

# GE863-PRO<sup>3</sup> Linux GSM Library User Guide

1vv0300782 Rev. 1 - 2011/01/17



## 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.

Copyright: Transmittal, reproduction, dissemination and/or editing of this document as well as utilization of its contents and communication thereof to others without express authorization are prohibited. Offenders will be held liable for payment of damages. All rights are reserved.  
Copyright © Telit Communications SpA 2011.©



## Applicable Products

| Product                              | Part Number |
|--------------------------------------|-------------|
| GE863-PRO <sup>3</sup> with Linux OS | 3990250698  |
| GG863-SR                             |             |

GSM Library Version

C0.00.08



# Contents

- 1 Introduction ..... 6**
  - 1.1 Scope .....6
  - 1.2 Audience .....6
  - 1.3 Contact Information, Support.....6
  - 1.4 Product Overview .....6
  - 1.5 Document Organization .....7
  - 1.6 Text Conventions .....9
  - 1.7 Related Documents .....10
  - 1.8 Document History .....10
- 2 Library setup ..... 11**
- 3 APIs summary..... 13**
- 4 Data types ..... 15**
  - 4.1 GSM\_Boolean\_t.....15
  - 4.2 GSM\_Baudrate\_t.....15
  - 4.3 GSM\_TimeoutMode\_t.....15
  - 4.4 GSM\_ErrorCode\_t.....16
  - 4.5 GSM\_PowerAction\_t.....17
  - 4.6 GSM\_ConfItem\_t .....17
  - 4.7 GSM\_CallType\_t.....17
  - 4.8 GSM\_PauseAction\_t .....18
  - 4.9 SMS\_Format\_t.....18
  - 4.10 GSM\_Numbering\_t .....18
- 5 APIs description ..... 19**
  - 5.1 GSM\_Configure() .....19
  - 5.2 GSM\_PowerOnOff() .....21
  - 5.3 GSM\_InitSerialModem() .....22
  - 5.4 GSM\_TermSerialModem().....24
  - 5.5 GSM\_SerialSignals() .....25
  - 5.6 GSM\_SendATcommand().....27



5.7 GSM\_ReadResponse().....30

5.8 GSM\_SendData().....33

5.9 GSM\_ReceiveData() .....34

5.10 GSM\_SendEscape().....35

5.11 GSM\_InsertPIN() .....36

5.12 GSM\_StartVoiceCall () .....37

5.13 GSM\_StopVoiceCall () .....38

5.14 GSM\_StartDataCall () .....39

5.15 GSM\_PauseDataCall () .....40

5.16 GSM\_StopDataCall () .....41

5.17 GSM\_InitGPRS() .....42

5.18 GSM\_TermGPRS() .....44

5.19 GSM\_InitPPPOverGPRS().....45

5.20 GSM\_TermPPPOverGPRS() .....47

5.21 GSM\_CheckPPPOverGPRS() .....48

5.22 GSM\_ConfigSMS().....49

5.23 GSM\_SendSMS().....50

5.24 GSM\_PrintLibVersion() .....51

6 Using GSM library with CMUX..... 52

6.1 CMUX and GSM\_InitPPPOverGPRS() API.....52

7 List of acronyms..... 53



# 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<sup>3</sup> module and need to use the GSM functionalities. With this library Telit offers 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<sup>3</sup> is an innovation to the quad-band, RoHS compliant GE863 product family which includes a powerful ARM9™ 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<sup>3</sup> 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 3, Library setup” describes briefly how to import the GSM APIs within a project.
- “Chapter 4, APIs summary” contains a list of the APIs provided by the GSM library.
- “Chapter 5, Data types” provides a description of the types used within the library.
- “Chapter 6, APIs” provides a detailed description of the methods of the GSM library.



## 2 “Chapter 7, Using GSM library with CMUX

The modem’s physical serial channel can be multiplexed into three virtual serial devices. This is done by the CMUX application [4] and can be useful if you need to send AT commands during a data connection (for example PPP).

From Linux console, you can launch the CMUX service by issuing for example

```
# cmuxt -p /dev/[modem tty] -b 115200 -d
```

After some seconds channels */dev/cmux1*, */dev/cmux2* and */dev/cmux3* become available.

### 2.1 CMUX and GSM\_InitPPPOverGPRS() API

Normally, within the *GSM\_PPPOverGPRS()* API, the connection to the current device is terminated, in order to make it available to PPPD daemon.

Nevertheless, when recent CMUX versions (1.2.0 or higher) are used the user don’t need to reopen the channel.

From Linux console, you can check the CMUX version by issuing

```
# cmuxt -v
```

- 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 1w0300781 [4] and TelitGE863PRO3 Linux Development Environment User Guide 1V0300780 [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<sup>3</sup>.











## 4 APIs summary

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

| Functionality Group         | API                   | Notes   |
|-----------------------------|-----------------------|---|
| Configuration               | GSM_Configure()       | Configures the GSM library.   |
| Power management            | GSM_PowerOnOff()      | Switches on/off or resets the GSM module.                                       |
| 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  |



## GE863-PRO3 Linux GSM Library User Guide

2011/01/17

|                     |                        |  |
|---------------------|------------------------|--|
|                     |                        | 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.   |





```

        GSM_INVALID_TIMEOUT_MODE
    } GSM_TimeoutMode_t;

```

## 5.4 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_NOTHING_TO_DO = 400,

```



```
GSM_UNKNOWN_ERROR = 500
} GSM_ErrorCode_t;
```

## 5.5 GSM\_PowerAction\_t

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

```
typedef enum {
    GSM_POWER_ON,
    GSM_POWER_OFF,
    GSM_POWER_RESET,
    GSM_INVALID_POWER_ACTION
} GSM_PowerAction_t;
```

## 5.6 GSM\_Confltem\_t

This type is an enum containing symbols used to specify the item to be configured by the *GSM\_Configure()* function.

```
typedef enum {
    GSM_CONF_TIMEOUT_FACTOR,
    GSM_CONF_PWRMON_PIN,
    GSM_CONF_ONOFF_PIN,
    GSM_CONF_RESET_PIN,
    GSM_CONF_LIST_SEPARATOR,
    GSM_CONF_RESERVED1,
    GSM_INVALID_CONF_ITEM
} GSM_Confltem_t;
```

## 5.7 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,
```







## GE863-PRO3 Linux GSM Library User Guide

2011/01/17

|                         |  |
|-------------------------|--|
| GSM_CONF_ONOFF_PIN      | Number of the GPIO connected to the ON# pin of the GSM modem (integer type, range 0-95).   |
| GSM_CONF_RESET_PIN      | Number of the GPIO connected to the Reset pin of the GSM modem (integer type, range 0-95).   |
| GSM_CONF_LIST_SEPARATOR | Separator character used by GSM_SendATcommand() and GSM_ReadResponse() functions in order to tokenize the <i>expected</i> parameter into several expected substrings. If 0 is passed, the “multiple expected” feature is disabled. |
| GSM_CONF_RESERVED1      | Configuration item reserved for future use.  |

<newSetting> It is the new value of the configurable parameter specified by the <item> identifier. Depending on <item> a different data type is expected. An internal casting is performed, so if wrong data type is given unexpected errors may occur. In order to avoid compilation warnings, make sure to cast the value to *{void\*}*.

In the following list are reported data types, range and default value (that is the value kept by the parameter when it's not configured) for each <item> value.

|                         |   |
|-------------------------|---|
| GSM_CONF_TIMEOUT_FACTOR | Unsigned integer value, expressed in percentage. Values smaller than 100 (factor < 1) are rejected. Default: 100 (100% = factor 1). <u>Passed by value.</u> |
| GSM_CONF_PWRMON_PIN     | Unsigned integer type, range 0-95. Default: 85. <u>Passed by value.</u>   |
| GSM_CONF_ONOFF_PIN      | Unsigned integer type, range 0-95. Default: 73. <u>Passed by value.</u>   |
| GSM_CONF_RESET_PIN      | Unsigned integer type, range 0-95. Default: 72. <u>Passed by value.</u>   |
| GSM_CONF_LIST_SEPARATOR | Char type (range 0-255). Default: 0 (“multiple expected” feature disabled). <u>Passed by value.</u>   |

### Return values

|                    |  |
|--------------------|--|
| GSM_EXEC_OK        | The configuration has been performed successfully. |
| GSM_BAD_PARAMETERS | One or more parameters are invalid.                |
| GSM_UNKNOWN_ERROR  | An unpredictable error occurred.                   |



## Examples

```
GSM_ErrorCode_t result;
result = GSM_Configure(GSM_CONF_TIMEOUT_FACTOR, (void *) 110);
```

increases all the AT commands timeouts by a factor of 1.1. Though the parameter type is *void\**, the <newSetting> parameter is passed by value.

```
GSM_ErrorCode_t result;
result = GSM_Configure(GSM_CONF_PWRMON_PIN, (void *) 41);
```

configures PWRMON pin to GPIO 41 (corresponding to PB9). Though the parameter type is *void\**, the <newSetting> parameter is passed by value.

```
GSM_ErrorCode_t result;
result = GSM_Configure(GSM_CONF_LIST_SEPARATOR, (void *) '!');
```

activates the “multiple expected” feature in *GSM\_SendATcommand()* and *GSM\_ReadResponse()* functions. Expected strings can be separated by the ‘!’ character. Though the parameter type is *void\**, the <newSetting> parameter is passed by value.

## 6.2 GSM\_PowerOnOff()

This function manages the power of the GSM module, by means of the GPIOs connected to the pins PWRMON, ON# and RESET of the GSM module.

Be aware that by default this function uses the GPIOs configuration of the Telit Pro3 EVK. If your hardware configuration is different, you have to call the *GSM\_Configure()* API before using *GSM\_PowerOnOff()*. For further information about GPIOs, please check TelitGE863-PRO<sup>3</sup> Hardware User Guide 1v0300773a [2].

The execution of this function takes normally a few seconds, but it can return immediately if no action is needed (for example if you issue a power-on and the modem is already on).

### Prototype

```
GSM_ErrorCode_t GSM_PowerOnOff (GSM_PowerAction_t action)
```

### Parameters







- 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("cmux -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.

## 6.4 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.

**NOTE:** if you're using CMUX please check chapter 7.

### Prototype

```
GSM_ErrorCode_t GSM_TermSerialModem (char * dev)
```

### Parameters

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

### Return values

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

### Example

```
GSM_TermSerialModem ("/dev/ttyS3");
```

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

## 6.5 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.

## 6.6 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. The "Multiple expected" feature can be used to extend the comparison to a list of expected strings.

**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











## 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.

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

GSM_Configure(GSM_CONF_LIST_SEPARATOR, (void *) '!');
if(GSM_ReadResponse ("/dev/ttyS3", 100, GSM_INTERCHAR_DELAY, NULL,
                    "RING!+CREG: ")==GSM_EXEC_OK) printf ("OK\n");
```

listens to the serial port ten seconds and returns as soon as either "RING" or "+CREG: " unsolicited messages are recognized.



## 6.8 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 <i>len</i> 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 <i>written_bytes</i> 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)
```





```
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.

## 6.10 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

|  |   |
|--|---|
| <p>&lt;dev&gt;</p> <p>&lt;response&gt;</p> | <p>It's the serial device where the escape sequence (+++) will be sent.</p> <p>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.</p> |
|--|---|

### Return values

|                    |  |
|--------------------|--|
| <p>GSM_EXEC_OK</p> | <p>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</p> |
|--------------------|--|









**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.

## 6.14 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.   |





### 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.

## 6.16 GSM\_StopDataCall ()

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







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

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.

## 6.18 GSM\_TermGPRS()

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

### Prototype

```
GSM_ErrorCode_t GSM_TermGPRS (char * dev, char context)
```

### Parameters

|                              |   |
|------------------------------|---|
| <code>&lt;dev&gt;</code>     | It's the serial device used for this API.   |
| <code>&lt;context&gt;</code> | 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');
```





<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.  |
| GSM_WRONG_DEVICE       | The selected device has not been opened.   |
| GSM_TIMEOUT_ERROR      | No response received from the modem.<br><b>NOTE:</b> if <i>ppp_timeout</i> 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 <i>ppp_timeout</i> expired. It may depend on the field level.                  |
| GSM_UNKNOWN_ERROR      | An unpredictable error occurred.   |

### Examples

```
GSM_ErrorCode_t result;
result = GSM_InitPPPOverGPRS("/dev/ttyS3", "ibox.tim.it", 15);
...
result = GSM_InitSerialModem("/dev/ttyS3", GSM_115200, GSM_DEFAULT_SERIAL);
...
result = GSM_TermPPPOverGPRS("/dev/ttyS3");
```

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 */etc/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.

Please note that the serial device has to be initialized again even before the PPP termination.

```
GSM_ErrorCode_t result;
result = GSM_InitPPPOverGPRS("/dev/cmux1", "ibox.tim.it", 30);
...
result = GSM_InitSerialModem("/dev/cmux1", GSM_115200, GSM_DEFAULT_SERIAL); (*)
```



...  
`result = GSM_TermPPPOverGPRS("/dev/ttyS3");`

initializes a PPP connection using the virtual channel CMUX1.

**(\*)** Also in this case the initialization of the serial modem must be performed again, but only if the CMUX version is lower than 1.2.0, as explained [chapter 7](#)..

## 6.20 GSM\_TermPPPOverGPRS()

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

**NOTE:** since `GSM_InitPPPOverGPRS()` closes the serial connection, before to call this function you have to open it again by means of the `GSM_InitSerialModem()` function.

**NOTE:** if you're using CMUX please check chapter 7.

### 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. <u>Note:</u> the escape sequence is sent anyway in order to hang up any pending data connection. |
| 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.

## 6.21 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.

## 6.22 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).







## 7 Using GSM library with CMUX

The modem's physical serial channel can be multiplexed into three virtual serial devices. This is done by the CMUX application [4] and can be useful if you need to send AT commands during a data connection (for example PPP).

From Linux console, you can launch the CMUX service by issuing for example

```
# cmuxt -p /dev/[modem tty] -b 115200 -d
```

After some seconds channels */dev/cmux1*, */dev/cmux2* and */dev/cmux3* become available.

### 7.1 CMUX and `GSM_InitPPPOverGPRS()` API

Normally, within the `GSM_PPPOverGPRS()` API, the connection to the current device is terminated, in order to make it available to PPPD daemon.

Nevertheless, when recent CMUX versions (1.2.0 or higher) are used the user don't need to reopen the channel.<sup>5</sup>

From Linux console, you can check the CMUX version by issuing

```
# cmuxt -v
```

---

<sup>5</sup> The reason for this modification is that when the serial port is closed, recent CMUX versions perform a hang up of any pending data connection, and the `GSM_TermSerialModem()` removal was needed in order to avoid `GSM_InitPPPOverGPRS()` systematic fail.



