User Manual

BAT Lib User Manual

Department:	WTBU - Cellular Systems		
Creation Date:	2005-01-12		
Last Modified:	2006-01-11 by TI Employee		
ID:	20_04_02_01902	Version:	010
Status:	Draft	ECCN:	5D991

0 Document Control

Texas Instruments Incorporated and / or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and / or services. To minimize the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and / or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and / or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and / or software may be based on or implement industry recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and / or the licensing of software do not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

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 TI.

0.1 Export Control Statement

Recipient agrees that it will not knowingly export or re-export, directly or indirectly, any product or technical data (as defined by the U.S, EU and other Export Administration Regulations) including software, or any controlled product restricted by other applicable national regulations, received from Disclosing party under this Agreement, or any direct product of such technology, to any destination to which such export or re-export is restricted or prohibited by U.S or other applicable laws, without obtaining prior authorisation from U.S. Department of Commerce and other competent Government authorities to the extent required by those laws. This provision shall survive termination or expiration of this Agreement.

According to our best knowledge of the state and end-use of this product or technology, and in compliance with the export control regulations of dual-use goods in force in the origin and exporting countries, this

technology is classified as given on the front page.

This product or technology may require export or re-export license for shipping it in compliance with certain countries regulations.

0.2 Document History

Date	Version	Status	Author
2005-01-12	001	Draft	TI Employee
Initial version.			
2005-01-13	002	Draft	TI Employee
Outline defined			
2005-01-20	003	Draft	TI Employee
First draft			
2005-01-26	004	Draft	TI Employee
First draft submitted for review			
2005-02-01	005	Draft	TI Employee
Submitted version after review			
2005-02-23	006	Draft	TI Employee
• Naming convention is changed from T_bat_xxx to T_BAT_xxx for all structures; • Definition of T_BAT_event is changed, element <client> is deleted; • Constant BAT_ADAPTER_DEVICE_SIZE is renamed to BAT_INSTANCE_HEADER_SIZE; • Parameter <config> in bat_new() is changed from T_BAT_config to T_BAT_config *; • A new parameter <client> is added to bat_ctrl(); • BusyResource is renamed to BAT_BUSY_RESOURCE or BAT_APP_READY_RESOURCE accordingly; • The pseudo code demonstrating the use of bat_send() is updated according to the update of p_bat.val; • Added a section about the not supported AT commands in chapter 1; • Added a section about the handling of UCS2 strings in chapter 1; • Added a section about the indicating of the omitted optional parameters in chapter 1.			
2005-03-30	007	Draft	TI Employee
• A new section 2.2.1 T_BAT_return is added; • In section 2.2.5 more details about T_BAT_event is added; • In section 2.2.7 more details about T_BAT_signal is added; • Description about finial result code in section 2.2.9.2 is updated; • Description about max instance to open in section 2.3.1 is updated; • The section about T_BAT_buffer is deleted; • An argument in Response_cb() and unsolicited_result_cb() is changed from type T_BAT_buffer to type T_BAT_cmd_response; • The returned type of all BAT Lib interface functions is changed to T_BAT_return; • bat_new () function interface is changed, unsolicited_result_cb is removed from the arguments; • Description of bat_open() BUSY case is updated; • A new interface function bat_uns_open() is added in section 2.3.6 for opening an unsolicited client channel; • A parameter <client> is added to the arguments of unsolicited_result_cb (); • Figure 5, 6, 9 and 10 are updated; • Use case “initializing a connection” in section 2.6.1 is updated; • A new section added to describe the source files that will be delivered with this document.			

Date	Version	Status	Author
2005-04-08	• 008	• Draft	• TI Employee
1) Removed test command and its response for CGSMS, because it is static. +CGSMS: (0-3) Static test responses are not supported on BAT level. 2) The unsolicited result code for [+|%]CREG and [+|%]CGREG do not have the parameter <n>. The <n> parameter enables/disables unsolicited result codes. The same for `+CCWA.` On BAT level it is always enabled. BAT clients can deny unsolicited results by do not offering a callback function for unsolicited results.			
2005-05-18	009	Draft	TI Employee
1) T_BAT_signal is extended. 2) In section 2.3.9 description of bat_ctrl() is modified. 3) Section 2.6.6 is modified.			
2006-01-11	010	Draft	TI Employee
Updated section 2.6.6 pseudo code and MSC.			

0.3 References, Abbreviations, Terms

Ref 1	ITU-T Draft new Recommendation V.25ter: "Serial asynchronous automatic dialling and control".
Ref 2	ETSI TS 127 007: AT command set for 3G User Equipment (UE)
Ref 3	ETSI TS 127 005: (DTE-DCE) interface for Short Message Service (SMS) and Cell Broadcast Service (CBS)

AT	ATtention; this two-character abbreviation is always used to start a command line to be sent from TE to TA
ATI	AT command Interpreter or AT command Interface
BAT	Binary AT command
BAT Lib	Binary AT command Library
ETSI	European Telecommunication Standards Institute
ITUT	International Telecommunication Union, Telecommunication [sector]
MOC	Mobile Originated Call
MTC	Mobile Terminated Call
TA	Terminal Adaptor, e.g. a GSM data card (equal to DCE; Data Circuit terminating Equipment)
TE	Terminal Equipment, e.g. a computer (equal to DTE; Data Terminal Equipment)
TI	Texas Instruments Inc
U8	Unsigned character (used in the pseudo code)
BOOL	unsigned char

Table of Contents

Table of Figures

1 Introduction

1.1 Introduction to Manual

This manual is intended to give the reader an insight idea of the product and the usage of it. It is assumed that the reader of this manual has the basic knowledge of software development and is aware of fhe ETSI (European Telecommunication Standards Institute) and ITUT standards, which contain a series of attention (AT) commands recognized by a mobile phone. For information on these standards, see: Ref 1, Ref 2 and Ref 3.

1.2 Introduction to Product

The product being introduced here is a library used by the application (in connection with the TI protocol stack) to send out binary AT commands to the protocol stack and receive response from the protocol stack in binary format. The library is called BAT Lib or BAT library throughout this document. The following figure shows the BAT Lib in context with the application and the TI protocol stack in a simplified way. Depending on the communication method, the connection between the BAT Lib and the TI protocol stack can either be hardware connection (2 processor solution) or software connection (1 processor solution).

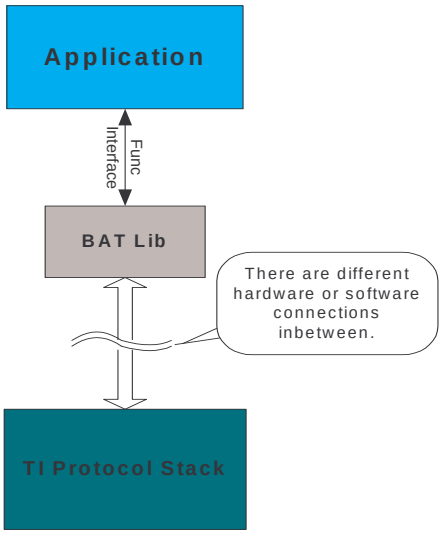

Figure 1: Binary Concept Overview

Another possible way of sending AT commands to the protocol stack is by sending ASCII strings such as “AT+CFUN=1”. But such an ASCII string based AT command communication module between the application domain and the modem is quite memory utilising because the application has to use an ASCII string based AT command generator and response parser.

With the introduction of binary AT command, the application side is able to save a considerable memory and time (no ASCII generator and parser is needed), especially on a single chip solution, where applications and modem reside on the same chip.

1.3 Not Supported AT Commands

BAT doesn’t support all the AT commands described in the specs. The not supported commands are basically ATI related commands. They are listed in appendix A. Please note that the fax commands are

not supported in this release but will be supported in the next release.

1.4 Handling of UCS2 Strings

For UCS2 related commands BAT provides an additional command structure with the same name as for the 7/8 Bit character set command structures, but with an added suffix _w, which stands for "wide character".

Example: the following structures define the response of the BAT command “AT+CPBR”, the first one is a 7/8-bit command structure and the second one is a UCS2 command structure. The only differences are the naming and the length of the text. All the binary AT command structures are defined in header file p_bat.h, which will be delivered together with this document.

```
typedef struct
{
  U8    index;
  U8    number[BAT_MAX_CC_ORIG_NUM_LEN];
  U8    type
  U8    text[BAT_MAX_ALPHA_LEN];
 } T_BAT_res_set_plus_cpbr;


typedef struct
{
  U8    index
  U8    number[BAT_MAX_CC_ORIG_NUM_LEN];
  U8    type;
  U16   text[BAT_MAX_ALPHA_LEN];
} T_BAT_res_set_plus_cpbr_w;
```

1.5 Indicating of Omitted Optimal Parameters

All the optional parameters are explicitly mentioned in the comment field of the elements in a structure definition in the sap document. The following sections describe how different types of optional parameters in the BAT command can be omitted.

1.5.1 Numerical Parameter

Numerical parameters can be omitted by assigning -1. For all numerical parameters with an enumeration definition the value –1 is defined in the enumeration by BAT_XXX_NOT_PRESENT.

If NOT_PRESENT is defined in an enumeration it does not necessary mean that all parameters with this type could be omitted. Please refer to the SAP document.

Example:

```
typedef enum
{
  BAT_CCUG_N_NOT_PRESENT = -0x1, /* This indicates that the value is omitted. */
  BAT_CCUG_N_DISABLETMP  = 0x0,  /* disable temporary mode         */
  BAT_CCUG_N_ENABLETMP   = 0x1   /* enable  temporary mode         */
} T_BAT_VAL_plus_ccug_n;
```

For all numerical parameters that are not enumerations it is obvious from the structure definition if the value could be omitted or not (Omissible parameters are signed and all others are unsigned). In this

case the global definition BAT_PARAMETER_NOT_PRESENT should be used to omit the parameter.

1.5.2 Arrays

For arrays, which consists of numerical values, the omission of such an array can be indicated by setting the counter c_xxx = 0, e.g. c_number=0 in the following example.

Example:

```
typedef struct
{
  T_BAT_plus_ccfc_reason reason;/*(enum=32bit)<->T_BAT_plus_ccfc_reason CCFC reason*/
  T_BAT_plus_ccfc_mode   mode;  /*(enum=32bit)<->T_BAT_plus_ccfc_mode CCFC mode */
  U8                     v_number; /*valid-flag*/
  U8                     c_number; /*counter*/
  U8                     number[BAT_MAX_CCFC_NUMBER_LEN]; /*number*/
    ...
} T_BAT_cmd_set_plus_ccfc;
```

1.5.3 Strings

For strings there is a special valid flag v_xxx, which is set to 0 in case of omission. This valid flag allows to distinguish between empty string (string based: “”) and parameter omitted (string based: ,,).

Example: The apn can be set to "", but has not been omitted. In such a case the v_apn=1 and c_apn=0.

```
typedef struct
{
  T_BAT_pdp_cid   pdp_cid;/*(enum=32bit)<->T_BAT_pdp_cid specifies a particular PDP context*/
  T_BAT_pdp_type  pdp_type;/*(enum=32bit)<->T_BAT_pdp_type the type of packet data protocol*/
  U8              v_apn;   /*<  8:  1> valid-flag*/
  U8              c_apn;   /*<  9:  1> counter*/
  U8              apn[BAT_MAX_APN_LEN]; /*< 10:102> Access Point Name*/
    ...
} T_BAT_cmd_set_plus_cgdcont;
```

2 BAT Lib

2.1 Introduction to BAT Lib

This chapter describes the use of binary AT commands, namely the use of the BAT Lib, which is linked to the application domain. The API of the BAT Lib provides a small set of low-level API functions (see section 2.3) and for each supported BAT command a C structure (defined in header file p_bat.h). Each C structure represents the BAT command on binary level. The following figure illustrates how the BAT library is linked to the application domain.

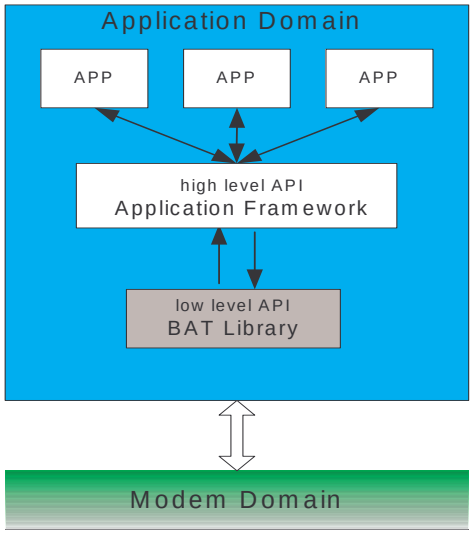

Figure 2: General Overview

The greyish box in the figure represents the BAT library. It has been assumed that there is a kind of middleware (application frame work), which hides the applications from knowing low-level details. Figure 3 shows the API of the BAT Lib to the application framework.

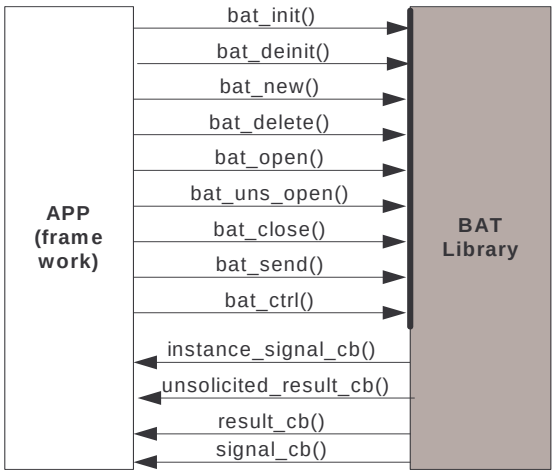

Figure 3: BAT Lib API

2.2 Data Structures

This section describes all the data structures used by the BAT Lib API. Since all the data structures are defined in p_bat.h, which is delivered as source code, detailed descriptions are only provided to some of them in this document.

2.2.1 T_BAT_return

Definition:

```
typedef enum
{
  BAT_OK = 0,
  BAT_BUSY_RESOURCE,
  BAT_ERROR
}T_BAT_return;
```

Description:

This enum defines the return value from the BAT library or from the application.

2.2.2 T_BAT_instance

Description:

This defines a handle for a BAT Lib instance. This instance handle is unique for each instance.

2.2.3 T_BAT_client

Description:

This defines a client handle in a BAT Lib instance. The client handle is unique for each client.

2.2.4 T_BAT_config

Definition:

```
typedef struct
{
   T_BAT_device_type    type;
   T_BAT_adapter        adapter;
   T_BAT_L2P            l2p;
} T_BAT_config;
```

Description:

This structure defines the configuration of the BAT library. A const structure instance will be provided to the application to be used like an enumeration tag, e.g. BAT_APP_CONFIG when calling the bat_new(). Each tag is corresponding to a fixed combination of device type, BAT adapter and L2P protocol. Please note that the application has to call the function in the bat_cfg.c (delivered as source code) to configure the correct settings.

2.2.4.1 T_BAT_device_type

Description:

This is an enumeration where currently only one value is supported: BAT_PACKET_DEVICE.

2.2.4.2 T_BAT_adapter

Description:

This structure will be used by the application only when the customer implements his own BAT adapter.

2.2.4.3 T_BAT_L2P

Description:

This structure will be used by the application only when the customer implements his own L2P protocol.

2.2.5 T_BAT_event

Definition:

```
typedef enum
{
  BAT_ABORT = 0,
  BAT_APP_READY_RESOURCE
}T_BAT_event;
```

Description:

It is enumeration type, which defines currently BAT_ABORT and BAT_APP_READY_RESOURCE.

2.2.6 T_BAT_ctrl

Definition:

```
typedef struct
{
   T_BAT_event event;
} T_BAT_ctrl;
```

Description:

This is a structure with the client handle and an enumeration type defining all the currently possible options for controlling.

2.2.7 T_BAT_signal

Definition:

```
typedef enum
{
  BAT_NEW_INSTANCE_SUCCEED = 0,
  BAT_NEW_INSTANCE_FAIL,
  BAT_OPEN_CLIENT_SUCCEED,
  BAT_OPEN_CLIENT_FAIL,
  BAT_ABORT_COMMAND_SUCCEED,
  BAT_ABORT_COMMAND_FAIL,
  BAT_READY_RESOURCE
}T_BAT_signal;
```

Description:

This is an enum type defining all the possible signals from BAT Lib to the application.

2.2.8 T_BAT_cmd_send

Definition:

```
typedef struct
{
   T_BAT_ctrl_params    ctrl_params;
   T_BAT_params         params;
} T_BAT_cmd_send;
```

Members:

ctrl_params	controller for union in enum type
params	union of pointers to all the supported BAT commands

Description:

This structure defines all the BAT commands.

2.2.8.1 T_BAT_ctrl_params

Definition:

```
typedef enum
{
   …
   BAT_CMD_SET_PLUS_CMGS          = 0x25,
   … /* and a lot more */
}T_BAT_ctrl_params;
```

Description:

This enum serves as binary AT command identifier for the supported BAT commands including set, test and query. E.g. `BAT_CMD_SET_PLUS_CMGS` is the identifier for +CMGS set command.

2.2.8.2 T_BAT_params

Definition:

```
typedef union
{
   …
   T_BAT_cmd_set_plus_cmgs   *ptr_set_plus_cmgs;
   … /* and a lot more */
}T_BAT_params;
```

Description:

It is a union of pointers to structures, which are defined according to the definitions in standard 127 007 (Ref 2), 127 005 (Ref 3) and V.25ter (Ref 1).

E.g. structure `T_BAT_cmd_set_plus_cmgs` is defined as:

```
typedef struct
{
  U8 length; /*number of octets without SMSC address octets */
  U8 c_pdu;  /*counter*/
  U8 pdu[BAT_MAX_SM_LEN]; /*Mandatory: PDU data */
} T_BAT_cmd_set_plus_cmgs;
```

If the BAT command doesn’t have parameters (e.g. for query and test commands), there will be no

T_BAT_params structure corresponding to the tag in T_BAT_ctrl_params. The user can always refer to the header file p_bat.h for details.

2.2.9 T_BAT_cmd_response

Definition:

```
typedef struct
{
   T_BAT_ctrl_response  ctrl_response;
   T_BAT_response       response;
} T_BAT_cmd_response;
```

Members:

ctrl_response	controller for union in enum type
response	union of pointers to all the supported BAT command responses

Description:

This structure defines all the possible responses to all BAT commands and the unsolicited result code from the protocol stack.

2.2.9.1 T_BAT_ctrl_response

Definition:

```
typedef enum
{
   BAT_RES_AT_OK        = 0x0,
   …
   BAT_RES_SET_PLUS_CMGS= 0x32,
   …/* and a lot more */
}T_BAT_ctrl_response;
```

Description:

The results from 0 … 9 are used for the basic V25.ter **(**Ref 1) result codes. All other enumerators are used to identify the other appropriate response.

2.2.9.2 T_BAT_response

Definition:

```
typedef union
{
   T_BAT_no_parameter        *ptr_at_ok;
   …
   T_BAT_res_set_plus_cmgs   *ptr_set_plus_cmgs;
   …/* and a lot more */
} T_BAT_response;
```

Description:

Each struct in the union defines the corresponding response. The response includes intermediate, finial and unsolicited result code. Finial result code includes OK, +CME, +CMS, CONNECT, NO CARRIER, BUSY, NO ANSWER in the TI protocol stack implementation. The application has to receive the finial result code before sending out another BAT command from the same client. Note that CONNECT is finial result code in the TI implementation and NO CARRIER can be either unsolicited or finial result code, in this case BAT Lib will distinguish them by defining different

control values such as BAT_RES_AT_NO_CARRIER_FINAL and BAT_RES_AT_NO_CARRIER_INTERMEDIATE.

2.3 BAT Lib API

This section describes the BAT Lib API to the application.

2.3.1 bat_init

Prototype:

```
T_BAT_return     bat_init  (void* mem,
                            U8 num);
```

Parameters:

mem	input parameter, the memory allocated by the application frame work for maintenance of the BAT instances
num	input parameter, the number of instances the application is planning to create

Return:

BAT_OK	OK
other values	Failure

Description:

This function is used to initialize the BAT Lib to handle as many BAT library instances as the number provided by the parameter num. Please note that according to the current implementation of the TI protocol stack, up to 4 instances can be opened. This function provides a pointer to the memory, which is allocated by the application framework, for BAT Lib internal maintenance.

If the application wants to open N instances of the BAT library, the application framework has to allocate (BAT_INSTANCE_HEADER_SIZE * N), where N is the number of instances the application is to open and BAT_INSTANCE_HEADER_SIZE will be defined in the header file bat.h.

The application framework is responsible for deciding how many instances it wants to open and it has to call bat_new() each time to create a new BAT library instance.

2.3.2 bat_deinit

Prototype:

```
T_BAT_return     bat_deinit();
```

Parameters:

No parameters.

Return:

BAT_OK	OK
other values	Failure

Description:

This function is used to clear the data for maintaining the BAT Lib instances. All BAT instances need to be deleted before. The memory provided in bat_init() could be freed by the application afterwards.

2.3.3 bat_new

Prototype:

```
T_BAT_return      bat_new    (T_BAT_instance* instance,
                              void* mem,
                              U8 num,
                              T_BAT_config* config,
                              void (*instance_signal_cb)( T_BAT_signal signal));
```

Parameters:

instance	output parameter, used to pass a handle of the instance to the application
mem	input parameter, the memory allocated by application frame work for maintenance of the instance and clients
num	input parameter, the number of clients the application is planning to open
config	input parameter, describes the chosen configuration for this instance. The configuration will be used in a similar way as using an enumeration tag
instance_signal_cb	call back function pointer, used to inform the application about successful or unsuccessful creation of a BAT instance

Return:

BAT_OK	OK
other values	failure

Description:

This function is used to create a new instance of BAT Lib. It provides BAT Lib the following information: an output parameter with which BAT Lib is able to pass the handle to the application; a pointer to the memory, which is allocated by the application framework, for internal maintenance; the number of clients to maintain; the appropriate configuration, a signal call back function pointer.

If the application wants to open n clients, the application framework has to allocate (BAT_INSTANCE_SIZE + (n * BAT_CLIENT_SIZE)) bytes of memory, where BAT_INSTANCE_SIZE and BAT_CLIENT_SIZE will be defined in the header file bat.h.

The application framework is responsible for deciding how many clients it wants to use and it has to call bat_open() each time to open a new client path. Please note that if more than one client is opened within one instance they can not call bat_send () concurrently but one after the other.

The signal callback function is used to inform the application about a successful established link between the application and modem domain. This happens asynchronously.

2.3.4 bat_delete

Prototype:

```
T_BAT_return bat_delete      (T_BAT_instance instance);
```

Parameters:

instance	input parameter, the handle of the instance obtained in bat_new()

Return:

BAT_OK	OK
other values	failure

Description:

This function is used to delete the BAT Lib instance created by function bat_new(). If this function returns BAT_OK, the memory allocated for this instance can be freed by the application. Please note that the application should first call bat_close() to close all the clients including the unsolicited result code client before deleting the BAT Lib instance.

2.3.5 bat_open

Prototype:

```
T_BAT_return bat_open       (T_BAT_instance instance,
                            T_BAT_client *client,
                            int  (*response_cb)( T_BAT_client client,
                                                 T_BAT_cmd_response *response),
                            void  (*signal_cb)(  T_BAT_client client,
                                                 T_BAT_signal signal));
```

Parameters:

instance	input parameter, the instance received with bat_new()
client	output parameter, used to pass the client handle to the application
response_cb	response call back function pointer see 2.4.3
signal_cb	signal call back function pointer see 2.4.4

Return:

BAT_OK	OK
BAT_BUSY_RESOURCE	BAT Lib is busy application should try later
other values	failure

Description:

This function is used to open a new binary AT command client path. This function passes a unique client handle to the application by the output parameter client. This client handle is used by bat_send(), bat_close() and bat_cntrl().The bat_open() function is not re-entrant and should be called in serial and not in concurrence of two tasks within one instance. If the returned value is BAT_BUSY_RESOURCE, the instance will be informed via instance_signal_cb() when the BAT Lib is ready again. The application then has to call bat_open() again with all the necessary information.

2.3.6 bat_uns_open

Prototype:

```
T_BAT_return bat_uns_open    (T_BAT_instance instance,
                             T_BAT_client   *client,
                             int            (*unsolicited_result_cb)
                                            ( T_BAT_client client,
                                              T_BAT_cmd_response *response));
```

Parameters:

instance	input parameter, the instance received with bat_new()
client	output parameter, used to pass the client handle to the application
unsolicited_result_cb	signal call back function pointer see 2.4.2

Return:

BAT_OK	OK
other values	failure

Description:

This function is used to open a path for receiving the unsolicited result code. This function passes a unique client handle to the application by the output parameter client. This client handle is used when application calls bat_close() to close the unsolicited code channel or calls bat_cntrl() to signal a ready resource. Please note that this is a special client and can only be used to receive unsolicited result code, not to send any BAT commands.

2.3.7 bat_close

Prototype:

```
T_BAT_return      bat_close( T_BAT_client client);
```

Parameters:

client	input parameter, the client handle obtained by calling bat_open()

Return:

BAT_OK	closing of BAT command channel is complete
BAT_BUSY_RESOURCE	BAT Lib is busy application should try later
other values	failure

Description:

This function is used to close a binary AT command channel. The close process is complete when function returns BAT_OK. The application can delete the BAT Lib instance only after successfully deleting all the clients by calling bat_close(). Please refer to 2.5 if the returned value is BAT_BUSY_RESOURCE.

2.3.8 bat_send

Prototype:

```
T_BAT_return bat_send       (T_BAT_client client,
                             T_BAT_cmd_send *cmd);
```

Parameters:

client	input parameter, client handle, which is obtained by calling bat_open()
cmd	input parameter, a pointer to a structure, which defines a binary AT command and its parameters

Return:

BAT_OK	OK; BAT command has been sent, but processing of it is ongoing at modem domain
BAT_BUSY_RESOURCE	BAT Lib is busy application should try later
other valus	failure

Description:

This function is used to send a BAT command. The final result is received asynchronously by response_cb(). That means the application is only blocked as long as bat_send() needs to pass all octets. Please refer to 2.5 if the returned value is BAT_BUSY_RESOURCE. Please note that the BAT Lib can only handle a serial call to the bat_send() function within one instance. If a client calls bat_send() while another client is running a BAT command, an error will be returned.

2.3.9 bat_ctrl

Prototype:

```
T_BAT_return      bat_ctrl(  T_BAT_client client,
                             T_BAT_ctrl* ctrl);
```

Parameters:

client	input parameter, client handle, which is obtained by calling bat_open()
ctrl	input parameter, defines the control parameters e.g. BAT_ABORT or BAT_APP_READY_RESOURCE

Return:

BAT_OK	OK, current control command is ongoing
BAT_BUSY_RESOURCE	BAT Lib is busy application should try later
other values	failure

Description:

This function is used to send control information, which is not directly related to a binary command request or response, to the BAT library. E.g. telling the BAT Lib to stop the currently running BAT command or to inform the BAT Lib that the application is no longer at BAT_BUSY_RESOURCE state, but is able to receive more data. Further reasons are to be defined. Please refer to 2.5 if the returned value is BAT_BUSY_RESOURCE. Please note that after BAT_APP_READY_RESOURCE is indicated to the BAT Lib with bat_ctrl(), the call back function (response_cb() or unsolicited_result_cb()) will be running in the same context of the bat_ctrl() function. If it is to abort a running command, the finial result (succeed/fail) of aborting the command will be sent to the application via signal_cb().

2.4 Call Back Functions

2.4.1 instance_signal_cb

Prototype:

```
void instance_signal_cb(     T_BAT_signal       signal);
```

Parameters:

signal	input parameter, the result of creating/deleting an instance of BAT Lib

Return:

void

Description:

This is the callback function used to indicate the finial result of creating a new instance by calling bat_new().

2.4.2 unsolicited_result_cb

Prototype:

```
int unsolicited_result_cb (  T_BAT_client       client
                             T_BAT_cmd_response *response);
```

Parameters:

client	input parameter, client handle, which is passed in with bat_uns_open()
response	input parameter, describes a response message

Return:

BAT_OK	OK, data has been processed by the application
BAT_BUSY_RESOURCE	BAT_BUSY_RESOURCE, the application could not handle the data currently
other values	failure

Description:

The unsolicited result callback is used to inform the application whenever unsolicited code is received. The unsolicited code will only be sent to this instance by calling this call back. If BAT_BUSY_RESOURCE is returned from the application, the Application needs to call bat_ctrl with this specific client handle obtained from bat_uns_open(). If the unsolicited result callback is not provided then there will be no unsolicited results sent to the instance.

2.4.3 response_cb

Prototype:

```
int  response_cb( T_BAT_client client,
                  T_BAT_cmd_response *response);
```

Parameters:

client	input parameter, client handle, which is obtained by calling bat_open()
response	input parameter, describes a response message

Return:

BAT_OK	OK, response has been processed by the application
BAT_BUSY_RESOURCE	the application could not handle the response currently
other values	failure

Description:

This is the callback function provided by the application. This function is used to pass the response from the protocol stack to the application. Please refer to 2.5 if the returned value is BAT_BUSY_RESOURCE.

2.4.4 signal_cb

Prototype:

```
Void signal_cb(    T_BAT_client client,
                   T_BAT_signal signal);
```

Parameters:

client	input parameter, client handle, which is obtained by calling bat_open() or bat_uns_open()
signal	input parameter, the signal indicated by BAT Lib

Return:

void

Description:

The signal callback function is used to inform the application about any events (signals), which are not responses, notifications or indications related to BAT commands or protocol stack activities (network activities). The signal number indicates the event, e.g. BAT_READY_RESOURCE.

2.5 Flow Control

There is a simple flow control mechanism implemented. The followingBAT Lib functions as well as the response callback implemented by the application could respond with BAT_BUSY_RESOURCE: bat_send(), bat_close(), bat_ctrl(), unsolicited_result_cb() and response_cb().

The counterpart that receives this BAT_BUSY_RESOURCE signal needs to store the last provided information – on the Application side the last command or the last request and on the BAT Lib side the last response or unsolicited code. The stored information should be resent when a ready signal is received. The application indicates the ready signal by calling bat_ctrl() and the BAT library by calling the signal callback function signal_cb() or instance_signal_cb().

This flow control works on the level of clients, so BAT_BUSY_RESOURCE indicates only the Busy status for one client. But the application has to make sure that a ready signal is sent out to the BAT Lib when the client is ready, otherwise the other clients are blocked since all clients share one channel in the communication between BAT Lib and the lower layers.

The following figure shows the flow control between the application and BAT Lib.

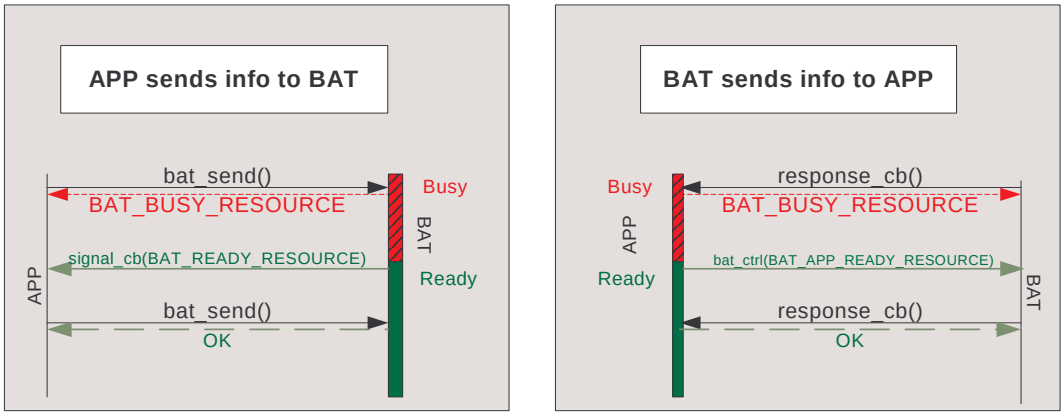

Figure 4: Flow Control

2.6 Use Cases

This section explains from the application's viewpoint how the BAT Lib can be used.

2.6.1 Initializing A Connection

Functions Involved:

- bat_init()
- bat_new()
- instance_signal_cb()

Pseudo code:

```
T_BAT_instance app_handle;//instance handle, defined globally
void instance_signal_cb (T_BAT_signal signal) {…}
void app_init ()//init function on the app side
{
   U8 num1 = 2;//number of instances as an example
   U8 num2 = 2;//number of clients as an example

   //configure the BAT Lib, this func is defined in bat_cfg.c
   app_set_config ();

   //allocate memory for maintaining the instances
   U8 *mem1 = (U8 *)malloc (num1 * BAT_INSTANCE_HEADER_SIZE);

   //call the init func in BAT Lib
   if (0 != bat_init (mem1, num1)) return;

   //allocate memory for maintaining the clients in the instance
   U8 *mem2 = (U8 *)malloc (BAT_INSTANCE_SIZE + (num2 * BAT_CLIENT_SIZE));

   //call the func in BAT Lib to create a new instance
   bat_new(&app_handle,mem2,num2, BAT_APP_CONFIG, instance_signal_cb);
}
```

MSC:

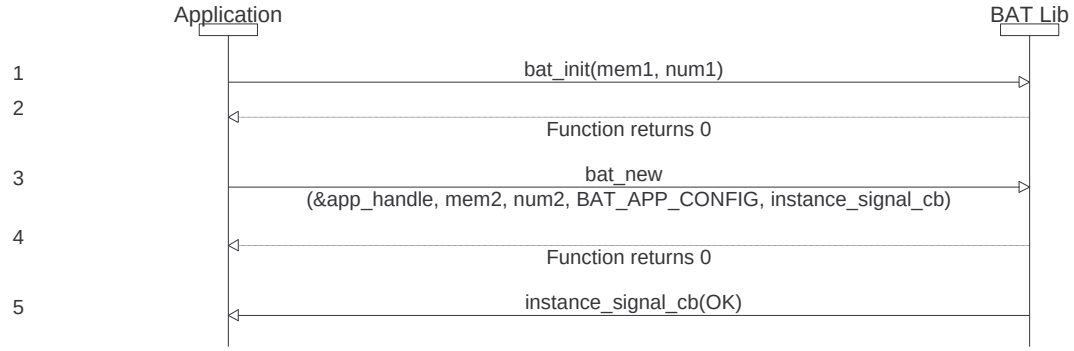

Figure 5: Initializing

1. Application calls the init function to pass in some initial information: the number of instances it wants to open and the allocated memory to maintain the instances
2. Function returns: 0 for ok, meaning that the process is ongoing, and other values for failures
3. Application calls function bat_new() in the BAT Lib to create a BAT Lib instance

4. Function returns: 0 for ok, meaning that the process is ongoing and other values for failures
5. BAT Lib calls instance_signal_cb() to indicate the final result of the creation of the BAT Lib instance, OK means creation of instance has succeeded

2.6.2 Establishing A Connection

Functions Involved:

- bat_open()
- signal_cb()
- response_cb()

Pseudo code:

```
T_BAT_client client;//client handle, defined globally
void  signal_cb(   T_BAT_client client, T_BAT_signal signal)
{
   …
}
int  response_cb( T_BAT_client client, T_BAT_cmd_response *response)
{
   …
}
void app_open ()
{
   bat_open (app_handle, &client, response_cb, signal_cb);
}
```

MSC:

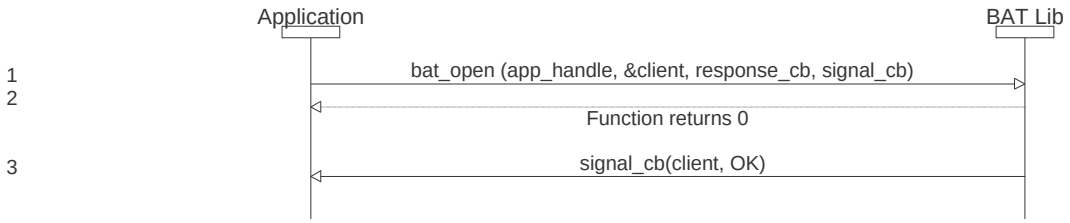

Figure 6: Connection Establishing

1. Application calls bat_open() to open 1 client path and gets back a unique client handle
2. bat_open() returns: 0 for ok, meaning that process is ongoing 1 for busy, meaning that the instance will be informed via instance_signal_cb() when the BAT Lib is ready and other values for driver errors
3. BAT Lib calls signal_cb() to indicate the final result for the client to open the path. OK means the opening of the client path has succeeded

2.6.3 Sending A BAT Command

Functions Involved:

- bat_send()

Pseudo Code:

```
void app_send ()
{
   T_BAT_cmd_set_plus_cfun my_set_plus_cfun;
   T_BAT_cmd_send my_bat_cmd_send;
   memset(&my_set_plus_cfun, sizeof(my_set_plus_cfun), FALSE);
   my_bat_cmd_send.ctrl_params = BAT_CMD_SET_PLUS_CMGS;
   my_bat_cmd_send.params.ptr_set_plus_cfun = &my_set_plus_cfun;
   /*full functionality mode*/
   my_set_plus_cfun.fun = BAT_CFUN_FUN_FULL;
   /* Reset the MT before changing functionality level */
   my_set_plus_cfun.rst = BAT_CFUN_RST_RESET;
   bat_send (client, &my_bat_cmd_send);
}
```

MSC: see Figure 7**.**

2.6.4 Receiving A Response

MSC:

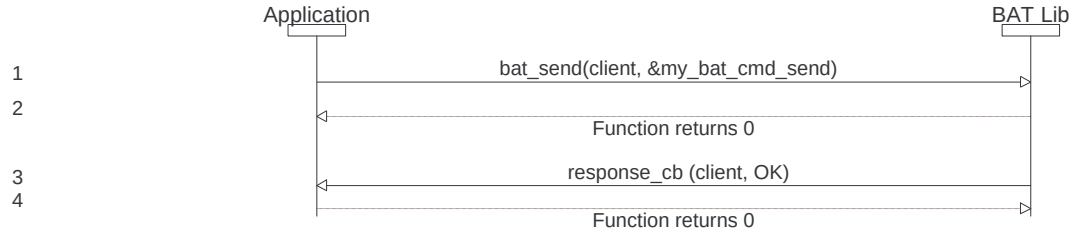

Figure 7: Data Sending and Receiving

1. Application calls the send function in BAT Lib to send a BAT command
2. Function returns an integer value: 0 for ok, 1 for BAT_BUSY_RESOURCE and other values for errors. Please refer to 2.5 if the returned value is BAT_BUSY_RESOURCE.
3. BAT Lib calls response_cb() to pass the final result of the execution of the BAT command. The “OK” here is the finial result code, which means that the BAT command has been successfully executed.
4. If the application is able to handle the response, the function should return a “0”.

2.6.5 Receiving Unsolicited Result Code

The BAT Lib sends the unsolicited result to the application only when the application provides the unsolicited_result_cb() function via the bat_uns_open.

MSC:

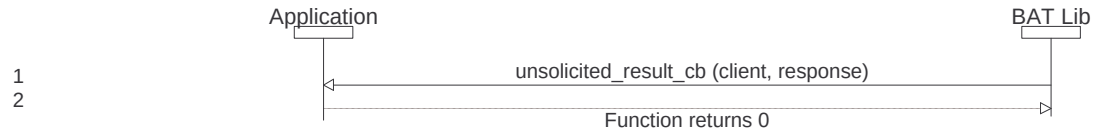

Figure 8: Unsolicited Result Code Receiving

1. BAT Lib sends out the the unsolicited code to the application;
2. If the application is able to handle the data, the function should return a “0”.

2.6.6 Aborting An Ongoing BAT Command

Functions Involved:

- bat_ctrl()
- signal_cb()

Pseudo Code:

```
void app_abort ()
{
   T_BAT_ctrl app_bat_ctrl;
   app_bat_ctrl.event = BAT_ABORT;
   bat_ctrl (client, &app_bat_ctrl);
}
```

MSC:

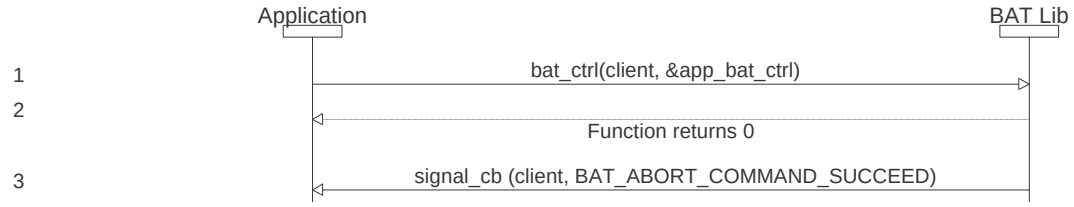

Figure 9: Ongoing BAT Command Aborting

1. Application calls bat_ctrl() to stop the running BAT command for this client;
2. Function returns an integer value: 0 for ok, 1 for BAT_BUSY_RESOURCE and other values for driver errors. Please refer to 2.5 if the returned value is BAT_BUSY_RESOURCE.
3. BAT calls signal_cb() to pass the final result.

2.6.7 Deleting A BAT Lib Instance

Functions Involved:

- bat_close()
- bat_delete()

Pseudo Code:

```
void app_delete ()
{
   bat_close(client);//perform this for all open clients
   bat_delete (instance);
}
```

MSC:

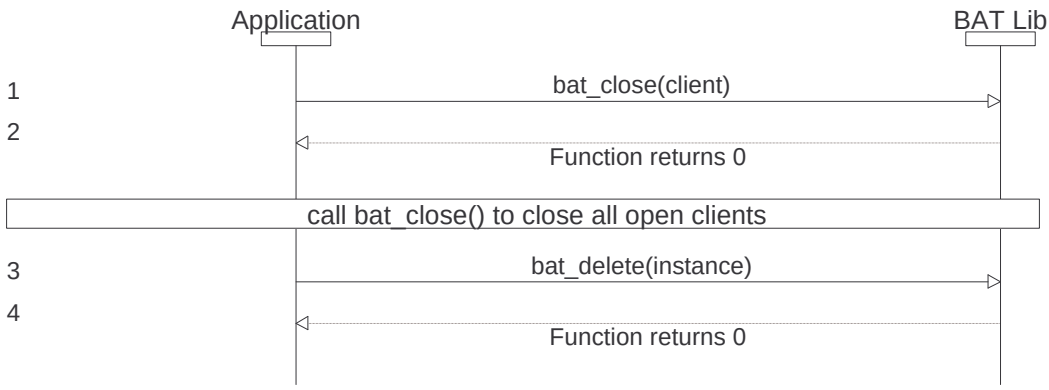

Figure 10: Instance Deleting

1. Application calls bat_close() to close the client path;
2. Function returns an integer value: 0 for ok and other values for error;
3. Application calls bat_delete() to delete a BAT Lib instance;
4. Function returns an integer value: 0 for ok, and other values for driver errors.

A List of Not Supported AT Commands

The not supported AT commands by BAT are:

```
+CLIP
+COLP
+COPN
+CMGF
+CSMP
+CSDH
A/
I
+GMI
+GMM
+GMR
+GSN
S0
S1
S3
S4
S5
S6
S7
S8
S9
S10
S99
E
Q
V
X
%CUNS
Zn
&W
+CGAUTO
```

B AT commands, which are not compliant to the specification.

The responses of +CREG, %CREG, +CGREG and %CGREG do not incorporate the parameter <n>.

```
+CREG: <n>,<stat>[,<lac>,<ci>]
%CREG: <n>,<stat>,[<lac>],[<ci>],<gprs_ind>
+CGREG: <n>,<stat>[,<lac>,<ci>]
%CGREG: <n>,<stat>,[<lac>],[<ci>],[<state>]

<n>: Registration mode:
   0  disable network registration unsolicited result code.
   1  enable network registration unsolicited result code
      +CREG: <stat>,,,<gprs_ind>
   2  enable network registration and location information
      unsolicited result code
      +CREG: <stat>,[<lac>],[<ci>],<gprs_ind>
```

The set command of CCWA does not incorporate the parameter <n>.

+CCWA=[<n>[,<mode>[,<class>]]]

<n> (sets/shows the result code presentation status in the TA):

0 disable

1 enable

With BAT the default mode is enable unsolicited result code. There is no possibility to disable it by a set command. A client of the BAT library can disable unsolicited result code in general by do not providing a callback function unsolicited_result_cb() via the bat_uns_open.

C List of Delivered Source Code

```
p_bat.h       : BAT command structures
p_bat.val     : enum definitions of the values used in p_bat.h
bat.h         : BAT Lib interface
bat_types.h   : BAT Lib type definition
bat_cfg.h     : BAT Lib configuration information
bat_cfg.c     : BAT Lib configuration information
CDG_ENTER.h   : Some TI specified file used only by p_bat.h
CDG_LEAVE.h   : Some TI specified files used only by p_bat.h
Gdd.h         : T_BAT_config needs this header file
L2p_types.h   : BAT_INSTANCE_SIZE needs some defines in this file
Typedefs.h    : TI definitions
```