Message Sequence Charts

DTILIB

Document Number:	8448.201.01
Version:	014
Status:	Proposed
Approval Authority:	
Creation Date:	2000-Oct-16
Last changed:	2004-Dec-13 by Steffen Winter
File Name:	dtilib.doc

Important Notice

Change History

Date	Changed by	Approved by	Version	Status	Notes
2000-Oct-16	Ola Flatus		001	Proposed	1
2000-Oct-30	Steffen Winter		002	Proposed	2
2001-Feb-19	Ola Flatus		003	Proposed	3
2001-Feb-28	Ola Flatus		004	Proposed	4
2001-Apr-11	Ola Flatus		005	Proposed	5
2001-Apr-26	Ola Flatus		006	Proposed	6
2001-May-25	Ola Flatus		007	Proposed	7
2001-Oct-09	Mafred Gutheins		008	Proposed	8

2001-Oct-12	Mafred Gutheins		009	Proposed	9
2001-Oct-16	Mafred Gutheins		010	Proposed	10
2001-Nov-27	Tobias Vogler		011	Proposed	11
2002-Mar-06	Steffen Winter		012	Proposed	12
2003-Apr-25	Steffen Winter		013	Proposed	13
2004-Dec-13	Steffen Winter		014	Proposed	14

Notes:

1. Design
2. Redesign
3. Some parameter in the entity structure changed
4. Some parameter by calling the functions changed
5. Some parameter by calling the functions changed
6. Some parameter by calling the functions changed
7. Introduction to DTIL IB, structure and chapter 9 added
8. Introduction revised
9. Introduction reworked
10. Introduction moved to separate document
11. Added flushing mechanism for dti_close(); corrected minor errors in parameter lists
12. Replace charcter MSCs by drawings
13. Use TI template; remove descriptorlist handling functions
14. Slight change of connection protocol in case of collision

Table of Contents

List of References

[1] Rec. T.4 Standardisation of group 3 facsimile apparatus for document transmission;
(CCITT-T.4, 1984)

[2] ITU-T Recommendation T.30; Series T: Terminal equipments and protocols for telematic services;
Procedures for document facsimile transmission in the general switched
telephone network;
(ITU-T.30, 1996)

[3] ITU-T Recommendation T.31; Terminals for telematic services;
Asynchronous facsimile DCE control - service class 1
(ITU-T.31, 1995)

[4] ITU-T Recommendation T.32; Terminals for telematic services;
Asynchronous facsimile DCE control - service class 2
(ITU-T.32, 1995)

[5] Rec. T.35; Terminal equipment and protocols for telematic services;
Procedures for the allocation of CCITT definde codes for non-standard facilities;
(CCITT-T.35, 1991)

[6] ITU-T Recommendation V.25 ter; Series V: data communication over the telephone network;
Interfaces and voiceband modems; Serial asynchronous automatic dialling and control
(ITU-T V.25 ter, 1997)

[7] Rec. V.42 bis Data compression procedures for data circuit terminating equipment (DCE) using error correction procedures;
(CCITT-V.42 bis, 1990)

[8] Rec. V.110 (Blue book, Vol. VIII, Fascicle VIII.1) Support of data terminal equipments (DTEs) with V-series type interfaces by an integrated services digital network (ISDN);
(CCITT-V.110, 1988)

[9] European digital cellular telecommunications system (Phase 2);
GSM Public Land Mobile Network (PLMN) connection types;
(GSM 3.10, September 1994, version 4.3.1)

[10] European digital cellular telecommunications system (Phase 2);
Technical realisation of facsimile group 3 transparent;
(GSM 3.45, September 1995, version 4.5.0)

[11] Digital cellular telecommunications system (Phase 2);
Mobile radio interface layer 3 specification;
(GSM 4.08, November 1996, version 4.17.0)

[12] European digital cellular telecommunications system (Phase 2);
Rate adaptation on the Mobile Station - Base Station System (MS - BSS) Interface;
(GSM 4.21, May 1995, version 4.6.0)

[13] European digital cellular telecommunications system (Phase 2);
Radio Link Protocol (RLP) for data and telematic services on the Mobile Station - Base Station System (MS - BSS) interface and the Base Station System - Mobile-service Switching Centre (BSS - MSC) interface
(GSM 4.22, September 1994, version 4.3.0)

[14] European digital cellular telecommunications system (Phase 2);
Radio Link Protocol (RLP) for data and telematic services on the Mobile Station - Base Station System (MS - BSS) interface and the Base Station System - Mobile-service Switching Centre (BSS - MSC) interface
(Amendment prA1 for GSM 4.22, version 4.3.0)
(GSM 4.22, March 1995, version 4.4.0)

[15] European digital cellular telecommunications system (Phase 2);
General on Terminal Adaptation Functions (TAF) for Mobile Stations (MS);
(GSM 7.01, December 1995, version 4.10.0)

[16] European digital cellular telecommunications system (Phase 2);
Terminal Adaptation Functions (TAF) for services using asynchronous bearer capabilities;
(GSM 7.02, September 1994, version 4.5.1)

[17] European digital cellular telecommunications system (Phase 2);
Terminal Adaptation Functions (TAF) for services using synchronous bearer capabilities;
(GSM 7.03, September 1994, version 4.5.1)
[18] Digital cellular telecommunications system (Phase 2);
Use of Data Terminal Equipment - Data Circuit terminating Equipment (DTE - DCE) interface for Short Message Service (SMS) and Cell Broadcast Services (CBS);
(GSM 7.05, November 1996, version 4.8.0)
[19] Digital cellular telecommunications system (Phase 2);
AT command set for GSM Mobile Equipment (ME)
(GSM 7.07, May 1996, version 4.1.0)
[20] Digital cellular telecommunication system (Phase 2);
Mobile Station (MS) conformance specification;
Part 1: Conformance specification
(GSM 11.10-1, November 1996, version 4.17.0)
[21] Digital cellular telecommunications system (Phase 2);
Mobile Station (MS) conformance specification;
Part 2: Protocol Implementation Conformance Statement (PICS)
proforma specification
(GSM 11.10-2, May 1996, version 4.15.0)
[22] Digital cellular telecommunications system (Phase 2);
Mobile Station (MS) conformance specification;
Part 3: Layer 3 (L3) Abstract Test Suite (ATS)
(GSM 11.10-3, November 1996, version 4.17.0)
[23] Proposal for Rate Adaptation implemented on a DSP;
(C. Bianconi, Texas Instruments, January 1998, version 1.0)
[24] MCU-DSP Interfaces for Data Applications;
Specification S844
(C. Bianconi, Texas Instruments, March 1998, version 0.1)
[25] Users Guide
6147.300.96.100; Condat GmbH
[26] Service Access Point RA
8411.100.98.100; Condat GmbH
[27] Service Access Point RLP
8411.101.98.100; Condat GmbH
[28] Service Access Point L2R
8411.102.98.100; Condat GmbH
[29] Service Access Point FAD
8411.103.98.100; Condat GmbH
[30] Service Access Point T30
8411.104.98.100; Condat GmbH
[31] Service Access Point ACI
8411.105.98.100; Condat GmbH
[32] Message Sequence Charts RLP
8411.201.98.100; Condat GmbH
[33] Message Sequence Charts L2R
8411.202.98.100; Condat GmbH
[34] Message Sequence Charts FAD
8411.203.98.100; Condat GmbH
[35] Message Sequence Charts T30
8411.204.98.100; Condat GmbH
[36] Message Sequence Charts ACI
8411.205.98.100; Condat GmbH
[37] Proposal for Fax & Data Integration; March 1998
8411.300.98.100; Condat GmbH
[38] Test Specification RLP
8411.401.98.100; Condat GmbH
[39] Test Specification L2R
8411.402.98.100; Condat GmbH

[40] Test Specification FAD
8411.403.98.100; Condat GmbH
[41] Test Specification T30
8411.404.98.100; Condat GmbH
[42] Test Specification ACI
8411.405.98.100; Condat GmbH
[43] SDL Specification RLP
8411.501.98.100; Condat GmbH
[44] SDL Specification L2R
8411.502.98.100; Condat GmbH
[45] SDL Specification FAD
8411.503.98.100; Condat GmbH
[46] SDL Specification T30
8411.504.98.100; Condat GmbH
[47] SDL Specification ACI
8411.505.98.100; Condat GmbH
[48] Technical Documentation RLP
8411.701.98.100; Condat GmbH
[49] Technical Documentation L2R
8411.702.98.100; Condat GmbH
[50] Technical Documentation FAD
8411.703.98.100; Condat GmbH
[51] Technical Documentation T30
8411.704.98.100; Condat GmbH
[52] Technical Documentation ACI
8411.705.98.100; Condat GmbH
[53] Technical Documentation DTI-SAP
8411.110.00.010
[54] Introduction DTI
8448.201.01

1 Introduction

This document is a specification for the DTI (Data Transmission Interface) library. An introduction into the DTI SAP and into the DTILIB can be found in [**54**].

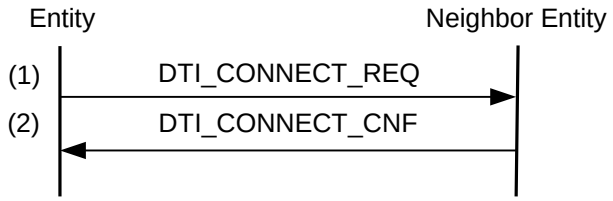
2 Primitive Communication

To understand DTI communication between two entities, the primitive flow is shown below. **In this description the entity sends data uplink and the neighbor entity sends data downlink. Both sides are equivalent.**

2.1 Initialization

The following description belongs to DTI version 2. The initialization procedure in DTI version 1 can be found in [54].
Only one side needs to send a Connect primitive to initialize the connection. If one side receives a Connect primitive it has to answer with a Connect Confirm primitive. The connection is established if the sender of a Connect pimitive receives an appropriate Connect Confirm primitive.

2.1.1 Initiated by the entity

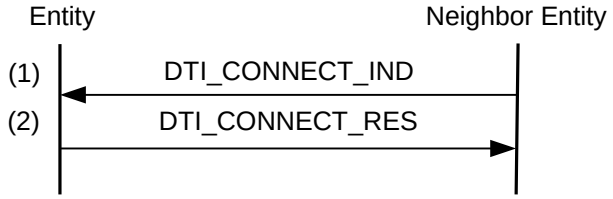


Description:

(1) To start or reset the DTI connection the entity sends a Connect primitive to there neighbor entity.

(2) If the neighbor entity is also ready to start the DTI connection it will answer with a Connect Confirm primitive. After sending respectively receiving the Connect Confirm primitive each side must send a Flow Control primitive befor it can receive a Data primitive.

2.1.2 Initiated by the neighbor entity

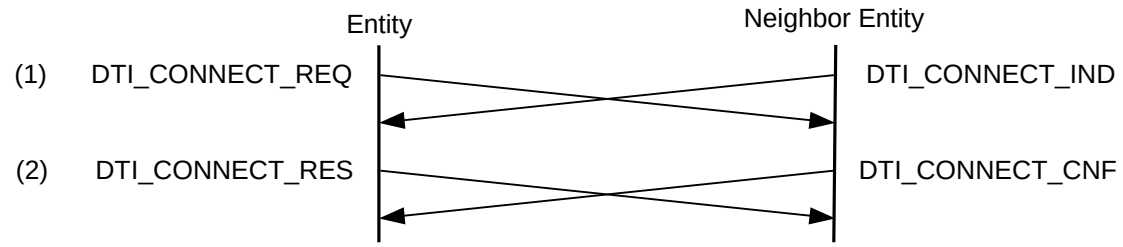


Description:

(1) To start or reset the DTI connection the neighbor entity sends Connect primitive to the entity.

(2) If the entity is also ready to start the DTI connection it will answer with a Connect Confirm primitive. After sending respectively receiving the Connect Confirm primitive each side must send a Flow Control primitive befor it can receive a Data primitive.

2.1.3 Collision



Description:

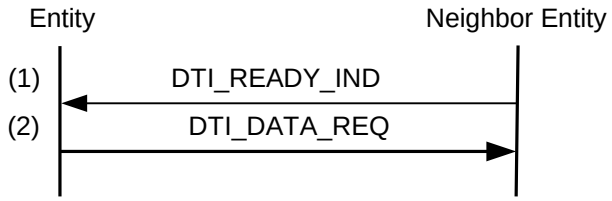
(1) To start or reset the DTI connection both sides send a Connect primitive at the same time.

(2) Each side must answer with a Connect Confirm primitive. After receiving the Connect Confirm primitive each side must send a Flow Control primitive before it can receive a Data primitive.

2.2 Send and Receive Data primitives

The flow control of each direction is independent to the other direction.

2.2.1 Sending Data primitives

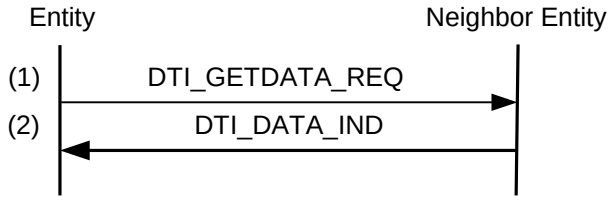


Description:

(1) The neighbor entity indicates by sending a Flow Control primitive that it is ready to receive one Data primitive.

(2) When there are data to send then the entity sends one Data primitive. Before the next Data primitive can be sent the neighbor entity must send a new Flow Control primitive.

2.2.2 Receiving Data primitives



Description:

(1) The entity indicates by sending a Flow Control primitive that it is ready to receive one Data primitive.

(2) When there are data to send then the neighbor entity sends one Data primitive. Before the next Data primitive can be sent the entity must send a new Flow Control primitive.

2.3 Disconnection

The following description belongs to DTI version 2. In DTI version 1 a diconnection procedure does not exist.

2.3.1 Initiated by the entity

Description:

(1) The entity indicates by sending a Disconnect primitive that the DTI connection can not be used any longer.

2.3.2 Initiated by the neighbor entity

Description:

(1) The neighbor entity indicates by sending a Disconnect primitive that the DTI connection can not be used any longer.

3 Internal Communication

To understand the communication between the entity and DTILIB, the signal flow is shown below. **In this description the entity sends data uplink und the neighbor entity sends data downlink. Both sides are equivalent.**

3.1 Initialization

The following description belongs to DTI version 2. See chapter 3.2 for the initialization with DTI version 1.
Only one side needs to send a Connect primitive to initialize the connection. If one side receives a Connect primitive it has to answer with a Connect Confirm primitive. The connection is established if the sender of a Connect pimitive receives an appropriate Connect Confirm primitive.

3.1.1 States

The states for communication are:

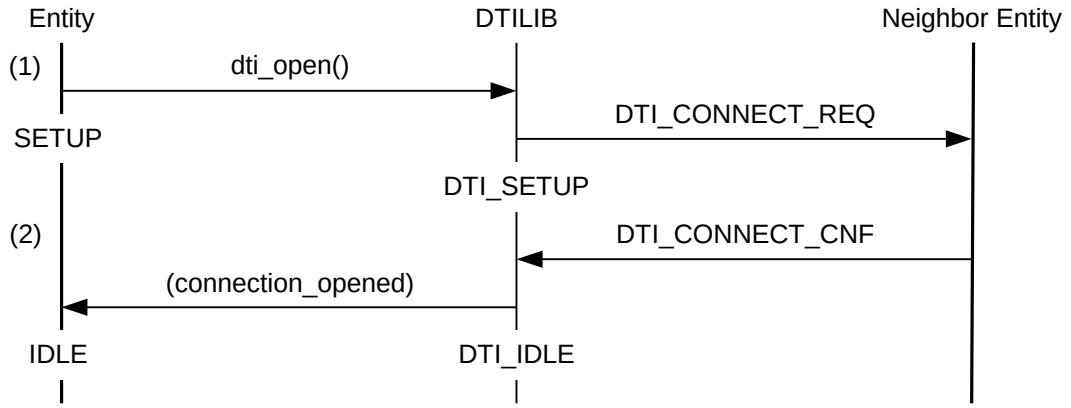
Entity:

CLOSED	- DTI connection is not established
SETUP	- Waiting for connection_opened signal from DTILIB
IDLE	- DTI connection is opened

DTILIB:

DTI_CLOSED	- DTI connection is not established
DTI_CONNECTING	- Connect primitive received, but no appropriate dti_open() was called
DTI_SETUP	- Waiting for Connect Confirm primitive
DTI_IDLE	- DTI connection is opened
DTI_CLOSING	- DTI connection is to be closed after all data primitives have been sent

3.1.2 Initiated by the entity

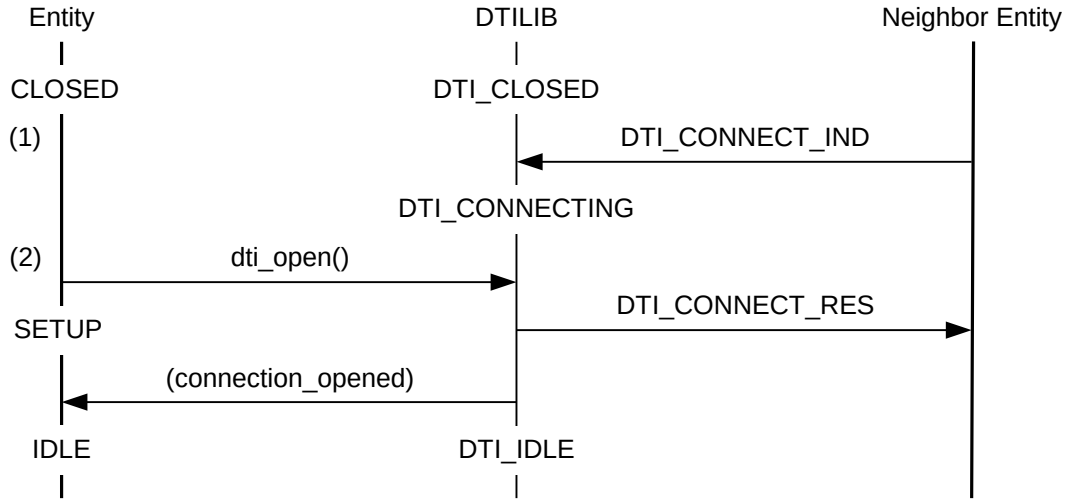


Description:

(1) DTILIB is in any state except DTI_SETUP and DTI_CONNECTING state. The entity is in any state except SETUP state. The entity calls the function dti_open(), DTILIB sends a Connect primitive and enters DTI_SETUP state. The entity enters SETUP state.

(2) DTILIB receives a Connect Confirm primitive, calls the callback function sig_callback() with the reason parameter (connection_opened) and enters DTI_IDLE state. If the callback function is called with this parameter the entity enters IDLE state.

3.1.3 Initiated by the neighbor entity

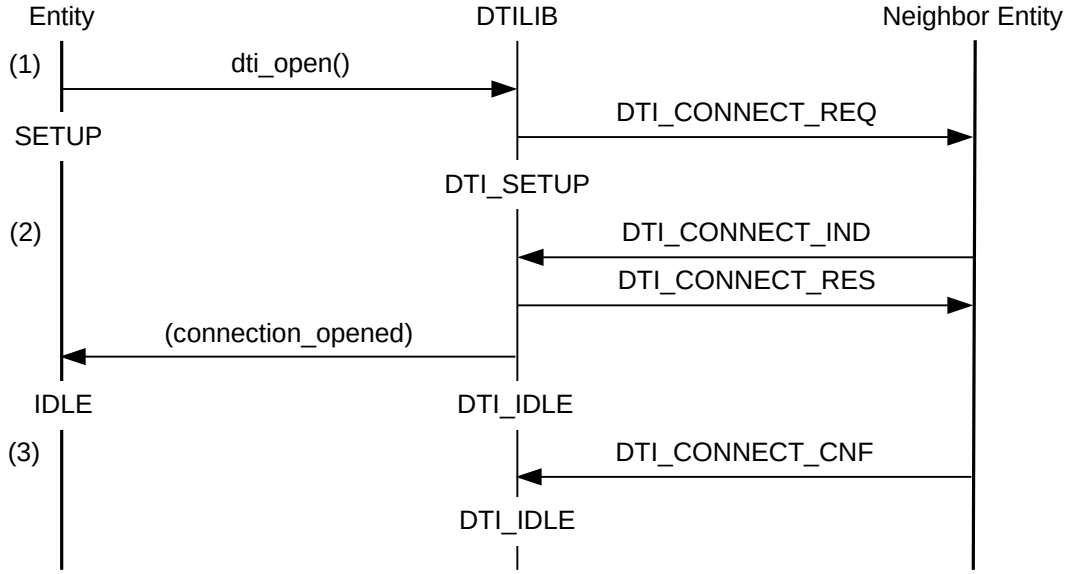


Description:

(1) DTILIB is in DTI_CLOSED state and the entity is in CLOSED state. DTILIB receives a Connect primitive and enters DTI_CONNECTING state.

(2) The entity calls the function dti_open() and enters SETUP state. DTILIB sends a Connect Confirm primitive, calls the callback function sig_callback() with the reason parameter (connection_opened) and enters DTI_IDLE state. If the callback function is called with this parameter the entity enters IDLE state.

3.1.4 Collision



Description:

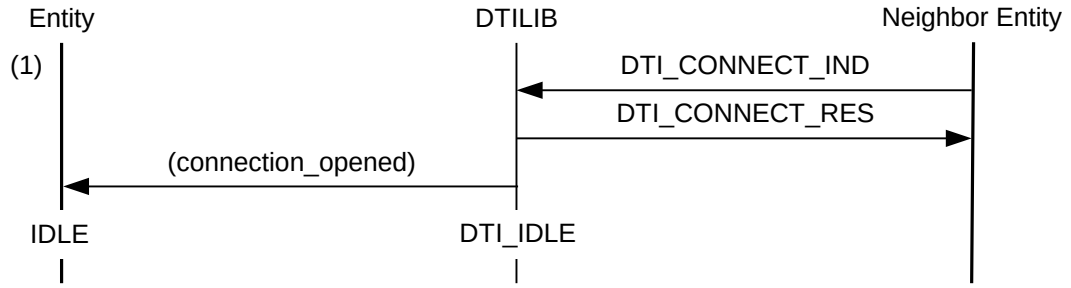
(1) DTILIB is in any state except DTI_SETUP and DTI_CONNECTING state. The entity is in any state except SETUP state. The entity calls the function dti_open(), DTILIB sends a Connect primitive and enters DTI_SETUP state. The entity enters SETUP state.

(2) DTILIB receives a Connect primitive instead of Connect Confirm primitive and sends an appropriate Connect Confirm primitive. Since the Connect primitive already indicates that the neighbor entity is ready to open the connection DTILIB calls the callback function sig_callback() with the reason parameter (connection_opened) and enters DTI_IDLE state. If the callback function is called with this parameter the entity enters IDLE state.

(3) DTILIB receives a Connect Confirm primitive and stays in DTI_IDLE state.

3.1.5 Reset

Upon reception of a Connect primitive for an already established connection the entity has to answer with a Connect Confirm primitive, the send and receive state machines must enter initial states and DTILIB has to release all Data primitives in send-queue.



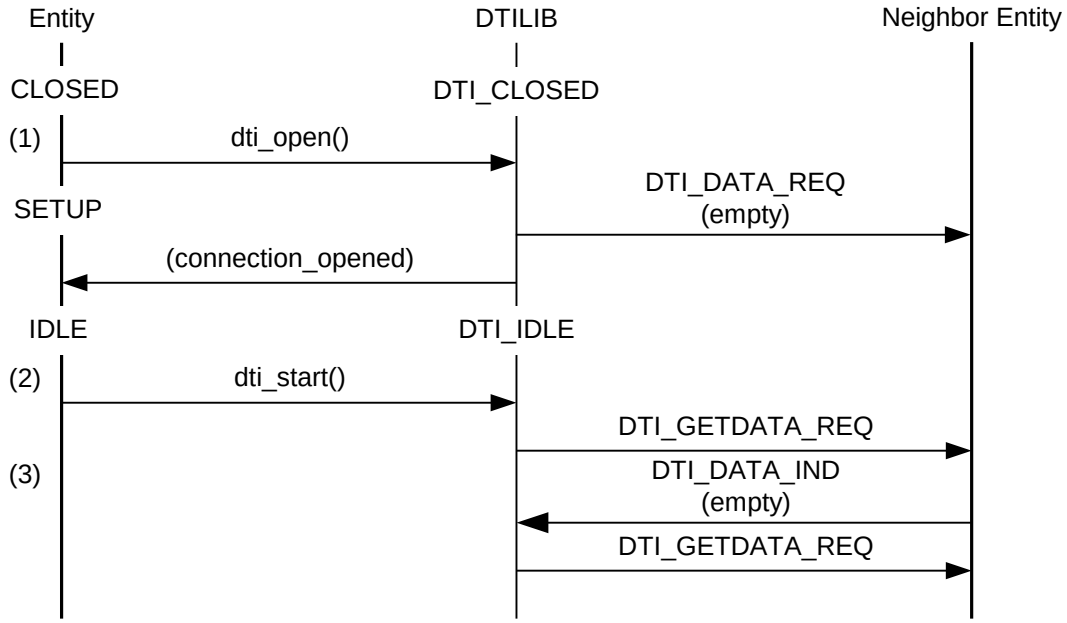
Description:

(1) DTILIB is in any state except DTI_CLOSED and DTI_SETUP state. The entity is in any state except CLOSED and SETUP state. DTILIB receives a Connect primitive, sends an appropriate Connect Confirm primitive, calls the callback function sig_callback() with the reason parameter (connection_opened) and enters DTI_IDLE state. If the callback function is called with this parameter the entity enters IDLE state.

3.2 Initialization with DTI version 1

DTI version 1 SAP does not support the Connect primitives of DTI version 2. That means, it is not possible to synchronize flow control states at initialization. To solve this problem DTILIB sends an empty Data primitive if the function dti_open() is called.

3.2.1 Initiated by the entity



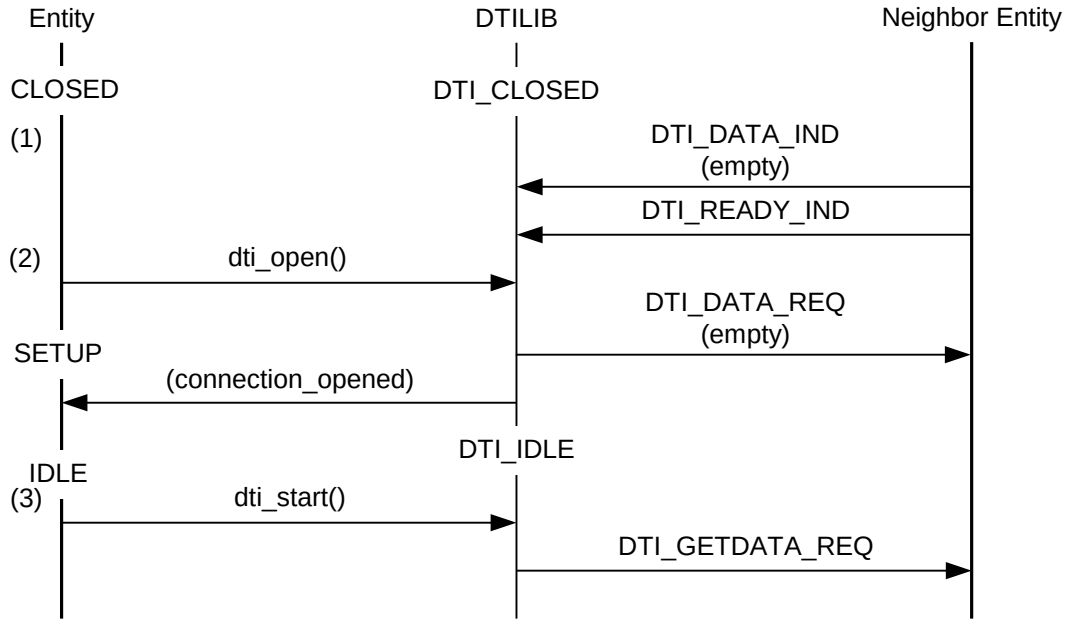
Description:

(1) DTILIB is in DTI_CLOSED state and the entity is CLOSED state. The entity calls the function dti_open() and enters SETUP state. DTILIB sends an empty Data primitive for synchronization and calls the callback function sig_callback() with the reason parameter (connection_opened). DTILIB enters DTI_IDLE state. If the callback function is called with this parameter the entity enters IDLE state.

(2) The entity calls the function dti_start() to indicate that it is ready to receive Data primitives. DTILIB sends a Flow Control primitive.

(3) If the first Data primitive, received by the DTILIB, is an empty Data primitive then DTILIB assumed that this is the sychronization primitive and responds automatically with a FLow Control primitive.

3.2.2 Initiated by the neighbor entity



Description:

(1) DTILIB is in DTI_CLOSED state and the entity is CLOSED state. DTILIB receives a Data primitive and a Flow COntrol primitive which it can not associate to a DTI link. So DTILIB discards these primitives.

(2) The entity calls the function dti_open() and enters SETUP state. DTILIB sends an empty Data primitive for synchronization and calls the callback function sig_callback() with the reason parameter (connection_opened). DTILIB enters DTI_IDLE state. If the callback function is called with this parameter the entity enters IDLE state.

(3) The entity calls the function dti_start() to indicate that it is ready to receive Data primitives. DTILIB sends a Flow Control primitive.

3.3 Sending Data primitives

3.3.1 States

The states for communication are:

Entity:

TX_IDLE	- The entity must not send Data primitives (initial state)
TX_READY	- The entity can send Data primitives

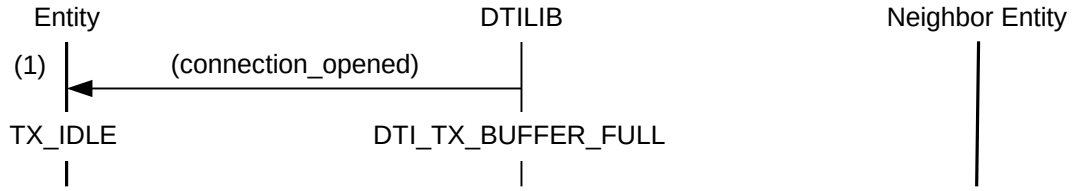
DTILIB:

DTI_TX_IDLE	- Waiting for Flow Control primitive (initial state)
DTI_TX_FLOW	- Flow Control primitive received
DTI_TX_BUFFER_FULL	- Waiting for Flow Control primitive and send-queue is full

The initial state of the entities sends state machine is TX_IDLE. That means the entity must not send Data primitives.

3.3.2 Set initial state (don't use send-queue)

If the developer of an entity does not want to use the send-queue of DTILIB for a DTI connection, the value *link_options* in the dti_open() function call must be set to DTI_QUEUE_UNUSED.

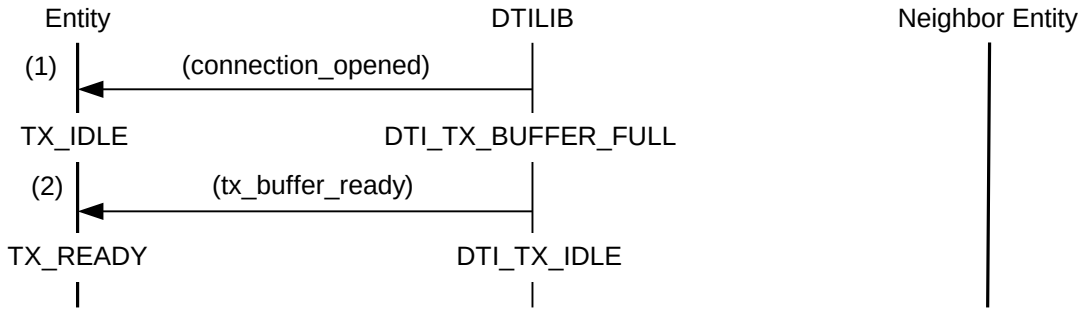


Description:

(1) If DTILIB calls the callback function sig_callback() with the reason parameter (connection_opened), it enters DTI_TX_BUFFER_FULL state. If the callback function is called with this parameter, the entity's send state machine must enter TX_IDLE state.

3.3.3 Set initial state (use send-queue)

If the developer of an entity wants to use the send-queue of DTILIB for a DTI connection, the value *queue_size* in the dti_open() function call must be set to an appropriate threshold value. In case of DTI version 1 DTILIB supports a maximum queue size of 1.



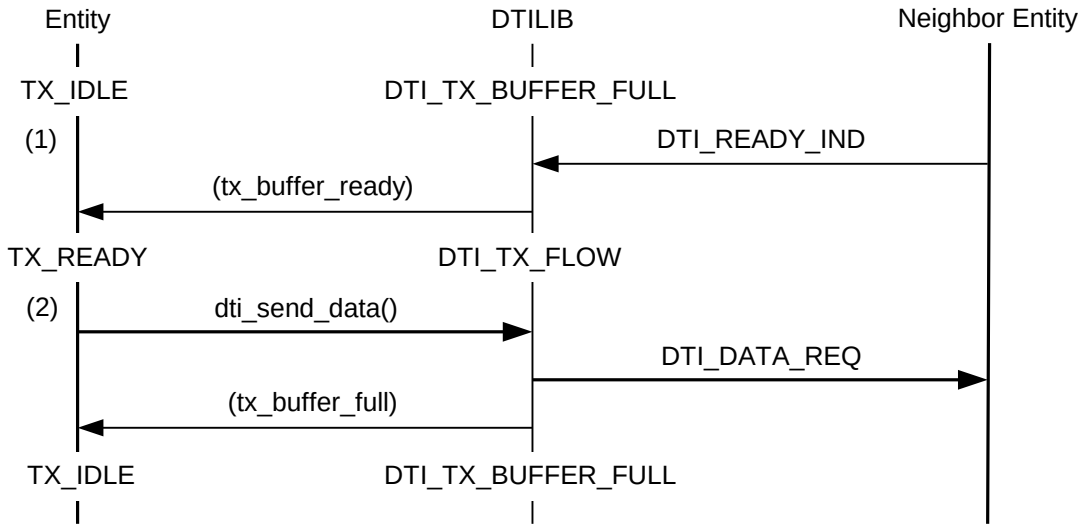
Description:

(1) If DTILIB calls the callback function sig_callback() with the reason parameter (connection_opened), it enters DTI_TX_BUFFER_FULL state. If the callback function is called with this parameter, the entity's send state machine must enter TX_IDLE state.

(2) Because the send-queue is empty, DTILIB calls the callback function sig_callback() with the reason parameter (tx_buffer_ready) and enters DTI_TX_IDLE state. If the callback function is called with this parameter, the entity's send state machine enters TX_READY state.

3.3.4 Flow Control without send-queue

If the developer of an entity does not want to use the send-queue of DTILIB for a DTI connection, the value *link_options* in the dti_open() function call must be set to DTI_QUEUE_UNUSED.



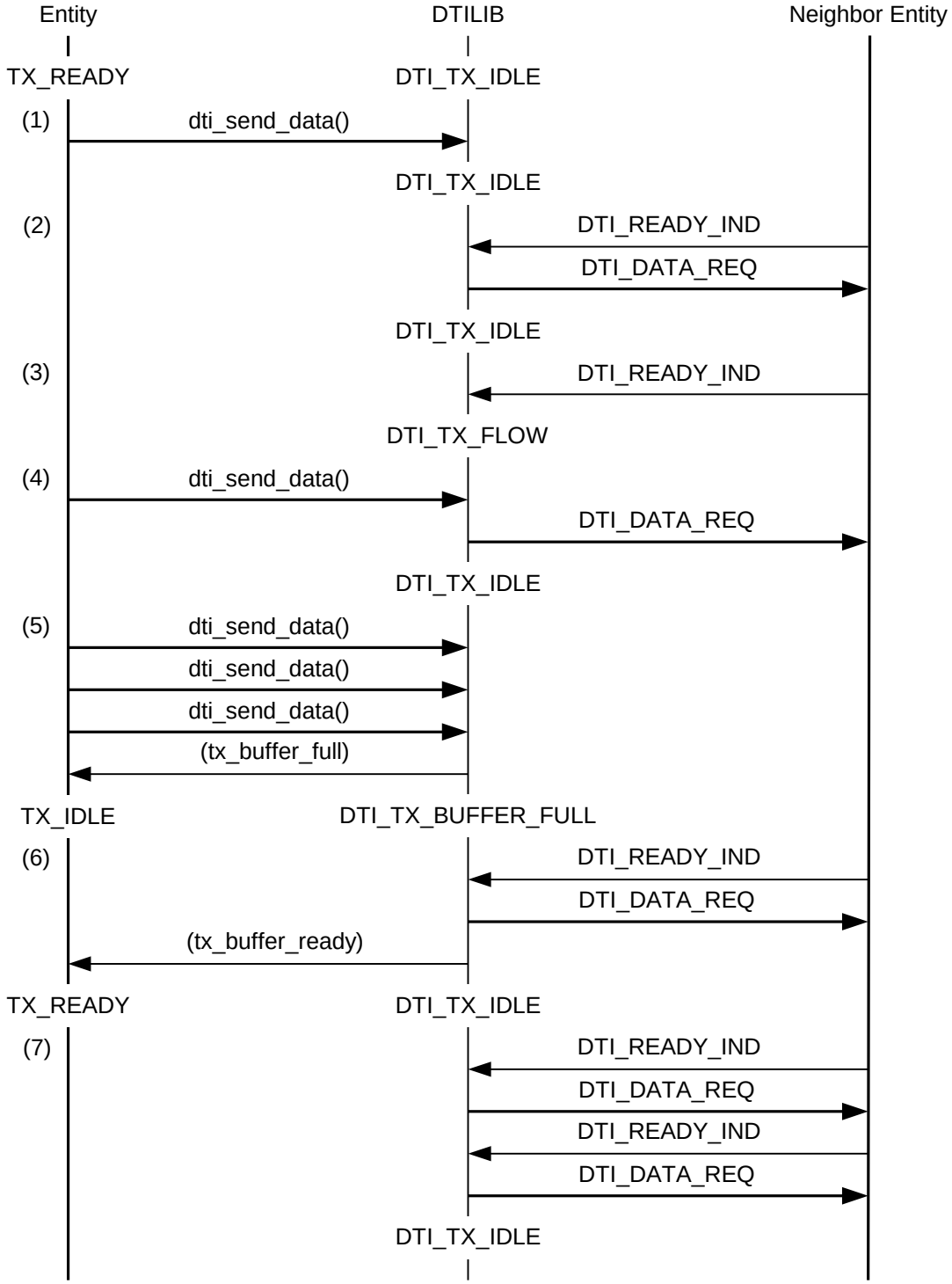
Description:

(1) DTILIB is in DTI_TX_BUFFER_FULL state and the entity is in TX_IDLE state. DTILIB receives a Flow Control primitive, enters DTI_TX_FLOW state and calls the callback function sig_callback() with the reason parameter (tx_buffer_ready) to indicate that the entity can send a Data primitive. The entity enters TX_READY state.

(2) If there are data to send available, then the entity calls the funtion dti_send_data(). DTILIB sends the Data primitive and calls the callback function sig_callback() with the reason parameter (tx_buffer_full) to indicate that the entity must not send send Data primitives any more. The entity enters TX_IDLE state again.

3.3.5 Flow Control with send-queue

If the developer of an entity wants to use the send-queue of DTILIB for a DTI connection, the value *queue_size* in the dti_open() function call must be set to an appropriate threshold value. In case of DTI version 1 DTILIB supports a maximum queue size of 1. In this example *queue_size* is set to 3 to show the mechanism clearly.



Description:

(1) DTILIB is in DTI_TX_IDLE state and the entity is in TX_READY state. The entity calls the funtion dti_send_data(), but DTILIB has not received a Flow Control primitive yet. So DTILIB stores the data to send and stays in DTI_TX_IDLE state.

(2) DTILIB receives a Flow Control primitive, sends the stored Data primitive and stays in DTI_TX_IDLE state.

(3) DTILIB receives a Flow Control primitive again, but it has not data to send. So DTILIB enters DTI_TX_FLOW state.

(4) The entity calls the function dti_send_data() and DTILIB sends immediately a Data primitive because it has the Flow Control primitive already received. DTILIB enters DTI_TX_IDLE state again.

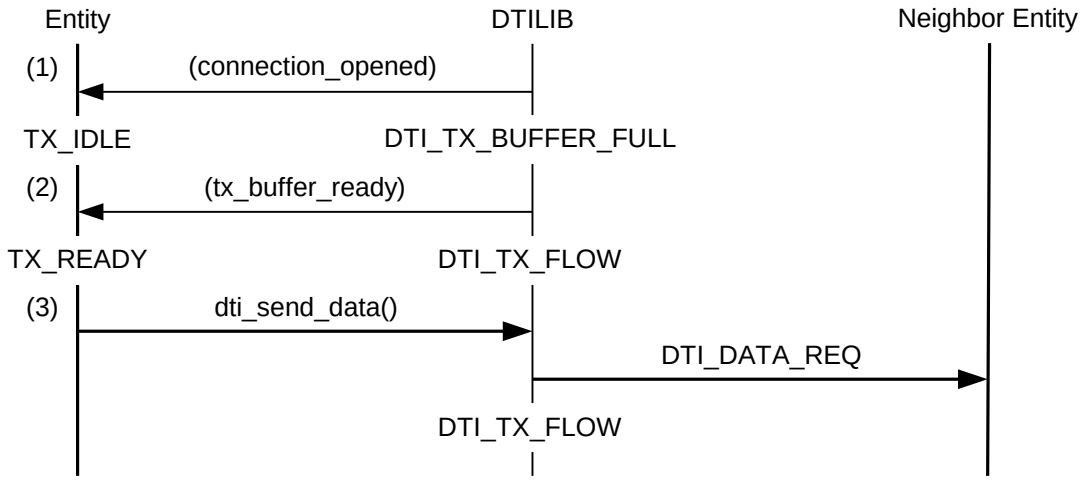
(5) The entity calls the funtion dti_send_data() three times. In this example the send-queue-size is defined as 3. At the third call of dti_send_data() the queue has reached the upper level. So DTILIB calls the callback function sig_callback() with the reason parameter set to (tx_buffer_full). If the callback function is called with this parameter the entity enters TX_IDLE state and must not call the dti_send_data() function. DTILIB enters DTI_TX_BUFFER_FUL state.

(6) DTILIB receives one Flow Control primitive, sends the next stored Data primitive and calls the callback function with the reason parameter set to (tx_buffer_ready). If the callback function is called with this parameter the entity enters TX_READY state and can call the dti_send_data() function again. DTILIB enters DTI_TX_IDLE state.

(7) DTILIB receives Flow Control primitives, sends the stored data in order and stays in DTI_TX_IDLE state.

3.3.6 Send data without Flow Control (for testing)

To simplify testcases it is possible to run DTILIB in a mode where it does not send or expect Flow Control primitives. **This mode is only useful for testing on PC. It is not possible to use this mode on target.** To enable this mode the value *link_options* in the dti_open() function call must be set to DTI_FLOW_CNTRL_DISABLED.



Description:

(1) If DTILIB calls the callback function sig_callback() with the reason parameter (connection_opened), it enters DTI_TX_BUFFER_FULL state. If the callback function is called with this parameter, the entity's send state machine must enter TX_IDLE state.

(2) Because DTILIB assumes reception of a Flow Control primitive, it calls the callback function sig_callback() with the reason parameter (tx_buffer_ready) and enters DTI_TX_FLOW state. If the callback function is called with this parameter, the entity's send state machine enters TX_READY state.

(3) The entity calls the function dti_send_data() and DTILIB sends immediately a Data primitive. Because DTILIB assumes reception of a Flow Control primitive it stays in DTI_TX_FLOW state.

3.4 Receiving Data primitives

3.4.1 States

The states for communication are:

Entity:

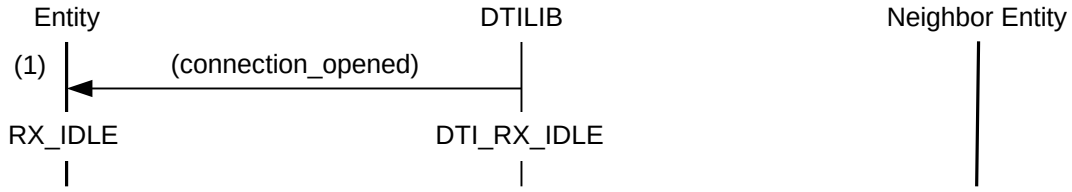
RX_IDLE - Data reception stopped (initial state)
RX_READY - Ready to reveive data

DTILIB:

DTI_RX_IDLE - No Flow Control primitive sent (initial state)
DTI_RX_READY - Flow Control primitive sent expect Data primitive
DTI_RX_STOPPED - Flow Control primitive sent, but data-flow was stopped by the entity
DTI_RX_READY_NFC – Data primitive received, but a flow control primitive is not sent yet.

The initial state of the entity's receive state machine is RX_IDLE. That means the entity must call dti_start() to receive Data primitives.

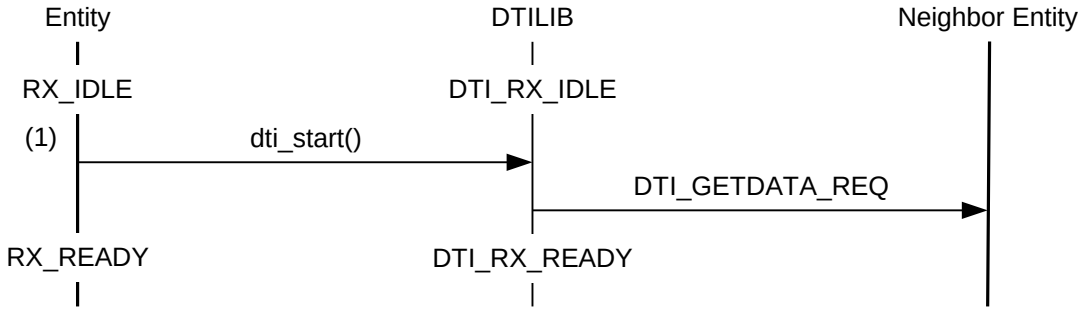
3.4.2 Set initial state



Description:

(1) If DTILIB calls the callback function sig_callback() with the reason parameter (connection_opened), it enters DTI_RX_IDLE state. If the callback function is called with this parameter the entity's receive state machine must enter RX_IDLE state.

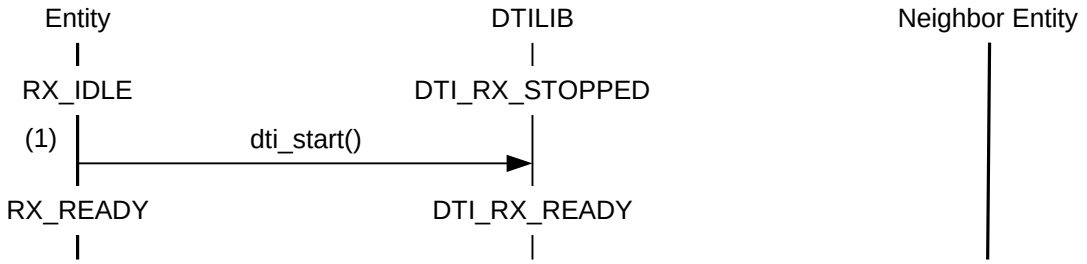
3.4.3 Start reception



Description:

(1) DTILIB is in DTI_RX_IDLE state and the entity is in RX_IDLE state. The entity calls the function dti_start() and DTILIB sends a Flow Control primitive to request a Data primitive. The entity enters RX_READY state and DTILIB enters DTI_RX_READY state.

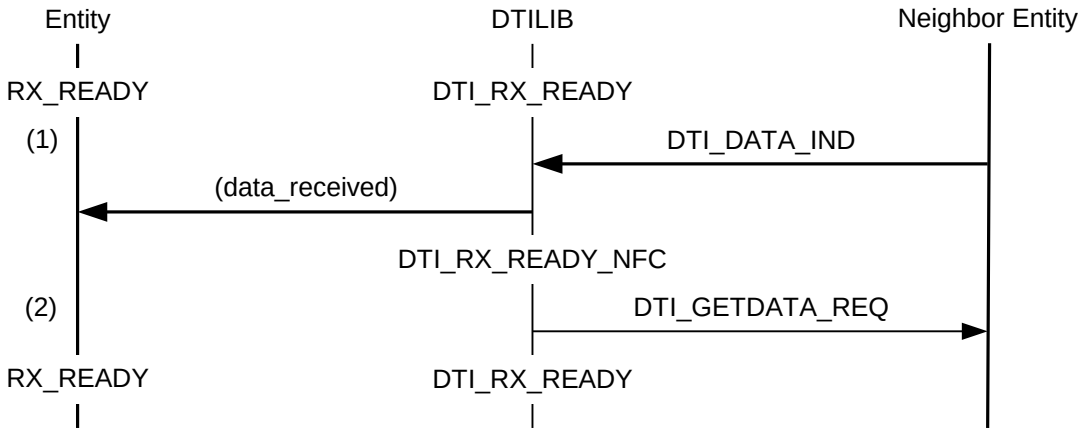
3.4.4 Start reception and Flow Control primitive is already sent



Description:

(1) DTILIB is in DTI_RX_STOPPED state and the entity is in RX_IDLE state. The entity calls the function dti_start() and DTILIB does not need to send a Flow Control primitive because it is already sent. The entity enters RX_READY state and DTILIB enters DTI_RX_READY state.

3.4.5 Receive data

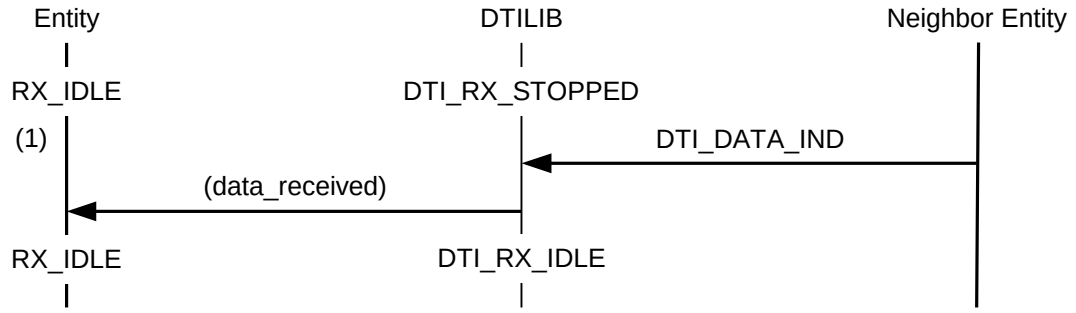


Description:

(1) DTILIB is in DTI_RX_READY state and the entity is in RX_READY state. DTILIB receives a Data primitive and calls the callback function sig_callback() with the reason parameter (data_received) to pass the received data to the entity. DTILIB enters intermediate state DTI_RX_READY_NFC.

(2) If the entity does not call dti_stop() function within the function call of sig_callback(), then DTILIB sends automatically a new Flow Control primitive. The entity stays in RX_READY state and DTILIB enters DTI_RX_READY state again.

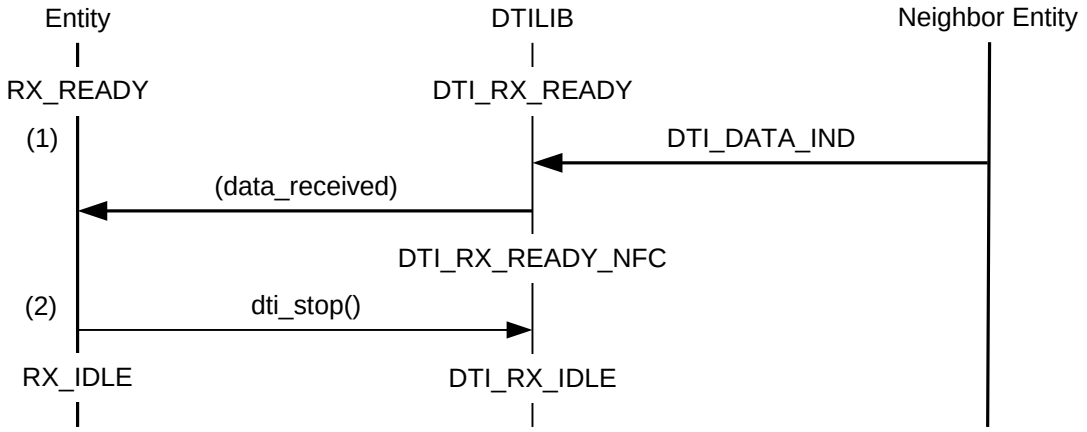
3.4.6 Receive data after reception is stopped



Description:

(1) DTILIB is in DTI_RX_STOPPED state and the entity is in RX_IDLE state. DTILIB receives a Data primitive which is a response to a previous sent Flow Control primitive and calls the callback function sig_callback() with the reason parameter (data_received) to pass the received data to the entity. DTILIB enters DTI_RX_IDLE state and the entity stays in RX_IDLE state.

3.4.7 Stop reception

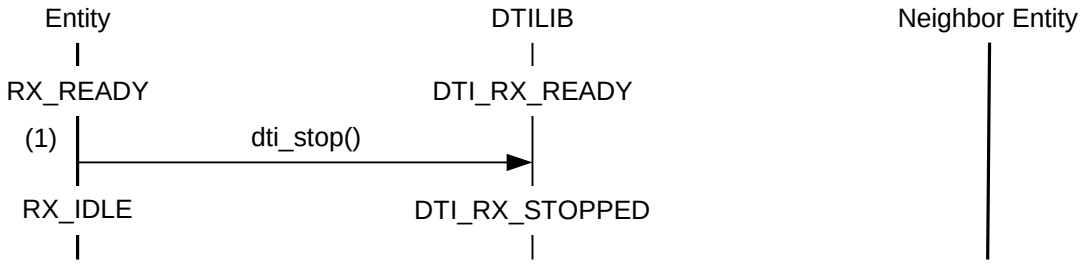


Description:

(1) DTILIB is in DTI_RX_READY state and the entity is in RX_READY state. DTILIB receives a Data primitive and calls the callback function sig_callback() with the reason parameter (data_received) to pass the received data to the entity. DTILIB enters intermediate state DTI_RX_READY_NFC.

(2) If the entity calls dti_stop() function within the function call of sig_callback() DTILIB does not send a new Flow Control primitive and enters DTI_RX_IDLE state. The entity enters RX_IDLE state.

3.4.8 Stop reception but Flow Control primitive was sent

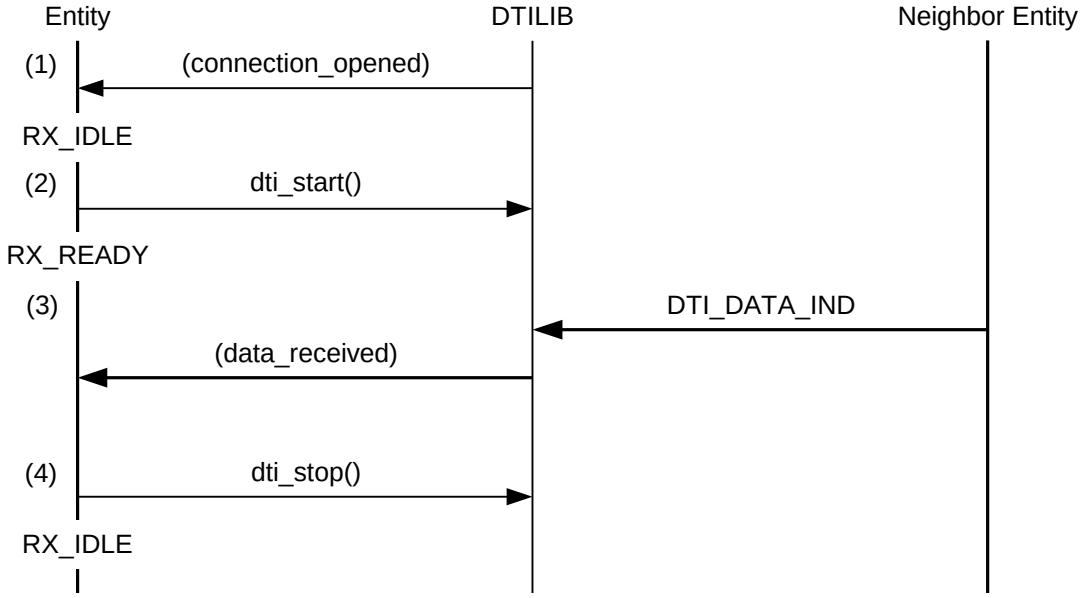


Description:

(1) DTILIB is in DTI_RX_READY state and the entity is in RX_READY state. The entity calls dti_stop() function but a Flow Control primitive was already sent. DTILIB enters DTI_RX_STOPPED state and the entity enters RX_IDLE state.

3.4.9 Receive data without Flow Control (for testing)

To simplify testcases it is possible to run DTILIB in a mode where it does not send or expect Flow Control primitives. **This mode is only useful for testing on PC. It is not possible to use this mode on target.** To enable this mode the value *link_options* in the dti_open() function call must be set to DTI_FLOW_CNTRL_DISABLED. DTILIB do not need to handle states in this mode.



Description:

(1) If DTILIB calls the callback function sig_callback() with the reason parameter (connection_opened) then the entity's receive state machine must enter RX_IDLE state.

(2) The entity calls the function dti_start(), but DTILIB does not send a Flow Control primitive. Actually this function call has no effect to the DTILIB in this mode. The entity enters RX_READY state.

(3) DTILIB receives a Data primitive and calls the callback function sig_callback() with the reason parameter (data_received) to pass the received data to the entity.

(4) The entity calls dti_stop() function to stopt data reception. Actually this function call has no effect to the DTILIB in this mode. The entity enters RX_IDLE state.

3.5 Disconnection

The following description belongs to DTI version 2. See chapter 3.6 for the disconnection with DTI version 1.
Only one side needs to send a Disconnect primitive to close the connection. If one side receives a Disconnect primitive it just closes the connection internally without any primitive response.

3.5.1 States

The states for communication are:

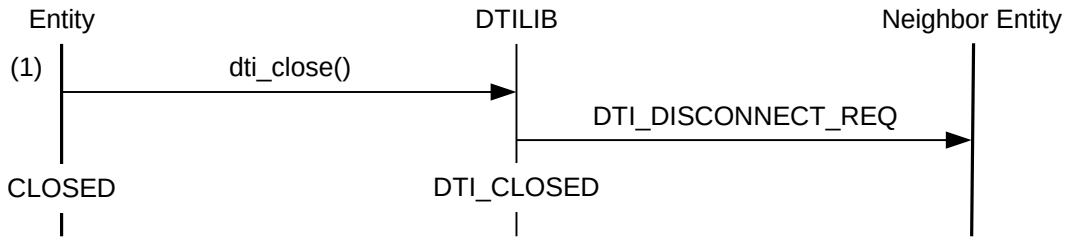
Entity:

CLOSED	- DTI connection is not established any more
CLOSING	- DTI connection is to be closed after all data primitives have been sent

DTILIB:

DTI_CLOSING	- DTI connection is to be closed after all data primitives have been sent
DTI_CLOSED	- DTI connection is not established any more

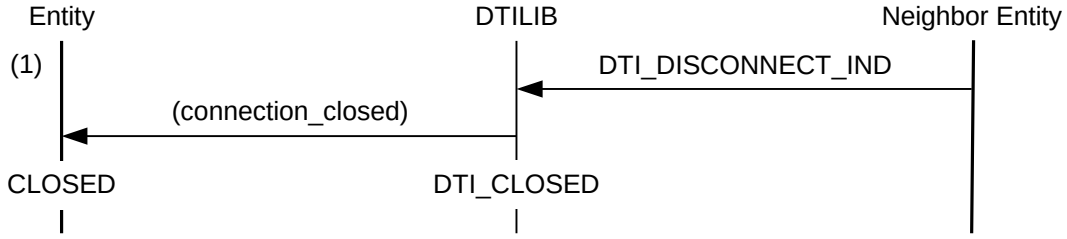
3.5.2 Initiated by the entity



Description:

(1) DTILIB is in any state except DTI_CLOSED state and the entity is in any state except CLOSED state. The entity calls the function dti_close(), DTILIB sends a Disconnect primitive and enters DTI_CLOSED state. The entity enters CLOSED state. DTILIB removes the entry in its internal look up table.

3.5.3 Initiated by the neighbor entity

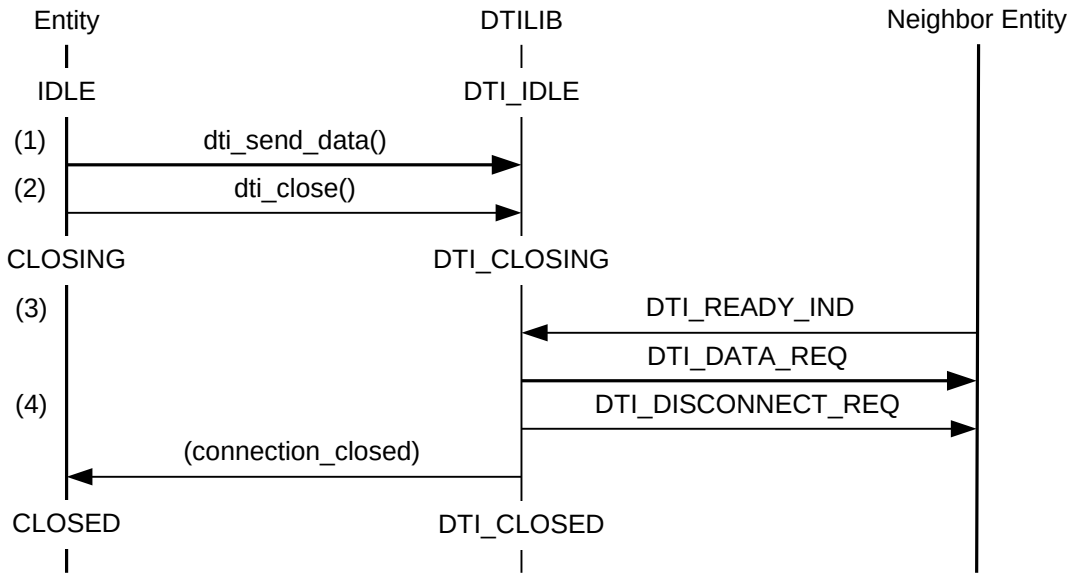


Description:

(1) DTILIB is in any state except DTI_CLOSED state and the entity is in any state except CLOSED state. DTILIB receives a Disconnect primitive, calls the callback function sig_callback() with the reason parameter (connection_closed), and enters DTI_CLOSED state. If the callback function is called with this parameter, the entity enters CLOSED state. DTILIB removes the entry in its internal look up table.

3.5.4 Initiated by the entity, with data to be flushed from the send-queue

If the send-queue of DTILIB is used, dti_close() can be made to close the dti connection only after the queue has been emptied. In this case, the callback function is called with the reason parameter (connection_closed) as soon as the queue is empty and the connection closed down.



Description:

(1) The send-queue is filled with one Data primitive from the entity.

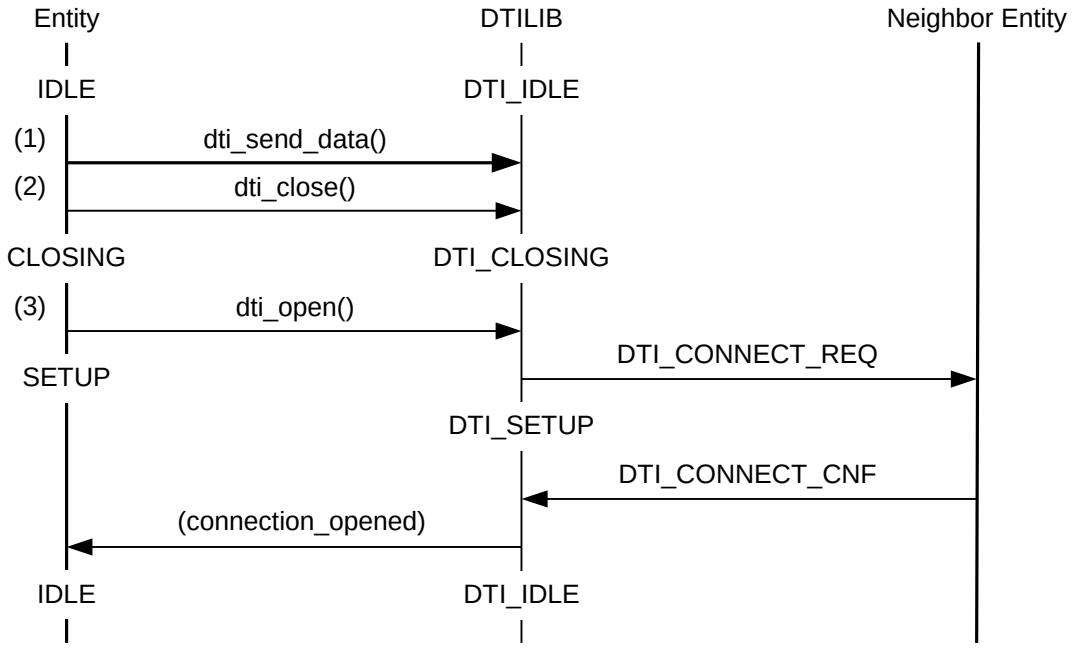
(2) The entity calls the function dti_close(), DTILIB enters the state DTI_CLOSING and waits to send the queued Data primitive to the peer entity.

(3) The peer entity sends a flow control primitive, which allows DTILIB to send the Data primitive and thus empty the send-queue.

(4) DTILIB sends a Disconnect primitive and calls the callback function sig_callback() with the reason parameter (connection_closed). The entity enters CLOSED state and DTILIB removes the entry in its internal look up table.

3.5.5 Initiated with data to be flushed, and cancelled by reset from the entity

If the send-queue of DTILIB is used, dti_close() can be made to close the dti connection only after the queue has been emptied. If the entity calls dti_open() between the call of dti_close() and the actual closing, the connection is being reset but not closed.



Description:

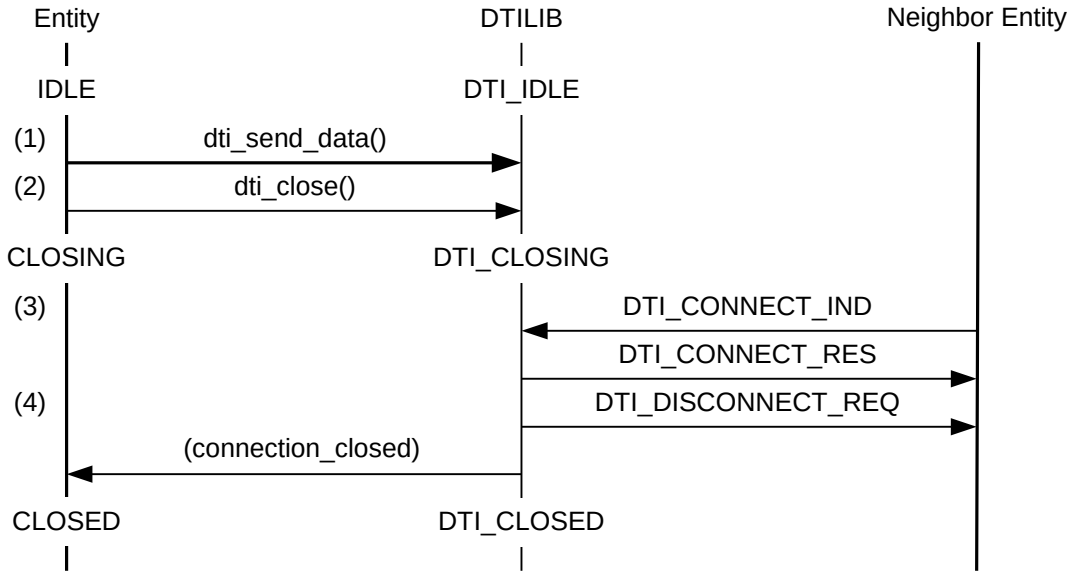
(1) The send-queue is filled with one Data primitive from the entity.

(2) The entity calls the function dti_close(), DTILIB enters state DTI_CLOSING and waits to send the queued Data primitive to the peer entity.

(3) The entity calls the function dti_open() in order to reset the connection. DTILIB enters the state DTI_SETUP. The send-queue is cleared. The reset is signalled to the peer entity by sending a Connect primitive. When DTILIB receives a Connect Confirm primitive it calls the callback function sig_callback() with the reason parameter (connection_opened) and enters DTI_IDLE state. If the callback function is called with this parameter the entity enters IDLE state.

3.5.6 Initiated with data to be flushed, and closed after reset from the peer entity

If the send-queue of DTILIB is used, dti_close() can be used to close the dti connection only after the queue has been emptied. In this case, the callback function is called with the reason parameter (connection_closed) after the queue has been emptied by a request for reset from the peer entity. After sending of a confirmation for the reset the connection is closed down.



Description:

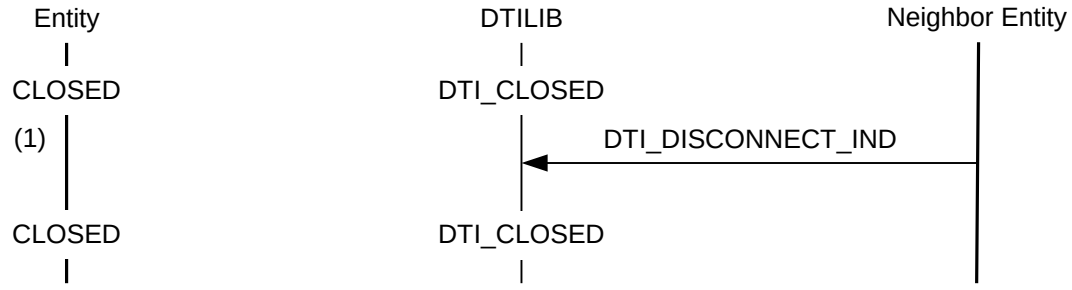
(1) The send-queue is filled with one Data primitive from the entity.

(2) The entity calls the function dti_close(), DTILIB enters state DTI_CLOSING and waits to send the queued Data primitive to the peer entity.

(3) The peer entity sends a Connect primitive in order to reset the connection. DTILIB clears its buffers and sends a Connect Confirm primitive.

(4) DTILIB sends a Disconnect primitive and calls the callback function sig_callback() with the reason parameter (connection_closed). The entity enters CLOSED state and DTILIB removes the entry in its internal look up table.

3.5.7 Disconnect an already closed connection



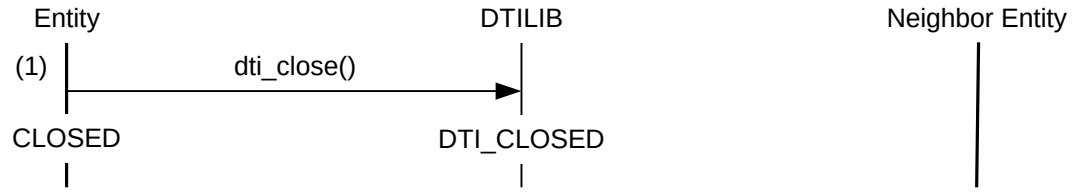
Description:

(1) DTILIB is in DTI_CLOSED state and the entity is in CLOSED state. DTILIB receives a Disconnect primitive and discards it because the connection is already disconnected.

3.6 Disconnection with DTI version 1

DTI version 1 SAP does not support the Disonnect primitives of DTI version 2. That means, it is not possible to inform the DTI peer entity about the disconnection. So DTILIB just removes the entry in its internal look up table.

3.6.1 Close connection

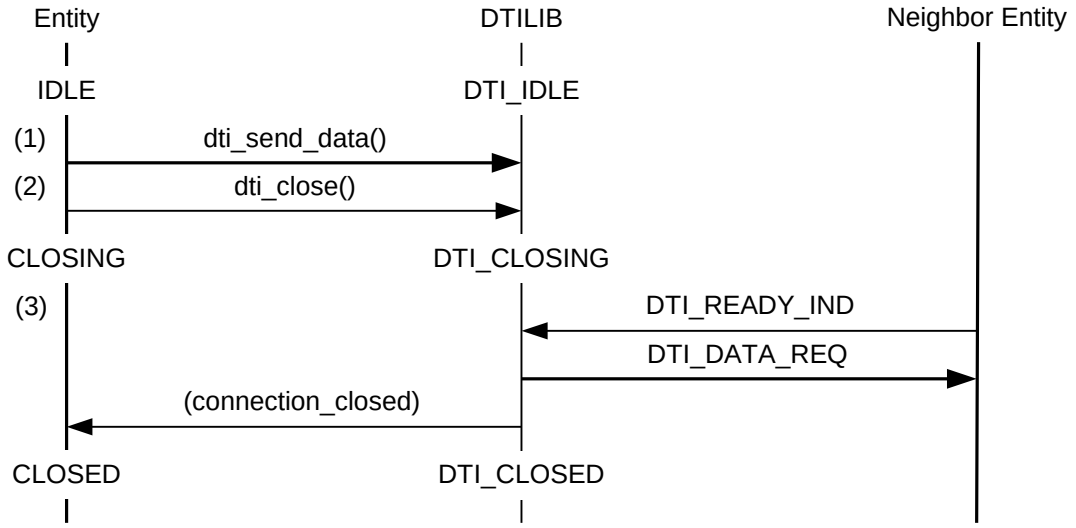


Description:

(1) DTILIB is in any state except DTI_CLOSED state and the entity is in any state except CLOSED state. The entity calls the function dti_close(), DTILIB enters DTI_CLOSED state. The entity enters CLOSED state. DTILIB removes the entry in its internal look up table.

3.6.2 Close connection with flushing

If the send-queue of DTILIB is used, dti_close() can be made to close the dti connection only after the queue has been emptied. In this case, the callback function is called with the reason parameter (connection_closed) as soon as the queue is empty and the connection closed down.



Description:

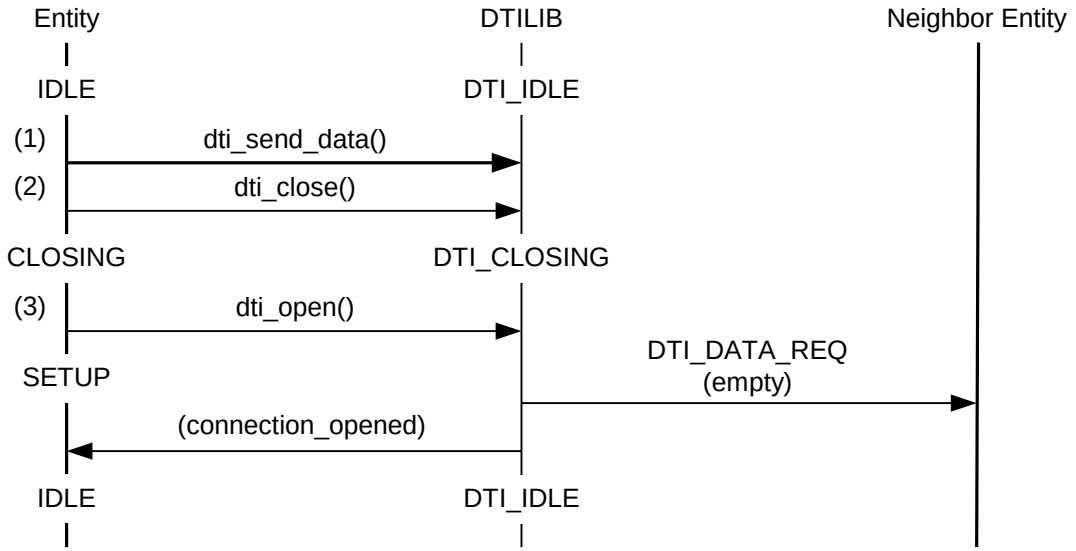
(1) The send-queue is filled with one Data primitive from the entity.

(2) The entity calls the function dti_close(), DTILIB enters the state DTI_CLOSING and waits to send the queued Data primitive to the peer entity.

(3) The peer entity sends a flow control primitive, which allows DTILIB to send the Data primitive and thus empty the send-queue. DTILIB calls the callback function sig_callback() with the reason parameter (connection_closed). The entity enters CLOSED state and DTILIB removes the entry in its internal look up table.

3.6.3 Close connection with flushing and cancelled by reset from the entity

If the send-queue of DTILIB is used, dti_close() can be made to close the dti connection only after the queue has been emptied. If the entity calls dti_open() between the call of dti_close() and the actual closing, the connection is being reset but not closed.



Description:

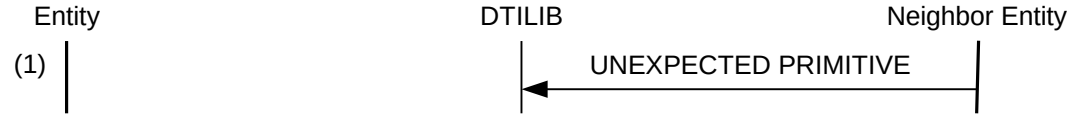
(1) The send-queue is filled with one Data primitive from the entity.

(2) The entity calls the function dti_close(), DTILIB enters state DTI_CLOSING and waits to send the queued Data primitive to the peer entity.

(3) The entity calls the function dti_open() in order to reset the connection. The send-queue is cleared. DTILIB sends an empty synchronization primitive, calls the callback function sig_callback() with the reason parameter (connection_opened) and enters DTI_IDLE state. If the callback function is called with this parameter the entity enters IDLE state.

3.7 Error handling

3.7.1 Unexpected primitives



Description:

(1) If DTILIB receives an unexpected primitive then the primitive will be silently discarded.

4 DTILIB Functions

4.1 dti_init

Prototype:

DTI_HANDLE dti_init(U8 *maximum_links*,
T_HANDLE *entity_handle*,
U32 *entity_options*,
void *(sig_callback(*
U8 instance,
U8 interfac,
U8 channel,
U8 reason,
*T_DTI_DATA_IND *dti_data_ind*
)
)
);

Parameters:

maximum_links	maximum amount of DTI connections	
entity_handle	handle of the entity received by pei_init()	
entity_options	DTI_DEFAULT_OPTIONS	no options selected
	DTI_NO_TRACE	no DTILIB traces
sig_callback	pointer to the entity callback function	

Return:

If the function succeeds, the return value is nonzero and must be used in each other DTI function call of the entity.

Description:

This function must be called once in each entity which wants to use DTI. The best place to call this function is the pei_init() function of each entity.

Example:

```
#define ENTITY_MAX_LINKS  4

LOCAL SHORT pei_init (T_HANDLE handle)
{
  TRACE_FUNCTION ("pei_init");

  ...

  /*
   * Initialize entity data (call init function of every service)
   */
  ...
  hDTI = dti_init(ENTITY_MAX_LINKS, handle, DTI_DEFAULT_OPTIONS,
sig_callback);

  ...

  return (PEI_OK);
}
```

4.2 dti_deinit

Prototype:

VOID dti_deinit(DTI_HANDLE *hDTI***);**

Parameters:

hDTI handle to DTI data-base, given by dti_init() function

Return:

This function does not have a return value.

Description:

This function frees all resources of this entity used by the DTILIB. After this function has been called, the DTI_HANDLE of the entity is no longer valid and DTI primitives can not be sent or received any longer. The best place to call this function is the pei_exit() function of each entity.

Example:

```
LOCAL SHORT pei_exit (void)
{
  TRACE_FUNCTION ("pei_exit");

  /*
   * Close communication channel
   */
  ...

  dti_deinit(hDTI);

  return PEI_OK;
}
```

4.3 dti_open

Prototype:

U8 dti_open(DTI_HANDLE *hDTI***,**

U8 *instance***,**
U8 *interfac***,**
U8 *channel***,**
U8 *queue_size***,**
U8 *direction***,**
U32 *link_options***,**
U32 *version***,**
U8 ***neighbor_entity*,**
U32 *link_id***);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function	
instance	instance of the entity	
interfac	DTI interface within the instance	
channel	channel number	
queue_size	size of send-queue for this connection	
direction	DTI_CHANNEL_TO_HIGHER_LAYER	send Indication primitives
	DTI_CHANNEL_TO_LOWER_LAYER	send Request primitives
	DTI_NULL_LINK	act as NULL device
link_options	DTI_FLOW_CNTRL_DISABLED	do not send and expect Flow Control primitives
	DTI_QUEUE_UNUSED	do not use send-queue
	DTI_QUEUE_UNBOUNDED	send-queue without size limitation
	DTI_QUEUE_RM_FIFO	remove oldest primitive in case of send-queue overflow
	DTI_QUEUE_RM_LIFO	remove newest primitive in case of send-queue overflow
	DTI_QUEUE_WATERMARK	do not remove primitive in case of send-queue overflow
version	DTI_VERSION_10	version 1.0 of DTILIB
neighbor_entity	name of the neighbor entity to connect with	
link_id	identifier of the DTI connection	

Return:

On success, this function returns non-zero, otherwise zero.

Description:

This function opens or resets a DTI connection. If the connection is successful established, DTI calls the entity callback function with the reason parameter set to DTI_REASON_CONNECTION_OPENED. After that the entity can call the functions dti_start() and dti_send_data(). The initial state of a connection after open or restart is that no Flow Control primitive has been sent.

Example:

```
#define CONNECTION_QUEUE_SIZE  5

  SET_STATE( ENTITY_SERVICE, SETUP );
  dti_open(hDTI, 0, 1, 2,
           CONNECTION_QUEUE_SIZE,
           DTI_CHANNEL_TO_LOWER_LAYER,
           DTI_QUEUE_WATERMARK,
           DTI_VERSION_10,
           PPP_NAME,
           link_id);
```

4.4 dti_close

Prototype:

VOID dti_close(DTI_HANDLE *hDTI*,
U8 *instance*,
U8 *interfac,*
U8 *channel,*
BOOL *flush*);

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
instance	instance of the entity
interfac	DTI interface within the instance
channel	channel number
flush	if true, DTILIB waits until the internal send-queue is empty before closing down the connection

Return:

This function does not have a return value.

Description:

This function closes a DTI connection. If this function was called subsequent calls of dti_start() and dti_send_data() are not allowed to this connection. If the neighbor entity closes the connection first then DTI calls the entity callback function with the reason parameter set to DTI_REASON_CONNECTION_CLOSED. After that the entity does not need to call the dti_close() function.

Example:

```
SET_STATE( ENTITY_SERVICE, CLOSED );
dti_close(hDTI, 0, 1, 1, FALSE);
```

4.5 dti_start

Prototype:

VOID dti_start(DTI_HANDLE *hDTI*, **U8** *instance*, **U8** *interfac,* **U8** *channel*);

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
instance	instance of the entity
interfac	DTI interface within the instance
channel	channel number

Return:

This function does not have a return value.

Description:

If the entity wants to receive Data primitives then it must call this function. DTILIB sends appropriate Flow Control primitives as long as dti_stop() function is called. Upon reception of a Data primitive DTI calls the entity callback funtion with the reason parameter set to DTI_REASON_DATA_RECEIVED.

Example:

```
SET_STATE( ENTITY_SERVICE_RX, RX_READY );
dti_start(hDTI, 0, 1, 1);
```

4.6 dti_stop

Prototype:

VOID dti_stop(DTI_HANDLE *hDTI*, **U8** *instance*, **U8** *interfac,* **U8** *channel*);

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
instance	instance of the entity
interface	DTI interface within the instance
channel	channel number

Return:

This function does not have a return value.

Description:

To stop reception of Data primitives the entity has to call this function. If this function was called the entity must still be able to receive one Data primitive because a Flow Control primitive may be sent already.

Example:

```
SET_STATE( ENTITY_SERVICE_RX, RX_IDLE );
dti_stop(hDTI, 0, 1, 1);
```

4.7 dti_send_data

Prototype:

VOID dti_send_data(DTI_HANDLE *hDTI*, **U8** *instance*, **U8** *interfac*, **U8** *channel*, **T_DTI_DATA_IND** ***dti_data_ind*);

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
instance	instance of the entity
interfac	DTI interface within the instance
channel	channel number
dti_data_ind	data primitive

Return:

This function does not have a return value.

Description:

This function is used to send Data primitives to the neighbor entity. DTI manages a send-queue. Upon reception of appropriate Flow Control primitives DTI sends the given Data primitives to the neighbor entity. If the amount of Data primitives has reached the queue-size, given in the dti_open() function, then DTILIB calls the entity callback funtion with the reason parameter set to DTI_REASON_TX_BUFFER_FULL. If the amount of Data primitives leaves the queue-size, then DTILIB calls the entity callback funtion with the reason parameter set to DTI_REASON_TX_BUFFER_READY.

Example:

```
{
  PALLOC (dti_data_ind, DTI_DATA_IND);
  /*
   * fill in Data primitive
   */
  ...

  dti_send_data(hDTI, 0, 1, 1, dti_data_ind);
}
```

4.8 sig_callback

Prototype:

VOID sig_callback(U8 *instance***, U8** *interfac***, U8** *channel***, U8** *reason***, T_DTI_DATA_IND ****dti_data_ind***);**

Parameters:

instance	instance of the entity, given by dti_open() function	
interfac	DTI interface within the instance, given by dti_open() function	
channel	channel number	
reason	DTI_REASON_CONNECTION_OPENED	DTI connection opened or reset
	DTI_REASON_CONNECTION_CLOSED	DTI connection closed
	DTI_REASON_DATA_RECEIVED	Data primitive received
	DTI_REASON_TX_BUFFER_FULL	DTI send-queue full
	DTI_REASON_TX_BUFFER_READY	DTI send-queue no longer full
dti_data_ind	if the reason parameter is set to DTI_REASON_DATA_RECEIVED this parameter points to DTI_DATA_IND primitive otherwise this parameter is NULL.	

Return:

This function does not have a return value.

Description:

This function is called by DTILIB to inform the entity about state changes and primitive receptions. There is one callback function for each entity. **The entity has to implement this function.**

Example:

```
GLOBAL void sig_callback(U8 instance, U8 interfac, U8 channel,
                         U8 reason, T_DTI_DATA_IND *dti_data_ind)
{
  TRACE_FUNCTION("sig_callback");

  /*
   * select entity instance
   */

  switch (reason)
  {
    case DTI_REASON_CONNECTION_OPENED:
      /*
       * Handle the connection opened signal
       */
      sig_dti_service_connection_opened_ind(interfac, channel);
      break;

    case DTI_REASON_CONNECTION_CLOSED:
      /*
       * Handle the connection opened signal.
       */
      sig_dti_service_connection_closed_ind(interfac, channel);
      break;

    case DTI_REASON_DATA_RECEIVED:
      /*
       * Handle received data packet.
       * - select interface and channel
       */
      sig_dti_service_data_received_ind(dti_data_ind);
      break;

    case DTI_REASON_TX_BUFFER_FULL:
      /*
       * Handle received data packet. The entity must now stop send data.
       * - select interface and channel
       */
      sig_dti_service_buffer_full_ind();
      break;

    case DTI_REASON_TX_BUFFER_READY:
      /*
       *  Handle signal tx_buffer_ready. The entity can now send data.
       * - select interface and channel
       */
      sig_dti_service_buffer_ready_ind();
      break;
  } /* end switch */
} /* sig_callback() */
```

4.9 Primitive process functions

These functions are used to process received DTI primitives. They should be inserted in the jump tables to primitive handler functions in the xxx_pei.c file.

Example:

```
/*
 * For uplink primitives:
 */
LOCAL const T_FUNC dul_table_dti[] = {
  MAK_FUNC_0( pei_dti_dti_connect_req   , DTI2_CONNECT_REQ   ),  /* xx00 */
  MAK_FUNC_0( pei_dti_dti_connect_res   , DTI2_CONNECT_RES   ),  /* xx01 */
  MAK_FUNC_0( pei_dti_dti_disconnect_req, DTI2_DISCONNECT_REQ),  /* xx02 */
  MAK_FUNC_0( pei_dti_dti_getdata_req   , DTI2_GETDATA_REQ   ),  /* xx03 */
  MAK_FUNC_0( pei_dti_dti_data_req      , DTI2_DATA_REQ      )   /* xx04 */
#if defined (WIN32)
  ,
  MAK_FUNC_S( pei_dti_dti_data_test_req , DTI2_DATA_TEST_REQ )   /* xx05 */
#endif
};

/*
 * For downlink primitives:
 */
LOCAL const T_FUNC ddl_table_dti[] = {
  MAK_FUNC_0( pei_dti_dti_connect_ind   ,  DTI2_CONNECT_IND   ), /* yy00 */
  MAK_FUNC_0( pei_dti_dti_connect_cnf   ,  DTI2_CONNECT_CNF   ), /* yy01 */
  MAK_FUNC_0( pei_dti_dti_disconnect_ind,  DTI2_DISCONNECT_IND), /* yy02 */
  MAK_FUNC_0( pei_dti_dti_ready_ind     ,  DTI2_READY_IND     ), /* yy03 */
  MAK_FUNC_0( pei_dti_dti_data_ind      ,  DTI2_DATA_IND      )  /* yy04 */
#if defined (WIN32)
  ,
  MAK_FUNC_S( pei_dti_dti_data_test_ind ,  DTI2_DATA_TEST_IND )  /* yy05 */
#endif
};
```

xx and yy indicate the SAP Operation Code Ids.
The functions of DTILIB have to map into “pei_” functions. It is not possible to insert the DTILIB functions in the table because DTILIB functions need an additional parameter (DTI handle). So the developer must create converter functions that include this parameter. In this example `pei_dti_dti_connect_req()`.

Example:

```
LOCAL void pei_dti_dti_connect_req(T_DTI2_CONNECT_REQ *dti_connect_req)
{
  dti_dti_connect_req(hDTI, dti_connect_req);
}
```

4.9.1 dti_dti_connect_req

Prototype:

VOID dti_dti_connect_req(DTI_HANDLE *hDTI*, **T_DTI_CONNECT_REQ ***_dti_connect_req_**);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_connect_req	Connect primitive

Return:

This function does not have a return value.

Description:

If the entity receives a DTI_CONNECT_REQ primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.2 dti_dti_connect_ind

Prototype:

VOID dti_dti_connect_ind(DTI_HANDLE *hDTI*, **T_DTI_CONNECT_IND ***dti_connect_ind**);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_connect_ind	Connect primitive

Return:

This function does not have a return value.

Description:

If the entity receives a DTI_CONNECT_IND primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.3 dti_dti_connect_cnf

Prototype:

VOID dti_dti_connect_cnf(DTI_HANDLE *hDTI*, **T_DTI_CONNECT_CNF ***dti_connect_cnf**);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_connect_cnf	Connect Confirm primitive

Return:

This function does not have a return value.

Description:

If the entity receives a DTI_CONNECT_CNF primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.4 dti_dti_connect_res

Prototype:

VOID dti_dti_connect_res(DTI_HANDLE *hDTI*, **T_DTI_CONNECT_RES ***dti_connect_res**);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_connect_res	Connect Confirm primitive

Return:

This function does not have a return value.

Description:

If the entity receives a DTI_CONNECT_RES primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.5 dti_dti_disconnect_req

Prototype:

VOID dti_dti_disconnect_req(DTI_HANDLE *hDTI*, **T_DTI_DISCONNECT_REQ ***dti_disconnect_req**);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_disconnect_req	Disconnect primitive

Return:

This function does not have a return value.

Description:

If the entity receives a DTI_DISCONNECT_REQ primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.6 dti_dti_disconnect_ind

Prototype:

VOID dti_dti_disconnect_ind(DTI_HANDLE *hDTI*, **T_DTI_DISCONNECT_IND ***dti_disconnect_ind**);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_disconnect_ind	Disconnect primitive

Return:

This function does not have a return value.

Description:

If the entity receives a DTI_DISCONNECT_IND primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.7 dti_dti_getdata_req

Prototype:

VOID dti_dti_getdata_req(DTI_HANDLE *hDTI*, **T_DTI_GETDATA_REQ ***dti_getdata_req**);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_getdata_req	Flow Control primitive

Return:

This function does not have a return value.

Description:

If the entity receives a DTI_GETDATA_REQ primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.8 dti_dti_ready_ind

Prototype:

VOID dti_dti_ready_ind(DTI_HANDLE *hDTI*, **T_DTI_READY_IND ***dti_ready_ind**);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_ready_ind	Flow Control primitive

Return:

This function does not have not a return value.

Description:

If the entity receives a DTI_READY_IND primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.9 dti_dti_data_ind

Prototype:

VOID dti_dti_data_ind(DTI_HANDLE *hDTI*, **T_DTI_DATA_IND ***dti_data_ind**);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_data_ind	Data primitive

Return:

This function does not have a return value.

Description:

If the entity receives a DTI_DATA_IND primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.10 dti_dti_data_req

Prototype:

VOID dti_dti_data_req(DTI_HANDLE *hDTI*, **T_DTI_DATA_REQ ***dti_data_req**);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_data_req	Data primitive

Return:

This function does not have a return value.

Description:

If the entity receives a DTI_DATA_REQ primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.11 dti_dti_data_test_req

Prototype:

VOID dti_dti_data_test_req(DTI_HANDLE *hDTI***, T_DTI_DATA_TEST_REQ *** *dti_data_test_req***);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_data_test_req	Data test primitive

Return:

This function does not have a return value.

Description:

If the entity receives a DTI_DATA_TEST_REQ primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

4.9.12 dti_dti_data_test_ind

Prototype:

VOID dti_dti_data_test_ind(DTI_HANDLE *hDTI***, T_DTI_DATA_TEST_IND *** *dti_data_test_ind***);**

Parameters:

hDTI	handle to DTI data-base, given by dti_init() function
dti_data_test_ind	Data test primitive

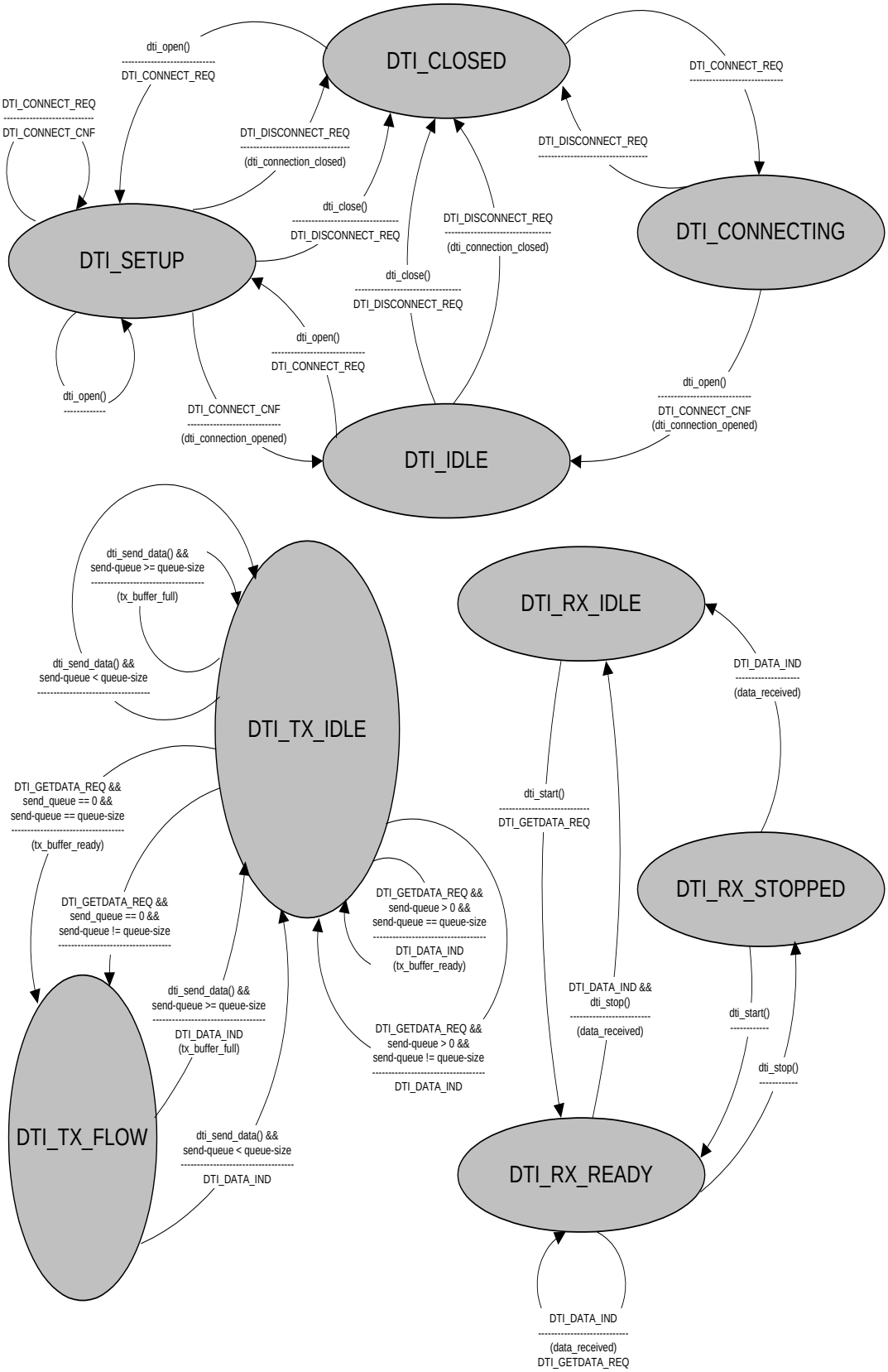
Return:

This function does not have a return value.

Description:

If the entity receives a DTI_DATA_TEST_IND primitive this function must be called. The best place for this function call is the pei_primitive() function of the entity.

6 State Transition



5 Data Base

The entities do not always communicate with only one neighbor. There can be several neighboring entities and more than one DTI connection to each neighboring entity. For example IP can have UDP and TCP in higher layer and PPP and SNDCP in lower layer. SNDCP communicates over the air interface and PPP with a local computer over the infrared interface. In this case it must be possible to route the primitives to the destination.
To achieve this DTILIB holds a Data Base for each entity. Only DTILIB has access to this Data Base. The memory for the Data Base will be allocated by the dti_init() function.
The routing information has ACI. It sends this information by configure primitives to the entity. Such configuration primitives are individual for each entity and outside of the scope of this document. The entity passes the routing information to DTILIB by call dti_open() function.

5.1 Entity Structure

DTILIB creates one structure for each entity. The function dti_init() allocates necessary memory for this structure and initializes it. The return value of dti_init() is a pointer to this structure called DTI_HANDLE. The DTI_HANDLE must be used in subsequent calls of other DTI function.

Definition:

```
typedef struct
{
  T_HANDLE entity_handle;            /* Handle from the entity */
  U8 max_links;                      /* Number of links in the link list
                                   - started from the pointer first_link
*/
  U32 entity_options;                 /* specific entity options */
  void ((*sig_callback)               /* Callback function */
    (U8 instance,                     /* Instance of the entity */
     U8 interfac,                     /* Interface of the entity */
     U8 channel,                      /* Channel */
     U8 reason,                       /* Reason for the callback */
     T_DTI2_DATA_IND *dti_data_ind)); /* Data primitive ul */
  U32 first_link;                     /* Pointer to link table */
} DTI_DATA_BASE;
```

Elements:

entity_handle **Entity handle**
DTILIB use thes parameter for traces, example TRACE_FUNCTION().

max_links **Maximum number of DTI connections**
This value determines the maximum number of DTI connections of the whole entity. That means all connections of all instances of the entity.

entity_options **Entity related options**
This value can be used to turn DTILIB traces off.

sig_callback **Pointer to the entity callback function**
The callback function is used to inform the entity about state transitions and Data primitive receptions. There is just one callback function for the whole entity. The callback function must be implemented by the developer of the entity.

first_link **Pointer to the first table of link structures**
For each DTI connection DTILIB uses a link structure. This parameter is a table with all needed link information for the entity.

5.2 Link Structure

DTILIB uses the following structure for each DTI connection. This structure is addressed over the pointer *first_link* of the entity structure, (DTI_DATA_BASE). It will be initialized by dti_open() function call.

Definition:

```
typedef struct
{
  U32             link_id;        /* Identity for the link communication */
  U8              direction;      /* Direction for the link communication
*/
  U32             version;        /* DTI Version */
  U8              instance;       /* Instance of the entity */
  U8              interfac;       /* Selected interface */
  U8              channel;        /* Channel number */
  U32             link_options;   /* Type of flow control and queueing*/
  T_HANDLE        link_handle;    /* Handle for the communication channel
*/
  U8              queue_size;     /* DTI queue size */
  U8              queue_len;      /* length of the queue */
  T_DTI2_DATA_IND *dti_data_ind;  /* DTI data primitive */
  U8              connect_state;  /* State for connect */
  U8              rx_state;       /* State for receive */
  U8              tx_state;       /* State for send */
  U32             next_link;      /* Pointer to next DTI_LINK struct, last
                                  /* linkpointer = NULL */
} DTI_LINK;
```

Elements:

link_id **Identifier of the DTI connection**
Each DTI connection has a unique identifier. This identifier is included in each DTI primitive. On transmission of Data primitives DTILIB converts the values *instance*, *interfac* and *channel* to *link_id*. On reception of Data primitives DTILIB converts the *link_id* to *instance*, *interfac* and *channel*.

direction **Direction of the link**
This parameter determines if DTILIB must send Request primitives (uplink) or Indication primitives (downlink) or this is a NULL link.

version **Version number**
This parameter describes the version number of DTILIB which the entity uses. By connecting -DTILIB selects the correct version.

instance **Instance of the entity**
Together with the values *interfac* and *channel* this value is used to identify the DTI connection within the entity. Usually this value determines the instance of the entity.

interfac **DTI interface within the instance**
Together with the values *instance* and *channel* this value is used to identify the DTI connection within the entity. Usually this value determines the DTI interface within one instance of the entity.

channel **DTI channel within the instance**
Together with the values *instance* and *interface* this value is used to identify the DTI connection within the entity. Usually this value determines the DTI channel within one instance and interface of the entity.

link_options **Type of flow control and queueing**
This value determines the behavior of DTILIB according flow control and queueing. Refer to the description of dti_open() to get the possible values.

link_handle **VSI communication channel**
To send primitives to an entity a VSI communication channel must be opened. DTILIB uses the parameter *neighbor_entity* of the dti_open() function to open such communication channel.

queue_size **Size of send-queue**
This value is given by the dti_open() function. It specifies the number of Data primitives which can be stored in the send-queue of this connection.

queue_len **Number of Data primitives in send-queue**
In this value DTILIB stores the current number of Data primitives in the send-queue.

dti_data_ind **Data primitives in send-queue**
DTILIB stores the Data primitives in a linked list. This parameter points to the next Data primitive to send. In this Data primitive the value *link_id* is used to point to the next Data primitive. In the last Data primitive of the list the value *link_id* is set to NULL.

connect_state **Connect state**
In this variable DTILIB stores the states of the connection state machine.

rx_state **Receiving state**
In this variable DTILIB stores the states of the receiving state machine.

tx_state **Sending state**
In this variable DTILIB stores the states of the sending state machine.

next_link **Pointer to next DTI_LINK structure**
This is a pointer to the next connected DTI_LINK structure. The last pointer is NULL.

6 Integration of DTILIB

6.1 Names of SAP documents and source files

The source files of DTILIB are placed in the same directory as the entity source files.

The names of the source files are:

dti.h Include File with constants and definition of the DTI databank and Function Prototypes. This file must be included in the source file, which calls the DTILIB functions.

dti_int_def.h Include Files containing definitions only for DTIIB.

dti_int_prot.h Include Files containing prototypes only for DTIIB.

dti_kerf.c This file contains library functions. An example is the function dti_open() that sets up the link structure.

dti_kerp.c	This file contains functions to handle primitive communication. One example is dti_dti_data_req().
dti.mak	This is the makefile for the Target Protocol Stack.

Names of project file:

dti.dsp	This is the project file in the GSM Protocol Stack.

Names of SAP files:

dti.doc	SAP document for old DTI primitives.
dti2.doc	SAP document for new DTI primitives.

6.2 Compilation of DTILIB

DTILIB is a static library which is compiled on the target with the makefile dti.mak. To call the makefile for DTILIB and the other entities for the target, use the script g23.pl.

The library can handle two SAPs. DTI SAP version 1 and the DTI SAP version 2. DTILIB can not handle both SAPs at the same time.

To select DTI SAP version 2 use the compiler switch:

DTI2

To use the old DTI SAP version 1 do not activate this switch.

6.3 Test cases with DTILIB

DTILIB supports conversion of data primitives into data test primitives. It also supports the process functions for the test primitives. For example the primitive DTI2_DATA_REQ is converted into DTI2_DATA_TEST_REQ and sent. By receiving the primitive the related process function in DTILIB, dti_dti_data_test_req(), it is being converted back to DTI2_DATA_REQ.

Please note that the function dti_dti_data_test_req() must be called in the related Pei Process Table of xxx_pei.c.

To switch the DTILIB to handle test cases, the following compiler option must be used:

SIMULATION

6.4 Rules for integration of DTILIB in old entities

Most of the entities in the GSM protocol stack must be changed to handle primitive communication with DTILIB. Some of the entities are IP, UDP, PPP and L2R. To get similarity of the entities with easier bug correction possibility and to make it easier for the developer, there are some rules that should be followed:

1. In DTI SAP version 2 the primitives, desc_list and desc have new names. Examples are:

   dti_data_req -> dti2_data_req
   desc_list -> desc_list2
   desc -> desc2

These are names which are used very often in the entities – so if a combination of old and new SAP is needed – the definitions shall be used for naming and selecting by a compiler switch. Example:

```
#if defined(DTILIB)

/*
 * Use of DTILIB
 */

#include "dti.h"

…Other defines for DTILIB, as use of old DTI SAP, can be done here

#else

/*
 * No use of DTILIB – renaming with defines
*/

#define T_DTI2_DATA_IND      T_DTI_DATA_IND
#define T_DTI2_DATA_REQ      T_DTI_DATA_REQ
#define T_DTI2_READY_IND     T_DTI_READY_IND
#define T_DTI2_GETDATA_REQ   T_DTI_GETDATA_REQ
#define T_DTI2_DATA_TEST_REQ T_DTI_DATA_TEST_REQ
#define T_DTI2_DATA_TEST_IND T_DTI_DATA_TEST_IND

#define T_desc2 T_desc
#define desc_list2 desc_list
#define T_desc_list2 T_desc_list

#endif
```

The definitions should be placed in an Include File which reaches all of the source files of the entity. Normally there is a header file for each entity which should be used.

This means: a) Rename all the changed primitives, descs, and desc_lists.
b) Put the “rename defines” into a header file as described above.

2. To change or add code in old functions, is done by the compiler switch DTILIB.

   Example:

```
#if defined(DTILIB)
        …source code only with DTILIB
#else
        …source code not with DTILIB
#endif
```

Please make the needed changes in the functions and set up new functions only if many changes must be done.

3. By developing new functions which use DTILIB, but have related functions without DTILIB, be sure to make note of this. Example:

```
#if defined(DTILIB)
    dti2_data_req()
    {
            /*
             * By correcting bugs also check the function dti_data_req()
             */
            ....code for DTILIB

    }
#else
dti_data_req()
{
            /*
             * By correcting bugs also check the function dti2_data_req()
             */
}
#endif /*dti2_data_req(), dti_data_req() */
```

4. The table for the functions that handle the primitive calls in xxx_pei.c shall be split into two tables. Example :

```
#if defined (DTILIB)
LOCAL const T_FUNC ip_table_dti[] = {
  MAK_FUNC_0( pei_ip_addr_req_dti ,  IP_ADDR_REQ_DTI    )
};
#else
LOCAL const T_FUNC ip_table[] = {
  MAK_FUNC_0( pei_ip_addr_req     ,  IP_ADDR_REQ       )
};
#endif
```

Please be aware that the last two bytes of the primitive ID must start with zero, see chapter 5.9.

5. The source file names should not be changed. The DTILIB functions are added in the source files related to the old naming.

6. The new added primitives for DTILIB have to be in the old SAP document. For example, Configuration Primitives from ACI.

7. In some cases the entities shall support DTI 1 and DTI 2 SAP with DTILIB. To select the difference in the entity, for example sending and setting of parameters for the old and new primitives, a compiler switch should be used:

DTILIB_WITH_OLD_SAP Compiler switch for the entity to select DTI1 SAP or else DTI 2 SAP is active

8. Be careful with the parameter that describes the links, which is set up by calling the function dti_open() in the entity. Important parameters are :

instance	Some primitives have own process instances. An example is L2R and PPP.
channel	Define a DTI Channel Number inside the entity. This is mostly set with an ACI primitive.
interface	The definition of a DTI Interface. This could be a special interface, like a serial interface, or it can just give information about the channel that is related to higher or lower layers. This means that the developer can group channels for higher layer or for lower layer.

link_id This is the DTI Identification Number for the link structure. By reaching a DTI primitive DTILIB selects the link structure which is used for this parameter. The link_id is sent with the primitive and must be the same in both entities. Likewise, the function dti_open() must be called with the same link_id in both entities.

9. The entity must be able to process even if the Flow Control mechanism is not activated. This means that by receiving a data primitive the data must be processed and sent at once. The only active signals which the callback functions support will be connected and data will be received.

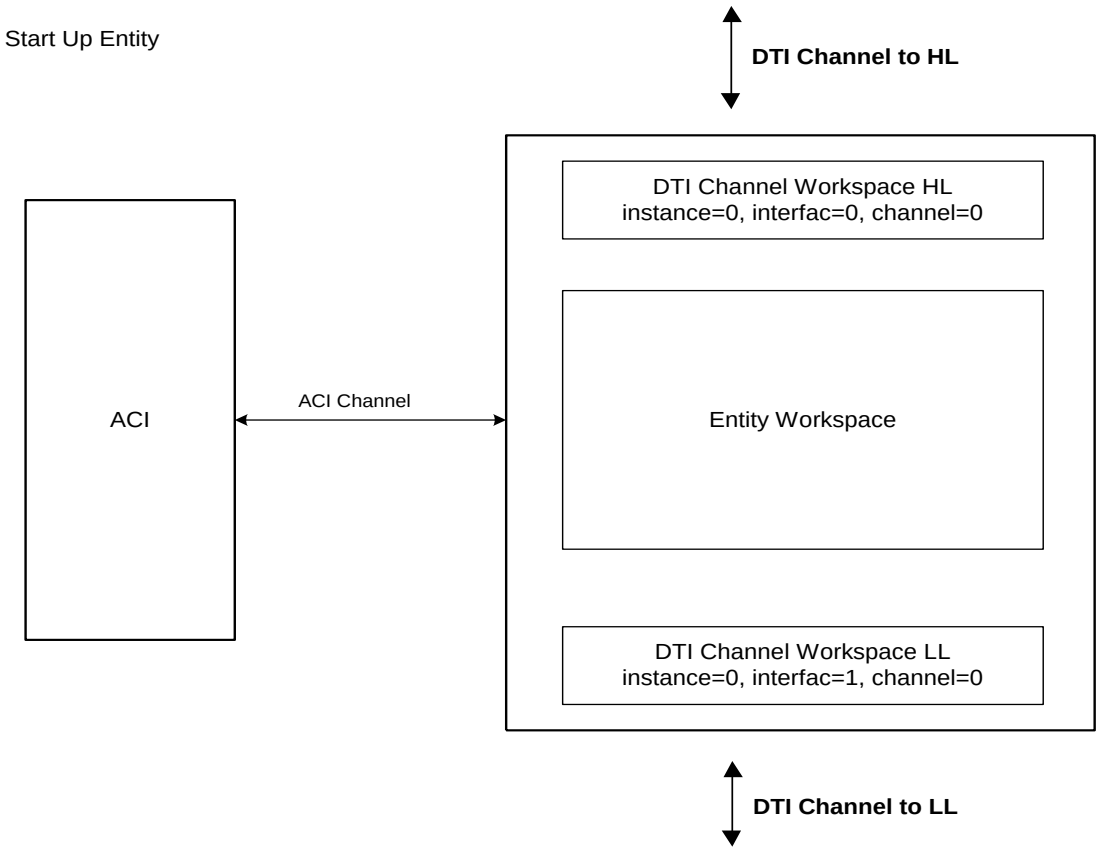
7 Start Up Entity with DTILIB

One main problem with the development of entities is that the finished source code is not uniform. There are global structure differences between the entities, which makes the correction of bugs and care of the entities very difficult. Another problem is that many developers do not communicate and encounter similar problems or difficulties, when working on entities. Problems may be solved many times without progress.
To solve these problems and to make it easier for an unpractised developer to start, this chapter describes a source code skeleton which can be used as a "start up" entity.

7.1 Structure of the entity

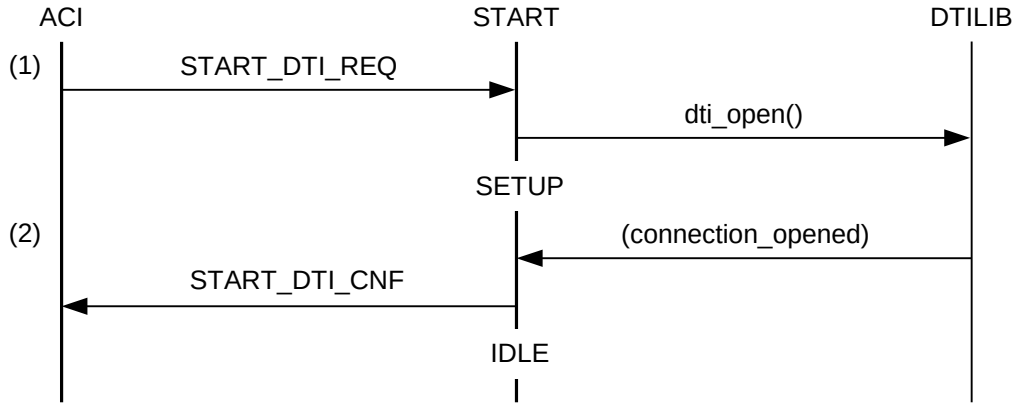
The structure of the skeleton is shown in the next picture. The entity contains one DTI channel against the Higher Layer and one DTI channel against the Lower Layer. It also has a configuration channel to ACI.

This entity can be used as a template for support to the entity developer.



Picture of the Start Up Entity, START.

7.2 Set up a DTI channel



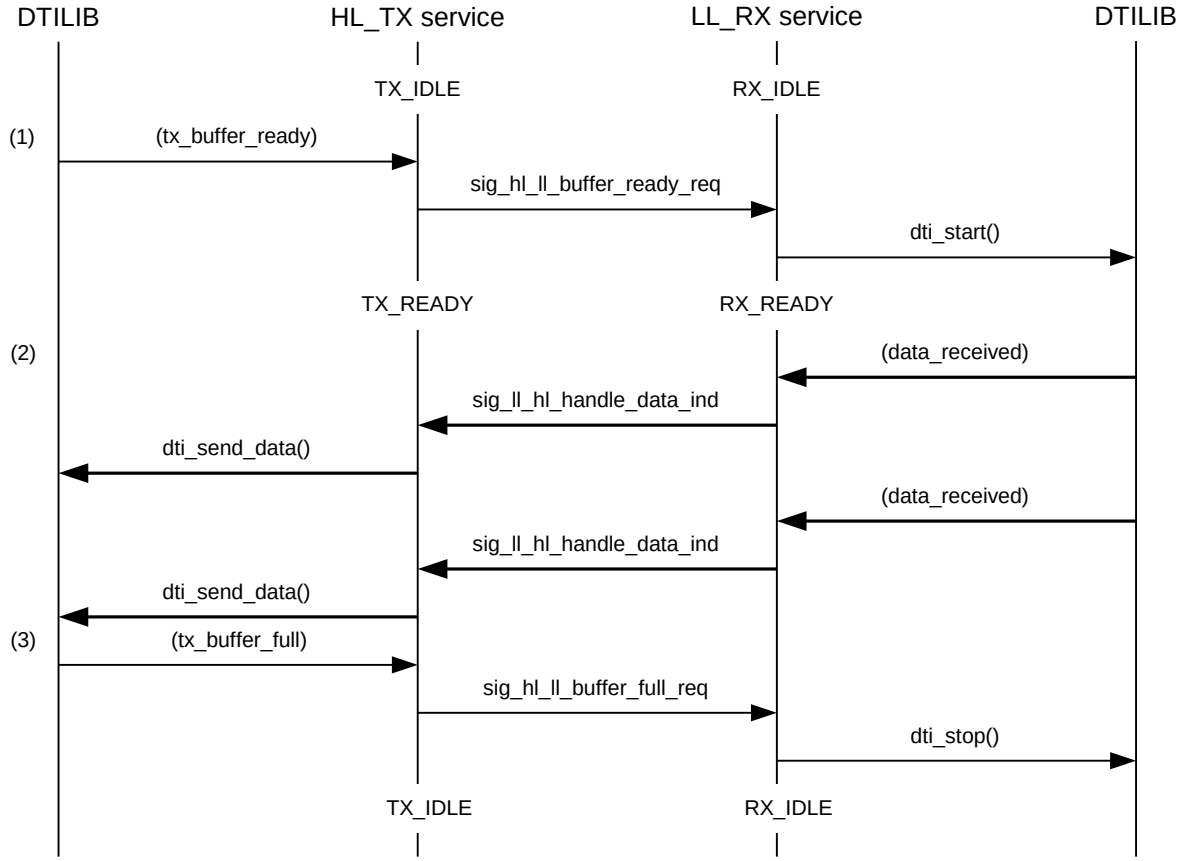
Description:

(1) The DTI service state is CLOSED or IDLE. If it is CLOSED the DTI channel will be set up and if IDLE the DTI channel will be reset. With this primitive the entity gets all parameters needed to set up a DTI channel. A special parameter indicates the request to open a DTI connection. The entity starts connecting with the neighbor entity by calling the DTILIB function dti_open().

(2) START gets the signal (connection_opened). The entity sends a confirm primitive. The entity change DTI states to IDLE.

7.3 DTI channel communication

It is very easy to connect two DTI links within an entity. The following example shows one side of the connection of the HL DTI link and the LL DTI link. If the LL DTI link receives data this data will be manipulated and after that forwarded to the HL DTI link.



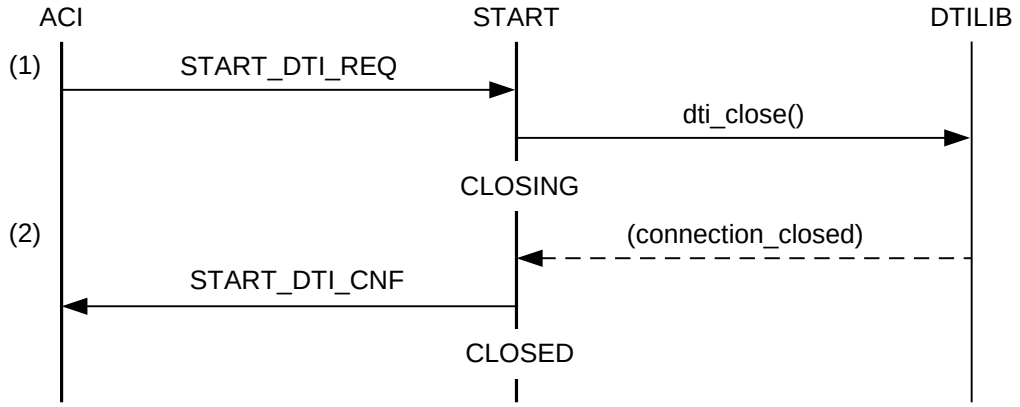
Description:

(1) Both LL_RX and HL_TX service are in IDLE state. DTILIB indicates for the HL connection that data can be sent. This leads to a call of dti_start() for the LL connection.

(2) Incomming data from the LL connection is processed and forwarded to the HL connection. The HL_TX service calls dti_send_data() to send the manipulated data via the HL connection.

(3) DTILIB indicates for the HL connection that is not allowed any longer to send data. This leads to a dti_stop() call for the LL connection.

7.4 ACI initiated DTI disconnection

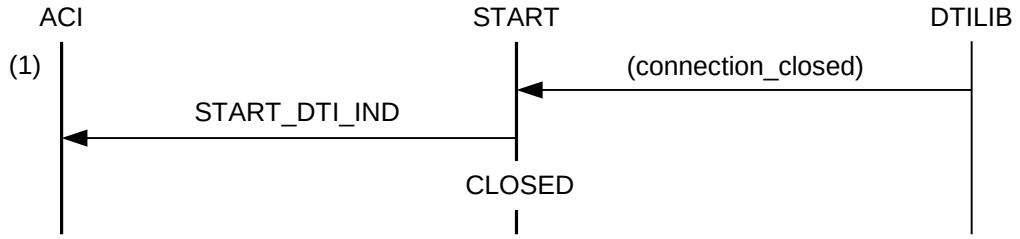


Description:

(1) A special parameter in this primitive indicates the request to close a DTI connection. The entity disconnects the link with the neighbor entity by calling the DTILIB function dti_close().

(2) If the *flush* parameter of the dti_close() call is set to TRUE the entity waits for the (connection_closed) signal from the DTILIB and sends a confirm primitive after reception. If the *flush* parameter of the dti_close() call is set to FALSE the entity send the confirm primitive immediately. The entity changes DTI states to CLOSED.

7.5 Neighbor entity initiated DTI disconnection



Description:

(1) The entity gets the signal (connection_closed) from DTILIB, closes the connection and sends an indication primitive to ACI. The entity changes DTI states to CLOSED.

Appendices

A. Acronyms

ACI	AT Command Interpreter
AGCH	Access Grant Channel
AT	Attention sequence "AT" to indicate valid commands of the ACI
BCCH	Broadcast Control Channel
BCS	Binary Coded Signals
BS	Base Station
BSIC	Base Station Identification Code
C/R	Command/Response
C1	Path Loss Criterion
C2	Reselection Criterion
CBCH	Cell Broadcast Channel
CBQ	Cell Bar Qualify
CC	Call Control
CCCH	Common Control Channel
CCD	Condat Coder Decoder
CKSN	Ciphering Key Sequence Number
CRC	Cyclic Redundancy Check
DCCH	Dedicated Control Channel
DISC	Disconnect Frame
DL	Data Link Layer
DM	Disconnected Mode Frame
DTX	Discontinuous Transmission
EA	Extension Bit Address Field
EL	Extension Bit Length Field
EMMI	Electrical Man Machine Interface
EOL	End Of Line
F	Final Bit
F&D	Fax and Data Protocol Stack
FACCH	Fast Associated Control Channel
FHO	Forced Handover
GP	Guard Period
GSM	Global System for Mobile Communication
HDLC	High level Data Link Control
HISR	High level Interrupt Service Routine
HPLMN	Home Public Land Mobile Network
I	Information Frame
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
ITU	International Telecommunication Union
IWF	Interworking Function
Kc	Authentication Key
L	Length Indicator
LAI	Location Area Information
LISR	Low level Interrupt Service Routine
LPD	Link Protocol Discriminator
M	More Data Bit
MCC	Mobile Country Code
MM	Mobility Management

MMI	Man Machine Interface
MNC	Mobile Network Code
MS	Mobile Station
MSG	Message phase in the GSM 3.45 protocol
N(R)	Receive Number
N(S)	Send Number
NCC	National Colour Code
NECI	New Establishment Causes included
OTD	Observed Time Difference
P	Poll Bit
P/F	Poll/Final Bit
PCH	Paging Channel
PCO	Point of Control and Observation
PDU	Protocol Description Unit
PL	Physical Layer
PLMN	Public Land Mobile Network
RACH	Random Access Channel
REJ	Reject Frame
RNR	Receive Not Ready Frame
RR	Radio Resource Management
RR	Receive Ready Frame
RTD	Real Time Difference
RTOS	Real Time Operating System
SABM	Set Asynchronous Balanced Mode
SACCH	Slow Associated Control Channel
SAP	Service Access Point
SAPI	Service Access Point Identifier
SDCCH	Slow Dedicated Control Channel
SIM	Subscriber Identity Module
SMS	Short Message Service
SMSCB	Short Message Service Cell Broadcast
SS	Supplementary Services
T.4	CCITT Standardisation for Document coding of Group 3 Facsimile Apparatus
TAP	Test Application Program
TCH	Traffic Channel
TCH/F	Traffic Channel Full Rate
TCH/H	Traffic Channel Half Rate
TDMA	Time Division Multiple Access
TE	Terminal Equipment - e. g. a PC
TMSI	Temporary Mobile Subscriber Identity
UA	Unnumbered Acknowledgement Frame
UI	Unnumbered Information Frame
V(A)	Acknowledgement State Variable
V(R)	Receive State Variable
V(S)	Send State Variable
VPLMN	Visiting Public Land Mobile Network
DTI	Data Transmission Interface
DTILIB	Data Transmission Interface Library
DESC	Generic Data Descriptor
DESC-LIST	Linked List of Generic Data Descriptors

B. Terms

Entity:	Program which executes the functions of a layer
Message:	A message is a data unit which is transferred between the entities of the same layer (peer-to-peer) of the mobile and infrastructure side. Message is used as a synonym to protocol data unit (PDU). A message may contain several information elements.
Primitive:	A primitive is a data unit which is transferred between layers on one component (mobile station or infrastructure). The primitive has an operation code which identifies the primitive and its parameters.
Service Access Point:	A Service Access Point is a data interface between two layers on one component (mobile station or infrastructure).