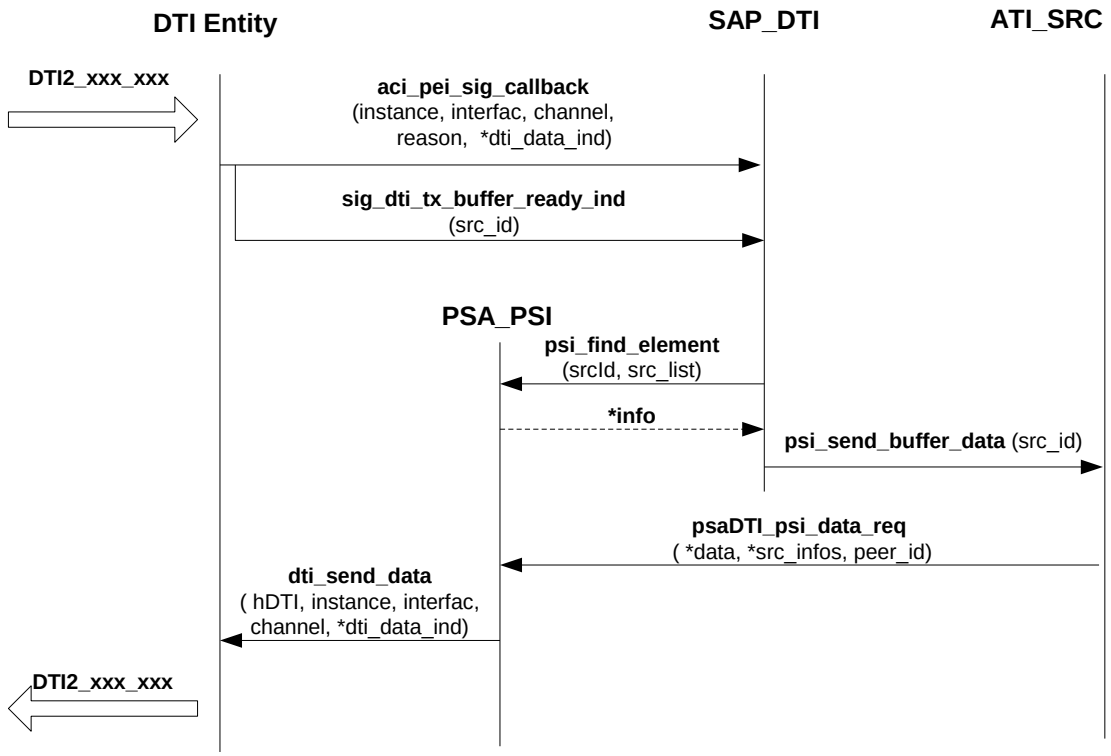


Figure 11 Handling of DTI callback – buffer full

4 Modules Implemented

4.1 Module PSA_PSI

This is a new module implemented for PSI. This module handles receiving and sending of the primitives from and to PSI. Together with CMH and the DTI control manager, PSA_PSI handles all the status managements.

4.1.1 Psa_psi.h

In this file the prototype of psa_psis.c will be defined. Additionally the following structs will be defined.

4.1.1.1 T_ACI_PSI_CAP

Definition:

```
typedef union
{
    T_dio_cap_ser  dio_cap_ser;
    T_dio_cap_ser_mux  dio_cap_ser_mux;
    T_dio_cap_pkt  dio_cap_pkt;
} T_ACI_PSI_CAP;
```

Use:

Defines the struct of the dio capability union. It is used to define a union in PSA to hold the dio capabilities.

4.1.1.2 T_ACI_PSI

Definition:

```
typedef struct
{
   U32 devId;
   T_ctrl_dio_cap psi_cap_ctrl;
   T_ACI_PSI_CAP psi_cap;
} T_ACI_PSI;
```

Use:

It defines the struct to contain the information of the driver capability.

4.1.1.3 T_DTI_MNG_PSI_MODE

Definition:

```
typedef enum{
  PSI_CONNECT_DTI = 0,
  PSI_DISCONNECT_DTI
}T_DTI_MNG_PSI_MODE;
```

Use:

in function psaPSI_Dti_Req() to decide whether to call the open request sub function or the close request sub function.

4.1.1.4 T_PSI_SHRD_PRM

Definition:

```
typedef struct
{
 T_ACI_DTR_BEHAVIOUR dtr_behaviour;
 BOOL                dtr_clearcall;
 BOOL                reconnect_to_aci;
}
T_PSI_SHRD_PRM;
```

Use:

To define <psiShrdPrm>. <PsiShrdPrm> is used to describe the line state.

4.1.1.5 T_ACI_DTI_PRC_PSI

Definition:

```
typedef struct
{
 U32 devId;
 UBYTE dlci;
 UBYTE srcId;   /* = c_id */
 T_ACI_DTI_REC_STATE  RecState;
 T_ACI_DTI_LINE_STATE LineState;
 BOOL              run_cmd;
 BOOL              first_output;
 BYTE              data_cntr;       /* DTI line flags (eg. SB-BIT) */
 T_ACI_UART_MUX_PARMS *MuxParms;
 BOOL              large_type;      /* is it a large output ? */
 T_ACI_DTI_DATA_BUF   data_buffer;
 T_ACI_DEVICE_TYPE    device_type;
```

} T_ACI_DTI_PRC_PSI;

Use:

Defines the struct of source related parameters.

4.1.2 Psa_psip.c

Here only the new added functions are described in detail. The functions over taken from the similar modules for UART or PKTIO will only be listed. This applies to all the .c file description following.

4.1.2.1 psi_dev_list

Definition:

GLOBAL T_ACI_LIST *psi_dev_list=NULL;

Use:

It is used to record the device list in ACI. See <**Error! Reference source not found.**> for more details.

4.1.2.2 psi_src_params

Definition:

GLOBAL T_ACI_LIST *psi_src_params = NULL;

Use:

It is used to record the parameters of the source list in ACI. See **Error! Reference source not found.** for more details.

4.1.2.3 psiShrdPrm

GLOBAL T_PSI_SHRD_PRM psiShrdPrm;

4.1.2.4 set_psi_share_params ()

Prototype:

GLOBAL void set_psi_share_params (SHORT dtr_behaviour, BOOL dtr_clearcall, BOOL reconnect_to_aci, USHORT src_id)

Parameters:

Returns:

Description:

for setting the shared parameter <psiShrdPrm>.

4.1.2.5 get_psi_share_params ()

Prototype:

get_psi_share_params (SHORT *dtr_behaviour, BOOL *dtr_clearcall, BOOL *reconnect_to_aci, USHORT src_id)

Parameters:

Returns:

Description:

To get the shared parameters <psiShrdPrm>.

4.1.2.6 find_psi_dev_id()

Prototype:

LOCAL BOOL find_psi_dev_id (U32 devId, void * elem)

Parameters:

Returns:

Description:

4.1.2.7 mng_psi_dev_list ()

Prototype:

LOCAL BOOL mng_psi_dev_list (T_PSI_CONNECT_IND *psi_connect_ind)

Parameters:

*psi_connect_ind:

Returns:

Description:

This function manages the PSI device list <psi_dev_list>.

4.1.2.8 mng_psi_src_param()

Prototype:

LOCAL BOOL mng_psi_src_param ( )

Parameters:

Returns:

Description:

This function manages the PSI source parameter <psi_src_params>.

4.1.2.9 find_usb_dev_no()

Prototype:

LOCAL BOOL find_usb_dev_no (U32 deviceNum, void * elem)

Parameters:

DeviceNum: device number

Elem: list to search.

Returns:

Description:

This is a help function for finding device in the USB device list.

4.1.2.10 psi_find_element ()

Prototype:

GLOBAL void *psi_find_element (UBYTE criterium, UBYTE criterium_type)

Parameters:

Criterium: what to be search.

criterium_type: type of criterium (e.g. TYPE_SRC_ID or TYPE_DEVICE).

Returns:

A pointer to the message recorded with the criterium.

Description:

This function is a find_element() function for PSI_PSA. Because PSI is composed of three different types of devices, but to the outside world only PSI would be consider. This function has the ability to define for the different request of criterium, in which list to search.

Note: The following functions are similar to the functions in PKTIO and UART, so the detailed description will not be provided, but a list of the prototype.

GLOBAL const void psaDTI_psi_data_req (T_desc2 *data, T_ACI_DTI_PRC *src_infos, T_DTI_ENTITY_ID peer_id)

GLOBAL const void psaDTI_psi_getdata(UBYTE src_id, T_DTI_ENTITY_ID peer_id)

GLOBAL const void psa_psi_connect_ind (T_PSI_CONNECT_IND *psi_connect_ind)

GLOBAL const void psa_psi_disconnect_ind(T_PSI_DISCONNECT_IND* psi_disconnect_ind)

GLOBAL const void psa_psi_line_stat_ind(T_PSI_LINE_STAT_IND *psi_line_stat_ind)

GLOBAL const void psa_psi_dti_close_cnf (T_PSI_DTI_CLOSE_CNF * psi_dti_close_cnf)

GLOBAL const void psa_psi_dti_close_ind (T_PSI_DTI_CLOSE_IND *psi_dti_close_ind)

GLOBAL const void psa_pkt_dti_open_cnf (T_PKT_DTI_OPEN_CNF * pkt_dti_open_cnf)

4.1.3 Psa_psis.c

GLOBAL const void psaDTI_data_req (T_desc2 *data, T_ACI_DTI_PRC *src_infos, T_DTI_ENTITY_ID peer_id)

GLOBAL const void psaDTI_getdata(UBYTE src_id, T_DTI_ENTITY_ID peer_id)

GLOBAL void psaPSI_ConnectRes (U32 device_no, T_ACI_PSI_CAP * psi_cap)

GLOBAL void psaPSI_ConnectRej (U32 device_no)

GLOBAL void psaPSI_Dti_Req (ULONG link_id, UBYTE peer, T_DTI_MNG_PSI_MODE con_mode)

GLOBAL void psaPSI_DTIOpenReq (U32 device_no, const char * peer_name,ULONG link_id, UBYTE dti_direction)

GLOBAL void psaPSI_DTICloseReq (U32 device_no)

4.2 Module CMH_PSI

4.2.1 Cmh_psi.h

4.2.1.1 Function prototype

Prototype definition for file cmh_psir.c and cmh_psif.c.

4.2.2 Cmh_psir.c

GLOBAL void cmhPSI_DetectedESC_DTR(U32 device, UBYTE dlci, UBYTE cause)

4.2.3 Cmh_psif.c

This module defines the functions used by the command handler for the PSI module.

GLOBAL T_ACI_DTI_PRC *cmhPSI_find_dlci (T_ACI_LIST *search_list, U32 device, UBYTE dlci)

GLOBAL void cmhPSI_erase_elem_received_cmd (UBYTE srcId)

GLOBAL void cmhPSI_SetComParToUnchanged(T_comPar *comPar)

4.3 Module ATI_SRC_PSI

4.3.1 Ati_src_psi.h

Function prototype definition for ati_src_psi.c and ati_src_psi_io.c.

4.3.2 Ati_src_psi.c

Other functions that should be implemented are listed below:

*GLOBAL void psi_InitCmdStruct(T_ACI_DTI_PRC *cmd_struct)*

*GLOBAL void psi_InitCmdStruct(T_ACI_DTI_PRC *cmd_struct)*

GLOBAL void psi_erase_source(UBYTE srcId)

GLOBAL BOOL atiPSI_dti_cb(UBYTE dti_id, T_DTI_CONN_STATE result_type)

*GLOBAL BOOL uart_src_proc_chars (UBYTE *chars, USHORT len, T_ACI_DTI_PRC *elem)*

4.3.3 Ati_src_psi_io.c

*GLOBAL void uart_src_result_cb (UBYTE src_id, T_ATI_OUTPUT_TYPE output_type, UBYTE *output, USHORT output_len)*

GLOBAL void uart_src_line_state_cb (UBYTE src_id, T_ATI_LINE_STATE_TYPE line_state_type, ULONG line_state_param)

GLOBAL void uart_send_buffer_data (UBYTE src_id)

*LOCAL void io_DTIsendString (UBYTE *string, USHORT string_len,T_ACI_DTI_PRC *src_infos, T_ATI_OUTPUT_TYPE output_type)*

4.4 Module SAP_DTI

In this module only some changes according to section **Error! Reference source not found.** will be done.

4.5 Other Necessary Modifications

All the functions that are UART related will adapt to PSI.

5 Test Plan

5.1 Windows Simulation Test

5.1.1 New Test Cases

About 20 new added test cases are added in ACIDTI from test case 101 onwards.

5.1.2 Simulation Tests

With and without flag: FF_PSI

ACI

GACI

ACIDTI

5.2 Target Test

5.2.1 Taget Build

MFW

ACI Only

GOLite

5.2.2 Target Test

MO Call

MO SMS

MT Call

AT Command

UART