GSM Protocol Stack

G2 3

GTI-BTI
Generic Target Interface
Bluetooth

Interface Description

Confidential

Author: Condat AG
Alt Moabit 91d
10559 Berlin
Germany

Date: September 04, 2000

ID: 8445.200.00.008

Table of Contents

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. Condat AG 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 Condat AG. Condat AG 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 Condat AG.

Condat AG
Alt Moabit 91d
10559 Berlin
Germany

Telephone:	+49.30.39094-0
Fax:	+49.30.39094-300
Internet:	www.condat.de
E-mail:	gsm@condat.de

0.1 Document History

ID	Author	Date	Approval	Remarks
8445.200.00.001	06-Mar-2000	MG	Initial	
8445.200.00.002	23-May-2000	MG	Added to Source Control with some changes	
8445.200.00.003	30-May-2000	MVI	BTP SAP added	
8445.200.00.004	30-May-2000	MG	GPI renamed to DTI	
8445.200.00.005	10-Aug-2000	MVI	BTP converted to generic interface	
8445.200.00.006	10-Aug-2000	MG	New document template	
8445.200.00.007	04-Sep-2000	MVI	New Generic Profile Interface	
8445.200.00.008	15-Sep-2000	MVI	Changes after implementation	

0.2 References

[GTI] GTI - Generic Target Interface; Interface Description
8410.008.99.004; Condat AG

0.3 Abbreviations

AGCH	Access Grant Channel
AT	Attention sequence "AT" to indicate valid commands of the ACI
BCCH	Broadcast Control Channel
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
CHAP	Challenge Handshake Authentication Protocol
CKSN	Ciphering Key Sequence Number
CRC	Cyclic Redundancy Check
DCCH	Dedicated Control Channel
DCOMP	Identifier of the user data compression algorithm used for the N-DPU
DISC	Disconnect Frame
DL	Data Link Layer
DM	Disconnected Mode Frame
DTX	Discontinuous Transmission
E	Extension bit
EA	Extension Bit Address Field
EL	Extension Bit Length Field
EMMI	Electrical Man Machine Interface
F	Final Bit
FACCH	Fast Associated Control Channel
FHO	Forced Handover
GACI	GPRS AT Command Interpreter
GMM	GPRS Mobility Management
GP	Guard Period
GRR	GPRS RR
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
IP	Internet Protocol
IPCP	Internet Protocol Control Protocol
ITU	International Telecommunication Union
IWF	Interworking Function
Kc	Ciphering Key
L	Length Indicator
LAI	Location Area Information
LCP	Link Control Protocol
LISR	Low level Interrupt Service Routine
LLC	Logical Link Control

LPD	Link Protocol Discriminator
LQM	Link Quality Monitoring
M	More bit used to indicate the last segment of N-DPU
MAC	Medium Access Control
MCC	Mobile Country Code
MM	Mobility Management
MMI	Man Machine Interface
MNC	Mobile Network Code
MS	Mobile Station
MT	Mobile Termination
N(R)	Receive Number
N(S)	Send Number
NC	Network Control
NCC	National Colour Code
NCP	Network Control Protocol
NECI	New Establishment Causes included
N-PDU	Network Protocol Data Unit
NSAPI	Network Layer Service Access Point Identifier
OTD	Observed Time Difference
P	Poll Bit
P/F	Poll/Final Bit
PACCH	Packet Associated Control Channel
PAP	Password Authentication Protocol
PBCCH	Packet BCCH
PCCCH	Packet CCCH
PCOMP	Identifier of the protocol control information compression algorithm used for the N-DPU
PDCH	Packet Data Channel
PDP	Packet Data Protocol e.g. IP or X.25
PDTCH	Packet Data Traffic Channel
PRACH	Packet RACH
PSI	Packet System Information
PCH	Paging Channel
PCO	Point of Control and Observation
PDU	Protocol Data Unit
PL	Physical Layer
PLMN	Public Land Mobile Network
PPP	Point-to-Point Protocol
PTP	Point to Point
QoS	Quality of Service
RACH	Random Access Channel
REJ	Reject Frame
RLC	Radio Link Control
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
SDU	Service Data Unit
SGSN	Serving GPRS Support Node
SIM	Subscriber Identity Module

SM	Session Management
SMS	Short Message Service
SMSCB	Short Message Service Cell Broadcast
SNDCP	Subnetwork Dependant Convergence Protocol
SNSM	SNDCP-SM
SS	Supplementary Services
TAP	Test Application Program
TBF	Temporary Block Flow
TCH	Traffic Channel
TCH/F	Traffic Channel Full Rate
TCH/H	Traffic Channel Half Rate
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TE	Terminal Equipment - e. g. a PC
TFI	Temporary Flow Identifier
TLLI	Temporary Logical Link Identifier
TMSI	Temporary Mobile Subscriber Identity
TQI	Temporary Queuing Identifier
UA	Unnumbered Acknowledgement Frame
UART	Universal Asynchronous Receiver Transmitter
UI	Unnumbered Information Frame
USF	Uplink State Flag
V(A)	Acknowledgement State Variable
V(R)	Receive State Variable
V(S)	Send State Variable
VPLMN	Visiting Public Land Mobile Network

0.4 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).

1 Overview

The Protocol Stacks are used to define the functionality of the GSM protocols for interfaces. The GSM specifications are normative when used to describe the functionality of interfaces, but the stacks and the subdivision of protocol layers does not imply or restrict any implementation.

The protocol stack for GPRS consists of several entities. Each entity has one ore more service access points, over which the entity provides a service for the upper entity.

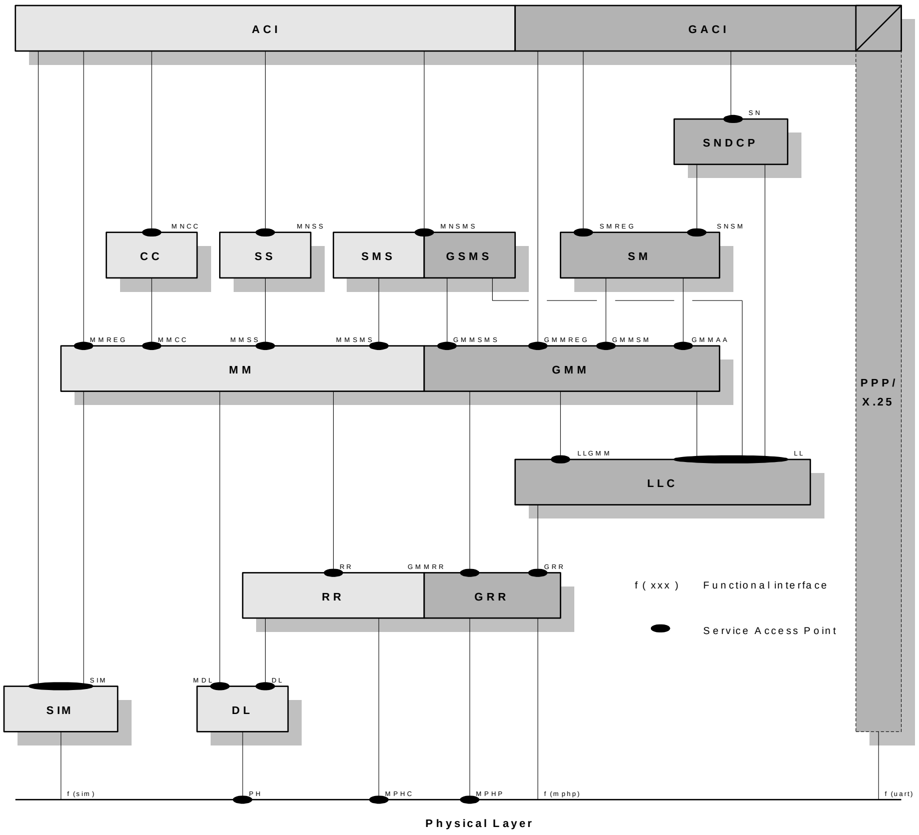

Figure 1.1: Architecture of the GSM/GPRS protocol stack

2 Introduction

The BTI (Bluetooth Interface) is an interface layer, by which an external Bluetooth protocol stack can be adapted to the GSM protocol stack. It provides a mapping of three SAPs onto functions and callback functions.

2.1 Integration of Bluetooth into GSM

From the point of view of the GSM protocol stack the Bluetooth protocol stack consists of a Bluetooth Profile (BPF) and a Bluetooth transport layer (BTL). The BPF is there to control the complete Bluetooth protocol stack and environment. For the GSM protocol stack it behaves like a user of the ACI (DTE) and it can be configured by the MMI.

The transport layer provides a means by which data can be transferred to and from other Bluetooth devices. It uses the data relay to exchange data with GSM data transport services.

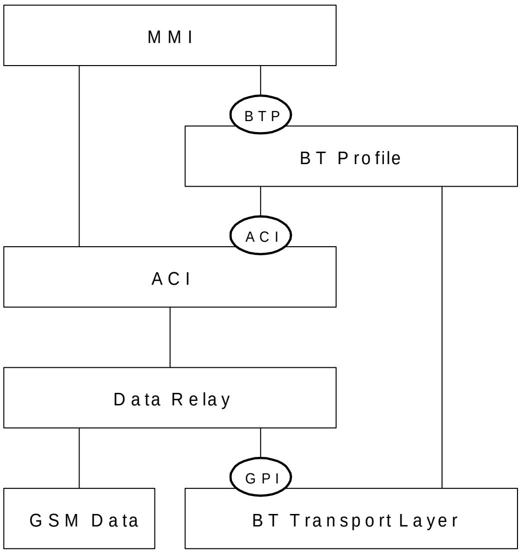

Figure 2.1 Model of Bluetooth Integration into GSM

The BPF uses the ACI SAP to communicate with the ACI. The BPF has the role of a DTE and the ACI has the role of a DCE, i.e. the BPF sends AT-Commands to the ACI and the ACI sends AT responses to the BPF.

The BPF itself provides a SAP, by which it can be configured by the MMI and by which it can send indications to the MMI. The indications can be used to display informations about the Bluetooth environment to the user (e.g. the availability of other Bluetooth devices). This configuration is used for all different Bluetooth profiles.

In addition the BTL provides a SAP, which is conform to the DTI (Data Transmission Interface). User data are exchanged between the GSM and Bluetooth protocol stacks via this SAP. This SAP is not used for all Bluetooth profiles. The Headset Profile does not require any data exchange between the two protocol stacks, because audio data are transferred directly on a lower level. Therefore this SAP is not used for this profile. The Dial Up Networking Profile on the other hand requires the BTL SAP to exchange data with the GSM data services.

2.2 Interfacing to an external Bluetooth Protocol Stack

The given model in the previous paragraph assumes a seamless integration into the frame of the GSM protocol stack.

If SAPs are used for inter task communication like in the given model, then primitives are to be sent via these SAPs. In this case the receiving side is responsible for freeing the memory used by the primitive. Allocating and freeing of memory must be done in the same environment. Since the Condat frame adds some features to the operating system, the usage of SAPs is only possible, if the Bluetooth stack is running in the same frame.

In order to enable the integration with an external Bluetooth protocol stack which is running independently from the GSM frame an interface layer is provided which maps the SAPs onto function calls. Each side provides callback functions, which are used by the other side to send a primitive. The data transfer in the callback functions is done by copying, so that no allocated memory blocks are passed from one side to the other side.

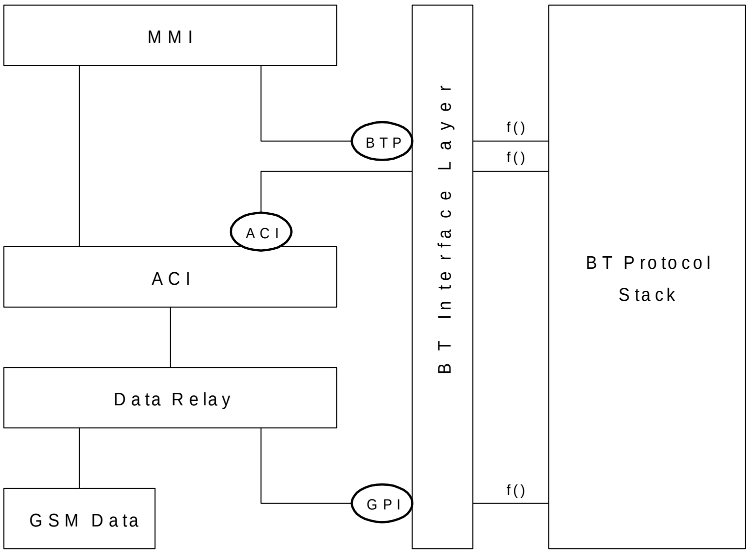

Figure 2.2 The Interface layer between the GSM and Bluetooth protocol stacks

2.3 Mapping of primitives onto function calls

There is a one to one mapping of primitives onto function calls. By this the interface layer becomes straightforward and does not perform any data processing. As a result of this testing is easier, because the standard test environment can be used, which is based on primitives. Moreover the interface layer, which can not be tested with the standard test environment becomes less likely to have errors.

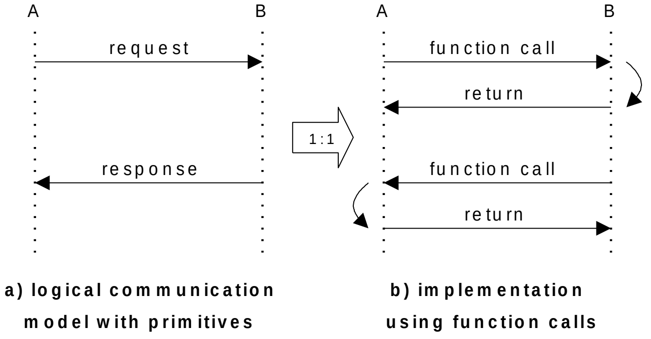
A
B
request
response
1:1
A
B
function call
return
function call
return
a) logical communication model with primitives
b) implementation using function calls

Figure 2.3 Mapping of primitives onto function calls

The callback functions should have no return values, otherwise a mapping on a primitive is not possible, because sending a primitive also does not give a direct return value. All results must be sent by a responding primitive, i.e. by calling a callback function of the other side. In general it is a requirement of all involved SAPs, that each request or indication must be acknowledged by a respective confirmation or response. This is also a means of flow control between the two protocol stacks, which ensures, that no congestion can take place in the interface layer.

Calling a callback function of the other side directly out of callback function is never done by the GSM side but may be done by the BT side. This gives some flexibility for the design of the BT side whereas on the GSM side the principle of mapping primitives onto function calls is kept.

A callback function must not wait for some event but should always return immediatly.

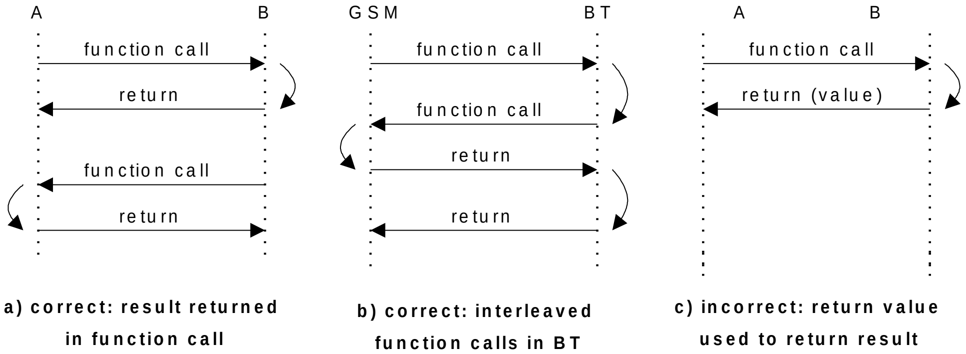
A
B
function call
return
function call
return
a) correct: result returned in function call
GSM
BT
function call
function call
return
return
b) correct: interleaved function calls in BT
A
B
function call
return (value)
c) incorrect: return value used to return result

Figure 2.4 Correct and incorrect usage of function calls

All callback functions of the Bluetooth stack, which are called from the GSM stack to start a transaction are called *request* (bti_xxx_req), the respective acknowledging callback functions of the GSM stack are called *confirm* (bti_xxx_cnf). The callback functions of the GSM stack by which a transaction is started are called *indication* (bti_xxx_ind) and the acknowledging functions are called *response* (bti_xxx_res). This notation system is conform with the BTP and DTI SAP, in which the GSM stack uses the service of the Bluetooth stack. It is contrary to the ACI SAP which is provided by the GSM stack and used by the Bluetooth side.

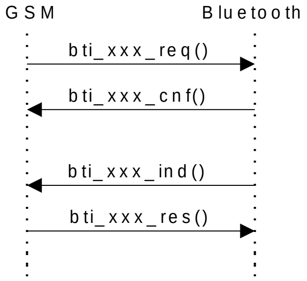

Figure 2.5 Notation of function calls

2.3.1 Mapping of the BTP SAP functions

♣♣Function	Mapped Primitive	Provided by	Called by
bti_init_profile_req()	BTP_INIT_PROFILE_REQ	BT	GSM
bti_init_profile_cnf()	BTP_INIT_PROFILE_CNF	GSM	BT
bti_deinit_profile_req()	BTP_DEINIT_PROFILE_REQ	BT	GSM
bti_deinit_profile_cnf()	BTP_DEINIT_PROFILE_CNF	GSM	BT
bti_device_search_req	BTP_DEVICE_SEARCH_REQ	BT	GSM
bti_device_search_cnf	BTP_DEVICE_SEARCH_CNF	GSM	BT
bti_device_found_ind	BTP_DEVICE_FOUND_IND	GSM	BT
bti_connect_device_req()	BTP_CONNECT_DEVICE_REQ	BT	GSM
bti_connect_device_cnf()	BTP_CONNECT_DEVICE_CNF	GSM	BT
bti_connect_device_ind()	BTP_CONNECT_DEVICE_IND	GSM	BT
bti_connect_device_res()	BTP_CONNECT_DEVICE_RES	BT	GSM

♣♣Function	Mapped Primitive	Provided by	Called by
bti_disconnect_device_req()	BTP_DISCONNECT_DEVICE_REQ	BT	GSM
bti_disconnect_device_cnf()	BTP_DISCONNECT_DEVICE_CNF	GSM	BT
bti_disconnect_device_ind()	BTP_DISCONNECT_DEVICE_IND	GSM	BT
bti_audio_connection_ind()	BTP_AUDIO_CONNECTION_IND	GSM	BT
bti_transfer_audio_in_req()	BTP_TRANSFER_AUDIO_IN_REQ	BT	GSM
bti_transfer_audio_in_cnf()	BTP_TRANSFER_AUDIO_IN_CNF	GSM	BT
bti_transfer_audio_out_req()	BTP_TRANSFER_AUDIO_OUT_REQ	BT	GSM
bti_transfer_audio_out_cnf()	BTP_TRANSFER_AUDIO_OUT_CNF	GSM	BT
bti_pin_ind()	BTP_PIN_IND	GSM	BT
bti_pin_res()	BTP_PIN_RES	BT	GSM
bti_reconfig_profile_req()	BTP_RECONFIG_PROFILE_REQ	BT	GSM
bti_reconfig_profile_cnf()	BTP_RECONFIG_PROFILE_CNF	GSM	BT
bti_device_control_req()	BTP_DEVICE_CONTROL_REQ	BT	GSM
bti_device_control_cnf()	BTP_DEVICE_CONTROL_CNF	GSM	BT
bti_device_control_ind()	BTP_DEVICE_CONTROL_IND	GSM	BT
bti_device_control_res()	BTP_DEVICE_CONTROL_RES	BT	GSM

The callback functions are descibed in detail in Appendix 1: The Functional Interface

2.3.2 Mapping of the ACI SAP functions

♣♣Function	Mapped Primitive	Provided by	Called by
bti_at_init_req()	ACI_INIT_IND	BT	GSM
bti_at_init_cnf()	ACI_INIT_RES	GSM	BT
bti_at_deinit_ind()	ACI_DEINIT_REQ	GSM	BT
bti_at_deinit_res()	ACI_DEINIT_CNF	BT	GSM
bti_at_open_port_ind()	ACI_OPEN_PORT_REQ	GSM	BT
bti_at_open_port_res()	ACI_OPEN_PORT_CNF	BT	GSM
bti_at_close_port_ind()	ACI_CLOSE_PORT_REQ	GSM	BT
bti_at_close_port_res()	ACI_CLOSE_PORT_CNF	BT	GSM
bti_at_cmd_ind()	ACI_CMD_REQ	GSM	BT
bti_at_cmd_res()	ACI_CMD_CNF	BT	GSM
bti_at_cmd_req()	ACI_CMD_IND	BT	GSM
bti_at_cmd_cnf()	ACI_CMD_RES	GSM	BT
bti_at_abort_ind()	ACI_ABORT_REQ	GSM	BT

♣♣Function	Mapped Primitive	Provided by	Called by
bti_at_abort_res()	ACI_ABORT_CNF	BT	GSM

The callback functions are descibed in detail in Appendix 1: The Functional Interface

2.3.3 Mapping of the DTI SAP functions

♣♣Function	Mapped Primitive	Provided by	Called by
bti_data_ready_ind()	DTI_DATA_IND	GSM	BT
bti_data_ready_res()	DTI_GETDATA_REQ	BT	GSM
bti_data_ready_req()	DTI_DATA_REQ	BT	GSM
bti_data_ready_cnf()	DTI_READY_IND	GSM	BT

The callback functions are descibed in detail in Appendix 1: The Functional Interface

3 Protocol

3.1 Interface to the BTP SAP

3.1.1 BT Profile Initialization

```
  SMI                         BTI                   Bluetooth PS
   |                            |                           |
   |   BTP_INIT_PROFILE_REQ    |                           |
   *===========================>*                           |
   |                        (BTI 1)                         |
   |                            |  bti_init_profile_req()   |
   |                            |==========================>|
   |                            |                        (BPS 1)
   |                            |  bti_init_profile_cnf()   |
   |                            |<==========================|
   |                        (BTI 2)                         |
   |   BTP_INIT_PROFILE_CNF    |                           |
   *<===========================*                           |
(SMI 1)                         |                           |
   |                            |                           |
```

(BTI 1)
SMI sends initialisation request parameters to the Bluetooth entity

(BPS 1)
BTI requests initialization from the BPS

(BTI 2)
BPS confirmes the initialization

(SMI 1)
BTI confirms the initialization

3.1.2 Deinitialization of a Bluetooth Profile

```
  SMI                         BTI                     Bluetooth PS
   |                           |                           |
   |  BTP_DEINIT_PROFILE_REQ   |                           |
   *=========================> *                           |
   |                       (BTI 1)                         |
   |                           |                           |
   |                           |  bti_deinit_profile_req() |
   |                           |==========================>|
   |                           |                       (BPS 1)
   |                           |                           |
   |                           |  bti_deinit_profile_cnf() |
   |                           |<==========================|
   |                       (BTI 2)                         |
   |                           |                           |
   |  BTP_DEINIT_PROFILE_CNF   |                           |
   *<========================= *                           |
(SMI 1)                        |                           |
   |                           |                           |
```

(BTI 1)
SMI request deinitialization of a Bluetooth profile

(BPS 1)
The request is forwarded to the BPS

(BTI 2)
The deinitialization is confirmed

(SMI 1)
The confirmation is forwarded to SMI

3.1.3 Reconfiguration of a profile

```
  SMI                         BTI                     Bluetooth PS
   | BTP_RECONFIG_PROFILE_REQ  |                           |
   *=========================> *                           |
   |                       (BTI 1)                         |
   |                           | bti_reconfig_profile_req() |
   |                           |==========================>|
   |                           |                       (BPS 1)
   |                           |                           |
   |                           | bti_reconfig_profile_cnf() |
   |                           |<==========================|
   |                       (BTI 2)                         |
   |                           |                           |
   | BTP_RECONFIG_PROFILE_CNF  |                           |
   *<========================= *                           |
(SMI 1)                        |                           |
   |                           |                           |
```

(BTI 1)
The SMI requests reconfiguration (first configuration is done during initialisation) of the profile

(BPS 1)
The BTI forwards the reconfiguration request

(BTI 2)
The BTP confirms the reconfiguration

(SMI 1)
BTI forwards the reconfiguration

3.1.4 Searching for available devices

```
  SMI                         BTI                     Bluetooth PS
   |                            |                           |
   |   BTP_DEVICE_SEARCH_REQ    |                           |
   *===========================>*                           |
   |                        (BTI 1)                         |
   |                            |                           |
   |                            |  bti_device_search_req()  |
   |                            |==========================>|
   |                            |                        (BPS 1)
   |                            |                           |
   |                            |  bti_device_found_ind()   |
   |                            |<==========================|
   |                        (BTI 2)                         |
   |                            |                           |
   |    BTP_DEVICE_FOUND_IND    |                           |
   *<===========================*                           |
(SMI 1)                         |                           |
   |                            |                           |
   |                            |                           |
   |                            |  bti_device_found_ind()   |
   |                            |<==========================|
   |                        (BTI 3)                         |
   |                            |                           |
   |    BTP_DEVICE_FOUND_IND    |                           |
   *<===========================*                           |
(SMI 2)                         |                           |
   |                            |                           |
   |                            |                           |
   |                            |  bti_device_search_cnf()  |
   |                            |<==========================|
   |                        (BTI 4)                         |
   |                            |                           |
   |   BTP_DEVICE_SEARCH_CNF    |                           |
   *<===========================*                           |
(SMI 3)                         |                           |
   |                            |                           |
```

(BTI 1)
SMI requests the BTI to search for available devices (of a specific type).

(BPS 1)
BTI forwards the request

(BTI 2)
A device is found

(SMI 1)
The device data is forwarded

(BTI 3)
Another device is found

(SMI 2)
The device data is forwarded

(BTI 3)
BPS confirmes when the search is finished. During this search two devices of the requested type was found.

(SMI 1)
BTI forwards the confirmation

3.1.5 Outgoing connection

```
  SMI                         BTI                     Bluetooth PS
   |                           |                           |
   |  BTP_CONNECT_DEVICE_REQ   |                           |
   *==========================>*                           |
   |                        (BTI 1)
   |                           |                           |
   |                           |  bti_connect_device_req() |
   |                           |==========================>|
   |                           |                        (BPS 1)
   |                           |                           |
   |                           |  bti_connect_device_cnf() |
   |                           |<==========================|
   |                        (BTI 2)                        |
   |                           |                           |
   |  BTP_CONNECT_DEVICE_CNF   |                           |
   *<==========================*                           |
(SMI 1)                        |                           |
   |                           |                           |
```

(BTI 1)
SMI request connection to the device

(BPS 1)
The request is forwarded to the BPS

(BTI 2)
The connection is confirmed

(SMI 1)
The confirmation is forwarded to SMI

3.1.6 Ingoing connection

```
  SMI                         BTI                     Bluetooth PS
   |                           |                            |
   |                           |  bti_connect_device_ind()  |
   |                           |<===========================|
   |                         (BTI 1)                        |
   |                           |                            |
   |  BTP_CONNECT_DEVICE_IND   |                            |
   *<========================= *                            |
(SMI 1)                        |                            |
   |                           |                            |
   |  BTP_CONNECT_DEVICE_RES   |                            |
   *=========================> *                            |
   |                         (BTI 2)                        |
   |                           |                            |
   |                           |  bti_connect_device_res()  |
   |                           |===========================>|
   |                           |                         (BPS 1)
   |                           |                            |
```

(BTI 1)
BPS indicates that a remote device wants to establish a connection

(SMI 1)
The indication is forwarded to the SMI

(BTI 2)
The SMI responds

(BPS 1)
The response is forwarded to BPS

3.1.7 Entering of PIN

```
  SMI                         BTI                     Bluetooth PS
   |                           |                            |
   |                           |        bti_pin_ind()       |
   |                           |<===========================|
   |                         (BTI 1)                        |
   |                           |                            |
   |        BTP_PIN_IND        |                            |
   *<========================= *                            |
(SMI 1)                        |                            |
   |                           |                            |
   |        BTP_PIN_RES        |                            |
   *=========================> *                            |
   |                         (BTI 2)                        |
   |                           |                            |
   |                           |        bti_pin_res()       |
   |                           |===========================>|
   |                           |                         (BPS 1)
   |                           |                            |
```

(BTI 1)
BPS indicates that a pin code is needed (if bonding is enabled)

(SMI 1)
The indication is forwarded to the SMI

(BTI 2)
The SMI responds with the pin code

(BPS 1)
The response is forwarded to BPS

3.1.8 Audio connection/disconnection/fail

```
  SMI                         BTI                     Bluetooth PS
   |                            |                           |
   |                            | bti_audio_connection_ind()|
   |                            |<==========================|
   |                         (BTI 1)                        |
   |                            |                           |
   |  BTP_AUDIO_CONNECTION_IND  |                           |
   *<===========================*                           |
(SMI 1)                         |                           |
   |                            |                           |
```

(BTI 1)
BPS indicates that ands audio connection to the device is setup or closed down or failed

(SMI 1)
The indication is forwarded to SMI

3.1.9 Transfer connection out

```
  SMI                         BTI                     Bluetooth PS
   |                            |                           |
   | BTP_TRANSFER_AUDIO_OUT_REQ |                           |
   *===========================>*                           |
   |                         (BTI 1)                        |
   |                            |                           |
   |                            |bti_transfer_audio_out_req()|
   |                            |==========================>|
   |                            |                        (BPS 1)
   |                            |                           |
   |                            |bti_transfer_audio_out_cnf()|
   |                            |<==========================|
   |                         (BTI 2)                        |
   | BTP_TRANSFER_AUDIO_OUT_CNF |                           |
   *<===========================*                           |
(SMI 1)                         |                           |
   |                            |                           |
```

(BTI 1)
SMI request transfer of the audio from the loudspeaker in the handset to the device

(BPS 1)
The request is forwarded to the BPS

(BTI 2)
A connection to the headset is established and an audio connection is established as well. When this is done the transfer is confirmed.

(SMI 1)
The confirmation is forwarded to SMI

3.1.10 Transfer connection in

```
   SMI                         BTI                     Bluetooth PS
    |                           |                           |
    | BTP_TRANSFER_AUDIO_IN_REQ |                           |
    *=========================> *                           |
    |                        (BTI 1)                        |
    |                           |bti_transfer_audio_in_req()|
    |                           |==========================>|
    |                           |                       (BPS 1)
    |                           |                           |
    |                           |bti_transfer_audio_in_cnf()|
    |                           |<==========================|
    |                        (BTI 2)                        |
    | BTP_TRANSFER_AUDIO_IN_CNF |                           |
    *<========================= *                           |
 (SMI 1)                        |                           |
    |                           |                           |
```

(BTI 1)
SMI request transfer of the audio from the headset to the loudspeaker in the handset

(BPS 1)
The request is forwarded to the BPS

(BTI 2)
The transfer is confirmed

(SMI 1)
The confirmation is forwarded to SMI

3.1.11 Disconnecting from a device initiated by SMI

```
   SMI                         BTI                     Bluetooth PS
    |                           |                           |
    | BTP_DISCONNECT_DEVICE_REQ |                           |
    *=========================> *                           |
    |                        (BTI 1)                        |
    |                           |                           |
    |                           |bti_disconnect_device_req()|
    |                           |==========================>|
    |                           |                       (BPS 1)
    |                           |                           |
    |                           |bti_disconnect_device_cnf()|
    |                           |<==========================|
    |                        (BTI 2)                        |
    |                           |                           |
    | BTP_DISCONNECT_DEVICE_CNF |                           |
    *<========================= *                           |
 (SMI 1)                        |                           |
    |                           |                           |
```

(BTI 1)
SMI request disconnection from the device

(BPS 1)
The request is forwarded to the BPS

(BTI 2)
The disconnection is confirmed

(SMI 1)
The confirmation is forwarded to SMI

3.1.12 Disconnecting from a device initiated by BTP

```
  SMI                         BTI                     Bluetooth PS
   |                           |                            |
   |                           |bti_disconnect_device_ind() |
   |                           |<===========================|
   |                         (BTI 1)                        |
   |                           |                            |
   | BTP_DISCONNECT_DEVICE_IND |                            |
   *<==========================*                            |
(SMI 1)                        |                            |
   |                           |                            |
```

(BTI 1)
BPS indicates disconnection from the device

(SMI 1)
The indication is forwarded to SMI

3.1.13 Control of vendor specific functionality initiated by the SMI

```
  SMI                         BTI                     Bluetooth PS
   |                           |                            |
   |  BTP_DEVICE_CONTROL_REQ   |                            |
   *==========================>*                            |
   |                         (BTI 1)                        |
   |                           |  bti_device_control_req()  |
   |                           |===========================>|
   |                           |                         (BPS 1)
   |                           |                            |
   |                           |  bti_device_control_cnf()  |
   |                           |<===========================|
   |                         (BTI 2)                        |
   |  BTP_DEVICE_CONTROL_CNF   |                            |
   *<==========================*                            |
(SMI 1)                        |                            |
   |                           |                            |
```

(BTI 1)
SMI sends control information to the BTI.

(BPS 1)
BTI forwards to the BPS.

(BTI 2)
BPS confirms the when the control request is processed.

(SMI 1)
BTI forwards the confirmation.

3.1.14 Control of vendor specific functionality initiated by the profile

```
  SMI                         BTI                     Bluetooth PS
   |                            |                           |
   |                            | bti_device_control_ind()  |
   |                            |<==========================|
   |                         (BTI 1)                        |
   |  BTP_DEVICE_CONTROL_IND    |                           |
   *<===========================*                           |
(SMI 1)                         |                           |
   |                            |                           |
   |  BTP_DEVICE_CONTROL_RES    |                           |
   *===========================>*                           |
   |                         (BTI 2)                        |
   |                            | bti_device_control_res()  |
   |                            |==========================>|
   |                            |                        (BPS 1)
   |                            |                           |
```

(BTI 1)
The Bluetooth PS needs some action from the SMI, or informs the SMI about changes in the state of the device.

(SMI 1)
BTI forwards the indication.

(BTI 2)
If a response from the SMI is required this is sent.

(BPS 1)
BTI forwards the response.

3.2 Interface to the ACI SAP

3.2.1 Initialisation of ACI

```
  ACI                         BTI                     Bluetooth PS
   |                            |                           |
   |       ACI_INIT_IND         |                           |
   *===========================>*                           |
   |                         (BTI 1)                        |
   |                            |     bti_at_init_req()     |
   |                            |==========================>|
   |                            |                        (BPS 1)
   |                            |                           |
   |                            |     bti_at_init_cnf()     |
   |                            |<==========================|
   |                         (BTI 2)                        |
   |       ACI_INIT_RES         |                           |
   *<===========================*                           |
(ACI 1)                         |                           |
   |                            |                           |
```

(BTI 1)
ACI wants to initialize the Bluetooth PS.

(BPS 1)
BTI requests the BPS to initialize.

(BTI 2)
BPS confirms the initialisation

(BPS 1)
BTI signals to ACI, that the Bluetooth PS has been initialised successfully.

3.2.2 Opening a port

```
   ACI                          BTI                     Bluetooth PS
    |                            |                           |
    |                            |  bti_at_open_port_ind()   |
    |                            |<==========================|
    |                         (BTI 1)                        |
    |     ACI_OPEN_PORT_REQ      |                           |
    *<========================== *                           |
 (ACI 1)                         |                           |
    |                            |                           |
    |     ACI_OPEN_PORT_CNF      |                           |
    *==========================> *                           |
    |                         (BTI 2)                        |
    |                            |  bti_at_open_port_res()   |
    |                            |==========================>|
    |                            |                        (BPS 1)
    |                            |                           |
```

(BTI 1)
The Bluetooth PS wants to open a port. Opening a port is only possible after initialisation.

(ACI 1)
BTI requests ACI to open a port.

(BTI 2)
ACI confirms the opening of a port

(BPS 1)
BTI signals to the Bluetooth PS, that a port has been opened.

3.2.3 Closing a port

```
   ACI                          BTI                     Bluetooth PS
    |                            |                           |
    |                            | bti_at_close_port_ind()   |
    |                            |<==========================|
    |                         (BTI 1)                        |
    |    ACI_CLOSE_PORT_REQ      |                           |
    *<========================== *                           |
 (ACI 1)                         |                           |
    |                            |                           |
    |    ACI_CLOSE_PORT_CNF      |                           |
    *==========================> *                           |
    |                         (BTI 2)                        |
    |                            | bti_at_close_port_res()   |
    |                            |==========================>|
    |                            |                        (BPS 1)
    |                            |                           |
```

(BTI 1)
The Bluetooth PS wants to close a port.

(ACI 1)
BTI requests ACI to close the port.

(BTI 2)
ACI confirms the closing of the port

(BPS 1)
BTI signals to the Bluetooth PS, that the port has been closed.

3.2.4 Deinitialisation of ACI

```
   ACI                         BTI                     Bluetooth PS
    |                           |                           |
    |                           |    bti_at_deinit_ind()    |
    |                           |<==========================|
    |                        (BTI 1)                        |
    |       ACI_DEINIT_REQ      |                           |
    *<==========================*                           |
 (ACI 1)                        |                           |
    |                           |                           |
    |       ACI_DEINIT_CNF      |                           |
    *==========================>*                           |
    |                        (BTI 2)                        |
    |                           |    bti_at_deinit_res()    |
    |                           |==========================>|
    |                           |                        (BPS 1)
    |                           |                           |
```

(BTI 1)
The Bluetooth PS wants to deinitialize ACI.

(ACI 1)
BTI requests ACI to deinitialize.

(BTI 2)
ACI confirms the deinitialisation

(BPS 1)
BTI signals to the Bluetooth PS, that ACI has been deinitialised successfully. This can be used to unregister the GSM PS in the Bluetooth environment.

3.2.5 Bluetooth sends an AT command

```
   ACI                         BTI                     Bluetooth PS
    |                           |                           |
    |                           |      bti_at_cmd_ind()     |
    |                           |            (AT)           |
    |                           |<==========================|
    |                        (BTI 1)                        |
    |        ACI_CMD_REQ        |                           |
    |            (AT)           |                           |
    *<==========================*                           |
 (ACI 1)                        |                           |
    |        ACI_CMD_CNF        |                           |
    |            (OK)           |                           |
    *==========================>*                           |
    |                        (BTI 2)                        |
    |                           |      bti_at_cmd_res()     |
    |                           |            (OK)           |
    |                           |==========================>|
    |                           |                        (BPS 1)
    |                           |                           |
```

(BTI 1)
Sending AT commands and responses is only possible after initialising and opening a port. The Bluetooth PS sends an AT command to BTI (“AT”). The next command may be sent only after receiveing the confirmation (BPS 1)

(ACI 1)
BTI forwards the command to ACI.

(BTI 2)
ACI returns a confirmation (“OK”)

(BPS 1)
BTI forwards the confirmation to the Bluetooth PS.

3.2.6 Bluetooth receives an unsolicited response

```
  ACI                         BTI                     Bluetooth PS
   |                           |                           |
   |        ACI_CMD_IND        |                           |
   |           (+FCO)          |                           |
   *==========================>*                           |
   |                        (BTI 1)                        |
   |                           |      bti_at_cmd_req()     |
   |                           |           (+FCO)          |
   |                           |==========================>|
   |                           |                         (BPS 1)
   |                           |      bti_at_cmd_cnf()     |
   |                           |<==========================|
   |                        (BTI 2)                        |
   |        ACI_CMD_RES        |                           |
   *<==========================*                           |
(ACI 1)                        |                           |
   |                           |                           |
```

(BTI 1)
The ACI sends an unsolicited response to Bluetooth (+FCO). The next unsolicited response or confirmation of any pending AT command may be sent only after reciving the ACI_CMD_RES (ACI 1).

(BPS 1)
BTI forwards the response to Bluetooth.

(BTI 2)
The Bluetooth PS confirms the response. No AT command or response is included in this confirmation.

(ACI 1)
BTI forwards the confirmation to the ACI.

3.2.7 Bluetooth establishes a voice call

```
  ACI                         BTI                     Bluetooth PS
   |                           |                           |
   |                           |      bti_at_cmd_ind()     |
   |                           |          (ATD123;)        |
   |                           |<==========================|
   |                        (BTI 1)                        |
   |        ACI_CMD_REQ        |                           |
   |         (ATD123;)         |                           |
   *<==========================*                           |
(ACI 1)                        |                           |
   |        ACI_CMD_CNF        |                           |
   |         (CONNECT)         |                           |
   *==========================>*                           |
   |                        (BTI 2)                        |
   |                           |      bti_at_cmd_res()     |
   |                           |          (CONNECT)        |
   |                           |==========================>|
   |                           |                         (BPS 1)
   |                           |                           |
```

(BTI 1)
The Bluetooth PS sends an dial command to BTI. The number must be terminated by a semicolon (';') to establish a voice call (s. GSM 07.07).

(ACI 1)
BTI forwards the command to ACI.

(BTI 2)
ACI returns a confirmation for the establishment of the call (“CONNECT xxxx”)

(BPS 1)
BTI forwards the confirmation to the Bluetooth PS.

3.2.8 Bluetooth receives an incoming call

```
  ACI                         BTI                      Bluetooth PS
   |                            |                            |
   |        ACI_CMD_IND         |                            |
   |           (RING)           |                            |
   *==========================> *                            |
   |                         (BTI 1)                         |
   |                            |     bti_at_cmd_req()       |
   |                            |           (RING)           |
   |                            |===========================>|
   |                            |                         (BPS 1)
   |                            |     bti_at_cmd_cnf()       |
   |                            |<===========================|
   |                         (BTI 2)                         |
   |        ACI_CMD_RES         |                            |
   *<========================== *                            |
(ACI 1)                         |                            |
   |                            |                            |
   |                            |     bti_at_cmd_ind()       |
   |                            |           (ATA)            |
   |                            |<===========================|
   |                         (BTI 3)                         |
   |        ACI_CMD_REQ         |                            |
   |           (ATA)            |                            |
   *<========================== *                            |
(ACI 2)                         |                            |
   |        ACI_CMD_CNF         |                            |
   |         (CONNECT)          |                            |
   *==========================> *                            |
   |                         (BTI 4)                         |
   |                            |     bti_at_cmd_res()       |
   |                            |         (CONNECT)          |
   |                            |===========================>|
   |                            |                         (BPS 2)
   |                            |                            |
```

(BTI 1)
The ACI sends a RING response to Bluetooth. The next unsolicited response or confirmation of any pending AT command may be sent only after reciving the ACI_CMD_RES (ACI 1).

(ACI 1)
BTI forwards the RING response to Bluetooth.

(BTI 2)
The Bluetooth PS confirms the RING response. No AT command or response is included in this confirmation. In particular the following ATA command is not included in this confirmation. The reason is, that ervery IND must acknowledged by a RES and every REQ must be acknowledged by a CNF. Therefore a separate pair of messages (ACI_CMD_REQ/ACI_CMD_CNF) is used for ATA/CONNECT.

(BPS 1)
BTI forwards the confirmation to the ACI.

(BTI 3)
The Bluetooth PS sends an ATA command to BTI in order to receive the call.

(ACI 2)
BTI forwards the command to ACI.

(BTI 4)
ACI returns a confirmation for the establishment of the call ("CONNECT xxxx")

(BPS 2)
BTI forwards the confirmation to the Bluetooth PS.

3.2.9 AT command, which includes more than one response

Some AT commands produce additional responses before the final acknowledgement appears in the form of a OK or CONNECT response. The additional responses are sent in an ACI_CMD_IND, whereas the final result is sent in an ACI_CMD_CNF. Therefore one or several ACI_CMD_IND/ACI_CMD_RES message pairs are inserted between the ACI_CMD_REQ and ACI_CMD_CNF. An example for this are all query commands.

```
  ACI                         BTI                     Bluetooth PS
   |                           |                           |
   |                           |      bti_at_cmd_ind()     |
   |                           |         (AT+CBST?)        |
   |                           |<==========================|
   |                        (BTI 1)                        |
   |        ACI_CMD_REQ        |                           |
   |         (AT+CBST?)        |                           |
   *<========================= *                           |
(ACI 1)                        |                           |
   |                           |                           |
   |        ACI_CMD_IND        |                           |
   |       (+CBST=7,0,1)       |                           |
   *=========================> *                           |
   |                        (BTI 2)                        |
   |                           |      bti_at_cmd_req()     |
   |                           |       (+CBST=7,0,1)       |
   |                           |==========================>|
   |                           |                        (BPS 1)
   |                           |      bti_at_cmd_cnf()     |
   |                           |<==========================|
   |                        (BTI 3)                        |
   |        ACI_CMD_RES        |                           |
   *<========================= *                           |
(ACI 2)                        |                           |
   |                           |                           |
   |        ACI_CMD_CNF        |                           |
   |            (OK)           |                           |
   *=========================> *                           |
   |                        (BTI 4)                        |
   |                           |      bti_at_cmd_res()     |
   |                           |            (OK)           |
   |                           |==========================>|
   |                           |                        (BPS 2)
   |                           |                           |
```

(BTI 1)
The Bluetooth PS sends a AT+CBST=? command to BTI to query the CBST parameters.

(ACI 1)
BTI forwards the command to ACI.

(BTI 2)
ACI sends the queried response with an ACI_CMD_IND to BTI. The acknowledgement for the AT+CBST command is still pending and will be given in (BTI 4). ACI is allowd to send the next message (ACI_CMD_IND of ACI_CMD_CNF) only after receiveing ACH_CMD_RES in (ACI 2)

(BPS 1)
BTI forwards the +CBST response to the Bluetooth PS

(BTI 3)
The Bluetooth PS acknowledges the +CBST response.

(ACI 2)
BTI forward the acknowledgement to ACI. Now ACI can send the next ACI_CMD_IND of ACI_CMD_CNF

(BTI 4)
ACI returns a confirmation for the AT+CBST command

(BPS 2)
BTI forwards the confirmation to the Bluetooth PS.

3.2.10 Collision of an incoming and outgoing call as seen in ACI

```
  ACI                         BTI                     Bluetooth PS
   |                            |                           |
   |        ACI_CMD_IND         |                           |
   |           (RING)           |                           |
   *==========================> *                           |
   |                         (BTI 1)                        |
   |                            |     bti_at_cmd_req()      |
   |                            |          (RING)           |
   |                            |==========================>|
   |                            |                        (BPS 1)
   |                            |     bti_at_cmd_ind()      |
   |                            |         (ATD123)          |
   |                            |<==========================|
   |                         (BTI 2)                        |
   |        ACI_CMD_REQ         |                           |
   |          (AT123)           |                           |
   *<========================== *                           |
(ACI 1)                         |                           |
   |                            |                           |
   |                            |     bti_at_cmd_cnf()      |
   |                            |<==========================|
   |                         (BTI 3)                        |
   |        ACI_CMD_RES         |                           |
   *<========================== *                           |
(ACI 2)                         |                           |
   |                            |                           |
   |        ACI_CMD_CNF         |                           |
   |        (NO CARRIER)        |                           |
   *==========================> *                           |
   |                         (BTI 4)                        |
   |                            |     bti_at_cmd_res()      |
   |                            |       (NO CARRIER)        |
   |                            |==========================>|
   |                            |                        (BPS 2)
   |                            |                           |
```

(BTI 1)
The ACI sends a RING response to BTI.

(BPS 1)
BTI forwards the RING response to the Bluetooth PS

(BTI 2)
BTI receives a dial command from the Bluetooth PS. Usually the Bluetooth PS should not send a dial command after receiving a RING response, but this command may have been sent before receiving the RING response and is received now with a delay.

(ACI 1)
BTI forwards the dial command to ACI. ACI can not reply immediatly, because the acknowledgment for the RING response is still pending. The reply will be given in BTI 4 after receiving the ACI_CMD_RES.

(BTI 3)
The Bluetooth PS acknowledges the RING response.

(ACI 2)
BTI forward the acknowledgement to ACI. Now ACI can send the next ACI_CMD_IND of ACI_CMD_CNF

(BTI 4)
ACI returns a confirmation for the ATD command. In this case the call is rejected with a NO CARRIER response, because an incoming call is pending.

(BPS 2)
BTI forwards the confirmation to the Bluetooth PS.

3.2.11 Collision of an incoming and outgoing call as seen in the Bluetooth PS

```
  ACI                         BTI                       Bluetooth PS
   |                           |                           |
   |                           |      bti_at_cmd_ind()     |
   |                           |          (ATD123)         |
   |                           |<==========================|
   |                        (BTI 1)                        |
   |        ACI_CMD_REQ        |                           |
   |          (AT123)          |                           |
   *<==========================*                           |
(ACI 1)                        |                           |
   |                           |                           |
   |        ACI_CMD_IND        |                           |
   |           (RING)          |                           |
   *==========================>*                           |
   |                        (BTI 2)                        |
   |                           |      bti_at_cmd_req()     |
   |                           |           (RING)          |
   |                           |==========================>|
   |                           |                        (BPS 1)
   |                           |      bti_at_cmd_cnf()     |
   |                           |<==========================|
   |                        (BTI 3)                        |
   |        ACI_CMD_RES        |                           |
   *<==========================*                           |
(ACI 2)                        |                           |
   |                           |                           |
   |        ACI_CMD_CNF        |                           |
   |        (NO CARRIER)       |                           |
   *==========================>*                           |
   |                        (BTI 4)                        |
   |                           |      bti_at_cmd_res()     |
   |                           |        (NO CARRIER)       |
   |                           |==========================>|
   |                           |                        (BPS 2)
   |                           |                           |
```

(BTI 1)
BTI receives a dial command from the Bluetooth PS.

(ACI 1)
BTI forwards the dial command to ACI.

(BTI 2)
BTI receives a RING response from the ACI. Usually the ACI should not send a RING response after receiving a dial command, but this response may have been sent before receiving the dial command and is received now with a delay.

(BPS 1)
BTI forwards the RING response to the Bluetooth PS

(BTI 3)
The Bluetooth PS acknowledges the RING response.

(ACI 2)
BTI forward the acknowledgement to ACI. Now ACI can send the next ACI_CMD_IND of ACI_CMD_CNF

(BTI 4)
ACI returns a confirmation for the ATD command. In this case the call is rejected with a NO CARRIER response, because an incoming call is pending.

(BPS 2)
BTI forwards the confirmation to the Bluetooth PS.

3.2.12 Abortion of an AT command

```
  ACI                         BTI                     Bluetooth PS
   |                           |                           |
   |                           |      bti_at_cmd_ind()     |
   |                           |<==========================|
   |                        (BTI 1)                        |
   |        ACI_CMD_REQ        |                           |
   *<==========================*                           |
(ACI 1)                        |                           |
   |                           |     bti_at_abort_ind()    |
   |                           |<==========================|
   |                        (BTI 2)                        |
   |       ACI_ABORT_REQ       |                           |
   *<==========================*                           |
(ACI 2)                        |                           |
   |                           |                           |
   |       ACI_ABORT_CNF       |                           |
   |            (OK)           |                           |
   *==========================>*                           |
   |                        (BTI 3)                        |
   |                           |     bti_at_abort_res()    |
   |                           |            (OK)           |
   |                           |==========================>|
   |                           |                        (BPS 1)
   |                           |                           |
```

(BTI 1)
The Bluetooth Stack sends an AT command to BTI.

(ACI 1)
The AT command is forwarded to ACI.

(BTI 2)
The Bluetooth PS aborts the command with a call to bti_at_abort_ind().

(ACI 2)
The abort request is forwarded to ACI. The running command is stopped. ACI will not send any result of the interrupted command to BTI.

(BTI 3)
ACI confirms the abortion of the command.

(BPS 1)
BTI forwards the confirmation to the Bluetooth PS.

3.2.13 Collision of command abortion and confirmation on GSM side

```
   ACI                          BTI                     Bluetooth PS
    |                            |                            |
    |                            |     bti_at_cmd_ind()       |
    |                            |<===========================|
    |                        (BTI 1)                          |
    |        ACI_CMD_REQ         |                            |
    *<========================== *                            |
 (ACI 1)                         |                            |
    |                            |    bti_at_abort_ind()      |
    |                            |<===========================|
    |                        (BTI 2)                          |
    |       ACI_ABORT_REQ        |                            |
    *<========================== *                            |
 (ACI 2)                         |                            |
    |                            |                            |
    |        ACI_CMD_CNF         |                            |
    |            (OK)            |                            |
    *==========================> *                            |
    |                        (BTI 3)                          |
    |                            |     bti_at_cmd_res()       |
    |                            |            (OK)            |
    |                            |==========================> |
    |                            |                        (BPS 1)
    |       ACI_ABORT_CNF        |                            |
    |            (OK)            |                            |
    *==========================> *                            |
    |                        (BTI 4)                          |
    |                            |    bti_at_abort_res()      |
    |                            |            (OK)            |
    |                            |==========================> |
    |                            |                        (BPS 2)
    |                            |                            |
```

(BTI 1)
The Bluetooth Stack sends an AT command to BTI.

(ACI 1)
The AT command is forwarded to ACI.

(BTI 2)
The Bluetooth PS aborts the command with a call to bti_at_abort_ind().

(ACI 2)
The abort request is forwarded to ACI.

(BTI 3)
ACI confirms the pending command. Actually ACI should have aborted the command already, but it is assumed, that this message has been sent before the reception of the ACI_ABORT_REQ and is received with a delay.

(BPS 1)
BTI forwards the confirmation to the Bluetooth PS. This confirmation can be ignored by the bluetooth PS.

(BTI 4)
ACI confirms the abortion of the command.

(BPS 2)
BTI forwards the confirmation to the Bluetooth PS.

3.2.14 Collision of command abortion and confirmation on BT side

```
  ACI                        BTI                     Bluetooth PS
   |                           |                           |
   |                           |      bti_at_cmd_ind()     |
   |                           |<==========================|
   |                        (BTI 1)                        |
   |        ACI_CMD_REQ        |                           |
   *<========================= *                           |
(ACI 1)                        |                           |
   |        ACI_CMD_CNF        |                           |
   |            (OK)           |                           |
   *=========================> *                           |
   |                        (BTI 2)                        |
   |                           |      bti_at_cmd_res()     |
   |                           |            (OK)           |
   |                           |==========================>|
   |                           |                        (BPS 1)
   |                           |     bti_at_abort_ind()    |
   |                           |<==========================|
   |                        (BTI 3)                        |
   |       ACI_ABORT_REQ       |                           |
   *<========================= *                           |
(ACI 2)                        |                           |
   |                           |                           |
   |       ACI_ABORT_CNF       |                           |
   |            (OK)           |                           |
   *=========================> *                           |
   |                        (BTI 4)                        |
   |                           |     bti_at_abort_res()    |
   |                           |            (OK)           |
   |                           |==========================>|
   |                           |                        (BPS 2)
   |                           |                           |
```

(BTI 1)
The Bluetooth Stack sends an AT command to BTI.

(ACI 1)
The AT command is forwarded to ACI.

(BTI 2)
ACI confirms the pending command.

(BPS 1)
BTI forwards the confirmation to the Bluetooth PS.

(BTI 3)
The Bluetooth PS aborts the command with a call to bti_at_abort_ind(). Actually the command is processed already and there is no need for an abortion, but it is assumed, that this call has been started before the reception of the ACI_CMD_CNF and is processed with a delay.

(ACI 2)
The abort request is forwarded to ACI.

(BTI 4)
ACI confirms the abortion of the command even if there is no command running in the ACI.

(BPS 2)
BTI forwards the confirmation to the Bluetooth PS.

3.2.15 Collision of command abortion and unsolicited result on GSM side

```
  ACI                         BTI                     Bluetooth PS
   |                           |                            |
   |                           |      bti_at_cmd_ind()      |
   |                           |<===========================|
   |                        (BTI 1)                         |
   |        ACI_CMD_REQ        |                            |
   *<==========================*                            |
(ACI 1)                        |                            |
   |                           |     bti_at_abort_ind()     |
   |                           |<===========================|
   |                        (BTI 2)                         |
   |       ACI_ABORT_REQ       |                            |
   *<==========================*                            |
(ACI 2)                        |                            |
   |                           |                            |
   |        ACI_CMD_IND        |                            |
   *==========================>*                            |
   |                        (BTI 3)                         |
   |                           |      bti_at_cmd_req()      |
   |                           |===========================>|
   |                           |                         (BPS 1)
   |       ACI_ABORT_CNF       |                            |
   |            (OK)           |                            |
   *==========================>*                            |
   |                        (BTI 4)                         |
   |                           |     bti_at_abort_res()     |
   |                           |            (OK)            |
   |                           |===========================>|
   |                           |                         (BPS 2)
   |                           |                            |
```

(BTI 1)
The Bluetooth Stack sends an AT command to BTI.

(ACI 1)
The AT command is forwarded to ACI.

(BTI 2)
The Bluetooth PS aborts the command with a call to bti_at_abort_ind().

(ACI 2)
The abort request is forwarded to ACI.

(BTI 3)
ACI sends an unsolicited result to BTI. It may be, that this message has been sent before the reception of the ACI_ABORT_REQ and is received with a delay. This result will not be acknowledged by the BT stack.

(BPS 1)
BTI forwards the unsolicited result to the Bluetooth PS.

(BTI 4)
ACI confirms the abortion of the command.

(BPS 2)
BTI forwards the confirmation to the Bluetooth PS.

3.2.16 Collision of command abortion and unsolicited result on BT side

```
  ACI                         BTI                    Bluetooth PS
   |                           |                           |
   |                           |     bti_at_cmd_ind()      |
   |                           |<==========================|
   |                        (BTI 1)                        |
   |        ACI_CMD_REQ        |                           |
   *<========================= *                           |
(ACI 1)                        |                           |
   |                           |                           |
   |        ACI_CMD_IND        |                           |
   *=========================> *                           |
   |                        (BTI 2)                        |
   |                           |     bti_at_cmd_req()      |
   |                           |==========================>|
   |                           |                        (BPS 1)
   |                           |                           |
   |                           |    bti_at_abort_ind()     |
   |                           |<==========================|
   |                        (BTI 3)                        |
   |       ACI_ABORT_REQ       |                           |
   *<========================= *                           |
(ACI 2)                        |                           |
   |       ACI_ABORT_CNF       |                           |
   |           (OK)            |                           |
   *=========================> *                           |
   |                        (BTI 4)                        |
   |                           |    bti_at_abort_res()     |
   |                           |            (OK)           |
   |                           |==========================>|
   |                           |                        (BPS 2)
   |                           |                           |
```

(BTI 1)
The Bluetooth Stack sends an AT command to BTI.

(ACI 1)
The AT command is forwarded to ACI.

(BTI 2)
ACI sends an unsolicited result to BTI. Because of the abortion this result will not be acknowledged by the BT stack.

(BPS 1)
BTI forwards the unsolicited result to the Bluetooth PS.

(BTI 3)
The Bluetooth PS aborts the command with a call to bti_at_abort_ind(). Usually the BT stack would confirm the unsolicited result first, but it may be, that this call has been started before the reception of the unsolicited result and is processeced with a delay.

(ACI 2)
The abort request is forwarded to ACI.

(BTI 4)
ACI confirms the abortion of the command.

(BPS 2)
BTI forwards the confirmation to the Bluetooth PS.

3.3 Interface to the DTI SAP

Every data transmission through the BTI is controlled by flow control, so that it can be ensured that both sides can receive the data and no buffer overflow occurs.

Before any data primitive is allowed to be sent, the corresponding side has to indicate that it is ready to receive data by sending a flow control signal. After sending a data packet, the corresponding side has to indicate again that it is ready to receive data before sending another data packet.

3.3.1 The GSM PS sends data to the Bluetooth PS

```
Data Relay                      BTI                     Bluetooth PS
    |                            |                           |
    |                            |    bti_data_ready_cnf()   |
    |                            |<==========================|
    |                         (BTI 1)                        |
    |       DTI_READY_IND        |                           |
    *<========================== *                           |
 (REL 1)                         |                           |
    |        DTI_DATA_REQ        |                           |
    *==========================> *                           |
    |                         (BTI 2)                        |
    |                            |    bti_data_ready_req()   |
    |                            |==========================>|
    |                            |                         (BPS 1)
    |                            |                           |
```

(BTI 1)
The Bluetooth PS sends a ready indication to BTI.

(REL 1)
BTI forwards the ready indication to the data relay. Now the data relay is allowed to send one data packet to the Bluetooth PS

(BTI 2)
The data relay sends a data packet to BTI. The next data packet may be sent only after receiving another ready indication.

(BPS 1)
BTI forwards the data packet to the Bluetooth PS.

3.3.2 The Bluetooth PS sends data to the GSM PS

```
Data Relay                      BTI                     Bluetooth PS
    |                            |                           |
    |      DTI_GETDATA_REQ       |                           |
    *==========================> *                           |
    |                         (BTI 1)                        |
    |                            |    bti_data_ready_res()   |
    |                            |==========================>|
    |                            |                         (BPS 1)
    |                            |    bti_data_ready_ind()   |
    |                            |<==========================|
    |                         (BTI 2)                        |
    |        DTI_DATA_IND        |                           |
    *<========================== *                           |
 (REL 1)                         |                           |
    |                            |                           |
```

(BTI 1)
The data relay sends a DTI_GETDATA_REQ to BTI as a signal that it can accept a data packet.

(BPS 1)
BTI forwards the get data request to the Bluetooth PS. Now the Bluetooth PS is allowed to send one data packet to the GSM PS

(BTI 2)
The Bluetooth PS sends a data packet to BTI. The next data packet may be sent only after receiving another get data request.

(REL 1)
BTI forwards the data packet to the data relay.

Appendix 1: The Functional Interface

A 1.1 Types used at the BTI functional interface

```
#define BTI_PIN_LENGTH          16 /*bytes*/
#define BTI_BD_ADDR_LENGTH      6 /*bytes*/
#define BTI_BD_NAME_LENGTH 248 /*bytes*/
typedef enum
{
        BTI_HEADSET = 0x01,
        BTI_DIAL_UP
}T_BTI_DEVICE_TYPE;

typedef UINT16 T_BTI_SEARCH_TIME;

typedef UINT16 T_BTI_SEARCH_BREAK;

typedef UINT16 T_BTI_SCAN_TIME;

typedef UINT16 T_BTI_SCAN_BREAK;

typedef enum
{
        BTI_SECURITY_MODE_1 = 0x01,
        BTI_SECURITY_MODE_2,
        BTI_SECURITY_MODE_3
} T_BTI_SECURITY_MODE;

typedef UINT8 T_BTI_ATTEMPTS;

typedef UINT8 T_BTI_MAX_RING;

typedef
{
        BTI_PARK_MODE_ON,
        BTI_PARK_MODE_OFF
} T_BTI_PARK_MODE;

typedef
{
        BTI_BONDING_ON,
        BTI_BONDING_OFF
}T_BTI_BONDIG_MODE;


typedef struct
{
        T_BTI_SEARCH_TIME       search_time,
        T_BTI_SEARCH_BREAK      search_break,
        T_BTI_SCAN_TIME         scan_time,
        T_BTI_SCAN_BREAK        scan_break,
        T_BTI_SECURITY_MODE     security_mode,
        T_BTI_ATTEMPTS          connect_attempts,
        T_BTI_MAX_RING          max_ring,
        T_BTI_PARK_MODE         park_mode,
        T_BTI_BONDING_MODE      bonding,
}T_BTI_CONFIG;
```

```
typedef
{
            BTI_OK,
            BTI_NOK
}T_BTI_RESULT;

typedef UINT8 T_BTI_BD_ADDR;
typedef UINT8 T_BTI_PIN;

typedef enum
{
}T_BTI_REQ_ID;

typedef enum
{
}T_BTI_RES_ID;

typedef enum
{
}T_BTI_CNF_ID;

typedef enum
{
}T_BTI_IND_ID;

typedef UINT8 T_BTI_BD_NAME;

typedef ENUM
{
            BTI_TIMEOUT = 0x01
} T_BTI_ERROR_CAUSE;

typedef enum
{
            BTI_CONNECTION_ESTABLISHED = 0x01,
            BTI_CONNECTION_RELEASED
}T_BTI_AUDIO_STATE;
```

A 1.2 Callback Functions of the BT side, provided by the BT protocol stack

A 1.2.1 bti_init_profile_req()

Prototype:

```
void bti_init_profile_req(T_BTI_DEVICE_TYPE device, T_BTI_CONFIG config);
```

Parameters:

device: This parameter indicates what profile is being initialised. BTI_HEADSET/BTI_DIAL_UP

config is a structure consisting of the following parameters.

search_time: This sets the duration of an inquiry.

search_break: This sets the time between two inquires.

scan_time: This sets the scanning time (ingoing connections) NO_SCAN = 0xFF

scan_break: This sets the time between scanning.

security_mode: This sets the security mode to be used as default. BTI_SECURITY_MODE_1/ BTI_SECURITY_MODE_2/ BTI_SECURITY_MODE_3

connect_attempts: This sets the maximum connection attempts allowed

max_ring: This sets the maximum number of RING to be sent to the remote device.

park_mode: BTI_PARK_MODE_ON/BTI_PARK_MODE_OFF

bonding_mode: BTI_BONDING_ON/BTI_BONDING_OFF

Description:

This is used to initialise profiles.

A 1.2.2 bti_deinit_profile_req()

Prototype:

void bti_deinit_profile_req(T_BTI_DEVICE_TYPE device);

Parameters:

device: This indicates the device to de-initialise BTI_HEADSET/BTI_DIAL_UP

Description:

This is used to de-initialise a profile

A 1.2.3 bti_device_search_req()

Prototype:

void bti_device_search_req(T_BTI_DEVICE_TYPE device);

Parameters:

device: This indicates what type of devices to search for BTI_HEADSET/BTI_DIAL_UP

Description:

This is used for starting an inquiry for devices.

A 1.2.4 bti_connect_device_req()

Prototype:

void bti_connect_device_req(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH]);

Parameters:

device: type of device BTI_HEADSET/BTI_DIAL_UP

bd_addr: Bluetooth device address.

Description:

This is used to request connection to a device.

A 1.2.5 bti_connect_device_res()

Prototype:

void bti_connect_device_res(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH], T_BTI_RESULT connect_accepted);

Parameters:

device: type of device BTI_HEADSET/BTI_DIAL_UP

bd_addr: Bluetooth device address.

connect_accepted: This is the actual response from the MMI (BTI_OK/BTI_NOK)

Description:

This is the response to bti_connect_device_ind

A 1.2.6 bti_disconnect_device_req()

Prototype:

void bti_disconnect_device_req(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH]);

Parameters:

device: type of device BTI_HEADSET/BTI_DIAL_UP

bd_addr: Bluetooth device address.

Description:

This is used to request disconnection from a device.

A 1.2.7 bti_transfer_audio_in_req()

Prototype:

void bti_transfer_audio_in_req(T_BTI_DEVICE_TYPE device);

Parameters:

device: type of device BTI_HEADSET

Description:

A 1.2.8 bti_transfer_audio_out_req()

Prototype:

void bti_transfer_audio_out_req(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH]);

Parameters:

device: type of device BTI_HEADSET

bd_addr: Bluetooth device address.

Description:

This is used to transfer the audio from the loudspeaker in the mobile phone to the headset.

A 1.2.9 bti_pin_res()

Prototype:

void bti_pin_res(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH], T_BTI_PIN pin_code[BTI_PIN_LENGTH]);

Parameters:

device: type of device BTI_HEADSET

bd_addr: Bluetooth device address.

pin_code: PIN of the device

Description:

This is used to respond to an bti_pin_ind.

A 1.2.10 bti_reconfig_profile_req()

Prototype:

void bti_reconfig_profile_req(T_BTI_DEVICE_TYPE device, T_BTI_CONFIG config);

Parameters:

device: This parameter indicates what profile is being initialised. BTI_HEADSET/BTI_DIAL_UP

config is a structure consisting of the following parameters.

search_time: This sets the duration of an inquiry.

search_break: This sets the time between two inquiries.

scan_time: This sets the scanning time (ingoing connections) NO_SCAN = 0xFF

scan_break: This sets the time between scanning.

security_mode: This sets the security mode to be used as default. BTI_SECURITY_MODE_1/ BTI_SECURITY_MODE_2/ BTI_SECURITY_MODE_3

connect_attempts: This sets the maximum connection attempts allowed

max_ring: This sets the maximum number of RING to be sent to the remote device.

park_mode: BTI_PARK_MODE_ON/BTI_PARK_MODE_OFF

bonding_mode: BTI_BONDING_ON/BTI_BONDING_OFF

Description:

This is used to reconfigure the a bluetooth profile.

A 1.2.11 bti_device_control_req()

Prototype:

void bti_device_control_req(T_BTI_DEVICE_TYPE device_type, T_BTI_REQ_ID request_identifier, unsigned char *control_data, unsigned short length);

Parameters:

device_type:	This indicates what profile this control information is for (headset, dial-up etc)
request_identifier:	type of request
data:	Pointer to the block of data corresponding to the request
length:	Length of data in bytes

Description:

This function is used by the GSM side to request something from the Bluetooth profile. This is used for accessing vendor specific functionality.

A 1.2.12 bti_device_control_res()

Prototype:

void bti_device_control_res(T_BTI_DEVICE_TYPE device_type, T_BTI_RES_ID response_identifier, unsigned char *control_data, unsigned short length);

Parameters:

device_type:	This indicates what profile this control information is for (headset, dial-up etc)
response_identifier:	type of response
data:	Pointer to the block of data corresponding to the response
length:	Length of data in bytes

Description:

This function is used by the GSM when a response to an indication is required for vendor specific functionality.

A 1.2.13 bti_at_init_req()

Prototype:

```
void bti_at_init_req(unsigned long max_blocksize);
```

Parameters:

max_blocksize: This parameter gives the number of bytes, which can be handled in a call of bti_data_ready_req() and bti_data_ready_ind(). It is proposed by the GSM side and negotiated either to the same value or to a lower value by the BT side in the confirming bti_at_init_cnf().

Description:

This function is called once after startup to register the GSM stack to the ATP.

A 1.2.14 bti_at_deinit_res()

Prototype:

```
void bti_at_deinit_res(void);
```

Parameters:

none

Description:

This function is used as a response to bti_at_deinit_ind.

A 1.2.15 bti_at_open_port_res()

Prototype:

```
void bti_at_open_port_res(T_BTI_SW_ENTITY_ID initiator_id, T_BTI_PORT_NB initiator_port_nb, T_BTI_ACK ackflg);
```

Parameters:

initiator_id:	ID number
initiator_port_nb:	Port number
ackflg:	BTI_ACK = opening of port successful BTI_NAK = opening of port failed

Description:

This function is the acknowledgement for a previous call of bti_at_open_port_ind(). The ID number and port number are taken from the call of the bti_at_open_port_ind() function.

A 1.2.16 bti_at_close_port_res()

Prototype:

```
void bti_at_close_port_res(T_BTI_PORT_NB initiator_port_nb);
```

Parameters:

initiator_port_nb:	Port number

Description:

This function is the acknowledgement for a previous call of bti_at_close_port_ind(). Closing a port must always be successful. The confirmation is there to indicate the completion of the operation.

A 1.2.17 bti_at_cmd_res()

Prototype:

void bti_at_cmd_res(T_BTI_PORT_NB initiator_port_nb, char *result);

Parameters:

initiator_port_nb:	Port number
result:	Text containing the result

Description:

This function is the acknowledgement for a previous call of bti_at_cmd_ind(). It is a signal, that a new AT command for the same port may be sent by the BT side to the GSM stack. The result is given as a pointer to a NULL terminated string. The string does not contain any <CR> or <LF> characters as a delimiter. Possible result strings are:

- OK
- NO CARRIER
- CONNECT xxxx (*xxxx = bit rate or blank)*
- BUSY
- NO ANSWER
- ERROR

All results or responses must be acknowledged by the BT stack, before a new response or result may be sent to the same port. Therefore after sending a result or an unsolicited response (A 1.2.18 bti_at_cmd_req()) the GSM stack has to wait for a corresponding bti_at_cmd_cnf() before it can send another result or response to the same port.

A 1.2.18 bti_at_cmd_req()

Prototype:

void bti_at_cmd_req(T_BTI_PORT_NB initiator_port_nb, char *response);

Parameters:

initiator_port_nb:	Port number
response:	Text containing the response

Description:

This function sends an unsolicited response from the GSM stack to the BT stack. The response is given as a pointer to a NULL terminated string. The string does not contain any <CR> or <LF> characters as a delimiter.

All results or responses must be acknowledged by the BT stack, before a new response or result may be sent to the same port. Therefore after sending an unsolicited response or a result (A 1.2.17 bti_at_cmd_res()) the GSM stack has to wait for a corresponding bti_at_cmd_cnf() before it can send another result or response to the same port.

A 1.2.19 bti_at_abort_res()

Prototype:

void bti_at_abort_res(T_BTI_PORT_NB initiator_port_nb);

Parameters:

initiator_port_nb: Port number

Description:

With this function GSM acknowledges a previous call to bti_at_abort_ind(). It shows, that the abortion is completed succussfully.

A 1.2.20 bti_data_ready_res()

Prototype:

void bti_data_ready_res(T_BTI_PORT_NB initiator_port_nb);

Parameters:

initiator_port_nb:	Port number

Description:

With this function GSM gives a signal to BT that it is ready to accept a block of data in a call of bti_data_ready_ind().

A 1.2.21 bti_data_ready_req()

Prototype:

void bti_data_ready_req(T_BTI_PORT_NB initiator_port_nb, unsigned long length);

Parameters:

initiator_port_nb:	Port number
length:	Length of data in bytes

Description:

With this function GSM indicates, that a block of data is available to BT. After sending data GSM has to wait for a confirmation in a bti_data_ready_cnf() before another block of data may be sent.

A 1.2.22 bti_get_bt_data()

Prototype:

void bti_get_bt_data (T_BTI_PORT_NB initiator_port_nb, char* buffer, unsigned long length);

Parameters:

initiator_port_nb:	Port number
buffer:	Pointer at the buffer where data can be copied into
length:	Length of data in bytes

Description:

This function is called by GSM in order to get BT data.

A 1.3 Callback Functions of the GSM side, provided by the GSM protocol stack

A 1.3.1 bti_init_profile_cnf()

Prototype:

void bti_init_profile_cnf(T_BTI_DEVICE_TYPE device, T_BTI_RESULT result);

Parameters:

device: This indicates the device type BTI_HEADSET/BTI_DIAL_UP

result: This indicates the result if the profile initialisation BTI_OK/BTI_NOK.

Description:

This is used to return the result of a profile initialisation.

A 1.3.2 bti_deinit_profile_cnf()

Prototype:

void bti_deinit_profile_cnf(T_BTI_DEVICE_TYPE device, T_BTI_RESULT result);

Parameters:

device: this indicates the device which is de-initialised BTI_HEADSET/BTI_DIAL_UP

result: this is the result BTI_OK/BTI_NOK

Description:

This is used to confirm the de-initialisation of a profile.

A 1.3.3 bti_device_search_cnf()

Prototype:

void bti_device_search_cnf(T_BTI_DEVICE_TYPE device);

Parameters:

device: This indicates the type of device BTI_HEADSET/BTI_DIAL_UP

Description:

This is used to confirm when an inquiry for the specified device type is finished

A 1.3.4 bti_device_found_ind()

Prototype:

void bti_device_found_ind(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH], T_BTI_BD_NAME name[BTI_BD_NAME_LENGTH]);

Parameters:

device: The type of device found BTI_HEADSET/BTI_DIAL_UP

bd_addr: Bluetooth device address of the found device

name: The name of the found device

Description:

This is used to indicate when a device is found. This can be received during requested inquiry.

A 1.3.5 bti_connect_device_cnf()

Prototype:

void bti_connect_device_cnf(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH], T_BTI_RESULT result, T_BTI_ERROR_CAUSE cause);

Parameters:

device: device type BTI_HEADSET/BTI_DIAL_UP bd_addr: Bluetooth device address

result: result BTI_OK/BTI_NOK

cause: BT_TIMEOUT

Description:

A 1.3.6 bti_connect_device_ind()

Prototype:

void bti_connect_device_ind(T_BTI_DEVICE_TYPE device, T_BTI_BD_NAME name[BTI_BD_NAME_LENGTH], T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH]);

Parameters:

device: BTI_HEADSET/BTI_DIAL_UP

name: Device name

bd_addr: Bluetooth device address

Description:

This is used to indicate that a remote device want's to connect.

A 1.3.7 bti_disconnect_device_cnf()

Prototype:

void bti_disconnect_device_cnf(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH]);

Parameters:

device: BTI_HEADSET/BTI_DIAL_UP

bd_addr: Bluetooth device address

Description:

A 1.3.8 bti_disconnect_device_ind()

Prototype:

void bti_disconnect_device_ind(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH], T_BTI_ERROR_CAUSE cause);

Parameters:

device: BTI_HEADSET/BTI_DIAL_UP

bd_addr: Bluetooth device address

cause: Reason for the disconnection

Description:

This is used to indicate disconnection to the MMI

A 1.3.9 bti_audio_connection_ind()

Prototype:

void bti_audio_connection_ind(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH], T_BTI_AUDIO_STATE connection_state, T_BTI_ERROR_CAUSE cause);

Parameters:

device: BTI_HEADSET/BTI_DIAL_UP

bd_addr: Bluetooth device address

connection_state: This indicates the state of the audio connection
(BTI_CONNECTION_ESTABLISHED/BTI_CONNECTION_RELEASED)

cause: reason for disconnection

Description:

This is used to indicate changes in the state of the audio connection.

A 1.3.10 bti_transfer_audio_in_cnf()

Prototype:

void bti_transfer_audio_in_cnf(T_BTI_DEVICE_TYPE device, T_BTI_RESULT result);

Parameters:

device: BTI_HEADSET

result: BTI_OK/BTI_NOK

Description:

This is an confirmation to bti_transfer_audio_in_req.

A 1.3.11 bti_transfer_audio_out_cnf()

Prototype:

void bti_transfer_audio_out_cnf(T_BTI_DEVICE_TYPE device, T_BTI_RESULT result);

Parameters:

device: BTI_HEADSET

result: BTI_OK/BTI_NOK

Description:

This is a confirmation to bti_transfer_audio_out_req.

A 1.3.12 bti_pin_ind()

Prototype:

void bti_pin_ind(T_BTI_DEVICE_TYPE device, T_BTI_BD_ADDR bd_addr[BTI_BD_ADDR_LENGTH]);

Parameters:

device: BTI_HEADSET/BTI_DIAL_UP

bd_addr: Bluetooth device address

Description:

This is used to indicate to the MMI that a PIN is required.

A 1.3.13 bti_reconfig_profile_cnf()

Prototype:

void bti_reconfig_profile_cnf(T_BTI_DEVICE_TYPE device, T_BTI_RESULT result);

Parameters:

device: This indicates the device type BTI_HEADSET/BTI_DIAL_UP

result: This indicates the result if the profile initialisation BTI_OK/BTI_NOK.

Description:

This is used to return the result of a reconfiguration.

A 1.3.14 bti_device_control_cnf()

Prototype:

void bti_device_control_cnf(T_BTI_DEVICE_TYPE device_type, T_BTI_CNF_ID confirm_identifier, unsigned char *control_data, unsigned short length);

Parameters:

device_type: This indicates what profile this control information is for BTI_HEADSET/BTI_DIAL_UP

confirm_identifier: type of confirmation

data: Pointer to the block of data corresponding to the confirmation

length: Length of data in bytes

Description:

This function is used by the bluetooth profile when a request has been processed and a confirmation is required. This is for vendor specific functionality.

A 1.3.15 bti_device_control_ind()

Prototype:

void bti_device_control_ind(T_BTI_DEVICE_TYPE device_type, T_BTI_IND_ID indication_identifier, unsigned char *control_data, unsigned short length);

Parameters:

device_type: This indicates what profile this control information is for BTI_HEADSET/BTI_DIAL_UP

indication_identifier: type of indication

data: Pointer to the block of data corresponding to the request

length: Length of data in bytes

Description:

This function is used by the bluetooth profile when an action is required from the MMI or to inform the MMI about changes in the state of the profile. This is used for vendor specific functionality.

A 1.3.16 bti_at_init_cnf()

Prototype:

void bti_at_init_cnf(T_BTI_ACK ackflg, unsigned long max_blocksize);

Parameters:

ackflg: BTI_ACK = initialisation successful
BTI_NAK = initialisation failed

max_blocksize: This parameter gives the negotiated number of bytes, which can be handled in a call of bti_data_ready_req() and bti_data_ready_ind(). It is proposed by the GSM side in the call of bti_at_init_req() and negotiated either to the same value or to a lower value by the BT side.

Description:

This function returns the result of a previous call of bti_at_init_req().

With the parameter max_blocksize BT returns the negotiated value for the maximum block size. It may be the proposed value in the call of bti_at_init_req() or a lower value.

A 1.3.17 bti_at_deinit_ind()

Prototype:

void bti_at_deinit_ind(void);

Parameters:

none

Description:

This function indicates deinitialization

A 1.3.18 bti_at_open_port_ind()

Prototype:

void bti_at_open_port_ind(T_BTI_SW_ENTITY_ID initiator_id, T_BTI_PORT_NB initiator_port_nb);

Parameters:

initiator_id:	ID number
initiator_port_nb:	Port number

Description:

This function is called by the BT stack in order to open a port. The ID number and port number are stored in the BTI. The port number is used for all subsequent calls to identify the port. Moreover the ID number will be returned in the call of the bti_at_open_port_res() function. There may be only one pending indication to open a port at a time, therefore BT has to wait for the corresponding bti_at_open_port_res() before opening another port.

A 1.3.19 bti_at_close_port_ind()

Prototype:

void bti_at_close_port_ind(T_BTI_PORT_NB initiator_port_nb);

Parameters:

initiator_port_nb:	Port number

Description:

This function is called by the BT stack in order to close a port. There may be only one pending indication to close a port at a time, therefore BT has to wait for the corresponding bti_at_close_port_res() before closing another port.

A 1.3.20 bti_at_cmd_ind()

Prototype:

void bti_at_cmd_ind(T_BTI_PORT_NB initiator_port_nb, char *cmd);

Parameters:

initiator_port_nb:	Port number
cmd:	Text containing the command

Description:

This function is called by the BT stack in order to send an AT command to the GSM stack. The command is given as a pointer to a NULL terminated string. The string should not contain any <CR> or <LF> characters as a command delimiter.

After a call to bti_at_cmd_ind() the BT side has to wait for the corresponding bti_at_cmd_res() call, before calling bti_at_cmd_ind() again.

A 1.3.21 bti_at_cmd_cnf()

Prototype:

void bti_at_cmd_cnf(T_BTI_PORT_NB initiator_port_nb);

Parameters:

initiator_port_nb:	Port number

Description:

This function is called by the BT stack in order to acknowledge a previous call of bti_at_cmd_req().

A 1.3.22 bti_at_abort_ind()

Prototype:

void bti_at_abort_ind(T_BTI_PORT_NB initiator_port_nb);

Parameters:

initiator_port_nb:	Port number

Description:

With this function the Bluetooth PS aborts a running command.

A 1.3.23 bti_data_ready_cnf()

Prototype:

void bti_data_ready_cnf(T_BTI_PORT_NB initiator_port_nb);

Parameters:

initiator_port_nb:	Port number

Description:

This function is called by the BT stack in order to signal, that it is ready to accept a block of data in a call of bti_data_ready_req().

A 1.3.24 bti_data_ready_ind()

Prototype:

void bti_data_ready_ind(T_BTI_PORT_NB initiator_port_nb, unsigned long length);

Parameters:

initiator_port_nb:	Port number
length:	Length of data in bytes

Description:

With this function BT sends an indication that a block of data is available to GSM. After sending data BT has to wait for a confirmation in a bti_data_ready_res() before another block of data may be sent.

A 1.3.25 bti_get_gsm_data()

Prototype:

void bti_get_gsm_data(T_BTI_PORT_NB initiator_port_nb, char* buffer[], unsigned long length);

Parameters:

initiator_port_nb:	Port number
buffer:	Pointer to the available buffer
length:	Length of data in bytes

Description:

This function is called by BT in order to get data from GSM.

Appendix 2: MSCs of function calls

These MSCs are an extract of Chapter 3 (Protocol). They show only the interaction at the functional interface without consideration of the primitive exchange between ACI and BTI.

A 2.1 Interface to the BTP SAP

A 2.1.1 Initialisation of a Bluetooth Profile

```
  MMI                     Bluetooth PS
   |                            |
   |   bti_init_profile_req()   |
   |==========================> |
   |                         (BPS 1)
   |   bti_init_profile_cnf()   |
   |<========================== |
(MMI 1)                         |
   |                            |
```

(BPS 1)
MMI requests initialization from the BPS

(MMI 1)
BPS confirmes the initialization

A 2.1.2 Deinitialisation of a Bluetooth Profile

```
  MMI                     Bluetooth PS
   |                            |
   |                            |
   |  bti_deinit_profile_req()  |
   |==========================> |
   |                         (BPS 1)
   |                            |
   |  bti_deinit_profile_cnf()  |
   |<========================== |
(MMI 1)                         |
   |                            |
```

(BPS 1)
MMI request deinitialization of a Bluetooth profile

(MMI 1)
The deinitialization is confirmed

A 2.1.3 Reconfiguration of a profile

```
   MMI                     Bluetooth PS
    |                            |
    | bti_reconfig_profile_req() |
    |==========================> |
    |                         (BPS 1)
    |                            |
    | bti_reconfig_profile_cnf() |
    |<========================== |
 (MMI 1)                         |
    |                            |
```

(BPS 1)
The MMI requests reconfiguration (first configuration is done during initialisation) of the profile

(MMI 1)
The BTP confirmes the reconfiguration

A 2.1.4 Searching for available devices

```
   MMI                     Bluetooth PS
    |                            |
    |                            |
    |  bti_device_search_req()   |
    |==========================> |
    |                         (BPS 1)
    |                            |
    |   bti_device_found_ind()   |
    |<========================== |
 (MMI 1)                         |
    |                            |
    |                            |
    |   bti_device_found_ind()   |
    |<========================== |
 (MMI 2)                         |
    |                            |
    |                            |
    |  bti_device_search_cnf()   |
    |<========================== |
 (MMI 3)                         |
    |                            |
```

(BPS 1)
MMI requests the BPS to search for available devices (of a specific type).

(MMI 1)
A device is found

(MMI 2)
Another device is found

(MMI 3)
BPS confirmes when the search is finished. During this search two devices of the requested type was found.

A 2.1.5 Outgoing connection

```
  MMI                     Bluetooth PS
   |                            |
   |  bti_connect_device_req()  |
   |===========================>|
   |                         (BPS 1)
   |                            |
   |  bti_connect_device_cnf()  |
   |<========================== |
(MMI 1)                         |
   |                            |
```

(BPS 1)
MMI request connection to the device

(MMI 1)
The connection is confirmed

A 2.1.6 Ingoing connection

```
  MMI                     Bluetooth PS
   |                            |
   |  bti_connect_device_ind()  |
   |<========================== |
(MMI 1)                         |
   |                            |
   |  bti_connect_device_res()  |
   |===========================>|
   |                         (BPS 1)
   |                            |
```

(MMI 1)
BPS indicates that a remote device wants to establish a connection

(BPS 1)
The MMI responds

A 2.1.7 Entering of PIN

```
  MMI                     Bluetooth PS
   |                            |
   |       bti_pin_ind()        |
   |<========================== |
(MMI 1)                         |
   |                            |
   |                            |
   |       bti_pin_res()        |
   |===========================>|
   |                         (BPS 1)
   |                            |
```

(MMI 1)
BPS indicates that a pin code is needed (if bonding is enabled)

(BPS 1)
The MMI responds with the pin code

A 2.1.8 Audio connection/disconnection/fail

```
 ´MMI                     Bluetooth PS
   |                            |
   | bti_audio_connection_ind() |
   |<========================== |
(MMI 1)                         |
   |                            |
```

(MMI 1)
BPS indicates that ands audio connection to the device is setup or closed down or failed

A 2.1.9 Transfer connection out

```
  MMI                     Bluetooth PS
   |                            |
   |                            |
   |bti_transfer_audio_out_req()|
   |==========================> |
   |                        (BPS 1)
   |                            |
   |bti_transfer_audio_out_cnf()|
   |<========================== |
(MMI 1)                         |
```

(BPS 1)
MMI request transfer of the audio from the loudspeaker in the handset to the device

(MMI 1)
A connection to the headset is established and an audio connection is established as well. When this has happened the transfer is confirmed.

A 2.1.10 Transfer connection in

```
  MMI                     Bluetooth PS
   |                            |
   |bti_transfer_audio_in_req() |
   |==========================> |
   |                        (BPS 1)
   |                            |
   |bti_transfer_audio_in_cnf() |
   |<========================== |
(MMI 1)                         |
```

(BPS 1)
MMI request transfer of the audio from the headset to the loudspeaker in the handset

(MMI 1)
The transfer is confirmed

A 2.1.11 Disconnecting from a device initiated by SMI

```
    MMI                     Bluetooth PS
     |                            |
     |                            |
     |bti_disconnect_device_req() |
     |==========================> |
     |                         (BPS 1)
     |                            |
     |bti_disconnect_device_cnf() |
     |<========================== |
  (MMI 1)                         |
     |                            |
```

(BPS 1)
MMI request disconnection from the device

(MMI 1)
The disconnection is confirmed

A 2.1.12 Disconnecting from a device initiated by BPS

```
    MM        Bluetooth PS
     |                            |
     |bti_disconnect_device_ind() |
     |<========================== |
   (MM1)                          |
     |                            |
```

(MMI 1)
BPS indicates disconnection from the device

A 2.1.13 Control of a device profile initiated by the MMI

```
    MMI                     Bluetooth PS
     |                            |
     |  bti_device_control_req()  |
     |==========================> |
     |                         (BPS 1)
     |                            |
     |  bti_device_control_cnf()  |
     |<========================== |
  (MMI 1)                         |
     |                            |
```

(BPS 1)
MMI sends control information to the BPS.

(MMI 1)
BPS confirms the when the control request is processed.

A 2.1.14 Control of a device profile initiated by the device profile

```
  MMI                       Bluetooth PS
   |                            |
   |  bti_device_control_ind()  |
   |<========================== |
 (MMI 1)                        |
   |  bti_device_control_res()  |
   |==========================> |
   |                         (BPS 1)
```

(BPS 1)
The Bluetooth PS needs some action from the MMI, or informs the MMI about changes in the state of the device.

(MMI 1)
If a response from the SMI is required this is sent.

A 2.2 Interface to the ACI SAP

A 2.2.1 Initialisation

```
  GSM                        BPS
   |                            |
   |      bti_at_init_req()     |
   |==========================> |
   |                         (BPS 1)
   |      bti_at_init_cnf()     |
   |<========================== |
 (GSM 1)                        |
   |                            |
```

(BPS 1)
GSM wants to initialise the Bluetooth PS.

(GSM 1)
The Bluetooth PS signals to GSM, that the Initialisation was successful.

A 2.2.2 Opening a port

```
  GSM                        BPS
   |                            |
   |   bti_at_open_port_ind()   |
   |<========================== |
 (GSM 1)                        |
   |   bti_at_open_port_res()   |
   |==========================> |
   |                         (BPS 1)
   |                            |
```

(GSM 1)
The Bluetooth PS wants to open a port. Opening a port is only possible after initialisation.

(BPS 1)
GSM signals to the Bluetooth PS, that a port has been opened.

A 2.2.3 Closing a port

```
  GSM                          BPS
   |                            |
   |  bti_at_close_port_ind()   |
   |<========================== |
(GSM 1)                         |
   |  bti_at_close_port_res()   |
   |==========================> |
   |                         (BPS 1)
   |                            |
```

(GSM 1)
The Bluetooth PS wants to close a port.

(BPS 1)
GSM signals to the Bluetooth PS, that the port has been closed.

A 2.2.4 Deinitialisation

```
  GSM                          BPS
   |                            |
   |    bti_at_deinit_ind()     |
   |<========================== |
(GSM 1)                         |
   |    bti_at_deinit_res()     |
   |==========================> |
   |                         (BPS 1)
   |                            |
```

(BPS 1)
Deinitialisiation indication is sent.

(GSM 1)
Deinitialisation response.

A 2.2.5 Bluetooth sends an AT command

```
  GSM                          BPS
   |                            |
   |     bti_at_cmd_ind()       |
   |            (AT)            |
   |<========================== |
(GSM 1)                         |
   |     bti_at_cmd_res()       |
   |            (OK)            |
   |=========================>  |
   |                         (BPS 1)
   |                            |
```

(GSM 1)
Sending AT commands and responses is only possible after initialising and opening a port. The Bluetooth PS sends an AT command to GSM (“AT”). The next command may be sent only after receiveing the confirmation (BPS 1)

(BPS 1)
GSM returns a confirmation (“OK”) to the Bluetooth PS.

A 2.2.6 Bluetooth receives an unsolicited response

```
  GSM                          BPS
   |                            |
   |      bti_at_cmd_req()      |
   |           (+FCO)           |
   |==========================> |
   |                         (BPS 1)
   |      bti_at_cmd_cnf()      |
   |<========================== |
(GSM 2)                         |
   |                            |
```

(BPS 1)
GSM sends an unsolicited response to Bluetooth (+FCO). The next unsolicited response or confirmation of any pending AT command may be sent only after a call of bti_at_cmd_cnf() (GSM 2).

(GSM 2)
The Bluetooth PS confirms the response. No AT command or response is included in this confirmation.

A 2.2.7 Bluetooth establishes a voice call

```
  GSM                          BPS
   |                            |
   |      bti_at_cmd_ind()      |
   |          (ATD123;)         |
   |<========================== |
(GSM 1)                         |
   |      bti_at_cmd_res()      |
   |          (CONNECT)         |
   |==========================> |
   |                         (BPS 1)
   |                            |
```

(GSM 1)
The Bluetooth PS sends an dial command to GSM. The number must be terminated by a semicolon (‘;’) to establish a voice call (s. GSM 07.07).

(BPS 1)
GSM returns a confirmation for the establishment of the call (“CONNECT xxxx”)

A 2.2.8 Bluetooth receives an incoming call

```
   GSM                          BPS
     |                            |
     |      bti_at_cmd_req()      |
     |           (RING)           |
     |==========================> |
     |                        (BPS 1)
     |      bti_at_cmd_cnf()      |
     |<========================== |
  (GSM 1)                         |
     |                            |
     |      bti_at_cmd_ind()      |
     |           (ATA)            |
     |<========================== |
  (GSM 2)                         |
     |      bti_at_cmd_res()      |
     |         (CONNECT)          |
     |==========================> |
     |                        (BPS 2)
     |                            |
```

(BPS 1)
The GSM sends an unsolicited RING response to Bluetooth. The next unsolicited response or confirmation of any pending AT command may be sent only after reciving the bti_at_cmd_cnf (GSM 1).

(GSM 1)
The Bluetooth PS confirms the RING response. No AT command or response is included in this confirmation. In particular the following ATA command is not included in this confirmation. The reason is, that ervery IND must acknowledged by a RES and every REQ must be acknowledged by a CNF. Therefore a separate pair of function calls (bti_at_cmd_ind()/bti_at_cmd_res()) is used for ATA/CONNECT.

(GSM 2)
The Bluetooth PS sends an ATA command to GSM in order to receive the call.

(BPS 2)
GSM returns a confirmation for the establishment of the call (“CONNECT xxxx”)

A 2.2.9 AT command, which includes more than one response

Some AT commands produce additional responses before the final acknowledgement appears in the form of a OK or CONNECT response. The additional responses are sent in an bti_at_cmd_req(), whereas the final result is sent in an bti_at_cmd_res(). Therefore one or several bti_at_cmd_req()/bti_at_cmd_cnf() function pairs are inserted between the bti_at_cmd_ind() and bti_at_cmd_res(). An example for this are all query commands.

```
   GSM                          BPS
    |                            |
    |     bti_at_cmd_ind()       |
    |        (AT+CBST?)          |
    |<========================== |
(GSM 1)                          |
    |     bti_at_cmd_req()       |
    |       (+CBST=7,0,1)        |
    |==========================> |
    |                        (BPS 1)
    |                            |
    |     bti_at_cmd_cnf()       |
    |<========================== |
(GSM 2)                          |
    |     bti_at_cmd_res()       |
    |           (OK)             |
    |==========================> |
    |                        (BPS 2)
    |                            |
```

(GSM 1)
The Bluetooth PS sends a AT+CBST=? command to GSM to query the CBST parameters.

(BPS 1)
GSM sends the queried response in a call of bti_at_cmd_req() to the Bluetooth PS. The acknowledgement for the AT+CBST command is still pending and will be given in (GSM 2). GSM is allowd to send the next message (bti_at_cmd_req() or bti_at_cmd_res()) only after a call to bti_at_cmd_cnf().

(GSM 2)
The Bluetooth PS acknowledges the +CBST response. Now GSM can call bti_at_cmd_req() or bti_at_cmd_res() again.

(BPS 2)
GSM returns a confirmation for the AT+CBST command

A 2.2.10 Collision of an incoming and outgoing call as seen in GSM

```
   GSM                          BPS
    |                            |
    |     bti_at_cmd_req()       |
    |          (RING)            |
    |==========================> |
    |                        (BPS 1)
    |     bti_at_cmd_ind()       |
    |         (ATD123)           |
    |<========================== |
(GSM 1)                          |
    |                            |
    |     bti_at_cmd_cnf()       |
    |<========================== |
(GSM 2)                          |
    |     bti_at_cmd_res()       |
    |       (NO CARRIER)         |
    |==========================> |
    |                        (BPS 2)
    |                            |
```

(BPS 1)
GSM sends a RING response to the Bluetooth PS with a call to bti_at_cmd_req().

(GSM 1)
GSM receives a dial command from the Bluetooth PS. Usually the Bluetooth PS should not send a dial command after receiving a RING response, but this command may have been sent before receiving the RING response and is received now with a delay. GSM

can not reply immediatly, because the acknowledgment for the RING response is still pending. The reply will be given in BPS 2 after the call of bti_at_cmd_cnf().

(GSM 2)
The Bluetooth PS acknowledges the RING response. Now GSM can call bti_at_cmd_res() or bti_at_cmd_req() again.

(BPS 2)
GSM returns a confirmation for the ATD command. In this case the call is rejected with a NO CARRIER response, because an incoming call is pending.

A 2.2.11 Collision of an incoming and outgoing call as seen in the Bluetooth PS

```
   GSM                            BPS
     |                              |
     |      bti_at_cmd_ind()        |
     |          (ATD123)            |
     |<==========================   |
 (GSM 1)                            |
     |      bti_at_cmd_req()        |
     |           (RING)             |
     |==========================>   |
     |                          (BPS 1)
     |      bti_at_cmd_cnf()        |
     |<==========================   |
 (GSM 2)                            |
     |      bti_at_cmd_res()        |
     |        (NO CARRIER)          |
     |==========================>   |
     |                          (BPS 2)
     |                              |
```

(GSM 1)
GSM receives a dial command from the Bluetooth PS.

(BPS 1)
The Bluetooth PS receives a RING response from the GSM. Usually the GSM should not send a RING response after receiving a dial command, but this response may have been sent before receiving the dial command and is received now with a delay.

(GSM 2)
The Bluetooth PS acknowledges the RING response. Now GSM can call bti_at_cmd_res() or bti_at_cmd_req() again.

(BPS 2)
GSM returns a confirmation for the ATD command. In this case the call is rejected with a NO CARRIER response, because an incoming call is pending.

A 2.2.12 Abortion of an AT command

```
   GSM                            BPS
     |                              |
     |      bti_at_cmd_ind()        |
     |<==========================   |
 (GSM 1)                            |
     |                              |
     |     bti_at_abort_ind()       |
     |<==========================   |
 (GSM 2)                            |
     |     bti_at_abort_res()       |
     |            (OK)              |
     |==========================>   |
     |                          (BPS 1)
     |                              |
```

(GSM 1)
The Bluetooth Stack sends an AT command to GSM.

(GSM 2)
The Bluetooth PS aborts the command with a call to bti_at_abort_ind(). The running command is stopped. GSM will not send any result of the interrupted command to the Bluetooth PS.

(BPS 1)
GSM confirms the abortion of the command.

A 2.2.13 Collision of command abortion and confirmation on GSM side

```
  GSM                         BPS
   |                            |
   |     bti_at_cmd_ind()       |
   |<========================== |
 (GSM 1)                        |
   |                            |
   |    bti_at_abort_ind()      |
   |<========================== |
 (GSM 2)                        |
   |     bti_at_cmd_res()       |
   |           (OK)             |
   |==========================> |
   |                         (BPS 1)
   |    bti_at_abort_res()      |
   |           (OK)             |
   |==========================> |
   |                         (BPS 2)
   |                            |
```

(GSM 1)
The Bluetooth Stack sends an AT command to GSM.

(GSM 2)
The Bluetooth PS aborts the command with a call to bti_at_abort_ind().

(BPS 1)
GSM confirms the pending command. Actually GSM should have aborted the command already, but it is assumed, that this function call has been started before the call of the bti_at_abort_ind() and is processed with a delay. This confirmation may be ignored by the Bluetooth PS.

(BPS 2)
GSM confirms the abortion of the command.

A 2.2.14 Collision of command abortion and confirmation on BT side

```
  GSM                          BPS
   |                            |
   |      bti_at_cmd_ind()      |
   |<========================== |
(GSM 1)                         |
   |      bti_at_cmd_res()      |
   |           (OK)             |
   |==========================> |
   |                         (BPS 1)
   |                            |
   |     bti_at_abort_ind()     |
   |<========================== |
(GSM 2)                         |
   |     bti_at_abort_res()     |
   |           (OK)             |
   |==========================> |
   |                         (BPS 2)
   |                            |
```

(GSM 1)
The Bluetooth Stack sends an AT command to GSM.

(BPS 1)
GSM confirms the pending command.

(GSM 2)
The Bluetooth PS aborts the command with a call to bti_at_abort_ind(). Actually the command is processed already and there is no need for an abortion, but it is assumed, that this call has been started before the call of bti_at_cmd_res() and is processed with a delay.

(BPS 2)
GSM confirms the abortion of the command even if there is no command running in the GSM.

A 2.2.15 Collision of command abortion and unsolicited result on GSM side

```
  GSM                          BPS
   |                            |
   |      bti_at_cmd_ind()      |
   |<========================== |
(GSM 1)                         |
   |                            |
   |     bti_at_abort_ind()     |
   |<========================== |
(GSM 2)                         |
   |                            |
   |      bti_at_cmd_req()      |
   |==========================> |
   |                         (BPS 1)
   |     bti_at_abort_res()     |
   |           (OK)             |
   |==========================> |
   |                         (BPS 2)
   |                            |
```

(GSM 1)
The Bluetooth Stack sends an AT command to GSM.

(GSM 2)
The Bluetooth PS aborts the command with a call to bti_at_abort_ind().

(BPS 1)
GSM sends an unsolicited result to the Bluetooth PS. It may be, that this result has been sent before the call of bti_at_abort_ind() and is processed with a delay. This result will not be acknowledged by the BT stack.

(BPS 2)
GSM confirms the abortion of the command.

A 2.2.16 Collision of command abortion and unsolicited result on BT side

```
   GSM                           BPS
    |                             |
    |      bti_at_cmd_ind()       |
    |<=========================== |
 (GSM 1)                          |
    |                             |
    |      bti_at_cmd_req()       |
    |===========================> |
    |                          (BPS 1)
    |                             |
    |     bti_at_abort_ind()      |
    |<=========================== |
 (GSM 2)                          |
    |     bti_at_abort_res()      |
    |            (OK)             |
    |===========================> |
    |                          (BPS 2)
    |                             |
```

(GSM 1)
The Bluetooth Stack sends an AT command to GSM.

(BPS 1)
GSM sends an unsolicited result to the Bluetooth PS. Because of the abortion this result will not be acknowledged by the BT stack.

(GSM 2)
The Bluetooth PS aborts the command with a call to bti_at_abort_ind(). Usually the BT stack would confirm the unsolicited result first, but it may be, that this call has been started before the reception of the unsolicited result and is processeced with a delay.

(BPS 2)
GSM confirms the abortion of the command.

A 2.3 Interface to the DTI SAP

Every data transmission through the BTI is controlled by flow control, so that it can be ensured that both sides can receive the data and no buffer overflow occurs.

Before any data transfer function can be called, the corresponding side has to indicate that it is ready to receive data by sending a flow control signal. After sending a data packet, the corresponding side has to indicate again that it is ready to receive data before sending another data packet.

A 2.3.1 The GSM PS sends data to the Bluetooth PS

```
  GSM                           BPS
   |    bti_data_ready_req()     |
   |==========================> |
   |                         (BPS 1)
   |                             |
   |     bti_get_gsm_data()      |
   |<========================== |
(GSM 1)           :              |
   |              :              |
   |     bti_get_gsm_data()      |
   |<========================== |
   |                             |
   |                             |
   |    bti_data_ready_cnf()     |
   |<========================== |
(GSM 2)                          |
```

(BPS 1)
GSM calls bti_data_ready_req() as a signal that data is available.

(GSM 1)
GSM copies the data to its buffer.

(GSM 2)
GSM signals that the copy procedure has finished and new data can be received.

A 2.3.2 The Bluetooth PS sends data to the GSM PS

```
  GSM                           BPS
   |                             |
   |    bti_data_ready_ind()     |
   |<========================== |
(GSM 1)                          |
   |                             |
   |                             |
   |      bti_get_bt_data()      |
   |==========================> |
   |              :          (BPS 1)
   |              :              |
   |      bti_get_bt_data()      |
   |==========================> |
   |                             |
   |                             |
   |    bti_data_ready_res()     |
   |==========================> |
   |                         (BPS 2)
```

(GSM 1)
BT calls bti_data_ready_ind() as a signal that data is available.

(BPS 1)
GSM copies the data to its buffer.

(BPS 2)
GSM signals that the copy procedure has finished and new data can be received.