4.9.2. FTP upload ............................................................................................................................ 26
4.9.3. FTP download .......................................................................................................................... 27
4.9.4. FTP delete file ......................................................................................................................... 28
4.9.5. FTP working directory ........................................................................................................... 28
4.9.6. Closing an FTP session ............................................................................................................ 28
4.10. TCP/UDP connections .............................................................................................................. 29
  4.10.1. Socket identifiers .................................................................................................................. 29
  4.10.2. Socket information structure ............................................................................................... 29
  4.10.3. Socket status structure ......................................................................................................... 30
  4.10.4. Creating a TCP/UDP client socket ...................................................................................... 31
  4.10.5. Creating a TCP/UDP server socket ...................................................................................... 32
  4.10.6. Sending data .......................................................................................................................... 33
  4.10.7. Receiving data ...................................................................................................................... 34
  4.10.8. Closing a socket .................................................................................................................... 35
  4.10.9. SSL sockets .......................................................................................................................... 35
4.11. GPS ................................................................................................................................................ 38
  4.11.1. Standalone or Autonomous GPS (S-GPS) ................................................................................ 39
  4.11.2. Assisted GPS (A-GPS) ......................................................................................................... 39
  4.11.3. Get GPS position .................................................................................................................. 39
  4.11.4. Indoor tracking using 4G and A-GPS mode (geolocation) ......................................................... 41
4.12. e-mail management functions .................................................................................................. 44
  4.12.1. Reseting e-mail parameters ................................................................................................. 44
  4.12.2. Setting the SMTP server ...................................................................................................... 44
  4.12.3. Configuring SMTP parameters ............................................................................................ 44
  4.12.4. Setting the sender parameters: address, username and password ..................................... 45
  4.12.5. Saving e-mail parameters .................................................................................................... 45
  4.12.6. Sending an e-mail ................................................................................................................ 45

5. Certifications .................................................................................................................................. 47
6. Code examples and extended information ..................................................................................... 48
7. API changelog .................................................................................................................................. 49
8. Documentation changelog .............................................................................................................. 50
1. Introduction

This guide explains the features and use of the new 4G module. This module was specifically integrated for our new product lines Waspmote v15, Plug & Sense! v15 and Meshlium v4.0, released on October 2016. The 4G module is not compatible Waspmote v12, Plug & Sense! v12 or Meshlium 3.x.

If you are using previous versions of our products, please use the corresponding guides, available on our Development website.

You can get more information about the generation change on the document “New generation of Libelium product lines”. The 4G module has been integrated into the devices Waspmote OEM, Plug & Sense! and Meshlium.

The new 4G module enables the connectivity to high speed LTE, HSPA+, WCDMA cellular networks in order to make possible the creation of the next level of worldwide compatible projects inside the new “Internet of Things” era.

The new communication module is specially oriented to work with Internet servers, implementing internally several application layer protocols, which make easier to send the information to the cloud. We can make HTTP navigation, downloading and uploading content to a web server. We can also set secure connections using SSL certificates and setting TCP/IP private sockets. In the same way, the FTP protocol is also available which is really useful when your application requires handling files.

The module includes a GPS/GLONASS receiver, able to perform geolocation services using NMEA sentences, offering information such as latitude, longitude, altitude and speed; that makes it perfect to perform tracking applications.

The new 4G module offers the maximum performance of the 4G network as it uses 2 different antennas (normal + diversity) for reception (MIMO DL 2x2), choosing the best received signal at any time and getting a maximum download speed of 100 Mbps.

We chose the LE910 chipset family from Telit as it comprises the most complete 4G/LTE set of variants released up to date. It counts with many different models, each one specifically designed for one market but all of them with the same footprint:

- LE910-EUG (Europe / Brazil): CE, GCF, ANATEL
- LE910-NAG (US / Canada): FCC, IC, PTCRB, AT&T approved
- LE910-AU V2 (Australia): RCM, Telstra approved

Important note: The current stock of the LE910 4G radio that the manufacturer ‘Telit’ is distributing comprises the v2 version which does not have GPS. The models are:

- LE910-EU V2 for Europe or Brazil
- LE910-NA V2 for US or Canada

These v2 radios are similar to the v1 ones, but support more bands and do not have a GPS receiver.

Important:

- All documents and any examples they contain are provided as-is and are subject to change without notice. Except to the extent prohibited by law, Libelium makes no express or implied representation or warranty of any kind with regard to the documents, and specifically disclaims the implied warranties and conditions of merchantability and fitness for a particular purpose.
- The information on Libelium’s websites has been included in good faith for general informational purposes only. It should not be relied upon for any specific purpose and no representation or warranty is given as to its accuracy or completeness.
2. 3G (SIM5215) vs 4G (LE910)

The new 4G module (specific for the new lines Wasp mote v15, Plug & Sense! v15 and Meshlium 4.0) introduces some changes with respect to the 3G module (available for the old lines Wasp mote v12, Plug & Sense! v12, Meshlium 3.5 and the new Wasp mote v15):

• The new 4G counts with many different models, one specifically designed for each market:
  - LE910-EU (Europe / Brazil): CE, GCF, ANATEL
  - LE910-NAG (US / Canada): FCC, IC, PTCRB, AT&T approved
  - LE910-AU V2 (Australia): RCM, Telstra approved
• The GPS module also makes it possible to perform geo-location services using NMEA sentences, offering information such as latitude, longitude, altitude and speed, what makes it perfect for tracking applications.
• The new 4G module offers the maximum performance of the 4G network as it uses 2 different antennas (normal + diversity) for reception (MIMO DL 2x2), choosing the best received signal at any time and getting a maximum download speed of 100 Mbps.

Features comparison:

<table>
<thead>
<tr>
<th>Features</th>
<th>[v12] 3G module (SIM5215)</th>
<th>[v15] 4G module (LE910)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipset manufacturer</td>
<td>SIMCom</td>
<td>Telit</td>
</tr>
<tr>
<td>Cellular protocols</td>
<td>3G / GPRS / GSM</td>
<td>4G / 3G / GPRS / GSM</td>
</tr>
<tr>
<td>Certifications</td>
<td>CE, GCF, FCC, IC, PTCRB</td>
<td>CE, GCF, ANATEL, FCC, IC, PTCRB, AT&amp;T Compliant, KCC, RCM, NTT DoCoMo, KDDi</td>
</tr>
<tr>
<td>GPS</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Camera option</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SD card</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>USB connectivity</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Download max speed</td>
<td>384 kbps</td>
<td>100 Mbps</td>
</tr>
<tr>
<td>Upload max speed</td>
<td>384 kbps</td>
<td>50 Mbps</td>
</tr>
<tr>
<td>Antenna diversity</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Cellular carriers (mobile network operator)</td>
<td>Any</td>
<td>Any + Specially tested with AT&amp;T, SK Telecom, Telstra, NTT DoCoMo or KDDi</td>
</tr>
<tr>
<td>FTP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FTPS (Secure)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>HTTP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>HTTPS (Secure)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TCP/UDP sockets</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SSL sockets</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Mails</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

4G compatibility:

<table>
<thead>
<tr>
<th>Item</th>
<th>Compatible</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wasp mote 12</td>
<td>Yes</td>
<td>New Wasp mote API needed (v025 or newer)</td>
</tr>
<tr>
<td>Wasp mote 15</td>
<td>Yes</td>
<td>New Wasp mote API needed (v025 or newer)</td>
</tr>
<tr>
<td>Old 3G codes</td>
<td>No</td>
<td>The new 4G module provides new improved examples and libraries</td>
</tr>
</tbody>
</table>
3. Hardware

3.1. Specifications

The 4G module is based on the LE910 chipset, manufactured by Telit. The module is managed by UART and it must be connected to socket 1 (direct connection, without Expansion Board). The main features of the module are listed below:

- **Output power:**
  - Class 4 (2 W, 33 dBm) @ GSM 850 / 900
  - Class 1 (1 W, 30 dBm) @ GSM 1800 / 1900
  - Class E2 (0.5 W, 27 dBm) @ EDGE 850 / 900
  - Class E2 (0.4 W, 26 dBm) @ EDGE 1800 /1900
  - Class 3 (0.25 W, 24 dBm) @ UMTS
  - Class 3 (0.2 W, 23 dBm) @ LTE

- **Data transmission:**
  - LTE:
    - Uplink up to 50 Mbps
    - Downlink up to 100 Mbps
  - HSPA+:
    - Uplink up to 5.76 Mbps
    - Downlink up to 42.0 Mbps
  - UMTS:
    - Uplink/Downlink up to 384 kbps

- **Protocols:**
  - TCP/UDP
  - HTTP
  - FTP

- **GPS receiver** (in certain versions)

3.2. Versions

Telit has different versions of the LE910 chipset. Each one of them was especially designed to comply with the RF and cellular regulations in different countries or regions of the world. Libelium has integrated the following versions:

<table>
<thead>
<tr>
<th>Features</th>
<th>LE910 EUG</th>
<th>LE910 NAG</th>
<th>LE910 AU V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region</td>
<td>Europe and Brazil</td>
<td>USA and Canada (Americas)</td>
<td>Australia</td>
</tr>
<tr>
<td>Supported 4G bands</td>
<td>B20 (800), B3 (1800), B7 (2600)</td>
<td>B17 (700), B5 (850), B4 (1700), B2 (1900)</td>
<td>B3 (1800), B7 (2600), B28 (700)</td>
</tr>
<tr>
<td>3G fall-back</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Supported 3G bands</td>
<td>B5 (850), B8 (900), B1 (2100)</td>
<td>B5 (850), B2 (1900)</td>
<td>None</td>
</tr>
<tr>
<td>2G fall-back</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Supported 2G bands</td>
<td>GSM 900, DCS 1800</td>
<td>GSM 850, PCS 1900</td>
<td>None</td>
</tr>
<tr>
<td>GPS/GLONASS</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Certifications</td>
<td>CE (R&amp;TTE), GCF</td>
<td>FCC, IC, PTCRB, AT&amp;T</td>
<td>RCM, Telstra</td>
</tr>
</tbody>
</table>
Important note: The current stock of the LE910 4G radio that the manufacturer ‘Telit’ is distributing comprises the v2 version which does not have GPS. The models are:

- LE910-EU V2 for Europe or Brazil
- LE910-NA V2 for US or Canada

These v2 radios are similar to the v1 ones, but support more bands and do not have a GPS receiver.

<table>
<thead>
<tr>
<th>Features</th>
<th>LE910 EU V2</th>
<th>LE910 NA V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region</td>
<td>Europe</td>
<td>USA and Canada (Americas)</td>
</tr>
<tr>
<td>Supported 4G bands</td>
<td>B20 (800), B8 (900), B3 (1800), B1 (2100), B7 (2600)</td>
<td>B12 (700), B13 (700), B5 (850), B4 (1700), B2 (1900)</td>
</tr>
<tr>
<td>3G fall-back</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Supported 3G bands</td>
<td>B8 (900), B1 (2100)</td>
<td>B5 (850), B2 (1900)</td>
</tr>
<tr>
<td>2G fall-back</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Supported 2G bands</td>
<td>900 / 1800</td>
<td>None</td>
</tr>
<tr>
<td>GPS/GLONASS</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

3.3. How to connect the module

This module must be connected to the SOCKET1 on the Waspmote board. Like any other cellular radio, the connection is native so the radio does not need the Expansion Radio Board.

Figure: Module connected to Waspmote in SOCKET1
3.4. Antennas

The 4G module comes with 2 cellular antennas for improving the signal reception: normal (main) antenna and diversity antenna. Besides, a 3rd antenna is also included for the GPS receiver (when it is available in the 4G module).

All these 3 antennas are the same model and can be used in any of the 4G sockets. The operating bands of the dipole antenna go from 698 to 960 MHz and from 1710 to 2690 MHz. The maximum gain of the antenna is observed at 2.6 GHz: 3.4 dBi.

To get the maximum performance, it is recommended to place the antennas like that:

- The main cellular antenna should be in vertical position, pointing to the sky, in order to radiate better to the cellular base stations around.
- The diversity cellular antenna should be in horizontal position (orthogonal, 90°, to the main antenna). Besides, the plane where the antenna is should be also orthogonal to the main antenna’s plain. Finally, it is advised to place this 2nd cellular antenna as far as possible from the main antenna. These 3 measures will maximize the gain due to reception diversity.
- The GPS antenna should be in horizontal position, because the GPS satellite signal will come from above.

![Figure: 4G module antennas](image-url)
3.5. Power consumption

The 4G module is directly powered by the battery. The next table shows the Waspmote’s peak current consumption in different states of the 4G module.

<table>
<thead>
<tr>
<th>State</th>
<th>Mean power consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td>100 mA</td>
</tr>
<tr>
<td>Transmitting data</td>
<td>400 mA</td>
</tr>
<tr>
<td>Receiving data</td>
<td>400 mA</td>
</tr>
</tbody>
</table>

Non-rechargeable batteries are not advised for the 4G module, because the high peaks of current consumption could make the voltage of these batteries to go below 3.3 V so Waspmote would reset. The rechargeable battery will not suffer this effect as long as its level is above 20%.

3.6. Time consumption

The following table describes the mean elapsed time for some actions in a single test for several attempts:

<table>
<thead>
<tr>
<th>Action</th>
<th>Mean elapsed time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power on</td>
<td>~11 s</td>
</tr>
<tr>
<td>Start data connection</td>
<td>~4 s</td>
</tr>
<tr>
<td>Perform HTTP GET or POST</td>
<td>~0.7 s</td>
</tr>
<tr>
<td>Open FTP session</td>
<td>~3 s</td>
</tr>
<tr>
<td>Perform FTP upload 10 kB file</td>
<td>~7 s</td>
</tr>
<tr>
<td>Perform FTP download 10 kB file</td>
<td>~6 s</td>
</tr>
</tbody>
</table>

Some of these actions approximately have a fixed elapsed time like powering on the module. However, the actions related to data transmission (HTTP, FTP, etc.) are dependent on external circumstances (MNO, coverage quality, etc) and show more variability from the mean value.
4. Software

4.1. Waspmote library

4.1.1. Waspmote 4G library

The files related to the 4G module library are:

/Wasp4G/Wasp4G.h
/Wasp4G/Wasp4G.cpp
/Wasp4G/utility/Wasp4G_constants.h
/Wasp4G/utility/Wasp4G_error_codes.h

It is mandatory to include the 4G library when using this module. So the following line must be added at the beginning of the code:

#include <Wasp4G.h>

4.1.2. Class constructor

To start using the Waspmote 4G library, an object from the Wasp4G class must be created. This object, called _4G, is already created by default inside the Waspmote 4G library. It will be used along this guide to show how Waspmote works.

When using the class constructor, all variables are initialized to their default values.

4.1.3. API constants

The API constants used in functions are:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG_WASP4G</td>
<td>This definition enables/disables the debug mode via USB port:</td>
</tr>
<tr>
<td></td>
<td>0: No debug mode enabled</td>
</tr>
<tr>
<td></td>
<td>1: Debug mode enabled for error output messages</td>
</tr>
<tr>
<td></td>
<td>2: Debug mode enabled for both error and OK messages</td>
</tr>
<tr>
<td>LE910_RATE</td>
<td>Module's communication baud rate</td>
</tr>
<tr>
<td>LE910_INCOMING_SMS</td>
<td>Constant to set incoming data type when SMS received</td>
</tr>
<tr>
<td>LE910_INCOMING_IP</td>
<td>Constant to set incoming data type when IP received</td>
</tr>
<tr>
<td>LE910_MAX_DL_PAYLOAD</td>
<td>Maximum data payload size to be stored in the data buffer</td>
</tr>
</tbody>
</table>

There are several enumeration definitions for the function inputs. Please refer to the corresponding section in order to know more about the functions input parameters.
### 4.1.4. API variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_buffer</td>
<td>The buffer of memory used for storing the responses from the module (512 bytes)</td>
</tr>
<tr>
<td>_length</td>
<td>The length of the contents stored in _buffer</td>
</tr>
<tr>
<td>_def_delay</td>
<td>The time to wait after sending every command until listen for a response</td>
</tr>
<tr>
<td>_baudrate</td>
<td>The baudrate to be used when the module is switched on</td>
</tr>
<tr>
<td>_uart</td>
<td>The selected UART (regarding the socket used: SOCKET0 or SOCKET1)</td>
</tr>
<tr>
<td>_errorCode</td>
<td>It stores the error code returned by the module when calling a function with error response</td>
</tr>
<tr>
<td>_ip</td>
<td>IP address assigned to the 4G module when it is connected to the network</td>
</tr>
<tr>
<td>_temp</td>
<td>It stores temperature from the module</td>
</tr>
<tr>
<td>_tempInterval</td>
<td>It stores temperature interval from the module</td>
</tr>
<tr>
<td>_rssi</td>
<td>It stores the module's RSSI level</td>
</tr>
<tr>
<td>_networkType</td>
<td>It stores the network type which module is connected to</td>
</tr>
<tr>
<td>_incomingType</td>
<td>It stores the incoming data type to be read via serial</td>
</tr>
<tr>
<td>_smsIndex</td>
<td>It stores the SMS index where the incoming message was saved</td>
</tr>
<tr>
<td>_smsStatus</td>
<td>It stores the status of the SMS read from module's memory</td>
</tr>
<tr>
<td>_smsNumber</td>
<td>It stores the phone number of the SMS read from module's memory</td>
</tr>
<tr>
<td>_smsDate</td>
<td>It stores the date of the SMS read from module's memory</td>
</tr>
<tr>
<td>_smsTime</td>
<td>It stores the time of the SMS read from module's memory</td>
</tr>
<tr>
<td>_socketIndex</td>
<td>It stores the socket index which is getting data from UDP or TCP</td>
</tr>
<tr>
<td>_httpCode</td>
<td>It stores the HTTP status code</td>
</tr>
<tr>
<td>_filesize</td>
<td>It stores the size of the file when FTP upload/download is performed</td>
</tr>
<tr>
<td>_ftpWorkingDirectory</td>
<td>It stores the current working directory of the FTP session</td>
</tr>
<tr>
<td>_latitude</td>
<td>It stores latitude from GPS</td>
</tr>
<tr>
<td>_latitudeNS</td>
<td>It stores the north/south indicator from GPS</td>
</tr>
<tr>
<td>_longitude</td>
<td>It stores longitude from GPS</td>
</tr>
<tr>
<td>_longitudeEW</td>
<td>It stores east/west indicator</td>
</tr>
<tr>
<td>_altitude</td>
<td>It stores altitude from GPS</td>
</tr>
<tr>
<td>_time</td>
<td>It stores time from GPS</td>
</tr>
<tr>
<td>_date</td>
<td>It stores date from GPS</td>
</tr>
<tr>
<td>_numSatellites</td>
<td>It stores the number of satellites “in sight” of the GPS</td>
</tr>
<tr>
<td>_fixMode</td>
<td>It stores fix mode set from GPS</td>
</tr>
<tr>
<td>_speedOG</td>
<td>It stores speed over ground from GPS</td>
</tr>
<tr>
<td>_courseOG</td>
<td>It stores course over ground from GPS</td>
</tr>
<tr>
<td>_hdop</td>
<td>It stores the horizontal dilution of precision from GPS</td>
</tr>
</tbody>
</table>
4.1.5. API functions

Through this guide there are lots of examples, showing the use of functions. In these examples, API functions are called to execute the commands, storing in their related variables the parameter value in each case. The functions are called using the predefined object _4G.

All public functions return different possible values:
- 0: OK
- Otherwise: ERROR. See corresponding function error code

4.1.6. Error codes

When the 4G module returns an error code, the _errorCode variable stores the corresponding error meaning. Do not confuse this error code with the returning value from the API functions. There are other types of errors like “no response from the module” which are not included in the next list. For each function answer, please refer to the corresponding error values described for each function within the libraries.

The possible module’s error codes are described by constants as the table below:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Error code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASP4G_CME_ERROR_0000</td>
<td>0</td>
<td>Phone failure</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0001</td>
<td>1</td>
<td>No connection to phone</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0002</td>
<td>2</td>
<td>Phone-adapter link reserved</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0003</td>
<td>3</td>
<td>Operation not allowed</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0004</td>
<td>4</td>
<td>Operation not supported</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0005</td>
<td>5</td>
<td>Phone SIM (Subscriber Identity Module) PIN (Personal Identification Number) required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0010</td>
<td>10</td>
<td>SIM not inserted</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0011</td>
<td>11</td>
<td>SIM PIN required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0012</td>
<td>12</td>
<td>SIM PUK (Personal Unlocking Key) required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0013</td>
<td>13</td>
<td>SIM failure</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0014</td>
<td>14</td>
<td>SIM busy</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0015</td>
<td>15</td>
<td>SIM wrong</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0016</td>
<td>16</td>
<td>Incorrect password</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0017</td>
<td>17</td>
<td>SIM PIN2 required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0018</td>
<td>18</td>
<td>SIM PUK2 required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0020</td>
<td>20</td>
<td>Memory full</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0021</td>
<td>21</td>
<td>Invalid index</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0022</td>
<td>22</td>
<td>Not found</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0023</td>
<td>23</td>
<td>Memory failure</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0024</td>
<td>24</td>
<td>Text string too long</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0025</td>
<td>25</td>
<td>Invalid characters in text string</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0026</td>
<td>26</td>
<td>Dial string too long</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0027</td>
<td>27</td>
<td>Invalid characters in dial string</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0030</td>
<td>30</td>
<td>No network service</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0031</td>
<td>31</td>
<td>Network time-out</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0032</td>
<td>32</td>
<td>Network not allowed - emergency calls only</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0040</td>
<td>40</td>
<td>Network personalization PIN required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0041</td>
<td>41</td>
<td>Network personalization PUK required</td>
</tr>
<tr>
<td>Error Code</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0042</td>
<td>42</td>
<td>Network subset personalization PIN required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0043</td>
<td>43</td>
<td>Network subset personalization PUK required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0044</td>
<td>44</td>
<td>Service provider personalization PIN required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0045</td>
<td>45</td>
<td>Service provider personalization PUK required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0046</td>
<td>46</td>
<td>Corporate personalization PIN required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0047</td>
<td>47</td>
<td>Corporate personalization PUK required</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0100</td>
<td>100</td>
<td>Unknown</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0103</td>
<td>103</td>
<td>Illegal Mobile Station (MS) (#3)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0106</td>
<td>106</td>
<td>Illegal Mobile Equipment (ME) (#6)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0107</td>
<td>107</td>
<td>GPRS service not allowed (#7)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0111</td>
<td>111</td>
<td>PLMN not allowed (#11)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0112</td>
<td>112</td>
<td>Location area not allowed (#12)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0113</td>
<td>113</td>
<td>Roaming not allowed in this location area (#13)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0132</td>
<td>132</td>
<td>Service option not supported (#32)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0133</td>
<td>133</td>
<td>Requested service option not subscribed (#33)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0134</td>
<td>134</td>
<td>Service option temporarily out of order (#34)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0148</td>
<td>148</td>
<td>Unspecified GPRS error</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0149</td>
<td>149</td>
<td>PDP authentication failure</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0150</td>
<td>150</td>
<td>Invalid mobile class</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0550</td>
<td>550</td>
<td>Generic undocumented error</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0551</td>
<td>551</td>
<td>Wrong state</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0552</td>
<td>552</td>
<td>Wrong mode</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0553</td>
<td>553</td>
<td>Context already activated</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0554</td>
<td>554</td>
<td>Stack already active</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0555</td>
<td>555</td>
<td>Activation failed</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0556</td>
<td>556</td>
<td>Context not opened</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0557</td>
<td>557</td>
<td>Cannot setup socket</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0558</td>
<td>558</td>
<td>Cannot resolve DN</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0559</td>
<td>559</td>
<td>Time-out in opening socket</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0560</td>
<td>560</td>
<td>Cannot open socket</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0561</td>
<td>561</td>
<td>Remote disconnected or time-out</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0562</td>
<td>562</td>
<td>Connection failed</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0563</td>
<td>563</td>
<td>Transmission error</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0564</td>
<td>564</td>
<td>Already listening</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0565</td>
<td>565</td>
<td>Wrong PDP</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0566</td>
<td>566</td>
<td>FTP not connected</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0567</td>
<td>567</td>
<td>FTP write data closed</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0568</td>
<td>568</td>
<td>Network survey error (Busy)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0615</td>
<td>615</td>
<td>Network survey error (No Carrier)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0623</td>
<td>623</td>
<td>Network survey error (Wrong request)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0658</td>
<td>658</td>
<td>Network survey error (Aborted)*</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0659</td>
<td>659</td>
<td>Network rejected request</td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0660</td>
<td>660</td>
<td>Retry operation</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0259</td>
<td>Invalid deflected to number</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0260</td>
<td>Deflected to own number</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0261</td>
<td>Unknown subscriber</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0262</td>
<td>Service not available</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0263</td>
<td>Unknown class specified</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0264</td>
<td>Unknown network message</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0680</td>
<td>LU processing</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0681</td>
<td>Network search aborted</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0682</td>
<td>PTM mode</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0683</td>
<td>Active call state</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_0684</td>
<td>SSL already activated</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0300</td>
<td>ME failure</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0301</td>
<td>SMS service of ME reserved</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0302</td>
<td>Operation not allowed</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0303</td>
<td>Operation not supported</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0304</td>
<td>Invalid PDU mode parameter</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0305</td>
<td>Invalid text mode parameter</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0310</td>
<td>SIM not inserted</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0311</td>
<td>SIM PIN required</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0312</td>
<td>Phone SIM PIN required</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0313</td>
<td>SIM failure</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0314</td>
<td>SIM busy</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0315</td>
<td>SIM wrong</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0316</td>
<td>SIM PUK required</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0317</td>
<td>SIM PIN2 required</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0318</td>
<td>SIM PUK2 required</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0320</td>
<td>Memory failure</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0321</td>
<td>Invalid memory index</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0322</td>
<td>Memory full</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0330</td>
<td>SMSC address unknown</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0331</td>
<td>No network service</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0332</td>
<td>Network time-out</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0340</td>
<td>No +CNMA acknowledgement expected</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CMS_ERROR_0500</td>
<td>Unknown error</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_1001</td>
<td>SSL certs and keys wrong or not stored</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_1003</td>
<td>SSL already activated</td>
<td></td>
</tr>
<tr>
<td>WASP4G_CME_ERROR_1008</td>
<td>SSL not connected</td>
<td></td>
</tr>
<tr>
<td>WASP4G_ERROR_TIMEOUT</td>
<td>Timeout error when running a command</td>
<td></td>
</tr>
<tr>
<td>WASP4G_ERROR_MESSAGE</td>
<td>Generic &quot;ERROR&quot; message from module</td>
<td></td>
</tr>
</tbody>
</table>
4.2. Switching on

The `ON()` function switches on the 4G module and it opens the MCU UART for communicating with the module. After this step, the module will be able to receive commands to manage it.

Example of use:

```c
{ 
  _4G.ON();
}
```

4.3. Switching off

The `OFF()` function allows the user to switch off the 4G module and close the UART. This function must be called in order to save battery when the module is not going to be used.

Example of use:

```c
{ 
  _4G.OFF();
}
```

4.4. SIM card

4.4.1. Entering PIN

The `enterPIN()` function allows the user to enter the PIN (Personal Identification Number) of the SIM (Subscriber Identification Module) card. If the SIM card has no PIN (or the PIN was disabled on the SIM card), it is not necessary to use this function.

Example for entering the PIN:

```c
{ 
  _4G.enterPIN("1234");
}
```

Besides, there is another function prototype in order to set a new one. It is mandatory to specify the current PIN number and the new one.

Example for setting a new PIN:

```c
{ 
  _4G.enterPIN("1234", "1111");
}
```

Example of entering the PIN number:

[www.libelium.com/development/waspmote/examples/4g-01-enter-pin-code](http://www.libelium.com/development/waspmote/examples/4g-01-enter-pin-code)
4.4.2. Getting IMEI, IMSI and ICCID

The `getInfo()` function can get more than one information field to the module. This function needs one input to indicate the type of information requested. The resulting information is stored in `_buffer`, and `_length` is the number of bytes in the buffer. The information possibilities are:

- `Wasp4G::INFO_HW`: To request the hardware revision
- `Wasp4G::INFO_MANUFACTURER_ID`: To request the manufacturer identifier
- `Wasp4G::INFO_MODEL_ID`: To request the model identifier
- `Wasp4G::INFO_REV_ID`: To request the firmware revision
- `Wasp4G::INFO_IMEI`: To request the IMEI (International Mobile Station Equipment Identity), which is the unique identifier of the module, similar to a MAC address
- `Wasp4G::INFO_IMSI`: To request the IMSI
- `Wasp4G::INFO_ICCID`: To request the ICCID

Examples of use:

```cpp
{
    // get IMEI number
    _4G.getInfo(Wasp4G::INFO_IMEI);

    // get IMSI number
    _4G.getInfo(Wasp4G::INFO_IMSI);

    // get ICCID number
    _4G.getInfo(Wasp4G::INFO_ICCID);
}
```

Related variables:

- `_4G._buffer` → Buffer which stores the information requested
- `_4G._length` → Number of bytes in buffer

Example of getting module info:

[www.libelium.com/development/waspmote/examples/4g-02-get-module-info](http://www.libelium.com/development/waspmote/examples/4g-02-get-module-info)
4.5. Checking network connection status

There are 2 functions to check the network connection status: `checkConnection()` and `checkDataConnection()`.

The `checkConnection()` function checks the module's network connection status and returns whether the module:

- is connected to a network
- is not connected to a network
- is searching for a new operator to register to
- registration was denied

This function will wait for the module to be connected to a network for the specified time in second units.

Example of use:

```
{  
   _4G.checkConnection(60);
}
```

Possible error codes for this function:

- 1: not registered, the Mobile Equipment (ME) is not currently searching for a new operator to register to
- 2: not registered, but ME is currently searching for a new operator to register to
- 3: registration denied
- 4: unknown

The `checkDataConnection()` function checks the module's network connection status, connects the module to data service and returns whether the module:

- is connected to data service
- is not connected to a network
- is searching for a new operator to register to
- registration was denied

This function will wait for the module to be connected to a network for the specified time in second units.

Example of use:

```
{  
   _4G.checkDataConnection(60);
}
```

Possible error codes for this function:

- 1: not registered, ME is not currently searching for a new operator to register to
- 2: not registered, but ME is currently searching for a new operator to register to
- 3: registration denied
- 4: unknown
- 6: not registered, ME is not currently searching for a new operator to register to
- 8: not registered, but ME is currently searching for a new operator to register to
- 9: registration denied
- 10: unknown
- 12: if error setting APN
- 13: if error setting login
- 14: if error setting password
- 15: if error activating GPRS connection
4.6. Setting operator parameters

When the 4G module uses data services like TCP/UDP connections, HTTP services, SMTP or FTP transfers, it is mandatory to configure the parameters provided by the user’s Mobile Network Operator (MNO): APN, login and password. The owner of a SIM should be notified with these parameters by the MNO.

The `set_APN()` function allows the user to save these parameters into Waspmote memory. Later, when data connection functions are called, Waspmote will configure these parameters into the 4G module.

Example of use:

```c
_4G.set_APN("apn", "login", "password");
```

The `show_APN()` function allows the user to display the current settings stored in Waspmote’s memory which are used by the libraries when data connections are performed.

Example of use:

```c
_4G.show_APN();
```

Related variable:

- `_4G._apn` → Stores the APN name
- `_4G._apn_login` → Stores the APN login
- `_4G._apn_password` → Stores the APN password

4.7. SMS

4.7.1. Setting SMS configuration

The `configureSMS()` function sets the SMS configuration to:

- Correct SMS format to be read from the module
- Select where SMSs are going to be stored
- Configure the indication method for new SMS received

This function must be called before handling SMSs in the main code loop.

Example of use:

```c
_4G.configureSMS();
```
4.7.2. Sending SMSs

The `sendSMS()` function sends an SMS to the given phone number. The maximum length of the message to be sent is 164 bytes.

Example of use:

```c
{
    char phone_number[] = "*********";
    char text_message[] = "This_is_a_text_message";
    _4G.sendSMS(phone_number, text_message);
}
```

Possible error codes for this function:

- 1: not registered, ME is not currently searching for a new operator to register to
- 2: not registered, but ME is currently searching for a new operator to register to
- 3: registration denied
- 4: unknown connection error
- 5: if error setting the phone number
- 6: if error sending the body

Example of sending a text message:

www.libelium.com/development/waspmote/examples/4g-04-sending-sms

4.7.3. Reading SMSs

The `readSMS()` function reads an SMS from the module storage. The user must give the index of the SMS to be read from memory. In the case a new SMS is received, all SMS related parameters are stored in the Waspmote's library structures: index number, status, sender number, date and time. The SMS text field can be found in `_4G._buffer`.

Example of use:

```c
{
    uint8_t index = 5;
    _4G.readSMS(index);
}
```

The `readNewSMS()` function reads the last unread SMS received by the module. There are 2 function prototypes: with or without timeout. If the user needs to wait for a new text message for a period of time, the timeout must be specified as input. If no input is defined, the request for new text messages is instantly performed. In the case a new SMS is received, all SMS related parameters are stored in the Waspmote's library structures: index number, status, sender number, date and time. The SMS text field can be found in `_4G._buffer`.

Example of use:

```c
{
    _4G.readNewSMS(30000);
}
```
Related variables:

- 4G._smsIndex → Stores the SMS index in memory storage
- 4G._smsStatus → Stores the SMS status:
  - “REC UNREAD” - new received message unread
  - “REC READ” - received message read
  - “STO UNSENT” - message stored not yet sent
  - “STO SENT” - message stored already sent
- 4G._smsNumber → Stores the sender phone number
- 4G._smsDate → Stores the date SMS was sent
- 4G._smsTime → Stores the time SMS was sent
- 4G._buffer → Stores SMS text field temporary, after calling the read function
- 4G._length → Stores the SMS message length

Example of receiving and deleting text messages:
www.libelium.com/development/waspmote/examples/4g-05-receiving-sms

4.7.4. Deleting SMSs

The deleteSMS() function deletes an SMS according to the given index number in the memory storage.

Examples of use:

```c
{
    uint8_t index = 2;
    _4G.deleteSMS(index);
}
```

There is a second deleteSMS() function prototype which deletes all the SMSs in the storage memory according to several possibilities. For this function, 2 inputs are introduced: the SMS index number and the delete flag. If delete flag is not set to delete one single message, then index is ignored and the module shall follow the rules for delete flag shown below:

- **Wasp4G::SMS_DELETE_MESSAGE**: To delete the message specified in index
- **Wasp4G::SMS_DELETE_ALL_1**: To delete all read messages from memory storage, leaving unread messages and stored mobile originated messages (whether sent or not) untouched
- **Wasp4G::SMS_DELETE_ALL_2**: To delete all read messages from memory storage and sent mobile originated messages, leaving unread messages and unsent mobile originated messages untouched
- **Wasp4G::SMS_DELETE_ALL_3**: To delete all read messages from memory storage, sent and unsent mobile originated messages, leaving unread messages untouched
- **Wasp4G::SMS_DELETE_ALL_4**: To delete all messages from memory storage

Example of use:

```c
{
    _4G.deleteSMS(index, Wasp4G::SMS_DELETE_ALL_1);
}
```

Example of receiving and deleting text messages:
www.libelium.com/development/waspmote/examples/4g-05-receiving-sms
4.8. HTTP client

4.8.1. HTTP connections

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 4G, 3G, GPRS, GPRS+GPS or WiFi modules 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. These communication modules can send HTTP instances to our web service.

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

**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>&<...>&view=html
```

Where:

- `<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
```

**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 data field is very similar as 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:

```
pruebas.libelium.com/getpost_frame_parser.php?variable1=3.141592
pruebas.libelium.com/getpost_frame_parser.php?var1=3.1415&var2=123456&var3=hello
```
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.

![Operating with the web service from a PC browser](image)

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](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...

### 4.8.2. HTTP request methods

The `http()` function configures HTTP parameters, performs the request selected by the user and handles the data returned from the server.

This function needs several parameters to work properly depending on the method used. The first parameter required by the function is the request method. User can choose among five methods: GET, HEAD, DELETE, POST and PUT:

- `Wasp4G::HTTP_GET`
- `Wasp4G::HTTP_HEAD`
- `Wasp4G::HTTP_DELETE`
- `Wasp4G::HTTP_POST`
- `Wasp4G::HTTP_PUT`

After choosing the method, the function needs the host URL, port and resource of the HTTP server requested. The data field is only necessary when POST or PUT methods are performed.

Example of use (GET, HEAD and DELETE methods):

```c
{  
  char  host[] = “test.libelium.com”;  
  uint16_t  port = 80;  
  char  resource[] = “/test-get-post.php?varA=1&varB=2&varC=3&varD=4”;  
  _4G.http(Wasp4G::HTTP_GET, host, port, resource);  
}
```
Example of use (POST and PUT methods):

```c
char  host[] = "test.libelium.com";
uint16_t  port = 80;
char  resource[] = "/test-get-post.php";
char  data[] = "varA=1&varB=2&varC=3&varD=4&varE=5";

_4G.http(Wasp4G::HTTP_POST, host, port, resource, data);
```

Once the request has been sent, the function waits for data from the server and stores it in `_buffer`. It also stores the HTTP status code from the server in `httpCode`.

Related variable:
- `_4G._httpCode` → Stores the HTTP code from the server
- `_4G._buffer` → Stores data received from server
- `_4G._length` → Stores data length

Possible error codes for this function:

- 1: not registered, ME is not currently searching for a new operator to register to
- 2: not registered, but ME is currently searching for a new operator to register to
- 3: registration denied
- 4: unknown
- 6: not registered, ME is not currently searching for a new operator to register to
- 8: not registered, but ME is currently searching for a new operator to register to
- 9: registration denied
- 10: unknown
- 12: if error setting APN
- 13: if error setting login
- 14: if error setting password
- 15: if error activating GPRS connection
- 16: if error setting URL and port
- 17: if error sending the request
- 18: if error sending POST / PUT data
- 19: if wrong method has been selected
- 20: if timeout waiting the URC
- 21: if error reading the URC
- 22: if error reading the HTTP status
- 23: if error reading the HTTP data length
- 24: if error reading the HTTP data
- 25: if error code from 4G module while waiting for HTTP response
- 26: if timeout waiting for HTTP response
- 27: if HTTP response data length is zero

Example of HTTP GET:
www.libelium.com/development/waspmote/examples/4g-06-http-get

Example of HTTP POST:
www.libelium.com/development/waspmote/examples/4g-07-http-post
4.8.3. Sending Waspmote frames to Meshlium

The `sendFrameToMeshlium()` function has been developed to send Waspmote frames from Waspmote to Meshlium. Meshlium will parse (chop) the frame and will store it in its internal MySQL database. This function requires the following parameters:

- Meshlium’s IP address
- Meshlium’s remote port
- Data to send: `frame.buffer` will be used from the generated frame
- Data length: `frame.length` will be used from the generated frame

![Diagram showing the process of sending frames to Meshlium via 4G.](image)

After calling the function, the response from Meshlium will be stored in `_buffer`. Besides, it will store the HTTP status code from server in `_httpCode`. Please refer to the [Data Frame Guide](#) in order to know more about how to create sensor frames with Waspmote libraries.

Example of use:

```c
char host[] = “pruebas.libelium.com”;
uint16_t port = 80;

// after frame has been created
_4G.sendFrameToMeshlium(host, port, frame.buffer, frame.length);
```

Related variable:

- `_4G._httpCode` → Stores the HTTP code from the server
- `_4G._buffer` → Stores data received from server
- `_4G._length` → Stores data length
- `frame.buffer` → Stores data frame that will be sent to Meshlium
- `frame.length` → Stores data frame length
Possible error codes for this function:

- 1: not registered, ME is not currently searching for a new operator to register to
- 2: not registered, but ME is currently searching for a new operator to register to
- 3: registration denied
- 4: unknown
- 6: not registered, ME is not currently searching for a new operator to register to
- 8: not registered, but ME is currently searching for a new operator to register to
- 9: registration denied
- 10: unknown
- 12: if error setting APN
- 13: if error setting login
- 14: if error setting password
- 15: if error activating GPRS connection
- 16: if error setting URL and port
- 17: if error sending the request
- 18: if error sending POST / PUT data
- 19: if wrong method has been selected
- 20: if timeout waiting the URC
- 21: if error reading the URC
- 22: if error reading the HTTP status
- 23: if error reading the HTTP data length
- 24: if error reading the HTTP data
- 25: if error code from 4G module while waiting for HTTP response
- 26: if timeout waiting for HTTP response
- 27: if HTTP response data length is zero

Example of sending frames to Meshlium:
www.libelium.com/development/waspmote/examples/4g-08-send-frames-to-meshlium

4.9. FTP client

4.9.1. Opening an FTP session

The `ftpOpenSession()` function configures FTP parameters and opens the connection. Several inputs are needed:

- FTP server: IP address or URL
- FTP port number
- Username
- Password

Example of use:

```c
{
    char ftp_server[] = "pruebas.libelium.com";
    uint16_t ftp_port = 21;
    char ftp_user[] = "t3g@libelium.com";
    char ftp_pass[] = "ftp1234";

    _4G.ftpOpenSession(ftp_server, ftp_port, ftp_user, ftp_pass);
}
```
Possible error codes for this function:

- 1: not registered, ME is not currently searching for a new operator to register to
- 2: not registered, but ME is currently searching for a new operator to register to
- 3: registration denied
- 4: unknown
- 6: not registered, ME is not currently searching for a new operator to register to
- 8: not registered, but ME is currently searching for a new operator to register to
- 9: registration denied
- 10: unknown
- 12: if error setting APN
- 13: if error setting login
- 14: if error setting password
- 15: if error activating GPRS connection
- 16: if error opening the FTP connection
- 17: if error setting the transfer type

After successfully calling this function, it will be possible to manage the rest of FTP functions for uploading and downloading files.

4.9.2. FTP upload

The `ftpUpload()` function uploads a file from the Waspmote's SD card to the FTP server. The FTP session must be already open. This function needs 2 different inputs: the complete path of the file to be created in the FTP server and the complete path of the file in the SD card to be uploaded.

Example of use for files in root directory:

```c
{
    char sd_file[] = “FILE1.TXT”;
    char server_file[] = “FILE2.TXT”;
    _4G.ftpUpload(server_file, sd_file);
}
```

In the case the file should be uploaded into a subdirectory instead of the root directory, the server filename must be accordingly defined. The user must keep in mind that subdirectories have to be already created in order to upload files into them. In other words, it is not possible to create subdirectories in the FTP server.

Example of use for files in subdirectories:

```c
{
    char sd_file[] = “/FOLDER1/FILE1.TXT”;
    char server_file[] = “/FOLDER2/FILE2.TXT”;
    _4G.ftpUpload(server_file, sd_file);
}
```
Possible error codes for this function:

- 1: if no SD present
- 2: if file does not exist
- 3: if error opening the file
- 4: if error setting the pointer of the file
- 5: if error getting the file size
- 6: if error opening the PUT connection
- 7: if error exiting from the data mode
- 8: if error sending data

Example of uploading files:
www.libelium.com/development/waspmote/examples/4g-09-ftp-upload

4.9.3. FTP download

The `ftpDownload()` function downloads a file from an FTP server to Waspmote's SD card. The FTP session must be already open. This function needs 2 different inputs: the complete path of the file in the FTP server and the complete path of the file to be created in the SD card.

Example of use for files in root directory:

```c
{
    char sd_file[] = "FILE1.TXT";
    char server_file[] = "FILE2.TXT";
    _4G.ftpDownload(server_file, sd_file);
}
```

In the case the file should be downloaded into an SD card's subdirectory instead of the root directory, the SD filename must be accordingly defined. The user must keep in mind that subdirectories have to be already created in the SD card in order to create files into them.

Example of use for files in subdirectories:

```c
{
    char sd_file[] = "/FOLDER1/FILE1.TXT";
    char server_file[] = "/FOLDER2/FILE2.TXT";
    _4G.ftpDownload(server_file, sd_file);
}
```

Possible error codes for this function:

- 1: if server file size is zero
- 2: if error reading the file size
- 3: if SD not present
- 4: if error creating the file in SD
- 5: if error opening the file
- 6: if error setting the pointer of the file
- 7: if error opening the GET connection
- 8: if the module returns error code after requesting data
- 9: if error getting packet size
- 10: if error in packet size mismatch
• 11: if error writing SD error
• 12: if no more retries getting data
• 13: if file size mismatch

Example of downloading files:
www.libelium.com/development/waspmote/examples/4g-10-ftp-download

4.9.4. FTP delete file

The ftpDelete() function deletes in the FTP server. The FTP session must be already open. The function expects the name of the file to be deleted as input.

Example of deleting a file:
```c
{ 
  _4G.ftpDelete("FILE_FTP.TXT");
}
```

4.9.5. FTP working directory

The ftpGetWorkingDirectory() function requests the current working directory of the previously open FTP session. The function updates the _ftpWorkingDirectory attribute which stores the information.

Example of getting the FTP working directory:
```c
{ 
  _4G.ftpGetWorkingDirectory();
}
```

Related variable:

`_4G._ftpWorkingDirectory` → Stores the current working directory

4.9.6. Closing an FTP session

The ftpCloseSession() function allows the user to close an active FTP session.

Example of closing an FTP session:
```c
{ 
  _4G.ftpCloseSession();
}
```
4.10. TCP/UDP connections

4.10.1. Socket identifiers

The 4G module permits to have up to 6 simultaneous TCP/UDP connections. For that purpose, the libraries define the following socket identifiers to be used when handling the multi-socket connections:

- Wasp4G::CONNECTION_1
- Wasp4G::CONNECTION_2
- Wasp4G::CONNECTION_3
- Wasp4G::CONNECTION_4
- Wasp4G::CONNECTION_5
- Wasp4G::CONNECTION_6

The 4G libraries define different structures in order to store the information related to all socket identifiers. After opening sockets or sending/receiving data, the structures are updated. So this is useful in order to manage the most important settings of the connection.

4.10.2. Socket information structure

The `SocketInfo_t` structure stores the information to be stored for all sockets. For each one of the connections, the information structure includes:

- Socket identifier
- Total number of bytes sent since the socket was opened
- Total number of bytes received since socket was opened
- Total number of pending bytes to read which arrived through the socket
- Total number of bytes sent and not yet acknowledged since the socket was opened

As there are six possible connections at the same time, the global variable is defined as follows:

```c
SocketInfo_t  socketInfo[6];
```

The definition of the structure is:

```c
struct SocketInfo_t
{
    uint8_t   id;
    uint16_t  sent;
    uint16_t  received;
    uint16_t  size;
    uint16_t  ack;
};
```

The `getSocketInfo()` function allows the user to update the socket information structure from the 4G module. It is mandatory to indicate the identifier of the socket to be updated.

Example of use:

```c
{
    uint8_t socketId = Wasp4G::CONNECTION_1;
    _4G.getSocketInfo(socketId);
}
```
Related variables:

- `_4G.socketInfo[socketId].id` → Socket identifier
- `_4G.socketInfo[socketId].sent` → Total number of bytes sent since the socket was opened
- `_4G.socketInfo[socketId].received` → Total number of bytes received
- `_4G.socketInfo[socketId].size` → Total number of pending bytes to read
- `_4G.socketInfo[socketId].ack` → Total number of bytes sent and not yet acknowledged

### 4.10.3. Socket status structure

The `SocketStatus_t` structure stores the status for all sockets. For each one of the connections, the status structure includes:

- Socket identifier
- Current socket status. The API defines several constants to describe it:
  - `Wasp4G::STATUS_CLOSED`
  - `Wasp4G::STATUS_ACTIVE`
  - `Wasp4G::STATUS_SUSPENDED`
  - `Wasp4G::STATUS_SUSPENDED_DATA`
  - `Wasp4G::STATUS_LISTENING`
  - `Wasp4G::STATUS_INCOMING`
  - `Wasp4G::STATUS_OPENING`
- Local IP address
- Local port
- Remote IP address
- Remote port

As it is possible to have up to 6 simultaneous connections, the global variable is defined as follows:

```c
SocketStatus_t   socketStatus[6];
```

The definition of the structure is:

```c
struct SocketStatus_t
{
    uint8_t  id;
    uint8_t  state;
    char     localIp[16];
    uint16_t localPort;
    char     remoteIp[16];
    uint16_t remotePort;
};
```

The `getSocketStatus()` function allows the user to update the socket status structure from the 4G module. It is mandatory to indicate the identifier of the socket to be updated. It is possible to update all socket status by calling the `getAllSocketStatus()` function which is faster than iterating through all different identifiers.

Example of use:

```c
{
    uint8_t socketId = Wasp4G::CONNECTION_1;
    _4G.getSocketStatus(socketId);
}
```
Related variables:

- `_4G.socketInfo[socketId].id` → Socket identifier
- `_4G.socketInfo[socketId].state` → Socket status
- `_4G.socketInfo[socketId].localIp` → Local IP address
- `_4G.socketInfo[socketId].localPort` → Local port
- `_4G.socketInfo[socketId].remoteIp` → Remote IP address
- `_4G.socketInfo[socketId].remotePort` → Remote port

### 4.10.4. Creating a TCP/UDP client socket

The `openSocketClient()` function configures and opens a socket. This function expects several input parameters:

- **Socket ID**: The first parameter indicates the identifier to be associated to the new TCP/UDP connection. This identifier is needed in order to send or receive data through the corresponding socket after creating it.
- **Protocol**: This parameter indicates whether use TCP or UDP protocol for the new socket. The possibilities are:
  - Wasp4G::TCP
  - Wasp4G::UDP
- **Host**: Address of the remote host, string type. This parameter can be either:
  - Any valid IP address in the format: “xxx.xxx.xxx.xxx”
  - Any host name to be solved with a DNS query
- **Remote port**: Remote host port to contact from 1 to 65535.
- **Local port**: Parameter is valid for UDP connections only and has no effect (if used) for TCP connections. UDP connections local port from 1 to 65535.

Example of creating a **TCP client** connection:

```c
#include <4G.h>

{   uint8_t socketId = Wasp4G::CONNECTION_1;
    char host[] = “xxx.xxx.xxx.xxx”;
    uint16_t remote_port = 15010;

    _4G.openSocketClient(socketId, Wasp4G::TCP, host, remote_port);
}
```

Example of creating a **UDP client** connection:

```c
#include <4G.h>

{   uint8_t socketId = Wasp4G::CONNECTION_2;
    char host[] = “xxx.xxx.xxx.xxx”;
    uint16_t remote_port = 15010;
    uint16_t local_port = 4000;

    _4G.openSocketClient(socketId, Wasp4G::UDP, host, remote_port, local_port);
}
```

Possible error codes for this function:

- 1: not registered, ME is not currently searching for a new operator to register to
- 2: not registered, but ME is currently searching for a new operator to register to
- 3: registration denied
- 4: unknown
- 6: not registered, ME is not currently searching for a new operator to register to
- 8: not registered, but ME is currently searching for a new operator to register to
- 9: registration denied
- 10: unknown
- 12: if error setting APN
- 13: if error setting login
- 14: if error setting password
- 15: if error activating GPRS connection
- 16: if error getting socket status
- 17: Socket with an active data transfer connection
- 18: Socket suspended
- 19: Socket suspended with pending data
- 20: Socket listening
- 21: Socket with an incoming connection. Waiting for the user accept or shutdown command
- 22: Socket in opening process. The socket is not closed but still not in Active or Suspended
- 23: if error in Socket Configuration
- 24: if error in Socket Configuration Extended 3
- 25: if error sending the open command
- 26: if timeout opening the socket

Example of creating TCP/UDP client sockets:

www.libelium.com/development/waspmote/examples/4g-11-tcp-client
www.libelium.com/development/waspmote/examples/4g-13-udp-client

4.10.5. Creating a TCP/UDP server socket

The `openSocketServer()` function configures and opens a listening socket. This function expects several input parameters:

- **Socket ID**: The first parameter indicates the identifier to be associated to the new TCP/UDP connection. This identifier is needed in order to send or receive data through the corresponding socket.
- **Protocol**: This parameter indicates whether use TCP or UDP protocol for the new socket. The possibilities are:
  - `Wasp4G::TCP`
  - `Wasp4G::UDP`
- **Local port**: Local listening port from 1 to 65535.
- **Keep-Alive**: TCP keep-alive timer timeout - The interval between 2 keep-alive transmissions in idle condition:
  - 0 → TCP keep-alive timer is deactivated (default)
  - 1..240 → TCP keep-alive timer timeout in minutes

Example of creating a TCP server:

```c
{
    uint8_t  socketId   = Wasp4G::CONNECTION_1;
    uint16_t local_port = 5000;
    uint8_t  keep_alive = 240;

    _4G.openSocketServer(socketId, Wasp4G::TCP, local_port, keep_alive);
}
```
Example of creating a **UDP server**:

```
{
    uint8_t socketId = Wasp4G::CONNECTION_2;
    uint16_t local_port = 5000;
    _4G.openSocketServer(socketId, Wasp4G::UDP, local_port);
}
```

Once the server is created, the `manageSockets()` function permits the user to update all socket status and accept connections if needed. So when setting up a server or listening socket in Waspmote 4G the best way to handle with connections is directly calling this function. If no input parameter is defined the calling is executed instantly. If the timeout is inserted as new input, the function will block until a new incoming connection needs to be managed or timeout. This timeout must be specified in milliseconds units.

Possible error codes for this function:

- 1: not registered, ME is not currently searching for a new operator to register to
- 2: not registered, but ME is currently searching for a new operator to register to
- 3: registration denied
- 4: unknown
- 6: not registered, ME is not currently searching for a new operator to register to
- 8: not registered, but ME is currently searching for a new operator to register to
- 9: registration denied
- 10: unknown
- 12: if error setting APN
- 13: if error setting login
- 14: if error setting password
- 15: if error activating GPRS connection
- 16: if error getting socket status
- 17: if error in Socket Configuration
- 18: if protocol input not valid
- 19: if error opening the socket

Example of creating TCP/UDP server sockets:
```
www.libelium.com/development/waspmote/examples/4g-12-tcp-server
www.libelium.com/development/waspmote/examples/4g-14-udp-server
```

### 4.10.6. Sending data

The `send()` function allows the user to send TCP/UDP packets once the socket is active. The function needs 2 different inputs parameters:

- **Socket ID**: the socket identifier used for opening the connection.
- **Data**: This is the stream of data to send to the TCP/UDP socket. This stream of data can be defined as a simple string message. On the other hand, the data can be defined as an array of bytes, specifying a third input for the length of the array of bytes to send.

Example for sending a string message:
```
{
    _4G.send(Wasp4G::CONNECTION_1, "This_is_the_data_payload");
}
```
Example for sending an array of bytes:

```c
uint8_t data[] = {0x31, 0x32, 0x33, 0x34, 0x35}
_4G.send(Wasp4G::CONNECTION_1, data, 6);
```

Possible error codes for this function:
- 1: if error checking socket status
- 2: if incorrect socket status
- 3: if error sending data
- 4: if error waiting confirmation from module
- 5: if error getting socket status
- 6: if timeout getting socket status

All examples related to TCP/UDP sockets (both client and server) show how to send data:
- [www.libelium.com/development/waspmote/examples/4g-11-tcp-client](http://www.libelium.com/development/waspmote/examples/4g-11-tcp-client)
- [www.libelium.com/development/waspmote/examples/4g-12-tcp-server](http://www.libelium.com/development/waspmote/examples/4g-12-tcp-server)
- [www.libelium.com/development/waspmote/examples/4g-14-udp-server](http://www.libelium.com/development/waspmote/examples/4g-14-udp-server)

### 4.10.7. Receiving data

The `receive()` function allows the user to receive TCP/UDP packets once the socket is active. The function needs different inputs:
- Socket ID: the socket identifier used for opening the connection.
- Timeout (optional input):
  - If no timeout input is specified, the receive function is a non-blocking function which answers if data has been received.
  - If the timeout is inserted as new input, the function will block until a new packet is received or time is up in the case no packet is received. This timeout must be specified in milliseconds units.

Example for instant reception:

```c
_4G.receive(Wasp4G::CONNECTION_1);
```

Example for blocking reception (i.e. 30 seconds):

```c
_4G.receive(Wasp4G::CONNECTION_1, 30000);
```

Related variables:
- `_4G._buffer` → Pointer to the buffer where received data is stored
- `_4G._length` → Length of the data received
Possible error codes for this function:

• 1: if no data received
• 2: if error getting socket info
• 3: if timeout waiting for data
• 4: if error receiving data from module
• 5: if error parsing length of data
• 6: if error reading incoming bytes

4.10.8. Closing a socket

The closeSocketClient() function allows the user to close a TCP/UDP client previously open. The function needs an input parameter for the socket identifier.

The closeSocketServer() function allows the user to close a TCP/UDP server previously open. The function needs 2 inputs:

• **Socket ID**: the socket identifier used for opening the connection.
• **Protocol**: This parameter indicates the protocol used by the socket:
  - Wasp4G::TCP
  - Wasp4G::UDP

4.10.9. SSL sockets

The 4G module includes a stack for establishing SSL sockets. For this feature, the user must keep in mind that it is necessary to install the proper security data in the module. For handling the SSL socket new functions are defined for opening the socket, sending data, receiving data and closing the socket.

Currently, for SSL sockets only one single connection is permitted. So, regarding the socket identifiers the only available option is: Wasp4G::CONNECTION_1.

The manageSSL() function allows the user to store, delete and read security data (Certificate, CA certificate, private key) into the non volatile memory of the module. The function expects several inputs:

• **Socket ID**: the socket identifier used for opening the connection.
• **Action**: the action to perform:
  - Wasp4G::SSL_ACTION_DELETE: Delete data from memory
  - Wasp4G::SSL_ACTION_STORE: Store data into memory
  - Wasp4G::SSL_ACTION_READ: Read data from memory
• **Data type**:
  - Wasp4G::SSL_TYPE_CERT: Certificate
  - Wasp4G::SSL_TYPE_CA_CERT: CA certificate
  - Wasp4G::SSL_TYPE_RSA: RSA Private key
• **Data (optional)**: this input is needed when the user selects to store a new security data into module's memory.

Possible error codes for this function:

• 1 if error setting security data
• 2 if error waiting module confirmation
• 3 if error getting security data
• 4 if error deleting security data
• 5 if invalid action input
The **openSocketSSL()** function allows the user to open a remote connection via socket secured through SSL. Several inputs are needed for calling this function:

- **Socket ID**: the socket identifier used for opening the connection
- **Host**: Remote SSL server address
- **Remote port**: Remote TCP port to contact from 1 to 65535.

Possible error codes for this function:

- 1: not registered, ME is not currently searching for a new operator to register to
- 2: not registered, but ME is currently searching for a new operator to register to
- 3: registration denied
- 4: unknown
- 6: not registered, ME is not currently searching for a new operator to register to
- 8: not registered, but ME is currently searching for a new operator to register to
- 9: registration denied
- 10: unknown
- 12: if error setting APN
- 13: if error setting login
- 14: if error setting password
- 15: if error activating GPRS connection
- 16: if error getting SSL Socket Status
- 17: if socket disabled
- 19: if socket already open
- 20: if error opening the socket
- 21: if no response from module

The **sendSSL()** function allows the user to send data through a secure socket. Several inputs are needed for calling this function:

- **Socket ID**: the socket identifier used for opening the connection.
- **Data**: Data to send.

Possible error codes for this function:

- 1: if error checking socket status
- 2: if incorrect socket status
- 3: if error sending data
- 4: if no response from module
- 5: if error getting socket status
- 6: if timeout waiting for correct socket status

The **receiveSSL()** function allows the user to receive data through a secure socket. Several inputs are needed for calling this function:

- **Socket ID**: the socket identifier used for opening the connection.
- **Timeout (optional input)**:
  - If no timeout input is specified, the receive function is a non-blocking function which answers if data has been received.
  - If the timeout is inserted as new input, the function will block until a new packet is received or time is up in the case no packet is received. This timeout must be specified in milliseconds units.
Possible error codes for this function:

- 1: if no answer from module
- 2: if SSL socket disconnected
- 3: if error code from module
- 4: if no response from module
- 5: if error parsing length of received data
- 6: if error getting received data
- 7: if error waiting module confirmation

The `closeSocketSSL()` function allows the user to close a secure socket. The function needs an input parameter for the socket identifier.

Example for SSL socket:
4.11. GPS

Nowadays there are several positioning techniques to provide the localization to end devices. One of them is the A-GPS positioning technique based on the help of a cellular network deploying an A-GPS server.

Remember the AU version does not have a GPS receiver.

At this point, it is advisable to introduce the definition of Time to First Fix (TTFF): TTFF indicates the time and process required for a GPS device to get adequate satellite signals and data to provide accurate navigation.

A-GPS uses the following sets of data to provide accurate position:

- GPS satellite signals
- Almanac data
- Ephemeris data

If a GPS device has been turned off for a long period of time, when it is turned on it will take longer to acquire these data sets and get a “Time to First Fix”. One way to speed up TTFF is to use the A-GPS Positioning Technique.

A “cold” start indicates the scenario in which the GPS must get all data in order to start navigation, and may take up to several minutes.

A “warm” start indicates the scenario in which the GPS has most of the data it needs in memory, and will start quickly, a minute or less.

Before dealing with the A-GPS service, a Standalone GPS solution is described. The figure below shows an overview of the involved functional entities.

![GPS overview](image)
4.11.1. Standalone or Autonomous GPS (S-GPS)

Standalone or autonomous GPS mode (S-GPS) is a feature that allows the GPS receiver, installed on the 4G module, to perform its First Fixing activity without assistance data coming from the cellular network. The GPS receiver estimates its position directly from GPS satellites in its line of sight. The S-GPS is sometimes slower to compute its First Fix; this phenomenon is evident in very poor signal conditions, for example in a city where the satellites signals are corrupted by the multipath propagation.

4.11.2. Assisted GPS (A-GPS)

Assisted GPS mode is a feature that allows the GPS receiver, installed on the module, to perform its First Fix using assistance data provided by entities deployed by the Cellular Network.

There are a couple of A-GPS standards. Waspmote libraries implement the Secure User Plane Location (SUPL) architecture. This Location Service architecture is composed of 2 basic elements: a SUPL Enabled Terminal (SET) and a SUPL Location Platform (SLP). The SET corresponds to the 4G module. The SLP manages several tasks: authentication, location request, etc. This module supports the SUPL ver. 1.0.

Waspmote libraries manage the SET Initiated Session scenario where the module, on its initiative, connects to an SLP Server through an IP network. 2 modes are available:

- MS-Assisted
- MS-Based

In **MS-Assisted** mode, the module receives acquisition assistance, reference time and other optional assistance data from the network. The mobile service provider continuously logs GPS information (mainly the almanac) from the GPS satellites using an A-GPS server in its system. With the help of this data, the A-GPS server calculates the position and sends it back to the module.

In **MS-Based** mode, the module receives ephemeris, reference location, reference time and other optional assistance data from the A-GPS server. With the help of this data, the module receives signals from the visible satellites and calculates the position.

**Note:** if the required satellites visibility is not available, no GPS position is provided by the A-GPS receiver.

4.11.3. Get GPS position

The `gpsStart()` function allows the user to power on the GPS engine. Depending on the input parameter a different GPS mode is selected. The possibilities are:

- `Wasp4G::GPS_MS_ASSISTED`: Assisted GPS in MS-Assisted mode
- `Wasp4G::GPS_MS_BASED`: Assisted GPS in MS-Based mode
- `Wasp4G::GPS_MS_AUTONOMOUS`: Standalone or Autonomous GPS mode

Possible error codes for this function:

- 1: if error setting the reset mode
- 2: if error checking current GPS status
- 3: if error starting the GPS engine in standalone mode
- 4: if error setting NETWORK_UTRAN mode
- 5: if error defining the PDP context
- 6: if error setting authentication user ID
- 7: if error setting authentication password
- 8: if error setting socket configuration
- 9: if error setting quality of service
- 10: if error setting the SLP server
- 11: if error setting the supported SUPL version
- 12: if error updating terminal information
- 13: if error enabling unsolicited response
- 14: if error locking context for LCS use
- 15: if error enabling GNSS (or GLONASS)
- 16: if error in GPS Start Location Service Request
- 17: if error checking data connection
- 18: if incorrect GPS mode

The `waitForSignal()` function waits until GPS signal is received for valid data. The input parameter defines the timeout to wait for signal in millisecond units. If the function returns a correct answer means that the GPS attributes have been updated:

- Latitude
- Latitude indicator: North or South
- Longitude
- Longitude indicator: East or West
- Time
- Date
- Number of satellites
- HDOP: Horizontal Dilution of precision. If this value is less than 1 indicates the highest possible confidence level to be used for applications.

The `convert2Degrees()` function performs the conversion from the latitude and latitude variables given by the 4G module to degrees so it is more legible and intuitive. The input parameters must be the latitude/longitude and the corresponding indicator. The returning value is the converted value in degrees units.

The `gpsStop()` function powers down the GPS engine of the 4G module. It is possible to switch from a SUPL session to the autonomous GPS mode. Firstly, the GPS feature must be stopped, and then restart with the autonomous mode.

Example of GPS modes:

- [www.libelium.com/development/waspmote/examples/4g-17-agps-ms-assisted](http://www.libelium.com/development/waspmote/examples/4g-17-agps-ms-assisted)
- [www.libelium.com/development/waspmote/examples/4g-18-agps-ms-based](http://www.libelium.com/development/waspmote/examples/4g-18-agps-ms-based)
4.11.4. Indoor tracking using 4G and A-GPS mode (geolocation)

Assisted GPS, also known as A-GPS or AGPS, enhances the performance of standard GPS in devices connected to the cellular network. A-GPS improves the location performance of cell phones (and other connected devices) in two ways:

- By helping obtain a faster "time to first fix" (TTFF). A-GPS acquires and stores information about the location of satellites via the cellular network so the information does not need to be downloaded via satellite.
- By helping position a phone or mobile device when GPS signals are weak or not available such as indoor locations. GPS satellite signals may be impeded by tall buildings, and do not penetrate building interiors well. A-GPS uses proximity to cellular towers to calculate position when GPS signals are not available.

In this section, the execution of the A-GPS in MS-Based mode is shown. For this purpose, the corresponding example was used: www.libelium.com/development/waspmote/examples/4g-18-agps-ms-based

In this example, the GPS is started in MS-based mode. Once location is acquired, the GPS is stopped and started again in Standalone mode. In the following figures, it is possible to see how the GPS module gets its first position 41 seconds after switching on the 4G module. The green icon is the true device position. The red icon is the position the 4G module returns along different iterations. Finally, we can see how the module achieves a great location detection after 73 seconds.

Figure: First iteration (41 seconds after starting the 4G module). Distance error: 42 meters.
Figure: Second iteration (53 seconds after starting the 4G module): Distance error: 28 meters.

Figure: Third iteration (63 seconds after starting the 4G module): Distance error: 28 meters.
The location given by the A-GPS module may vary depending on the spot used to perform the test. The accuracy will improve when the device is situated in a high density or poor cellular antennas area. The location accuracy may vary from **10 to 100 meters** so a real test in each case is mandatory before implementing a final application.
4.12. e-mail management functions

4.12.1. Resetting e-mail parameters

The `emailReset()` function resets the current e-mail parameters in the memory of the module to the default ones. The values reset are:

- e-mail user name
- e-mail password
- e-mail sender address
- e-mail SMTP server

Example:

```
{   _4G.emailReset();
}
```

4.12.2. Setting the SMTP server

The `emailSetServerSMTP()` function sets the SMTP server address, used for e-mail sending. The function expects an input parameter for the SMTP server address. This parameter can be either:

- Any valid IP address in the format: xxx.xxx.xxx.xxx
- Any host name to be solved with a DNS query (factory default is the empty string "")

Example:

```
{   _4G.emailSetServerSMTP("smtp.mydomain.com");
}
```

4.12.3. Configuring SMTP parameters

The `emailConfigureSMTP()` function sets the parameters needed to the SMTP connection. The input parameters for this function are:

- **Security**: parameter indicating if the SSL encryption is enabled. The possibilities are:
  - `Wasp4G::EMAIL_NONSSL`
  - `Wasp4G::EMAIL_SSL`
- **Port**: SMTP port to contact (default 25). Range is from 1 to 65535.

Example:

```
{   _4G.emailConfigureSMTP(Wasp4G::EMAIL_NONSSL, 25);
}
```

*Note*: some servers support an obsolete implementation of SMTPS on port 465. The module only supports the standard implementation of SMTP over SSL/TLS described in RFC 3207. So do not use port 465 on servers with an obsolete implementation of SMTPS; the module will not work properly. Use instead port 25 or port 587.
4.12.4. Setting the sender parameters: address, username and password

The `emailSetSender()` function sets the sender parameters. The input parameters related are:

- **Address**: the address string to be used for sending the e-mail.
- **Username**: the username string to be used for sending the e-mail.
- **Password**: the authentication password to be used during the authentication step of the SMTP.

Example:

```c
char address[] = "me@email.box.com";
char user[] = "me@email.box.com";
char password[] = "myPassword"

_emailSetSender(address, user, password);
```

Possible error codes for this function:

- 1: if error setting the sender address
- 2: if error setting the sender user
- 3: if error setting the sender password

4.12.5. Saving e-mail parameters

The `emailSave()` function saves the actual e-mail parameters in the memory of the module. The values reset are:

- e-mail user name
- e-mail password
- e-mail sender address
- e-mail SMTP server

Example:

```c
_emailSave();
```

4.12.6. Sending an e-mail

The `emailSend()` function sends an e-mail message. The input parameters needed for this function are:

- **Address**: destination address.
- **Subject**: subject of the message. The maximum length is 100 characters.
- **Body**: the main text message of the e-mail.

Example:

```c
char address[] = "receiver@email.box.com"
char subject[] = "Subject of email"
char message[] = "This is an e-mail message from Waspmote"

_emailSend(address, subject, message);
```
Possible error codes for this function:

- 1: if error sending mail
- 2: if error waiting for module confirmation

Example of sending e-mail:
www.libelium.com/development/waspmote/examples/4g-19-send-email-smtp
5. Certifications

Libelium offers 2 types of sensor platforms, Waspmote OEM and Plug & Sense!:

- **Waspmote OEM** is intended to be used for research purposes or as part of a major product so it needs final certification on the client side. More info at: [http://www.libelium.com/products/waspmote/](http://www.libelium.com/products/waspmote/)
- **Plug & Sense!** is the line ready to be used out of the box. It includes market certifications. See below the specific list of regulations passed. More info at: [http://www.libelium.com/products/plug-sense/](http://www.libelium.com/products/plug-sense/)

Plug & Sense! 4G is certified for:

- CE (Europe)
- FCC (US)
- IC (Canada)
- ANATEL (Brazil)
- RCM (Australia)
- PTCRB (US)
- AT&T (US)

Figure : Certifications of the Plug & Sense! 4G product line

The 4G module is also used on Meshlium, which was certified too.

You can find all the certification documents at: [http://www.libelium.com/certifications](http://www.libelium.com/certifications)
6. Code examples and extended information

In the WaspMote Development section you can find complete examples:
www.libelium.com/development/waspmote/examples
7. API changelog

Keep track of the software changes on this link:
www.libelium.com/development/waspmote/documentation/changelog/#4g
8. Documentation changelog

From v7.1 to v7.2

- Added description about the new V2 radio versions

From v7.0 to v7.1

- Added a new section to show the user how to connect the module to waspmote