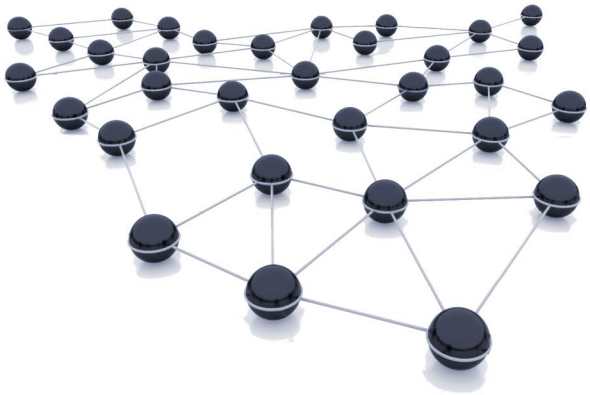
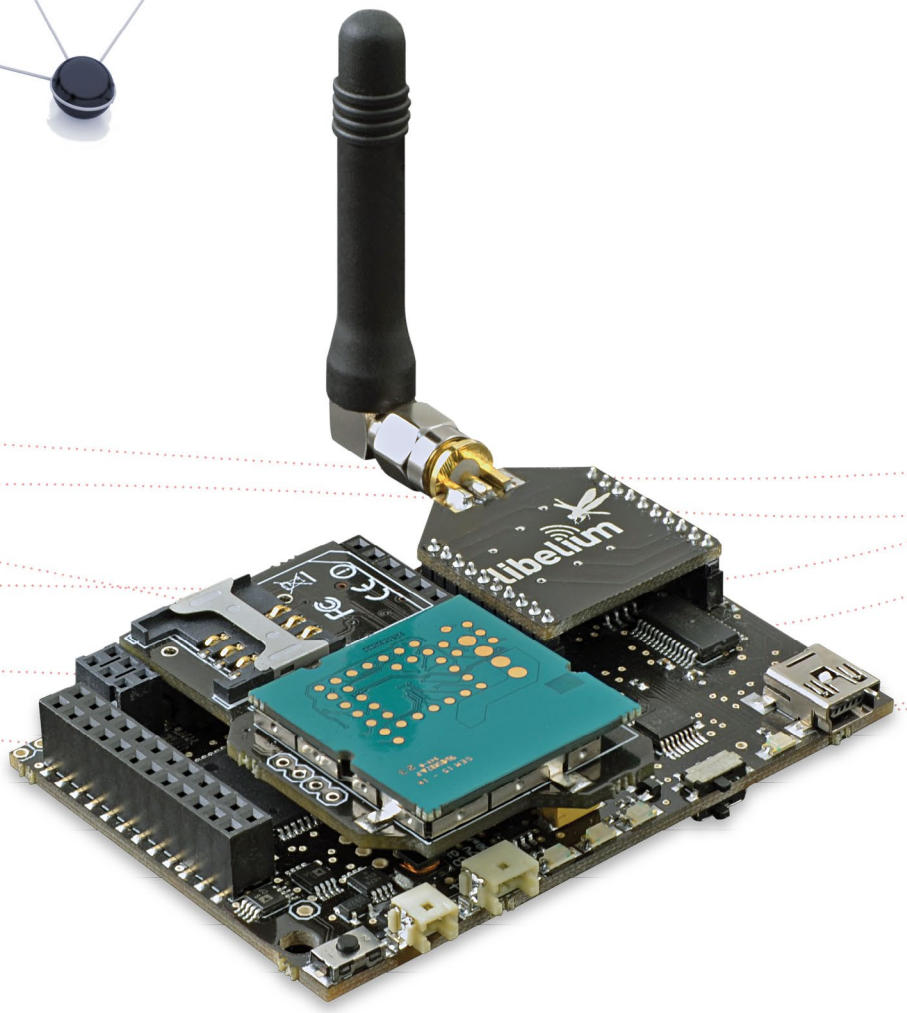
Waspmote GSM/GPRS

GPRS PRO Networking Guide





libelium

wasp mote

Document version: v4.9 - 06/2015

INDEX

1. General Considerations

1.1. Hardware

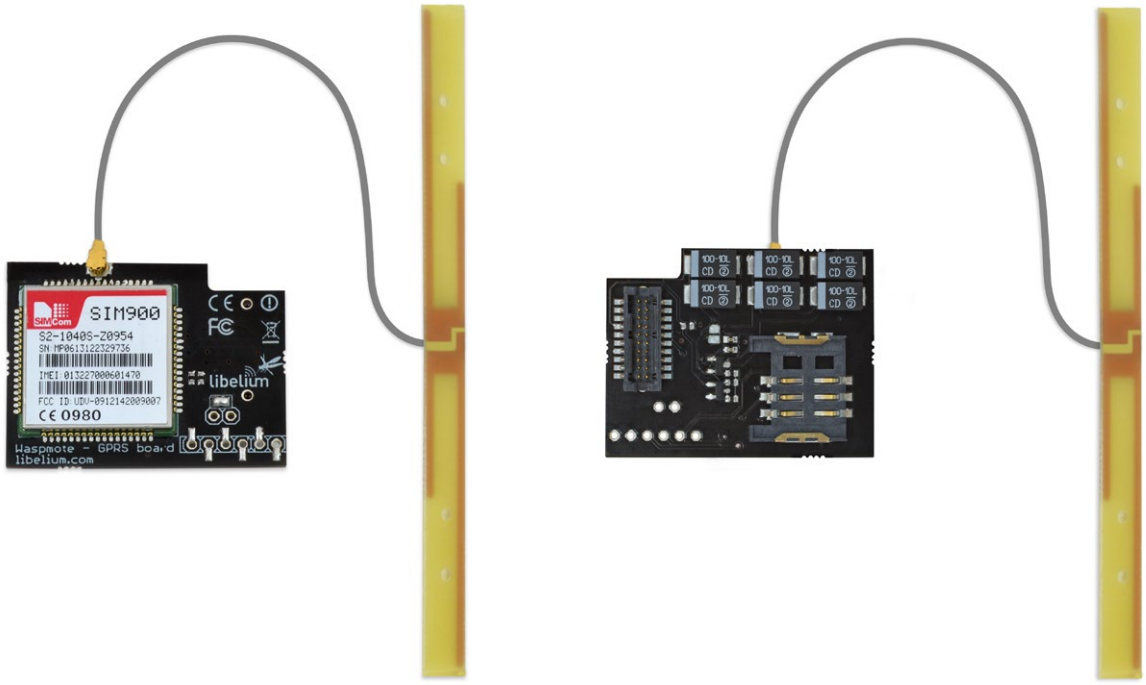


Figure: GSM/GPRS module

Model: SIM900 (SIMCom)
Quadband: 850MHz/900MHz/1800MHz/1900MHz
TX Power: 2W (Class 4) 850MHz/900MHz, 1W (Class 1) 1800MHz/1900MHz
Sensitivity: -109dBm
Antenna connector: UFL
External Antenna: 0dBi
Consumption in power down mode: 30µA
Actions:

- Making/Receiving calls.
- Making 'x' tone missed calls.
- Sending/Receiving SMS.
- Single connection and multiple connections TCP/IP and UDP/IP clients.
- TCP/IP server.
- HTTP service.
- FTP service (downloading and uploading files).
- OTA feature can be performed now by Waspmote's GSM/GPRS module. Refer to the Over the Air Programming Guide for more information:
  http://www.libelium.com/development/waspmote/documentation/over-the-air-programming-guide-otap/

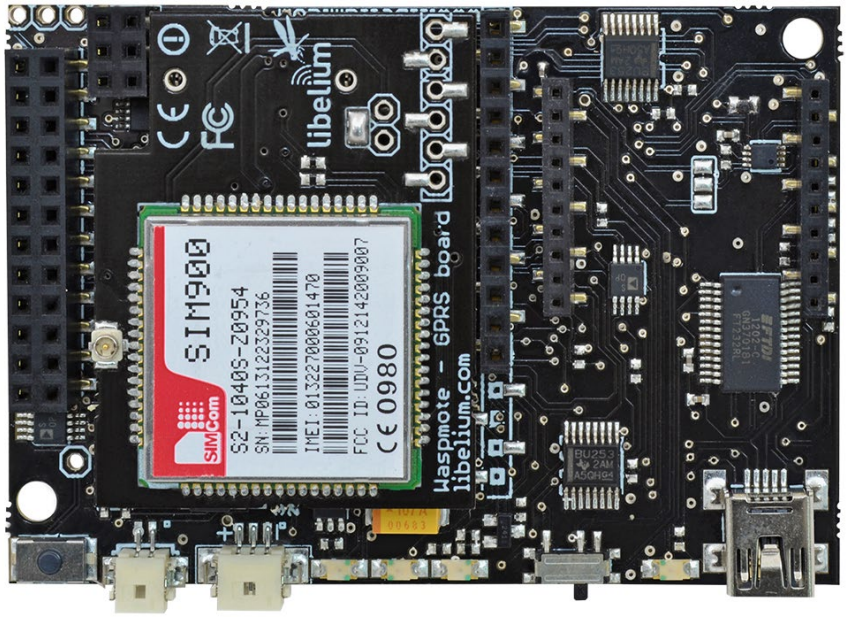


Figure: GSM/GPRS module in Waspmote

> ***Note:*** *There are 2 different GPRS modules, the GPRS module and the GPRS+GPS module. The GPRS module offers GPRS connectivity. The GPRS+GPS module offers exactly the same GPRS features, and adds GPS features too. Both modules are controlled in the same way for the GPRS features: the software library is the same.*
>
> *This is the guide for the GPRS Pro module. If you are interested in adding GPS features to your GPRS module, you may consider the GPRS+GPS module.*

1.2. Waspmote Libraries

1.2.1. Waspmote GPRS_Pro Files

WaspGPRS_Pro_core.h, WaspGPRS_Pro_core.cpp, WaspGPRS_Pro.h , WaspGPRS_Pro.cpp

It is mandatory to include the WaspGPRS_Pro library when using this module. The following line must be introduced at the beginning of the code:

```
#include <WaspGPRS_Pro.h>
```

1.2.2. Constructor

In order to start using Waspmote GPRS_Pro library, an object from class 'WaspGPRS_Pro' must be created. This object, called 'GPRS_Pro', is created inside Waspmote GPRS_Pro library and it is public to all libraries. It is used throughout the guide to show how the Waspmote GPRS_Pro library works.

When creating this constructor, some variables are defined with a value by default, that can be modified later. These variables are:

- `baudRate` : specifies the baudrate used to communicate with the module (57600 by default).
- `_socket` : specifies the `socket` used to communicate with the module (UART1 by default).
- `_pwrMode` : specifies the power mode (ON by default).

1.2.3. Working Modes

Some constants have been defined to set the different working modes for GPRS_Pro module.

- 0: `GPRS_PRO_ON`. Module is powered on. Starts in full functionality mode.
- 1: `GPRS_PRO_FULL`. Module is set in full functionality (power consumption around 10-40mA).
- 2: `GPRS_PRO_RF_OFF`. Module deactivates RF circuits (power consumption around 10-40mA).
- 3: `GPRS_PRO_MIN`. Module is set on minimum functionality (power consumption around 10-40mA).
- 4: `GPRS_PRO_SLEEP`. Module enters in sleep mode when there is no data on serial port. From time to time, it communicates with carrier to maintain the connection (power consumption 1-2mA).
- 5: `GPRS_PRO_POWER_OFF`. Module is powered off. The power transistors are switched off.

Minimum functionality mode and RF disabled functionality mode cannot be switched to each other. When the module communicates with carrier, current peaks between 1-1.4A are produced.

1.2.4. Library fuses

The library for this module is divided in 4 sections and a section with functions and constants that always are active. Each section is activated/deactivated by a fuse. This fuses are located in WaspGPRS_Pro.h. By default this fuses have a '1' allowing to use the functions. The WaspGPRS_Pro.h have this fuses:

- `GSM_FUSE:` Call and SMS related functions and constants
- `HTTP_FUSE:` HTTP related functions and constants
- `FTP_FUSE:` FTP related functions and constants
- `IP_FUSE:` TCP and UDP related functions and constants
- `OTA_FUSE: Over The Air Programming related functions and constants`

For deactivate a section change the '1' in the related fuse by '0'.

Note**: If you use a function with the related fused with a value of '0', the compiler will give you an error.* ***Please, be carefully using the fuses when compiling.

1.2.5. Debug modes

The library has two debug modes implemented that allows to the experimented user show the data transmitted and received between the Waspmote and the GPRS_Pro module. The value of the `GRPS_debug_mode` constant selects the debug mode:

- '0' Disables all debug messages
- '1' Shows only the commands sent to the GPRS_Pro module and some extra messages
- '2' Shows commands sent to the GPRS_Pro module, the data answered by the module and some extra messages

The constant `GPRS_debug_mode` is located in WaspGPRS_Pro.h file.

1.2.6. Special errors

Some functions can give an extra error code (CME or CMS error codes).

This code is stored on `GPRS_Pro.CME_CMS_code`. To know the description of the code see the apendix.

2. Initialization

2.1. Initializing the GPRS_Pro module

The GPRS_Pro module is connected to a multiplexer, since it is connected to the same microcontroller socket. To start using the GPRS_Pro module, this multiplexer must be switched on and choose the correct combination for the GPRS_Pro to be selected.

To open the socket and set the multiplexer to the correct combination, a function has been developed. This function powers on `(GPRS_Pro.setMode(GPRS_PRO_ON))` the module too.

Example of use:

```
{
  GPRS_Pro.ON(); // Opens socket and sets multiplexer
}
```

2.2. Setting the GPRS_Pro Power Mode

Sets the current internal Power Mode on the GPRS_Pro. GPRS_Pro module has five different power modes: `GPRS_PRO_ON, GPRS_PRO_FULL, GPRS_PRO_RF_OFF, GPRS_PRO_MIN, GPRS_PRO_SLEEP, GPRS_PRO_POWER_OFF.` The function will set up the `pwrMode` variable to one of the five values, but also sends the serial command to the GPRS_Pro module. It returns '1' on success, '0' and '-2' if error and '-3' when the module starts successfully with low battery.

Example of use:

```
{
  uint8_t powerMode = 0;
  GPRS_Pro.setMode(GPRS_PRO_ON); // Powers GPRS_Pro on
  powerMode = GPRS_Pro.getMode(); // Get GPRS_Pro power mode
}
```

Related variables

`GPRS_Pro.pwrMode` → stores GPRS_Pro power mode

2.3. Closing the GPRS_Pro module

It closes the socket to which the GPRS_Pro module is connected. It means disconnecting the internal socket drivers inside the ATMEGA1281 processor.

Example of use:

```
{
  GPRS_Pro.close(); // Closes socket
}
```

2.4. Switching GPRS_Pro off

It closes the socket to which the GPRS_Pro module is connected to and turns it off.

Example of use:

```
{
  GPRS_Pro.OFF(); // Closes socket and turns it off
}
```

2.5. Setting time and date

This function updates the time and the date of the GPRS_Pro module from the RTC.

It returns '1' on success, '0' or '-2' if error

Example of use:

```
{
  GPRS_Pro.setTime(); // Updates time and date
}
```

2.6. Checking the GSM connection

It checks if the module is connected to the network for a time desired by the input parameter.

If the GPRS_Pro module does not connect within these attempts, function exits with '0'.

It returns '1' when connected and '0' if not.

Example of use:

```
{
  GPRS_Pro.check(180); // Waits 180 seconds for connection
}
```

According to Libelium experience, 180 seconds is the recommended timeout for connecting the GPRS network. The timeout may vary depending on the network range and quality of service. If the user detects this timeout does not allow his GPRS to connect, he can experiment with higher values. 180 seconds would be the maximum advised timeout: if the GSM/GPRS module cannot connect in 2 minutes, it is better to stop trying and check again in the next loop.

2.7. Setting operator parameters

When the GPRS module uses some services like IP connections (TCP/UDP), HTTP services or FTP transfers, it's mandatory to configure operator parameters like APN, login and password.

There are two ways to configure these settings. The first one is to use the definitions into the file WaspGPRS_Pro_core.h.

```
#define   AT_GPRS_APN          "apn"
#define   AT_GPRS_LOGIN        "login"
#define   AT_GPRS_PASSW        "password"
```

The second one is to use the function `set_APN()`.
Example of use:

```
{
  // If only APN is necessary
  GPRS_Pro.set_APN("provider_apn");
  // If APN, login and password are necessary
  GPRS_Pro.set_APN("provider_apn", "login", "password");
}
```

3. SIM related functions

Some functions have been developed to configure some settings related with GSM, specifically with managing calls and SMS.

3.1. Setting the PIN

It sets the PIN to the SIM and it returns '1' on success, '0' if error and '-2' if CME error code available.

Example of use

```
{
  GPRS_Pro.setPIN("1234");   // Sets PIN = 1234 to the SIM
}
```

3.2. Changing PIN number

It changes PIN number of SIM card.

It returns '1' on success, '0' if error and '-2' if CME error code available.

Example of use

```
{
 GPRS_Pro.changePIN("1234", "4321"); // Changes "1234" to "4321" PIN number
}
```

3.3. Getting IMSI

It gets the IMSI from the SIM card and it stores the IMSI into the `buffer_GPRS` variable.

It returns '1' if connected, '0' if not.

Example of use

```
{
 GPRS_Pro.getIMSI();
}
```

Getting IMSI example:

http://www.libelium.com/development/waspmote/examples/gprs-22-getting-imsi-from-sim-and-imei

3.4. Getting IMEI

It gets the IMEI from the SIM card and it stores the IMEI into the `buffer_GPRS` variable.

It returns '1' if connected, '0' if not.

Example of use

```
{
    GPRS_Pro.getIMEI();
}
```

Getting IMEI example:

http://www.libelium.com/development/waspmote/examples/gprs-22-getting-imsi-from-sim-and-imei

4. Call functions

4.1. Setting Information returned when receiving a call

This function configures the information returned by the module when a call is received. It is useful to generate interruptions or to store data from the incoming call .

It returns '1' on success, '0' if error and '-1' if no memory and '-2' if error with CME error code available.

Example of use

```
{
  GPRS_Pro.setInfoIncomingCall(); // Sets information returned by the module when incoming call
}
```

4.2. Making Calls

It makes a call to the given telephone number.

It returns '1' on success, '0' if error.

***Note**: the number can be using the country code or only the phone number.*

Example of use

```
{
  GPRS_Pro.makeCall("**********"); // Makes a call to the desired number
}
```

Making call example:

http://www.libelium.com/development/waspmote/examples/gprs-07-making-call

4.3. Making Missed Calls

It makes a specified duration missed call to the given telephone number.

It returns '1' on success, '0' if error.

Note 1: the calling time includes the time to send the request to the carrier, so the receiving call time is a bit shorter than the input parameter.

Note 2: the number can be using the country code or only the phone number.

Example of use

```
{
  GPRS_Pro.makeLostCall("**********",10); // Makes a lost call of 10 seconds to the desired number
}
```

Making lost call example:

http://www.libelium.com/development/waspmote/examples/gprs-06-making-lost-call

4.4. Hanging up calls

It hangs up all the active calls. It returns '1' on success and '0' if error.

Example of use

```
{
  GPRS_Pro.hangUp(); // Hangs all the active calls up
}
```

5. SMS related functions

5.1. Setting Information returned when receiving an SMS

This function configures the information returned by the module when an SMS is received. It is useful to generate interruptions or to store data from the incoming SMS.

It returns '1' on success and '0' if error.

Example of use

```
{
   GPRS_Pro.setInfoIncomingSMS(); // Sets information returned by the module when incoming SMS
}
```

5.2. Setting Mode for SMS

It sets the text mode for the SMS and it returns '1' on success and '0' if error.

It returns '1' on success and '0' if error.

Example of use

```
{
  GPRS_Pro.setTextModeSMS(); // Sets text mode for SMS
}
```

5.3. Sending SMS

It sends a SMS to the specified number.

It returns '1' on success, '0' if error, '-2' if error sending the SMS and '-3' if error sending the SMS with CMS error code available.

Note: the maximum length is 90 Bytes (90 characters).

Example of use

```
{
  GPRS_Pro.sendSMS("Hello World!","6********"); // Sends this text in an SMS to the desired number
}
```

Sending SMS example:

http://www.libelium.com/development/waspmote/examples/gprs-08-sending-sms

5.4. Deleting SMS

It deletes a SMS in memory selected by `sms_index.`

It returns '1' on success, '0' if error and '-2' if error with CMS error code available.

Example of use:

```
{
   GPRS_Pro.deleteSMS("2"); // Deletes the SMS at index 2
}
```

5.5. Getting the number of total SMSs stored in the SIM

It gets the number of total SMSs stored in the SIM

It returns the number of SMSs, or '-2' if error

Example of use:

```
{
   total_SMS = GPRS_Pro.getTotalSMS(); //  Gets the number of SMS
}
```

6. HTTP and FTP protocols

6.1. Configuring HTTP/FTP profile

It configures GPRS connection with login, password and some other parameters for use with HTTP and FTP functions.

The configuration parameters from 'WaspGPRS_Pro.h' are `AT_GPRS_APN`, `AT_GPRS_LOGIN` and `AT_GPRS_PASSW`.

It returns:

- '1' on success
- '0' if error
- '-2' if error setting the type of internet connection,
- '-3' if error setting the APN
- '-4' if error setting the user name
- '-5' if error setting the password
- '-6' if error saving the configuration

Example of use:

```
{
  GPRS_Pro.configureGPRS_HTTP_FTP(1); // Configures profile 1 for use it
}
```

6.2. Creating your own FTP server

First, you should get a server. This server will receive your frames and store them. There is no need to purchase a physical server since there a companies that offer remote servers.

Note: *The server used by Libelium to realize the upload and download tests is a Pure-FTPD server (**www.pureftpd.org**) and it is hosted into a OVH server (**www.ovh.com**). The Pure-FTPD server has the settings by default:*

- *TLS encryption support: Optional*
- *TLS cipher suite: High-Medium + TLSv1*
- *Allow anonymous logins: No*
- *Allow anonymous uploads: No*
- *Maximum load for anonymous downloads: 4*
- *Maximum idle time (minutes): 15*
- *Maximum connections: 50*
- *Maximum connections per IP address: 8*
- *Allow logins with root password: Yes*
- *Broken clients compatibility: No*

*From Libelium, we recommend the use of this server hosting provider to obtain good results to upload and download files, but **we can't guarantee the perfect performance of the FTP server**.*

*Also, Libelium has tested with Guebs hosting (**www.guebs.com**) with good results.*

Follow the next steps to create your own FTP server using the terminal:

1 - Install the pure-ftpd server:

```
sudo apt-get install pure-ftpd
```

2 - Stop the server:

```
sudo /etc/init.d/pure-ftpd stop
```

3 - Before creating the user, it is necessary create a directory to stored the received data:
sudo mkdir /home/ftp and inclide a false shell. To check if the shell exist:

```
sudo more /etc/shells
```

4 - If it isn't the line `/bin/false`, edit the file, for example with vim, and include it:

```
sudo vim /etc/shells
```

5 - After creating a new folder to store data, you have to create a group and a user with false shell, because this type of user don't need a valid shell (more secure), therefore select `/bin/false` shell for user and `/dev/null` as directory:

```
sudo groupadd ftpgroup
sudo useradd -g ftpgroup -d /dev/null -s /bin/false user1
```

6 - Modify folder permissions:

```
sudo chown -R user1 /home/ftp
sudo chmod -R 755  /home/ftp
```

7 - Add the new user to the pure-ftpd database:

```
sudo pure-pw useradd username -u ftpuser -g ftpgroup -d /home/ftp
```

Set the user password when it request.

8 - Update the pure-ftpd database:

```
sudo pure-pw mkdb
```

9 - When the users are include in the ftp, start the server:

```
sudo /etc/init.d/pure-ftpd start
```

Remember to open the ports in your router. After this, you will be able to receive FTP transmissions on your server. We advise to use FileZilla to visualize and manage your FTP server.

6.3. Uploading a file

It uploads a file to a FTP server.

It returns:

- '1' on success
- '0' if error opening connection with the GPRS provider
- '-1' if no GPRS connection
- '-2' if error opening the connection
- '-3' if error getting the IP address
- '-4' if error setting the FTP/HTTP ID (no answer from module)
- '-5' if error setting the FTP mode (no answer from module)
- '-6' if error setting the FTP type (no answer from module)
- '-7' if error setting the FTP server(no answer from module)
- '-8' if error setting the FTP port (no answer from module)
- '-9' if error setting the user name
- '-10' if error setting the password
- '-11' if error starting the SD
- '-12' if error taking the file size in the SD
- '-13' if error setting the file name in the FTP server (no answer from module)
- '-14' if error setting the path of the file in the FTP server (no answer from module)
- '-15' if error opening the FTP session (no answer from module)
- '-16' if error when request to send data
- '-17' error sending data to the FTP
- '-18' if error waiting for send more data
- '-19' if error when setting Upload File (with CME error code available)
- '-20' if error closing the FTP session (no answer from module)
- '-21' setting the file name in the FTP to get the file size
- '-22' setting the path in the FTP to get the file size
- '-23' if error getting the file size (no answer from module)
- '-24' if error opening SD file
- '-25' if error when setting pointer to the beginning of the file
- '-40' if error from FTP when the module sends data with error code available
- '-41' if error setting the FTP/HTTP ID with CME error code available
- '-42' if error setting the FTP mode (with CME error code available)
- '-43' if error setting the FTP type (with CME error code available)
- '-44' if error setting the FTP server (with CME error code available)
- '-45' if error setting the FTP port (with CME error code available)
- '-46' if error setting the user name (with CME error code available)
- '-47' if error setting the password (with CME error code available)
- '-48' if error setting the file name in the FTP server (with CME error code available)

- '-49' if error setting the path of the file in the FTP server (with CME error code available)
- '-50' if error opening the FTP session (with CME error code available)
- '-51' if error closing the FTP session (with CME error code available)
- '-53' setting the file name in the FTP to get the file size (with CME error code available)
- '-54' setting the path in the FTP to get the file size (with CME error code available)
- '-55' if error getting the file size (with CME error code available)
- '-56' if FTP is busy error getting the size of the file from the FTP
- '-57' if FTP is busy sending data
- '-58' if timeout

Example of use

```
{
  // Uploads file.txt file with HTTP/FTP profile 1
  GPRS_Pro.uploadFile("/SD_folder/file.txt"
                          , "/FTP_folder/test_file.txt"
                          , "username"
                          , "password"
                          , "FTP_server"
                          , "FTP_port"
                          , 1);
}
```

Uploading files example:

http://www.libelium.com/development/waspmote/examples/gprs-20-uploading-files-to-a-ftp-server

6.4. Downloading a file

It downloads a file from an FTP server.

It returns:

'1' on success
'0' if error opening connection with the GPRS provider
'-1' error downloading the file
'-2' if error getting the IP address
'-3' if error setting the FTP/HTTP ID
'-4' if error setting the FTP mode
'-5' if error setting the FTP type
'-6' if error setting the FTP server
'-7' if error setting the FTP port
'-8' if error setting the user name
'-9' if error setting the password
'-12' if error setting the file name in the FTP server,
'-13' if error setting the path of the file in the FTP server
'-14' if error opening the FTP session
'-15' if error starting the SD
'-16' if error creating the file
'-17' error requesting data to the FTP
'-18' if error saving data into the SD
'-19' if error requesting more data to the FTP
'-21' setting the file name in the FTP to get the file size
'-22' setting the path in the FTP to get the file size
'-23' if error getting the file size
'-41' if error setting the FTP/HTTP ID with CME error code available
'-42' if error setting the FTP mode with CME error code available,
'-43' if error setting the FTP type with CME error code available
'-44' if error setting the FTP server with CME error code available
'-45' if error setting the FTP port with CME error code available
'-46' if error setting the user name with CME error code available,
'-47' if error setting the password with CME error code available
'-48' if error setting the file name in the FTP server with CME error code available
'-49' if error setting the path of the file in the FTP server with CME error code available
'-50' if error opening the FTP session with CME error code available
'-51' if error requesting data to the FTP with CME error code available
'-52' if error requesting more data to the FTP with CME error code available
'-53' setting the file name in the FTP to get the file size with CME error code available
'-54' setting the path in the FTP to get the file size with CME error code available
'-55' if error getting the file size with CME error code available
'-56' if FTP is busy error getting the size of the file from the FTP
'-57' if FTP is busy sending data
'-58' if timeout

Example of use:

```
{
  // Uploads test_file.txt file with HTTP/FTP profile 2
  GPRS_Pro.downloadFile("/FTP_folder/test_file.txt"
                        , "/SD_folder/file.txt"
                        , "username"
                        , "password"
                        , "FTP_server"
                        , "FTP_port"
                        , 2);
}
```

Downloading files example:

http://www.libelium.com/development/waspmote/examples/gprs-21-downloading-files-from-a-ftp-server

6.5. HTTP protocol

HTTP is a great protocol because it is a standard, simple and light way to send information to web servers.

Libelium has created a little web service in order to allow GPRS, WiFi or 3G modules users to test the HTTP mode. This web service is a little code, written in PHP, which is continuously listening to the HTTP port (port number 80) of our test server "pruebas.libelium.com". This is a kind of RESTful service. GPRS, WiFi or 3G modules can send HTTP instances to our web service.

HTTP instances should have the following structures so that our web service can understand.

6.5.1. GET method

In GET method the data are sent to the server append to the main URL with the '?' character. The base sentence to perform GET method is shown below:

```
pruebas.libelium.com/getpost_frame_parser.php?<variable1=value1>&<variable2=value2>&<...>&v
iew=html
```

Where:

- `getpost_frame_parser.php?` : It is the main URL, where the web service is running.
- `<variable1=value1>` : It is a couple with the variable name and value which we want the web service to parse.
- `view=html` : It is an optional argument. It shows a "pretty" response (HTML formatted)

All arguments must be separated by "&". The variable name and value must be separated by "=".

Some examples:

pruebas.libelium.com/getpost_frame_parser.php?var1=3.1415

pruebas.libelium.com/getpost_frame_parser.php?var1=3.1415&view=html

pruebas.libelium.com/getpost_frame_parser.php?var1=3.1415&var2=123456&var3=hello&view=html

6.5.2. POST method

Unlike GET method, with POST method the data are sent to the server into an extra data field. The URL only includes the site name and the PHP direction:

```
pruebas.libelium.com/getpost_frame_parser.php
```

The body is very similar to the used in GET method:

```
<variable1=value1>&<variable2=value2>&<...>&view=html
```

Where:

- `<variable1=value1>` : It is a couple with the variable name and value which we want the web service to parse. All arguments must be separated by "&". The variable name and value must be separated by "=".

- Some examples of data field:

```
var1=3.1415
var1=3.1415&var2=123456
var1=3.1415&var2=123456&var3=hello
```

6.5.3. Server response

If the web service receives one instance with the appropriate format, some actions will happen:

- the web service grabs the string and parses it. So the PHP code creates couples with the variables name and value.
- the web service responses to the sender device (to the sender IP) with an HTML-formatted reply.

Figure: Operating with the web service from a PC browser

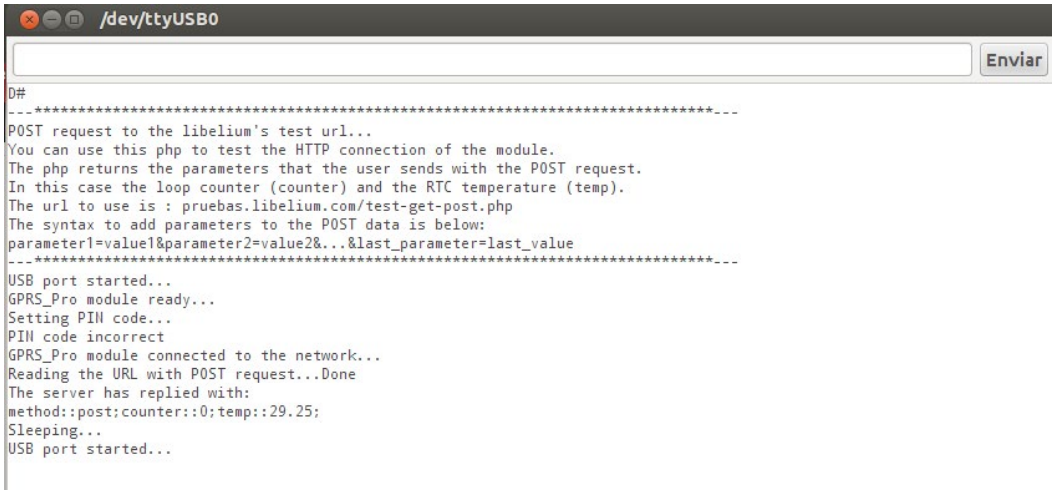


Figure: Operating with the web service from a Waspmote GPRS

Remember this PHP code is really simple and is offered with the only purpose of testing, **without any warranty**. The source code is available here:

downloads.libelium.com/waspmote-html-get-post-php-parser-tester.zip

The user may find it interesting to copy this code and make it run on his own server (physical or virtual). If the user wants to go further, he can complete the code. For example, once the couples are parsed, the user can modify the PHP to save data into a txt file, or insert couples into a database, or include a timestamp...

6.6. HTTP queries with GET and POST methods

The function `readURL()` can be used to send data, using the GET method, to a server and receive a response. The syntax of the function depends of the data to send.

If the data to be sent is a plain string, the syntax is shown below:

```
GPRS_Pro.readURL(url, HTTP_FTP_profile);
```

Where:

- `url`: the URL to get the information from
- `HTTP_FTP_profile`: profile used with the function `configureGPRS_HTTP_FTP()`

Example of GET query:

```
{
   // GET method without frame using HTTP/FTP profile 1
   GPRS_Pro.readURL("http://pruebas.libelium.com/getpost_frame_parser.
php?var1=2&var2=2.1516", 1);
}
```

The function `readURL()` can be used to send data, using the **POST** method, to a server and receive a response. The syntax of the function depends of the data to send.

If the data to be sent is a plain string, the syntax is shown below:

```
GPRS_Pro.readURL(url, POST_data, HTTP_FTP_profile);
```

Where:

- `url:` the URL to get the information from
- `POST_data`: data to send wit the POST request
- `HTTP_FTP_profile`: profile used with the function `configureGPRS_HTTP_FTP()`

Example of POST query:

```
{
   // POST method without frame using HTTP/FTP profile 1
   GPRS_Pro.readURL("http://pruebas.libelium.com/getpost_frame_parser.php"
                    , "var1=2&var2=2.1516"
                    , 1);
}
```

The funcions returns:

'0' if error

'1' on success

'2' if buffer_GPRS is full. The answer from the server is limited by the length of `buffer_GPRS`. To increase the length of the answer, increase the `BUFFER_SIZE` constant.

'-1' if no GPRS connection

'-2' if error opening the connection,
'-3' if error getting the IP address,
'-4' if error initializing the HTTP service
'-5' if error setting CID the HTTP service
'-6' if error setting the URL the HTTP service
'-7' if error starting HTTP session
'-8' if error getting data from URL
'-9' if error closing the HTTP service
'-10' if error initializing the HTTP service with CME error code available
'-11' if error setting CID the HTTP service with CME error code available
'-12' if error setting the URL the HTTP service with CME error code available
'-13' if error starting HTTP session with CME error code available
'-14' if error getting data from URL with CME error code available
'-15' if error closing the HTTP service with CME error code available
'-20' if error getting data from URL with CME error code available

Reading an URL using GET and POST methods examples:

http://www.libelium.com/development/waspmote/examples/gprs-19a-reading-url-with-get
http://www.libelium.com/development/waspmote/examples/gprs-19c-reading-url-with-post

6.7. Sending a frame to Meshlium

Since June 2014, it is possible to send HTTP queries from Waspmote to Meshlium. All data sent using the Waspmote Frame to Meshlium is stored in the Meshlium's database using the Frame Parser. Thus, it is possible to access to this data or synchronize it to external systems.

For this purpose, we need to use the Waspmote Frame which permits to create sensor formatted frames in an easy way. Then the frame is used as seen below.

The function `readURL()` can be used to send data, using both GET and POST methods to Meshlium and receive a response.

```
GPRS_Pro.readURL(url, frame.buffer, frame.length, HTTP_FTP_profile, HTTP_method);
```

Where:

- `url`: the URL to get the information from
- `frame.buffer`: buffer from frame class. The frame must composed before to use the readURL function
- `frame.length`: the length of `frame.buffer`
- `HTTP_FTP_profile`: profile used with the function `configureGPRS_HTTP_FTP()`
- `HTTP_method`: must be `GET` in order to use the GET method or `POST` in order to use the POST method

Example of **GET** query with **frame**:

```
{
   // GET method with frame using HTTP/FTP profile 1
   GPRS_Pro.readURL("http://pruebas.libelium.com/getpost_frame_parser.php?"
                    , frame.buffer
                    , frame.length
                    , 1
                    , GET);
}
```

Example of **POST** query with **frame**:

```
{
   // POST method with frame using HTTP/FTP profile 1
   GPRS_Pro.readURL("http://pruebas.libelium.com/getpost_frame_parser.php"
                    , frame.buffer
                    , frame.length
                    , 1
                    , POST);
}
```

It returns:

- '0' if error
- '1' on success
- '2' if `buffer_GPRS` is full. The answer from the server is limited by the length of `buffer_GPRS`. To increase the length of the answer, increase the `BUFFER_SIZE` constant.
- '-1' if no GPRS connection
- '-2' if error opening the connection,
- '-3' if error getting the IP address,
- '-4' if error initializing the HTTP service
- '-5' if error setting CID the HTTP service
- '-6' if error setting the URL the HTTP service
- '-7' if error starting HTTP session
- '-8' if error getting data from URL
- '-9' if error closing the HTTP service
- '-10' if error initializing the HTTP service with CME error code available
- '-11' if error setting CID the HTTP service with CME error code available
- '-12' if error setting the URL the HTTP service with CME error code available
- '-13' if error starting HTTP session with CME error code available
- '-14' if error getting data from URL with CME error code available
- '-15' if error closing the HTTP service with CME error code available
- '-16' if error configuring the HTTP content parameter
- '-17' if error configuring the HTTP content parameter with CME error code available
- '-18' if error sending the POST data
- '-19' if error sending the POST data with CME error code available
- '-20' if error getting data from URL with CME error code available

HTTP query using GET and POST methods with Waspmote Frame examples:

http://www.libelium.com/development/waspmote/examples/gprs-19b-reading-url-with-get-and-frame
http://www.libelium.com/development/waspmote/examples/gprs-19d-reading-url-with-post-and-frame

7. TCP and UDP connections

7.1. Configuring TCP/UDP profile

It configures GPRS connection with login, password and some other parameters for use with TCP/UDP connections with parameters from 'WaspGPRS_Pro.h' file. The configuration parameters from 'WaspGPRS_Pro.h are `AT_GPRS_APN`, `AT_GPRS_LOGIN` and `AT_GPRS_PASSW`.

There are two modes of connection for TCP/UDP application: Single connection and multi-connection. When in single connection mode, GPRS_Pro module can work at both transparent mode and non-transparent mode. When in multi-connection mode, GPRS_Pro module only can work at non-transparent mode.

It returns:

- '1' on success
- '0' if error
- '-2' if error dettaching the GPRS connection
- '-3' if error attaching the GPRS connection
- '-4' if error setting the application mode,
- '-5' if error setting the connection mode
- '-6' if error establishing the connection with the GPRS provider
- '-15' if error dettaching the GPRS connection with CME error code available
- '-16' if error attaching the GPRS connection with CME error code available

Example of use:

```
{
  GPRS_Pro.configureGPRS_TCP_UDP(MULTI_CONNECTION); // Configures the profile for multiple connections
  GPRS_Pro.configureGPRS_TCP_UDP(SINGLE_CONNECTION, NON_TRANSPARENT); // Configures the profile for
  single connection and non transparent mode
}
```

7.2. Setting DNS servers

It sets the directions of DNS servers from 'GPRS_Proconstants.h' file or by the user into the code.

It returns '1' on success, '0' if error and '-2' if CME error code available.

Example of use:

```
{
 GPRS_pro.setDNS(); // Sets DNS servers from GPRS_Proconstans.h
 GPRS_pro.setDNS("***.***.***.***"); // Sets DNS server
 GPRS_pro.setDNS("***.***.***.***", "***.***.***.***"); // Sets DNS servers
}
```

7.3. Set local port

It sets the internal port for TCP or UDP connection. This function will be effective only in single connection mode and when module is set as a Client.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.setLocalPort("UDP", 32000);    //Set local port for an UDP connection
}
```

7.4. Saving configuration profile

It saves the configuration profile into the internal NVRAM of the GPRS_Pro module.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.saveGPRS_TCP_UDPconfiguration();
}
```

7.5. Creating a TCP/UDP client or a TCP server

In single connection mode, GPRS_Pro can be configured as either TCP/UDP client or TCP server. When in multi-connection mode, GPRS_Pro can work as an absolute TCP/UDP client, which can establish 8 connections in total. It also can be configured as one TCP server, which allows 7 TCP/UDP clients to connect in; and the TCP server also can act as a client, establishing 7 connections to one remote server.

It returns '1' on success, '0' if error setting the connection, '-2' if error setting the connection with CME error code available and '-3' if time out waiting the connection.

Example of use:

```
{
  GPRS_Pro.createSocket( TCP_SERVER, "512"); // Creates a TCP server
 GPRS_Pro.createSocket( UDP_CLIENT, 3, "82.200.15.92", "12002"); //  Creates an UDP client
in connection number 3
  GPRS_Pro.createSocket( TCP_CLIENT, "82.200.15.92", "12002"); // Creates an TCP client
in single connection mode
}
```

7.6. Sending data

This function sends 'data' to the specified 'n_connection'.

It returns '1' on success, '0' if error waiting the response of the module, '-2' if error with CME error code available, '-3' if no feedback detected and '-4' if the send fails.

Example of use:

```
{
  GPRS_Pro.sendData("Test message in single connection!"); // Sending data in single connection mode
  GPRS_Pro.sendData("Test message in single connection!", 1); // Sending data in multiconnection by connection number 1
}
```

For send binary data or raw data the function needs an uint8_t pointer to the data and the length of the data.

It returns '1' on success, '0' if error waiting the response of the module, '-2' if error with CME error code available, '-3' if no feedback detected and '-4' if the send fails.

Example of use:

```
{
  GPRS_Pro.sendData(frame.buffer, frame.length); // Send a frame in single connection mode
  GPRS_Pro.sendData(frame.buffer, frame.length, 1); // Send a frame in multiconnection by connection number 1
}
```

7.7. Closing connections

It closes TCP/UDP connection.

It returns '1' on success, '0' if error.

Example of use

```
{
 GPRS_Pro.closeSocket(); // Closes connection in single connection mode
 GPRS_Pro.closeSocket(5); // Closes connection 5 in multi connection mode
 GPRS_Pro.closeSocket(8); // Closes TCP server
}
```

7.8. Enabling/disabling close connection quickly

It enables/disables to close the connection quickly.

It returns '1' on success, '0' if error.

Example of use:

```
{
 GPRS_Pro.QuickcloseSocket(DISABLE);
}
```

7.9. Getting IP direction from an URL

It gets the IP direction from a URL using DNS servers. Stores the IP direction in `buffer_GPRS`.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.getIPfromDNS("www.libelium.com"); // Gets IP direction from Libelium's web site
}
```

7.10. Enabling/disabling to adds a IP head

It enables/disables to add an IP head at the beginning of a package received.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.IPHeader(ENABLE);
}
```

7.11. Setting autosending timer

It sets a timer when module is sending data. Time is in seconds from 0 to 100.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.SetAutoSendingTimer(DISABLE);
  GPRS_Pro.SetAutoSendingTimer(ENABLE, 5);       // Sets timer at 5 seconds
}
```

7.12. Enabling/disabling to show remote IP address and port when received data

It enables or disables to show remote IP address and port when received data.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.ShowRemoteIP(DISABLE);
}
```

7.13. Enabling/disabling to show transfer protocol in IP head

It enables or disables to show transfer protocol in IP head when received data.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.ShowProtocolHeader(ENABLE);
}
```

7.14. Enabling/disabling to discard input AT data in TCP data send

It enables or disables to discard input AT data in TCP data send.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.DiscardInputATData(ENABLE);
}
```

7.15. Enabling/disabling to get data manually

It enables or disables to get data manually from a TCP/UDP connection.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.SetDataManually(ENABLE); // Enable to get data manually in single connection
  GPRS_Pro.SetDataManually(DISABLE, 6); // Disable to get data manually in connection 6 in
multi-connection mode
}
```

7.16. Getting data manually

It gets data manually from a TCP or UDP connection.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.GetDataManually(25); // Gets 25 characters from single connection
  GPRS_Pro.GetDataManually(20, 2); // Gets 20 character from connection 2 in multi-connection mode
}
```

8. General functions

8.1. Managing incoming data

It waits for incoming calls, SMSs or TCP/UDP data up to 20 seconds. When a call or SMS is received, this function stores in the `tlfIN` variable the telephone number which sent the SMS or made the call, in the `buffer_GPRS` variable the text of the SMS received and in `sms_index` the last SMS received index.

It executes the function `readCall()` if a call is received, `readSMS()` if an SMS is received and `readIPData`.

Returns '1' for call, '2' for SMS, '3' for IP data and '0' for error or not data.

Note 1: the phone number has a maximum length of 15 characters, including country code.

Note 2: the SMS text has a maximum size of 100 characters.

Note 3: the TCP/UDP data string has a maximum size of 150 characters.

Example of use:

```
{
  GPRS_Pro.manageIncomingData(); // Manages incoming data
}
```

Related variables:

- `GPRS_Pro.tlfIN` → it stores the telephone number of the incoming SMS
- `GPRS_Pro.sms` → it stores the text of the received SMS
- `GPRS_Pro.buffer_GPRS` → it stores the data of TCP/UDP connections or the phone number of the incoming call.

8.2. Sending any AT command

It sends any AT command to the GPRS_Pro module and it stores in the `buffer_GPRS` variable the answer returned by the GPRS_Pro module. This variable ends with the character '\0'.

It returns '1' on success and '0' if error.

Example of use :

```
{
  GPRS_Pro.sendCommand("+CCLK?"); // It sends the AT command 'CCLK'
}
```

8.3. Setting volume and mode for monitor speaker

These functions configure volume (0 to 9) and mode (0 to 9) of the monitor speaker.

They return '1' on success, '0' if error .

Example of use:

```
{
 GPRS_Pro.setMonitorVolume(7);
 GPRS_Pro.setMonitorMode(1);
}
```

8.4. Setting CLI presentation in incoming calls

This function enables or disables the presentation of the incoming call.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.setCLIPresentation(ENABLE);
}
```

8.5. Setting CLI presentation in outcoming calls

/*This function restricts or enables the presentation of the CLI to the called party when originating a call. Allowed modes `DEFAULT_CLIR`, `INVOKE_CLIR` or `SUPPRESS_CLIR`.

It returns '1' on success, '0' if error.

Example of use:

```
{
  GPRS_Pro.setCLIRestriction(SUPPRESS_CLIR); // Restricts the CLI for called party
}
```

8.6. Gets the phone activity status

This function gets the phone activity status and it returns '0' for error, '1'= Ready, '2'= Unknown, '3'= Ringing and '4'= Call in progress.

Example of use:

```
{
  status=GPRS_Pro.getPhoneActStatus();
}
```

8.7. Setting loudspeaker level

This function sets loudspeaker volume level between 0-100.

It returns '1' on success, '0' if error and '-2' if error with CME error code available..

Example of use:

```
{
  GPRS_Pro.setLoudspeakerLevel(55);
}
```

8.8. Setting call alert mode and level

This function configures mode, level and melody (0 to 19) of calls alert. Allowed modes are `NORMAL_MODE` or `SILENT_MODE`.

Allowed levels are `LEVEL_OFF`, `LEVEL_LOW`, `LEVEL_MEDIUM`, `LEVEL_HIGH` and `LEVEL_CRESCENDO`.

It returns:

- '1' on success
- '0' if error setting the sound mode
- '-2' if error setting the sound mode with CME error code available,
- '-3' if error setting the sound type
- '-4' if error setting the sound type with CME error code available,
- '-5' if error setting the ring level
- '-6' if error setting the ring level with CME error code available

Example of use:

```
{
  GPRS_Pro.setCallAlert(SILENT_MODE); // All sounds from module are prevented
  GPRS_Pro.setCallAlert(NORMAL_MODE,2, LEVEL_CRESCENDO);
}
```

8.9. Mute

This function enables or disables mute during a call.

It returns '1' on success, '0' if error and '-2' if error with CME error code available.

Example of use:

```
{
  GPRS_Pro.setMute(ENABLE);
}
```

8.10. Getting current operator

It gets the currently selected operator from network and stores it in `operator_name`.

It returns '1' on success and '0' if error and '-2' if CME error code available.

Example of use:

```
{
  GPRS_Pro.getcurrentOperator();
}
```

8.11. Getting available operators

This function gets the currently available operators from network and stores it in `operators_list`.

It returns '1' on success, '0' if error and '-2' if CME error code available.

Example of use:

```
{
  GPRS_Pro.getAvailableOperators();
}
```

Related variables

`operators_list` → struct that stores the format and the name of the operators.

8.12. Setting preferred operator

This function sets the preferred operator in the operators list into GPRS_Pro module.

It returns '1' on success, '0' if error and '-2' if CME error code available.

Example of use:

```
{
  GPRS_Pro.setPreferredOperator(1, 0, "operator"); // Sets "operator" in short format in the
second index
}
```

8.13. Getting cell information

This function gets the information from the cell where the module is connected and stores it in the `RSSI` and `cellID` variables.

It returns '1' on success, '0' if error and '-2' if CME error code available.

Example of use:

```
{
  GPRS_Pro.getCellInfo();
}
```

The RSSI value is measured in -dBm.

Getting RSSI and cell ID example:

http://www.libelium.com/development/waspmote/examples/gprs-10-getting-rssi-and-cellid

8.14. Getting module information

The function `whoamI()` get the model of the module and saves it in `buffer_GPRS`. The function `firmware_version()` get the firmware of the module and saves it in `buffer_GPRS`.

These functions return '1' on success, '0' if error.

Example of use:

```
{
        USB.print(F("WhoamI: "));
        GPRS_Pro.whoamI();
        USB.println(GPRS_Pro.buffer_GPRS);
        USB.print(F("Firmware version: "));
        GPRS_Pro.firmware_version();
        USB.println(GPRS_Pro.buffer_GPRS);
}
```

9. Working with Hibernate mode

When using the GPRS_Pro module with Hibernate mode, a little trick is necessary. Hibernate mode is woken up by the RTC's interrupt pin which is connected to the socket1's RX pin. So, it is necessary to add the following lines to get the GPRS_Pro module working after calling `PWR.ifHibernate`:

```
{
  // Checks if we come from a normal reset or an hibernate reset
  PWR.ifHibernate();
  delay(3000);
  RTC.ON();
  RTC.clearAlarmFlag();
  RTC.OFF();
  // now, GPRS_Pro can start
}
```

Lost call with hibernate example:

http://www.libelium.com/development/waspmote/examples/gprs-23-lost-call-with-hibernate

10. Code examples and extended information

In the Waspmote Development section you can find complete examples:

http://www.libelium.com/development/waspmote/examples

Example:

```
#include “WaspGPRS_Pro.h”

int answer;

void setup()
{
}

void loop()
{
  // setup for Serial port over USB:
  USB.ON();
  USB.println(F(“USB port started...”));

  // activates the GPRS_Pro module:
  answer = GPRS_Pro.ON();
  if ((answer == 1) || (answer == -3))
  {
    USB.println(F(“GPRS_Pro module ready...”));

    // sets pin code:
    USB.println(F(“Setting PIN code...”));
    // **** must be substituted by the SIM code
    if (GPRS_Pro.setPIN(“****”) == 1)
    {
      USB.println(F(“PIN code accepted”));
    }
    else
    {
      USB.println(F(“PIN code incorrect”));
    }

    // waits for connection to the network:
    answer = GPRS_Pro.check(180);
    if (answer == 1)
    {
      USB.println(F(“GPRS_Pro module connected to the network...”));

      // configures GPRS Connection for HTTP or FTP applications:
      answer = GPRS_Pro.configureGPRS_HTTP_FTP(1);
      if (answer == 1)
      {
        USB.println(F(“Uploading the file 1...”));

        // uploads file from SD card to the FTP server:
        answer = GPRS_Pro.uploadFile(“/dirSD1/dirSD2/fileSD1.txt”, “/ftp1/ftp2/fileFTP1.txt”, “username”, “password”,
“IP_server”, “port”, 1);
        if (answer == 1)
        {
          USB.println(F(“Upload done”));
        }
```

```
        else if(answer < -40)
        {
          USB.print(F("Upload failed. Error code: "));
          USB.println(answer, DEC);
          USB.print(F("CME error code: "));
          USB.println(GPRS_Pro.CME_CMS_code, DEC);
        }
        else
        {
          USB.print(F("Upload failed1. Error code: "));
          USB.println(answer, DEC);
        }

        USB.println(F("Uploading the file 2"));

        // uploads file from SD card to the FTP server:
        answer = GPRS_Pro.uploadFile("/fileSD2.txt", "/fileFTP2.txt", "username", "password", "IP_server", "port", 1);
        if (answer == 1)
        {
          USB.println(F("Upload done"));
        }
        else if (answer < -40)
        {
          USB.print(F("Upload failed. Error code: "));
          USB.println(answer, DEC);
          USB.print(F("CME error code: "));
          USB.println(GPRS_Pro.CME_CMS_code, DEC);
        }
        else
        {
          USB.print(F("Upload failed. Error code: "));
          USB.println(answer, DEC);
        }
      }
      else
      {
        USB.println(F("Configuration failed. Error code:"));
        USB.println(answer, DEC);
      }
    }
    else
    {
      USB.println(F("GPRS_Pro module cannot connect to the network"));
    }
  }
  else
  {
    USB.println(F("GPRS_Pro module not ready"));
  }

  GPRS_Pro.OFF(); // powers off the GPRS_Pro module

  USB.println(F("Sleeping..."));

 // activates the RTC:
  RTC.ON();
  // sleeps one hour
  PWR.deepSleep("00:01:00:00", RTC_OFFSET, RTC_ALM1_MODE1, ALL_OFF);

  // deactivates the RTC:
  RTC.OFF();
}
```

11. API Changelog

Keep track of the software changes on this link:

www.libelium.com/development/waspmote/documentation/changelog/#GPRS-GPS

12. Documentation changelog

From v4.8 to v4.9

- Few changes for the managing incoming data process

From v4.7 to v4.8

- Link to the new online API changelog

From v4.6 to v4.7

- API changelog updated to API v011
- Added new core files in section "Waspmote GPRS_Pro files"
- Added new section "Setting operator parameters" in chapter "Initialization"

From v4.5 to v4.6

- API changelog updated to API v010
- Added info about sending HTTP queries to Meshlium

From v4.4 to v4.5:

- API changelog updated to API v008

From v4.3 to v4.4:

- API changelog updated to API v007
- Added new section "HTTP protocol" in chapter "HTTP and FTP functions"
- Added new section "Reading an URL with GET method" in chapter "HTTP and FTP functions"
- Added new section "Reading an URL with GET method" in chapter "HTTP and FTP functions"

From v4.2 to v4.3:

- API changelog updated to API v006

From v4.1 to v4.2:

- API changelog updated
- Connection timeout increased
- All references to WaspGPRS_Procontants.h are deleted and substituted by WaspGPRS_Pro.h

From v4.0 to v4.1:

- Added new return codes in section "Donwloading a file"
- New section "Getting module information" added in chapter "General functions".
- Added return codes in section "Setting the GPRS_Pro power mode"
- Section "Checking the GSM connection" has been updated
- OTA feature added in section "Hardware"
- Added new section "Creating your own FTP server" in chapter "HTTP and FTP functions"
- API changelog updated

Appendix 1: CME error codes

CME error code	Description
0	phone failure
1	no connection to phone
2	phone-adaptor link reserved
3	operation not allowed
4	operation not supported
5	PH-SIM PIN required
6	PH-FSIM PIN required
7	PH-FSIM PUK required
10	SIM not inserted
11	SIM PIN required
12	SIM PUK required
13	SIM failure
14	SIM busy
15	SIM wrong
16	incorrect password
17	SIM PIN2 required
18	SIM PUK2 required
20	memory full
21	invalid index
22	not found
23	memory failure
24	text string too long
25	invalid characters in text string
26	dial string too long
27	invalid characters in dial string
30	no network service
31	network timeout
32	network not allowed - emergency call only
40	network personalisation PIN required
41	network personalisation PUK required
42	network subset personalisation PIN required
43	network subset personalisation PUK required
44	service provider personalisation PIN required
45	service provider personalisation PUK required
46	corporate personalisation PIN required
47	corporate personalisation PUK required
99	resource limitation

CME error code	Description
100	unknown
103	Illegal MS
106	Illegal ME
107	GPRS services not allowed
111	PLMN not allowed
112	Location area not allowed
113	Roaming not allowed in this location area
132	service option not supported
133	requested service option not subscribed
134	service option temporarily out of order
148	unspecified GPRS error
149	PDP authentication failure
150	invalid mobile class
151	Operation barred – Fixed dialing numbers only

Appendix 2: CMS error codes

CMS error code	Description
300	ME failure
301	SMS reserved
302	operation not allowed
303	operation not supported
304	invalid PDU mode
305	invalid text mode
310	SIM not inserted
311	SIM pin necessary
312	PH SIM pin necessary
313	SIM failure
314	SIM busy
315	SIM wrong
316	SIM PUK required
317	SIM PIN2 required
318	SIM PUK2 required
320	memory failure
321	invalid memory index
322	memory full
323	invalid input parameter
324	invalid input format
330	SMSC address unknown
331	no network
332	network timeout
340	no cnma ack
500	Unknown
512	SIM not ready
513	unread records on SIM
514	CB error unknown
515	PS busy
517	SIM BL not ready
528	Invalid (non-hex) chars inPDU
529	Incorrect PDU length
530	Invalid MTI
531	Invalid (non-hex) chars in address
532	Invalid address (no digits read)
533	Incorrect PDU length (UDL)
534	Incorrect SCA length
536	Invalid First Octet (should be 2 or 34)
537	Invalid Command Type
538	SRR bit not set
539	SRR bit set

CMS error code	Description
540	Invalid User Data Header IE
753	missing required cmd parameter
754	invalid SIM command
755	invalid File Id
756	missing required P1/2/3 parameter
757	invalid P1/2/3 parameter
758	missing required command data
759	invalid characters in command data
765	Invalid input value
766	Unsupported mode
767	Operation failed
768	Mux already running
769	Unable to get control
770	SIM network reject
765	Invalid input value
766	Unsupported mode
767	Operation failed
768	Mux already running
769	Unable to get control
770	SIM network reject
771	Call setup in progress
772	SIM powered down
773	SIM file not present