GE863-PRO^3 Linux GSM Library User Guide

1vv0300782 Rev. 0 - 30/07/08

Making machines talk.

Disclaimer

The information contained in this document is the proprietary information of Telit Communications S.p.A. and its affiliates (“TELIT”).

The contents are confidential and any disclosure to persons other than the officers, employees, agents or subcontractors of the owner or licensee of this document, without the prior written consent of Telit, is strictly prohibited.

Telit makes every effort to ensure the quality of the information it makes available. Notwithstanding the foregoing, Telit does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information.

Telit disclaims any and all responsibility for the application of the devices characterized in this document, and notes that the application of the device must comply with the safety standards of the applicable country, and where applicable, with the relevant wiring rules.

Telit reserves the right to make modifications, additions and deletions to this document due to typographical errors, inaccurate information, or improvements to programs and/or equipment at any time and without notice.

Such changes will, nevertheless be incorporated into new editions of this document.

Applicable Products

Product	Part Number
GE863-PRO^3 with Linux OS	3990250698

GSM Library Version

C0.00.07

Contents

1 Introduction

1.1 Scope

The aim of this document is to illustrate a GSM library that exports on Linux operative system the main GSM features (such as GSM calls and GPRS or PPP connections) and to provide detailed examples in order to show the correct use of the APIs.

1.2 Audience

This User Guide is intended for software developers who develop applications on the GE863-PRO^3 module and need to use the GSM functionalities. With this library Telit offers a user friendly integration for the developers.

1.3 Contact Information, Support

Our aim is to make this guide as helpful as possible. Keep us informed of your comments and suggestions for improvements.

For general contact, technical support, report documentation errors and to order manuals, contact Telit's Technical Support Center at:

TS-EMEA@telit.com or http://www.telit.com/en/products/technical-support-center/contact.php

Telit appreciates feedback from the users of our information.

1.4 Product Overview

The GE863-PRO^3 is an innovation to the quad-band, RoHS compliant GE863 product family which includes a powerful ARM9^{TM} processor core exclusively dedicated to customer applications. The concept of collocating a powerful processor core with the GSM/GPRS engine allows developers to host their application directly. The PRO^3 incorporates much of the necessary hardware for communicating microcontroller solutions, including the critical element of memory, significant simplification of the bill of material, vendor management, and logistics effort are achieved.

1.5 Document Organization

This manual contains the following chapters:

- “Chapter 1, Introduction” provides a scope for this manual, target audience, technical contact information, and text conventions.
- “Chapter 2, Library setup” describes briefly how to import the GSM APIs within a project.
- “Chapter 3, APIs summary” contains a list of the APIs provided by the GSM library.
- “Chapter 4, Data types” provides a description of the types used within the library.
- “Chapter 5, APIs” provides a detailed description of the methods of the GSM library.
- “Chapter 6, List of acronyms” provides definition for all the acronyms and abbreviations used in this guide.

How to Use

If you are new to this product, it is highly recommended to start by reading through TelitGE863PRO3Linux_SW_UserGuide 1vv0300781 [4] and TelitGE863PRO3 Linux Development Environment User Guide 1VV0300780 [5] manuals and this document in their entirety in order to understand the concepts and specific features provided by the built in software of the GE863-PRO^3.

1.6 Text Conventions

This section lists the paragraph and font styles used for the various types of information presented in this user guide.

Format	Content
Italic Arial	C code examples and types definition.

1.7 Related Documents

The following documents are related to this user guide:

[1] Telit AT Commands Reference Guide 80000ST10025
[2] TelitGE863-PRO³ Hardware User Guide 1vv0300773a
[3] TelitGE863PRO^3 EVK User Guide 1VV0300776
[4] TelitGE863PRO^3 Linux SW User Guide 1vv0300781
[5] TelitGE863PRO^3 Linux Development Environment User Guide1VV0300780
[6] TelitGE863PRO^3 Product Description 80285ST10036a
[7] TelitGE863PRO^3 U-BOOT Software User Guide 1vv0300777

All documentation can be downloaded from Telit's official web site www.telit.com if not otherwise indicated.

1.8 Document History

Revision	Date	Changes
ISSUE#0	30/07/08	First release

2 Library setup

In order to check if the version of GSM library in the *Telit Development Environment* [5] is the last one please check on http://www.telit.com in download zone or contact Telit's Technical Support Center at: TS-EMEA@telit.com .

It is possible to update the library version on your development environment simply replacing the header file and the library, respectively located within the */opt/crosstools/telit/include/* and */opt/crosstools/telit/lib/* directories:

- Start the Linux console (Windows **Start Menu → All Programs → Telit Development Platform → Console**).
- Copy the library typing
  cp /mnt/windows/<PATH>/libGSM.a /opt/crosstools/telit/lib
- Copy the header file typing
  cp /mnt/windows/<PATH>/GSM_lib.h /opt/crosstools/telit/include

where **<PATH>** is the windows folder where you have stored the new version of the library files. For example, if you store them within C:\Temp you have to digit

cp /mnt/windows/Temp/libGSM.a /opt/crosstools/telit/lib

and

cp /mnt/windows/Temp/GSM_lib.h /opt/crosstools/telit/include

3 APIs summary

In the following table a summary of the functionalities/APIs is shown.

Functionality Group	API	Notes
Serial Modem management	GSM_InitSerialModem()	Initializes a serial modem device.
	GSM_TermSerialModem()	Terminates current serial modem link.
	GSM_SerialSignals()	Allows performing basic actions on the physical serial device signals.
AT commands & data exchange	GSM_SendATcommand()	Sends a single AT command.
	GSM_SendData()	Sends binary data to the modem.
	GSM_sendEscape()	Sends the escape sequence (+++).
	GSM_ReadResponse()	Reads modem's responses (to AT commands Reference Guide).
	GSM_ReceiveData()	Receives binary data from the modem.
SIM related actions	GSM_InsertPIN()	Inserts the PIN code.
GSM calls	GSM_StartVoiceCall()	Starts a voice call.
	GSM_StopVoiceCall()	Ends the current voice call.
	GSM_StartDataCall()	Starts a data (data/fax) call.
	GSM_PauseDataCall()	Suspends the current data call.
	GSM_StopDataCall()	Ends the current data call.
GPRS connections	GSM_InitGPRS()	Initializes a GPRS connection using the internal PPP stack of the GE863 module.
	GSM_TermGPRS()	Terminates the current GPRS connection.
	GSM_InitPPPoverGPRS()	Initializes a PPP connection using the Linux PPP daemon.
	GSM_TermPPPoverGPRS()	Terminates current PPP connection.
	GSM_CheckPPPoverGPRS()	Checks if the "ppp0" interface is up.
SMS related actions	GSM_ConfigSMS()	Configures the modem to send Short Messages.
	GSM_SendSMS()	Send a Short Message.
Version	GSM_PrintLibVersion()	Prints the current version of the library.

4 Data types

4.1 GSM_Boolean_t

This type is an enum containing GSM_True and GSM_False values.

typedef enum {GSM_False, GSM_True} GSM_Boolean_t;

4.2 GSM_Baudrate_t

This type is an enum containing all allowed baudrates. Each symbol is coupled with the corresponding Linux standard baudrate constant.

```
typedef enum {
        GSM_300 = B300,
        GSM_600 = B600,
        GSM_1200 = B1200,
        GSM_2400 = B2400,
        GSM_4800 = B4800,
        GSM_9600 = B9600,
        GSM_19200 = B19200,
        GSM_38400 = B38400,
        GSM_57600 = B57600,
        GSM_115200 = B115200,
        GSM_INVALID_BAUDRATE
} GSM_Baudrate_t;
```

4.3 GSM_TimeoutMode_t

This type is an enum containing all allowed timeout modes.

```
typedef enum {
        GSM_ABS_TIMEOUT,
        GSM_INTERCHAR_DELAY,
        GSM_INVALID_TIMEOUT_MODE
} GSM_TimeoutMode_t;
```

4.4 GSM_ErrorCode_t

This type is an enum containing codes for all errors that may occur during GSM operations. Each method described within chapter 5 returns an error code.

```
typedef enum {
        GSM_EXEC_OK,
        GSM_SERIAL_ALREADY_OPEN,
        GSM_OPEN_ERROR,
        GSM_GET_PARAMS_ERROR,
        GSM_SET_PARAMS_ERROR,
        GSM_WRONG_DEVICE,
        GSM_CLOSE_ERROR,
        GSM_SIGNALS_ERROR,
        GSM_SERIAL_WRITE_ERROR,
        GSM_SERIAL_READ_ERROR,
        GSM_TIMEOUT_ERROR,
        GSM_SIM_NOT_INSERTED,
        GSM_PIN_ALREADY_SET,
        GSM_WAITING_PUK,
        GSM_WRONG_PIN,
        GSM_NOT_REGISTERED,
        GSM_WRONG_CLASS,
        GSM_CANT_DIAL,
        GSM_BAD_NR,
        GSM_STOP_VOICE_ERROR,
        GSM_PAUSE_DATA_ERROR,
        GSM_STOP_DATA_ERROR,
        GSM_GPRS_ERROR,
        GSM_PPP_ERROR,
        GSM_PPP_ALREADY_OPENED,
        GSM_TERM_GPRS_ERROR,
        GSM_PPP_ALREADY_CLOSED,
        GSM_TERM_PPP_ERROR,
        GSM_BAD_PARAMETERS,
        GSM_SMS_CONFIG_ERROR,
        GSM_CANT_SEND_SMS,
        GSM_UNKNOWN_ERROR = 500
} GSM_ErrorCode_t;
```

4.5 GSM_CallType_t

This type is an enum containing symbols for each type of data call, and is used in the GSM_StartDataCall function.

```
typedef enum {
        GSM_DATA_CALL,
        GSM_FAX_CALL,
        GSM_INVALID_CALL_TYPE
} GSM_CallType_t;
```

4.6 GSM_PauseAction_t

This type is an enum containing symbols used to specify the action to be performed in the *GSM_PauseDataCall()* function.

```
typedef enum {
        GSM_PAUSE_CALL,
        GSM_RESTORE_CALL,
        GSM_INVALID_ACTION
} GSM_PauseAction_t;
```

4.7 SMS_Format_t

This type is an enum containing symbols for each type of SMS, and is used in the *GSM_ConfigSMS* function.

```
typedef enum {
        SMS_PDU_FORMAT,
        SMS_TEXT_FORMAT,
        SMS_INVALID_FORMAT
} GSM_Format_t;
```

4.8 GSM_Numbering_t

This type is an enum containing symbols for each type of numbering format, and is a parameter in some functions.

```
typedef enum {
        SMS_INTERNATIONAL_NR,
        SMS_NATIONAL_NR,
```


```
                    SMS_INVALID_NUMBERING
} GSM_Numbering_t;
```

5 APIs description

The GSM library is integrated into the development environment so, in order to use the GSM library, you only have to include the header file into your application:

#include <GSM_lib.h>

5.1 GSM_InitSerialModem()

This function opens the connection to the modem's serial port and sets all needed parameters like flow control, parity check etcetera. Only a subset of the serial port settings is supported by this API: for example, the data word can be only seven bits or eight bits long, because all the others data lengths are not supported by the modem.

Prototype

GSM_ErrorCode_t GSM_InitSerialModem (char * dev, GSM_Baudrate_t speed, int options)

Parameters

<dev> It's the serial device to be open. It can be both a physic device and a virtual one (if the CMUX is enabled[1]).
<speed> Baudrate of the serial port. See type *baudrate_t*.
<options> It's an integer containing control settings for the modem serial port. It can be set using the following constants:

GSM_DEFAULT_SERIAL:	sets the default configuration (8n1 with flow control).
GSM_7DATA:	7 data bits.
GSM_8DATA:	8 data bits.
GSM_NO_PARITY:	no parity check.
GSM_ODD_PARITY:	odd parity check enabled.
GSM_EVEN_PARITY:	even parity check enabled.
GSM_1STOP:	1 stop bit.
GSM_2STOP:	2 stop bits.
GSM_FLOW_ON:	Hardware flow control activated.
GSM_FLOW_OFF:	Hardware flow control disabled.

[1] Please consult CMUX User Guide for more information

The allowed configurations for data bits, parity and stop bits are: 7e1, 7o1, 7n2, 8n1, 8e1, 8o1, 8n2. If a different configuration is issued, the default one will be set automatically . This configuration is set also if mutually exclusive flags are chosen at the same time (for example GSM_1STOP | GSM_2STOP). No error code is given if the default configuration is set due to a wrong parameters combination.
If GSM_FLOW_ON and GSM_FLOW_OFF are both enabled, the flow control is enabled; if they are both disabled, the flow control is disabled.

Return values

GSM_EXEC_OK	Connection to GSM module correctly established.
GSM_BAD_PARAMETERS	An illegal baudrate has been passed.
GSM_SERIAL_ALREADY_OPEN	The selected serial port has already been open by a previous instance of this function.
GSM_OPEN_ERROR	Can't open modem's serial port.
GSM_GET_PARAMS_ERROR	Can't retrieve port parameters.
GSM_SET_PARAMS_ERROR	Can't set port parameters.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Examples

GSM_InitSerialModem ("/dev/ttyS3", GSM_115200, GSM_8DATA | GSM_NO_PARITY | GSM_1STOP | GSM_FLOW_ON);

initializes the serial device "ttyS3" to 115200 bauds, with 8n1 configuration and with hardware flow control.

GSM_InitSerialModem ("/dev/ttyS3", GSM_115200, GSM_7DATA | GSM_EVEN_PARITY | GSM_2STOP | GSM_FLOW_ON);

initializes the serial device "ttyS3" to the default configuration (8n1) because 7e2 is an illegal configuration.

CMUX:
system("cmuxt –p /dev/ttyS3 –b 115200 -d");
…
GSM_InitSerialModem ("/dev/cmux1", GSM_115200, GSM_DEFAULT_SERIAL);

initializes the virtual device "/dev/cmux1" to the default configuration (8n1). This device is enabled by means of a system call that launches the CMUX daemon (on the physic tty at 115200 bauds). For further information about the CMUX syntax, check the CMUX documentation.

5.2 GSM_TermSerialModem()

This function terminates the serial modem connection, freeing all its resources and making it available for any other application.
NOTE: the GSM_InitPPPoverGPRS() functions calls GSM_TermSerialModem() internally, thus it's not needed after this API.

Prototype

GSM_ErrorCode_t GSM_TermSerialModem (char * dev)

Parameters

<dev> It's the serial device to be terminated.

Return values

GSM_EXEC_OK	The serial modem has been successfully closed.
GSM_CLOSE_ERROR	Can't close modem's serial port. It may mean that it has already been closed.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

GSM_TermSerialModem ("/dev/ttyS3");

terminates the */dev/ttyS3* serial connection (if it was previously opened).

5.3 GSM_SerialSignals()

This method exports the Linux ioctl() function, allowing the user to handle the physical serial port signals.

Prototype

GSM_ErrorCode_t GSM_SerialSignals(char * dev, int action, int * signals)

Parameters

<dev> It's the serial device which physical signals will be handled.
<action> It's the action to be performed. Possible values are:

- TIOCMGET: get the status of modem bits.
- TIOCMSET: set the status of modem bits (only output bits can be written).
- TIOCMBIC: clear the indicated modem bits (only output bits can be cleared).
- TIOCMBIS: set the indicated modem bits (only output bits can be set).

<signals> It's the pointer to an integer containing the TTY signals values. It is an output if TIOCMGET action is issued else it is an input. The masks to be used are:

- TIOCM_LE: DSR (data set ready/line enable).
- TIOCM_DTR: DTR (data terminal ready).
- TIOCM_RTS: RTS (request to send).
- TIOCM_ST: Secondary TXD (transmit).
- TIOCM_SR: Secondary RXD (receive).
- TIOCM_CTS: CTS (clear to send).
- TIOCM_CAR: DCD (data carrier detect).
- TIOCM_CD: *(see TIOCM_CAR)*.
- TIOCM_RNG: RNG (ring).
- TIOCM_RI: *(see TIOCM_RNG)*.
- TIOCM_DSR: DSR (data set ready).

This masks can be composed using bit a bit C operators (see examples below).

Return values

GSM_EXEC_OK	The action has been correctly performed.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_BAD_PARAMETERS	An invalid <action> has been specified.
GSM_SIGNALS_ERROR	The selected action has not correctly been performed (*ioctl* error). If this function is used on a CMUX virtual channel, this error code is returned.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

```
int signals;

/* Initialize the device */
GSM_InitSerialModem("/dev/ttyS3", GSM_115200, GSM_DEFAULT_SERIAL);

/* Read the signals of the open device */
GSM_Serial ("/dev/ttyS3", TIOCMGET, &signals);
```

cts = (signals & TIOCM_CTS)? 1 : 0;

gets the signals of the physical device ttyS3, and then retrieves (using the proper mask) the value of the CTS signal.

```
int signals, cts;

/* Initialize the device */
GSM_InitSerialModem("/dev/ttyS3", GSM_115200, GSM_DEFAULT_SERIAL);

signals = TIOCM_DTR |TIOCM_RTS;

/* Clear the DTR and RTS signals */
GSM_Serial ("/dev/ttyS3", TIOCMBIC, &signals);
/* Set the DTR and RTS signals */
GSM_Serial ("/dev/ttyS3", TIOCMBIS, &signals);
```

first of all the *signals* variable is defined using the *DTR* and *RTS* masks; then it's used to clear/set both of them.
NOTE: DTR and RTS are outputs. Inputs (such as CTS) can only be read. Write operations on inputs won't take effect.

5.4 GSM_SendATcommand()

This function sends a single AT command to the GE863 Telit module through the serial port. The parsing of the response can be managed by the user by means of the *response* parameter. With the proper settings also a comparison with the expected response can be performed.
NOTE: a flush of the serial port is performed when the function starts. Any unread character remaining from a previous action will be lost.

Prototype

```
GSM_ErrorCode_t GSM_SendATcommand (char * dev, char * cmd, int timeout,
                                   GSM_TimeoutMode_t mode,    char ** response,
                                   char * expected)
```

Parameters

<dev>	It's the serial device where the AT command will be sent.
<cmd>	AT command to be sent. Since it is a string, double quotes have to be written with the escape character (\").
<timeout>	Customizable timeout. Time unit is the decisecond and no default value is set. Its meaning changes according to the value of <mode> parameter (see

	below). If it is set to 0, the functions returns immediately after the write operation, and no read is performed (negative values cause the same behavior).
<mode>	This parameter allows to change the meaning of the <timeout> parameter. If GSM_ABS_TIMEOUT is passed, the function returns *timeout* deciseconds after its call (even if further characters are still being received!); else, if GSM_INTERCHAR_DELAY is passed, the function returns *timeout* deciseconds after the last received character (inter-character delay).
<response>	It is pointer to the string containing the module's response, a buffer that can contain up to MAX_RESPONSE_SIZE-1 characters (255); if more characters are received, they are discarded (but not read). It can be checked by the user by means of string management APIs (it's always terminated by the '\0' character).
<expected>	It is a string where the user can store an expected substring of the response (a case sensitive compare is performed). When it's recognized, the function returns without to wait for the timeout expiration. It can be NULL; in this case, the function returns normally after the set timeout. In both cases the response is returned. **NOTE**: any character received after the expected string is not read; the user can handle this possibility using the *GSM_ReadResponse()* API.

Return values

GSM_EXEC_OK	The AT command has been correctly sent. If the "expected" parameter is specified, GSM_EXEC_OK is returned only if the expected string has been recognized.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_SERIAL_WRITE_ERROR	An error during the write operations on the modem tty occurred.
GSM_TIMEOUT_ERROR	If "expected" parameter is not set, it means that a timeout occurred with any response (no characters received). Otherwise it means that the string hasn't been recognized, and this error code is returned even if any other string has been received. Anyway, the response can be processed by the user by means of the "response" string.

Example

```
char *response;
response = (char *) malloc (MAX_RESPONSE_SIZE);
memset(response,'\0', MAX_RESPONSE_SIZE);

GSM_SendATcommand ("/dev/cmux1", "AT", 30, GSM_ABS_TIMEOUT, &response,"OK");
```

sends command “AT” through the */dev/cmux1* virtual device with a maximum absolute timeout of 30 deciseconds (3 seconds) and returning as soon as the “OK” substring is recognized. The modem’s response is stored within the *response* string (if this parameter is not NULL).

5.5 GSM_ReadResponse()

This function performs a read on the modem’s serial port for a time period depending on the setting of the <timeout> parameter (see below for details) and returns the read string. It can be used when unsolicited responses are expected or to read modem responses after a *GSM_SendATcommand()* call with *timeout* set to 0 (see above).
If unsolicited messages are enabled, this function can be used to handle them.

Prototype

GSM_ErrorCode_t GSM_ReadResponse (char * dev, int timeout, GSM_TimeoutMode_t mode,
char ** response, char * expected)

Parameters

<dev>	It’s the serial device where the read will be performed.
<timeout>	Customizable timeout. Time unit is the decisecond and no default value is set. Its meaning changes according to the value of <mode> parameter (see below).
<mode>	This parameter allows to change the meaning of the <timeout> parameter. If GSM_ABS_TIMEOUT is passed, the function returns *timeout* deciseconds after its call (even if further characters are still being received!); else, if GSM_INTERCHAR_DELAY is passed, the function returns *timeout* deciseconds after the last received character (inter-character delay).
<response>	It is pointer to the string containing the module’s response, a buffer that can contain up to MAX_RESPONSE_SIZE-1 characters (255); if more characters are received, they are discarded (but not read). It can be checked by the user by means of string management APIs (it’s always terminated by the ‘\0’ character).
<expected>	It is a string where the user can store an expected substring of the response (a case sensitive compare is performed). When it’s recognized, the function returns without waiting for the timeout to expire. It can be NULL; in this case, the function returns normally after the set timeout. In both cases the response is returned. **NOTE**: any character received after the expected string is not read; the user can handle this possibility using another call of *GSM_ReadResponse()* API without setting the expected parameter.

Return values

GSM_EXEC_OK	At least a character has been received within the set timeout, and the received string is stored in the response pointer. If the "expected" parameter is specified, GSM_EXEC_OK is returned only if the expected string has been recognized.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	If "expected" parameter is not set, it means that a timeout occurred with any response (no characters received). Otherwise it means that the string hasn't been recognized, and this error code is returned even if any other string has been received. Anyway, the response can be processed by the user by means of the "response" string.

Examples

```
char *response;
response = (char *) malloc (MAX_RESPONSE_SIZE);
memset(response,'\0', MAX_RESPONSE_SIZE);

while(readResponse ("/dev/ttyS3", 1, GSM_INTERCHAR_DELAY, &response, NULL) ==
                                GSM_TIMEOUT_ERROR);
        if(strstr(response,"RING") != NULL) printf ("OK\n");
```

waits until at least a character is received from the */dev/ttyS3* device and compares the response with the RING string (that occurs when a call is received). GSM_INTERCHAR_DELAY is used instead of GSM_ABS_TIMEOUT in order to avoid the splitting of the response in two different samples (if the absolute timeout expires just while the response is being received).

```
if(receiveResponse ("/dev/ttyS3", 100, GSM_INTERCHAR_DELAY, NULL,
                                "RING")==GSM_EXEC_OK ) printf ("OK\n");
```

listens to the serial port ten seconds and return as soon as a RING string (that occurs when a call is received) is recognized. No response buffer is needed.

5.6 GSM_SendData()

This function sends binary data through the serial device. It is used to exchange data when the modem is in data mode.

Prototype

GSM_ErrorCode_t GSM_SendData (char * dev, void * data, unsigned int len,
unsigned int * written_bytes)

Parameters

<dev>	It's the serial device where the data will be sent.
<data>	Pointer to the buffer containing the data to be sent.
<len>	Number of bytes to be sent. It is limited only by the buffer size (to be allocated by the user).
<written_bytes>	Pointer to the integer where the function will store the number of bytes correctly sent.

Return values

GSM_EXEC_OK	All the *len* bytes have correctly been sent.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_SERIAL_WRITE_ERROR	One or more bytes haven't correctly been sent. In this case the user has to check the *written_bytes* parameter.

Example

```
int written = -1;
char * data;

data = (char *) malloc(SIZE_OF_THE_BUFFER);
…
[Fill the buffer with the data to be sent, for example with data retrieved from a file]
…
[Go to data mode, for example starting a data call]
…
if(GSM_SendData("/dev/ttyS3", (void *) data, SIZE_OF_THE_BUFFER, &written) ==
                    GSM_SERIAL_WRITE_ERROR)
        printf("\nBytes remaining: %d", SIZE_OF_THE_BUFFER - written);
```

sends SIZE_OF_THE_BUFFER bytes contained in the *data* buffer through the */dev/ttyS3* device, and if something goes wrong prints the number of remaining bytes.

5.7 GSM_ReceiveData()

This function receives binary data through the serial device. It is used to exchange data when the modem is in data mode.

Prototype

GSM_ErrorCode_t GSM_ReceiveData (char * dev, void * data, unsigned int len,
unsigned int * read_bytes)

Parameters

<dev>	It's the serial device where the read will be performed.
<data>	Pointer to the buffer where received data will be stored.
<len>	Number of bytes to be read. It is limited only by the buffer size (to be allocated by the user.
<read_bytes>	Pointer to the integer where the function will store the number of bytes correctly read.

Return values

GSM_EXEC_OK	All the *len* bytes have correctly been read.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_SERIAL_READ_ERROR	The number of read bytes is shorter than *len*. It may mean that there aren't characters to read, anymore.

Example

```
int read = -1;
char * data;

data = (char *) malloc(SIZE_OF_THE_BUFFER);
memset(data,00,SIZE_OF_THE_BUFFER);
…
[Go to data mode, for example starting a data call]
…
if(GSM_ReceiveData("/dev/ttyS3", (void *) data, SIZE_OF_THE_BUFFER, &read) ==
                    GSM_SERIAL_READ_ERROR)
        printf("\n%d bytes received instead of %d", read, SIZE_OF_THE_BUFFER);
```

receives up to SIZE_OF_THE_BUFFER bytes from the */dev/ttyS3* device and stores them within the *data* buffer; then prints the number of received bytes.

5.8 GSM_SendEscape()

This function sends to the modem the escape sequence (+++) in order to bring it back to command mode if it is in data mode. The *Escape Prompt Delay* must be set to the factory default (ATS12=50; 1 second delay) or this API may not work correctly.

Prototype

GSM_ErrorCode_t GSM_sendEscape (char * dev, char ** response)

Parameters

<dev>	It's the serial device where the escape sequence (+++) will be sent.
<response>	It is pointer to the string containing the module's response. This string gives to the user the capability to check if the response is the desired one. The valid responses are "\r\nOK\r\n" and "\r\nNO CARRIER\r\n" and have different meanings: the first is returned when a data session is suspended (i.e. GSM data call) and the second occurs when it is terminated (i.e. PPP). Instead, NULL is returned if no valid answer is recognized.

Return values

GSM_EXEC_OK	The sequence has been correctly sent and the response received from the modem contains 'OK' or 'NO CARRIER'. The modem is now in command mode. A further check to the response is in charge of the user that can process the *response* parameter.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	None of the expected responses ('OK' or 'NO CARRIER') has been received from the modem in 10 secs. It may mean that the modem was already in command mode.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

```
GSM_ErrorCode_t ret;
char *response;
```

*response = (char *) malloc (20);*
memset(response,'\0', 20);
…
[send commands to initialize a data communication, for example AT#SKTD]
[exchange data]
…
ret=GSM_sendEscape("/dev/ttyS3", response):

sends the escape sequence through */dev/ttyS3* device after a data session. If GSM_EXEC_OK is returned, the user can perform a further check on the response string in order to verify if the desired response ("OK" or "NO CARRIER") has been received.

5.9 GSM_InsertPIN()

This function sets the PIN code to the GSM SIM.

Prototype

GSM_ErrorCode_t GSM_InsertPIN (char * dev, char* pin, int *remaining)

Parameters

<dev>	It's the serial device used for this API.
<pin>	PIN number to be inserted.
<remaining>	In case of wrong pin, it is updated with the remaining attempts.

Return values

GSM_EXEC_OK	The PIN has been correctly set.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	Timeout expired.
GSM_SIM_NOT_INSERTED	No SIM inserted.
GSM_PIN_ALREADY_SET	Pin already set. No action performed.
GSM_WAITING_PUK	Three wrong PIN insertions have already been done, so PUK code has to be given to unlock the SIM.
GSM_WRONG_PIN	Wrong PIN inserted.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

int remaining = -1;
GSM_InsertPIN((“/dev/ttyS3”, “1234”, &remaining);
if(remaining != -1) printf(“\nRemaining attempts: %d”, remaining);

inserts the PIN code (using */dev/ttyS3* device). If it’s wrong, the remaining attempts are stored in the *remaining* variable (if this parameter is not NULL).

5.10 GSM_StartVoiceCall ()

This function starts a GSM voice call. The check for the network registration is in charge of the user.

Prototype

GSM_ErrorCode_t GSM_StartVoiceCall (char * dev, char * number)

Parameters

<dev>	It’s the serial device used for this API.
<number>	It is a string containing the phone number to call. If an international call is issued, it must include the international prefix (a substring starting with the ‘+’ character).

Return values

GSM_EXEC_OK	GSM voice call correctly started. This code is returned only if the receiver answers to the call, otherwise GSM_CANT_DIAL is returned (i.e. when the receiver is busy).
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	Timeout expired (no response from the modem).
GSM_BAD_NR	The phone number has an illegal format.
GSM_CANT_DIAL	The dialing operation did not succeed (i.e. wrong number, busy or no response). This function waits about 80 seconds after the dial for the receiver’s response.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

GSM_ErrorCode_t result;
result = GSM_StartVoiceCall("/dev/ttyS3", "+39XXXYYYYYYY");

starts a voice call to the number specified (sending commands through */dev/ttyS3* device). The error code is stored in the *result* variable.

5.11 GSM_StopVoiceCall ()

This function hangs up the GSM call, freeing all its resources.

Prototype

GSM_ErrorCode_t GSM_StopVoiceCall (char * dev)

Parameters

<dev> It's the serial device used for this API. It can be different from the one used to start the call.

Return values

GSM_EXEC_OK	The GSM voice call has been stopped with no errors.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	Timeout expired.
GSM_STOP_VOICE_ERROR	Can't stop GSM call. It may mean that it has already been stopped.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

GSM_ErrorCode_t result;
result = GSM_StopVoiceCall("/dev/ttyS3");

stops the voice call previously started.

5.12 GSM_StartDataCall ()

This function starts a GSM data call. The check for the network registration is in charge of the user.
WARNING: this function doesn't check if a data call has already been started. In this case the AT commands sent by this API would be addressed to the data socket and won't receive any response (GSM_TIMEOUT_ERROR will be returned).

Prototype

GSM_ErrorCode_t GSM_StartDataCall	(char * dev, char* number, GSM_CallType_t type, GSM_Baudrate_t * rate)

Parameters

<dev>	It's the serial device used for this API.
<number>	It is a string containing the phone number to call. If an international call is issued, it must include the international prefix (a substring starting with the '+' character).
<type>	Flag that makes possible to the user to choose between a data call or a fax call.
<rate>	Pointer to the location where, if the call starts successfully, is stored the baudrate of the communication.

Return values

GSM_EXEC_OK	GSM voice call correctly started. This code is returned only if the receiver answers to the call; else GSM_CANT_DIAL is returned (i.e. when the receiver is busy).
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	Timeout expired (no module's response).
GSM_WRONG_CLASS	"type" parameter specifies an illegal class.
GSM_CANT_DIAL	The dialing operation did not succeed (i.e. wrong number, busy or no response). This function waits about 80 seconds after the dial for the receiver's response.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

```
GSM_ErrorCode_t result;
GSM_Baudrate_t rate;
result = GSM_StartDataCall("/dev/ttyS3", "+39XXXYYYYYYY", GSM_DATACALL, &rate);
```

starts a data call to the number specified (data format) sending commands through */dev/ttyS3* device. The error code is stored in the *result* variable, and in case of successful attempt the baudrate of the connection is stored in the *rate* variable.

5.13 GSM_PauseDataCall ()

This function suspends temporarily a data call, giving the user the possibility to send AT commands. With the proper value of the <pause_restore> parameter the user can restore the call.
WARNING: in PAUSE mode this function sends an escape sequence, so it will terminate any other data connection (for example PPP).

Prototype

GSM_ErrorCode_t GSM_PauseDataCall (char * dev, GSM_PauseAction_t pause_restore)

Parameters

<dev>	It's the serial device used for this API.
<pause_restore>	If it's set to GSM_PAUSE_CALL a suspension is issued; if it is set to GSM_RESTORE_CALL, the communication is restored.

Return values

GSM_EXEC_OK	The GSM data connection has been paused/restored with no errors (**WARNING**: even in case of improper call of this function!!!).
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_PAUSE_DATA_ERROR	Can't suspend/restore GSM data call. If GSM_PAUSE_CALL action is issued, this error code is returned whenever the module is in command mode (and then when no data call is active); else, if GSM_RESTORE_CALL action is issued, it's returned whenever the module is in data mode (or, anyway, when there's no data call to be restored).
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Examples

GSM_ErrorCode_t result;
result = GSM_PauseDataCall ("/dev/ttyS3", GSM_PAUSE_CALL*);*

suspends temporarily the data call previously started. If the execution is successful, the user can now send AT commands to the module.

GSM_ErrorCode_t result;
result = GSM_PauseDataCall (“/dev/ttyS3”, GSM_RESTORE_CALL*);*

restores the data call previously suspended.

5.14 GSM_StopDataCall ()

This function hangs up the GSM data call, freeing all its resources.
WARNING: this function sends an escape sequence, so it will terminate any other data connection (for example PPP).

Prototype

GSM_ErrorCode_t GSM_StopDataCall (char * dev)

Parameters

<dev>	It’s the serial device used for this API.

Return values

GSM_EXEC_OK	The GSM data connection has been terminated with no errors. It’s returned even if there isn’t any active data call.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	Timeout expired.
GSM_STOP_DATA_ERROR	Can’t stop GSM data call.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

GSM_ErrorCode_t result;
result = GSM_StopDataCall (“/dev/ttyS3”);

stops the data call previously suspended, sending commands through */dev/ttyS3* device.

5.15 GSM_InitGPRS()

This function opens a GPRS connection according to the chosen context and using the internal PPP stack of the GE863 module. If the function succeeds, the resulting IP address is returned.

Prototype

GSM_ErrorCode_t GSM_InitGPRS (char * dev, char* APN, char** IP, char context)

Parameters

<dev>	It's the serial device used for this API.
<APN>	String containing the Access Point name to which the module has to be connected.
<IP>	Pointer to a string containing the IP number assigned to the Telit Module by the APN. It can't be NULL.
<context>	The GSM module can store up to five contexts. For this parameter, only '1', '2', '3', '4' and '5' characters are admitted. Contexts '2', '3', '4' and '5' work only with multisocket AT commands[2].

Return values

GSM_EXEC_OK	The GPRS connection has been activated with no errors.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	Timeout expired (the modem could be in data mode).
GSM_NOT_REGISTERED	The SIM is not registered to the GPRS network yet. If the PIN has been correctly inserted, simply try again after a few seconds.
GSM_GPRS_ERROR	The attempt to start the GPRS connection failed. It may depend on the field level or on an invalid APN. **NOTE**: the function returns as soon as the response of the network is received. The maximum response time is 180 seconds.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

```
GSM_ErrorCode_t result;
char *IP;
IP = (char *) malloc (MAX_RESPONSE_SIZE);
memset(IP,'\0', MAX_RESPONSE_SIZE);

result = GSM_InitGPRS("/dev/ttyS3", "ibox.tim.it", &IP, '2');
```

[2] Please consult Easy GPRS User Guide for more information

initializes a GPRS connection to the TIM operator on the context ‘2’. If the connection is successful, the resulting IP is stored in the *IP* string.

5.16 GSM_TermGPRS()

This function terminates a GPRS connection, freeing all its resources.

Prototype

GSM_ErrorCode_t GSM_TermGPRS (char * dev, char context)

Parameters

<dev>	It’s the serial device used for this API.
<context>	GPRS context to be terminated. For this parameter, only ‘1’, ‘2’, ‘3’, ‘4’ and ‘5’ characters are admitted.

Return values

GSM_EXEC_OK	The GPRS connection has been terminated with no errors.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	Timeout expired without response from the modem.
GSM_TERM_GPRS_ERROR	Can’t close GPRS connection. It may mean that the specified context it has already been closed.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

GSM_ErrorCode_t result;
result = GSM_TermGPRS(“/dev/ttyS3”, ‘1);

terminates the GPRS context ‘1’ connection previously open.

5.17 GSM_InitPPPoverGPRS()

This function opens a PPP connection according to the chosen APN (context '1' is used), and starts a pppd daemon in order to create a ppp0 interface under Linux. The modem's serial port is closed in order to make it available to the pppd, so the user has to open it again before to call further APIs.
NB: the pppd is launched using the call *pppd file /etc/ppp/peers/options*: it means that the options (i.e. serial device to connect to, baudrate, authentication, flow control etcetera) are retrieved from the *options* file located within the */etc/ppp/peers* directory. The pap-secrets file, contained in the */etc/ppp/* directory is needed for the authentication.
The user has to edit the */etc/ppp/peers/options* file before to call this API in order to check if the serial device and the speed are correct:

```
vi /etc/ppp/peers/options
```

For example, if you call the GSM_InitGPRSoverPPP() API on the /dev/ttyS3 (115200 baud):

```
# Serial Device to which the GPRS phone is connected
/dev/ttyS3
# Serial port line speed
115200
```

If you don't find these files within your Linux Release, please ask for Telit Technical Support in order to get the needed files. These files can easily be integrated in the Linux Release following the procedure of download file on GE863-PRO3.[3]

Prototype

GSM_ErrorCode_t GSM_InitPPPoverGPRS (char * dev, char* APN, unsigned int ppp_timeout)

Parameters

<dev>	It's the serial device used for this API.
<APN>	String containing the Access Point name to which the module has to be connected.
<ppp_timeout>	Since the time needed by the pppd daemon to set up the interface, this parameter gives the user the capability to decide the maximum period after which the ppp0 interface has to be ready. If not, the API kills the process and restores the command mode.

Return values

GSM_EXEC_OK	The PPP connection has been activated with no errors.

[3] For more information please refer to the Linux Software User Guide

GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	No response received from the modem. **NOTE**: if *ppp_timeout* expires GSM_PPP_ERROR is returned.
GSM_NOT_REGISTERED	The SIM is not registered to the GPRS network yet. If the PIN has been correctly inserted, simply try again after a few seconds.
GSM_PPP_ALREADY_OPENED	A PPP connection is already up.
GSM_PPP_ERROR	The attempt to start the PPP connection failed or *ppp_timeout* expired. It may depend on the field level.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

GSM_ErrorCode_t result;
result = GSM_InitPPPoverGPRS("/dev/ttyS3", "ibox.tim.it", 15);

initializes a PPP connection to the TIM operator and launches the PPP daemon in order to create a Linux interface. The *options* file (in the */etch/ppp/peers* directory) and the *pap-secrets* file (in the */etc/ppp* directory) must be properly configured. If the ppp0 interface isn't set in 15 seconds, the API fails.

5.18 GSM_TermPPPoverGPRS()

This function terminates the PPP connection, freeing all its resources and bringing back the modem to command mode.

Prototype

GSM_ErrorCode_t GSM_TermPPPoverGPRS(char * dev)

Parameters

<dev> It's the serial device used for this API.

Return values

GSM_EXEC_OK	The PPP connection has been terminated with no errors.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	Timeout expired with no module's response.

GSM_PPP_ALREADY_CLOSED	There are not PPP connections to terminate.
GSM_TERM_PPP_ERROR	Can't close PPP connection. It may mean that a problem occurred in killing PPPD process or in sending escape sequence.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

GSM_ErrorCode_t result;
result = GSM_TermPPPoverGPRS("/dev/ttyS3");

terminates the PPP connection previously started. It also kills the *pppd* process removing the ppp0 network interface.

5.19 GSM_CheckPPPoverGPRS()

This function checks if a PPP Linux interface is up.
WARNING: it is used by *GSM_InitPPPoverGPRS()* (*GSM_TermPPPoverGPRS()*) in order to verify if the initialization (termination) was successful. The expected interface name is "ppp0", so any other PPP connection created by the user may make the check fail (if it was created first).

Prototype

GSM_Boolean_t GSM_CheckPPPoverGPRS()

Parameters

none

Return values

GSM_True	The PPP interface is up.
GSM_False	The PPP interface is down.

Example

GSM_Boolean_t result;

```
result = GSM_CheckPPPoverGPRS();

if(result == GSM_True)
                printf("\nppp0 interface is up");
else
                printf("\nppp0 interface is down");
```

checks if the "**ppp0**" interface is up.

5.20 GSM_ConfigSMS()

This function handles the messages configuration, setting parameters such as the message format or the service center address. This command can be issued also when the SIM is not registered, yet.

Prototype

GSM_ErrorCode_t GSM_ConfigSMS (char * dev, SMS_format_t format, char * SC_addr, GSM_Numbering_t type)

Parameters

<dev>	It's the serial device used for this API.
<format>	Parameter containing the SMS format. Two choices are possible: PDU format and text format.
<SC_addr>	Address of the service center (a phone number).
<type>	Flag that makes possible to the user to choose between a national or international numbering scheme.

Return values

GSM_EXEC_OK	The configuration succeeded with no errors.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	Timeout expired.
GSM_BAD_PARAMETERS	One or more parameters are invalid.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

```
GSM_ErrorCode_t result;
result = GSM_ConfigSMS("/dev/ttyS3", SMS_TEXT_FORMAT, "+393359609600",
```

GSM_INTERNATIONAL_NR);

configures the module to send text short messages, allowing international numbers and setting the Service Centre (in this case TIM Italy).

5.21 GSM_SendSMS()

This function sends an SMS using the settings specified with the *GSM_ConfigSMS()* function (make sure to call it at least one time in order to set the desired parameters).

Prototype

GSM_ErrorCode_t GSM_SendSMS (char * dev, char* number, char* message)

Parameters

<dev>	It's the serial device used for this API.
<number>	It is the phone number which the SMS will be sent.
<message>	It is the string containing the message to be sent. Its max permitted length is 1530. The module will split it in the necessary number of messages. If PDU mode is issued, the PDU message construction is in charge of the user.

Return values

GSM_EXEC_OK	The SMS has been delivered to the SC with no error.
GSM_WRONG_DEVICE	The selected device has not been opened.
GSM_TIMEOUT_ERROR	Timeout expired with no response.
GSM_BAD_PARAMETERS	One or more parameters are invalid.
GSM_CANT_SEND_SMS	A problem occurred in sending SMS: the delivery to service center failed. It may mean that the configured SC number is wrong. In the worst case the service center response can take 180 seconds.
GSM_UNKNOWN_ERROR	An unpredictable error occurred.

Example

Text mode:
GSM_ErrorCode_t result;
result = GSM_SendSMS("/dev/ttyS3", "+39XXXYYYYYYY", "Greetings from Telit!");

sends to the specified number the message “Greetings from Telit!”.

PDU mode:
GSM_ErrorCode_t result;
result = GSM_SendSMS(“/dev/ttyS3”, “xyz[4]*”,*
“0011000C9193430700000000F5AA054765726461”);

sends to the number “+393470000000” the message “Gerda”, using the service center of the Wind operator. In this case the number parameter isn’t needed, because the destination number is embedded within the PDU string.

5.22 GSM_PrintLibVersion()

This function prints the version of the library.

Prototype

char * GSM_PrintLibVersion (void)

Parameters

none

Return values

A string containing the version of the library (i.e. “C0.00.04”).

Example

GSM_PrintLibVersion();

prints to standard output the message “Ver:XX.YY.ZZ” where XX.YY.ZZ is the version of the library.

[4] In this case the parameter can have any value

6 List of acronyms

Term	Definition
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
IP	Internet Protocol
PDP	Packet Data Protocol
PDU	Protocol Data Unit
PIN	Personal Identification Number
PPP	Point to Point Protocol
PUK	Personal Unblocking Key
SC	Service Center
SIM	Subscriber Identity Module
SMS	Short Message Service
TCP	Transmission Control Protocol