Waspmote

Technical Guide

libelium

waspmote

Document version: v4.6 - 09/2013

INDEX

1. Waspmote Kit

1.1. Box content

- 5 x Waspmote
- 1 x Waspmote Gateway
- 5 x Lithium Batteries
- 6 x USB cable
- 6 x XBee Radio
- 6 x XBee Antennas (2dBi / 5dBi)
- 0-5 x GPS + antenna
- 0-5 x GSM/GPRS + antenna
- 0-5 x 3G/GPRS + antenna
- 0-5 x SD Card

1.2. General and safety information

- In this section, the term "Waspmote" encompasses both the Waspmote device itself and its modules and sensor boards.
- Please read carefully through the document "General Conditions of Libelium Sale and Use".
- Do not let the electronic parts come into contact with any steel elements, to avoid injuries and burns.
- NEVER submerge the device in any liquid.
- Keep the device in a dry place and away from any liquids that might spill.
- Waspmote contains electronic components that are highly sensitive and can be accessed from outside; handle the device with great care and avoid hitting or scratching any of the surfaces.
- Check the product specifications section for the maximum allowed power voltage and amperage range and always use current transformers and batteries that work within that range. Libelium will not be responsible for any malfunctions caused by using the device with any batteries, power supplies or chargers other than those supplied by Libelium.
- Keep the device within the range of temperatures stated in the specifications section.
- Do not connect or power the device with damaged cables or batteries.
- Place the device in a location that can only be accessed by maintenance operatives (restricted area).
- In any case, keep children away from the device at all times.
- If there is an electrical failure, disconnect the main switch immediately and disconnect the battery or any other power supply that is being used.
- If using a car lighter as a power supply, be sure to respect the voltage and current levels specified in the "Power Supplies" section.
- When using a battery as the power supply, whether in combination with a solar panel or not, be sure to use the voltage and current levels specified in the "Power supplies" section.
- If a software or hardware failure occurs, consult the Libelium Web **Development section**
- Check that the frequencies and power levels of the radio communication modules and the integrated antennas are appropriate for the location in which you intend to use the device.
- The Waspmote device should be mounted in a protective enclosure, to protect it from environmental conditions such as light, dust, humidity or sudden changes in temperature. The board should not be definitively installed "as is", because the electronic components would be left exposed to the open-air and could become damaged. For a ready-to-install product, we advise our Plug & Sense! line.

DO NOT TRY TO RECHARGE THE NON-RECHARGEABLE BATTERY. IT MAY EXPLODE AND CAUSE INJURIES AND DESTROY THE EQUIPMENT. DEVICES WITH NON-RECHARGEABLE BATTERIES MUST BE PROGRAMMED THROUGH THE USB CABLE WITHOUT THE BATTERIES CONNECTED. PLEASE DOUBLE CHECK THIS CONDITION BEFORE CONNECTING THE USB. DO NOT CONNECT EITHER UNDER ANY CIRCUMSTANCE THE SOLAR PANEL TO A DEVICE WITH A NON-RECHARGEABLE BATTERY AS IT MAY EXPLODE AND CAUSE INJURIES AND DESTROY THE EQUIPMENT.

The document "General Conditions of Libelium Sale and Use" can be found at:

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

1.3. Conditions of use

General:

- Read the "General and Safety Information" section carefully and keep the manual for future reference.
- Read carefully the "General Conditions of Sale and Use of Libelium". This document can be found at: **http://www.libelium.com/development/waspmote/technical_service**. As specified in the Warranty document, the client has **7 days** from the day the order is received to detect any failure and report that to Libelium. Any other failure reported after these 7 days may not be considered under warranty.
- Use Waspmote in accordance with the electrical specifications and in the environments described in the "Electrical Data" section of this manual.
- Waspmote and its components and modules are supplied as electronic boards to be integrated within a final product. This product must have an enclosure to protect it from dust, humidity and other environmental interactions. If the product is to be used outside, the enclosure must have an IP-65 rating, at the minimum. For a ready-to-install product, we advise our Plug & Sense! line.
- Do not place Waspmote in contact with metallic surfaces; they could cause short-circuits which will permanently damage it.

Specific:

- Reset and ON/OFF button: Handle with care, do not force activation or use tools (pliers, screwdrivers, etc) to handle it.
- Battery: Only use the original lithium battery provided with Waspmote.
- Mini USB connection: Only use mini USB, mod. B, compatible cables.
- Solar panel connection: Only use the connector specified in the Power supplies section and always respect polarity.
- Lithium battery connection: Only use the connector specified in the Battery section and always respect polarity.
- Micro SD card connection: Only use 2GB maximum micro SD cards. HC cards are not compatible. There are many SD card models; any of them has defective blocks, which are ignored when using the Waspmote's SD library. However, when using OTA, those SD blocks cannot be avoided, so that the execution could crash. Libelium implements a special process to ensure the SD cards we provide will work fine with OTA. The only SD cards that Libelium can assure that work correctly with Waspmote are the SD cards we distribute officially.
- Micro SD card: Make sure Waspmote is switched off before inserting or removing the SD card. Otherwise, the SD card could be damaged.
- Micro SD card: Waspmote must not be switched off or reseted while there are ongoing read or write operations in the SD card. Otherwise, the SD card could be damaged and data could be lost.
- GSM/GPRS board connection: Only use the original Waspmote GSM/GPRS board.
- 3G/GPRS board connection: Only use the original Waspmote 3G/GPRS board.

- GPS board connection: Only use the original Waspmote GPS board.
- XBee module connection: Waspmote allows the connection of any module from the XBee family, respect polarity when connecting (see print).
- Antenna connections: Each of the antennas that can be connected to Waspmote (or to its GPS - GPRS boards) must be connected using the correct type of antenna and connector in each case, or using the correct adaptors.
- USB voltage adaptors: To power and charge the Waspmote battery, use only the original accessories: 220V AC – USB adaptor and 12V DC (car cigarette lighter) – USB adaptor

Usage and storage recommendations for the batteries:

The rechargeable, ion-lithium batteries, like the ones provided by Libelium (capacities of 2300 and 6600 mAh), have certain characteristics which must be taken into account:

- Charge the batteries for 24 hours before a deployment. The aim is to have the charge of the batteries at 100% of their capacity before a long period in which they must supply current, but it is not necessary to improve the performance.
- It is not advised to let the charge of the batteries go below 20% of capacity, since they suffer stress. Thus, it is not advised to wait for the battery to be at 0% to charge it.
- Any battery self-discharges: connected to Waspmote or not, the battery loses charges by itself.
- Maximum capacity loss: as the charge and discharge cycles happen, the maximum charge capacity is reduced.
- Batteries work better in cool environments: their performance is better at 10 ºC than at 30 ºC.
- At temperatures below 0 ºC, batteries can supply current (discharge), but the charge process cannot be done. In particular:
  - discharge range = [-10, 60] ºC
  - charge range = [0, 45] ºC

It is not reccommended to have the non-rechargeable batteries (13000, 26000, 52000 mA·h) connected to Waspmote when the USB cable is conneted too. The reason is, Waspmote will try to inject current in them if the USB is connected. This is dangerous for the good working of a non-rechargeable battery. It could be damaged or even damage Waspmote. That is to say, when you need to upload code to Waspmote via USB, disconnect the battery if it is non-rechargeable. That applies to Waspmote OEM, but not to the Plug & Sense! line, since its hardware is modified to avoid this.

Plug & Sense! line:

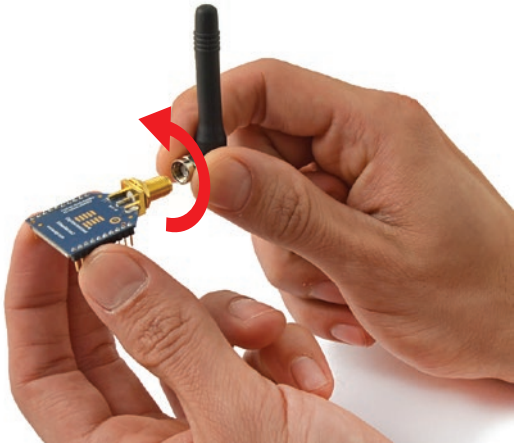
Libelium may provide the nodes with enclosures which are suitable to operate outdoors. The user, as final installer, must take great care when handling the product. We advise to read the Plug & Sense! Technical Guide to enlarge the life of your devices.

Remember that inappropriate use or handling of Waspmote will immediately invalidate the warranty.

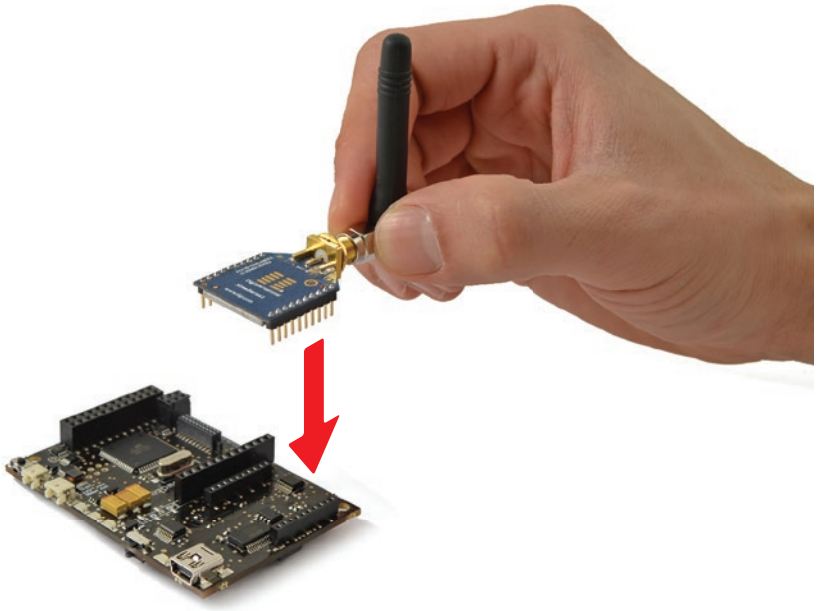
For further information, please visit **http://www.libelium.com/development/waspmote**

1.4. Assembly

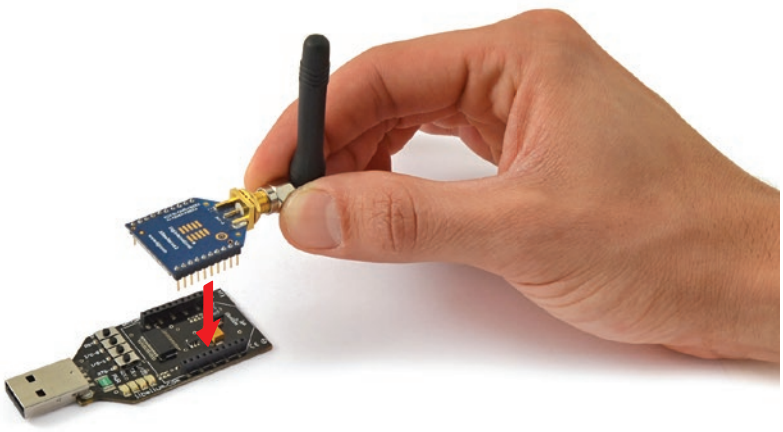
- **Connect the antenna to the XBee module**



- **Place the XBee module in Waspmote**

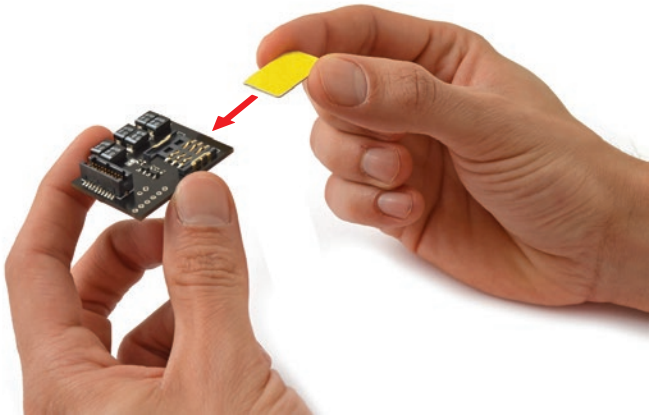


- **Place the XBee module in Waspmote Gateway**

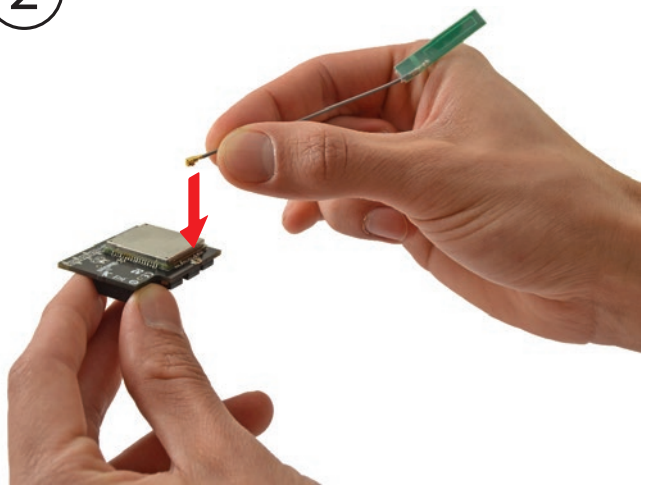


- **Connect the antenna in the GSM/GPRS module**

1

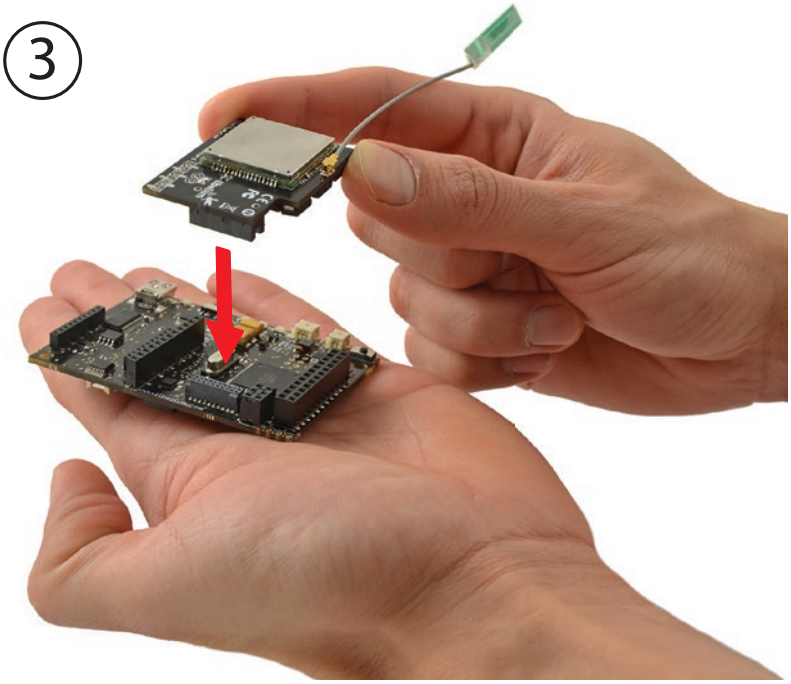


2

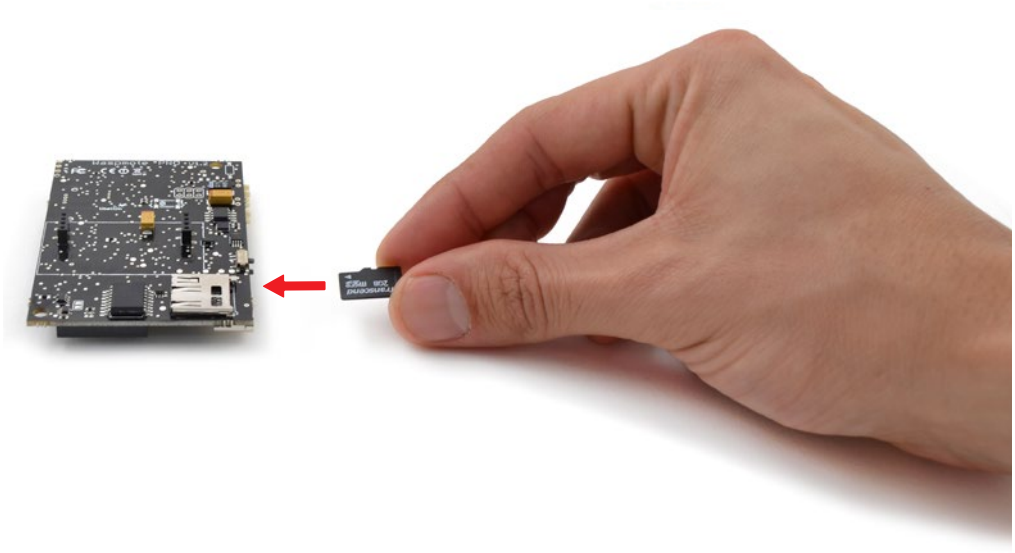


- **Place the GSM/GPRS module in Waspmote**

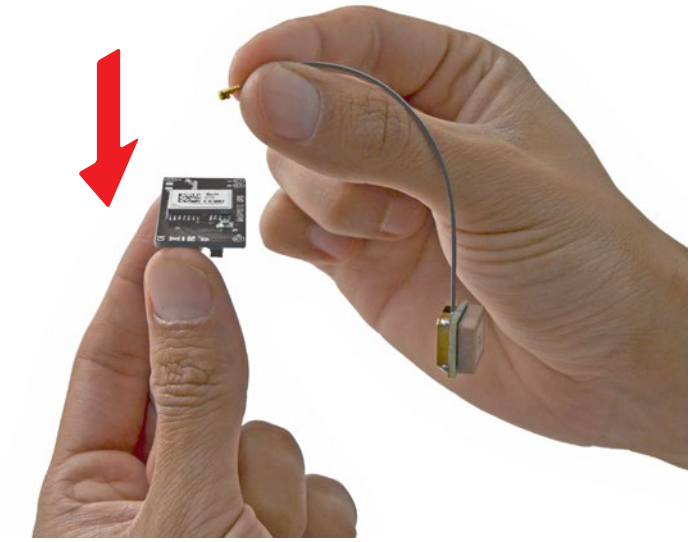
3



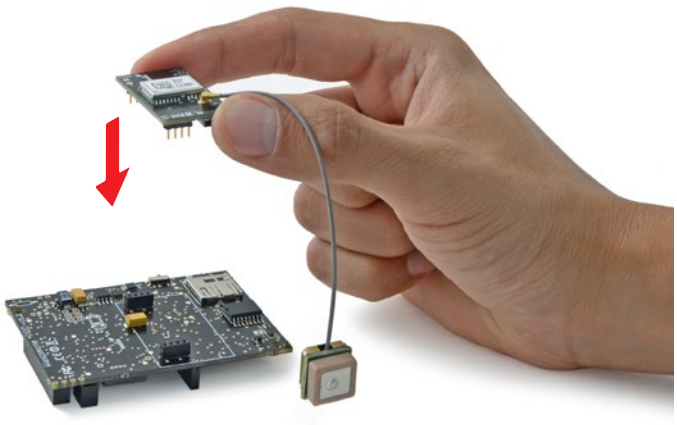
- **Place the SD card in Waspmote**



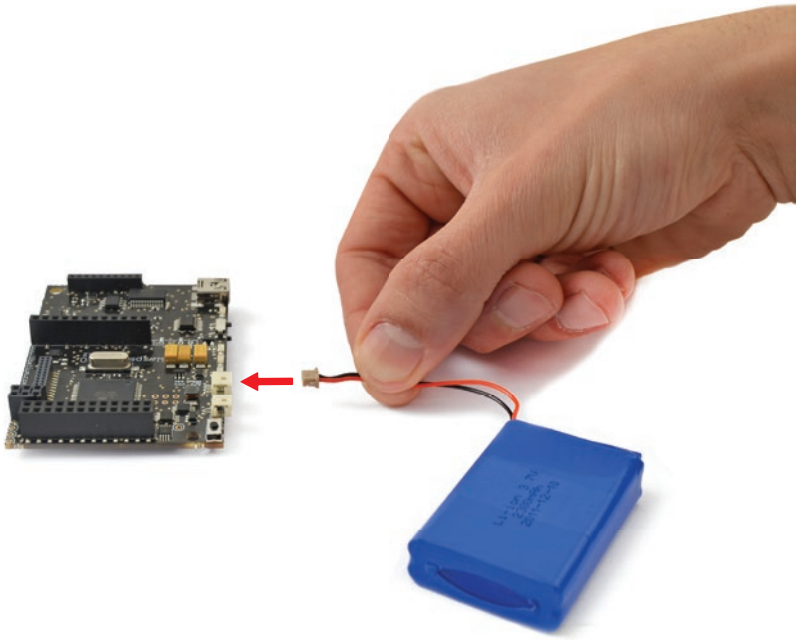
- **Connect the antenna in the GPS module**



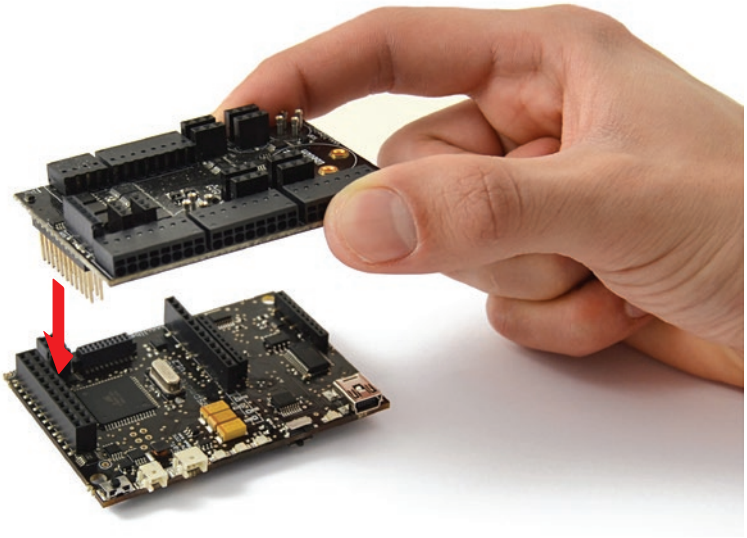
- **Place the GPS module in Waspmote**



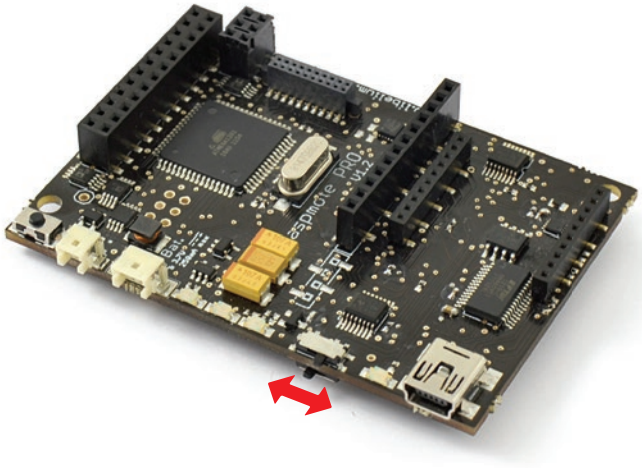
- **Connect the battery in Waspmote**



- **Connect the sensor board**

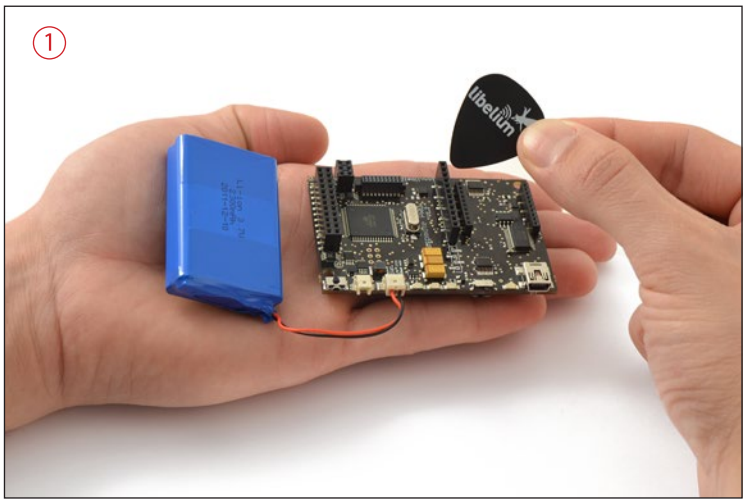


- **Switch it on**

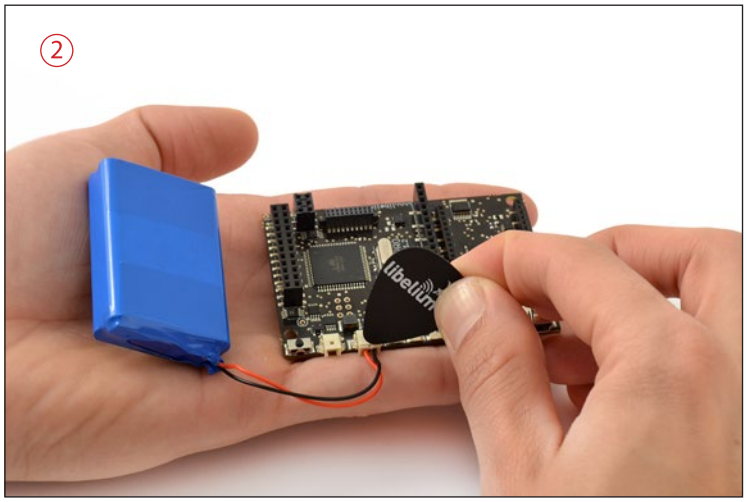


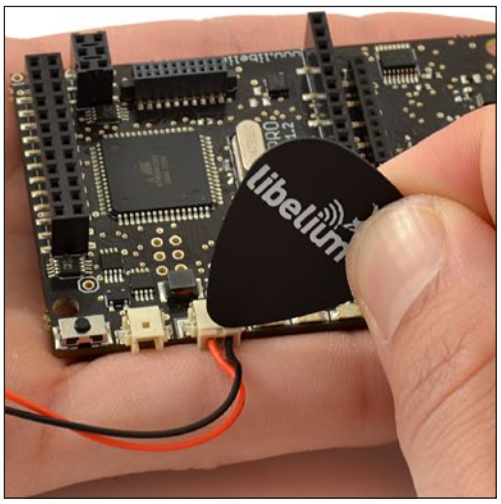
- **Waspmote battery disconnection**

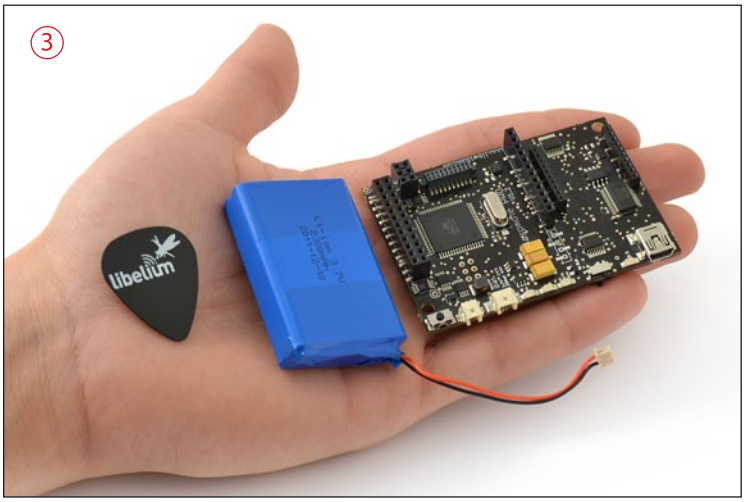
  Use the pick supplied by Libelium in order to disconnect Waspmote battery.



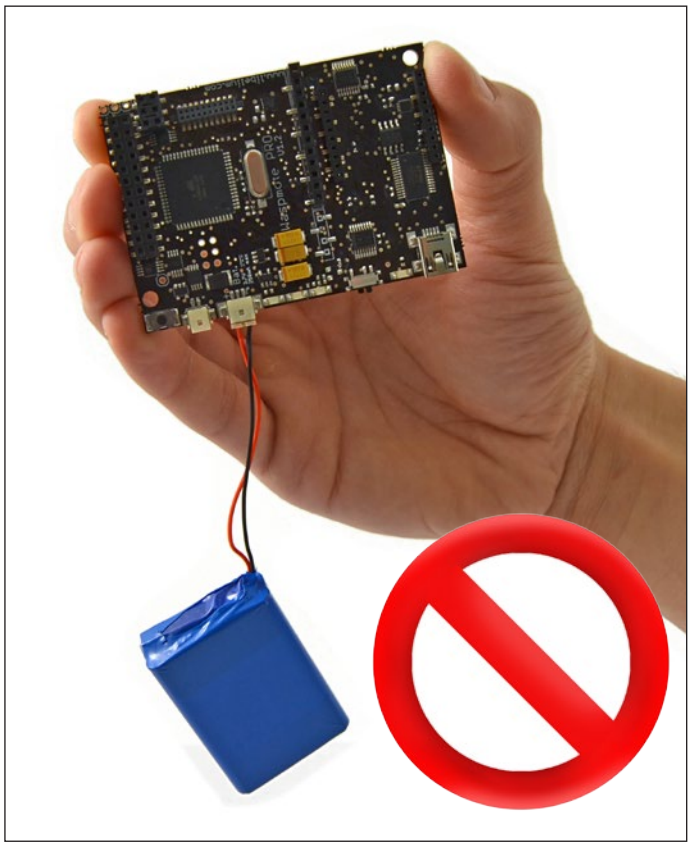
Insert the pick on the slot of the battery connector and pull straight out.
Do not pull the battery cables.



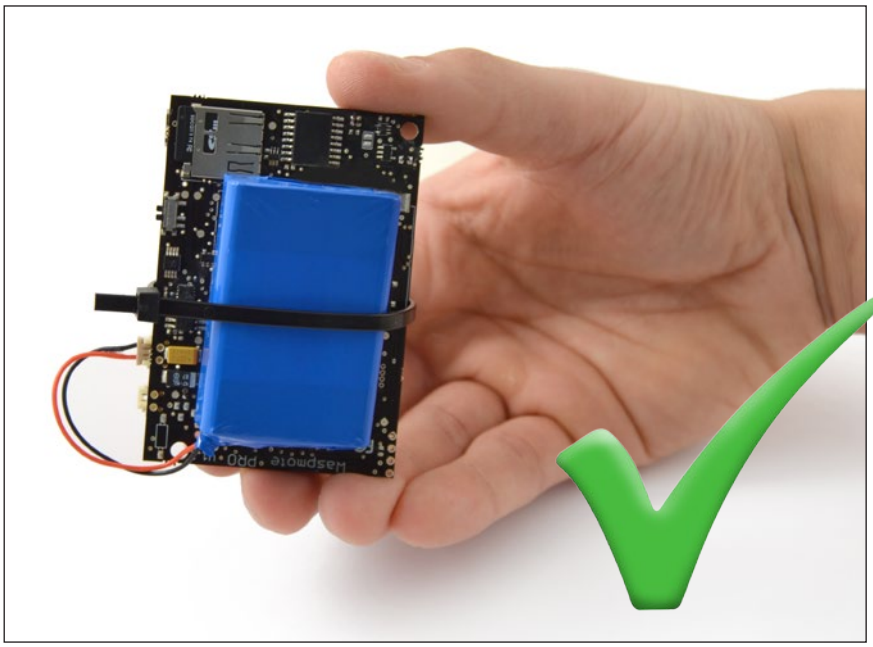




- **Battery handling instructions**
  In order to prevent from cable breaking, avoid leaving battery freely suspended.



Use a nylon clamp in order to attach battery to Waspmote.



2. Waspmote (v1.1) vs Waspmote PRO (v1.2)

2.1. Introduction: Understanding the changes from Waspmote (v1.1) to Waspmote PRO (v1.2)

Dear Developer,

The first version of Waspmote (v1.1) was released in 2009. Since then more than 2000 developers have been using the platform, posting suggestions and informing of possible improvements in our forum.

We have carefully listened to all of them and modified both the Waspmote API and Hardware in order to include all these improvements and suggestions. The result was launched in February 2013 with the name of Waspmote PRO (v.1.2).

In the present section you can find what changes have been made related to API and Hardware and what implications they have comparing with the previous version.

It is important to have in mind that not all the code of v1.1 will be compatible with v1.2 due to changes in the API libraries although the code translation will not be difficult and we will support you in the process. For this purpose, we have created a thread in our forum (**http://www.libelium.com/forum/viewforum.php?f=23**) where you will learn how to make your v1.1 codes work in Waspmote PRO (v1.2).

As a proof of concept, the new encapsulated line we have just launched "Waspmote Plug & Sense!" (**http://www.libelium.com/plug_&_sense**) has already Waspmote PRO (v.1.2) inside.

We want to specially thank all developers that have given us their feedback to improve the platform and encourage them to keep on doing the same with Waspmote PRO. We hope developers can take advantage of all these new features to perform easily any wireless sensor networks project.

Best regards.

David Gascón.

Libelium CTO.

2.2. Hardware Changes

- New Frequency Clock: 8 MHz → 14MHz.
- USB port at 115200 bps. It is faster to upload code now (x2 or x3 quicker).
- New accelerometer higher range (±8g).
- All the jumpers have been removed. There is no risk of broken jumpers.
- The coin battery is removed. Now RTC is always powered by the main battery, with much bigger load.
- Added a unique Serial Number ID chip on each board. This ID can be used to differentiate natively each of the nodes. This value can be read with an API function.
- The battery socket position has changed, in Waspmote PRO it is close to the solar panel socket. Now it is easier to connect the battery and there is no risk of broken battery cables.
- The solar panel socket orientation has changed, in Waspmote PRO it is horizontal. Now it is easier to connect the solar panel cable too.
- The hibernate jumper has been removed, in Waspmote PRO there is a manual switch to enable the hibernate mode.
- Added the SPI port connection in one of the sensor sockets.
- Added specific default sockets for "basic sensors": temperature, humidity and light sensor (LDR). No need of using a Sensor Board to integrate those sensors.
- Waspmote PRO can be programmed with the XBee connected. This reduces the process of uploading code and avoids the risk of broken sockets.
- The XBee RSSI indicator leds have been removed. The RSSI signal can be read now with an API function.
- A new interrupt is added, XBee can wake Waspmote PRO up when using the XBee cyclic sleep mode. That is great to enable advanced sleep strategies.

Sensor Boards Compatibility

Sensor Board	Compatible with Waspmote v11	Compatible with Waspmote PRO
Gases Sensor Board	Yes	Yes
Gases Sensor Board v2.0	Yes	Yes
Events Sensor Board	Yes	No
Events Sensor Board v2.0	Yes	Yes
Smart Cities Board	Yes	No
Smart Cities Board 1.4	Yes	Yes
Smart Parking Board	Yes	No
Smart Parking Board (purchased with Waspmote PRO)	Yes	Yes
Smart Metering Board	Yes	No
Smart Metering Board v2.0	Yes	Yes
Agriculture Sensor Board	Yes	No
Agriculture Sensor Board PRO	Yes	No
Agriculture Sensor Board v2.0	Yes	Yes
Agriculture Sensor Board v2.0 PRO	Yes	Yes
Radiation Sensor Board	Yes	Yes
Prototyping Sensor Board	Yes	No
Prototyping Sensor Board v2.0	Yes	Yes

2.3. Specs Comparative Table

		Waspmote V11	Waspmote PRO
General data	Microcontroller	ATmega 1281	ATmega 1281
	Frequency	8 MHz	14.7456 MHz
	SRAM	8 kB	8 kB
	EEPROM	4 kB	4 kB (1kB reserved)
	FLASH	128 kB	128 kB
	SD Card	2 GB	2 GB
	Weight	20 g	20 g
	Dimensions	73.5 x 51 x 13 mm	73.5 x 51 x 13 mm
	Temperature range	-20ºC + 65ºC	-10ºC + 65ºC
	Clock	RTC (32 kHz)	RTC (32 kHz)
Consumption	ON	9 mA	15 mA
	Sleep	62 uA	55 uA
	Deep Sleep	62 uA	55 uA
	Hibernate	0.7 uA	0.06 uA
Inputs/Outputs		7 Analog Inputs 8 Digital I/O 2 UART 1 I2C 1 USB	7 Analog Inputs 8 Digital I/O 2 UART 1 I2C 1 SPI 1 USB Specific Default Socket for "basic sensors" Temperature, Humidity, Light (LDR)
Electrical data	Battery voltage	3.3V – 4.2V	3.3V – 4.2V
	USB charging	5V – 100 mA	5V – 100 mA
	Solar panel charging	6 – 12 V – 280 mA	6 – 12 V – 280 mA
	Auxiliary button/coin battery voltage (RTC)	3V	Not needed (main battery powers RTC in all cases)
Built-in sensors on the board	Temperature	-40ºC, +85ºC Accuracy 0.25ºC	-40ºC, +85ºC Accuracy 0.25ºC
	Accelerometer	±2g (1024 LSB/g) | ±6g (340 LSB/g) 40 Hz / 160 Hz / 640 Hz / 2560 Hz	±2g/±4g/±8g Low power: 0.5 Hz / 1 Hz / 2 Hz / 5 Hz / 10 Hz Normal mode: 50 Hz / 100 Hz / 400 Hz / 1000 Hz

2.4. API Changes

API file system

- New API file system in order to reduce program sizes and avoid wasting RAM memory. Now it is possible to find a core library and divided libraries for each module. It will be necessary to include each library when using it. For example, XBee-802.15.4 library would be included like this:

```
#include <WaspXBee802.h>
```

XBee

- WaspXBee class is deleted. This frees memory. There are not any XBee calls.
- XBee initialization has changed. `init` function is now called inside the API. For example, XBee-802.15.4 initialization changes:
  - Before:

```
// Inits the XBee 802.15.4 library
xbee802.init(XBEE_802_15_4, FREQ2_4G, NORMAL);

// Powers XBee
 xbee802.ON();
```

  - Now:

```
// Powers XBee
 xbee802.ON();
```

- `WaspXBeeCore::ON` function has been changed in order to support SOCKET selection. Before, this was done by init function. Selecting UART1 changes in XBee-802.15.4:
  - Before:

```
// Init the XBee 802.15.4 library in UART1
xbee802.init(XBEE_802_15_4, FREQ2_4G, NORMAL, UART1);

// Powers XBee
 xbee802.ON();
```

  - Now:

```
// Powers XBee in UART1 (SOCKET1)
 xbee802.ON(SOCKET1);
```

- Sending process has been simplified. `setOriginParams` function has been deleted. Also, some class attributes have been deleted too. For example, XBee-802.15.4 sending changes:
  - Before:

```
// Set params to send
paq_sent=(packetXBee*) calloc(1, sizeof(packetXBee));
paq_sent->mode=BROADCAST;
paq_sent->MY_known=0;
paq_sent->packetID=0x52;
paq_sent->opt=0;
xbee802.hops=0;
xbee802.setOriginParams(paq_sent, “5678”, MY_TYPE);
xbee802.setDestinationParams(paq_sent, mac_address, data, MAC_TYPE, DATA_ABSOLUTE);
xbee802.sendXBee(paq_sent);
```

- Now:

```
// set parameters to packet:
packet=(packetXBee*) calloc(1, sizeof(packetXBee)); // Memory allocation
 packet->mode=UNICAST; // Choose transmission mode: UNICAST or BROADCAST

 // set destination XBee parameters to packet
 xbee802.setDestinationParams( packet, MAC_ADDRESS, data, MAC_TYPE);

 // send XBee packet
 xbee802.sendXBee(packet);
```

- New `setDestinationParams` function prototype allows to set an array of uint8_t as data field. It is necessary to indicate the length of this array:

```
uint8_t data[5] = {0x00, 0x01, 0x54, 0x76, 0x23};
xbee802.setDestinationParams(packet, MAC, data, 5, MAC_TYPE);
```

- `WaspXBeeCore::nodeSearch` function has been changed, now a pointer to an array permits to store the destination address, which might be network address for XBee-802.15.4 or MAC address for the rest of the protocols:

  - Before:

```
// enable encryption mode
packetXBee* paq_sent;
xbee802.nodeSearch("forrestNode-01", 14, paq_sent); // '14' is string length
```

  - Now:

```
uint8_t naD[2];
xbee802.nodeSearch("forestNode-01", naD);
```

- `encryptionMode` function is renamed to `setEncryptionMode`. More intuitive than before:

  - Before:

```
// enable encryption mode
 xbee802.encryptionMode(1);
```

  - Now:

```
// enable encryption mode
 xbee802.setEncryptionMode(1);
```

- `getEncryptionMode` function is created in order to read the encryption mode
- `WaspXBee900` class has been created so as to use the XBee-900 modules. WaspXBeeDM is still used for XBee-900 Digimesh modules. As the other XBee modules it will be necessary to include this class:

```
#include <WaspXBee900.h>
```

- Regarding XBee-ZigBee protocol, `getExtendedPAN` function is now called `getOperating64PAN` in order to clearify the function goal. Thus, `extendedPAN` attribute is now called `operating64PAN`:

- Before:

```
// Get Operating Extended PAN ID
xbeeZB.getExtendedPAN();

xbeeZB.extendedPAN[0-7] → stores the 64-bit Operating Extended PAN ID
```

- Now:

```
// Get Operating Extended PAN ID
xbeeZB.getOperating64PAN();

xbeeZB.operating64PAN[0-7] → stores the 64-bit Operating Extended PAN ID
```

Regarding Xbee-ZigBee protocol, `getOperatingPAN` function is now called `getOperating16PAN`, in order to clearify the function goal. Thus, `operatingPAN` is now called `operating16PAN`:

- Before:

```
// Get Operating PAN ID
xbeeZB.getOperatingPAN();

xbeeZB.operatingPAN[0-1] → stores the 16-bit Operating PAN ID
```

- Now:

```
// Get Operating PAN ID
xbeeZB.getOperating16PAN();

xbeeZB.operating16PAN[0-1] → stores the 16-bit Operating PAN ID
```

- `send` function has been deleted, because both packet fragmentation and API header have been deleted from libraries. Before, a large packet was fragmented in to different short packets in order to fit the maximum payload. Now, the user must be careful with the packet size because it will be truncated to its maximum payload.

SD card

- There is a constraint in file names and directory names. `mkdir` and `create` are functions which now create files and directories with SFN 8.3 (short filename format):

- Before:

```
// create file
SD.create("Test_File_1.txt")
```

- Now:

```
// create a 8.3 SFN file
SD.create("FILENAME.TXT")
```

- `ls` now prints all the info related to files and directories. No "trash" files are printed. USB class is used inside `ls` function, so there is no need of calling it:

- Before:

```
// list directory
USB.println(SD.ls());
```

- Now:

```
// list directory
SD.ls();
```

- The following functions do not exist any more:

```
SD.getDiskFree();
SD.getAttributes();
SD.delFile();
SD.delDir();
```

- `del` function now only deletes files. There is a new function which permits to delete empty directories:

- Before:

```
// delete an empty directory called “Folder”
SD.del(“Folder”);
```

- Now:

```
// delete an empty directory called “FOLDER”
SD.rmDir(“FOLDER”);
```

- There is a new function which allows the user to delete folders which contain files and subdirectories:

```
// delete folder and all contained files
SD.rmRfDir(“FOLDER”);
```

- New `writeSD` function prototype which permits to write an array of bytes (uint8_t) always the length is indicated as a parameter:

```
// define hexadecimal data
uint8_t data[10]={0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0xAA,0xAA};

// it writes hexadecimal data indicating the length (length=3 in this case) to write
SD.writeSD(filename, data, 0, 3);
```

USB

- `begin` / `close` functions change their names to `ON` / `OFF` in order to follow the same directives as other classes:

- Before:

```
// init USB port
USB.begin();
```

- Now:

```
// init USB port

USB.ON();
```

- It is possible to print debug messages from Flash memory in order to save RAM memory. This new function is called as:

```
USB.println(F("string in Flash memory"));
```

- New `USB.printf` function which permits to print formatted strings directly:

```
USB.printf("ACC x:%d,y:%d,z:%d - temp:%d - bat: %d%c%c%c",
                            ACC.getX(),
                            ACC.getY(),ACC.getZ(),
                            (int)RTC.getTemperature(),
                            PWR.getBatteryLevel(),
                            '%',
                            '\r',
                            '\n');
```

New `USB.printHex` function which allows to print bytes in hexadecimal format with two ciphers: %2X. Thus, an hexadecimal byte will be printed as two capital letters:

```
// print bytes in hexadecimal format
uint8_t data=0x01;
USB.printHex(data);
```

EXPANSION BOARD

- `SOCKET0` / `SOCKET1` are used as constants in order to select between both sockets in Waspmote. This can be used in several communication modules: XBee, Bluetooth, WiF, etc.

SENSOR BOARDS

- `ON` and `OFF` functions have been implemented to connect and disconnect the boards, though setBoardMode functions are still available.

GPS

- New `waitForSignal` function which tries to connect to satellites for a specific time. This function returns true or false depending on the result. This function is used as follows:

```
// wait for GPS signal for specific time
status=GPS.waitForSignal(GPS_TIMEOUT);
```

- New `convert2Degrees` function which provides latitude and longitude in degrees. This is useful for search websites:

```
GPS.convert2Degrees(GPS.latitude, GPS.NS_indicator);
GPS.convert2Degrees(GPS.longitude, GPS.EW_indicator);
```

EEPROM

It is only possible to write EEPROM addresses from 1024 to 4095. The first 1kB of EEPROM memory is reserved:

```
// Writing in the EEPROM
Utils.writeEEPROM(address,value);
```

3. Waspmote Plug & Sense! - Encapsulated Line

Waspmote is the original line in which developers have a total control over the hardware device. You can physically access to the board and connect new sensors or even embed it in your own products as an electronic sensor device.

The new Waspmote Plug & Sense! line allows developers to forget about electronics and focus on services and applications. Now you can deploy wireless sensor networks in a easy and scalable way ensuring minimum maintenance costs. The new platform consists of a robust waterproof enclosure with specific external sockets to connect the sensors, the solar panel, the antenna and even the USB cable in order to reprogram the node. It has been specially designed to be scalable, easy to deploy and maintain.

Note: *For a complete reference guide download the "Waspmote Plug & Sense! Technical Guide" in the* ***Development section*** *of the* ***Libelium website*****.**

3.1. Quick Overview

3.1.1. Features

- Robust waterproof IP65 enclosure
- Add or change a sensor probe in seconds
- Solar powered with internal and external panel options
- Radios available: Zigbee, 802.15.4, WiFi, 868MHz, 900MHz and 3G/GPRS
- Over the air programming (OTAP) of multiple nodes at once
- Special holders and brackets ready for installation in street lights and building fronts
- Graphical and intuitive programming interface
- External, contactless reset with magnet

3.1.2. Sensor Probes

Sensor probes can be easily attached by just screwing them into the bottom sockets. This allows you to add new sensing capabilities to existing networks just in minutes. In the same way, sensor probes may be easily replaced in order to ensure the lowest maintenance cost of the sensor network.

Figure 1: Connecting a sensor probe to Waspmote Plug & Sense!

3.1.3. Solar Powered

Battery can be recharged using the internal or external solar panel options.
The external solar panel is mounted on a 45º holder which ensures the maximum performance of each outdoor installation.

Figure 2: Waspmote Plug & Sense! powered by an external solar panel

For the internal option, the solar panel is embedded on the front of the enclosure, perfect for use where space is a major challenge.

Figure 3: Internal solar panel

Figure 4: Waspmote Plug & Sense! powered by an internal solar panel

3.1.4. Programming the Nodes

Waspmote Plug & Sense! can be reprogrammed in two ways:

The basic programming is done from the USB port. Just connect the USB to the specific external socket and then to the computer to upload the new firmware.

Figure 5: Programming a node

Over the Air Programming is also possible once the node has been installed. With this technique you can reprogram wirelessly one or more Waspmote sensor nodes at the same time by using a laptop and the Waspmote Gateway.

Figure 6: Typical OTA process

3.1.5. Radio Interfaces

Model	Protocol	Frequency	txPower	Sensitivity	Range *
XBee-802.15.4	802.15.4	2.4GHz	1mW	-92dB	500m
XBee-802.15.4-Pro	802.15.4	2.4GHz	100mW	-100dBm	7000m
XBee-ZB	ZigBee-Pro	2.4GHz	2mW	-96dBm	500m
XBee-ZB-Pro	ZigBee-Pro	2.4GHz	50mW	-102dBm	7000m
XBee-868	RF	868MHz	315mW	-112dBm	12km
XBee-900	RF	900MHz	50mW	-100dBm	10Km

** Line of sight and Fresnel zone clearance with 5dBi dipole antenna*

3.1.6. Program in minutes

In order to program the nodes an intuitive graphic interface has been developed. Developers just need to fill a web form in order to obtain the complete source code for the sensor nodes. This means the complete program for an specific application can be generated just in minutes. Check the Code Generator to see how easy it is at:
http://www.libelium.com/development/plug_&_sense/sdk_and_applications/code_generator

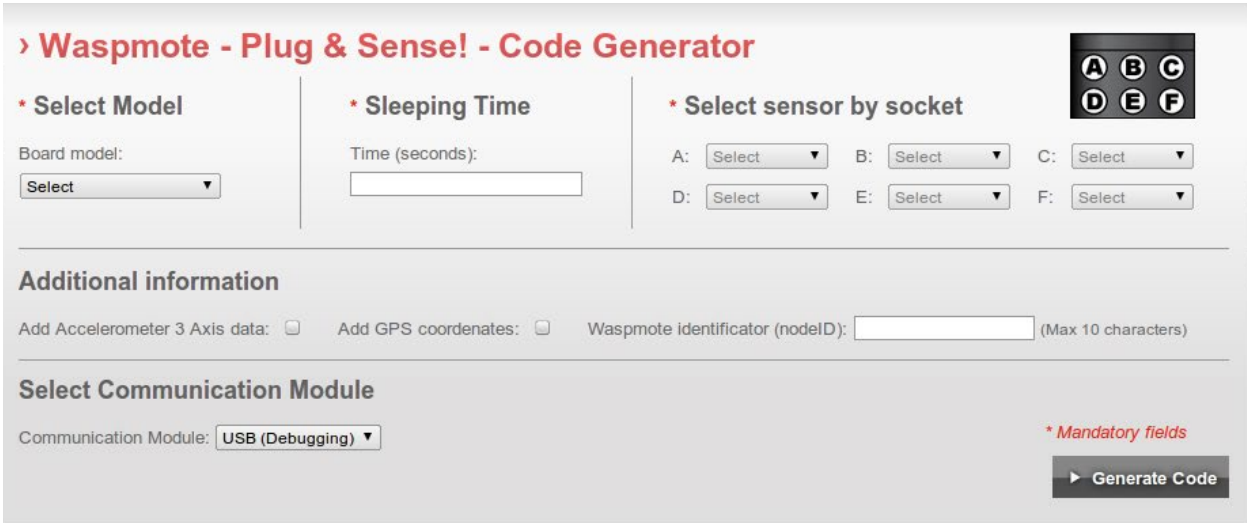


Figure 7: Code Generator

3.1.7. Data to the Cloud

The Sensor data gathered by the Waspmote Plug & Sense! nodes is sent to the Cloud by **Meshlium**, the Gateway router specially designed to connect Waspmote sensor networks to the Internet via Ethernet, WiFi and 3G interfaces.

Figure 8: Meshlium

3.1.8. Models

There are some defined configurations of Waspmote Plug & Sense! depending on which sensors are going to be used. Waspmote Plug & Sense! configurations allows connecting up to six sensor probes at the same time.

Each model takes a different conditioning circuit to enable the sensor integration. For this reason each model allows to connect just its specific sensors.

This section describes each model configuration in detail, showing the sensors which can be used in each case and how to connect them to Waspmote. In many cases, the sensor sockets accept the connection of more than one sensor probe. See the compatibility table for each model configuration to choose the best probe combination for the application.

It is very important to remark that each socket is designed only for one specific sensor, so **they are not interchangeable**. Always be sure you connected probes in the right socket, otherwise they can be damaged.

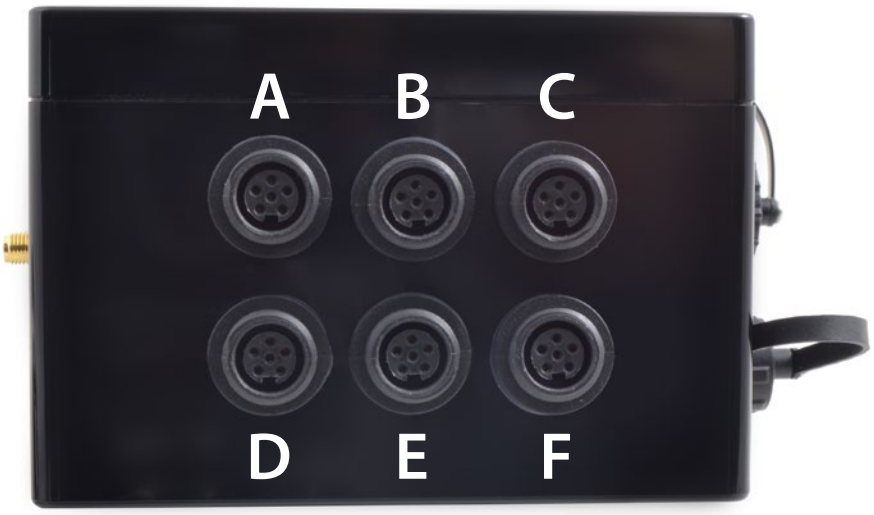


Figure 9: Identification of sensor sockets

3.1.8.1. Smart Environment

Smart Environment model is designed to monitor environmental parameters such as temperature, humidity, atmospheric pressure and some types of gases. The main applications for this Waspmote Plug & Sense! configuration are city pollution measurement, emissions from farms and hatcheries, control of chemical and industrial processes, forest fires, etc. Go to the Applications section in the **Libelium website** for a complete list of services.

Figure 10: Smart Environment Waspmote Plug & Sense! model

Sensor sockets are configured as shown in the figure below.

Sensor Socket	Sensor probes allowed for each sensor socket	
	Parameter	**Reference**
A	Temperature	9203
	Carbon monoxide - CO	9229
	Methane - CH_4	9232
	Ammonia – NH_3	9233
	Liquid Petroleum Gases: H_2, CH_4, ethanol, isobutene.	9234
	Air pollutants 1: $C_6H_5CH_3$, H_2S, CH_3CH_2OH, NH_3, H_2	9235
	Air pollutants 2: C_4H_{10}, CH_3CH_2OH, H_2, CO, CH_4	9236
	Alcohol derivates: CH_3CH_2OH, H_2, C_4H_{10}, CO, CH_4	9237
B	Humidity	9204
	Atmospheric pressure	9250
C	Carbon dioxide - CO_2	9230
D	Nitrogen dioxide - NO_2	9238
E	Ozone - O_3	9258
	Hydrocarbons - VOC	9201
	Oxygen - O_2	9231
F	Carbon monoxide - CO	9229
	Methane - CH_4	9232
	Ammonia – NH_3	9233
	Liquid Petroleum Gases: H_2, CH_4, ethanol, isobutene.	9234
	Air pollutants 1: $C_6H_5CH_3$, H_2S, CH_3CH_2OH, NH_3, H_2	9235
	Air pollutants 2: C_4H_{10}, CH_3CH_2OH, H_2, CO, CH_4	9236
	Alcohol derivates: CH_3CH_2OH, H_2, C_4H_{10}, CO, CH_4	9237

Figure 11: Sensor sockets configuration for Smart Environment model

Note: *For more technical information about each sensor probe go to the* ***Development section*** *in Libelium website.*

Note: *In Fall 2013, Libelium expects to launch the new line of calibrated gas sensors. This line will improve the accuracy of gas metering.*

3.1.8.2. Smart Security

The main applications for this Waspmote Plug & Sense! configuration are perimeter access control, liquid presence detection and doors and windows openings.

Figure 12: Smart Security Waspmote Plug & Sense! model

Note: *The probes attached in this photo could not match the final location. See next table for the correct configuration.*

Sensor Socket	Sensor probes allowed for each sensor socket	
	Parameter	**Reference**
A	Temperature + Humidity (Sensirion)	9247
B	Liquid flow	9296, 9297, 9298
C	Presence - PIR	9212
D	Luminosity (LDR)	9205
	Liquid level	9239, 9240, 9242
	Liquid presence	9243
	Hall effect	9207
E	Luminosity (LDR)	9205
	Liquid level	9239, 9240, 9242
	Liquid presence	9243
	Hall effect	9207
F	Luminosity (LDR)	9205
	Liquid level	9239, 9240, 9242
	Liquid presence	9243
	Hall effect	9207

Figure 13: Sensor sockets configuration for Smart Security model

As we see in the figure below, thanks to the directionable probe, the presence sensor probe (PIR) may be placed in different positions. The sensor can be focused directly to the point we want.

Figure 14: Configurations of the Presence sensor probe (PIR)

Note: *For more technical information about each sensor probe go to the* ***Development section*** *in Libelium website.*

3.1.8.3. Smart Metering

The main applications for this Waspmote Plug & Sense! model are energy measurement, water consumption, pipe leakage detection, liquid storage management, tanks and silos level control, supplies control in manufacturing, industrial automation, agricultural irrigation, etc. Go to the Applications section in the **Libelium website** for a complete list of services.

Figure 15: Smart Metering Waspmote Plug & Sense! model

Sensor sockets are configured as shown in the figure below.

Sensor Socket	Sensor probes allowed for each sensor socket	
	Parameter	**Reference**
A	Temperature	9203
	Soil temperature	86949*
B	Humidity	9204
C	Ultrasound (distance measurement)	9246
	Liquid flow	9296, 9297, 9298
D	Current sensor	9266
E	Ultrasound (distance measurement)	9246
	Liquid flow	9296, 9297, 9298
F	Luminosity (LDR)	9205

* Ask Libelium **Sales Department** for more information.

Figure 16: Sensor sockets configuration for Smart Metering model

As we see in the figure below, thanks to the directionable probe, the ultrasound sensor probe may be placed in different positions. The sensor can be focused directly to the point we want to measure.

Figure 17: Configurations of the ultrasound sensor probe

Note: *For more technical information about each sensor probe go to the* ***Development section*** *in Libelium website.*

3.1.8.4. Smart Cities

The main applications for this Waspmote Plug & Sense! model are noise maps (monitor in real time the acoustic levels in the streets of a city), air quality, waste management, structural health, smart lighting, etc. Refer to **Libelium website** for more information.

Figure 18: Smart Cities Waspmote Plug & Sense! model

Sensor sockets are configured as shown in the figure below.

Sensor Socket	Sensor probes allowed for each sensor socket	
	Parameter	**Reference**
A	Temperature	9203
	Soil temperature	86949*
	Ultrasound (distance measurement)	9246
B	Humidity	9204
	Ultrasound (distance measurement)	9246
C	Luminosity (LDR)	9205
D	Noise sensor	9259
E	Dust sensor	9320
F	Linear displacement	9319

* Ask Libelium **Sales Department** for more information.

Figure 19: Sensor sockets configuration for Smart Cities model

As we see in the figure below, thanks to the directionable probe, the ultrasound sensor probe may be placed in different positions. The sensor can be focused directly to the point we want to measure.

Figure 20: Configurations of the ultrasound sensor probe

Note: *For more technical information about each sensor probe go to the* ***Development section*** *in Libelium website.*

3.1.8.5. Smart Parking

Smart Parking allows to detect available parking spots by placing the node under the pavement. It works with a magnetic sensor which detects when a vehicle is present or not. Waspmote Plug & Sense! can act as a repeater for a Smart Parking node.

Figure 21: Smart Parking enclosure

Sensor sockets are no used for this model.

There are specific documents for parking applications at **Libelium website**. Refer to Smart Parking Technical guide to see typical applications for this model and how to make a good installation.

3.1.8.6. Smart Agriculture

The Smart Agriculture models allow to monitor multiple environmental parameters involving a wide range of applications. It has been provided with sensors for air and soil temperature and humidity (Sensirion), solar visible radiation, wind speed and direction, rainfall, atmospheric pressure, etc.

The main applications for this Waspmote Plug & Sense! model are precision agriculture, irrigation systems, greenhouses, weather stations, etc. Refer to **Libelium website** for more information.

Two variants are possible for this model, normal and PRO. Next section describes each configuration in detail.

Figure 22: Smart Agriculture Waspmote Plug & Sense! model

Normal

Sensor sockets are configured as shown in the figure below.

Sensor Socket	Sensor probes allowed for each sensor socket	
	Parameter	**Reference**
A	Humidity + Temperature (Sensirion)	9247
B	Atmospheric pressure	9250
C	Soil temperature	86949*
	Soil moisture	9248
D	Weathermeters + pluviometer	9256
E	Soil moisture	9248
F	Leaf wetness	9249
	Soil moisture	9248

* Ask Libelium **Sales Department** for more information.

Figure 23: Sensor sockets configuration for Smart Agriculture model

Note: *For more technical information about each sensor probe go to the* ***Development section*** *in Libelium website.*

PRO

Sensor sockets are configured as shown in the figure below.

Sensor Socket	Sensor probes allowed for each sensor socket	
	Parameter	**Reference**
A	Humidity + Temperature (Sensirion)	9247
B	Soil temperature	9255
C	Solar radiation	9251, 9257
D	Soil temperature	86949*
	Soil moisture	9248
E	Dendrometers	9252, 9253, 9254
	Soil moisture	9248
F	Lear wetness	9249
	Soil moisture	9248

* Ask Libelium **Sales Department** for more information.

Figure 24: Sensor sockets configuration for Smart Agriculture PRO model

Note: *For more technical information about each sensor probe go to the* ***Development section*** *in Libelium website.*

3.1.8.7. Ambient Control

This model is designed to monitor main environment parameters in an easy way. Only three sensor probes are allowed for this model, as shown in next table.

Figure 25: Ambient Control Waspmote Plug & Sense! model

Sensor sockets are configured as it is shown in figure below.

Sensor Socket	Sensor probes allowed for each sensor socket	
	Parameter	**Reference**
A	Humidity + Temperature (Sensirion)	9247
B	Luminosity (LDR)	9205
C	Luminosity (Luxes accuracy)	9325
D	Not used	
E	Not used	
F	Not used	

Figure 26: Sensor sockets configuration for Ambient Control model

As we see in the figure below, thanks to the directionable probe, the Luminosity sensor (Luxes accuracy) probe may be placed in different positions. The sensor can be focused directly to the light source we want to measure.

Figure 27: Configurations of the Luminosity sensor probe (luxes accuracy)

Note: *For more technical information about each sensor probe go to the* ***Development section*** *in Libelium website.*

3.1.8.8. Radiation Control

The main application for this Waspmote Plug & Sense! configuration is to measure radiation levels using a Geiger sensor. For this model, the Geiger tube is already included inside Waspmote, so the user does not have to connect any sensor probe to the enclosure. The rest of the other sensor sockets are not used.

Figure 28: Radiation Control Waspmote Plug & Sense! model

Sensor sockets are not used for this model.

> ***Note:*** *For more technical information about each sensor probe go to the* ***Development section*** *in Libelium website.*

4. Hardware

4.1. Modular Architecture

Waspmote is based on a modular architecture. The idea is to integrate only the modules needed in each device. These modules can be changed and expanded according to needs.

The modules available for integration in Waspmote are categorized in:

- ZigBee/802.15.4 modules (2.4GHz, 868MHz, 900MHz). Low and high power.
- GSM/GPRS Module (Quadband: 850MHz/900MHz/1800MHz/1900MHz)
- 3G/GPRS Module (Tri-Band UMTS 2100/1900/900MHz and Quad-Band GSM/EDGE, 850/900/1800/1900 MHz)
- GPS Module
- Sensor Modules (Sensor boards)
- Storage Module: SD Memory Card

4.2. Specifications

- **Microcontroller:** ATmega1281
- **Frequency:** 14.7456 MHz
- **SRAM:** 8KB
- **EEPROM:** 4KB
- **FLASH:** 128KB
- **SD Card:** 2GB
- **Weight:** 20gr
- **Dimensions:** 73.5 x 51 x 13 mm
- **Temperature Range:** [-10ºC, +65ºC]

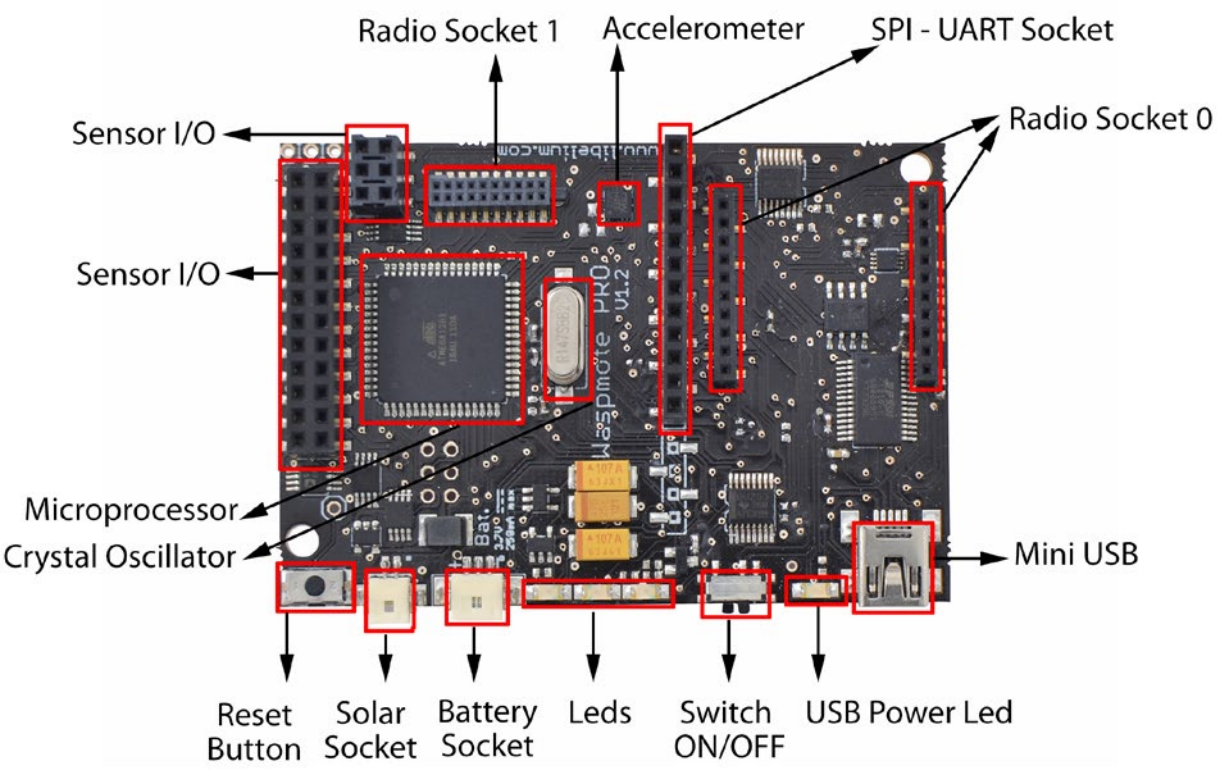


Figure 29: Main Waspmote components – Top side

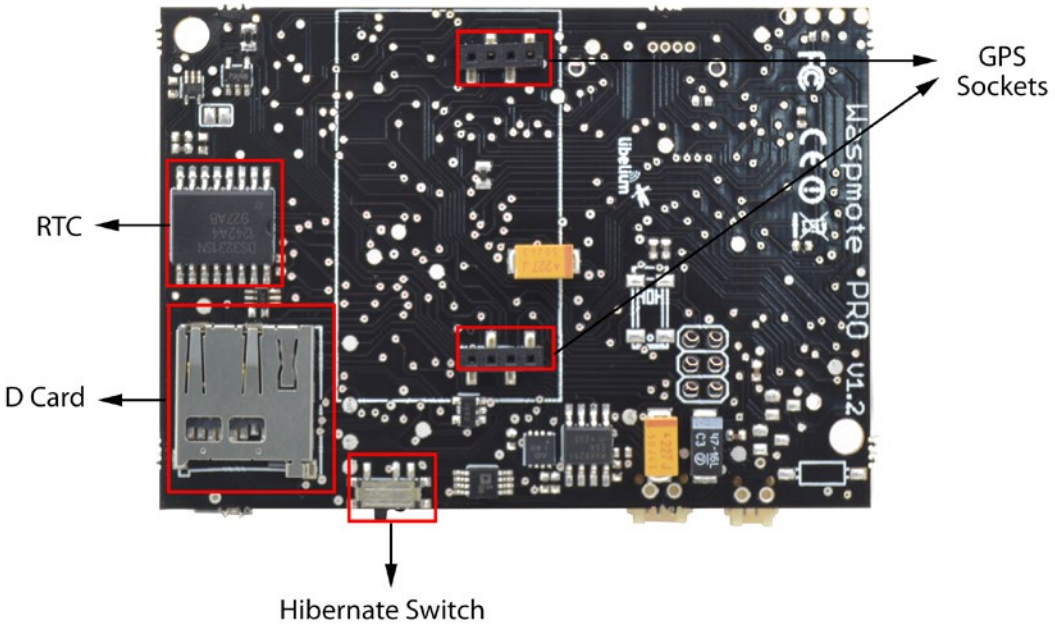


Figure 30: Main Waspmote components – Bottom side

4.3. Block Diagram

Data signals:

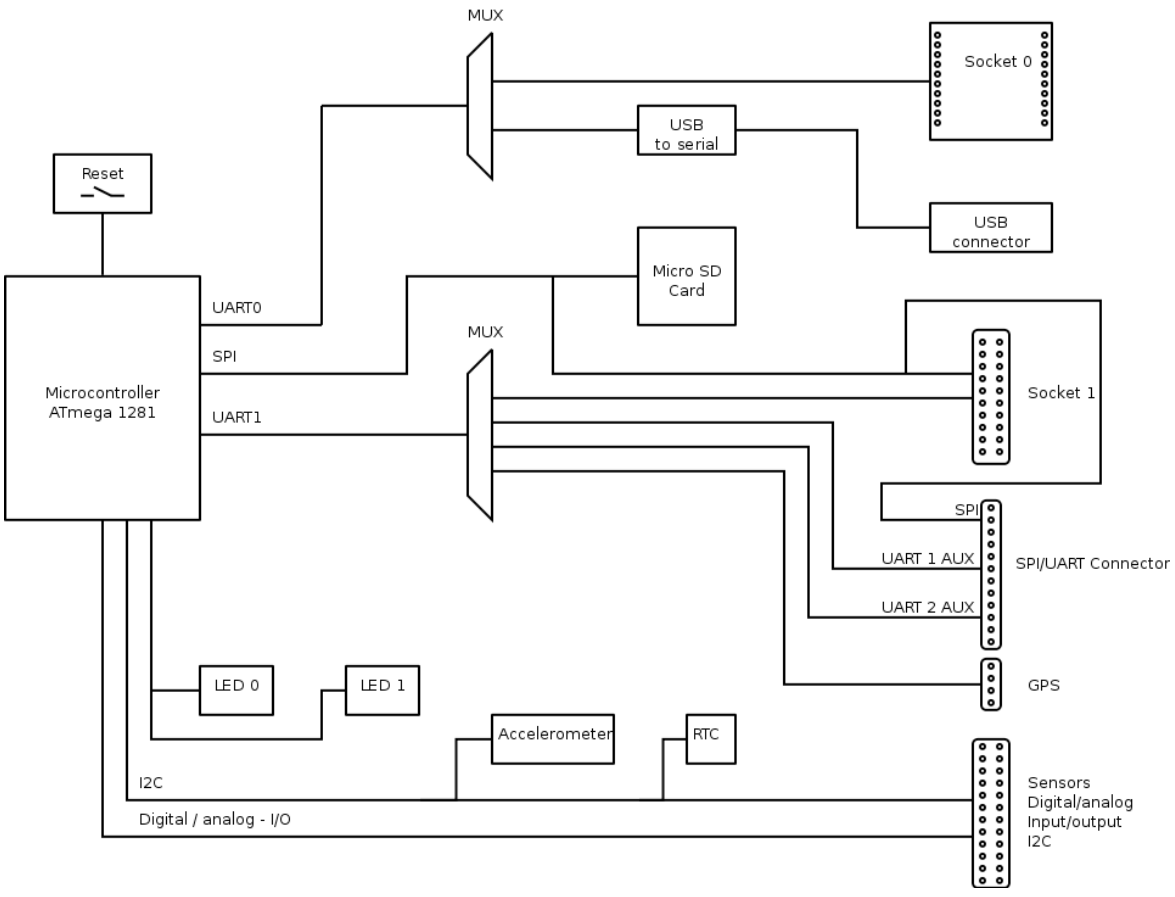


Figure 31: Waspmote block diagrams – Data signals

Power signals:

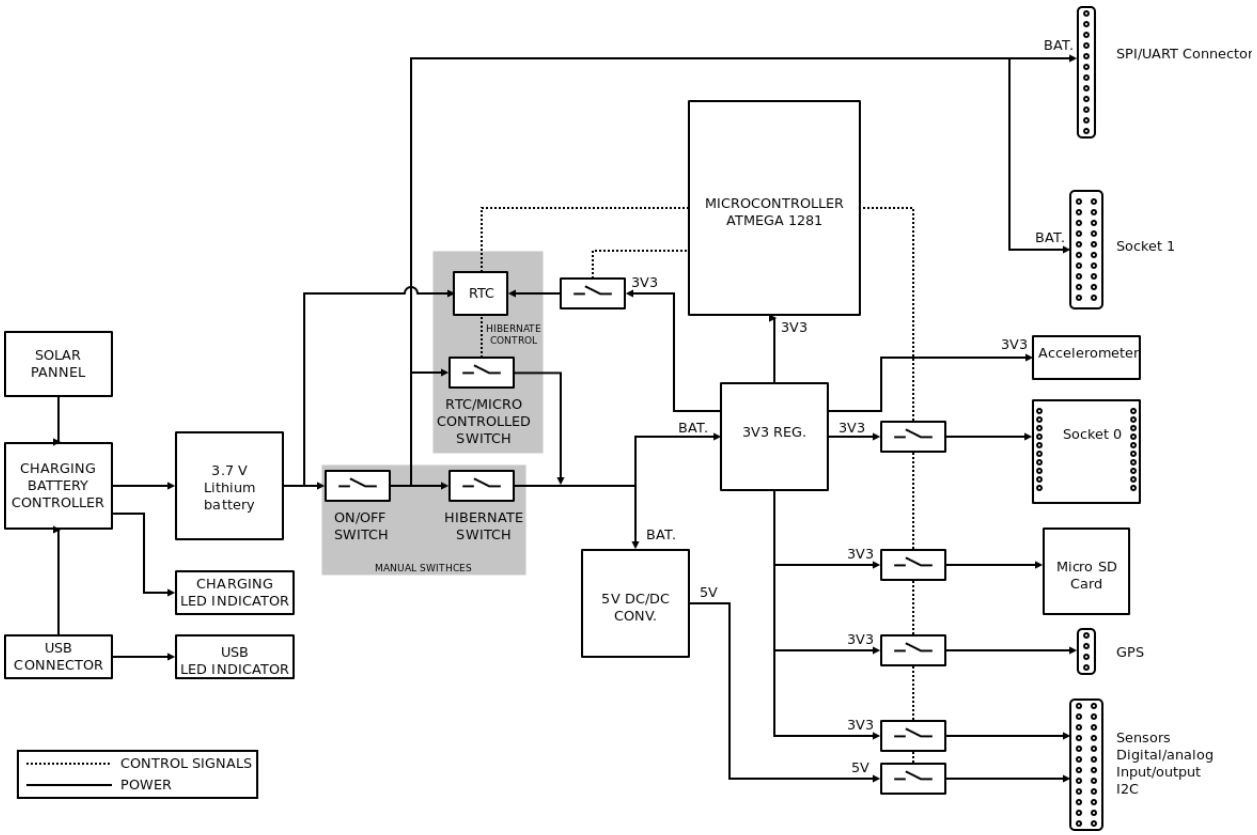


Figure 32: Waspmote block diagrams – Power signals

4.4. Electrical Data

Operational values:

- Minimum operational battery voltage 3.3 V
- Maximum operational battery voltage 4.2V
- USB charging voltage 5 V
- Solar panel charging voltage 6 - 12 V
- Battery charging current from USB 100 mA (max)
- Battery charging current from solar panel 280 mA (max)

Absolute maximum values:

- Voltage in any pin [-0.5 V, +3.8 V]
- Maximum current from any digital I/O pin 40 mA
- USB power voltage 7V
- Solar panel power voltage 18V
- Charged battery voltage 4.2 V

4.5. I/O

Waspmote can communicate with other external devices through the using different **input**/**output** ports.

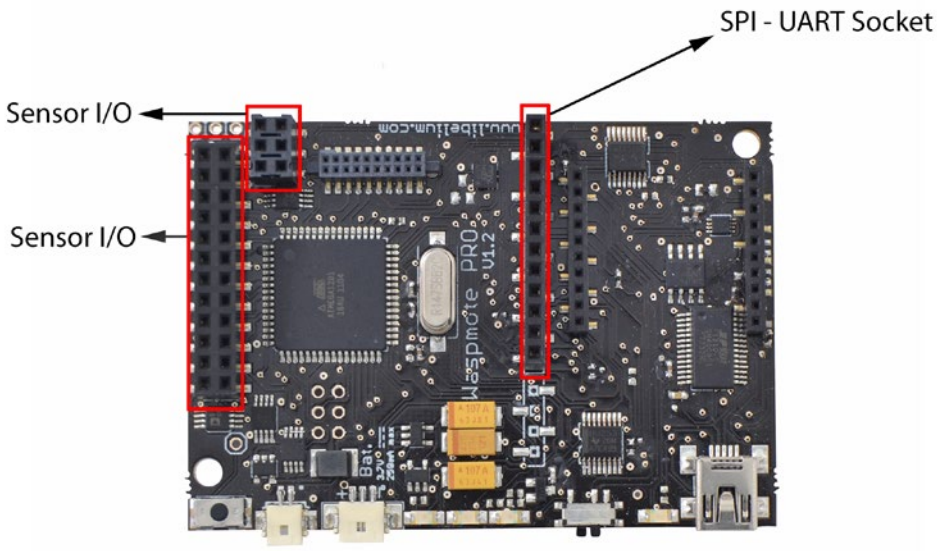


Figure 33: I/O connectors in Waspmote

Sensor connector:

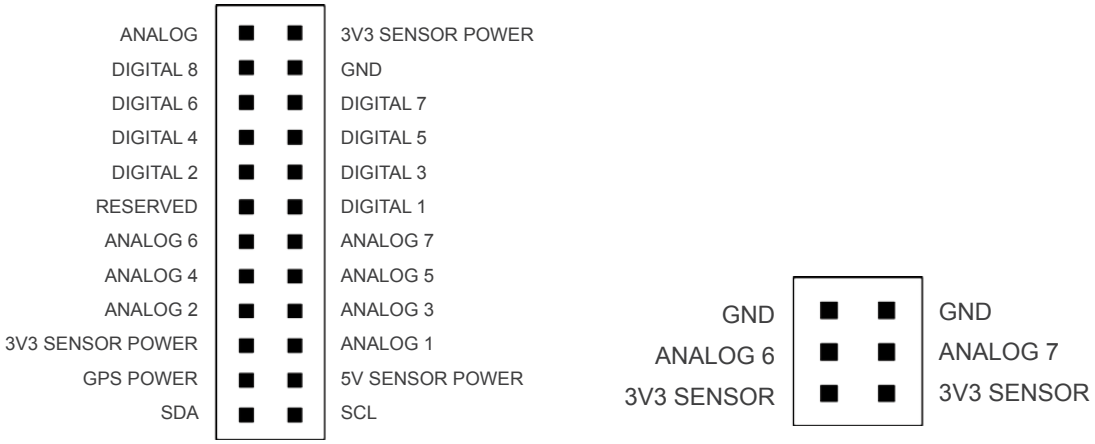


Figure 34: Description of sensor connector pins

Auxiliary SPI-UART connector:

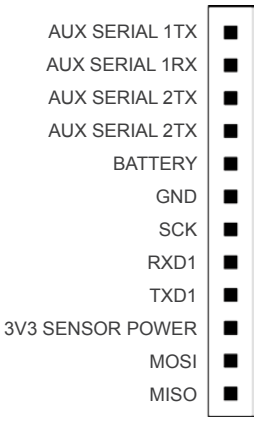


Figure 35: Description of auxiliary SPI-UART connector pins

4.5.1. Analog

Waspmote has **7** accessible analog inputs in the sensor connector. Each input is directly connected to the microcontroller. The microcontroller uses a **10 bit** successive approximation analog to digital converter (**ADC**). The reference voltage value for the inputs is **0V** (GND). The maximum value of input voltage is **3.3V** which corresponds with the microcontroller's general power voltage.

To obtain input values, the function `analogRead(analog input)` is used, the function's input parameter will be the name of the input to be read "ANALOG1, ANALOG2…" (see sensor connector figure). The value obtained from this function will be an integer number between **0 and 1023**, 0 corresponds to 0 V and 1023 to 3.3 V.

The analog input pins can also be used as digital input/output pins. If these pins are going to be used as digital ones, the following correspondence list for pin names must be taken into account:

Analog pin		Digital pin
ANALOG1	=>	14
ANALOG2	=>	15
ANALOG3	=>	16
ANALOG4	=>	17
ANALOG5	=>	18
ANALOG6	=>	19
ANALOG7	=>	20

```
{
  val = analogRead(ANALOG1);
}
```

4.5.2. Digital

Waspmote has digital pins which can be configured as **input or output** depending on the needs of the application. The voltage values corresponding to the different digital values would be:

- **0V** for logic 0
- **3.3V** for logic 1

The instructions for control of digital pins are:

```
{
  // set DIGITAL3 pin as input and read its value
  pinMode(DIGITAL3, INPUT);
  val = digitalRead(DIGITAL3);

  // set DIGITAL3 pin as output and set it LOW
  pinMode(DIGITAL3 ,OUTPUT);
  digitalWrite(DIGITAL3, LOW);
}
```

4.5.3. PWM

DIGITAL1 pin can also be used as output **PWM** (**Pulse Width Modulation**) with which an analog signal can be "simulated". It is actually a square wave between 0V and 3.3V for which the proportion of time when the signal is high can be changed (its working cycle) from 0% to 100%, simulating a voltage of 0V (0%) to 3.3V (100%).The resolution is **8 bit**, so up to 255 values between 0-100% can be configured. The instruction to control the PWM output is `analogWrite(DIGITAL1, value);` where value is the analog value (0-255).

```
{
  analogWrite(DIGITAL1, 127);
}
```

4.5.4. UART

There are two UARTs in Waspmote: UART0 and UART1. Besides, there are several ports which might be connected to these UARTs through two different multiplexers, one for each UART.

- **UART0** is shared by the USB port and the Socket0. This socket is used for XBee modules, RFID modules, Bluetooth modules, WiFi modules,etc. The multiplexer in this UART controls the data signal which by default is always switched to Socket0. When the USB needs to send info through the UART0, the multiplexer is momentarily switched to the USB port and set back again to Socket0 after printing.
- **UART1** is shared by four ports: Socket1, GPS socket, Auxiliar1 and Auxiliar2 sockets. It is possible to select in the same program which of the four ports is connected to UART1 in the microcontroller. UART1 multiplexer configuration is carried out using the following instructions:

```
{
  Utils.setMuxAux1(); // set Auxiliar1 socket
  Utils.setMuxAux2(); // set Auxiliar2 socket
  Utils.setMuxGPS();  // set GPS socket
  Utils.setMuxSocket1(); // set Socket1
}
```

4.5.5. I2C

The I2C communication bus is also used in Waspmote where two devices are connected in parallel: the accelerometer and the RTC . In all cases, the microcontroller acts as master while the other devices connected to the bus are slaves.

4.5.6. SPI

The SPI port on the microcontroller is used for communication with the micro SD card. All operations using the bus are performed clearly by the specific library. The SPI port is also available in the SPI/UART connector.

4.5.7. USB

USB is used in Waspmote for communication with a computer or compatible USB devices. This communication allows the microcontroller's program to be loaded.

For USB communication, microcontroller's UART0 is used. The **FT232RL** chip carries out the conversion to USB standard.

4.6. Real Time Clock - RTC

Waspmote has a built in Real Time Clock – RTC, which keeps it informed of the time. This allows Waspmote to be programmed to perform time-related actions such as:

"Sleep for 1h 20 min and 15sec, then wake up and perform the following action"

Or even programs to perform actions at absolute intervals, e.g.:

"Wake on the 5th of each month at 00:20 and perform the following action"

All RTC programming and control is done through the I2C bus.

Alarms:

Alarms can be programmed in the RTC specifying day/hour/minute/second. That allows total control about when the mote wakes up to capture sensor values and perform actions programmed on it. This allows Waspmote to be in the saving energy modes (**Deep Sleep** and **Hibernate**) and makes it wake up just at the required moment.

As well as relative alarms, periodic alarms can be programmed by giving a time measurement, so that Waspmote reprograms its alarm automatically each time one is triggered.

The RTC chosen is the Maxim **DS3231SN**, which operates at a frequency of **32.768Hz** (a second divisor value which allows it to quantify and calculate time variations with high precision).

The DS3231SN is one of the most accurate clocks on the market because of its internal compensation mechanism for the oscillation variations produced in the quartz crystal by changes in temperature (**Temperature Compensated Crystal Oscillator – TCXO**).

Most RTCs on the market have a variation of ± 20ppm which is equivalent to a 1.7s loss of accuracy per day (10.34min/year), however, the model chosen for Waspmote has a loss of just **± 2ppm**, which equates to variation of 0.16s per day (1min/year).

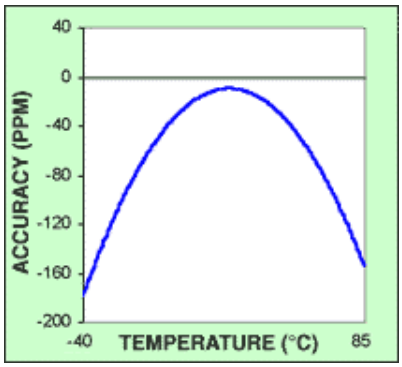


Figure 36: Uncompensated variation curve

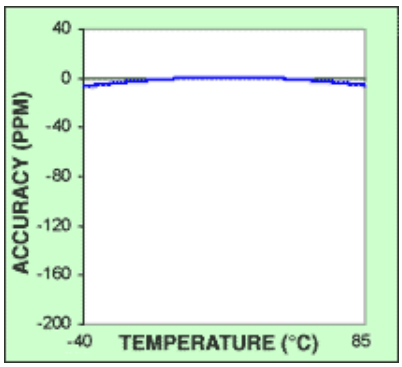


Figure 37: Compensated variation curve

Source: Maxim-ic.com

Figure 32 shows the temperature variation curve in a typical commercial clock, and in figure 33 that for the DS3231SN model built into Waspmote. As can be seen, variations in accuracy are practically zero at room temperature and minimal when moved to the ends of the temperature scale.

(For more information about clock calibrating methods in real time, consult web page:

http://www.maxim-ic.com/appnotes.cfm/an_pk/3566)

The recalibration process of the oscillation crystal is carried out thanks to the data received by the RTC's **internal temperature sensor**. The value of this digital sensor can be accessed by Waspmote through the I2C bus, which lets it know the **temperature of the board** at anytime in the range of **-40ºC to +85ºC** with an accuracy of 0.25°C. For more information about the acquisition of this value by the microprocessor, see the section "Sensors in Waspmote → Temperature".

Note: *the RTC's internal temperature sensor is only meant for the time derive compensation, but not for common air temperature sensing (we advise our Sensor Boards for that).*

The RTC is powered by the battery. When the mote is connected, the RTC is powered through the battery, but take into account that if the battery is removed or out of load, then time data will be not maintained. That is why we suggest to use RTC time like 'relative' and not 'absolute' (see Programming Guide for more info).

A coin or button battery is no longer needed. They had a limited life and therefore Waspmote can now have a much longer power life expectancy. This is so because the RTC is powered from the "main" battery which has a much bigger charge.

The RTC is responsible for waking Waspmote up from 2 of the maximum energy saving modes **Deep Sleep** and **Hibernate**. This makes possible for the Waspmote to use its battery just to power the RTC in sleep modes. The RTC controls when it has to wake Waspmote up and perform a particular action. This allows a consumption of **0.06µA** to be obtained in the Hibernate mode. See sections "Energy System" → Sleep mode and Deepsleep mode".

Related API libraries: **WaspRTC.h**, **WaspRTC.cpp**

All information about their programming and operation can be found in the document: **RTC Programming Guide.**

All the documentation is located in the **Development section** in the Libelium website.

4.7. LEDs

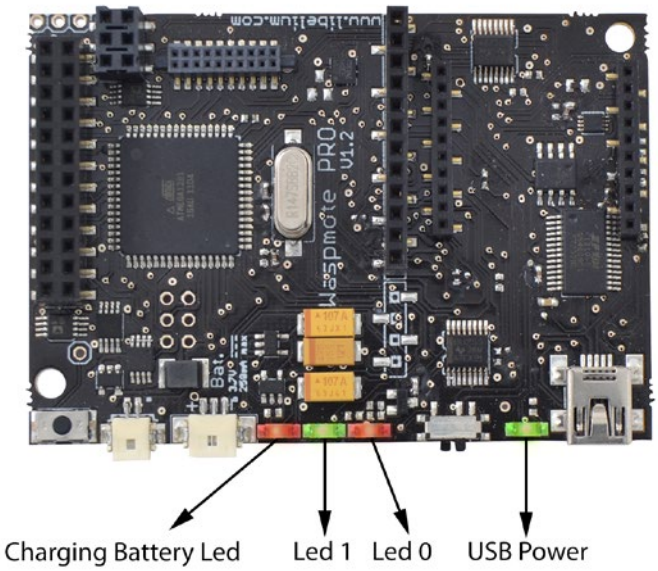


Figure 38: Visual indicator LEDs

- **Charging battery LED indicator**
  A red LED indicating that there is a battery connected in Waspmote which is being charged, the charging can be done through a mini USB cable or through a solar panel connected to Waspmote. Once the battery is completely charged, the LED switches off automatically.

- **LED 0 – programmable LED**
  A green indicator LED is connected to the microcontroller. It is totally programmable by the user from the program code. In addition, the LED 0 indicates when Waspmote resets, blinking each time a reset on the board is carried out.

- **LED 1 – programmable LED**
  A red indicator LED is connected to the microcontroller. It is totally programmable by the user from the program code.

- **USB Power LED indicator**
  A green LED which indicates when Waspmote is connected to a compatible USB port either for battery charging or programming. When the LED is on it indicates that the USB cable is connected correctly, when the USB cable is removed the LED will switch off automatically.

Programming

LED0 and LED1 are programmable. The functions for handling these LEDs are `Utils.setLED(LED_SELECTED, LED_MODE)` and `Utils.getLED(LED_SELECTED)` and `Utils.blinkLEDs()` (see the API manual for more information about these functions).

The other two LEDs switch on and off automatically according to their function.

```
{
  Utils.setLED(LED0, LED_ON);
  Utils.setLED(LED1, LED_OFF);
  Utils.blinkLEDS(1000);
}
```

5. Architecture and System

5.1. Concepts

The Waspmote's architecture is based on the Atmel **ATMEGA 1281** microcontroller. This processing unit starts executing the **bootloader** binary, which is responsible for loading into the memory the compiled programs and libraries previously stored in the FLASH memory, so that the main program that has been created can finally begin running.

When Waspmote is connected and starts the **bootloader**, there is a waiting time (62.5ms) before beginning the first instruction, this time is used to start loading new compiled programs updates. If a new program is received from the USB during this time, it will be loaded into the FLASH memory (128KB) substituting already existing programs. Otherwise, if a new program is not received, the last program stored in the memory will start running.

The structure of the codes is divided into 2 basic parts: **setup** and **loop**. Both parts of the code have sequential behaviour, executing instructions in the set order.

The **setup** is the first part of the code, which is only run once when the code is initialized. In this part it is recommended to include the initialization of the modules which are going to be used, as well as the part of the code which is only important when Waspmote is started.

The part named **loop** runs continuously, forming an infinite loop. Because of the behavior of this part of the code, the use of interruptions is recommended to perform actions with Waspmote.

A common programming technique to save energy would be based on blocking the program (either keeping the micro awake or asleep in particular cases) until some of interruptions available in Waspmote show that an event has occurred. This way, when an interruption is detected the associated function, which was previously stored in an interruption vector, is executed.

To be able to detect the capture of interruptions during the execution of the code, a series of flags have been created and will be activated to indicate the event which has generated the interruption (see chapters "Interruptions" and "Energy System").

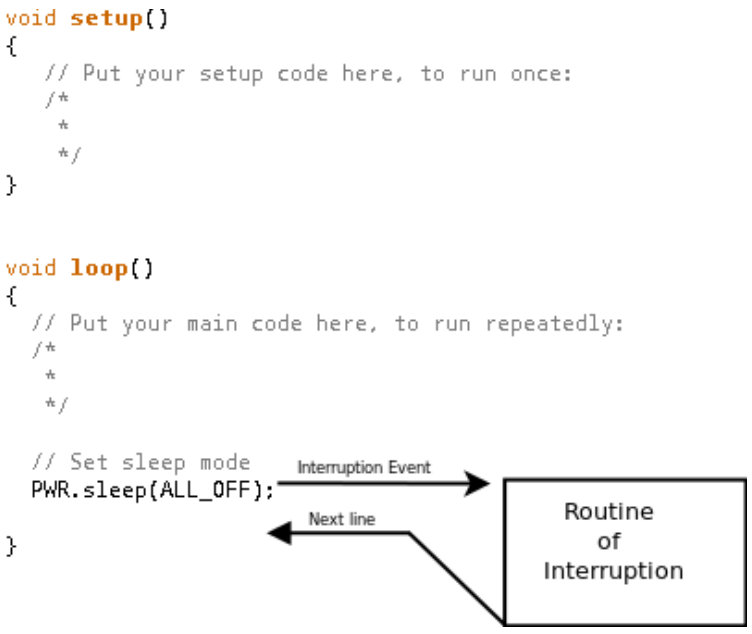


Figure 39: Blocking loop, interruption appears and is dealt with

When Waspmote is reset or switched on, the code starts again from the setup function and then the loop function.

By default, variable values declared in the code and modified in execution will be lost when a reset occurs or there is no battery. To store values permanently, it is necessary to use the microcontroller's **EEPROM (4KB)** non-volatile memory. EEPROM addresses from 0 to 1023 are used by Waspmote to save important data, so they must not be over-written. Thus, the available storage addresses go from 1024 to 4095. Another option is to use of the high capacity **2GB SD card**.

5.2. Timers

Waspmote uses a quartz oscillator which works at a frequency of **14.7456 MHz** as a system clock. In this way, every **125ns** the microcontroller runs a low level (machine language) instruction. It must be taken into account that each line of **C++** code of a program compiled by Waspmote includes several instructions in machine language.

Waspmote is a device prepared for operation in adverse conditions with regards to noise and electromagnetic contamination, for this reason, to ensure stable communication at all times with the different modules connected through a serial line to the UARTs (XBee, GPRS, USB) a maximum transmission speed of 115200bps has been set for XBee, GRPS and USB, and 4800 for the GPS, so that the success rate in received bits is 100%.

5.2.1. Watchdog

The Atmega 1281 microcontroller has an internal Enhanced Watchdog Time – WDT. The WDT **precisely** counts the clock cycles generated by a **128KHz oscillator.** The WDT generates an interruption signal when the counter reaches the set value. This interruption signal can be used to wake the microcontroller from the **Sleep** mode or to generate an internal alarm when it is running ON the mode, which is very useful when developing programs with timed interruptions.

The WDT allows the microcontroller to wake up from a low consumption Sleep mode by generating an interruption. For this reason, this clock is used as a time-based alarm associated with the microcontroller's **Sleep** mode. This allows very precise control of small time intervals: **16ms, 32ms, 64ms, 128ms, 256ms, 500ms, 1s, 2s, 4s, 8s**. For intervals over 8s (Deep Sleep mode) the RTC is used.

More information about the interruptions generated by the Watchdog can be found in Energy chapter.

Related API libraries: **WaspPWR.h**, **WaspPWR.cpp**

All information about their programming and operation can be found in the document: **Energy and Power Programming Guide.**

All the documentation is located in the **Development section** in the Libelium website.

5.2.2. RTC

As shown in the Hardware chapter, Waspmote has a real time clock (RTC) running a 32KHz (32.786Hz) which allows to set an absolute time.

Alarms can be programmed in the RTC specifying day/hour/minute/second. This allows total control when the mote wakes up to capture values and perform actions programmed on it. Also, the RTC allows Waspmote to function in the maximum energy saving modes (**Deep Sleep and Hibernate**) and to wake up just at the required moment.

The RTC allows the microcontroller to be woken from the low consumption state by generating an interruption. For this reason, it has been associated to the microcontroller's Deep Sleep and Hibernate modes, making it possible to put the microcontroller to sleep, and wake it up by activating an alarm in the RTC. Sleeping intervals can go from 8s, to minutes, hours or even days.

More information about the interruptions generated by the RTC and DeepSleep and Hibernate modes can be found in the Energy management chapter.

Related API libraries: **WaspRTC.h**, **WaspRTC.cpp**

All information about the RTC programming and operation can be found in the document: **RTC Programming Guide.**

All the documentation is located in the **Development section** in the Libelium website.

6. Interruptions

Interruptions are signals received by the microcontroller which indicate it must stop the task is doing to attend to an event that has just happened. Interruption control frees the microcontroller from having to control sensors all the time. It also makes the sensors warn Waspmote when a determined value (threshold) is reached.

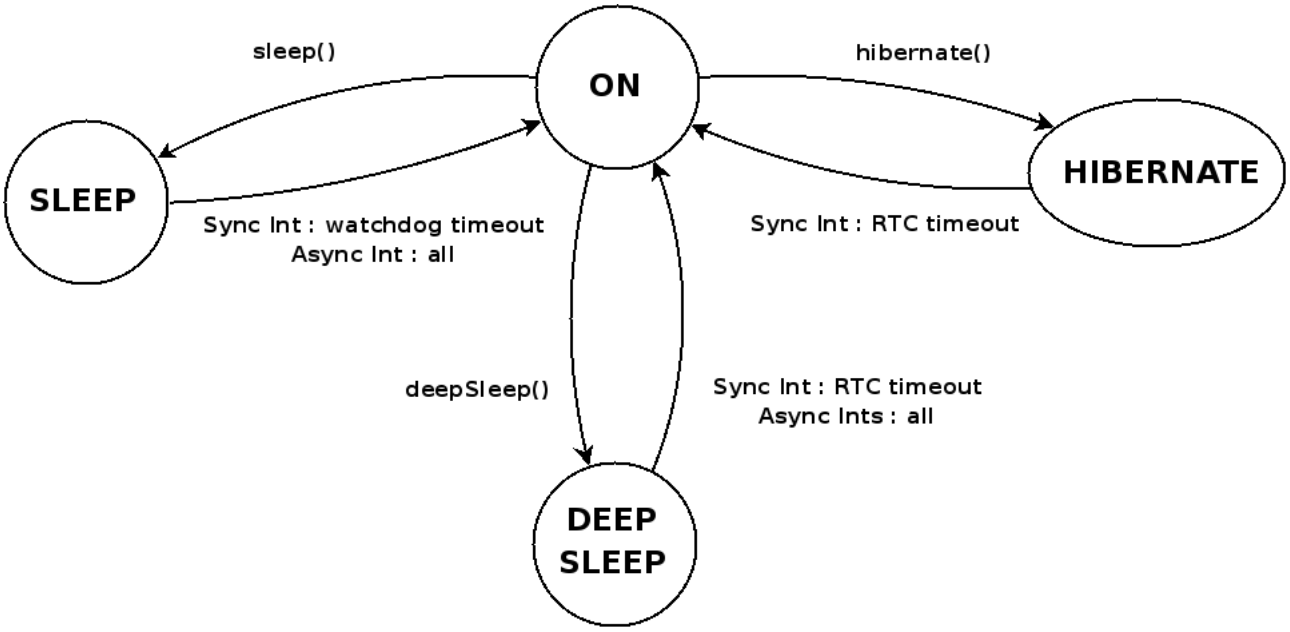


Figure 40: Diagram of mode in Waspmote

Waspmote is designed to work with 2 types of interruptions: Synchronous and asynchronous

- **Synchronous Interruptions**
  They are programmed by **timers**. They allow to program when we want them to be triggered. There are two types of timer alarms: **periodic** and **relative**.
  - **Periodic Alarms** are those to which we specify a particular moment in the future, for example: "*Alarm programmed for every fourth day of the month at 00:01 and 11 seconds*", they are controlled by the RTC.
  - **Relative alarms** are programmed taking into account the current moment, eg: *"Alarm programmed for 5 minutes and 10 seconds"*, they are controlled through the RTC and the microcontroller's internal Watchdog.

- **Asynchronous Interruptions**
  These are not programmed so it is not known when they will be triggered. Types:
  - **Sensors:** the sensor boards can be programmed so that an alarm is triggered when a sensor reaches a certain threshold.
  - **Accelerometer**: The accelerometer that is built into the Waspmote can be programmed so that certain events such as a fall or change of direction generate an interruption.
  - **XBee module:** Digimesh protocol allows the XBee to set cyclic sleep modes which can interrupt Waspmote each time the module wakes up. This permits to set up cyclic sleep networks.

All interruptions, both synchronous and asynchronous can **wake** Waspmote up from the **Sleep** and the **Deep Sleep mode**. However, only the synchronous interruption by the RTC is able to wake it up from the **Hibernate** mode.

The **Hibernate** mode totally disconnects the Waspmote power, leaving only the battery powering the RTC to wake Waspmote up when the time alarm is reached. Because of this disconnection, when the RTC generates the corresponding alarm, the power in Waspmote is reconnected and the code starts again from the setup.

The way of detecting whether a reboot from the **Hibernate** mode has happened is to check whether the corresponding flag has been activated. Activation of this flag happens when the `ifHibernate()` function is called, which must be done at the beginning of the **setup** part of the code. This way, when Waspmote starts, it tests if it is a normal start or if it is an start from the **Hibernate** mode.

All information about the programming and operation of interruptions can be found in the document: **Interruption Programming Guide.**

7. Energy System

7.1. Concepts

Waspmote has 4 operational modes.

- **ON**: Normal operation mode. Consumption in this state is **15mA**.
- **Sleep**: The main program is paused, the microcontroller passes to a latent state, from which it can be woken up by **all** asynchronous interruptions and by the synchronous interruption generated by the Watchdog. The duration interval of this state is from **32ms to 8s**. Consumption in this state is **55µA**.
- **Deep Sleep**: The main program pauses, the microcontroller passes to a latent state from which it can be woken up by **all** asynchronous interruptions and by the synchronous interruption triggered by the RTC. The interval of this cycle can be from **seconds to minutes, hours, days**. Consumption in this state is **55µA**.
- **Hibernate**: The main program stops, the microcontroller and all the Waspmote modules are completely disconnected. The only way to reactivate the device is through the previously programmed alarm in the RTC (synchronous interrupt). The interval of this cycle can be from **seconds to minutes, hours, days**. Almost all devices are totally disconnected from the battery: only the RTC is powered through the battery, from which it consumes **0.06µA**.

	Consumption	Micro	Cycle	Accepted Interruptions
ON	15mA	ON	-	Synchronous and Asynchronous
Sleep	55µA	ON	32ms - min/hours/days	Synchronous (Watchdog) and Asynchronous
Deep Sleep	55µA	ON	1s – min/hours/days	Synchronous (RTC) and Asynchronous
Hibernate	0.06µA	OFF	1s – min/hours/days	Synchronous (RTC)

On the other hand, each **module** might have up to 4 operation modes.

- **ON**: Normal operation mode.
- **Sleep**: In this mode **some** module functions are stopped and passed to asynchronous use, normally guided by events. It functions differently in each module and is specific to each one (programmed by the manufacturer).
- **Hibernate**: In this mode **all** module functions are stopped and passed to asynchronous use, normally guided by events. It operates differently in each module and is specific to each one (programmed by the manufacturer).
- **OFF**: By using digital switches controlled by the microcontroller the module is switched off completely. This mode has been implemented by **Libelium** as an **independent layer of energy control**, so that it can reduce consumption to a minimum (**~0µA**) without relegating to techniques implemented by the manufacturer.

For complete information about interruption types and their handling, see the Interruption chapter.

Related API libraries: **WaspPWR.h**, **WaspPWR.cpp**

All information about the programming and operation of interruptions can be found in the document: **Energy and Power Programming Guide**.

All the documentation is located in the **Development section** in the Libelium website.

Note: *The sleep mode for XBee is not a very useful feature, since the advised action is to switch XBee off after transmission. If the user puts XBee in sleep mode and also switches Waspmote to sleep or deepsleep, and if the SD card is plugged, there will be an excessive power consumption: 220 µA or more (instead of the expected 110 µA). This is due to parasite power. To solve that, the user should not use the XBee sleep mode. Another solution is to call* `sleep()` *or* `deepsleep()` *with* `ALL_OFF` *or* `SOCKET0_OFF` *parameters.*

7.2. Sleep mode

The main program is paused, the microcontroller passes to a latent state, from which it can be woken by **all** asynchronous interruptions and by the synchronous interruption generated by the Watchdog. When the Watchdog Timer is set up, the duration interval of this state is from **16ms to 8s**. Consumption in this state is **55µA**.

In this mode the microcontroller stops executing the main program. The program stack where all the variables and log values are stored keep their value, so when Waspmote returns to ON mode, the next instruction is executed and the variable values are maintained.

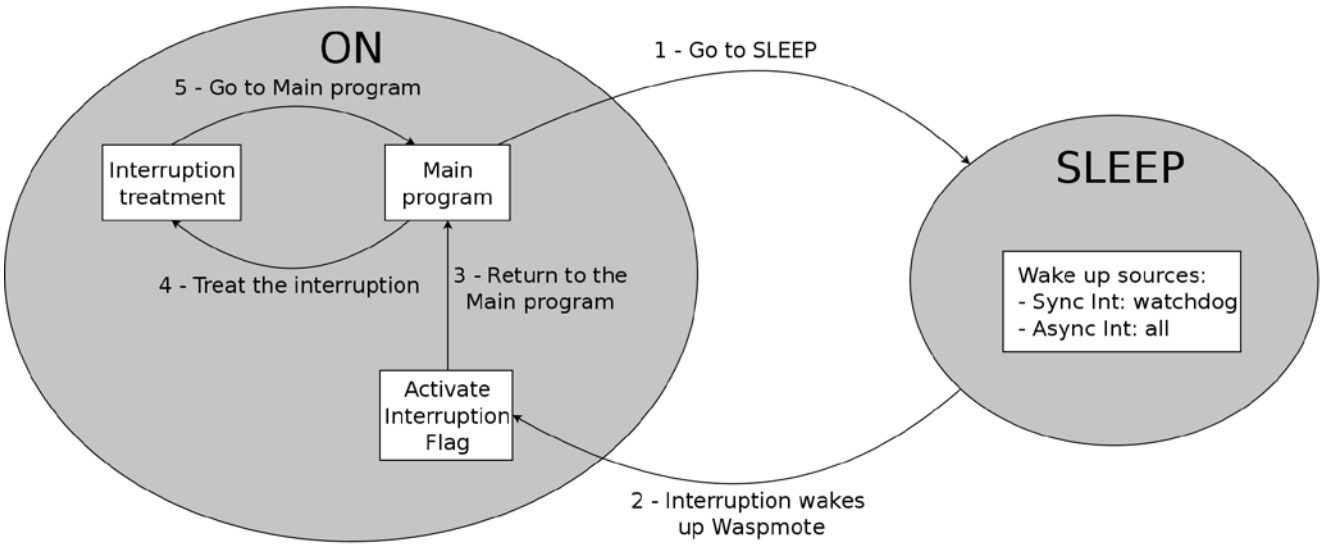


Figure 41: From ON to Sleep

The following example would set Waspmote in the Sleep mode for 32ms. The microprocessor would be in a state of minimum consumption waiting for the synchronous interruption from the Watchdog.

```
{
  PWR.sleep(WTD_32MS, ALL_OFF);
}
```

7.3. Deep Sleep mode

The main program is paused, the microcontroller passes to a latent state from which it can be woken by all the asynchronous interruptions and by the synchronous interruption launched by the RTC. The interval of this cycle can go from **seconds to minutes, hours, days**. Consumption in this state is **55µA**.

In this mode the microcontroller stops executing the main program. The program stack where all the variables and log values are stored keep their value, so when Waspmote returns to ON mode, the next instruction is executed and the variable values are maintained.

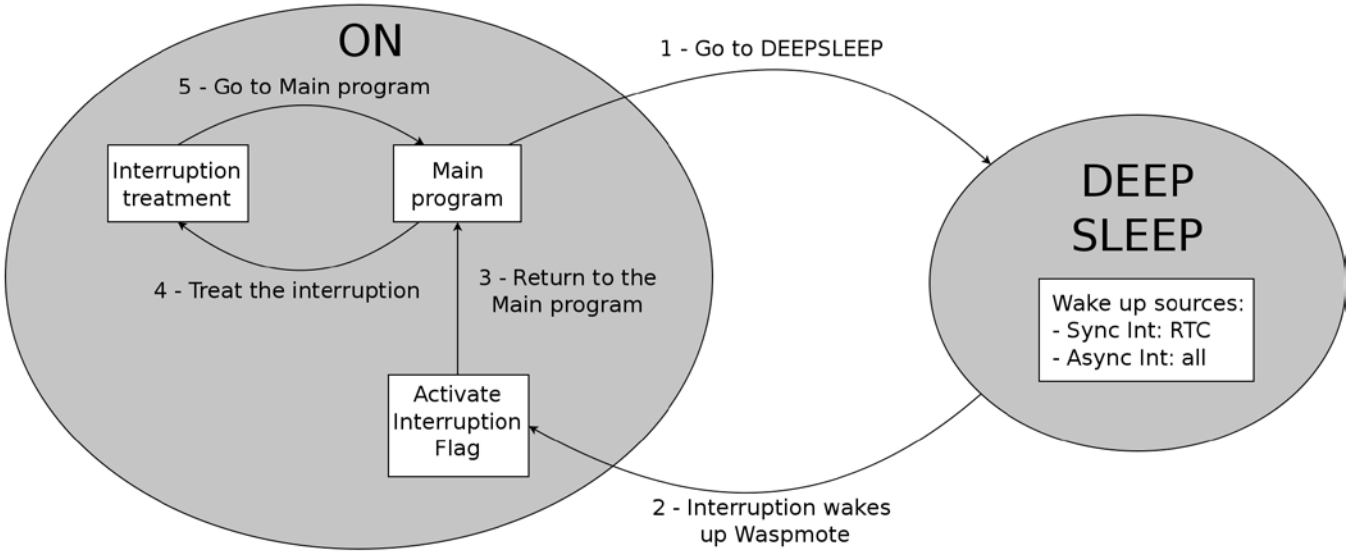


Figure 42: From ON to Deep Sleep

7.4. Hibernate mode

The main program stops, the microcontroller and all the Waspmote modules are completely disconnected. The only way to reactivate the device is through the previously programmed alarm in the RTC (synchronous interrupt). The interval for this cycle can go from **seconds to minutes, hours or days**. Almost all devices are totally disconnected from the battery: only the RTC is powered through the battery, from which it consumes **0.06µA**.

In this mode the microcontroller does not store any values from variables or from the program stack. When leaving the Hibernate state the micro is reset, so the **setup** and **loop** routines are run as if the main switch were activated.

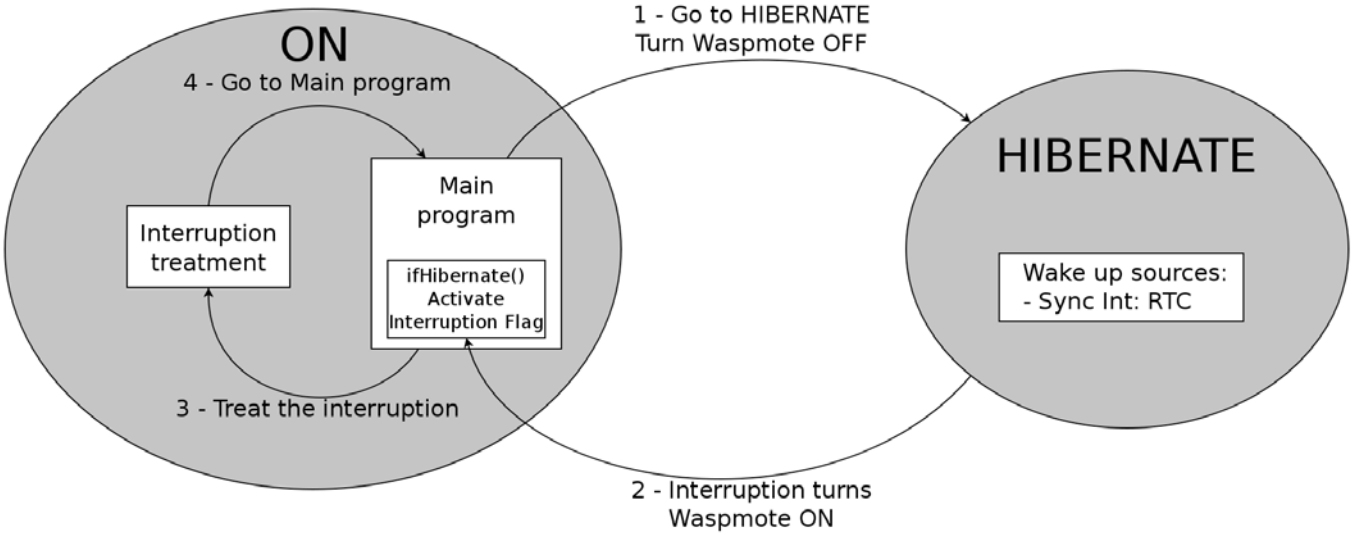


Figure 43: From ON to Hibernate

Hibernate mode requires the hibernate switch to be turned off correctly. It is necessary to follow the next steps when executing the program for first time after uploading it to Waspmote:

1. Connect the battery
2. Switch Waspmote on.
3. Wait for the red led to light on and turn off the "Hibernate switch" while the red led is on.
4. Once the "Hibernate switch" is off, the green led must blink to indicate the program is running.

The following example would set Waspmote in the Hibernate mode for 2 days, 1 hour and 30 minutes. The microcontroller would be switched off waiting for the RTC to switch the device on again with a synchronous interruption.

```
{
  PWR.hibernate("02:01:30:00", RTC_OFFSET, RTC_ALM1_MODE2);
}
```

Note: *when the hibernate jumper is not connected, RTC alarms must only be used to set the wake up from hibernate. See more details in the Programming Guides for the RTC and Power Modes.*

Related API libraries: **WaspPWR.h**, **WaspPWR.cpp**

All information about the programming and operation of sleep modes can be found in the document: **Energy and Power Programming Guide.**

All the documentation is located in the **Development section** in the Libelium website.

8. Sensors

8.1. Sensors in Waspmote

8.1.1. Temperature

The Waspmote RTC (**DS3231SN from Maxim**) has a built in internal temperature sensor which it uses to **recalibrate itself.** Waspmote can access the value of this sensor through the I2C bus.

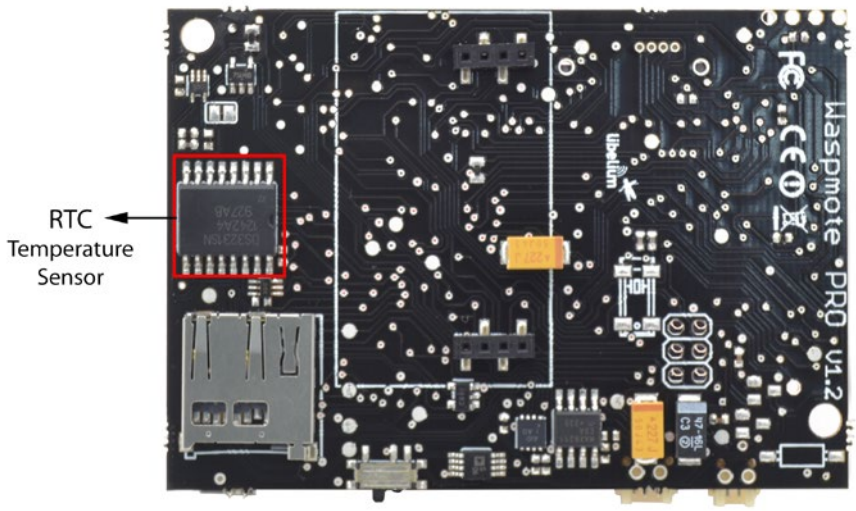


Figure 44: Temperature sensor in the RTC

Obtaining the temperature:

```
{
  RTC.getTemperature();
}
```

The sensor is shown in a 10-bit two's complement format. It has a resolution of **0.25º C**. The measurable temperature range is between **-40ºC** and **+85ºC**.

As previously specified, the sensor is prepared to measure the temperature of the board itself and can thereby compensate for oscillations in the quartz crystal it uses as a clock. As it is a sensor built in to the RTC, for any application that requires a probe temperature sensor, this must be integrated from the micro's analog and digital inputs, as has been done in the case of the sensor boards designed by Libelium.

More information about the RTC can be found in the "Hardware" and "Energy System" chapters.

Related API libraries: **WaspRTC.h, WaspRTC.cpp**

All information about their programming and operation can be found in the document: **RTC Programming Guide.**

All the documentation is located in the **Development section** in the Libelium website.

8.1.2. Accelerometer

Waspmote has a built in acceleration sensor LIS3331LDH STMicroelectronics which informs the mote of acceleration variations experienced on each one of the 3 axes (X,Y,Z).

The integration of this sensor allows the measurement of acceleration on the 3 axes (X,Y,Z), establishing 4 kind of events: Free Fall, inertial wake up, 6D movement and 6D position which are explained in the Interruptions Programming Guide.

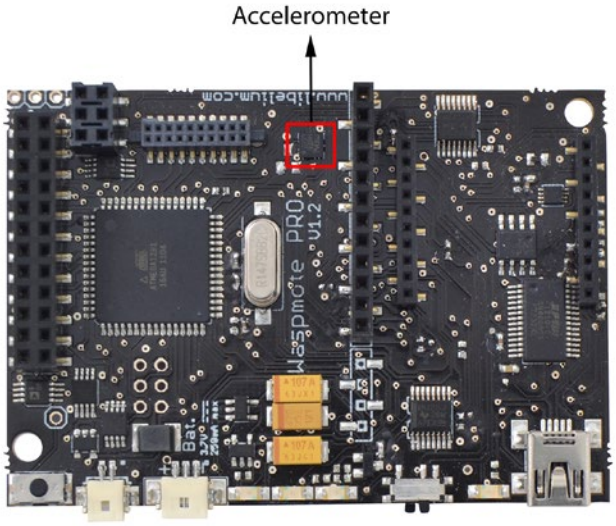


Figure 45: Accelerometer

The LIS331DLH has dynamically user selectable full scales of **±2g/±4g/±8g** and it is capable of measuring accelerations with output data rates from **0.5 Hz to 1 kHz**.

The device features ultra low-power operational modes that allow advanced power saving and smart sleep to wake-up functions.

The accelerometer has 7 power modes, the output data rate (ODR) will depend on the power mode selected. The power modes and output data rates are shown in this table:

Power mode	Output data rate (Hz)
Power down	--
Normal mode	1000
Low-power 1	0,5
Low-power 2	1
Low-power 3	2
Low-power 4	5
Low-power 5	10

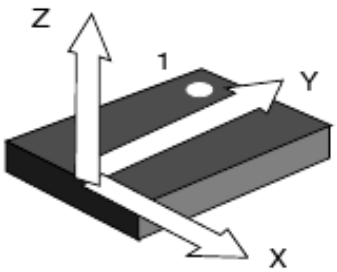


Figure 46: Axes in the LIS3LV02DL accelerometer

This accelerometer has an auto-test capability that allows the user to check the functioning of the sensor in the final application. Its operational temperature range is between -40ºC and +85ºC.

The accelerometer communicates with the microcontroller through the I2C interface. The pins that are used for this task are the SCL pin and the SDA pin, as well as another INT pin to generate the interruptions.

The accelerometer has 4 types of event which can generate an interrupt: free fall, inertial wake up, 6D movement and 6D position.

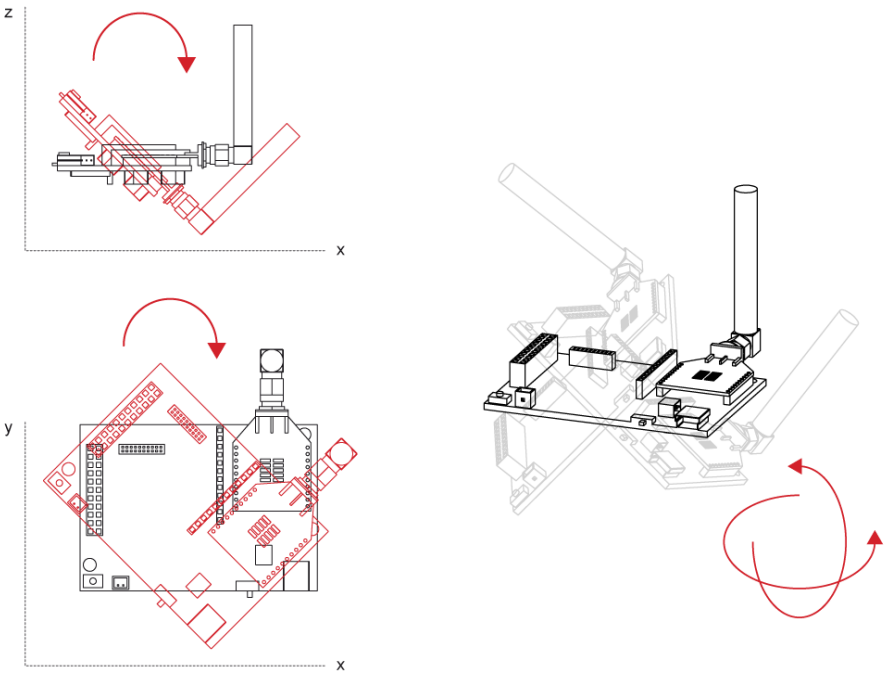
These thresholds and times are set in the WaspACC.h file.

To show the ease of programming, an extract of code about how to get the accelerometer values is included below:

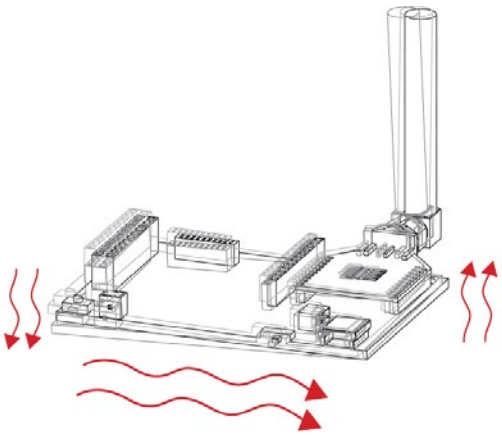
```
{
  ACC.ON();
  ACC.getX();
  ACC.getY();
  ACC.getZ();
}
```

Some figures with possible uses of the accelerometer are shown below:

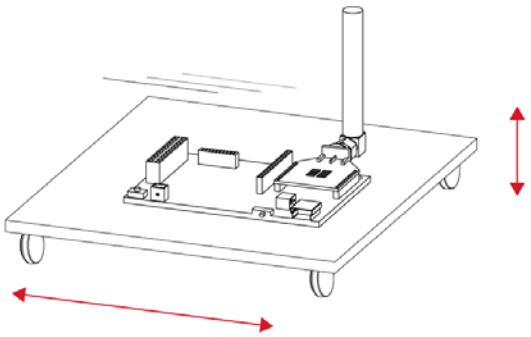
Rotation and Twist:



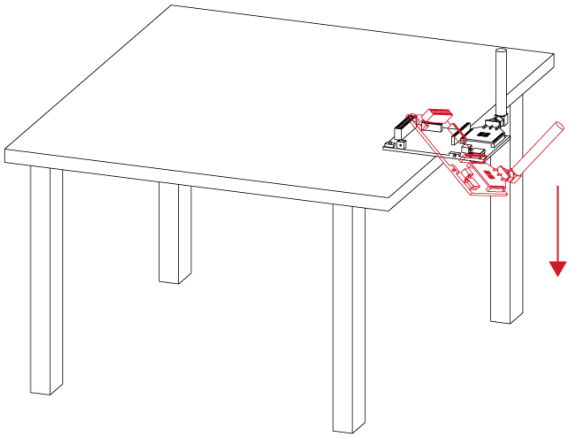
Vibration:



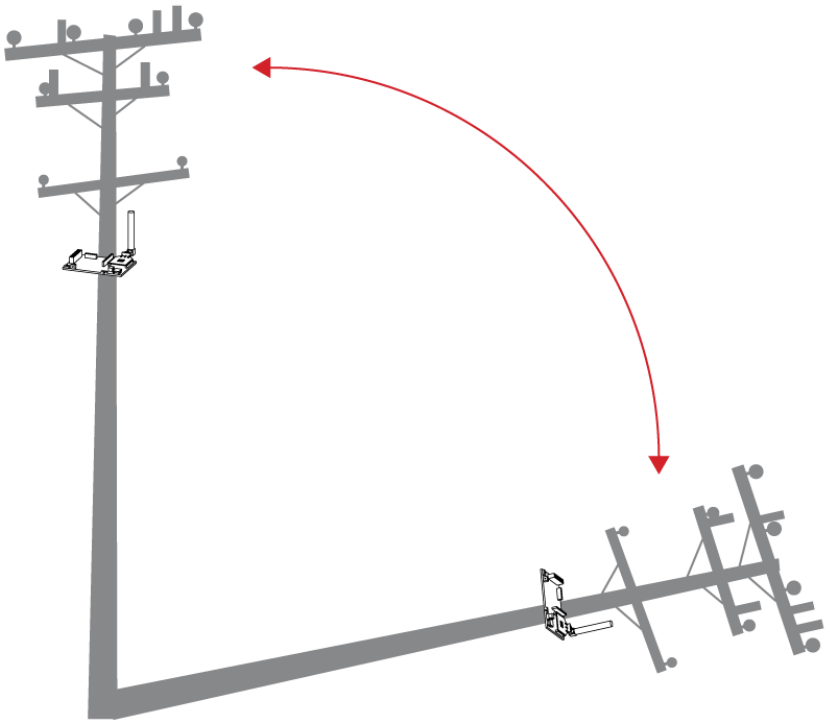
Acceleration:



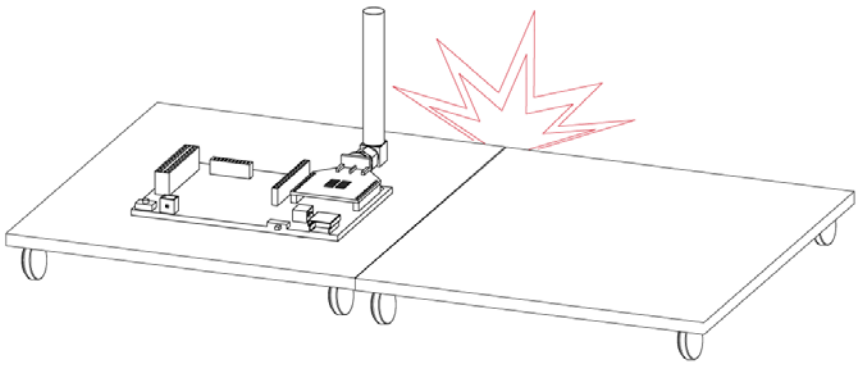
Free fall:



Free fall of objects in which it is installed:



Crash:



More information about interruptions generated by the accelerometer can be found in the chapter "Interruptions" and in the **Interruptions Programming Guide**.

Related API libraries: **WaspACC.h, WaspACC.cpp**

All information about their programming and operation can be found in the document: **Accelerometer Programming Guide.**

All the documentation is located in the **Development section** in the Libelium website.

8.2. Integration of new sensors

The Waspmote design is aimed at easing integration of both **input** (**sensors**) and **output** (**actuators**) which allow expansion of the already wide range of mote responses. These are connected to the board by its **2x12** and **1x12** pin connectors, which allow communication of **16** digital input and output signals, of which **7** can be used as analog inputs and **1** as a **PWM** (Pulse Width Modulation) output signal, as well as a line to ground, 3.3V and 5V power feeds, 2 selectable connections to the serial communication (**UART**) inputs and outputs, connection to the two lines of the (**I2C**) SCL and SDA Inter-Integrated Circuit bus, and connection to inputs for high level and low level interrupt. An image of the Waspmote output connectors can be seen in the section of the manual on "Inputs/Outputs".

The management of sensor board's two power lines (described in more depth in the section "Sensors" → Power) is carried out through two solid state switches which allow the continuous passage of a current of up to **200mA** and whose control can be programmed using the functions included in the WaspPWR library, described in the files WaspPWR.h and WaspPWR.cpp.

The input and output voltage values for both digital and analog pins will be between 0/ and 3.3V, logic zero **('0')** being found in values less than **0.5V** and logic one **('1')** in values higher than **2.30V**. To read analog signals, the microprocessor has a **10-bit** analog-to-digital converter which allows a resolution of 3mV. Waspmote also has one 8-bit resolution PWM output pin for the generation of analog signals. Information on the libraries and instructions used for reading and writing on these pins can be found in the API manual.

Waspmote includes 2 interruption pins, a low level (**TXD1**) one and a high level (**RXD1**) one, which offer an alternative to reading the sensors by survey, allowing the microprocessor to be woken up when an **event** occurs (such as exceeding a certain threshold in a comparator) which generates a change in a digital signal connected to one of the above pins, facilitating the sensor reading only at the moments when a remarkable event occurs.

This option is especially recommended for low consumption sensors that may remain active for long periods of time. Reading by survey (switched on and cyclical sensor reading after a set time) is more appropriate for those that, in addition to showing greater consumption, do not require monitoring that generates an alarm signal. The interruptions can be managed using the warning functions and vectors (flags) defined in the Winterruptions library, file Winterruptions.c. More can be learnt about their use in the Interruptions Programming Guide.

Sensors reading can generate three types of response: storage of collected data (on the SD card), wireless transmission of data (using a radiofrequency signal through the XBee module or through the mobile communications network using the GRPS module) or automatic activation through an actuator directly controlled by the microprocessor's output signals or through a switch or relay.

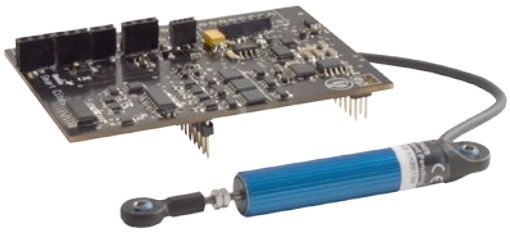
8.3. Sensor Boards

The integration of sensors requiring some type of electronic adaptation stage or signal processing prior to reading by the microprocessor is carried out by the various microprocessor sensor boards. Connection between these and the mote takes place pin to pin using the two 2x11 and 1x12 connectors mentioned in the section "Hardware" → I/O. Currently, Waspmote has eight integration boards:

GASES	APPLICATIONS	SENSORS
	• **City pollution** CO, CO_2, NO_2, O_3 • **Emissions from farms and hatcheries** CH_4 ,H_2S , NH_3 • **Control of chemical and industrial processes** C_4H_{10} ,H_2, VOC • **Forest fires** CO, CO_2	• Carbon Monoxide – CO • Carbon Dioxide – CO_2 • Oxygen – O_2 • Methane – CH_4 • Hydrogen – H_2 • Ammonia – NH_3 • Isobutane – C_4H_{10} • Ethanol – CH_3CH_2OH • Toluene – $C_6H_5CH_3$ • Hydrogen Sulfide – H_2S • Nitrogen Dioxide – NO_2 • Ozone – O_3 • Hydrocarbons – VOC • Temperature • Humidity • Pressure atmospheric

EVENTS	APPLICATIONS	SENSORS
	• **Security** Hall effect (doors and windows), person detection PIR • **Emergencies** Presence detection and water level sensors, temperature • **Control of goods in logistics**	• Pressure/Weight • Bend • Hall Effect • Temperature (+/-) • Liquid Presence • Liquid Level • Liquid flow • Luminosity • Presence (PIR) • Stretch

SMART CITIES



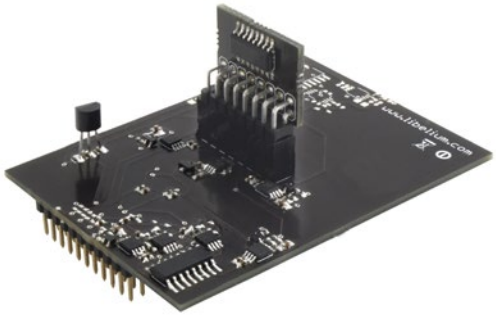
APPLICATIONS

- **Noise maps**
  Monitor in real time the acoustic levels in the streets of a city
- **Structural health monitoring**
  Crack detection and propagation
- **Air quality**
  Detect the level of particulates and dust in the air
- **Waste management**
  Measure the garbage levels in bins to optimize the trash collection routes

SENSORS

- Microphone (dBSPLA)
- Crack detection gauge
- Crack propagation gauge
- Linear displacement
- Dust - PM-10
- Ultrasound (distance measurement)
- Temperature
- Humidity
- Luminosity

SMART PARKING



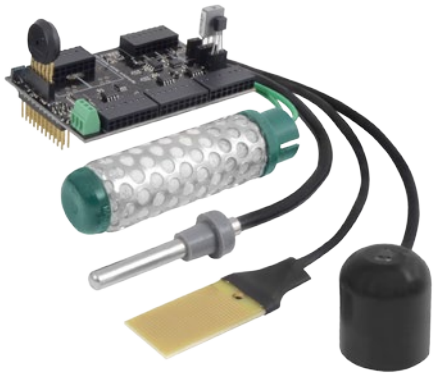
APPLICATIONS

- Car detection for available parking information
- Detection of free parking lots outdoors
- Parallel and perpendicular parking lots control

SENSORS

- Magnetic Field
- Temperature

AGRICULTURE



APPLICATIONS

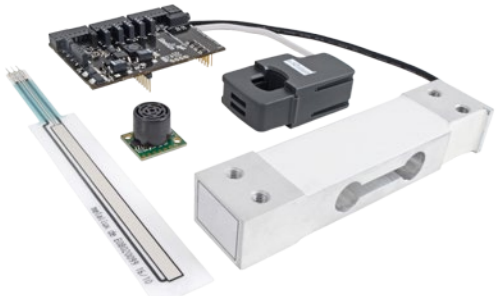
- **Precision Agriculture**
  Leaf temperature, fruit diameter
- **Irrigation Systems**
  Soil moisture, leaf wetness
- **Greenhouses**
  Solar radiation, humidity, temperature
- **Weather Stations**
  Anemometer, wind vane, pluviometer

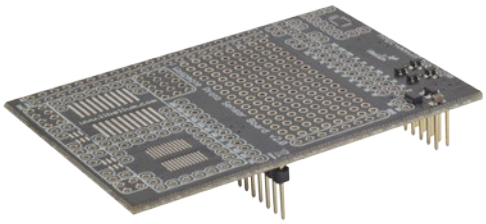
SENSORS

- Air Temperature / Humidity
- Soil Temperature / Moisture
- Leaf Wetness
- Atmospheric Pressure
- Solar Radiation - PAR
- Ultraviolet Radiation - UV
- Trunk Diameter
- Stem Diameter
- Fruit Diameter
- Anemometer
- Wind Vane
- Pluviometer
- Luminosity

VIDEO CAMERA	APPLICATIONS	SENSORS
	• Security and surveillance • Take photos (640 x 380) • Record video (320 x 240) • Realtime Videocall using 3G network • Night Vision mode available	• Image sensor • Luminosity • Infrared • Presence (PIR)

RADIATION	APPLICATIONS	SENSORS
	• Monitor the radiation levels wirelessly without comprising the life of the security forces • Create prevention and control radiation networks in the surroundings of a nuclear plant • Measure the amount of Beta and Gamma radiation in specific areas autonomously	• Geiger tube [β, γ] (Beta and Gamma)

SMART METERING	APPLICATIONS	SENSORS
	• Energy measurement • Water consumption • Pipe leakage detection • Liquid storage management • Tanks and silos level control • Supplies control in manufacturing • Industrial Automation • Agricultural Irrigation	• Current • Water flow • Liquid level • Load cell • Ultrasound • Distance Foil • Temperature • Humidity • Luminosity

PROTOTYPING SENSOR	APPLICATIONS	
	• Prepared for the **integration of any kind of sensor**.	• Pad Area • Integrated Circuit Area • Analog-to-Digital Converter (16b)

It is possible to find more detailed information in the manual for each board at:

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

8.4. Power

In the sensor connector there are also several power pins, specifically GND, SENSOR POWER, 5V SENSOR POWER and GPS POWER.

- **SENSOR POWER**: 3.3V power voltage (200 mA maximum) which is controlled from the Waspmote execution code.
- **5V SENSOR POWER**: 5V power voltage (200 mA maximum) which is controlled from the Waspmote execution code.
- **GPS POWER**: 3.3V power voltage (200mA maximum) which is controlled from the Waspmote execution code

9. 802.15.4/ZigBee

Waspmote integrates the Digi **XBee** modules for communication in **the ISMB** (Industrial Scientific Medical Band) bands.

These modules communicate with the microcontroller using the UART_0 and UART_1 at 115200bps.

There are 7 possible XBee modules distributed by Libelium for integration in Waspmote.

Model	Protocol	Frequency	txPower	Sensitivity	Range *
XBee-802.15.4	802.15.4	2.4GHz	1mW	-92dB	500m
XBee-802.15.4-Pro	802.15.4	2.4GHz	100mW	-100dBm	7000m
XBee-ZB	ZigBee-Pro	2.4GHz	2mW	-96dBm	500m
XBee-ZB-Pro	ZigBee-Pro	2.4GHz	50mW	-102dBm	7000m
XBee-868	RF	868MHz	315mW	-112dBm	12km
XBee-900	RF	900MHz	50mW	-100dBm	10Km

** Line of sight and Fresnel zone clearance with 5dBi dipole antenna*

These modules have been chosen for their high receiving sensitivity and transmission power, as well as for being 802.15.4 compliant (XBee-802.15.4 model) and ZigBee-Pro v2007 compliant (XBee-ZB model).

The XBee modules integrated in Waspmote include **RPSMA** antenna connectors.

9.1. XBee-802.15.4

Module	Frequency	TX power	Sensitivity	Channels	Distance
Normal	2,40 – 2,48GHz	1mW	-92dBm	16	500m
PRO	2,40 – 2,48GHz	63.1mW	-100dBm	13	7000m

Figure 47: XBee 802.15.4

Figure 48: XBee 802.15.4 PRO

The frequency used is the free band of 2.4GHz, using 16 channels with a bandwidth of 5MHz per channel.

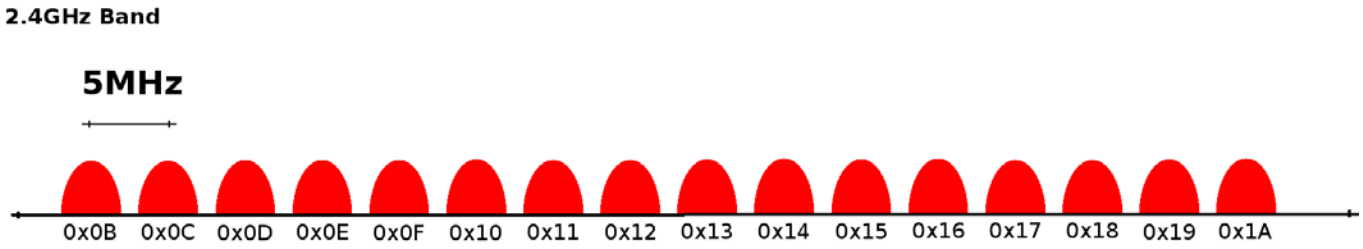


Figure 49: Frequency channels in the 2.4GHz band

Channel Number	Frequency	Supported by
0x0B – Channel 11	2,400 – 2,405 GHz	Normal
0x0C – Channel 12	2,405 – 2,410 GHz	Normal / PRO
0x0D – Channel 13	2,410 – 2,415 GHz	Normal / PRO
0x0E – Channel 14	2,415 – 2,420 GHz	Normal / PRO
0x0F – Channel 15	2,420 – 2,425 GHz	Normal / PRO
0x10 – Channel 16	2,425 – 2,430 GHz	Normal / PRO
0x11 – Channel 17	2,430 – 2,435 GHz	Normal / PRO
0x12 – Channel 18	2,435 – 2,440 GHz	Normal / PRO
0x13 – Channel 19	2,440 – 2,445 GHz	Normal / PRO
0x14 – Channel 20	2,445 – 2,450 GHz	Normal / PRO
0x15 – Channel 21	2,450 – 2,455 GHz	Normal / PRO
0x16 – Channel 22	2,455 – 2,460 GHz	Normal / PRO
0x17 – Channel 23	2,460 – 2,465 GHz	Normal / PRO
0x18 – Channel 24	2,465 – 2,470 GHz	Normal
0x19 – Channel 25	2,470 – 2,475 GHz	Normal
0x1A – Channel 26	2,475 – 2,480 GHz	Normal

Figure 50: Channels used by the XBee modules in 2.4GHz

The XBee 802.15.4 modules comply with the standard **IEEE 802.15.4** which defines the physical level and the link level (MAC layer). The XBee modules add certain functionalities to those contributed by the standard, such as:

- **Node discovery:** certain information has been added to the packet headers so that they can discover other nodes on the same network. It allows a node discovery message to be sent, so that the rest of the network nodes respond indicating their data (Node Identifier, @MAC, @16 bits, RSSI).
- **Duplicated packet detection:** This functionality is not set out in the standard and is added by the XBee modules.

With a view to obtain frames totally compatible with the IEEE802.15.4 standard and enabling inter-operability with other chipsets, the `XBee.setMacMode(m)` command has been created to select at any time if the modules are to use a totally compatible heading format, or conversely enable the use of extra options for node discovery and duplicated packets detection.

Encryption is provided through the **AES 128b** algorithm. Specifically through the **AES-CTR type.** In this case the Frame Counter field has a unique ID and encrypts all the information contained in the **Payload** field which is the place in the 802.15.4 frame where data to be sent is stored.

The way in which the libraries have been developed for the module programming makes encryption activation as simple as running the initialization function and giving it a key to use in the encryption process.

```
{
 xbee802.setEncryptionMode(1);
 xbee802.setLinkKey(key);
}
```

Extra information about the encryption systems in 802.15.4 and ZigBee sensor networks can be accessed in the **Development section** of the Libelium website, specifically in the document: "Security in 802.15.4 and ZigBee networks"

The classic topology of this type of network is a star topology, as the nodes establish point to point connections with brother nodes through the use of parameters such as the MAC or network address.

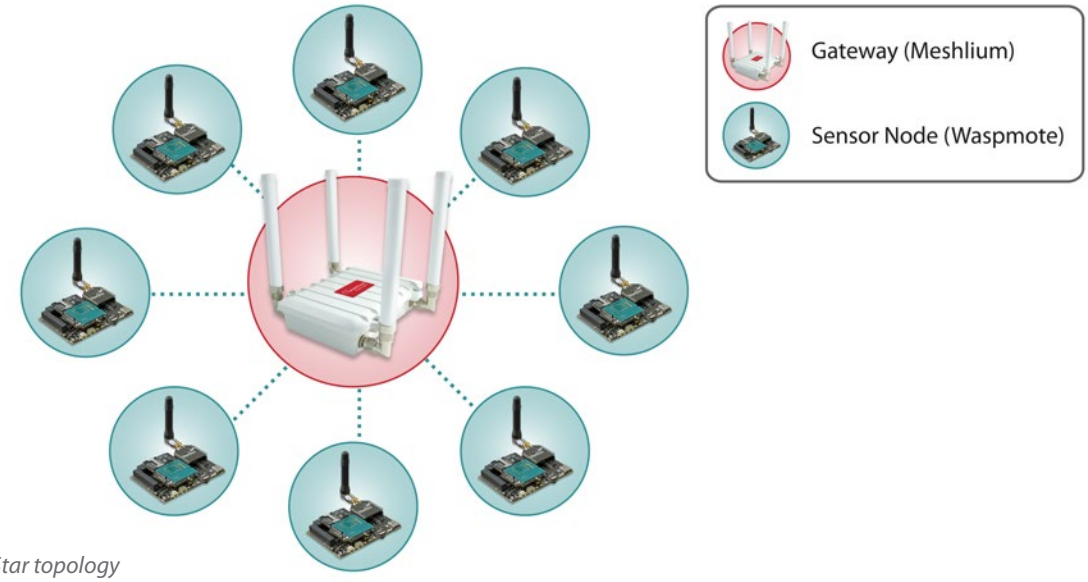


Figure 51: Star topology

Regarding the "Energy" section, the transmission power can be adjusted to several values:

Parameter	Tx XBee	Tx XBee-PRO
0	-10dBm	10dBm
1	-6dBm	12dBm
2	-4dBm	14dBm
3	-2dBm	16dBm
4	0dBm	18dBm

Figure 52: Transmission power values

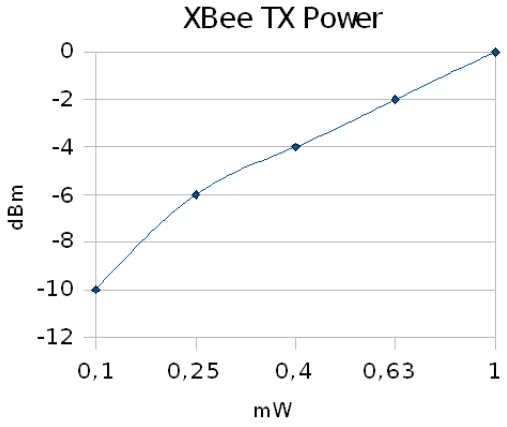


Figure 53: XBee TX Power

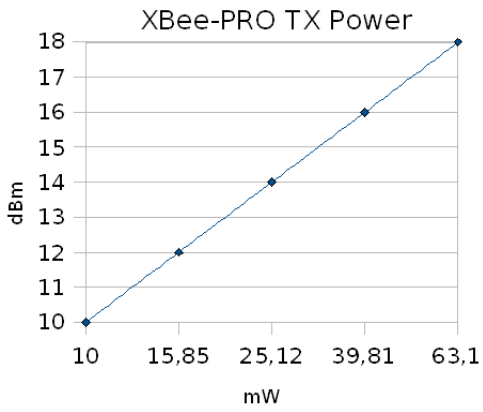


Figure 54: XBee-PRO TX Power

Related API libraries: **WaspXBeeCore.h, WaspXBeeCore.cpp, WaspXBee802.h, WaspXBee802.cpp**

All information about their programming and operation can be found in the document: **802.15.4 Networking Guide.**

All the documentation is located in the **Development section** in the Libelium website.

9.2. XBee - ZigBee

Module	Frequency	Transmission Power	Sensitivity	Number of channels	Distance
XBee-ZB	2,40 – 2,48GHz	2mW	-96dBm	16	500m
XBee-ZB-PRO	2,40 – 2,48GHz	50mW	-102dBm	13	7000m

Figure 55: XBee ZigBee

Figure 56: XBee ZigBee PRO

As ZigBee is supported in the IEEE 802.15.5 link layer, it uses the same channels as described in the previous section, with the peculiarity that the XBee-ZB-PRO model limits the number of channels to 13.

The XBee-ZB modules comply with the **ZigBee-PRO v2007** standard. These modules add certain functionalities to those contributed by ZigBee, such as:

- **Node discovery:** some headings are added so that other nodes within the same network can be discovered. It allows a node discovery message to be sent, so that the rest of the network nodes respond indicating their specific information (Node Identifier, @MAC, @16 bits, RSSI).
- **Duplicated packet detection:** This functionality is not set out in the standard and is added by the XBee modules.

The topologies in which these modules can be used are: star and tree.

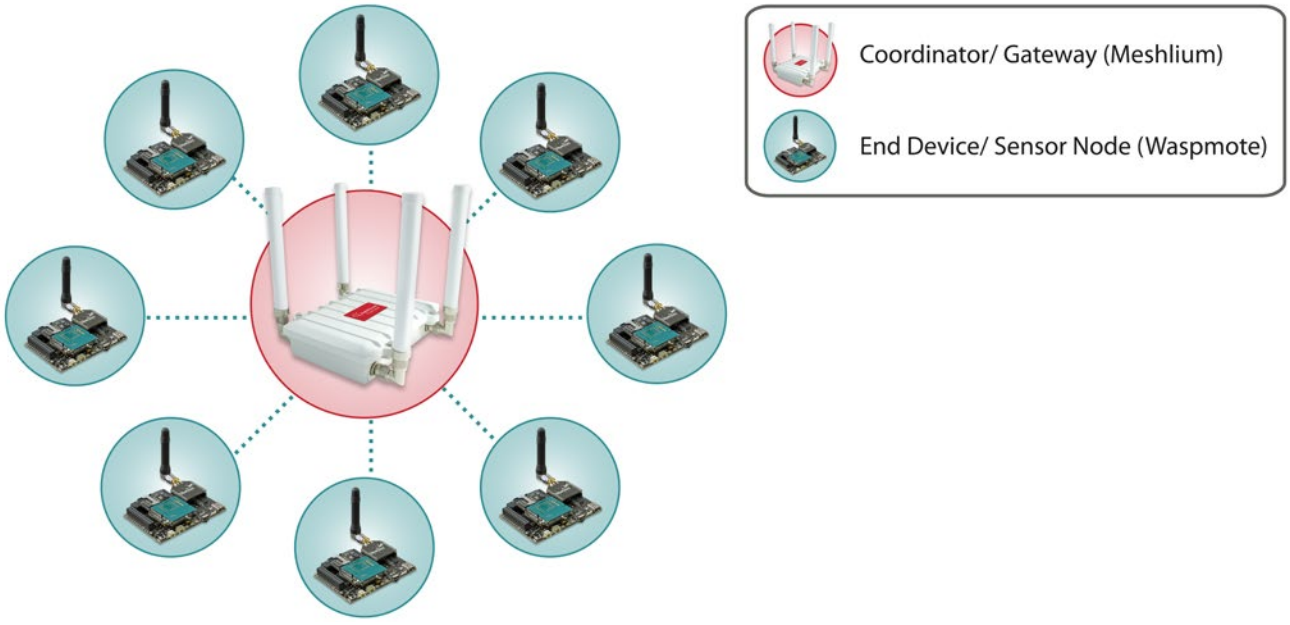


Figure 57: Star topology

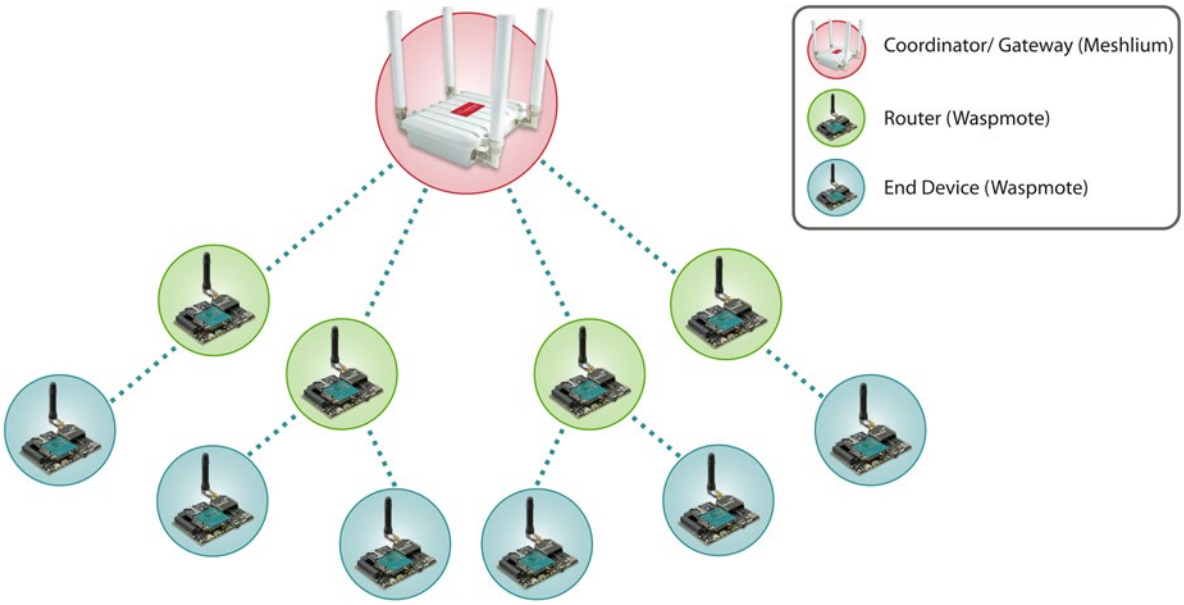


Figure 58: Tree topology

Regarding the "Energy" section, the transmission power can be adjusted to several values:

Parameter	Tx XBee ZB
0	-8dBm
1	-4dBm
2	-2dBm
3	0dBm
4	2dBm

Figure 59: Transmission power values

It is possible to set up the boost mode to improve reception sensibility 1dB and transmission power 2dB. Using this mode improves reach, but also increases consumption.

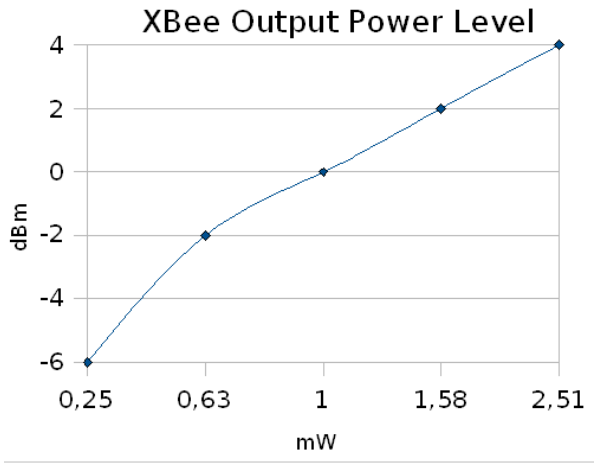


Figure 60: XBee Output Power Level (Boost Mode ON)

Related API libraries: **WaspXBeeCore.h, WaspXBeeCore.cpp, WaspXBeeZB.h, WaspXBeeZB.cpp**

All information about their programming and operation can be found in the document: **ZigBee Networking Guide.**

All the documentation is located in the **Development section** in the Libelium website.

9.3. XBee - 868

Module	Frequency	Transmission Power	Sensitivity	Channels	Distance
XBee 868	869,4 – 869,65MHz	315mW	-112dBm	1	12km

Figure 61: XBee 868

The frequency used is the 869MHz band (Europe), using 1 single channel. The use of this module is only allowed in Europe. In the chapter "Certifications", more information can be obtained about the **Certifications**.

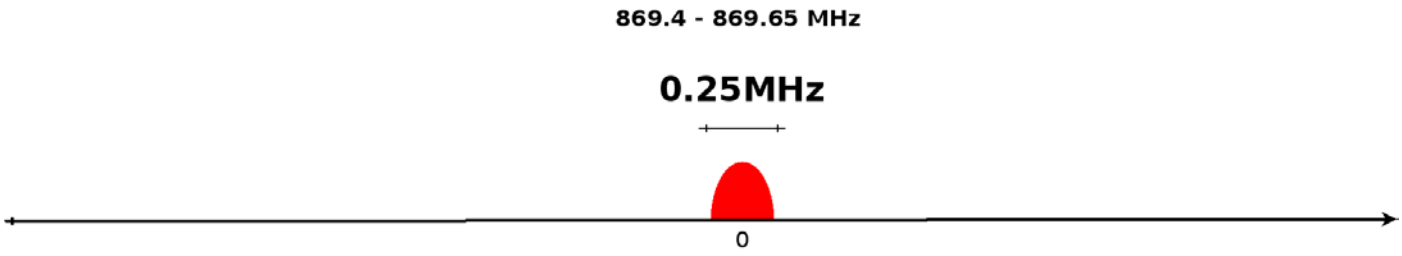


Figure 62: Channel frequency on 869MHz

Encryption is provided through the **AES 128b algorithm**. Specifically through the type **AES-CTR**. In this case the Frame Counter field has a unique ID and encrypts all the information contained in the **Payload** field which is the place in the link layer frame where the data to be sent is stored.

The way in which the libraries have been developed for module programming means that encryption activation is as simple as running the initialization function and giving it a key to use in the encryption.

```
{
 xbee868.setEncryptionMode(1);
 xbee868.setLinkKey(key);
}
```

The classic topology for this type of network is a star topology, as the nodes can establish point to point connections with brother nodes through the use of the MAC address.

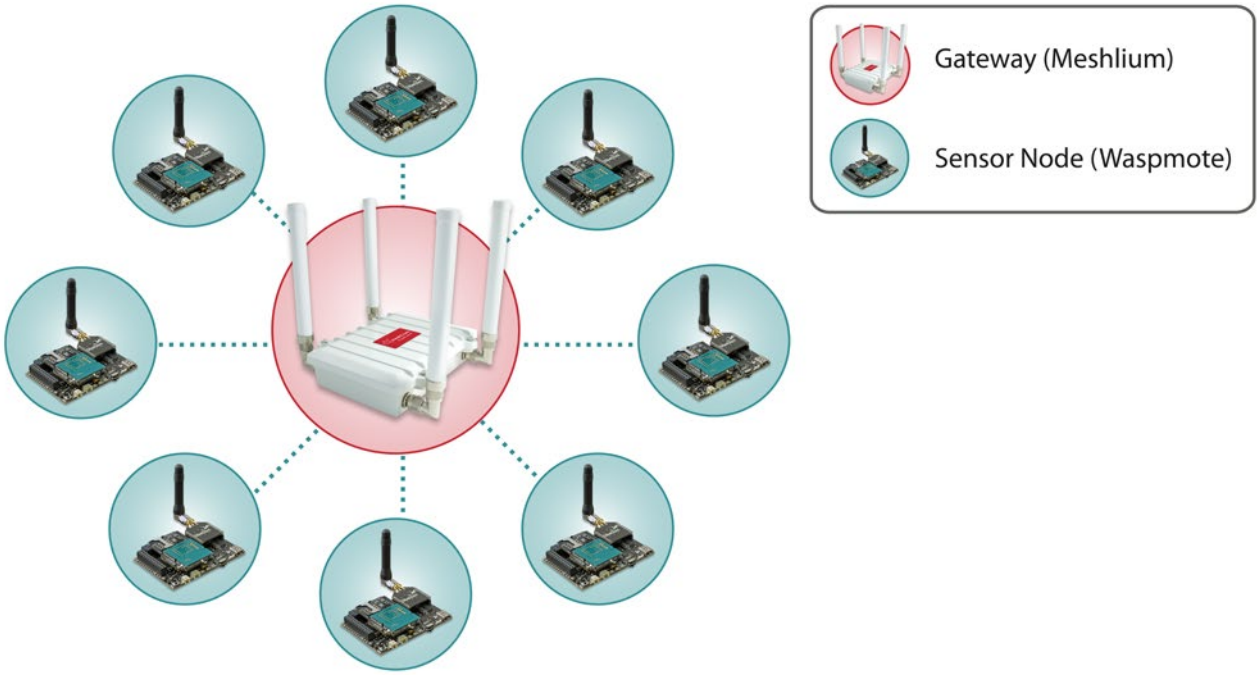


Figure 63: Star topology

Regarding the "Energy" section, the transmission power can be adjusted to several values:

Parameter	Tx XBee - 868
0	0dBm
1	13.7dBm
2	20dBm
3	22dBm
4	25dBm

Figure 64: Transmission power values

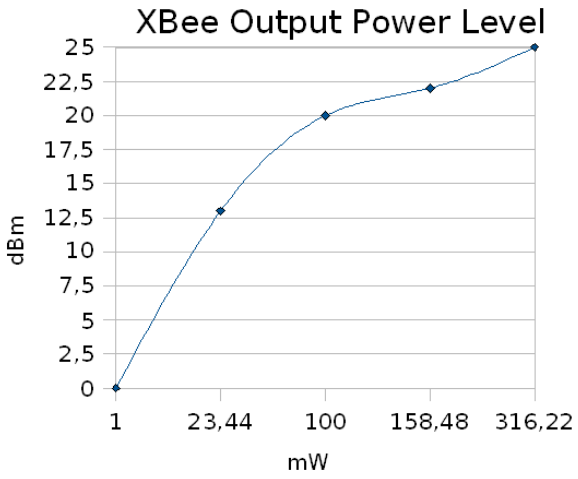


Figure 65: XBee TX Power

Related API libraries: **WaspXBeeCore.h, WaspXBeeCore.cpp, WaspXBee868.h, WaspXBee868.cpp**

All information about their programming and operation can be found in the document: **868MHz Networking Guide.**

All the documentation is located in the **Development section** in the Libelium website.

9.4. XBee - 900

Module	Frequency	Tx Power	Sensitivity	Channels	Distance
XBee 900	902-928MHz	50mW	-100dBm	12	10km

Figure 66: XBee 900MHz

The frequency used is the 900MHz band, using 12 channels with a bandwidth of **2.16MHz** per channel and a transmission rate of 156.25kbps. The use of this module is only allowed in the United States and Canada. In the chapter "Certifications", more information can be obtained about the **Certifications**.

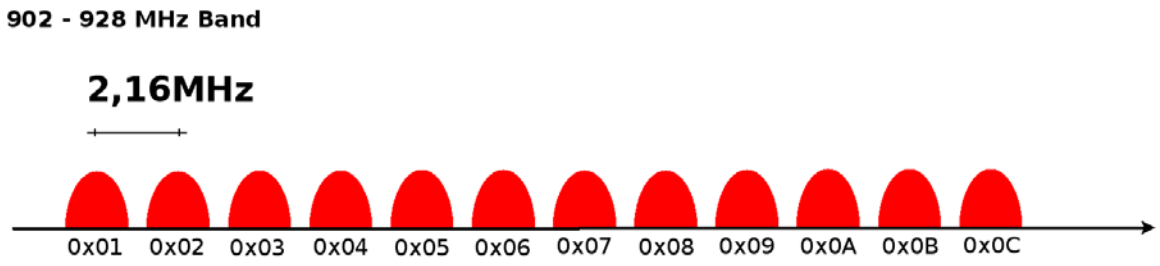


Figure 67: Channel frequencies in the 900MHz band

Encryption is provided through the **AES 128b algorithm**. Specifically through the type **AES-CTR**. In this case the Frame Counter field has a unique ID and encrypts all the information contained in the **Payload** field which is the place in the link layer frame where the data to be sent is stored.

The way in which the libraries have been developed for module programming means that encryption activation is as simple as running the initialization function and giving it a key to use in the encryption.

```
{
 xbee900.setEncryptionMode(1);
 xbee900.setLinkKey(key);
}
```

The classic topology for this type of network is a star topology, as the nodes can establish point to point connections with brother nodes through the use of parameters such as the MAC address or that of the network.

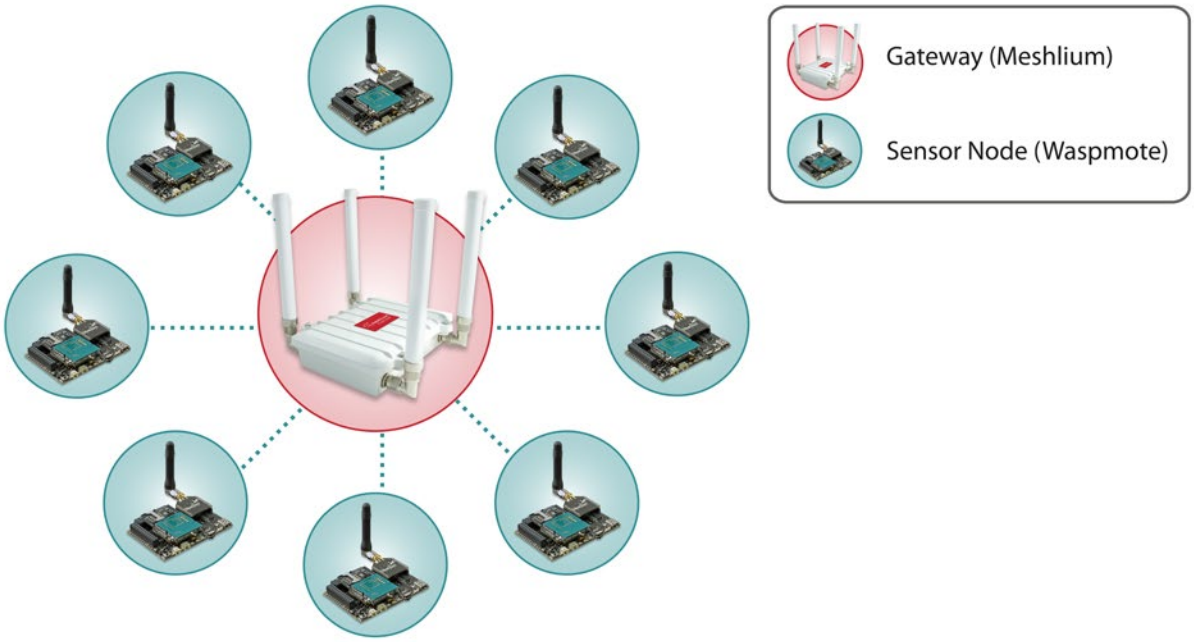


Figure 68: Star topology

API libraries: **WaspXBeeCore.h, WaspXBeeCore.cpp, WaspXBee900.h, WaspXBee900.cpp**

All information about their programming and operation can be found in the document: **900MHz Networking Guide.**

All the documentation is located in the **Development section** in the Libelium website.

9.5. XBee-DigiMesh

The XBee-802.15.4 and XBee-900 modules can use an optional firmware (**DigiMesh**) so that they can create **mesh networks** instead of the usual point to point topology. This firmware has been developed by Digi aimed for allowing modules to sleep, synchronize themselves and work on equal terms, avoiding the use of node routers or coordinators that have to be permanently powered on. Characteristics of the implemented protocol:

- **Self Healing:** any node can join or leave the network at any moment.
- **All nodes are equal:** there are no father-son relationships.
- **Silent protocol:** reduced routing heading due to using a reactive protocol similar to AODV (Ad hoc On-Demand Vector Routing).
- **Route discovery:** instead of keeping a route map, routes are discovered when they are needed.
- **Selective ACKs:** only the recipient responds to route messages.
- **Reliability:** the use of ACKs ensures data transmission reliability.
- **Sleep Modes:** low energy consumption modes with synchronization to wake at the same time.

The classic topology of this type of network is mesh, as the nodes can establish point to point connections with brother nodes through the use the MAC address doing **multi-hop connections** when it is necessary.

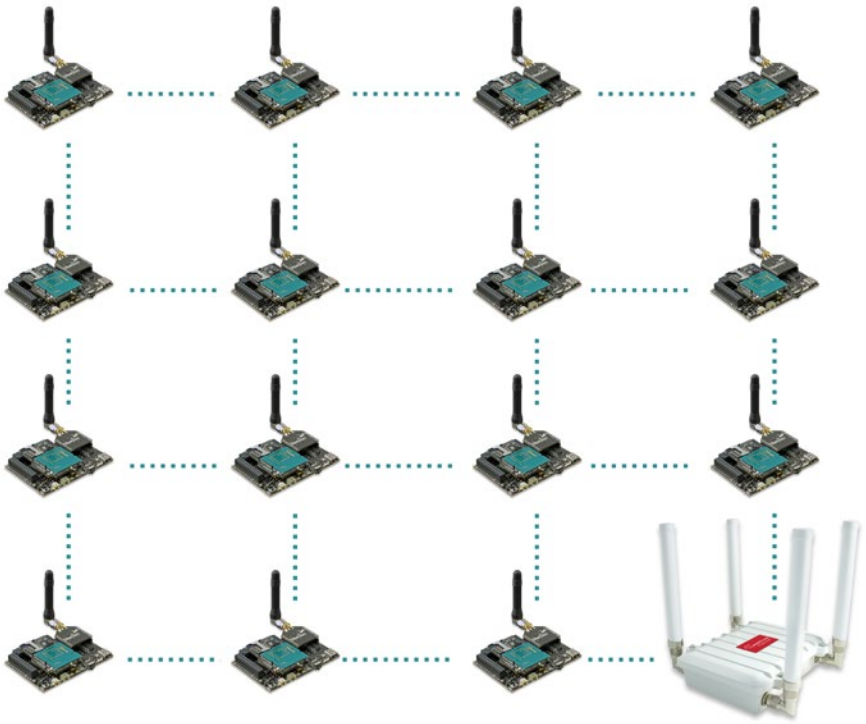


Figure 69: Mesh topology

DigiMesh 2.4GHz

Module	Frequency	Tx Power	Sensitivity	Channels	Distance
Normal	2,40 – 2,48GHz	1mW	-92dBm	16	500m
PRO	2,40 – 2,48GHz	100mW	-100dBm	13	7000m

The XBee DigiMesh modules share the hardware module with the XBee-802.15.4. So it is possible to change the firmware of this kind of modules from one to another and vice versa. For this reason, the characteristics relating to the hardware are the same, changing those related with the protocol used.

The XBee DigiMesh modules are based on the standard **IEEE 802.15.4** that supports functionalities enabling mesh topology use.

DigiMesh 900MHz

Frequency	Tx Power	Sensitivity	Channels	Distance
902-928MHz	50mW	-100dBm	12	10km

The XBee DigiMesh modules share the hardware module with the XBee-900. So it is possible to change the firmware of this kind of modules from one to another and vice versa. For this reason, the characteristics relating to the hardware are the same, changing those related with the protocol used.

Related API libraries: **WaspXBeeCore.h, WaspXBeeCore.cpp, WaspXBeeDM.h, WaspXBeeDM.cpp**

All information about their programming and operation can be found in the document: **DigiMesh Networking Guide.**

All the documentation is located in the **Development section** in the Libelium website.

9.6. RSSI

The **RSSI** parameter (Received Signal Strength Indicator) indicates the signal quality of the last packet received. The XBee modules provide this information in all protocol and frequency variants.

One of the most common functionalities in the use of RSSI is the creation of **indoor localization** systems by signal triangulation.

In Waspmote this value is obtained simply by executing the function (i.e. XBee-802.15.4):

```
{
xbee802.getRSSI();
}
```

10. WiFi

The WiFi module for the Waspmote platform completes the current connectivity possibilities enabling the direct communication of the sensor nodes with any WiFi router in the market. As well as this, this radio allows Waspmote to send directly the information to any iPhone or Android Smartphones without the need of an intermediate router, what makes possible to create WiFi sensor networks anywhere using just Waspmote and a mobile device as all of them run with batteries.

With this radio, Waspmote can make HTTP connections retrieving and sending information to the web and FTP servers in both normal and secure modes (HTTPS/FTPS), as well as using TCP/IP and UDP/IP sockets in order to connect to any server located on the Internet.

Features:

- Protocols: 802.11b/g - 2.4GHz
- TX Power: 0dBm - 12dBm (variable by software)
- RX Sensitivity: -83dBm
- Antenna connector: RPSMA
- Antenna: 2dBi/5dBi antenna options
- Security: WEP, WPA, WPA2
- Topologies: AP and Adhoc
- 802.11 roaming capabilities

Actions:

- TCP/IP - UDP/IP socket connections
- HTTP and HTTPS (secure) web connections
- FTP and FTPS (secure) file transfers
- Direct connections with iPhone and Android
- Connects with any standard WiFi router
- DHCP for automatic IP assignation
- DNS resolution enabled

Figure 70: WiFi module with 2dBi and 5dBi antennas

Related API libraries: **WaspWifi.h, WaspWifi.cpp**

All information about their programming and operation can be found in the document: **WiFi Networking Guide.**

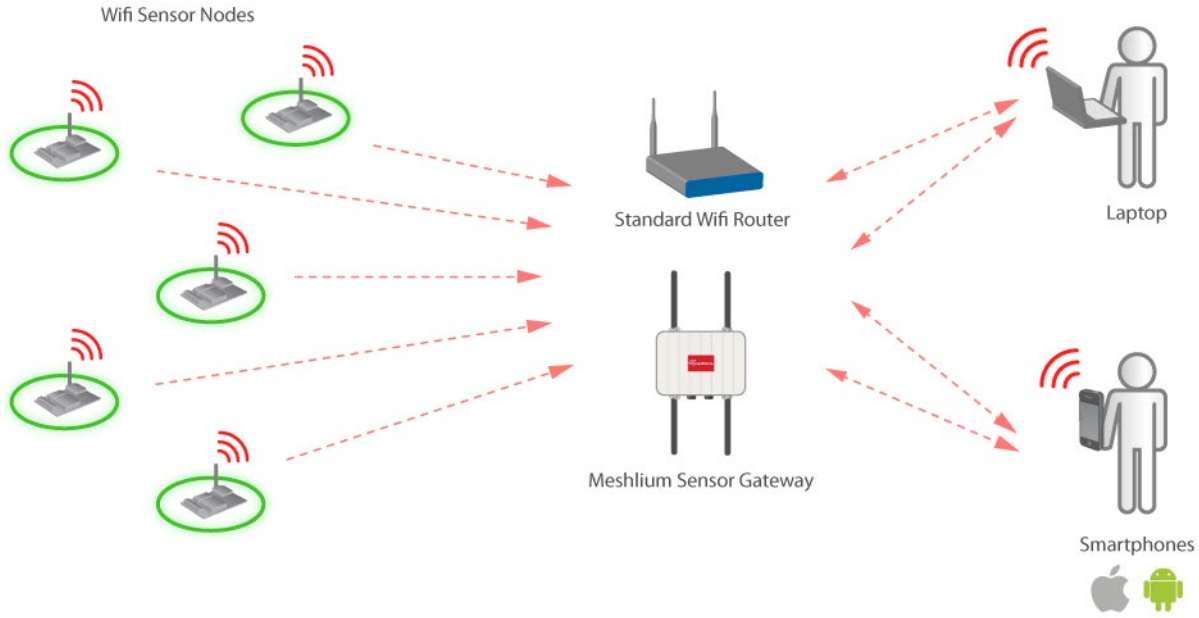
All the documentation is located in the **Development section** in the Libelium website.

10.1. WiFi Topologies

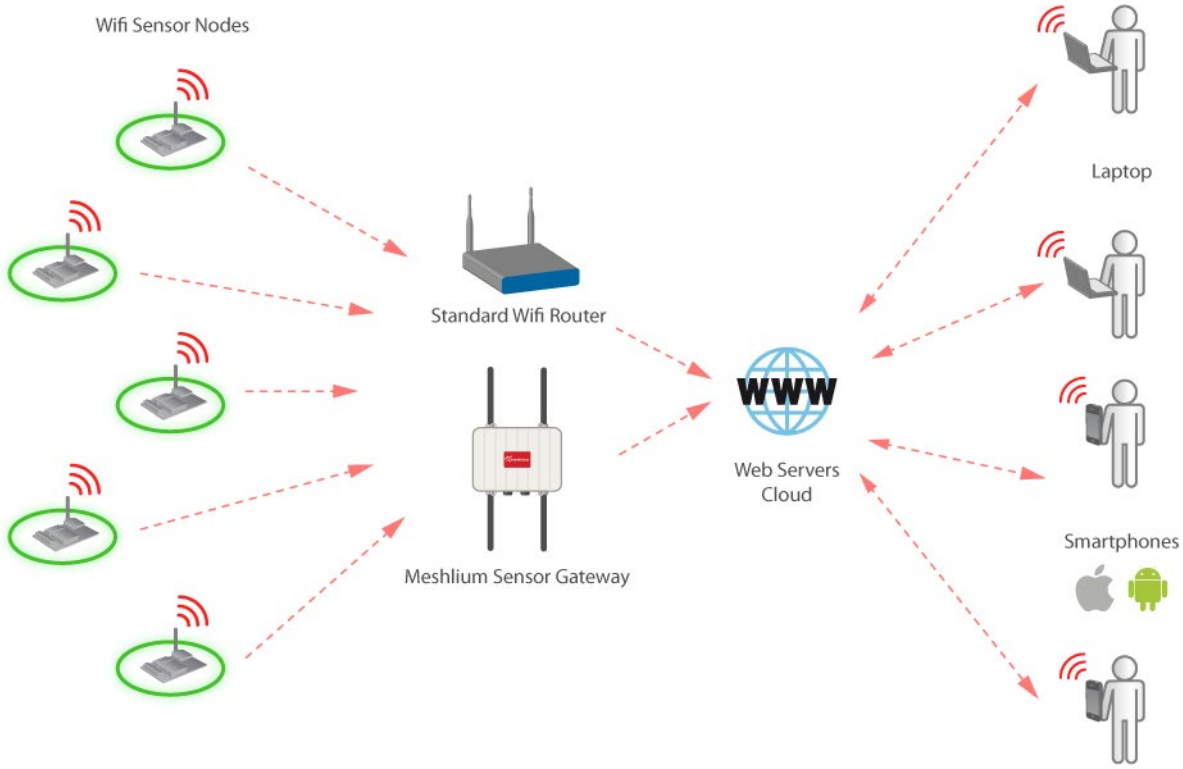
10.1.1. Access Point

Sensor nodes may connect to any standard WiFi router which is configured as Access Point (AP) and then send the data to other devices in the same network such as laptops and smartphones. This is the common case when implementing home sensor networks and when using the data inside an Intranet.

Once associated with the Access Point, the nodes may ask for an IP address by using the DHCP protocol or use a preconfigured static IP. The AP connection can be encrypted, in this case, you have to specify also the pass-phrase or key to the WiFi module. The WiFi module supports these security modes: WEP-128, WPA2-PSK , WPA1-PSK, and WPA-PSK mixed mode.



Nodes may also connect to a standard WiFi router with DSL or cable connectivity and send the data to a web server located on the Internet. Then users are able to get this data from the Cloud. This is the typical scenario for companies which want to give data accessibility services.



As pointed before the WiFi module can join any standard WiFi router, however the connection may also be performed using **Meshlium** instead of a standard WiFi router. Meshlium is the multiprotocol router designed by Libelium which is specially recommended for outdoor applications as it is designed to resist the hardest conditions in real field deployments.

When is recommended to use Meshlium instead a standard WiFi router?

As pointed before the new WiFi module for Waspmote can connect to any standard WiFi router (*"home oriented"*) in the market. However when deploying sensor networks outdoors you need a robust machine capable of resist the hardest conditions of rain, wind, dust, etc. Meshlium is specially designed for real deployments of wireless sensor networks as it is waterproof (IP-65) and counts with a robust metallic enclosure ready to resist the hardest atmospheric conditions.

Meshlium is also ready to deal with hundreds of nodes at the same time, receiving sensor data from all of them and storing it in its internal database or sending it to an Internet server. As well as this, Meshlium may work as a WiFi to 3G/GPRS gateway, giving access to the internet to all the nodes in the network using the mobile phones infrastructure.

It is also important to mention that the transmission power of the WiFi interface integrated in Meshlium is many times higher than the ones available in *"home oriented"* WiFi routers so the distance we can get increases dramatically from a few meters to dozens or even hundreds depending on the location of the nodes.

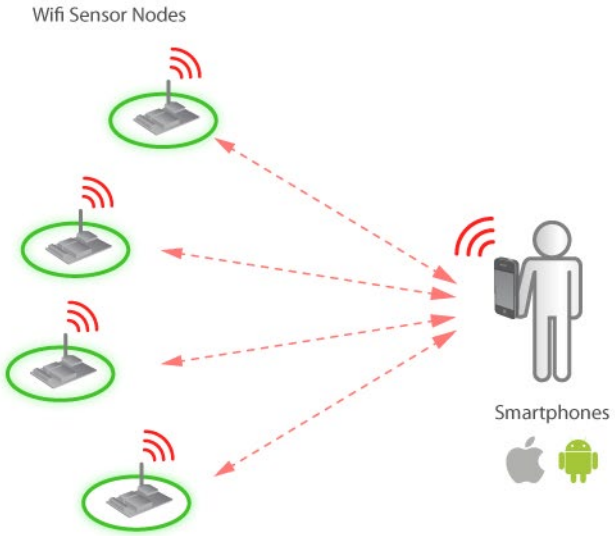
Using Meshlium as WiFi Access Point allows to control and to store the messages received from the WiFi module, or allows to combine WiFi technology with other protocols such as ZigBee. Meshlium may work as:

- a ZigBee to Ethernet router for Waspmote nodes
- a ZigBee to 3G/GPRS router for Waspmote nodes
- a WiFi Access Point
- a WiFi Mesh node (dual band 2.4GHz-5GHz)
- a WiFi to 3G/GPRS router
- a Bluetooth scanner and analyzer
- a GPS-3G/GPRS real time tracker
- a SmartPhone scanner (detects iPhone and Android devices)

For more information about Meshlium go to: **http://www.libelium.com/meshlium**

10.1.2. Ad-hoc mode with iPhone/Android

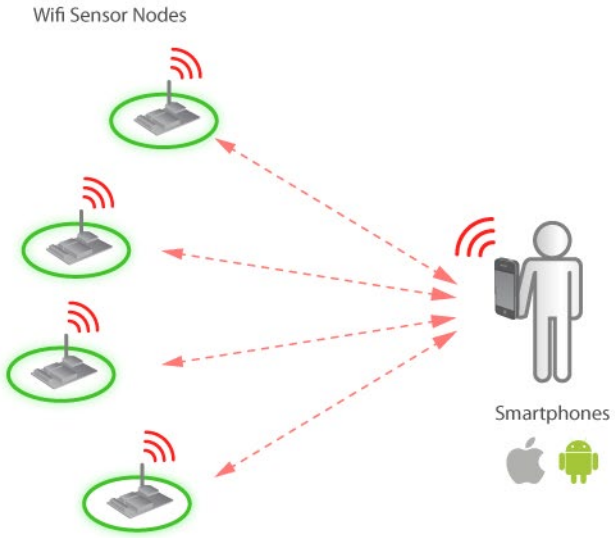
The following diagram shows how Android and iPhone devices can communicate directly with the WiFi integrated in Waspmote through an Adhoc WiFi network without any extra router or gateway.



More information about this topology in the next section "WiFi" → Connecting to a Smartphone directly.

10.2. Connecting to a Smartphone directly

The new WiFi radio may perform direct communications with iPhone and Android devices without the need of an intermediate router by creating an Adhoc network between them. This is useful when testing the network or for quick deployments where no access points are required.



We have developed the application ***Waspmote WiFi***, for both iPhone and another for Android platforms. The application may be downloaded from the official app markets or from the Libelium website for free: **http://www.libelium.com/apps**

Official app markets URL's:

- iphone: **http://itunes.apple.com/us/app/waspmote-wifi/id501974230**
- Android: **https://market.android.com/details?id=com.libelium.WIFI_module2**

10.2.1. Connecting to an iPhone

10.2.1.1. Installation

a) Download the application from **App Store:**

- From the iPhone, go to App Store. Go to Search screen and search "Waspmote WiFi".
- Select Waspmote WiFi app. Press FREE button, and then INSTALL button.
- Accept the rights and then the app will appear in your iPhone screen.
- You can do the same from the Computer with iTunes. Open iTunes and search "Waspmote WiFi": **http://itunes.apple.com/us/app/waspmote-wifi/id501974230**
- Save the app in iTunes and synchronize it with your iPhone or iPod.

b) Download the application (WaspmoteWifi.ipa) from the Libelium website: **http://www.libelium.com/apps**

- Then double click on the icon, or right click and open with iTunes.
- Inside iTunes, on the left panel, click on DEVICES → Your Device.
- Select on the top "Apps", and select Sync Apps. Drag into the desired screen Waspmote WiFi app.

Once installed, the app appears in your iPhone/iPod screen:

10.2.1.2. iPhone App tutorial

The use of the app is very simple, first you have to connect to one of the Waspmote nodes selecting it in Settings → Wi-Fi, and then launch the application.

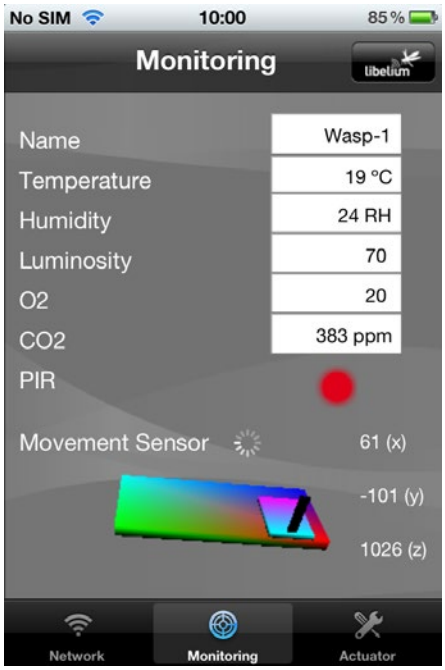
To connect to the network created by the WiFi module of Waspmote: Go to Settings → Wi-Fi and select iPHONE_ADHOC.

Once connected, you will see a blue WiFi icon in the top of the screen:

Inside the app, the first tab "Network" shows the information of the connection:

- Name of the Adhoc network
- MAC address of the node acting as gateway
- Connection status

The Monitoring tab shows the information the nodes are sending which contains the sensor data gathered. As an example some parameters are shown: temperature, humidity, luminosity, O_2, CO_2, PIR and a 3D model of Waspmote that moves with the accelerometer information in real time:



Finally, in the Actuator tab, there are three switches to send ON and OFF commands and a fader to control the exact value sent. In the Libelium website there is a video which shows how the application works with a set of lights.

The Waspmote code used in this program can be downloaded from the Libelium website:
http://www.libelium.com/development/waspmote

If you are interested in the source code of the iPhone App in order to create your own software on top of it contact our Commercial Department at: **commercial@libelium.com**

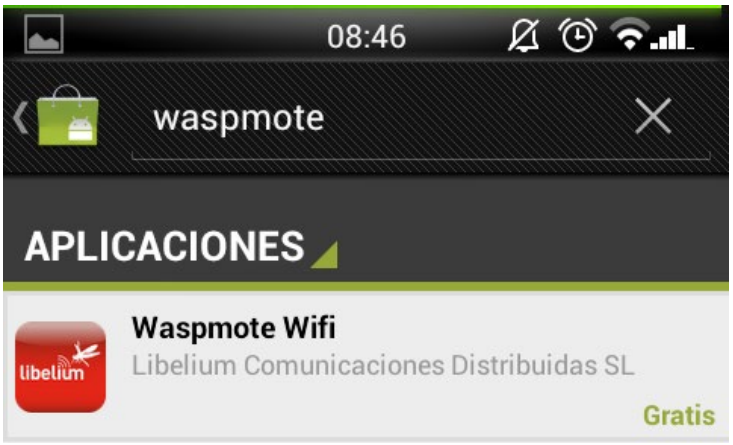
10.2.2. Connecting to an Android

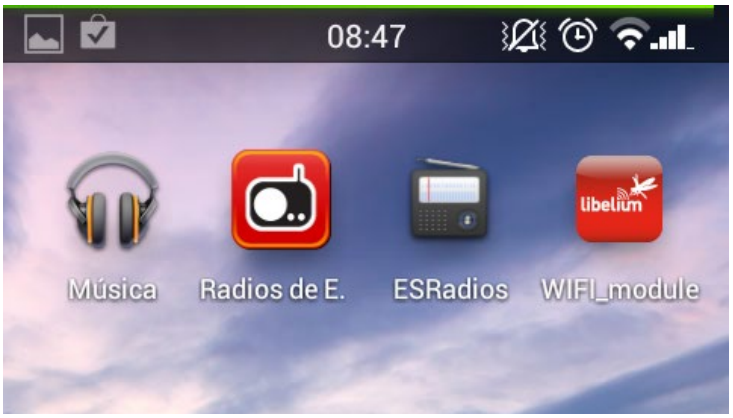
There is a feature since the version Android 2.2 or higher that allows us to create an Adhoc network and allows Waspmote to connect to it.

10.2.2.1. Installation

a) Download the application from **Android Market:**

- From the Android device, go to Android Market.
- Search "Libelium" or "Waspmote WiFi" and press enter.





b) Download the application (WaspmoteWifi.apk) from **Libelium website: http://www.libelium.com/apps**

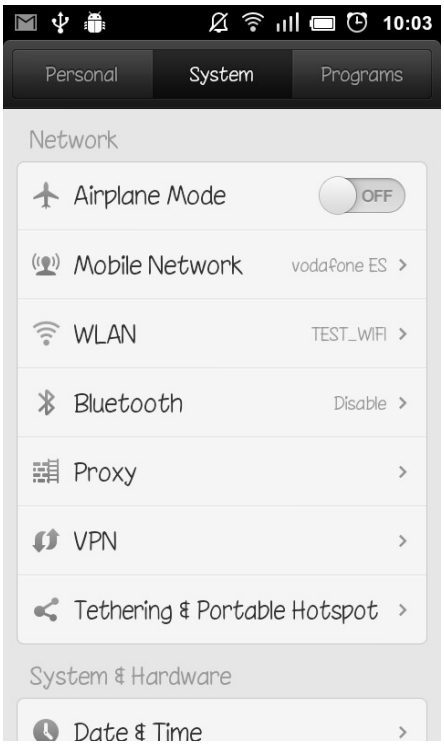
- Insert it to the SD card of your Android device.
- Then explore the SD card in your Android device and install the application. You can explore the SD card with "Astro", "ES Explora", or "File Explorer" applications.

10.2.2.2. Android App tutorial

The use of the app is very simple, first you have to create an AP from your Android device and then set Waspmote to connect to it.

To create the AP from the Android device:

Go to Settings → Tethering & Portable Hotspot or **Settings → WiFi → My WiFi Zone** (depending of the version of mobile)



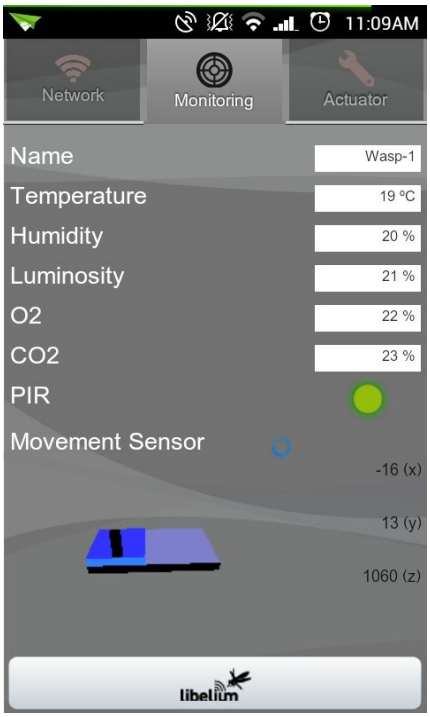
Then configure the WLAN hotspot (name= ANDROID, Security= Open). This settings you can change if you change as well the Waspmote code.

Finally, enable Portable WLAN hotspot (or My WiFi Zone), and Waspmote will connect to the Android device. Once connected, you can launch the Waspmote WiFi Demo app.

Inside the app, the first tab "Network" shows the information of the connection:

- Name of the Adhoc network
- MAC address of the node acting as gateway
- Connection status

The Monitoring tab shows the information the nodes are sending which contains the sensor data gathered. As an example some parameters are shown: temperature, humidity, luminosity, O_2, CO_2, PIR and a 3D model of Waspmote that moves with the accelerometer information in real time.



Finally, in the Actuator tab, there are three switches to send ON and OFF commands and a fader to control the exact value sent. In the Libelium website there is a video which shows how the application works with a set of lights.

The Waspmote code used in this program can be downloaded from the Libelium website:
http://www.libelium.com/development/waspmote

If you are interested in the source code of the Android App in order to create your own software on top of it contact our Commercial Department at: **commercial@libelium.com**

11. Bluetooth

The Waspmote Bluetooth module uses the same socket as the XBee does. This means you can change the XBee module for the Bluetooth module as they are pin to pin compatible.

11.1. Technical specifications

- Bluetooth v2.1 + EDR. Class 2
- TX Power: 3dBm
- Antenna: 2dBi
- Up to 250 unique devices in each inquiry
- Received Strength Signal Indicator (RSSI) for each scanned device
- Class of Device (CoD) for each scanned device
- 7 Power levels [-27dBm, +3dBm]
- Scan devices with maximum inquiry time
- Scan devices with maximum number of nodes
- Scan devices looking for a certain user by MAC address
- Classification between pedestrians and vehicles

Figure 71: Libelium Bluetooth module

Waspmote may integrate a Bluetooth module for communication in the 2.4GHz **ISMB** (Industrial Scientific Medical Band) band.

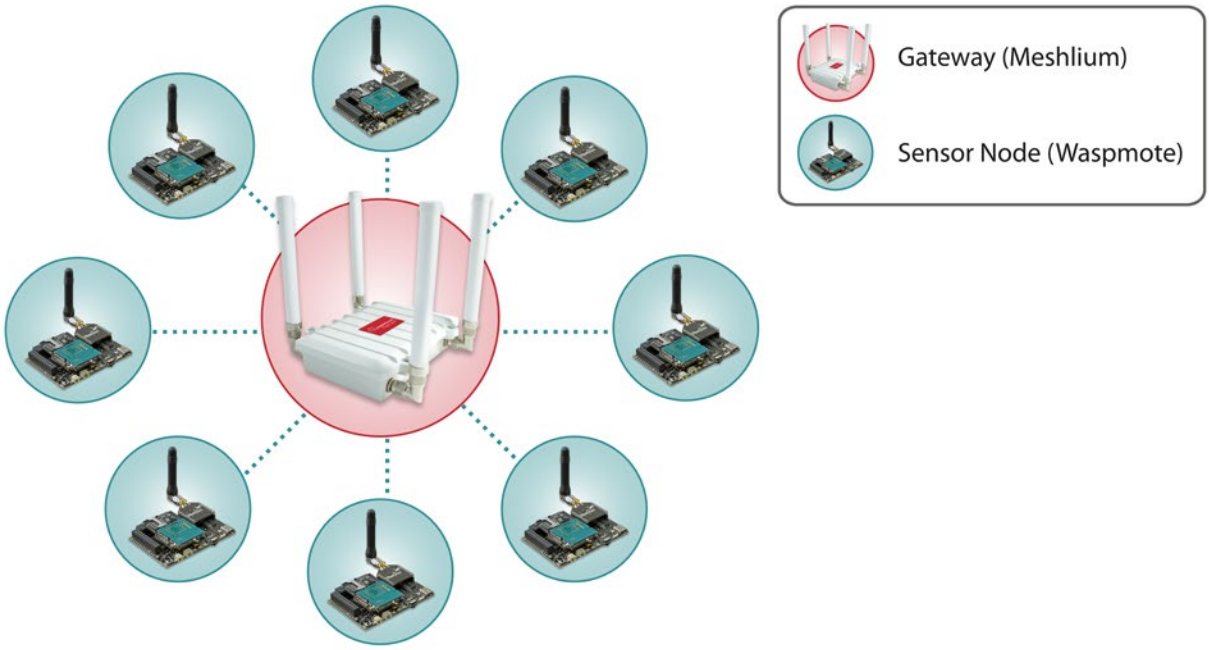


Figure 72: Start topology

Bluetooth uses 79 channels with a bandwidth of 1MHz per channel. In addition, Adaptive Frequency Hopping (AFH) is used to enhance the transmissions.

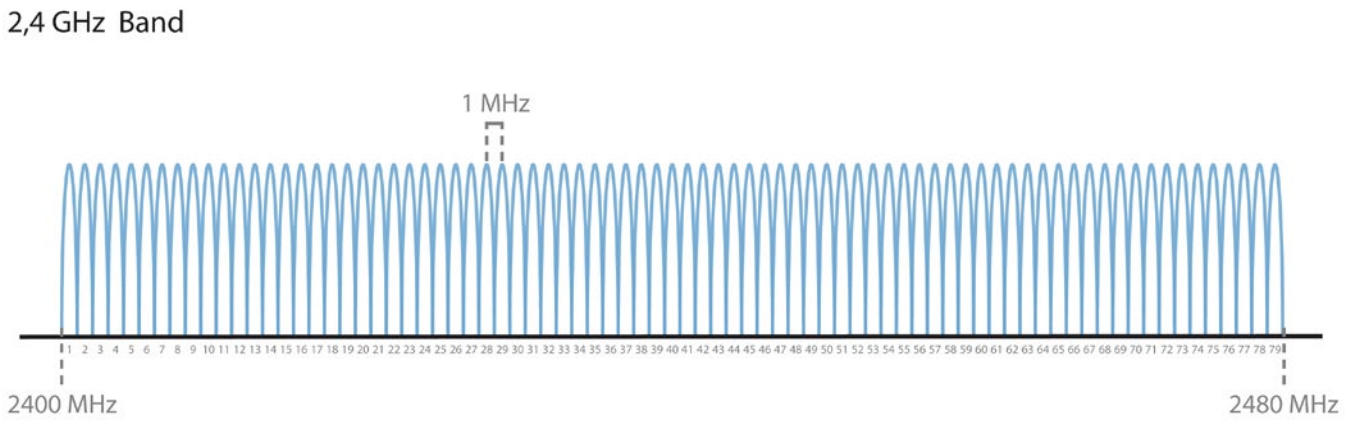


Figure 73: Frequency channels in the 2.4GHz band

Bluetooth modules have some important parameters for their configuration:

- **MAC address:** It is the unique identification number of the Bluetooth device. It has 12 hexadecimal digits separated by ":". One example could be "12:34:56:aa:bb".
- **Public Name:** It is the name that appears when a scan is performed in order to find new devices.
- **Class of Device (CoD):** Bluetooth devices are classified according to the device which they are integrated. Therefore a vehicle hands free device will belong to a different class than a pedestrian mobile phone. This parameter has 6 hexadecimal digit and it allows distinguish if the detected Bluetooth device is a vehicle, a pedestrian, and so on.
- **RSSI (Received Signal Strength Indicator):** This parameters shows quality of the radio link. It can be used to know the distance between the Bluetooth module and the inquired device. It is shown as a negative value between -40 dBm (close devices) and -90 dBm (far devices).

11.2. Bluetooth module for device discovery

The Bluetooth radio module has been specifically designed in order to scan up to 250 devices in a single inquiry. The main purpose is to be able to detect as many Bluetooth users as possible in the surrounding area.

How do we differentiate if the Bluetooth device is a car's hands-free or a mobile phone?

In the scanning process each Bluetooth device gives its "Class of Device" (CoD) attribute which allows to identify the type of service it gives. We can differentiate easily the CoD's generated by the car's handsfree from the people's phone ones.

How do I control the inquiry area?

There are seven different power levels which go from -27dBm to +3dBm in order to set different inquiry zones from 10 to 50m. These zones can also be increased or decreased by using a different antenna for the module as it counts with an standard SMA connector. The default antenna which comes with the module has a gain of 2dBi.

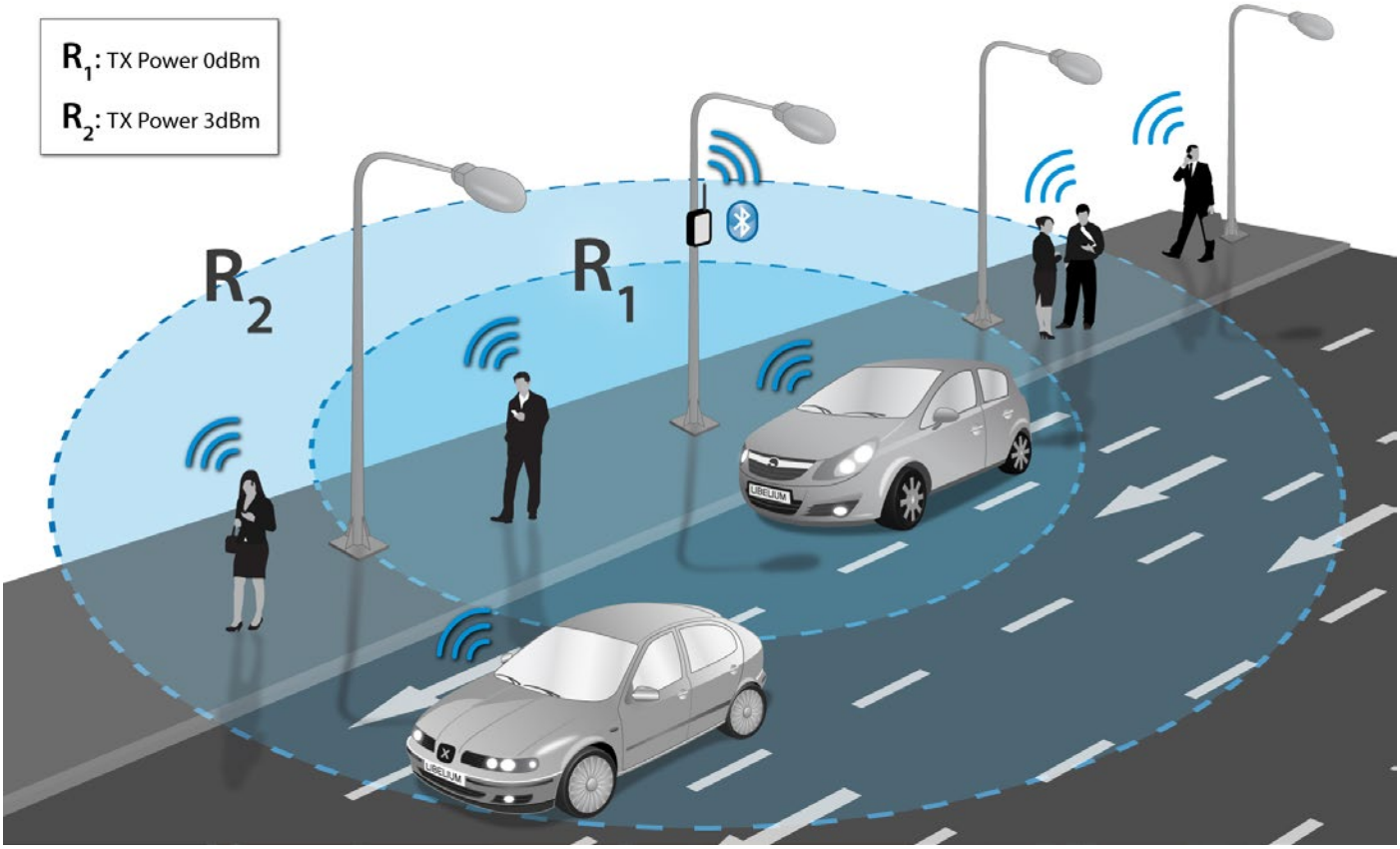


Figure 74: Example of TX power levels

How do I calculate the distance of any of the devices detected?

In the inquiry process we receive the MAC address of the Bluetooth device, its CoD and the Received Signal Strength Indicator (RSSI) which gives us the quality of the transmission with each device. RSSI values usually go from -40dBm (nearest nodes) to -90dBm (farthest ones). In the tests performed Bluetooth devices at a distance of 10m reported -50dBm as average, while the ones situated at 50m gave us an average of -75dBm.

How do the Bluetooth and ZigBee radios coexist without causing interferences with each other?

ZigBee and Bluetooth work in the 2.4GHz frequency band (2.400 - 2.480MHz), however, the Bluetooth radio integrated in Waspmote uses an algorithm called Adaptive Frequency Hopping (AFH) which improves the common algorithm used by Bluetooth (FHSS) and enables the Bluetooth radio to dynamically identify channels already in use by ZigBee and WiFi devices and to avoid them.

Can I use this radio to connect to other Bluetooth devices?

No. The idea is to use this radio "as a sensor". All the API functions developed are thought to detect as many Bluetooth devices as possible. In order to comunicate with other Bluetooth devices another module is available for the Waspmote platform. For further information read the chapter "Bluetooth" about the Communication Bluetooth Module.

What about privacy?

The anonymous nature of this technique is due to the use of MAC addresses as identifiers. MAC addresses are not associated with any specific user account or mobile phone number not even to any specific vehicle. Additionally, the "inquiry mode" (visibility) can be turned off so people have always chosen if their device will or wont be detectable.

Related API libraries: **WaspBT_Pro.h, WaspBT_Pro.cpp**

All information on their programming can be found in document: Bluetooth for device discovery **Networking Guide**.

All the documentation is located in the **Development section** in the Libelium website.

Note: *If you want to detect iPhone and Android devices using the WiFi interface as well as the Bluetooth radio go to the "Smartphone Detection" section in the Meshlium website:* ***http://www.libelium.com/meshlium***

12. GSM/GPRS

Waspmote can integrate a GSM (Global System for Mobile communications) / GPRS (General Packet Radio Service) module to enable communication using the mobile telephone network.

- **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

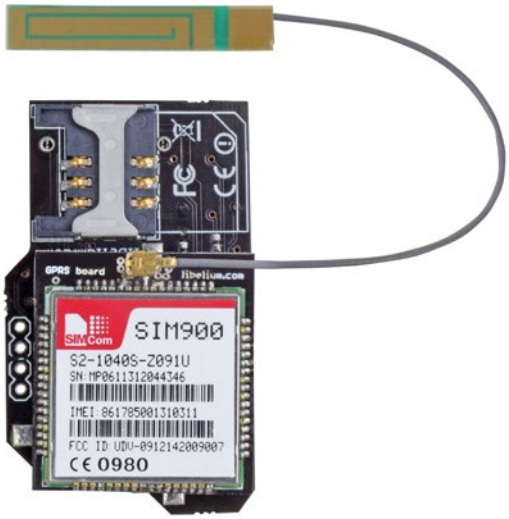


Figure 75: GSM/GPRS module

This module can carry out the following tasks:

- Making/Receiving calls
- Making 'x' second lost 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)

The functions implemented in the API allow to send information in a simple way, calling functions such as:

```
{
  GPRS_Pro.sendSMS(message, number);
  GPRS_Pro.makeLostCall(number, timeCall);
}
```

This model uses the UART_1 at a baudarte of 57600bps speed to communicate with the microcontroller.

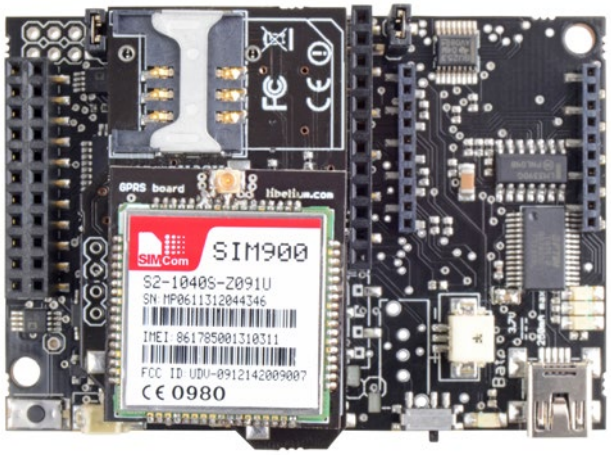


Figure 76: GSM/GPRS module in Waspmote

Related API libraries: **WaspGPRS_Pro.h**, **WaspGPRS_Pro.cpp** and **WaspGPRS_Proconstans.h**

All information about their programming and operation can be found in the document: **GSM/GPRS Programming Guide**.

All the documentation is located in the **Development section** in the Libelium website.

**** Note 1:** Battery must be connected when using this module (USB power supply is not enough).*

13. 3G + GPS

Waspmote can integrate a UMTS (Universal Mobile Telecommunication System based in WCDMA technology) / GPRS (General Packet Radio Service) module to enable communication using the 3G/GPRS mobile telephone network.

- Model: SIM5218E (SIMCom)
- Tri-Band UMTS 2100/1900/900MHz
- Quad-Band GSM/EDGE, 850/900/1800/1900 MHz
- HSDPA up to 7.2Mbps
- HSUPA up to 5.76Mbps
- TX Power:
  - UMTS 900/1900/2100 0,25W
  - GSM 850MHz/900MHz 2W
  - DCS1800MHz/PCS1900MHz 1W
- Sensitivity: -106dBm
- Antenna connector: UFL
- External Antenna: 0dBi

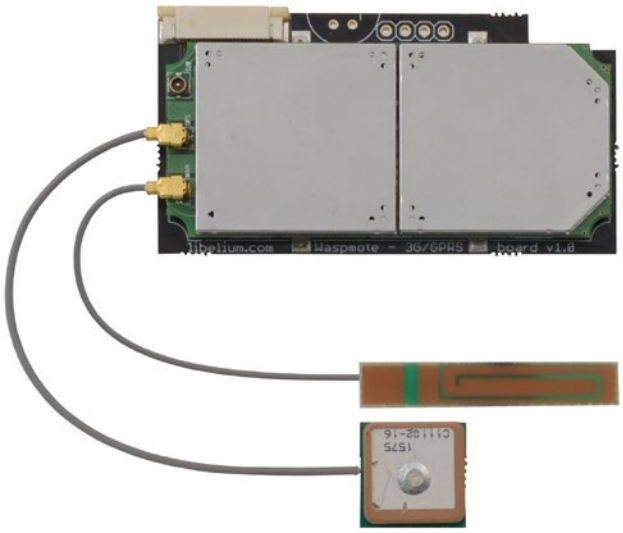


Figure 77: 3G/GPRS module

This module can carry out the following tasks:

- WCDMA and HSPA 3G networks compatibility
- Videocall using 3G network available with Video Camera Sensor Board
- Record video (res. 320 x 240) and take pictures (res. 640 x 480) available with Video Camera Sensor Board
- Support microSD card up to 32GB
- 64MB of internal storage space
- Making/Receiving calls
- Making 'x' -second lost calls
- MS-assisted (A-GPS), MS-based (S-GPS) or Stand-alone GPS positioning
- Sending/Receiving SMS
- Single connection and multiple connections TCP/IP and UDP/IP clients
- TCP/IP server
- HTTP and HTTPS service
- FTP and FTPS Service (downloading and uploading files)
- Sending/receiving email (SMTP/POP3)

The functions implemented in the API allow to send information in a simple way, calling functions such as:

```
{
  _3G.sendSMS(message, number);
  _3G.makeLostCall(number, timeCall);
}
```

This model uses the UART_1 at a baudrate of 115200 speed to communicate with the microcontroller.

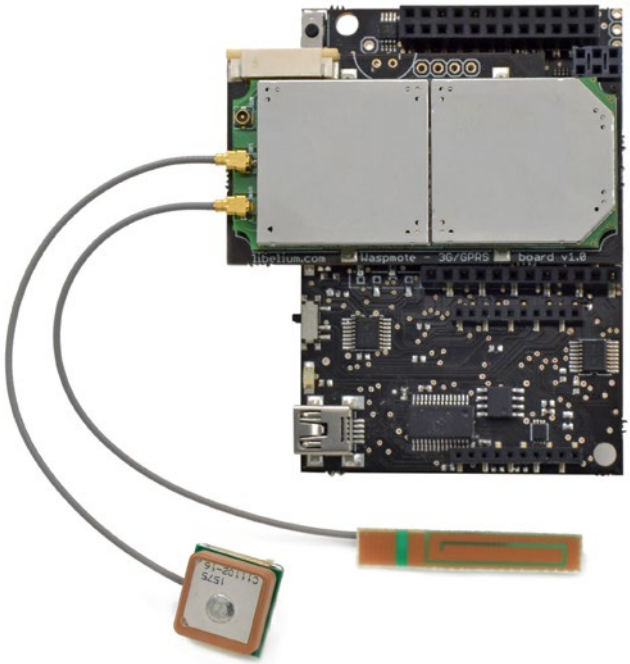


Figure 78: 3G/GPRS module in Waspmote

Related API libraries: **Wasp3G.h, Wasp3G.cpp**

All information about programming and operation can be found in the document: **3G + GPRS Networking Guide**.

All the documentation is located in the **Development section** of Libelium website.

**** Note 1:** Battery must be connected when using this module (USB power supply is not enough).*

14. RFID/NFC

13.56MHz

- **Compatibility:** Reader/writer mode supporting ISO 14443A / MIFARE / FeliCaTM / NFCIP-1
- **Distance:** 5cm
- **Max capacity:** 4KB
- **Tags:** cards, keyrings, stickers

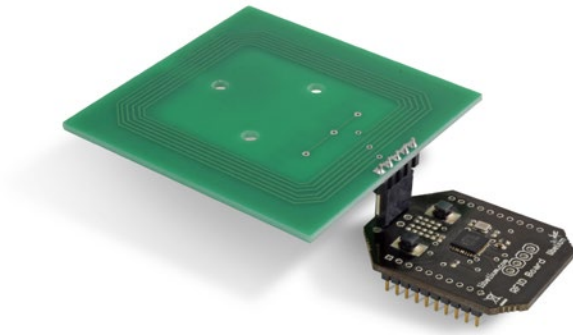


Figure 79: 13.56MHz RFID/NFC module

Applications:

- Located based services (LBS)
- Logistics (assets tracking, supply chain)
- Access management
- Electronic prepaid metering (vending machines, public transport)
- Smartphone interaction (NFCIP-1 protocol)

125KHz

- **Compatibility:** Reader/writer mode supporting ISO cards - T5557 / EM4102
- **Distance:** 5cm
- **Max capacity:** 20B
- **Tags available:** cards, keyrings

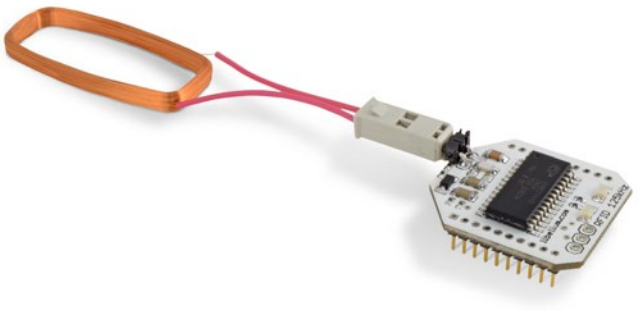


Figure 80: 125KHz RFID module

Applications:

- Located based services (LBS)
- Logistics (assets tracking, supply chain)
- Product management
- Animal farming identification

Related API libraries: **WaspRFID13.cpp** , **WaspRFID13.h**, **WaspRFID125.cpp**, **WaspRFID125.h**

All information on their programming can be found in documents: **RFID 13.56MHz Networking Guide** and **RFID 125KHz Networking Guide**.

All the documentation is located in the **Development section** in the Libelium website.

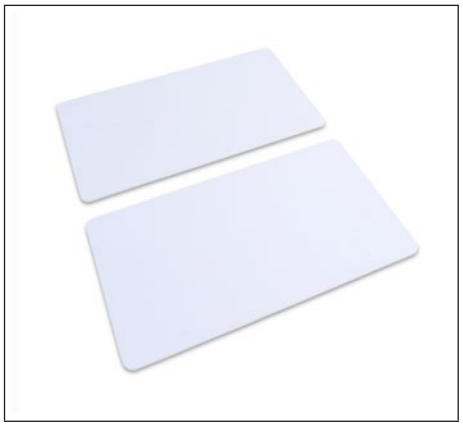
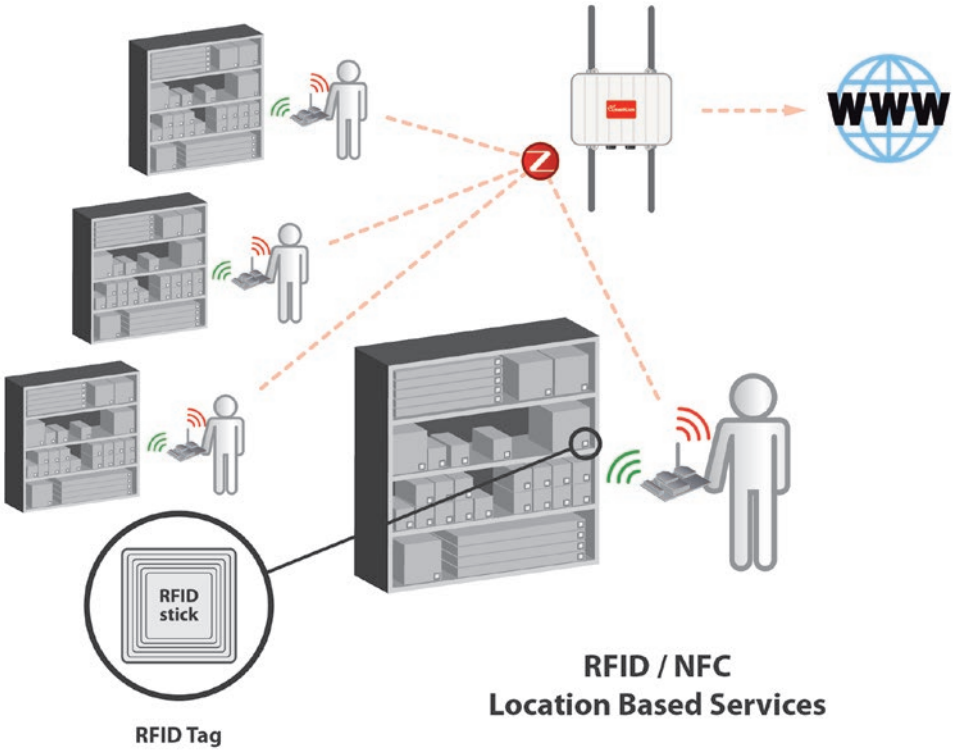
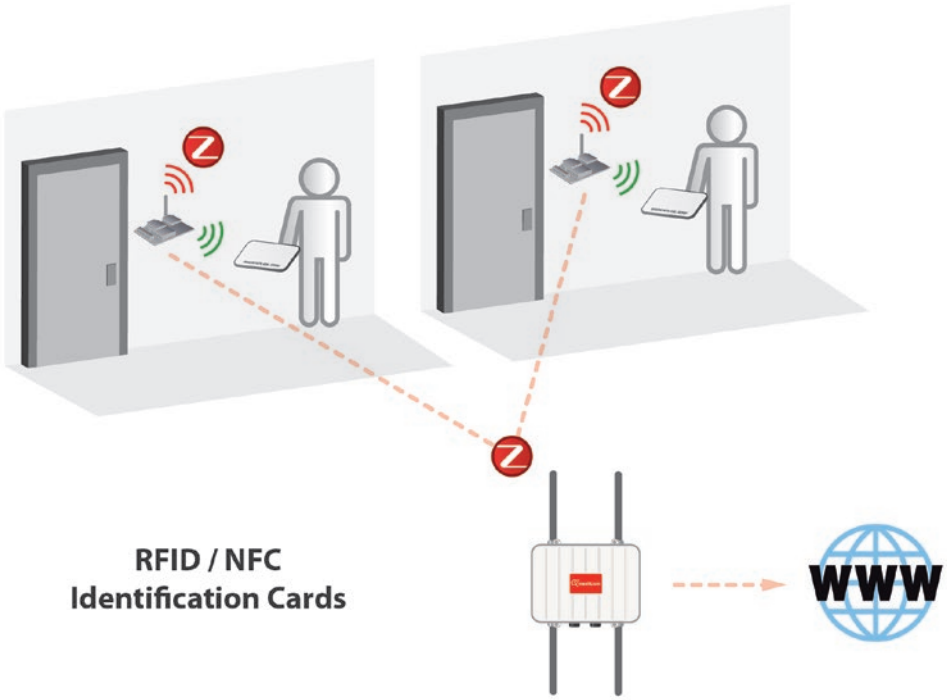


Figure 81: RFID cards

Figure 82: RFID keyrings

Figure 83: RFID sticker


WWW
RFID
stick
RFID Tag
RFID / NFC
Location Based Services


RFID / NFC
Identification Cards
WWW

15. Expansion Radio Board

The Expansion Radio Board allows to connect two radios at the same time in the Waspmote sensor platform. This means a lot of different combinations are now possible using any of the 10 radios available for Waspmote: **802.15.4**, **ZigBee**, **Bluetooth**, **RFID**, **RFID/NFC, WiFi**, **GSM/GPRS, 3G/GPRS**, **868** and **900**.

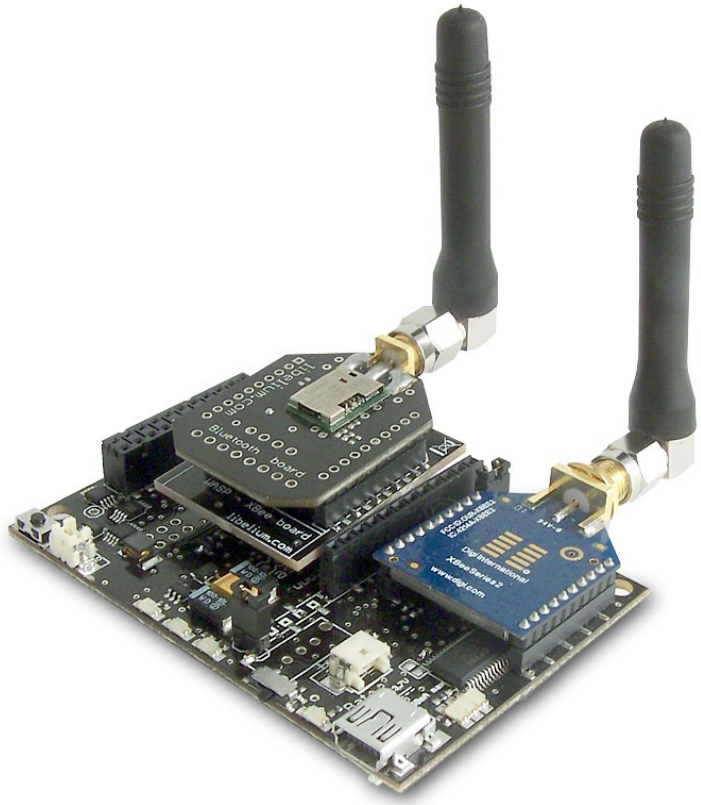


Figure 84: Expansion Radio Board

Some of the possible combinations are:

- ZigBee - Bluetooth
- ZigBee - RFID
- ZigBee - WiFi
- ZigBee - GSM/GPRS
- Bluetooth - RFID
- RFID - 3G/GPRS
- etc.

Remark: the GSM/GPRS module and the 3G/GPRS module do not need the Expansion Board to be connected to Waspmote. It can be plugged directly in the GPRS socket.

Applications:

- Multifrequency Sensor Networks (2.4GHz - 868/900MHz)
- Bluetooth - ZigBee hybrid networks
- NFC (RFID) applications with 3G/GPRS
- ZigBee - WiFi hybrid networks

16. Over the Air Programming (OTA)

16.1. Overview

The concept of Wireless Programming or commonly known as Programming Over the Air (OTA) has been used in the past years overall for the reprogramming of mobile devices such as cell phones. However, with the new concepts of Wireless Sensor Networks and the Internet of Things where the networks consist of hundreds or thousands of nodes OTA is taken to a new direction, and for the first time it is applied using unlicensed frequency bands (2.4GHz, 868MHz, 900MHz) and with low consumption and low data rate transmission using protocols such as 802.15.4 and ZigBee.

Besides, Libelium provides a new OTA method based on FTP transmissions to be used with GPRS, 3G and WiFi modules.

Note that the concept of OTA may have some other names such as:

- Over the air -> OTA
- Over the air Programming -> OTAP
- Firmware over the air -> FOTA
- Programming Over the air-> POTA
- Over the air service provisioning -> OTASP
- Over the air provisioning -> OTAP
- Over the air parameter administration -> OTAPA
- Over the air upgrade -> OTAU
- Over the air update -> OTAUR
- Over the air Download -> OAD
- Over the air flashing -> OTAF
- Over the air parameter administration -> OTAPA
- Multihop Over the air programming (MOTAP)

16.2. Benefits

Libelium OTA Benefits:

OTA with 802.15.4/ZigBee:

- Enables the upgrade or change of firmware versions without physical access.
- Discover nodes in the area just sending a broadcast discovery query.
- Upload new firmware in few minutes.
- No interferences: OTA is performed using a change of channel between the programmer and the desired node so no interferences are generated to the rest of the nodes.

OTA with 3G/GPRS/WiFi:

- Enables the upgrade or change of firmware versions without physical access.
- Upgrades the new firmware by querying a FTP server which helps to keep battery life.
- Upload new firmware in few minutes.

To know more about OTA benefits and process, please read the Over the Air Programming Guide:

http://www.libelium.com/development/waspmote/documentation/over-the-air-programming-guide-otap/

16.3. Concepts

There are two different OTA methodologies:

- OTA with 802.15.4/ZigBee modules
- OTA with 3G/GPRS/WiFi modules via FTP

16.3.1. OTA with 802.15.4/ZigBee modules

The idea is simple. When the programmer (normally the Gateway) sends a new program it is stored in the SD card. A second command "start_new_program" is needed in order to make them start. Then, the nodes copy the program from the SD card to the Flash memory and start the new program.

Steps:

- Locate the node to upgrade
- Check current software version
- Send the new program
- Reboot and start with the new program
- Restore the previous program if the process fails

OTA modes:

- Unicast: Reprogram an specific node
- Multicast: Reprogram several nodes at the same time sending the program just once
- Broadcast: Reprogram the entire network sending the program just once

Topologies:

- Direct access: when the nodes are accessed in just one hop (no forwarding of the packets is needed).
- Multihop: when the nodes are accessed in two or more hops. In this mode some nodes have to forward the packets sent by the Gateway in order to reach the destination

Protocols supported:

- 802.15.4 - 2.4GHz (Worldwide)
- ZigBee - 2.4GHz (Worldwide). ***Important:*** OTA operations only available from the Gateway, not from Meshlium.
- DigiMesh - 2.4GHz (Worldwide)
- RF - 868MHz (Europe)
- RF - 900MHz (US, Canada, Australia)

Storage System:

Once we have sent the program to Waspmote it will store it in the internal memory, a 2GB SD card.

If we have into account that the maximum size for a program is 128KB, this means we can store thousands different firmware versions inside each node.

Encryption and Authentication:

All the data which is sent in the OTA process can be secured by activating the encryption algorithm AES 128b which works in the link layer. As well as this, a second pass key is needed to be known by the OTA programmer (the Gateway) in order to be authenticated and validated by each node before starting with the OTA action requested.

OTA-Shell:

The OTA-Shell application can be used in Windows, Linux and MacOS. It allows to control in a quick and powerful way all the options available in OTA. If you are using Meshlium as the Gateway of the network, the OTA-Shell environment comes already preinstalled and ready to use. This is the recommended way when deploying a real scenario.

16.3.2. OTA with 3G/GPRS/WiFi modules via FTP

The reprogramming process in this type of OTA is initiated by Waspmote and it is supported by an FTP server.

Steps:

• Waspmote queries the FTP server for a new program version

• Check if program name, path and version are correct

• Download the new program

• Reboot and start with the new program

Topologies:

• Protocols which support FTP transmissions are directly connected to the Network Access Point

Protocols supported:

• 3G - Tri-Band (2100/1900/900 MHz), Quad-Band (850/900/1800/1900 MHz)

• GPRS - 850/900/1800/1900 MHz

• WiFi - 2.4GHz (Worldwide).

Storage System:

Once the program is downloaded to Waspmote it is stored it in the 2GB SD card.

Meshlium OTA-FTP plug-in

Meshlium provides an FTP server and Manager System plug-in which permits to configure the server automatically by attaching the program binary file to be used.

16.4. OTA with 802.15.4/ZigBee modules

16.4.1. OTA Step by Step

- **Locate the node or nodes to upgrade**

  Using the 'scan_nodes' function we can search for a specific node or send a global query looking for any node which is ready to be reprogrammed with the OTA process.

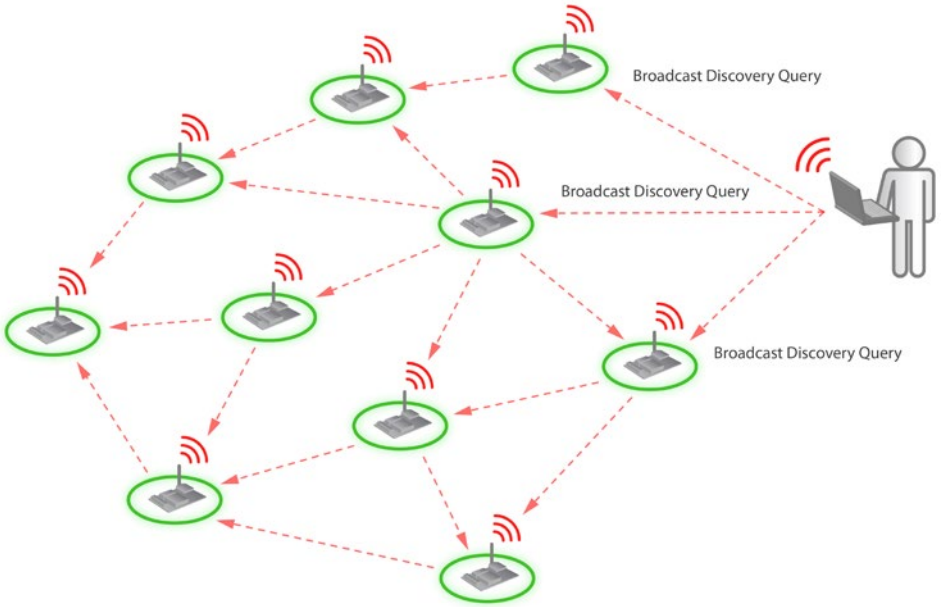


Figure 85: Sending Broadcast discovery queries

The nodes which are ready at this moment will answer with a "Ready to OTA" frame.

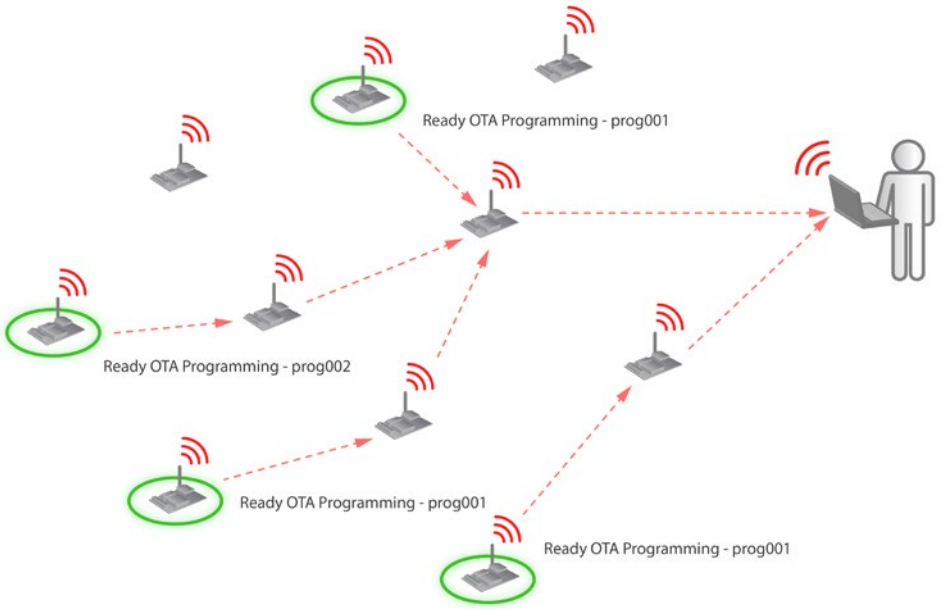


Figure 86: Waspmotes reply to discovery queries

- **Send the new program**

  We can use the 'send' command with the unicast, multicast or broadcast option depending on how many nodes we want to reprogram at the same time.

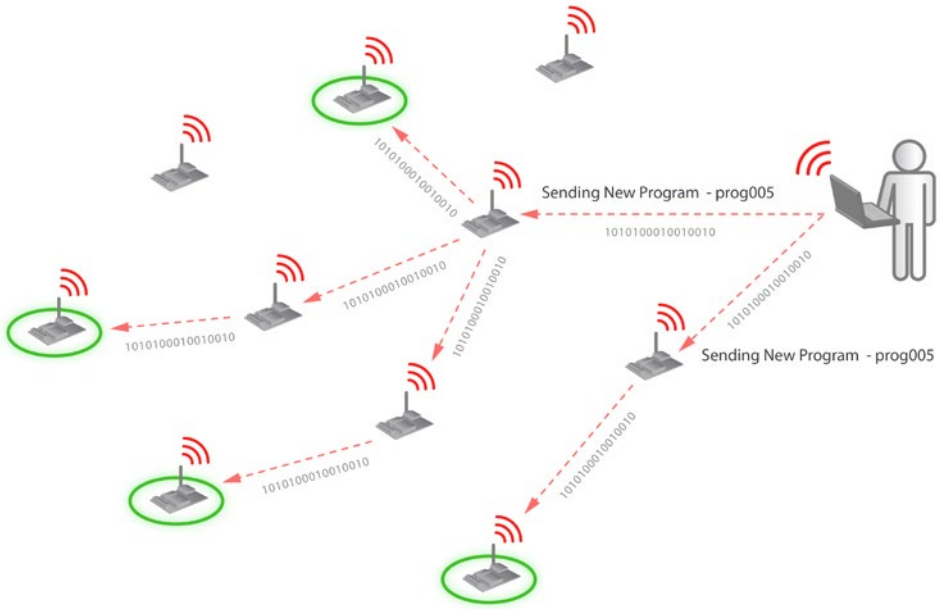


Figure 87: Sending new program via OTA

Each node which receives the program sends a message to the gateway to inform of the success of the process.

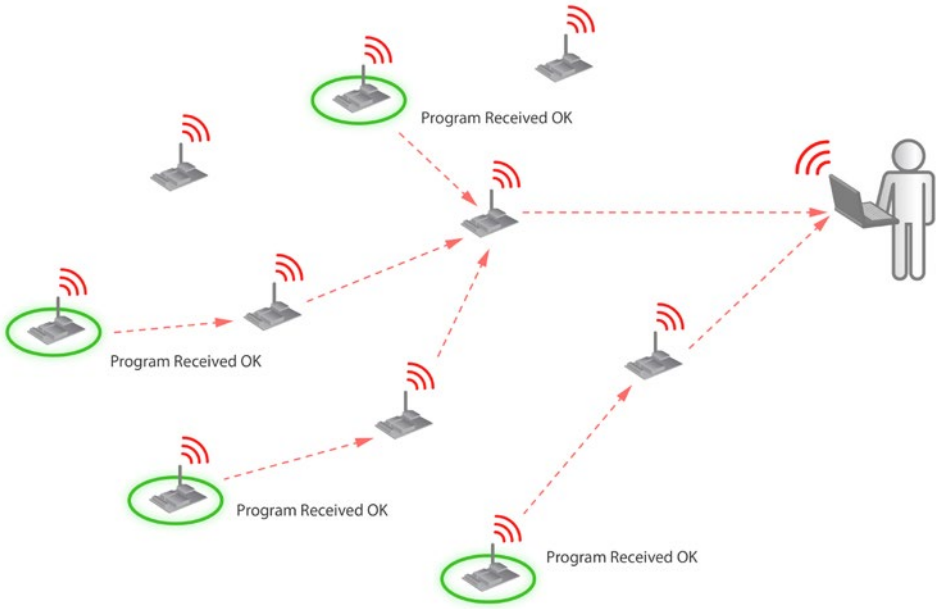


Figure 88: Waspmotes reply OTA process was alright

- **Reboot and start with the new program**

  In order to make the nodes start executing the new program, the gateway needs to send the 'start_new_program' command.

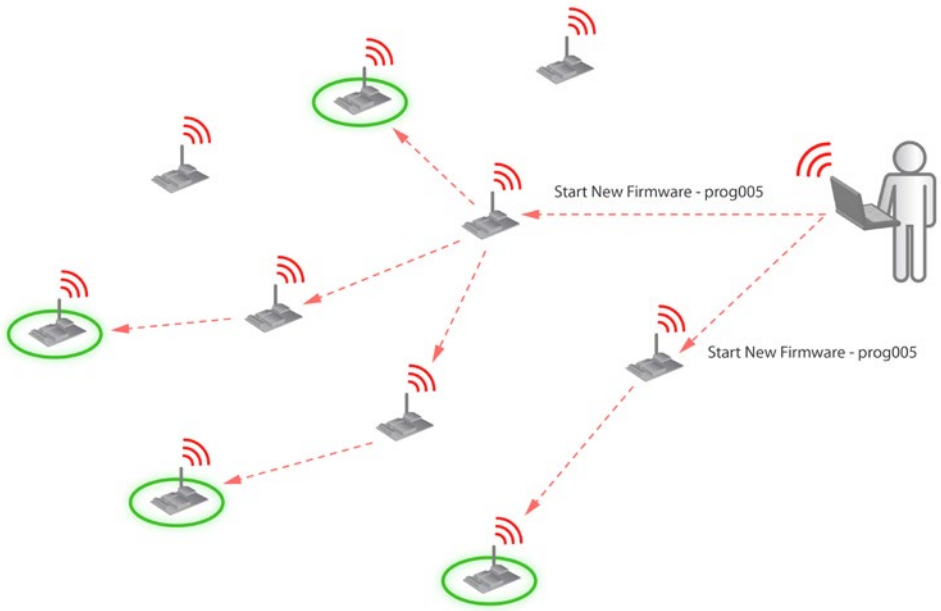


Figure 89: OTA Gateway commands some Waspmotes to start a new program

Each node which receives this packet will copy the program from the SD to the Flash memory and will start running the new binary.

Figure 90: Waspmotes confirm the new program was started

16.4.2. OTA Shell

A powerful command line application called 'OTA Shell' has been developed in order to manage all the features of OTA.

The environment needed to execute OTA Shell comes already preinstalled in Meshlium (the Linux router developed by Libelium which acts as the XBee Gateway of the sensor network. See Meshlium chapter), although it can also be executed in a Linux, Windows and Mac OS system.

Related API libraries: **Included inside WaspXBeeCore.h, WaspXBeeCore.cpp**

All information on their programming can be found in document: **Over the Air Programming (OTA).**

All the documentation is located in the **Development section** in the Libelium website.

In order to know more about OTA including how to download and use the OTA Shell application please go to the Development section:

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

16.5. OTA with 3G/GPRS/WiFi modules via FTP

It is possible to update the Waspmote's program using Over The Air Programming and the following modules: 3G, GPRS or WiFi module.

16.5.1. Procedure

The Waspmote reprogramming is done using an FTP server and an FTP client which is Waspmote itself. The FTP server can be configured by Meshlium. Otherwise, the user will have to setup an FTP server.

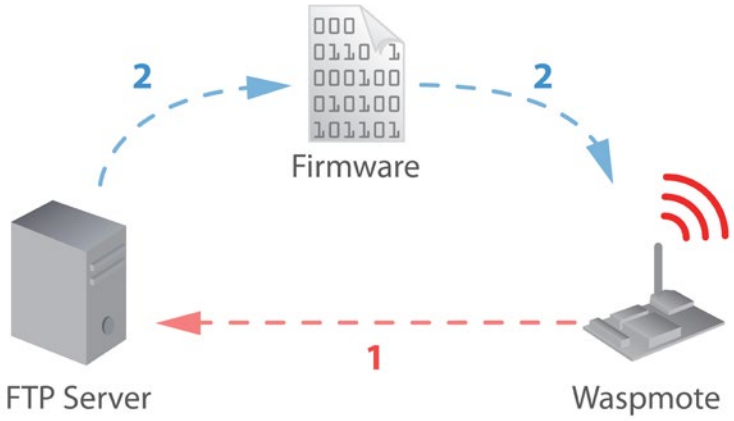


Figure 91: OTA via FTP protocol

There are two basic steps involved in OTA procedure:

- **Step 1:** Waspmote requests a special text file which gives information about the program to update: program name, version, size, etc.
- **Step 2:** If the information given is correct, Waspmote queries the FTP server for a new program binary file and it updates its flash memory in order to run the new program.

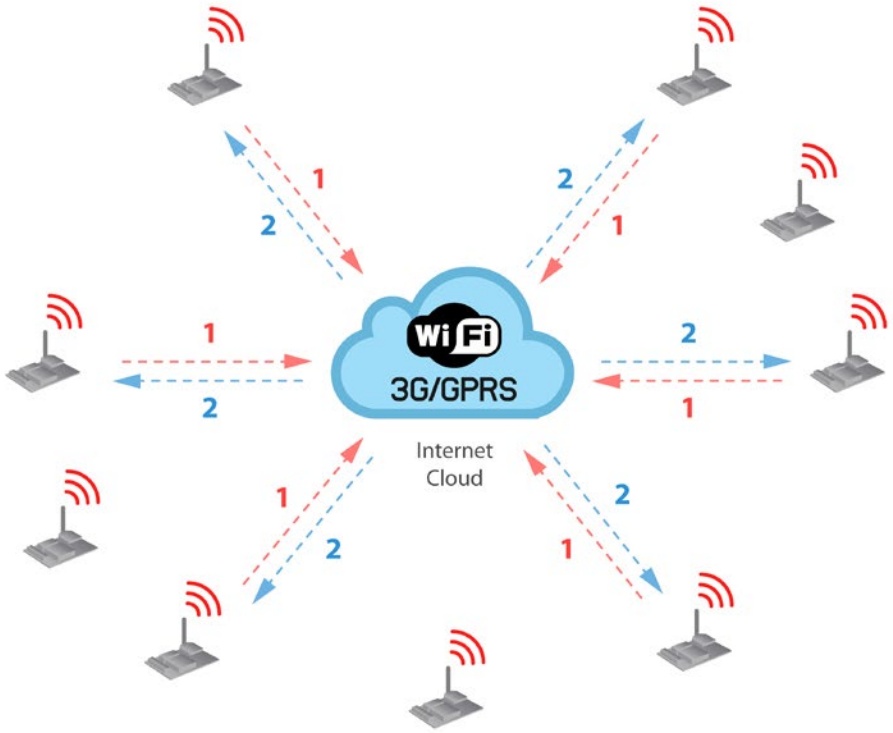


Figure 92: OTA steps via FTP protocol

16.5.2. Setting the FTP server configuration

The FTP server that Waspmote connects to needs a specific configuration so as to OTA work properly. There are two ways to set up the FTP server:

- Extern user's FTP server: The user sets up an FTP server following the specific settings which are described within OTA Guide.
- Meshlium FTP server: There is a specific plugin which allows the user to setup the FTP server automatically indicating the new binary to be downloaded.

17. Encryption Libraries

The new Encryption Libraries are designed to add to the Waspmote sensor platform the capabilities necessary to protect the information gathered by the sensors. To do so three cryptography layers are defined:

1º- In the first one all the nodes of the network share a common **preshared key** which is used to encrypt the information using **AES 128**. This process is carried out by specific hardware integrated in the same 802.15.4/ZigBee radio, allowing the maximum efficiency of the sensor nodes energy consumption. This first security layer ensures no third party devices will be able to even connect to the network (access control).

2º- In the second security layer each node uses a point to point encryption scheme with Meshlium -the Gateway of the network-. This way even the intermediate nodes of the network (the ones which forward the information to the destination) will not be able to see the sensor data transmitted. To perform this technique each node interchanges with the Gateway a new encryption key using **RSA 1024 (Public/Private keys)** what ensures at the same time authentication and integrity. Once the key has been confidentially interchanged the rest of the communication is encrypted by using **AES 256** via software as it ensures complete confidentiality and privacy while maintaining the minimum resources of the node in term of computing cicles and energy consumption.

The "*point to point*" encryption key is cyclically changed using again RSA encryption in a process know as **key renewal**.

3º- The third security technique is carried out in Meshlium -the Gateway- where **HTTPS** and **SSH** connections are used to send the information to the Cloud server located on the Internet.

A fourth optional encryption layer allows each node to encrypt the information using the Public key of the Cloud server. This way the information would keep confidential all the way from the sensor device to the web or data base server on the Internet.

The two main cases of the usage of the Encryption Libraries for Waspmote are:

- Transmission of sensor data

- Key initial sharing and key renewal

17.1. Transmission of sensor data

Information is encrypted in the application layer via software with **AES 256** using the key shared exclusively between the origin and the destination. Then the packet is encrypted again in the link layer via hardware with **AES 128** so that only trusted packets be forwarded, ensuring access control and improving the usage of resources of the network.

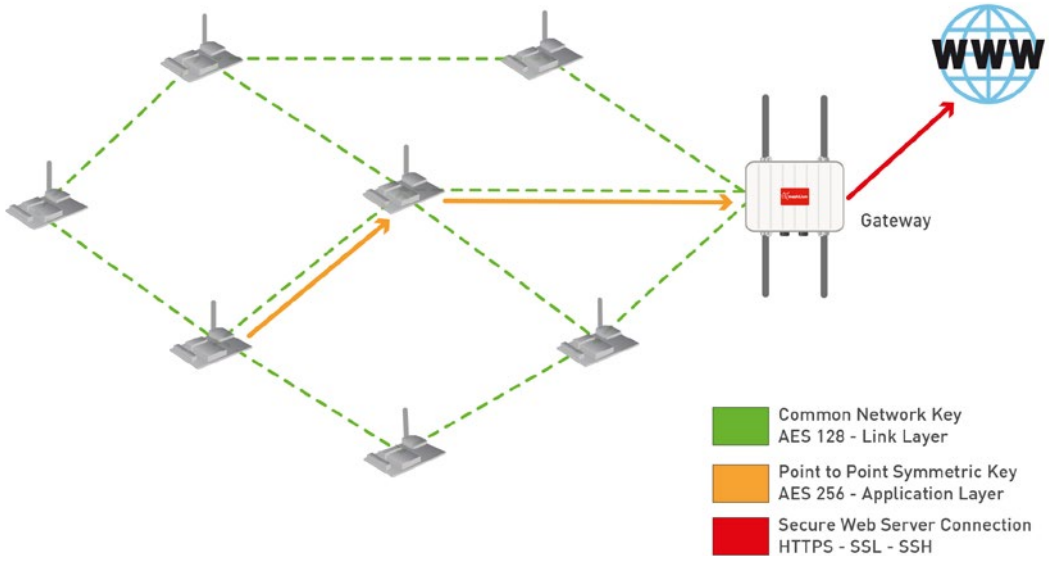


Figure 93: Communication diagram

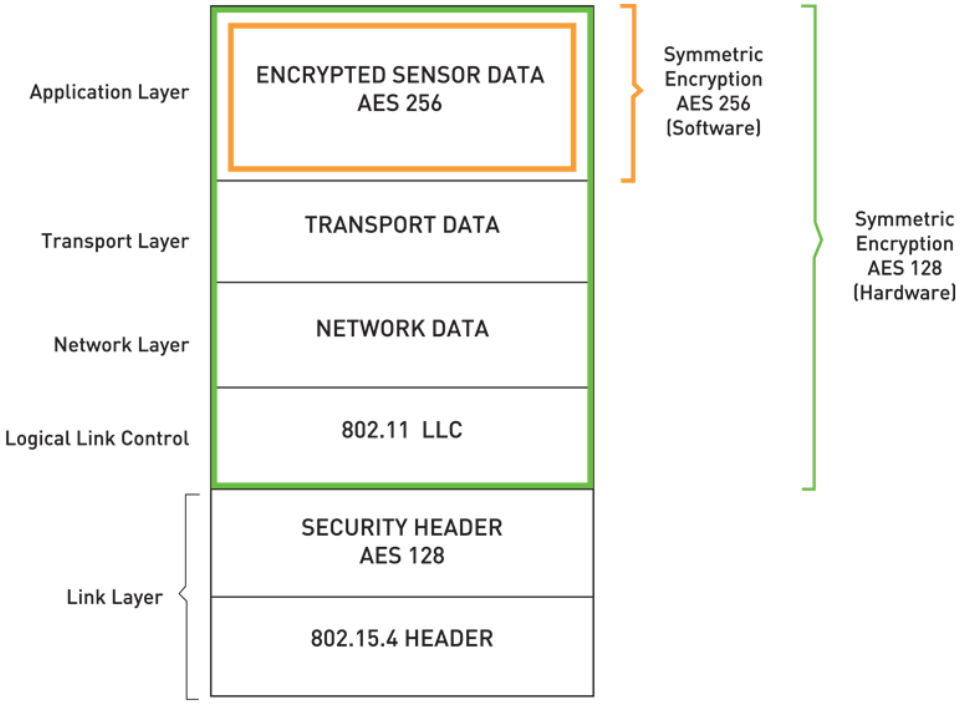


Figure 94: Waspmote frame on OSI stack for communication

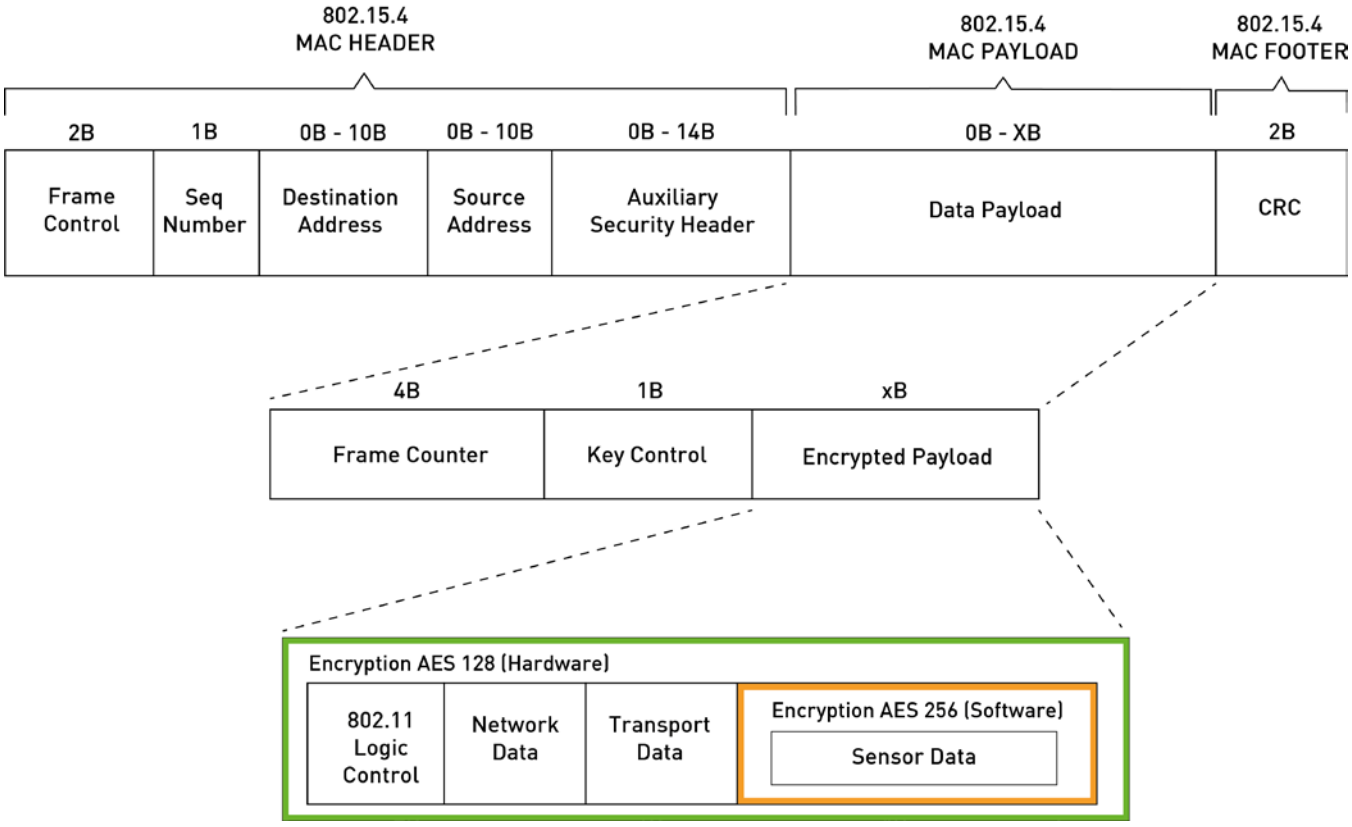


Figure 95: Waspmote frame structure for communication

17.2. Key sharing and key renewal

Prior to start with the software encryption with AES 256 we need to share a key between each node (origin) and the Gateway or the Cloud Server (destination). To do so we encrypt the new key using **RSA 1024** using both **Public/Private** keys. This way, we ensure authentication, confidentiality and message integrity (as we add also a seed along with the key to generate randomness in the packet transmission). Once we get the shared key we will use it to start encrypting the sensor information as seen in the previous diagram as **AES** it ensures the maximum performance and minimum message overload.

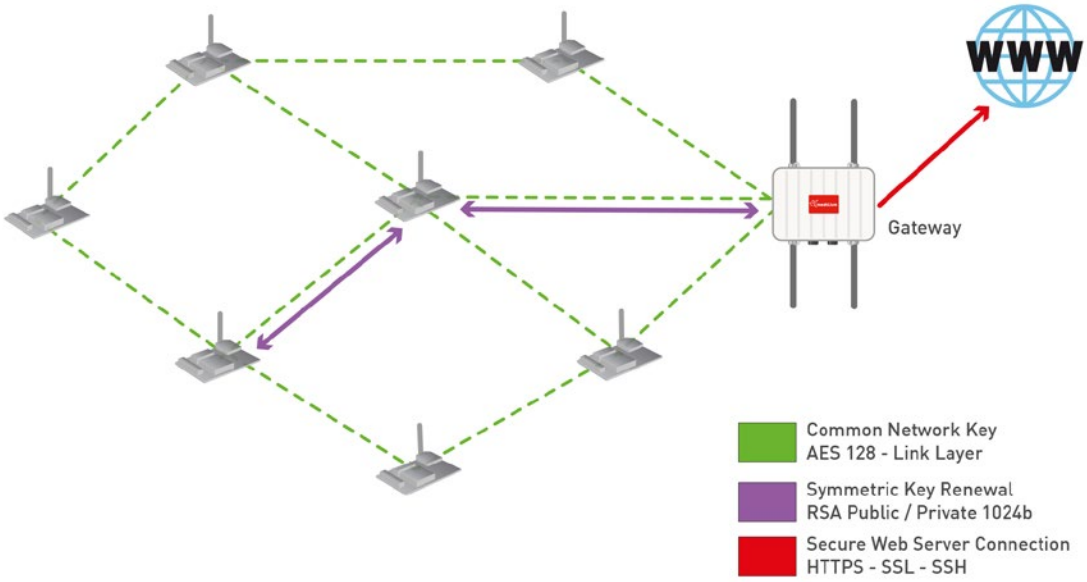


Figure 96: Symetric key renewal diagram

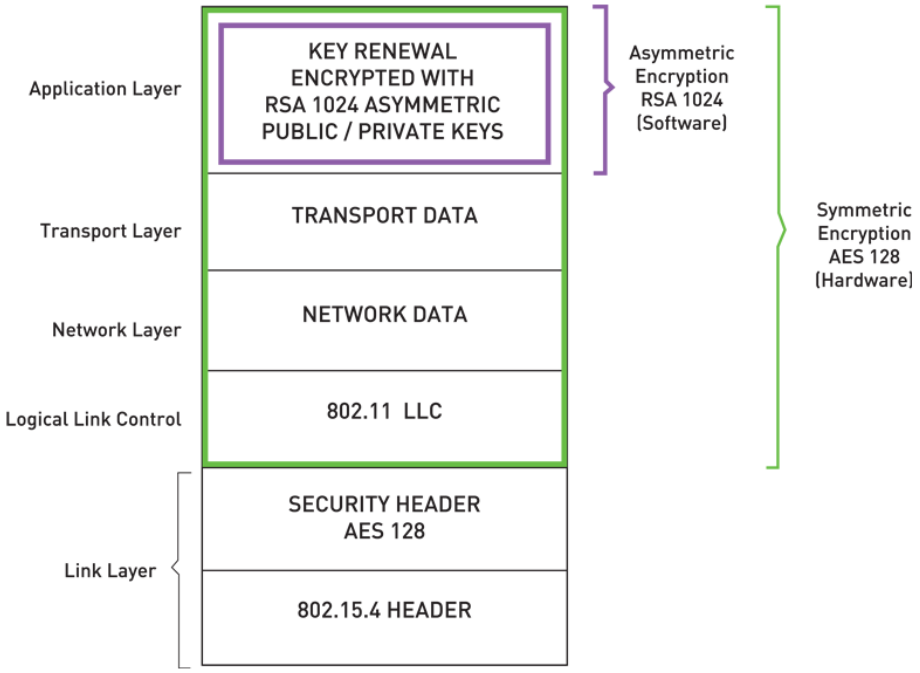


Figure 97: Waspmote frame on OSI for symmetric key renewal

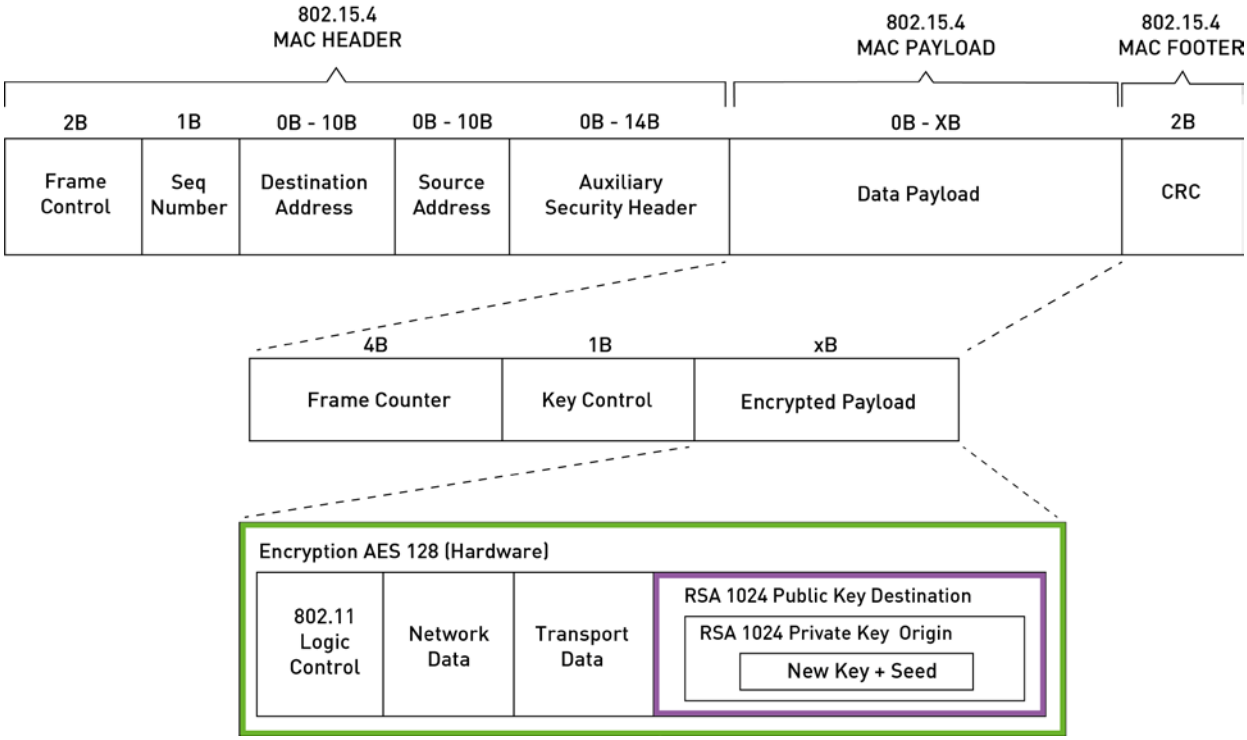


Figure 98: Waspmote frame structure for key renewal

17.3. Common security issues which are solved include

- Access control: by using AES 128 in the link layer we ensure that only nodes with the shared key can access to the routing capabilities of the sensor network. If a strange node sends a message to the network the message will be discharged in the first hop so no extra communication resources will be used. The AES 128 algorithm is implemented in the same radio using specific hardware, for this reason the information will be automatically discarded and not even send to the microcontroller. This provides an extra layer of security as the main control unit of the node will not be interrupted from performing basic tasks or event not awaken from the sleep mode (what ensures optimum energy usage).

- Authentication: the library implements also RSA with asymmetric key scheme. Each node has a pair of Public/Private keys (1024b) which uses to sign the messages in order to ensure the authenticity of origin and destination. By using the SD card each node may store all the Public keys of the nodes of the network in order to even perform symmetric key renewal by encrypting the new key with the public key of the node destination and with the private key of the origin node.

- Data Confidentiality (Privacy): by doubling encryption of the messages we ensure that first that only the nodes which form part of the network can see the general routing packets (AES 128 in the link layer) and after that we stablish an encryption tunnel by direct P2P encryption between origin and destination (using AES 256).

- Data Integrity: the new library uses hash algorithms such as MD5 and SHA to create the checksum of the message and to ensure that the final information received correspond with the original sent.

- Data Freshness (avoiding packet injection): each packet has an exclusive seed which protects the gateway from receiving several identical packets which could be injected from a third party

- Non-repudation: by signing the messages with RSA keys we have also the legal proof that the information sent really was sent by an specific sensor node and not by other. Important in the future when all the sensitive sensor information has to be legally approved.

Each node may store thousands of different Public Keys of the nodes of the network in its SD card. So we can stablish a real P2P encryption among any sensor and the Gateway and even between any sensor and any web or data base server directly.

The new Libraries are specifically designed to be used in the Waspmote hardware sensor platform and are distributed along with the Waspmote IDE which is distributed under an open source license.

Note: *For more information read the "Encryption Programming Guide" in the Waspmote* ***Development section***

18. GPS

Waspmote can integrate a GPS receiver which allows to know the exact outside location of the mote anytime. Thus, the exact position of the mote can be obtained and even the current time and date, to synchronize the Waspmote internal clock (RTC) with the real time.

- **Model:** JN3 (Telit)
- **Sensitivity:**
  - Acquisition: -147 dBm
  - Navigation: -160 dBm
  - Tracking: -163 dBm
- **Hot Start time:** <1s
- **Cold Start Time:** <35s
- **Antenna connector:** UFL
- **External Antenna:** 26dBi
- **Positional accuracy error** < 2.5 m
- **Speed accuracy** < 0.01 m/s
- **EGNOS, WAAS, GAGAN and MSAS capability**

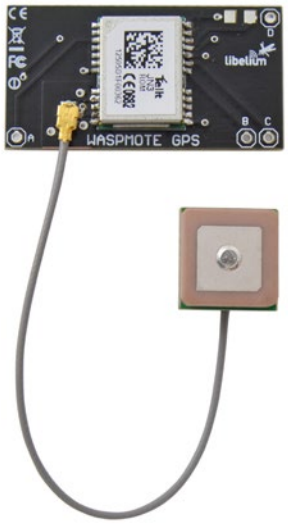


Figure 99: GPS module

The GPS module gives us information about:

- latitude
- longitude
- altitude
- speed
- direction
- date/time
- ephemeris

The functions implemented in the API allow this information to be extracted simply, calling functions such as:

```
{
```

```
        GPS.getAltitude();
        GPS.getSpeed();
        GPS.getLongitude();
        GPS.getLatitude();
    }
```

The GPS receiver uses the UART_1 to communicate with the microcontroller, sharing this UART with the GSM/GPRS or 3G/GPRS module. As the 2 modules share this UART, a multiplexer has been enabled in order to select the module with which we wish to communicate at any time. This is not a problem; since all actions are **sequential**, in practice there is **parallel availability** of both devices.

The GPS starts up by default at 4800bps. This speed can be increased using the library functions that have been designed for controlling and managing the module.

The GPS receiver has 2 operational modes: **NMEA** (National Marine Electronic Association) mode and **binary mode**. NMEA mode uses statements from this standard to obtain **location**, **time** and **date**. The binary mode is based on the sending of structured frames to establish communication between the microcontroller and the GPS receiver, i.e. to read/set ephemeris.

The different types of NMEA statements that the Waspmote's built in GPS receiver supports are:

- NMEA GGA: provides location data and an indicator of data accuracy.
- NMEA GSA: provides the status of the satellites the GPS receiver has been connected to.
- NMEA GSV: provides information about the satellites the GPS receiver has been connected to.
- NMEA RMC: provides information about the date, time, location and speed.
- NMEA VTG: provides information about the speed and course of the GPS receiver.
- NMEA GLL: provides information about the location of the GPS receiver.

The most important NMEA statements are the GGA statements which provide a validity indicator of the measurement carried out, the RMC statement which provides location, speed and date/time and the GSA statement which provides information about the status of the satellites the GPS receiver has been connected to.

(To obtain more information about the NMEA standard and the NMEA statements, visit the website:

http://www.gpsinformation.org/dale/nmea.htm)

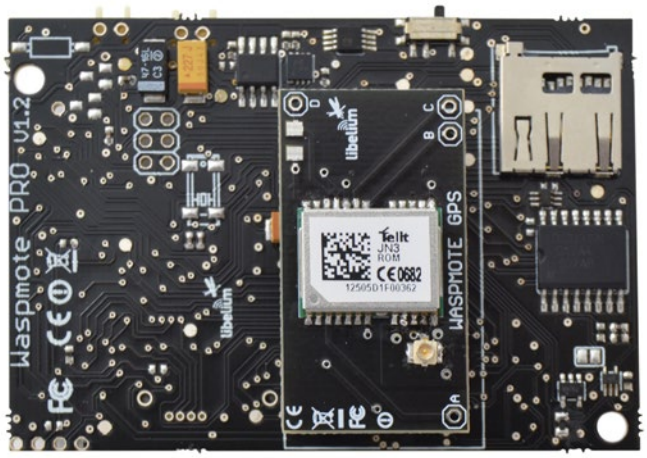


Figure 100: GPS module connected to Waspmote

The GPS receiver needs a certain time to obtain and structure the information that the satellites send. This time can be reduced if there is certain prior information. This information is stored in the almanacs and ephemerides. The information that can be found out is relative to the current position of the satellites (ephemerides) and the trajectory they are going to follow over the next days (almanacs). The almanacs indicate the trajectory that the satellites are going to follow during the next days, having a validity of some 2-3 months. The ephemerides indicate the current position of the satellites and have a validity of some 3-5 hours.

Depending on the information that the GPS receiver has, the start ups can be divided into these types:

- Hot Start: once the time and date are established and the **ephemerides** and valid almanacs are in the memory. Time: **<1s**
- Cold Start: without having established the time, date, almanacs or ephemerids. Time: **<35s**

As can be observed, the start up time reduces greatly, particularly when **ephemerides are stored**. For this reason a series of functions have been created in the libraries to **store ephemerides on the SD card and enable them to be loaded later.**

Related API libraries: **WaspGPS.h, WaspGPS.cpp**

All information about their programming and operation can be found in the document: **GPS Programming Guide.**

All the documentation is located in the **Development section** in the Libelium website.

19. SD Memory Card

Waspmote has external storage support such as SD (Secure Digital) cards. These micro-SD cards are used specifically to reduce board space to a minimum.

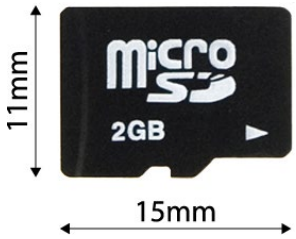


Figure 101: Micro-SD card

Waspmote uses the **FAT16** file system and can support cards up to **2GB**. The information that Waspmote stores in files on the SD can be accessed from different operating systems such as Linux, Windows or Mac-OS. There are many SD card models; any of them has defective blocks, which are ignored when using the Waspmote's SD library. However, when using OTA, those SD blocks cannot be avoided, so that the execution could crash.

Libelium implements a special process to ensure the SD cards we provide will work fine with OTA. The only SD cards that Libelium can assure that work correctly with Waspmote are the SD cards we distribute officially.

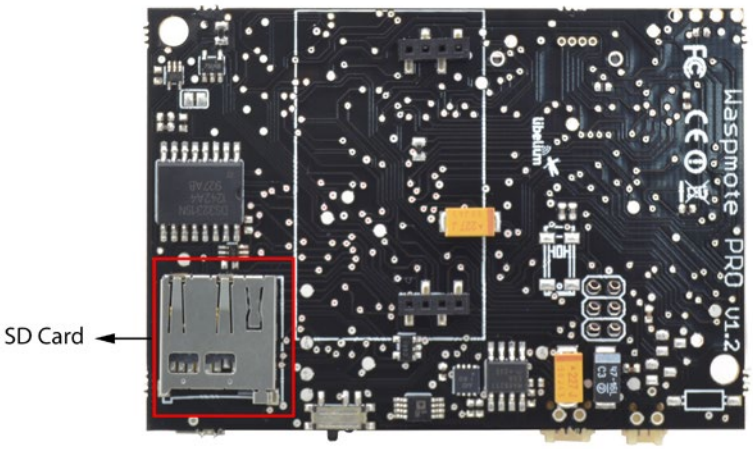


Figure 102: SD Card slot

To communicate with the SD module we use the **SPI** bus. This bus is a communication standard used to transfer information between electronic devices which accept clock regulated bit flow. The SPI includes lines for the clock, incoming data and outgoing data, and a selection pin.

The SD card is powered through a **digital pin** from the microcontroller. It is not therefore necessary to use a switch to cut the power, putting a low pin value is enough to set the SD consumption to **0µA**.

To get an idea of the capacity of information that can be stored in a **2GB** card, simply divide its size by the average for what a sensor frame in Waspmote usually occupies (approx. 100 Bytes):

2GB/100B = 20 million measurements

The limit in files and directories creation per level is 256 files per directory and up to 256 sub-directories in each directory. There is no limit in the number of nested levels.

To show the ease of programming, an extract of code is included below:

```
{
   SD.create(“FILE.TXT”);
   SD.appendln(“FILE.TXT”, “This is a message”);
}
```

Related API libraries: **WaspSD.h, WaspSD.cpp**

All information about their programming and operation can be found in the document: **SD Card Programming Guide.**

All the documentation is located in the **Development section** in the Libelium website.

Note: *Make sure Waspmote is switched off before inserting or removing the SD card. Otherwise, the SD card could be damaged.*

Note: *Waspmote must not be switched off or reseted while there are ongoing read or write operations in the SD card. Otherwise, the SD card could be damaged and data could be lost.*

20. Energy Consumption

20.1. Consumption tables

Waspmote

ON	15mA
Sleep	55µA
Deep Sleep	55µA
Hibernate	0,06µA

XBee

	ON	SLEEP	OFF (Waspmote switches)	SENDING	RECEIVING
XBee 802.15.4	50,36mA	0,1mA	0µA	49,56mA	50,26mA
XBee 802.15.4 PRO	56,68mA	0,12mA	0µA	187,58mA	57,08mA
XBee ZigBee	37,38mA	0,23mA	0µA	37,98mA	37,68mA
XBee ZigBee PRO	45,56mA	0,71mA	0µA	105mA	50,46mA
XBee 868	60,82mA	--	0µA	160mA	73mA
XBee 900	64,93mA	0,93mA	0µA	77mA	66mA

Bluetooth Module

ON	14 mA
OFF	0 mA
Scanning	40 mA
Sending	39 mA
Receiving	20 mA

GPS

ON (tracking)	32 mA
OFF (Waspmote switch)	0µA

GSM/GPRS

Connecting	~100mA
Calling	~100mA
Receiving Calls	~100mA
Transmitting GPRS	~100mA
SLEEP	1mA
OFF	~0µA

3G/GPRS

Connecting	~100mA
Transmitting/Receiving GPRS	~100mA (1,2A – 2A during transmission slot every 4.7ms)
Transmitting/Receiving 3G	~300mA - 500mA
SLEEP	1mA
OFF	~0µA

SD

ON	0.14mA
Reading	0.2mA
Writing	0.2mA
OFF	0µA

Accelerometer

Sleep	0,08mA
Hibernate	0,65mA
OFF	~0µA

21. Power supplies

21.1. Battery

The battery included with Waspmote is a Lithium-ion battery (Li-Ion) with 3.7V nominal voltage. With regard to battery capacity, there are several possibilities: 2300mA and 6600mA Li-Ion rechargeable, and 13000mAh, 26000mAh and 52000mAh **non - rechargeable.**

Waspmote has a control and safety circuit which makes sure the battery charge current is always adequate.

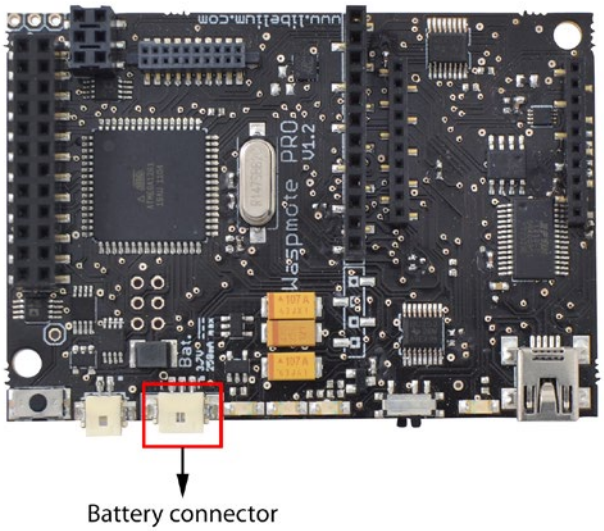


Figure 103: Battery connector

Battery connection

The figure below shows the connector in which the battery is to be connected. The position of the battery connector is unique, therefore it will always be connected correctly (unless the connector is forced).

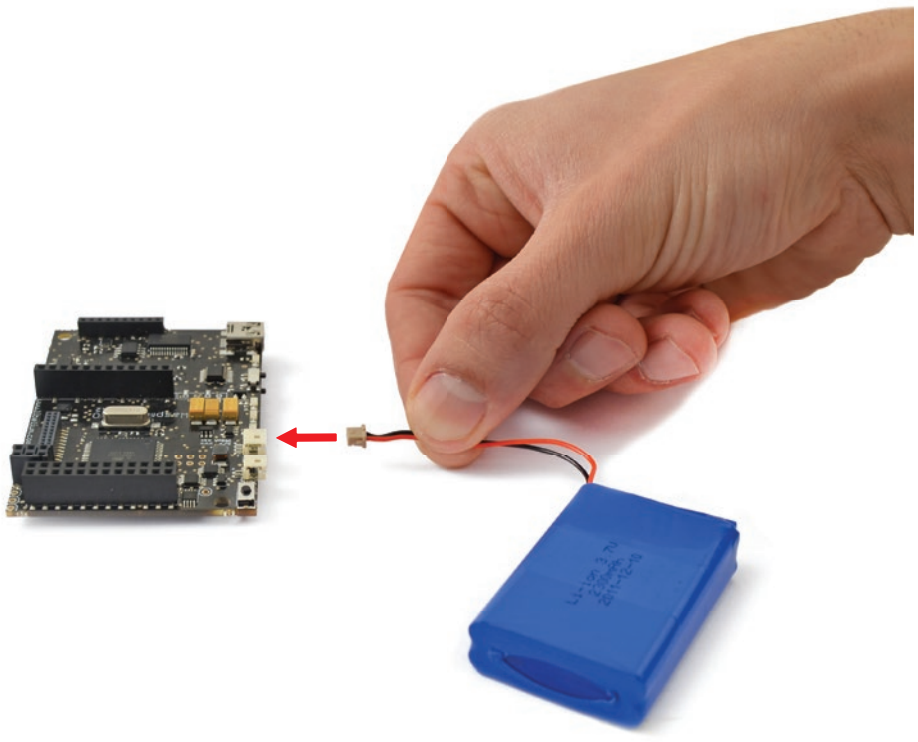


Figure 104: Battery connection

Battery discharging and charging curves

The following two images show battery discharging and charging curves.

Battery discharging

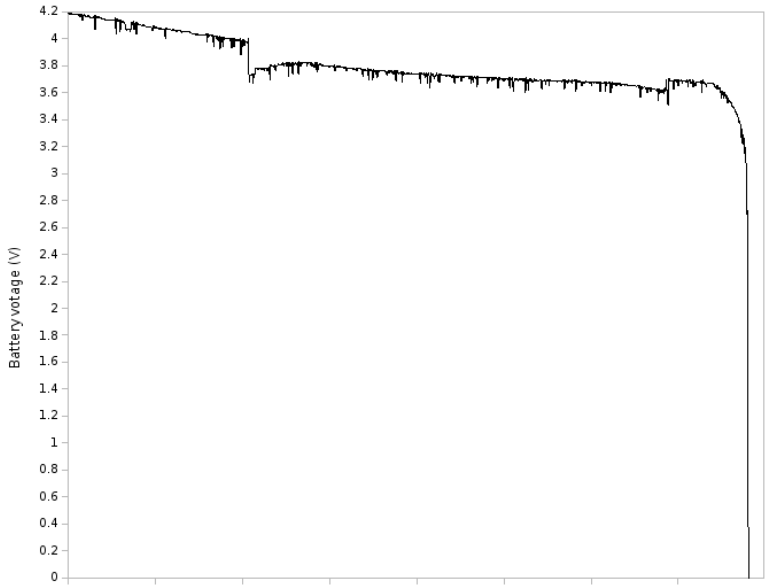


Figure 105: Typical discharging curve for battery

Battery charging using USB

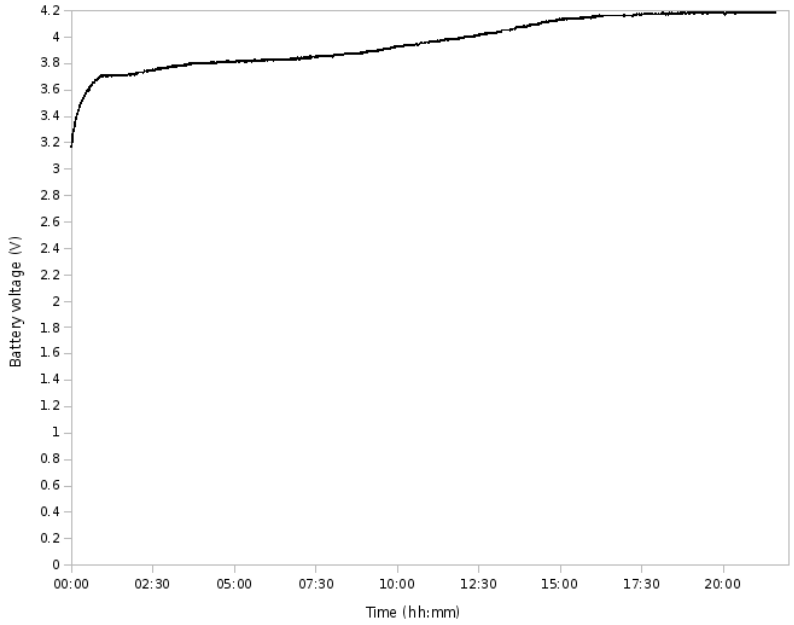


Figure 106: Typical charging curve for battery

Characteristics of the equipment used to generate charging curves:

- Battery used	3.7V - 1150 mAh battery
- Charging	Charging by USB (with Waspmote operating)

Warning: Batteries with voltage over 3.7V could irreparably damage Waspmote.
Incorrect battery connection could irreparably damage Waspmote.

DO NOT TRY TO RECHARGE THE NON-RECHARGEABLE BATTERY. IT MAY EXPLODE AND CAUSE INJURIES AND DESTROY THE EQUIPMENT. DEVICES WITH NON-RECHARGEABLE BATTERIES MUST BE PROGRAMMED THROUGH THE USB CABLE WITHOUT THE BATTERIES CONNECTED. PLEASE DOUBLE CHECK THIS CONDITION BEFORE CONNECTING THE USB. DO NOT CONNECT EITHER UNDER ANY CIRCUMSTANCE THE SOLAR PANEL TO A DEVICE WITH A NON-RECHARGEABLE BATTERY AS IT MAY EXPLODE AND CAUSE INJURIES AND DESTROY THE EQUIPMENT.

21.2. Solar Panel

The solar panel must be connected using the cable supplied.

Both the mini USB connector and the solar panel connector allow only one connection position which must be respected without being forced into the incorrect position. In this way connection polarity is respected.

Solar panels up to **12V** are allowed. The maximum charging current through the solar panel is **280mA**.

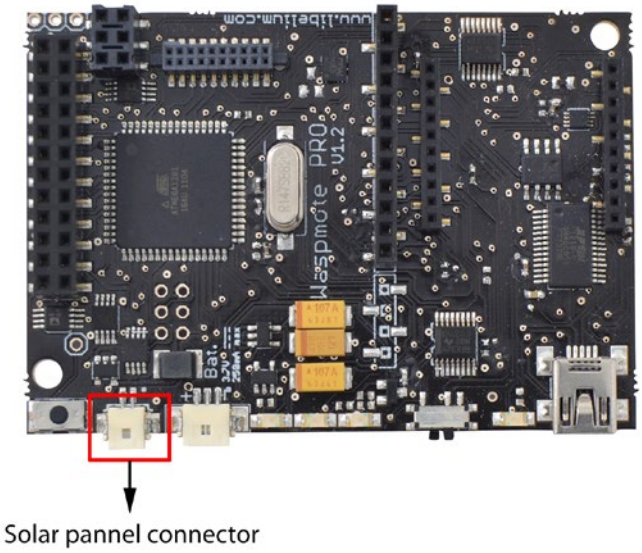


Figure 107: Solar panel connector

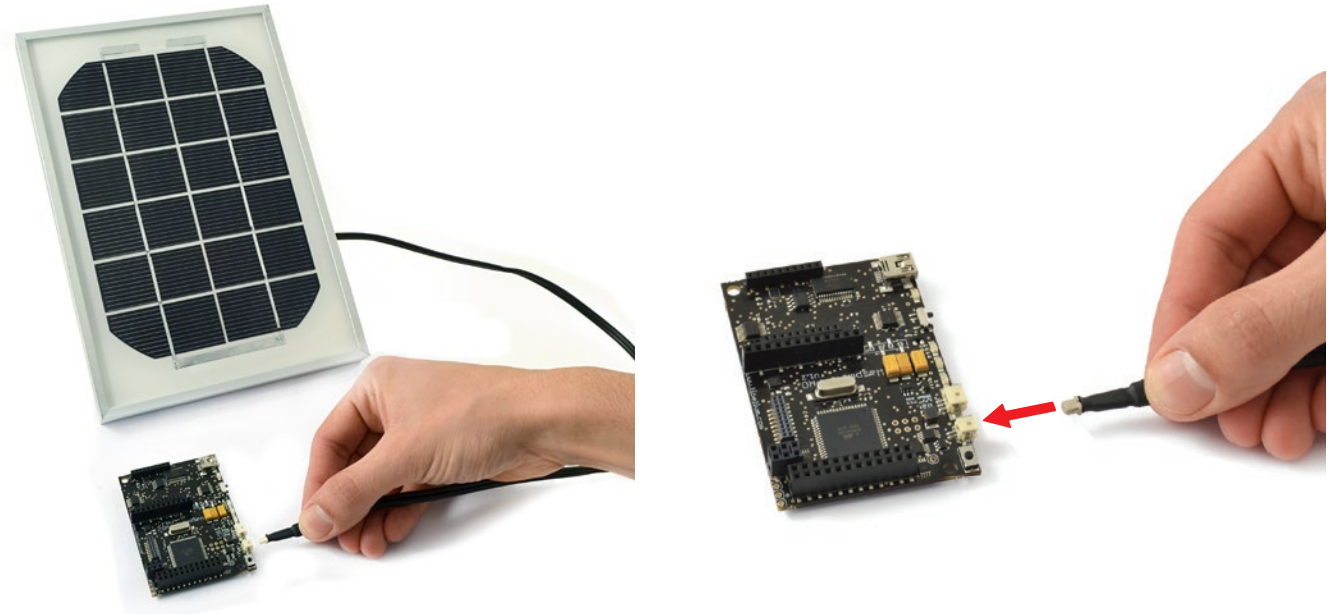


Figure 108: Solar panel connection

The models supplied by Libelium are shown below:

- **Rigid Solar Panel**

  7V - 500mA

Figure 109: Rigid Solar Panel

- **Flexible Solar Panel**

  7.2V - 100mA

Figure 110: Flexible Solar Panel

21.3. USB

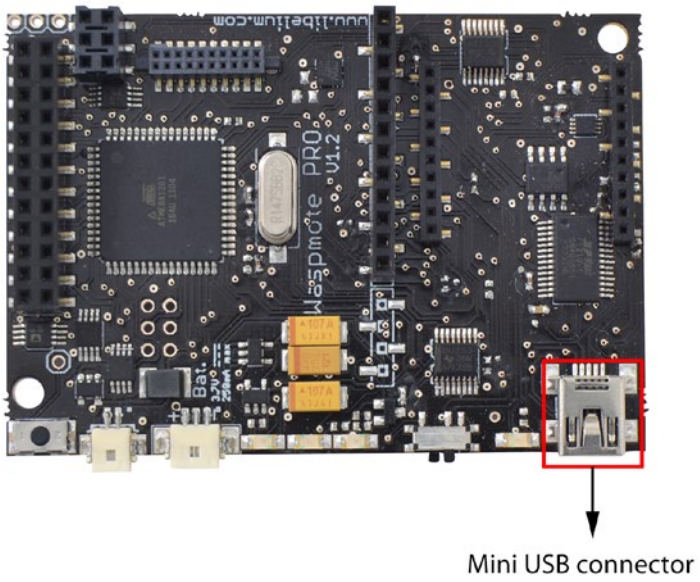


Figure 111: Mini USB connector

Waspmote's USB power sources are:

- USB to PC connection
- USB to 220V connection
- USB to Vehicle connector connection

The charging voltage through the USB has to be **5V**.

The maximum charging current through the USB is **100mA**.

The mini USB connector must be standard mini USB model B.

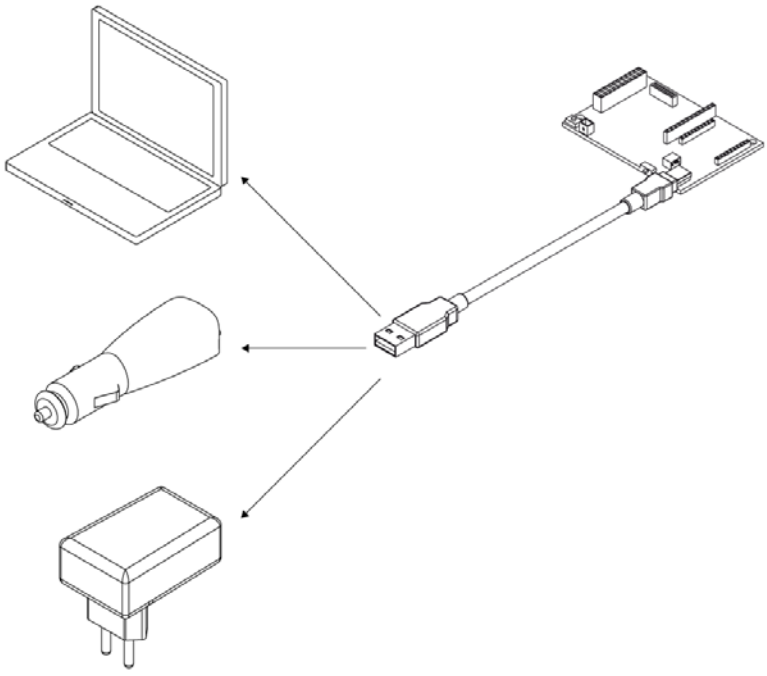


Figure 112: Possible connections for the USB

The models supplied by Libelium are shown below:

Figure 113: 220V AC – USB adapter

Figure 114: 12V DC – USB car lighter adapter

22. Working environment

The first step is to install the **Waspmote IDE** (Integrated Development Environment) used to program Waspmote. This IDE can be found on: **http://www.libelium.com/development/waspmote**

The Waspmote IDE is based on open source Arduino platform compiler, following the same style of libraries and operation. It is important to use the version found on the Waspmote website and no other version of the Arduino IDE. This is because the version available on the Libelium website has been properly tested so we can assure optimum operation.

The Waspmote IDE includes all the API libraries necessary to compile the programs; it is valid for both Waspmote and Waspmote Plug & Sense! platforms.

The file which contains the compiler and the libraries is called "waspmote-pro-ide-vxx" -<OS>" (xx corresponds to the version name and OS to the operating system). This file contains a folder with the Waspmote compiler, which must be extracted to the desired route. The Waspmote libraries are integrated in this folder, being available when the compiler is run.

To be able to run the compilation from a code successfully, a series of applications must be installed on the computer. The applications to install vary according to the O.S. used.

The API is divided into two different folders: **core** and **libraries**. The core folder contains the basic files and the most common utilities for the Waspmote device. The libraries folder contains the API related to the different modules and features Waspmote can manage.

In order to update to future library versions, the API must be modified within the "hardware/cores" and "hardware/libraries" folders found inside the previously unzipped folder.

The next step will be to install the Waspmote IDE. Libelium created a dedicated guide for this task. It is called "Waspmote IDE: User Guide", and can be found on Libelium website software section:

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

This guide will explain in detail how to install the IDE, how to use it, to compile programs or upload sketches. There are details on the libraries structure too. We advise to read this guide carefully.

22.1. First steps

Waspmote comes from factory preconfigured with a program which lets you check the right operation of the device.

Steps:

1. Install the Waspmote IDE on the computer (previous point).
2. Connect the antennas and the rest of the desired components to Waspmote and Waspmote Gateway.
3. Plug Waspmote Gateway to the USB port on the computer.
4. Launch the serial monitor application and set the next parameters:
   - USB port:115200bps
   - 8bits
   - 1 bit stop
   - no parity setting
5. Connect the batteries to the Waspmotes.
6. Switch Waspmotes to the ON position.

When the program starts, it executes sequentially these actions:

- State 1 – Leds ON for 2 seconds
- State 2 – Leds blinking for 3 seconds
- State 3 – Sending messages

State 1 and 2 are only executed once (when program starts) whereas state 3 will loop indefinitely every 3 seconds (if we reset Waspmote, the program starts again).

Every packet contains a message with sensor data formatted as Waspmote Data Frame. For further information, please check the Waspmote Data Frame Guide in:

http://www.libelium.com/development/waspmote/documentation/programming

Example:

```
~\0x00I\0x90\0x00}3\0xa2\0x00@z\0xcb\0x92\0xd8\0xd3\0x02<=>\0x80\0x03#35689722#WASPMOTE#7#A
CC:80;10;987#IN_TEMP:22.50#BAT:93#\0xb4
```

Initially there are some hexadecimal characters, which belong to the XBee API frame, followed by the message. In the above example the message is:

```
<=>\0x80\0x03#35689722#WASPMOTE#7#ACC:80;10;987#IN_TEMP:22.50#BAT:93#
```

In the next chapter is shown how to compile and upload a first program in Waspmote.

22.2. Compilation

To use the Waspmote IDE compiler we must run the executable script called 'Waspmote', which is in the folder where the compiler has been installed.

Waspmote is divided into 4 main parts which can be seen in the following figure.

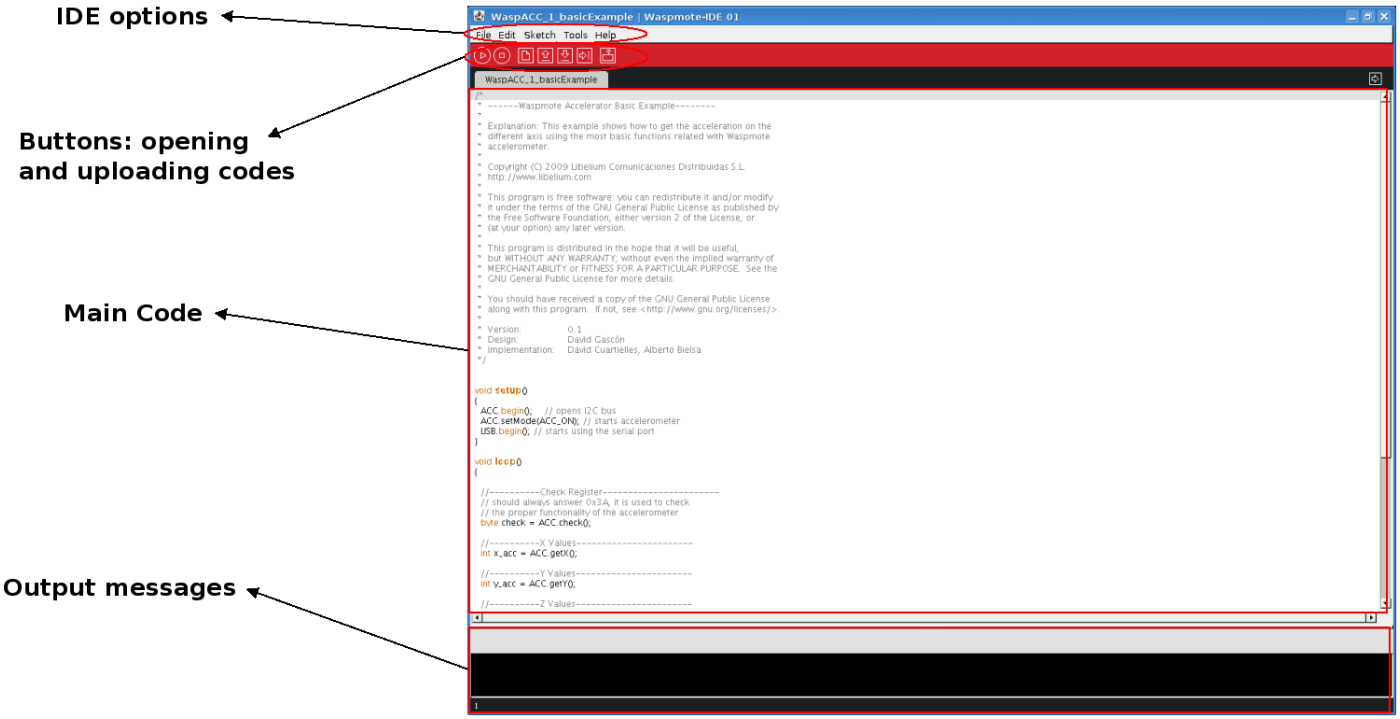


Figure 115: IDE – Waspmote parts

- The first part is the menu which allows configuration of general parameters such as the selected serial port.
- The second part is a button menu which allows verification, opening, saving or loading the selected code on the board.
- The third part contains the main code which will be loaded in Waspmote.
- The fourth part shows us the possible compilation and load errors, as well as the success messages if the process is carried out satisfactorily.

The Waspmote IDE buttons panel allows certain functions to be carried out such as opening a previously saved code, creating a new one or loading the code on the board. The following figure shows the panel and the functions of each button.

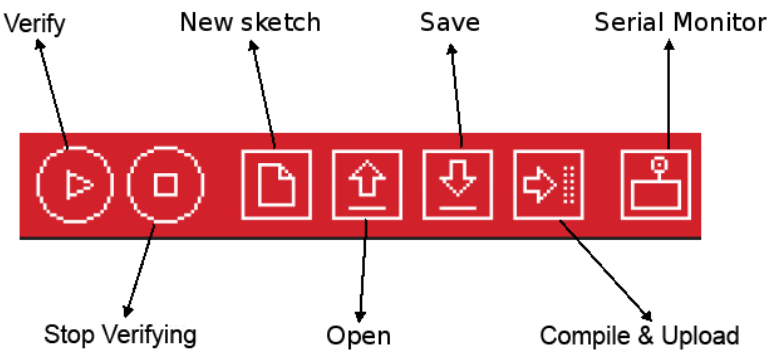


Figure 116: IDE – Waspmote panel of buttons

Once the program has been opened correctly some configuration changes must be made so that the programs load correctly in Waspmote.

In the **'Tools/Board'** tab the Waspmote board must be selected. This refers to the API selected.

In the **'Tools/Serial Port'** tab, the USB to which Waspmote has been connected to the computer must be selected.

Once these 2 parameters have been configured we can load a program onto Waspmote. The process will be explained using a very simple example. A series of examples for learning and familiarizing yourself with the Waspmote environment have been included in the downloaded file that contains the compiler.

The simplest example is the file called 'test.pde'. In this example the text string "Hello World!" appears on the screen. The example shows how to load a program onto Waspmote and how to show information on the screen.

The next step is to configure the folder where the created programs are going to be saved. In the Waspmote-IDE this folder is called **'sketchbook'** and can be configured by accessing the **'File/Preferences'** tab. Clicking on this tab will open a new window where the location of the sketchbook can be indicated. Once the sketchbook folder path is indicated, the downloaded test program must be saved in this folder.

Waspmote IDE must be closed so that the changes and the newly saved program in the sketchbook folder are reflected.

Run Waspmote again and open the downloaded test program by clicking on **'Open'**.

Select the 'test.pde' file in the path where it has been unzipped and open it. As can be seen, it is a very simple code which lights up a LED every 3 seconds and writes "Hello World!" on the screen.

The next step is to load the program onto Waspmote. To do this Waspmote must be connected to the computer through the USB and the button **'upload'** must be clicked. Then, it will start compiling the program. When the program has been compiled correctly, a message will appear on the lower part of the window indicating this event. Conversely, if a fault occurs, red messages will appear indicating the bugs in the code. When compiling is over, the code will be loaded onto Waspmote.

When the program has been loaded correctly, a message appears in the Waspmote window indicating **'Done Uploading'**. Conversely, if some problem occurs during loading, red messages will appear indicating the failures.

Once this program is loaded onto the board, the loaded code will run as was explained in the Architecture and System chapter.

Note: *The Gateway is just a UART-USB bridge. This means that the Gateway cannot be programmed and no code can not be uploaded. Its function is to pass data from the XBee to the USB, and vice-versa.*

22.3. API

An API (Application Programming Interface) has been developed to facilitate applications programming using Waspmote. This API includes all the modules integrated in Waspmote, as well as the handling of other functionalities such as interruptions or the different energy modes.

The API has been developed in C/C++, structured in the following way: **core** folder and **libraries** folder.

22.3.1. Cores folder

The **'hardware/cores'** folder contains the different source cores folders (boards) which might be selected in the IDE window. The core folder contains the general API files which are always compiled, such as:

General configuration

Files: WaspClasses.h, WaspVariables.h, WaspConstants.h, Wconstants.h, pins_waspmote.h, pins_waspmote.c, WaspUtils.h, WaspUtils.cpp, WProgram.h

The basis for correct API operation is defined in these files.

1. WaspClasses.h: all the types to be run on the Waspmote API are defined. If any new type wants to be added, it will be necessary to include it in this file for correct compilation.

2. WaspVariables.h: 4 global variables used as flags for interruptions are defined. These variables are accessible from the files in C, C++ or the main code in the Waspmote compiler.

3. WaspConstants.h: multiple general constants used in the API are defined, as well as all the pins and constants related to the interruptions.

4. Wconstants.h: more constants are defined.

5. pins_waspmote.h, pins_waspmote.c: the microcontroller's pins and the names to which they are associated are defined.

6. WaspUtils.h, WaspUtils.cpp: series of functions for generic use such as light up LEDs, number conversions, strings handling, EEPROM memory, etc.

7. Wprogram.h: is the file which runs when launching the Waspmote compiler. WaspClasses.h and WaspVariables.h are included in it.

Shared

Files: binary.h, HardwareSerial.h, HardwareSerial.cpp, WaspRegisters.h, WaspRegisters.c, wiring_analog.c, wiring.h, wiring.c, wiring_digital.c, wiring_private.h, wiring_pulse.c, wiring_serial.c, wiring_shift.c

Generic functions used are defined in these files, such as the treatment of number types, writing in the UARTs, etc.

SD Storage

Files: Sd2Card.h, Sd2Card.cpp, Sd2Fat.h, Sd2FatStructs.h, Sd2File.cpp, Sd2Info.h, Sd2PinMap.h, Sd2Volume.cpp, WaspSD.h, WaspSD.cpp

The functions needed for storing writing and reading the SD card are defined in these files.

- Sd2Card.h, Sd2Card.cpp, Sd2Fat.h, Sd2FatStructs.h, Sd2File.cpp, Sd2Info.h, Sd2PinMap.h, Sd2Volume.cpp: files that manage the SD card at a low level.
- WaspSD.h, WaspSD.cpp: files that define the necessary functions to read and write information on the SD card.

I2C communication

Files: twi.h, twi.c, Wire.h, Wire.cpp

The functions needed for communication using the I2C bus. These functions are subsequently used by the modules which work with the I2C, such as the accelerometer, the RTC and the sensors.

Accelerometer

Files: WaspACC.h, WaspACC.cpp

The functions needed for reading the accelerometer are defined in these files. The functions needed to activate or deactivate interruptions in this sensor are also defined.

Energy Control

Files: WaspPWR.h, WaspPWR.cpp

The functions needed to activate the different low consumption modes (Sleep, Deep Sleep o Hibernate). The functions needed to obtain the remaining battery value, close the I2C bus and clear interruptions that have been captured are also defined.

RTC

Files: WaspRTC.h, WaspRTC.cpp

The functions needed to obtain the date and time from the internal clock (RTC). The functions needed to activate the alarms and interruptions generated by this module are also defined.

USB

Files: WaspUSB.h, WaspUSB.cpp

The functions needed to use the USB and send/receive information from the computer.

Interruptions

Files: Winterruptions.c

The functions needed for interruptions activation and their subsequent treatment are defined in this file. The interruption subroutines that run when interruptions are captured are defined, as well as the functions for interruption activation and deactivation. Flags corresponding to these functions are marked.

XBee Core

Files: WaspXBeeCore.h, WaspXBee.cpp

The functions that are common to all the XBee modules are defined, such as sending and receiving packets, node discovery or configuration functions that most XBee modules available on Waspmote have. Besides, there are constants used in the libraries related to the XBee modules.

22.3.2. Libraries folder

The **'hardware/libraries'** folder contains the different libraries dedicated to the different modules that can be used with Waspmote. It is necessary to include the library to the code when using it. The subfolders included in libraries are:

GPRS_Pro

Files: WaspGPRS_Pro.h, WaspGPRS_Pro.cpp, WaspGPRS_Proconstants.h

The functions needed for receiving and sending calls, sms or data using the GSM/GPRS network.

3G/GPRS

Files: Wasp3G.h, Wasp3G.cpp

The functions needed for receiving and sending calls, sms or data using the 3G/GPRS network and for manage the Video Camera Sensor Board.

GPS

Files: WaspGPS.h, WaspGPS.cpp

The functions needed to obtain position, date and time from the GPS receiver are defined in these files. The functions needed for managing ephemerids are also defined.

Sensors

Files:

SensorCities: *WaspSensorCities.h, WaspSensorCities.cpp*
SensorAgr_v20: *WaspSensorAgr_v20.h, WaspSensorAgr_v20.cpp*
SensorEvent_v20: *WaspSensorEvent_v20.h, WaspSensorEvent_v20.cpp*
SensorGas_v20: *WaspSensorGas_v20.h, WaspSensorGas_v20.cpp*
SensorParking: *WaspSensorParking.h, WaspSensorParking.cpp*
SensorPrototyping_v20: *WaspSensorPrototyping_v20.h, WaspSensorPrototyping_v20.cpp*
SensorRadiation: *WaspSensorRadiation.h, WaspSensorRadiation.cpp*
SensorSmart_v20: *WaspSensorSmart_v20.h, WaspSensorSmart_v20.cpp*
The functions needed to manage the different sensor boards available on Waspmote.

XBee Libraries

The functions needed to set up, control and use a 802.15.4/ZigBee network.

XBee802: WaspXBee802.h, WaspXBee802.cpp: the specific functions of the XBee 802.15.4 and the shared general library functions are inherited.

XBeeZB: WaspXBeeZB.h, WaspXBeeZB.cpp: the specific functions of the XBee ZigBee modules are defined and the shared general library functions are inherited.

XBeeDM: WaspXBeeDM.h, WaspXBeeDM.cpp: the specific functions of the XBee DigiMesh and 900MHz are defined, and the shared general library functions are inherited.

XBee868: WaspXBee868.h, WaspXBee868.cpp: the specific functions of the XBee 868MHz modules are defined and the shared general library functions are inherited.

XBee900: WaspXBee900.h, WaspXBee900.cpp: the specific functions of the XBee 900MHz modules are defined and the shared general library functions are inherited.

Frame

Files: WaspFrame.h, WaspFrame.cpp

The functions needed to create new data frames by adding different sensor values.

Bluetooth Pro

Files: WaspBT_Pro.h, WaspBT_Pro.cpp

The functions needed to manage the Bluetooth module for scanning devices.

WiFi

Files: WaspWIFI.h, WaspWIFI.cpp

The functions needed to manage the WiFi module.

RFID

Files: WaspRFID13.h, WaspRFID13.cpp

The functions needed to manage the RFID module.

22.4. Updating the libraries

To update the libraries, some files in the folder where the **Waspmote IDE** compiler was installed must be modified. The libraries are compatible with the different environments explained previously: Linux, Windows and Mac-OS.

New versions of the libraries can be downloaded from the page:

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

These new versions are downloaded in a file similar to "waspmote-pro-api-vxxx.zip" (xxx being the current version). This file contains 2 folders: "waspmote-api" and "libraries". The content of these 2 folders must be overwritten on the IDE folders of the same name:

/hardware/cores/waspmote-api and **/hardware/libraries**

Once these folders are replaced, the API is updated to the new version.

It is not possible to have 2 different APIs in the IDE at the same time. The solution is simple: to have several IDEs installed in the PC, one IDE for each API we want to handle. However, it is not recommended to work with old API versions, new versions are more stable and offer more features.

23. Interacting with Waspmote

23.1. Receiving 802.15.4/ZigBee frames with Waspmote Gateway

23.1.1. Waspmote Gateway

This device allows to collect data which flows through the sensor network into a **PC** or device with a standard USB port. Waspmote Gateway will act as a ***"data bridge or access point"*** between the sensor network and the receiving equipment. This receiving equipment will be responsible for storing and using the data received depending on the specific needs of the application.

Figure 117: Waspmote Gateway

The receiving equipment can be a PC with Linux, Windows or Mac-OS, or any device compatible with standard USB connectivity. The gateway offers a "plug" **USB A** connector, so the receiving device has to have a "receptacle" USB A connector.

Once the gateway is correctly installed, a new communication serial port connecting directly to the XBee module's UART appears in the receiving equipment, which allows the XBee to communicate directly with the device, being able to both receive data packets from the sensor network as well as modify and/or consult the XBee's configuration parameters.

Another important function worth pointing out is the possibility of **updating or changing the XBee module's firmware**.

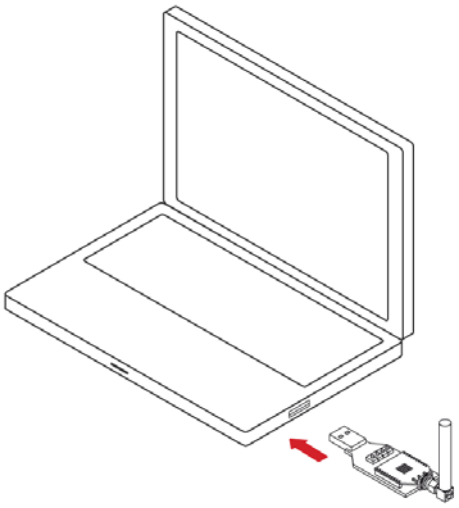


Figure 118: Waspmote Gateway connected in a PC

LEDs

Four indicator LEDs are included in the Gateway:

- USB power LED: indicates that the board is powered through the USB port
- X LED: indicates that the board is receiving data from the USB port.
- TX LED: Indicates that the board is sending data to the USB port
- I/O 5 configurable LED: associate

The configurable LED connected to the XBee's I/O 5 pin can be configured either as the XBee's digital output or as the XBee's indicator of association to the sensor network.

Buttons

- Reset: allows the XBee module to be reset.
- I/O - 0: button connected to the XBee's I/O pin 0.
- I/O -1: button connected to the XBee's I/O pin 1.
- RTS - I/O – 6: button connected to the XBee's I/O pin 6.

All the buttons connect each one of its corresponding data lines with GND with when pressed. None of these have pull-up resistance so it may be necessary to activate any of the XBee's internal pull-up resistances depending on the required use.

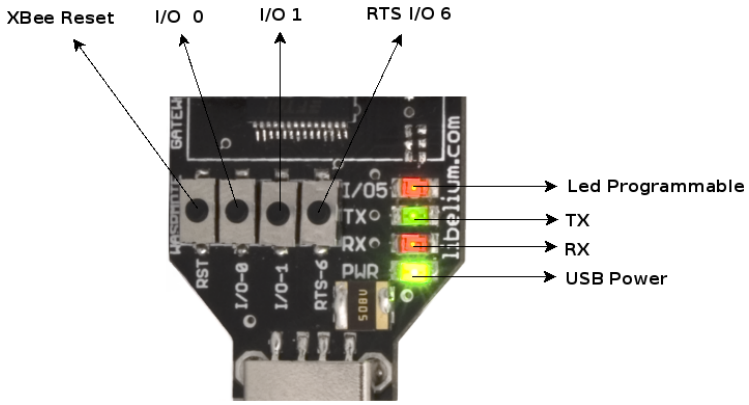


Figure 119: LEDs in Waspmote Gateway

23.1.2. Linux receiver

When using Linux it is possible to use various applications to capture the input from the serial port. Libelium recommends to use the 'Cutecom' application.

Once the application is launched the speed and the USB where Waspmote has been connected must be configured.

The speed that must be selected is 115200 which is the standard speed set up for Waspmote.

The USB where Waspmote has been connected must be added the first time this application is run, adding USB0, USB1, etc (up to the USB number of each computer) according to where Waspmote has been connected. For this, the 'Device' window must be modified so that if Waspmote is connected to USB0, this window contains '/dev/ttyUSB0'.

Once these parameters are configured, capture is started by pressing the 'Open Device' button.

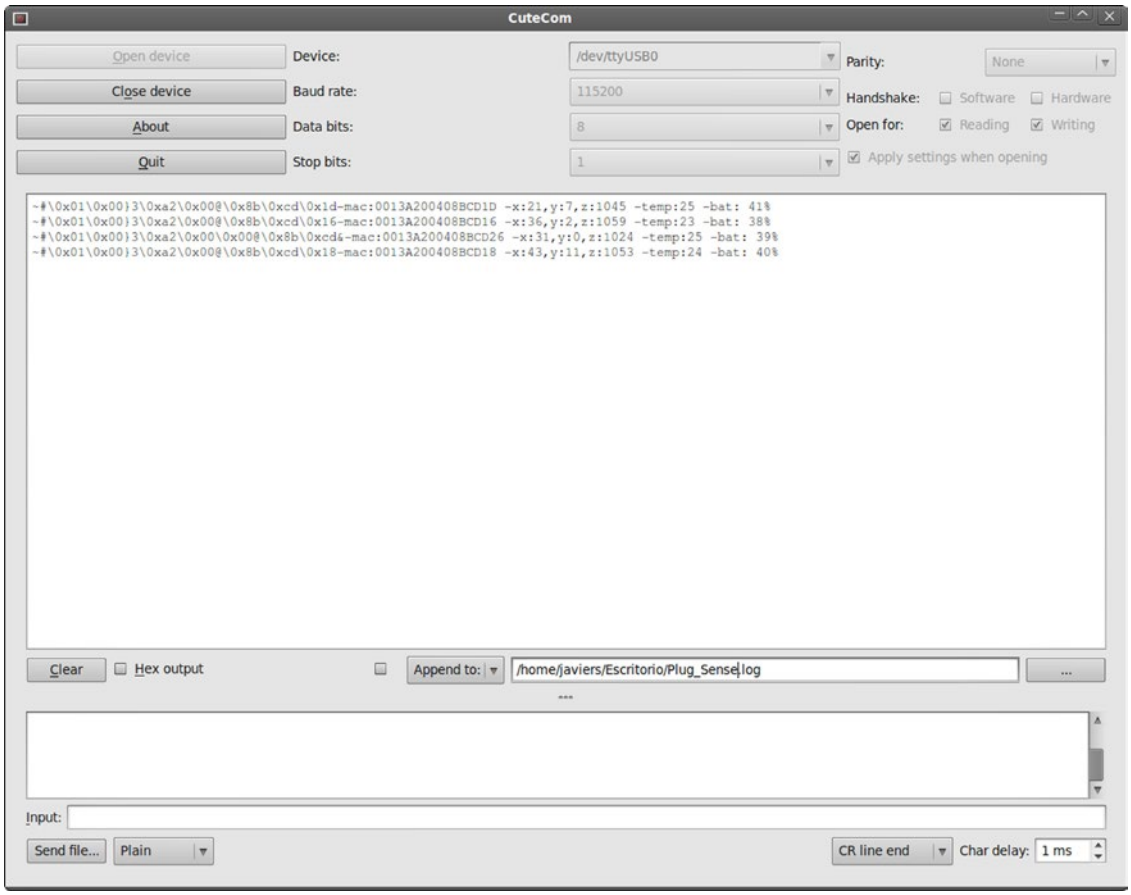


Figure 120: Cutecom application capturing Waspmote's output

Linux Sniffer

As well as using the terminal to see the sensor information, an application which allows this captured data to be dumped to a file or passed to another program to be used or checked has been developed.

File:
"sniffer.c"

Compilation on Meshlium:
`gcc sniffer.c -o sniffer`

Examples of use:

- Seeing received data: `./sniffer USB0`
- Dumping of received data to a file: `./sniffer USB0 >> data.txt`
- Passing received values to another program: `./sniffer USB0 | program`

Note: *the speed used for the example is 19200 baud. The final speed will depend on the speed the XBee module has been configured with (default value 115200).*

Code:

```
#include   <stdio.h>
```

```
          #include   <string.h>
          #include   <unistd.h>
          #include   <fcntl.h>
          #include   <errno.h>
          #include   <stdlib.h>
          #include   <termios.h> /* Terminal control library (POSIX) */

          #define MAX 100

          main(int argc, char *argv[])
          {
            int sd=3;
            char *serialPort="";

char *serialPort0 = "/dev/ttyS0";
char *serialPort1 = "/dev/ttyS1";
char *USBserialPort0 = "/dev/ttyUSB0";
char *USBserialPort1 = "/dev/ttyUSBS1";
char valor[MAX] = "";
char c;
char *val;
struct termios opciones;
int num;
char *s0 = "S0";
char *s1 = "S1";
char *u0 = "USB0";
char *u1 = "USB1";

 if(argc!=2)
  {
    fprintf(stderr,"Usage: %s [port]\nValid ports: (S0, S1, USB0, USB1)\n",argv[0], serial-
Port);
    exit(0);
  }

if (!strcmp(argv[1], s0))
{
  fprintf(stderr,"ttyS0 chosen\n...");
  serialPort = serialPort0;
}
if (!strcmp(argv[1], s1))
{
  fprintf(stderr,"ttyS1 chosen\n...");
  serialPort = serialPort1;
}
if (!strcmp(argv[1], u0))
{
  fprintf(stderr,"ttyUSB0 chosen\n...");
  serialPort = USBserialPort0;
}
if (!strcmp(argv[1], u1))
{
  fprintf(stderr,"ttyUSB1 chosen\n...");
  serialPort=USBserialPort1;
}
if (!strcmp(serialPort, ""))
{
  fprintf(stderr, "Choose a valid port (S0, S1, USB0, USB1)\n", serialPort);
  exit(0);
}

if ((sd = open(serialPort, O_RDWR | O_NOCTTY | O_NDELAY)) == -1)
{
  fprintf(stderr,"Unable to open the serial port %s - \n", serialPort);
  exit(-1);
}
```

```
else
{
  if (!sd)
  {
    sd = open(serialPort, O_RDWR | O_NOCTTY | O_NDELAY);
  }
  //fprintf(stderr,"Serial Port open at: %i\n", sd);
  fcntl(sd, F_SETFL, 0);
}
  tcgetattr(sd, &opciones);
  cfsetispeed(&opciones, B19200);
  cfsetospeed(&opciones, B19200);
  opciones.c_cflag |= (CLOCAL | CREAD);
  /*No parity*/

  opciones.c_cflag &= ~PARENB;
  opciones.c_cflag &= ~CSTOPB;
  opciones.c_cflag &= ~CSIZE;
  opciones.c_cflag |= CS8;
  /*raw input:
  * making the applycation ready to receive*/
  opciones.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
  /*Ignore parity errors*/
  opciones.c_iflag |= ~(INPCK | ISTRIP | PARMRK);
  opciones.c_iflag |= IGNPAR;
  opciones.c_iflag &= ~(IXON | IXOFF | IXANY | IGNCR | IGNBRK);
  opciones.c_iflag |= BRKINT;
  /*raw output
  * making the applycation ready to transmit*/
  opciones.c_oflag &= ~OPOST;
  /*aply*/
  tcsetattr(sd, TCSANOW, &opciones);
  int j = 0;
  while(1)
  {
    read(sd, &c, 1);
    valor[j] = c;
    j++;

    // We start filling the string until the end of line char arrives
    // or we reach the end of the string. Then we write it on the screen.

    if ((c=='\n') || (j==(MAX-1)))
    {
      int x;
      for (x=0; x<j; x++)
      {
        write(2, &valor[x], 1);
        valor[x] = '\0';
      }
      j = 0;
    }
  }
  close(sd);
}
```

The code can be downloaded from: **http://www.libelium.com/development/waspmote**

23.1.3. Windows receiver

If Windows is used, the application 'Hyperterminal' can be used to capture the output of the serial port.

This application can be found installed by default in 'Start/Programs/Accessories/Communication', but if it is not available it can be downloaded from: **http://hyperterminal-private-edition-htpe.en.softonic.com/**

Once this application is launched the connection must be configured. The first step is to give it a name:

Figure 121: Step 1 of establishing connection

The next step is to specify the port on which Waspmote has been connected, in this case the system recognizes it as 'COM9', (this will vary on each computer):

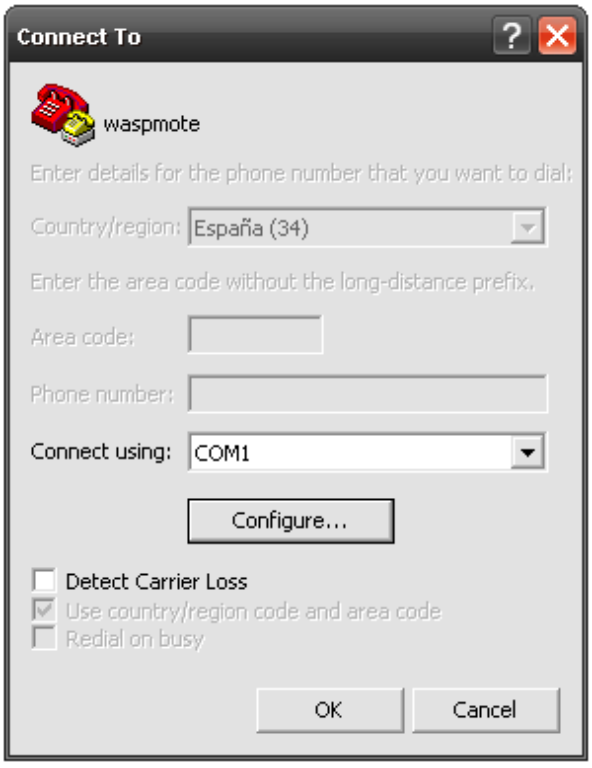


Figure 122: Step 2 of establishing connection

The next step is to specify the speed and configuration parameters:

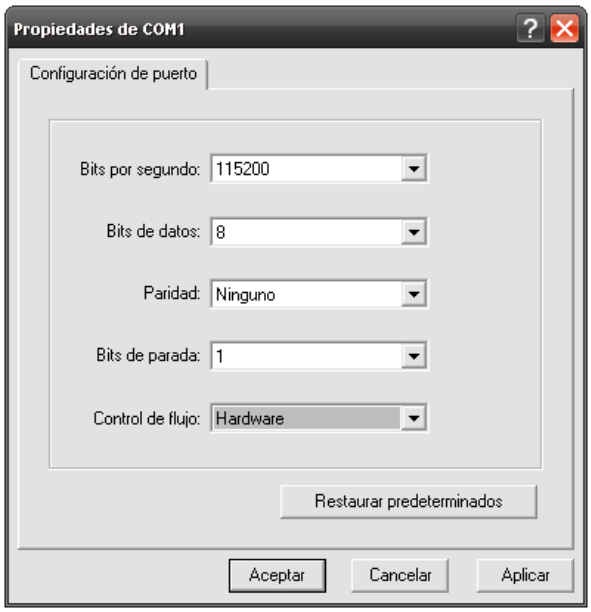


Figure 123: Step 3 of establishing connection

Once these steps have been performed connection with Waspmote has been established, and listening to the serial port begins.

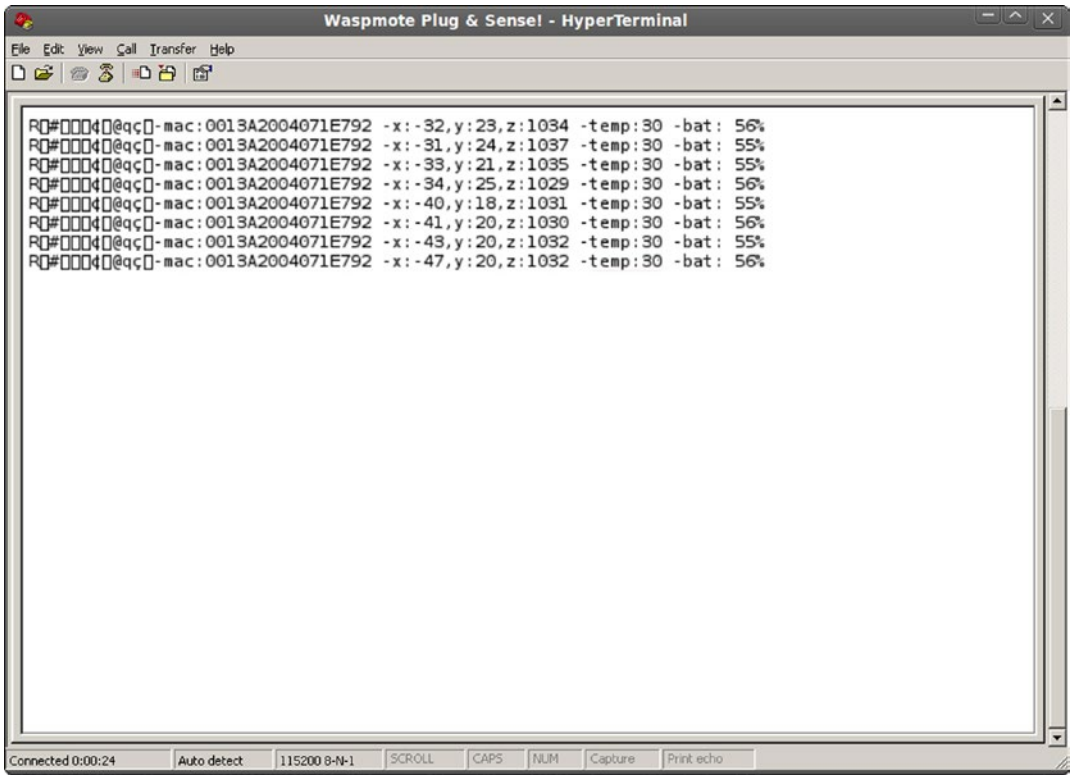


Figure 124: HyperTerminal application capturing Waspmote's output.

23.1.4. Mac-OS receiver

If MAC OS X is used (version later than 10.3.9) the application 'ZTERM' can be used to capture the serial port output. This application can be downloaded from: **http://homepage.mac.com/dalverson/zterm/**

This application is configured automatically, establishing the USB on which Waspmote has been connected and the speed.

The following image shows this application capturing Waspmote's output, while the example code 'Waspmote Accelerator Basic Example' is run.

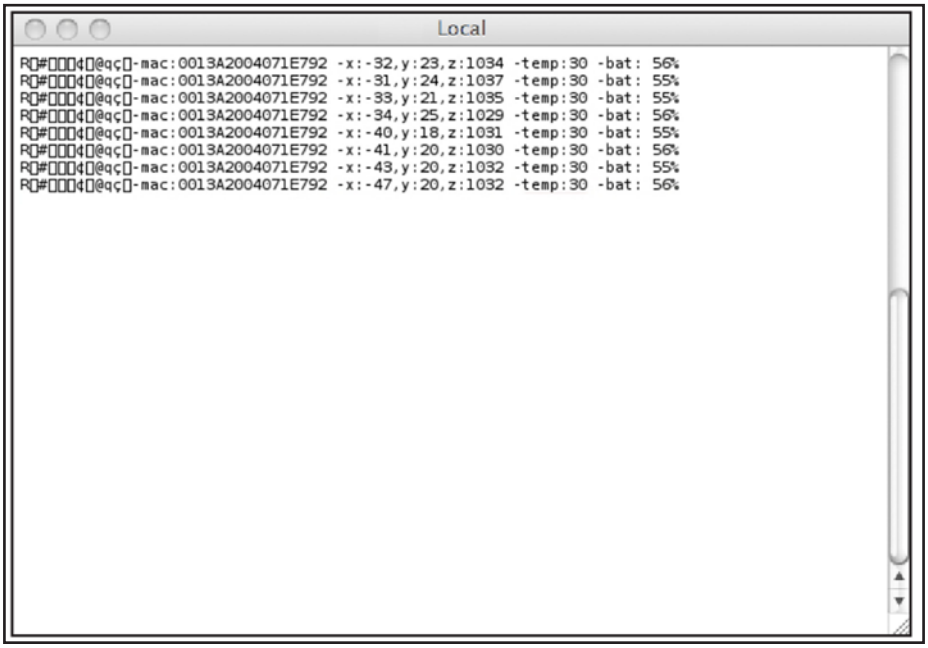


Figure 125: Waspmote's output capture

23.2. Meshlium

Figure 126: Meshlium router

Meshlium is a Linux router which works as the Gateway of the Waspmote Sensor Networks. It can contain 5 different radio interfaces: WiFi 2.4GHz, WiFi 5GHz, 3G/GPRS, Bluetooth and **ZigBee.** As well as this, Meshlium can also integrate a GPS module for mobile and vehicular applications and be solar and battery powered. These features a long with an aluminium IP-65 enclosure allows Meshlium to be placed anywhere outdoor. Meshlium comes with the Manager System, a web application which allows to control quickly and easily the WiFi, ZigBee, Bluetooth and 3G/GPRS configurations a long with the storage options of the sensor data received.

The new Meshlium Xtreme allows to detect iPhone and Android devices and in general any device which works with WiFi or Bluetooth interfaces. The idea is to be able to measure the amount of people and cars which are present in a certain point at a specific time, allowing the study of the evolution of the traffic congestion of pedestrians and vehicles.

More info: **http://www.libelium.com/meshlium**

23.2.1. What can I do with Meshlium?

- Connect your ZigBee network to Internet through Ethernet and 3G/GPRS
- Store the ZigBee sensor data in a local or external data base in just one click!

- Create a WiFi Mesh Network in just two steps!
- Set a WiFi Access point in 1 minute

- Discover Bluetooth users and store their routes
- Trace the GPS location and store it in a local or external database in real time

23.2.2. How do they work together?

Meshlium receives the sensor data sent by Waspmote using its ZigBee radio.

Then 4 possible actions can be performed:

1. Store the sensor data in the Meshlium Local Data Base (MySQL)
2. Store the ZigBee sensor data in an External Data Base (MySQL)
3. Send the information to the Internet using the Ethernet or WiFi connection
4. Send the information to the Internet using the 3G/GPRS connection

23.2.2.1. Meshlium Storage Options

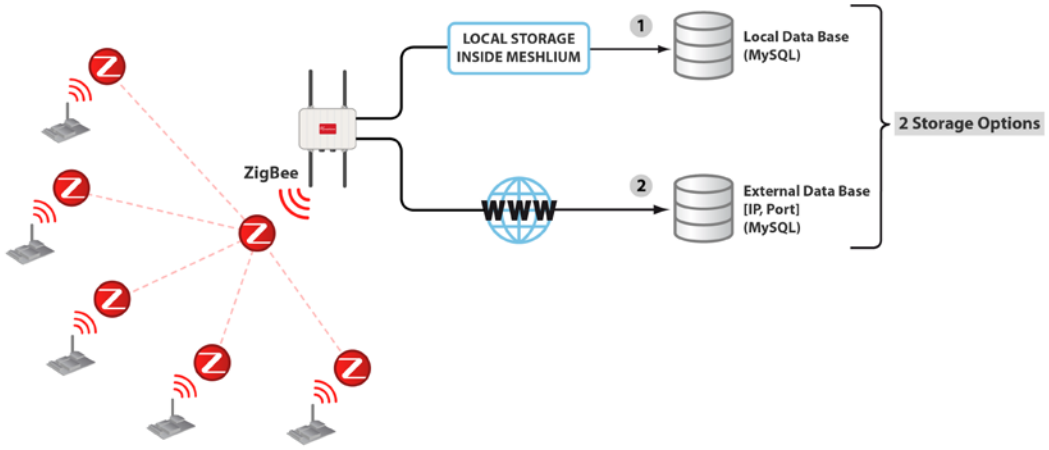


Figure 127: Meshlium Storage Options

- Local Data Base
- External Data Base

23.2.2.2. Meshlium Connection Options

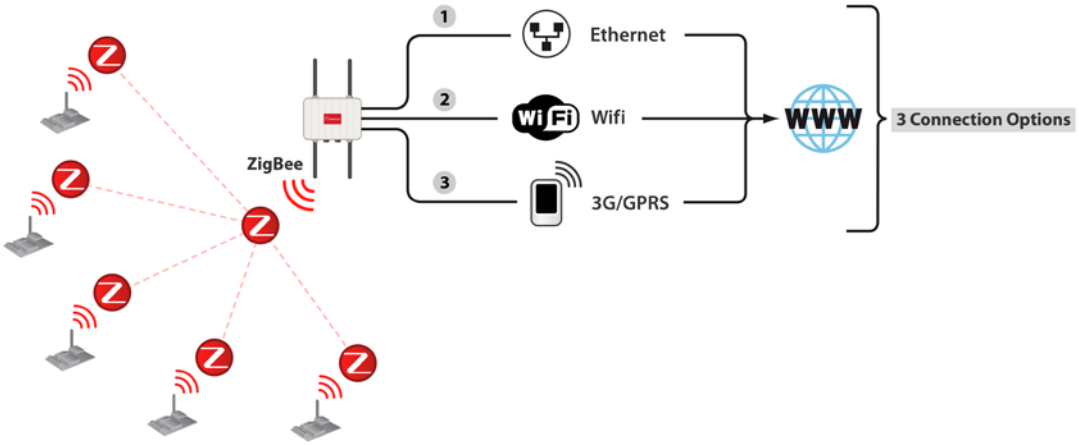


Figure 128: Meshlium Connection Options

- ZigBee → Ethernet
- ZigBee → WiFi
- ZigBee → 3G/GPRS

23.2.3. Capturing and storing sensor data in Meshlium from a Waspmote sensor network

When you buy a kit containing Waspmotes, Gateway and Meshlium, the Waspmotes come already configured to send frames to the Gateway. Later, once the user has developed the code for transmitting to Gateway, he can switch to Meshlium.

Meshlium will receive the sensor data sent by Waspmote using the ZigBee radio and it will store the frames in the Local Data Base. That can be done in an automatic way now thanks to the new **Sensor Parser**.

The **Sensor Parser** is a new feature for Meshlium (version 3.0.5 or older). It is a new software system which is able to do the following tasks in an easy and transparent way:

- receive frames from XBee (with the Data Frame format)
- parse these frames
- store the data in a local Database
- synchronize the local Database with an external Database

Besides, the user can add his own sensors.

The initial ZigBee frames sent by Waspmote contain the next sequence (XBee API frame characters are removed here):

```
<=>\0x80\0x03#35689722##7#ACC:80;10;987#IN_TEMP:22.50#BAT:93#
```

They are formed by the accelerometer values, RTC internal temperature value, and battery level. The MAC address is added and other helpful information.

Meshlium comes with all the radios ready to be used. Just "plug & mesh!". All the Meshlium nodes come with the WiFi AP ready so that users can connect using their WiFi devices. Connect the ethernet cable to your network hub, restart Meshlium and it will automatically get an IP from your network using DHCP *.

(*) For the Meshlium Mesh AP and for the Meshlium ZigBee Mesh AP the Internet connection depends on the GW of the network.

Then access Meshlium through the WiFi connection. First of all search the available access points and connect to "Meshlium".

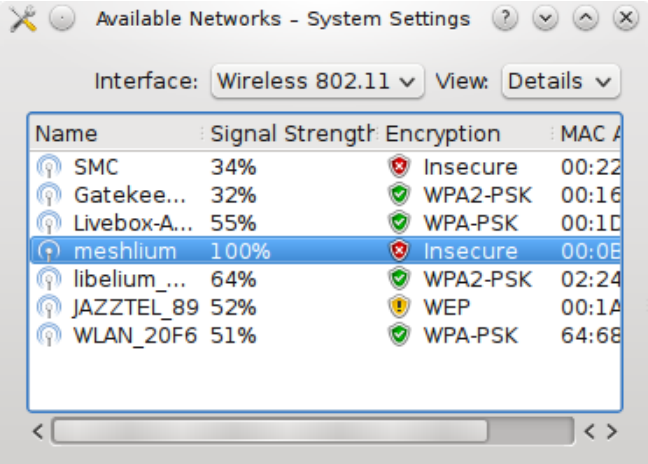


Figure 129: Available Networks screenshot

No password is needed as the network is public (you can change it later in the WiFi AP Interface options). When you select it, Meshlium will give an IP from the range 10.10.10.10 - 10.10.10.250.

Now you can open your browser and access to the Meshlium Manager System:

- **URL:** http://10.10.10.1/ManagerSystem
- **user:** root
- **password:** libelium

Figure 130: Meshlium Manager System Login screen

Now we go to the "Sensor Networks" tab.

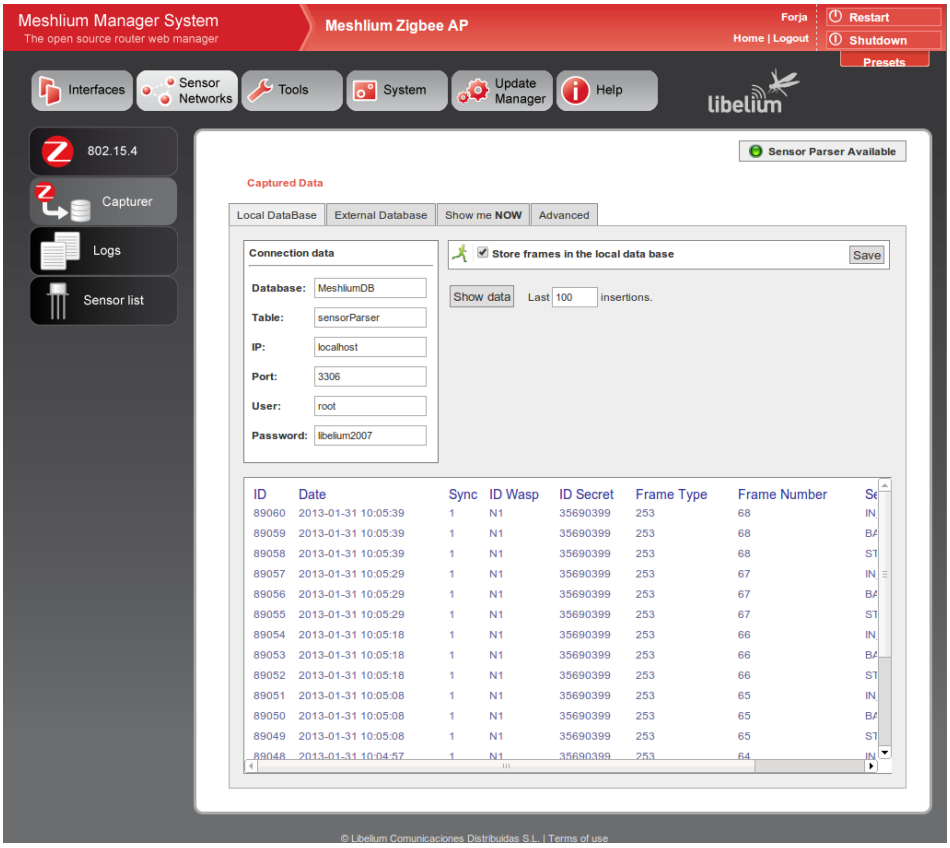


Figure 131: Sensor Networks tab

There are 5 different XBee models can be configured:

Figure 132: ZigBee radio models

Depending the kind of XBee model the parameters to be configured may vary.

Complete list:

- **Network ID:** Also known as PAN ID (Personal Arena Network ID)
- **Channel:** frequency channel used
- **Network Address:** 16b address (hex field) - MY
- **Node ID:** maximum 20 characters (by default "Meshlium")
- **Power level:** [0..4] (by default 4)
- **Encrypted mode:** true/false (by default false)
- **Encryption Key:** 16 characters maximum
- **MAC:** 64b hardware address. It is a read only value divided in two parts:
  - MAC-high: 32b (hex field)
  - MAC-low: 32b (hex field)

These parameters must be also configured in the Waspmote sensor nodes. Access to all the information related to Waspmote at: **http://www.libelium.com/waspmote**

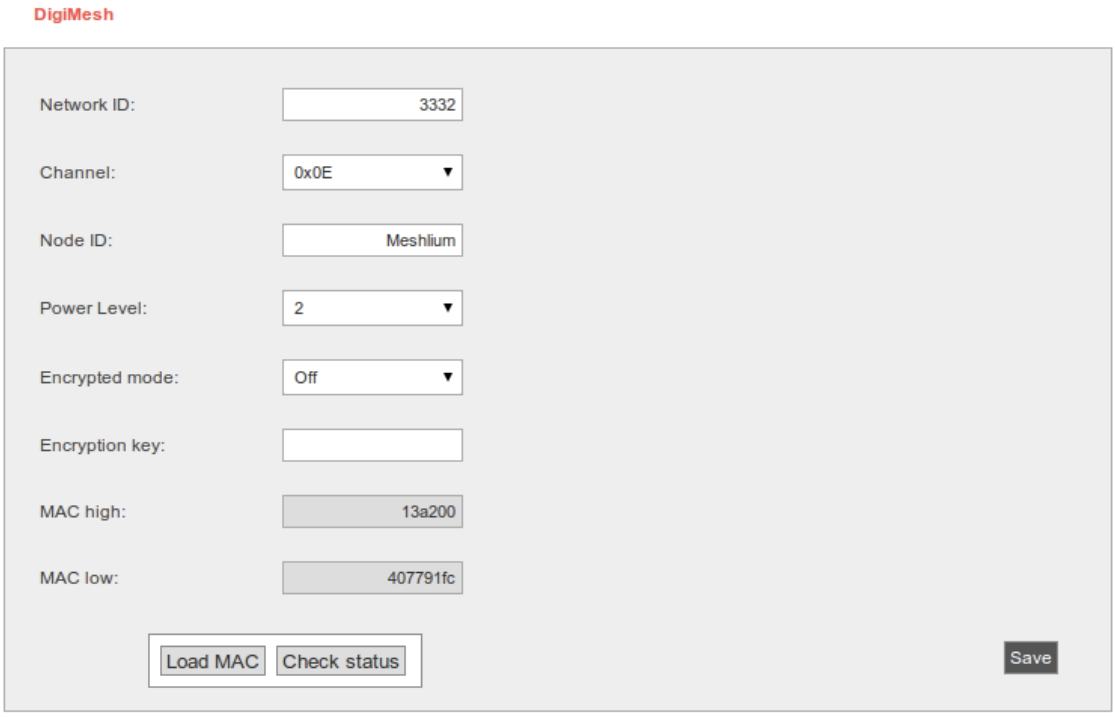


Figure 133: XBee parameters configuration

To discover the MAC address of the XBee module just press the "Load MAC" button.

The "Check status" option allows to see if the ZigBee radio is working properly and if the configuration stored on it matches the values set in the Manager System.

Both process ("Load MAC" and "Check status") require the ZigBee capturer daemon to be stopped. This means no frames will be received while executing this actions. Be patient this can take up to 1 minute to finish.

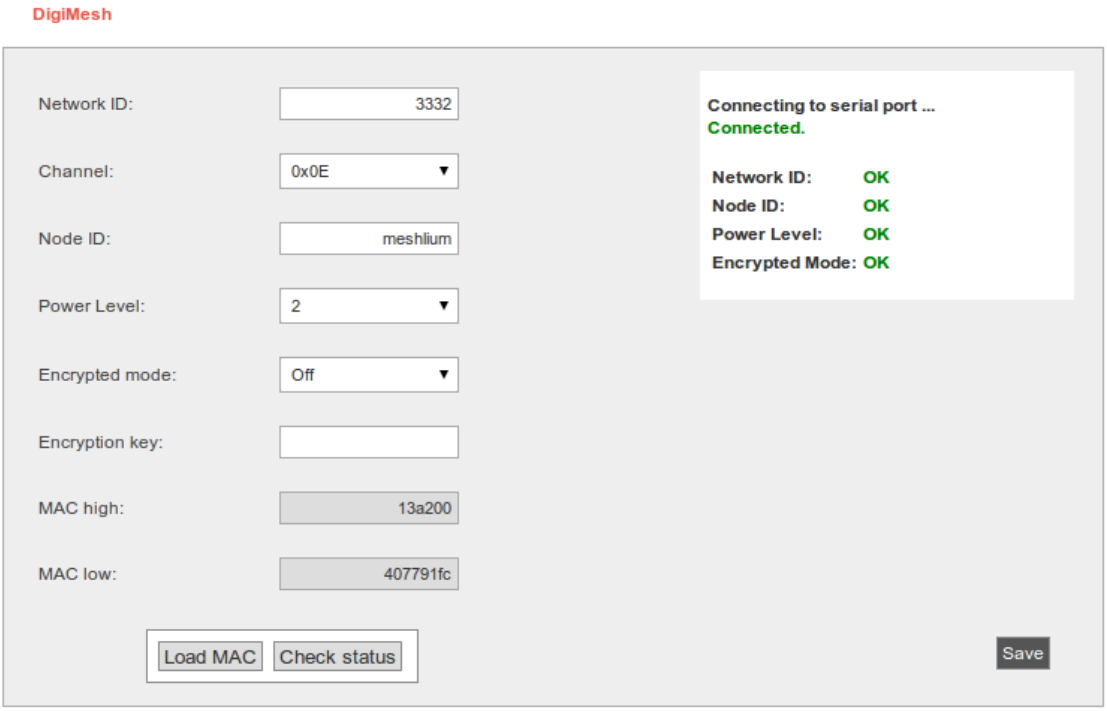


Figure 134: XBee parameters configuration

Note: *When you buy a Waspmote Developer kit with Meshlium and with the XBee ZB as ZigBee radio both the Waspmote GW and Meshlium come configured as Coordinator of the network. Take into account that only one of them can be working at the same time.*

Note: *If the encryption check fails but the rest of parameters are OK, it means the ZigBee radio has an old version of the firmware but it is working perfectly.*

• Capturing and storing sensor data

As said before, in a kit containing Waspmotes, Gateway and Meshlium, the Waspmotes come already configured to send frames to the Gateway. Later, once the user has developed the code for transmitting to Gateway, he can switch to Meshlium.

Meshlium will receive the sensor data sent by Waspmote using the ZigBee radio and it will store the frames in the Local Data Base. That can be done in an automatic way now thanks to the new **Sensor Parser**.

The **Sensor Parser** is a new feature for Meshlium (version 3.0.5 or older). It is a new software system which is able to do the following tasks in an easy and transparent way:

- receive frames from XBee (with the Data Frame format)
- parse these frames
- store the data in local Database
- synchronize the local Database with an external Database

Besides, the user can add his own sensors.

The initial ZigBee frames sent by Waspmote contain the next sequence (XBee API frame characters are removed here):

```
<=>\0x80\0x03#35689722##7#ACC:80;10;987#IN_TEMP:22.50#BAT:93#
```

They are formed by the accelerometer values, RTC internal temperature value, and battery level. The MAC address is added and other helpful information.

In order to add your own sensor frames properly go to the section "Sensors". All frames captured will be able to stored on Local Database, however the frame has not been defined is stored in the database. See the picture below in order to see different frames types and how they are saved in the database.

ID	Date	Sync	ID Wasp	ID Secret	Frame Type	Frame Number	Se
87493	2013-01-31 08:33:38	0	N1	35690399	253	57	IN
87492	2013-01-31 08:33:38	0	N1	35690399	253	57	BA
87491	2013-01-31 08:33:38	0	N1	35690399	253	57	ST
87489	2013-01-31 08:33:27	0	<=>#35690399#N1#56#STR:XBee frame#BAT:90#IN_TE				
87488	2013-01-31 08:33:17	1	N1	35690399	253	55	IN
87487	2013-01-31 08:33:17	1	N1	35690399	253	55	BA
87486	2013-01-31 08:33:17	1	N1	35690399	253	55	ST
87485	2013-01-31 08:33:06	1	N1	35690399	253	54	IN
87484	2013-01-31 08:33:06	1	N1	35690399	253	54	BA
87483	2013-01-31 08:33:06	1	N1	35690399	253	54	ST
87482	2013-01-31 08:32:56	1	N1	35690399	253	53	IN
87481	2013-01-31 08:32:56	1	N1	35690399	253	53	BA
87480	2013-01-31 08:32:56	1	N1	35690399	253	53	ST

Figure 135: Different frames types

In order to work with new sensor information added to the ZigBee frames go to the "Capturing and Storing new sensor data frames" chapter.

If you change any of the parameters in Waspmote or Meshlium you will have to do it in both platforms so that they still can communicate.

We can perform two different storage options with the ZigBee frames captured:

- Local Data Base
- External Data Base

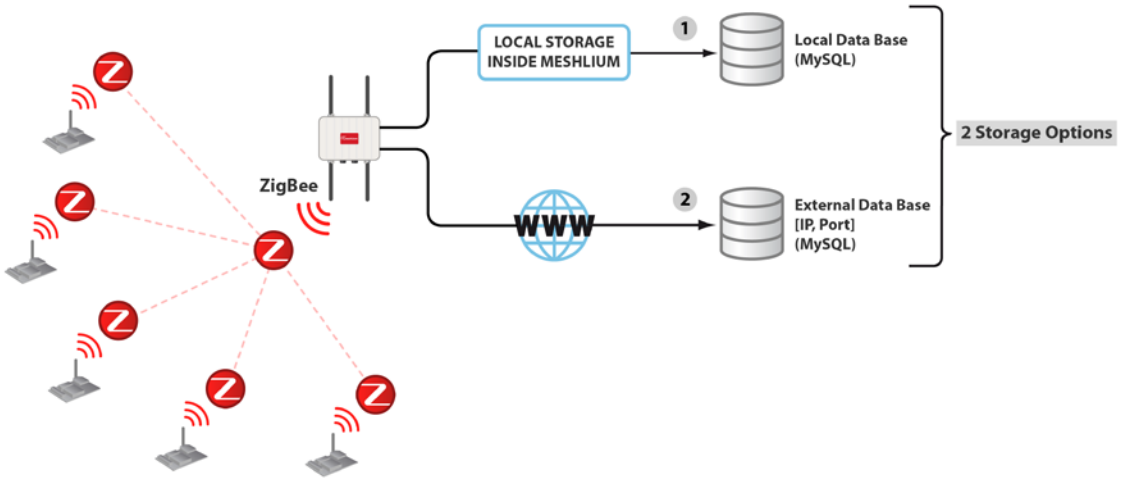


Figure 136: Meshlium Storage options

You can also send the information received to the Internet using the Ethernet, WiFi and 3G/GPRS interfaces.

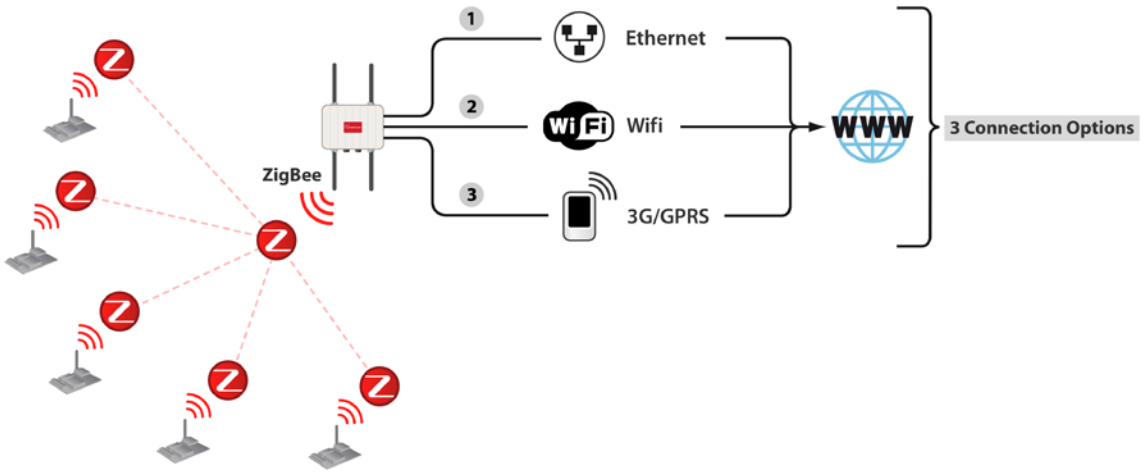


Figure 137: Meshlium Connection options

Local Data Base

Meshlium has a MySQL data base up and running which is used to store locally the information captured. In the "Local Data Base" tab you can see the connection parameters.

- **Database:** MeshliumDB
- **Table:** sensorParser
- **IP:** localhost / 10.10.10.1 *
- **Port:** 3306
- **User:** root
- **Password:** libelium2007

You can change the password, see the "Users Manager" section.

(*) Depending on the parameters set in the "Interfaces" section.

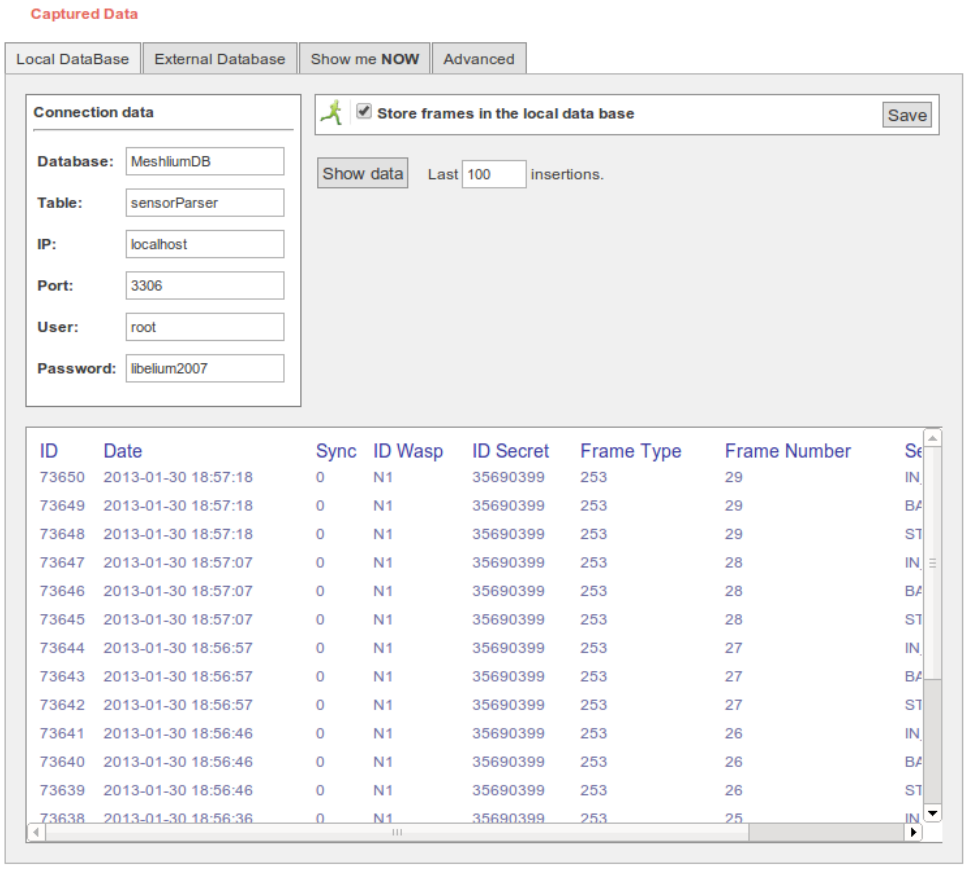


Figure 138: Local Data Base tab

Steps:

1. Set the check box "Store frames in the local data base" and press the "Save" button.

From this time Meshlium will automatically perform Scans and will store the results in the Local Data Base. This process will also continue after restarting Meshlium.

At any time you can see the last "x" records stored. Just set how many insertions you want to see and press the "Show data" button.

External Data Base

Meshlium can also store the information captured in an External Data Base.

Steps:

1. Pressing the "Show sql script" you will get the code needed to create the data base along with the table and the right privileges.

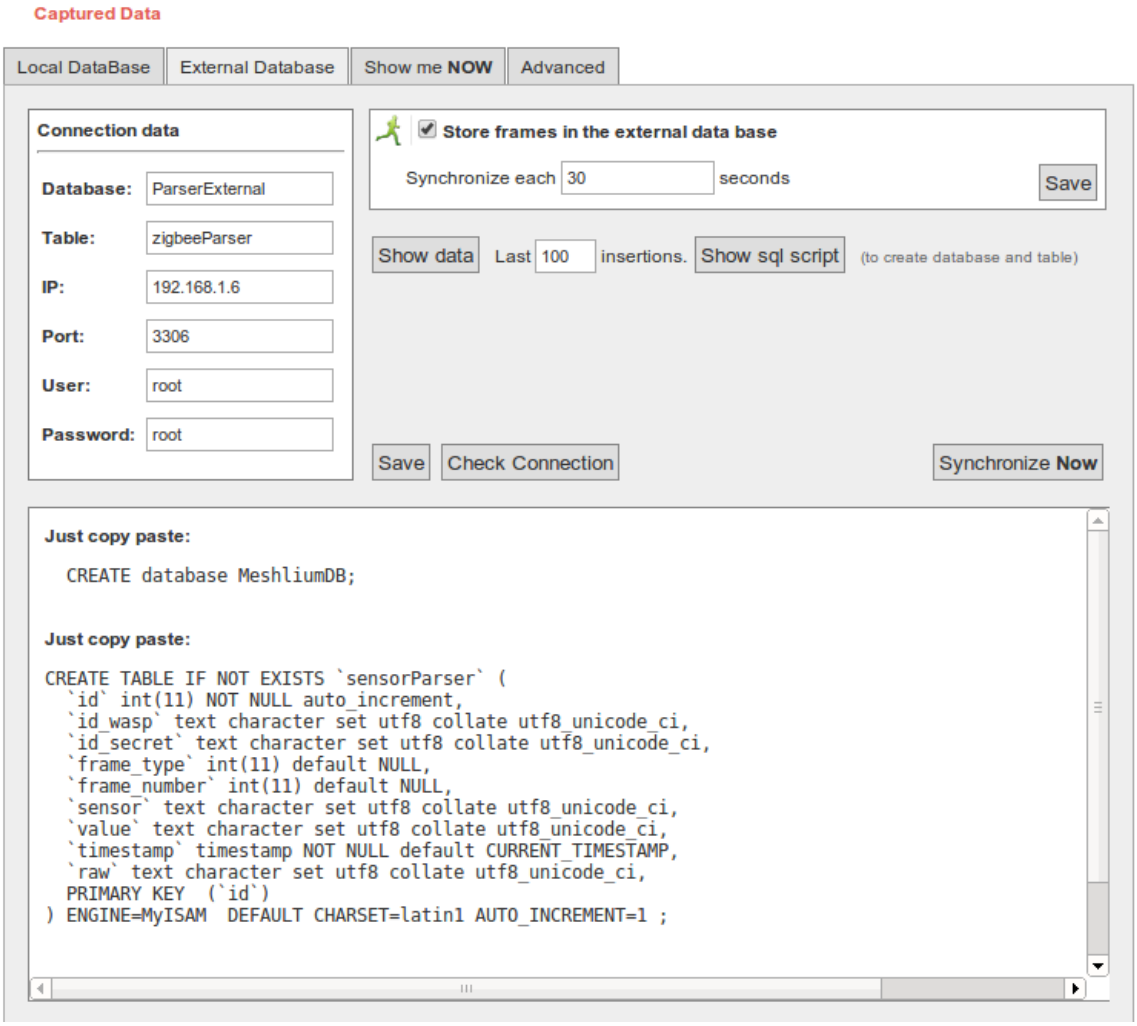


Figure 139: External Database tab - showing SQL Script

2. Insert this code in your MySQL management application.

3. Fill the Connection Data fields with the information about where the data base is located (IP, Port) and with the authentication options (Database, Table, User, Password).

This data are stored in `/mnt/lib/cfg/sensorExternalDB` file.

4. Now press the "Check Connection" button to see if the configuration is correct.

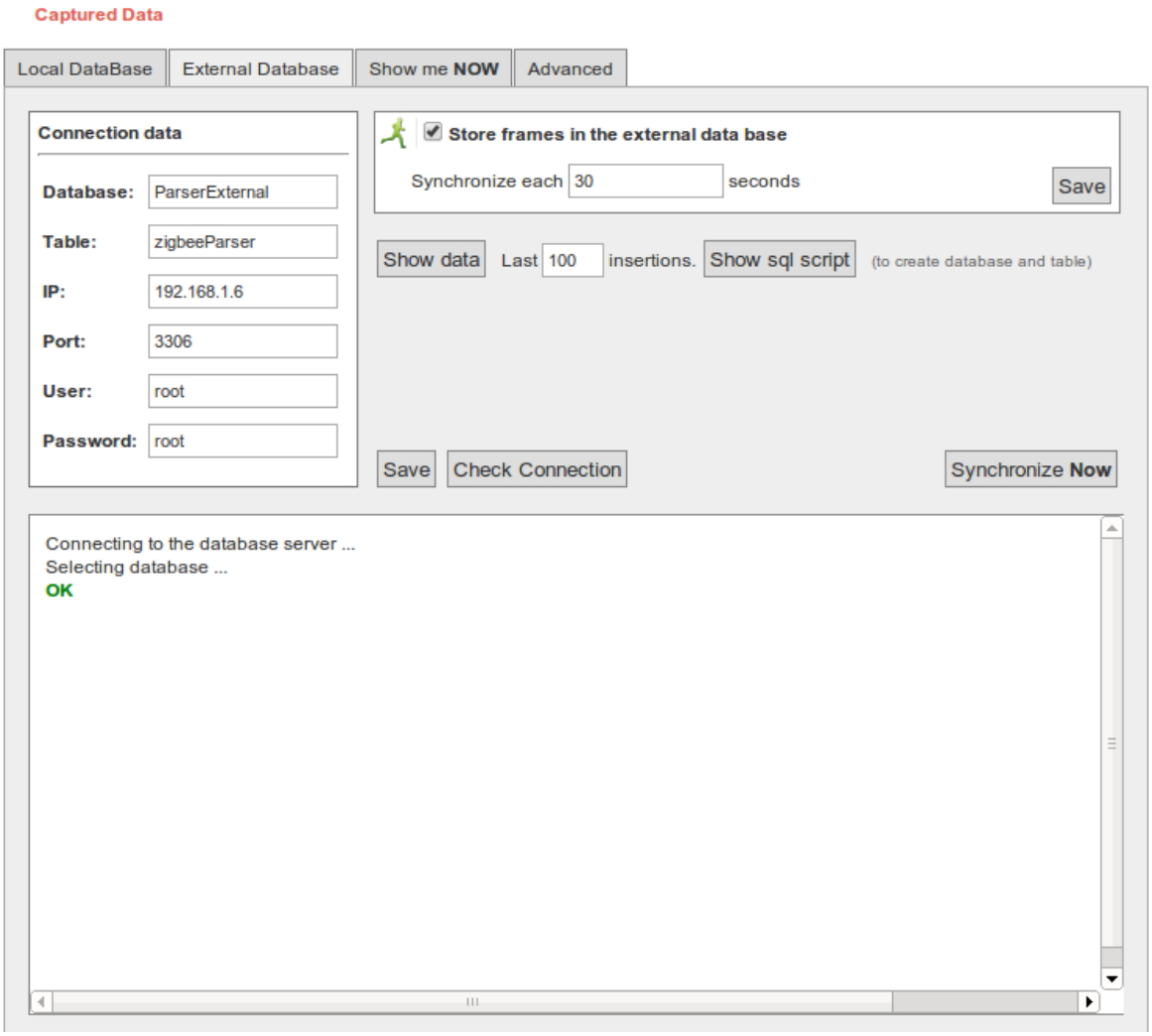


Figure 140: External Database tab - checking connection

5. Set the check box "Store frames in external database", you can defined the interval how often to synchronize the local database with external database and press the "Save" button.

From this time Meshlium will automatically perform Scans and will store the results in the External Data Base each . This process will also continue after restarting Meshlium.

You can also choose to sync when you want. Just press the "Synchronize Now" button.

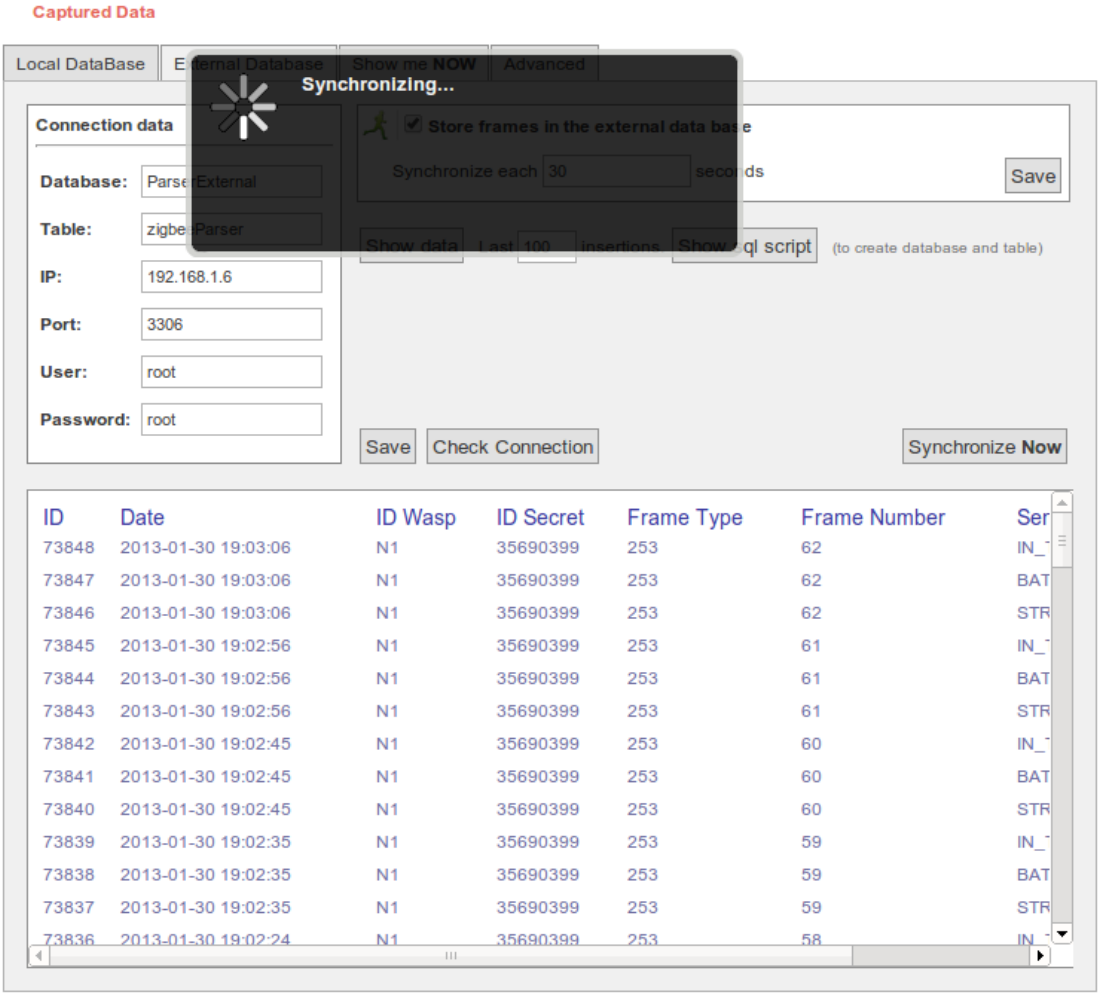


Figure 141: External Database tab - Synchornize

At any time you can see the last "x" records stored. Just set how many insertions you want to see and press the "Show data" button.

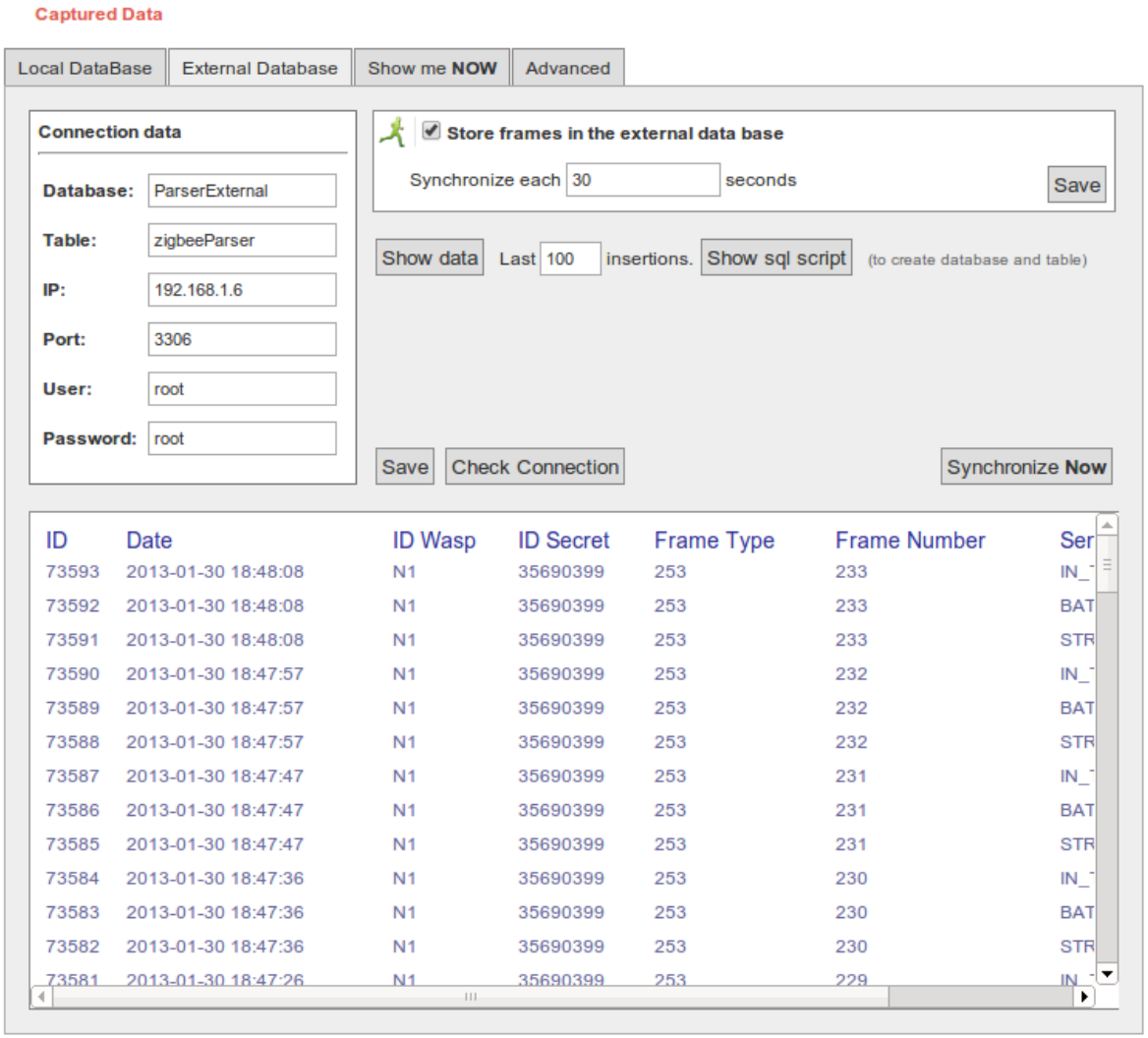


Figure 142: External Database tab - last "x" records stored

Show me now!

In the "Show me now!" tab you can see in real time the Scans captured.

You can specify if you want the information to be updated periodically with the defined interval just checking the "Use the Defined Interval" button.

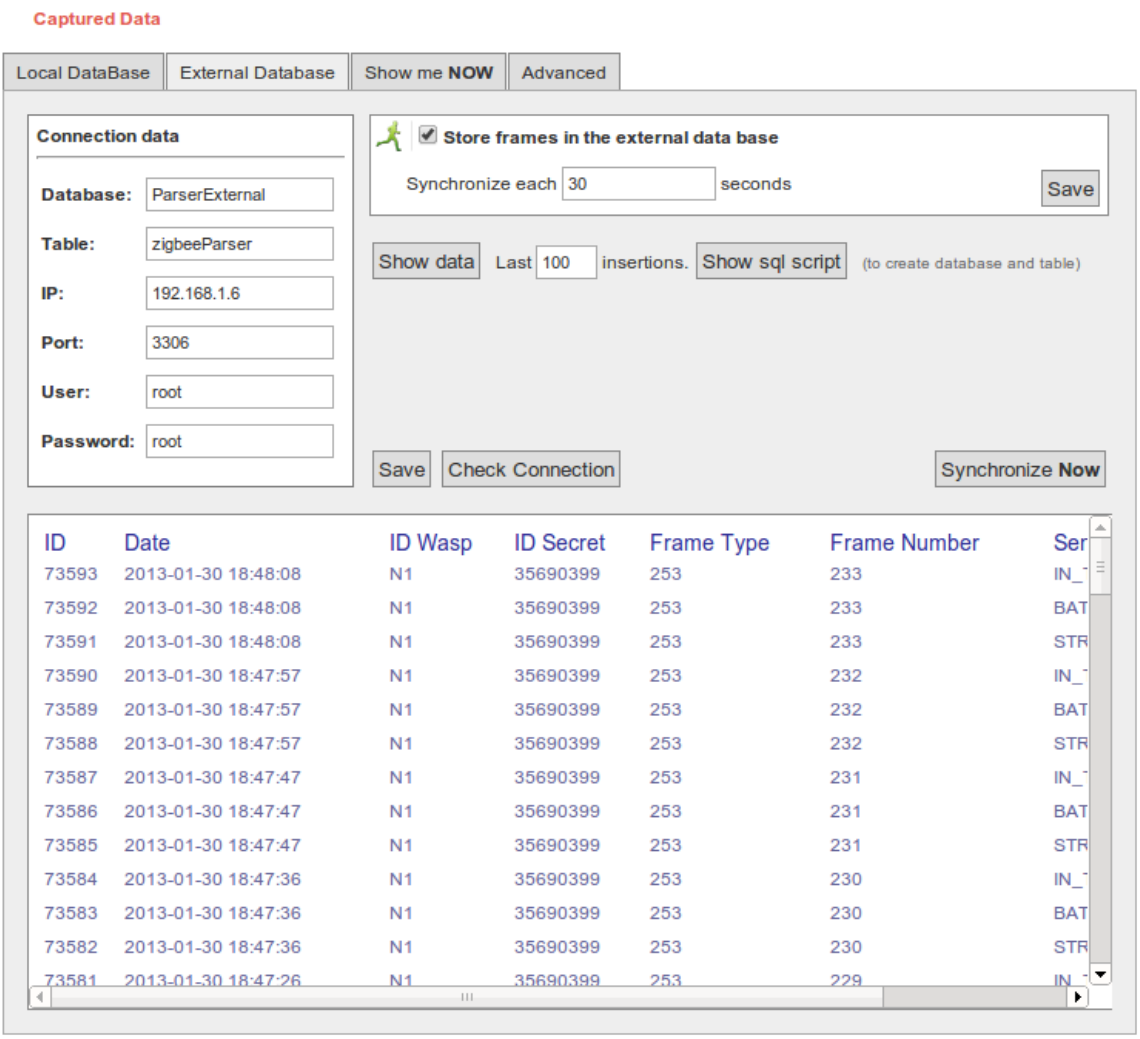


Figure 143: Show me now! tab

Advanced Database

In the "Advanced" tab you can see information about the state in which they are databases.

It displays information about the Loca and Externall database, showing the following information:

- Local and External Database names
- Local and External Database sizes
- Local and External Tables
- Total Local and External Entries
- Synchronized Local Frames
- Unsynchronized Local Frames

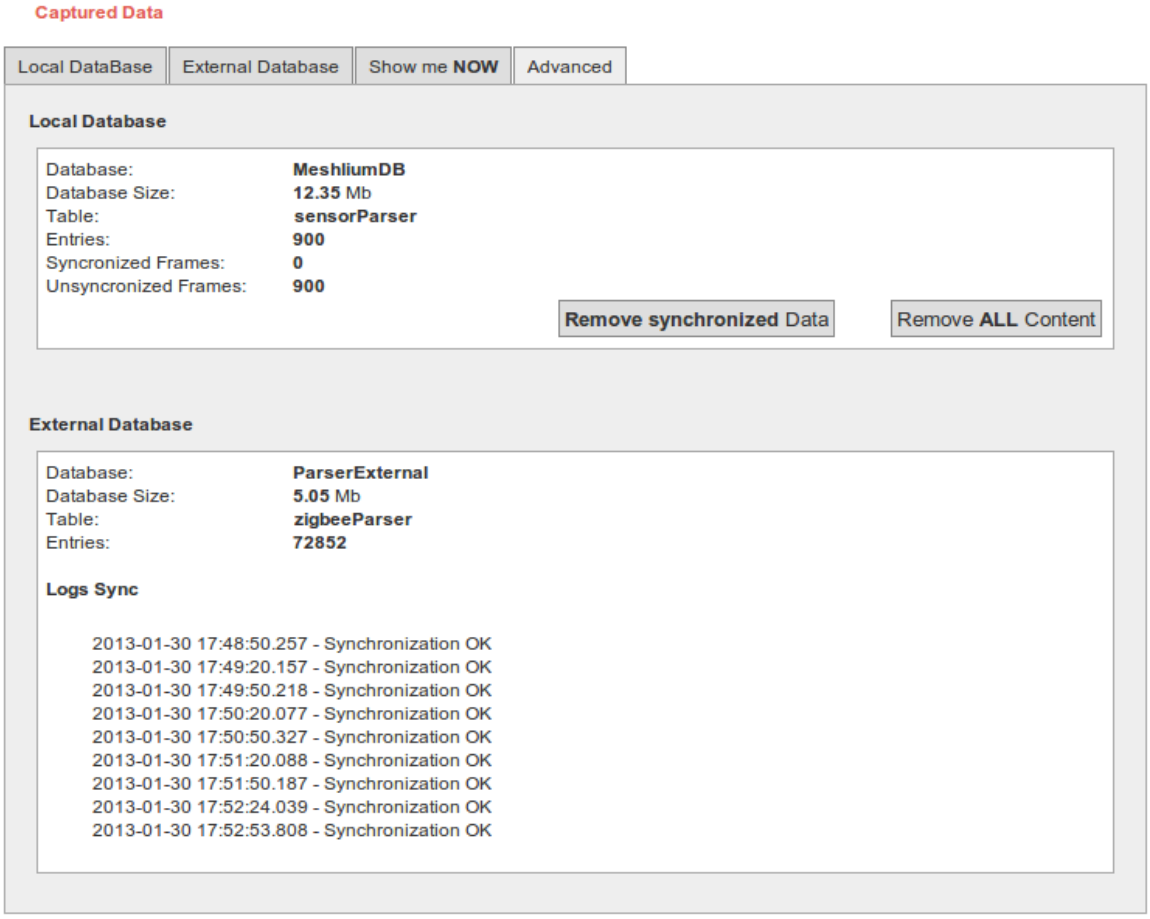


Figure 144: Advanced Tab

From this tab, **you can delete all the information contained in the Local database** or **Remove synchronized data**. Before performing these actions, a confirmation message will be displayed.

Note: *Before running these options, it is recommended to have a backup or having synchronized your local database with external database.*

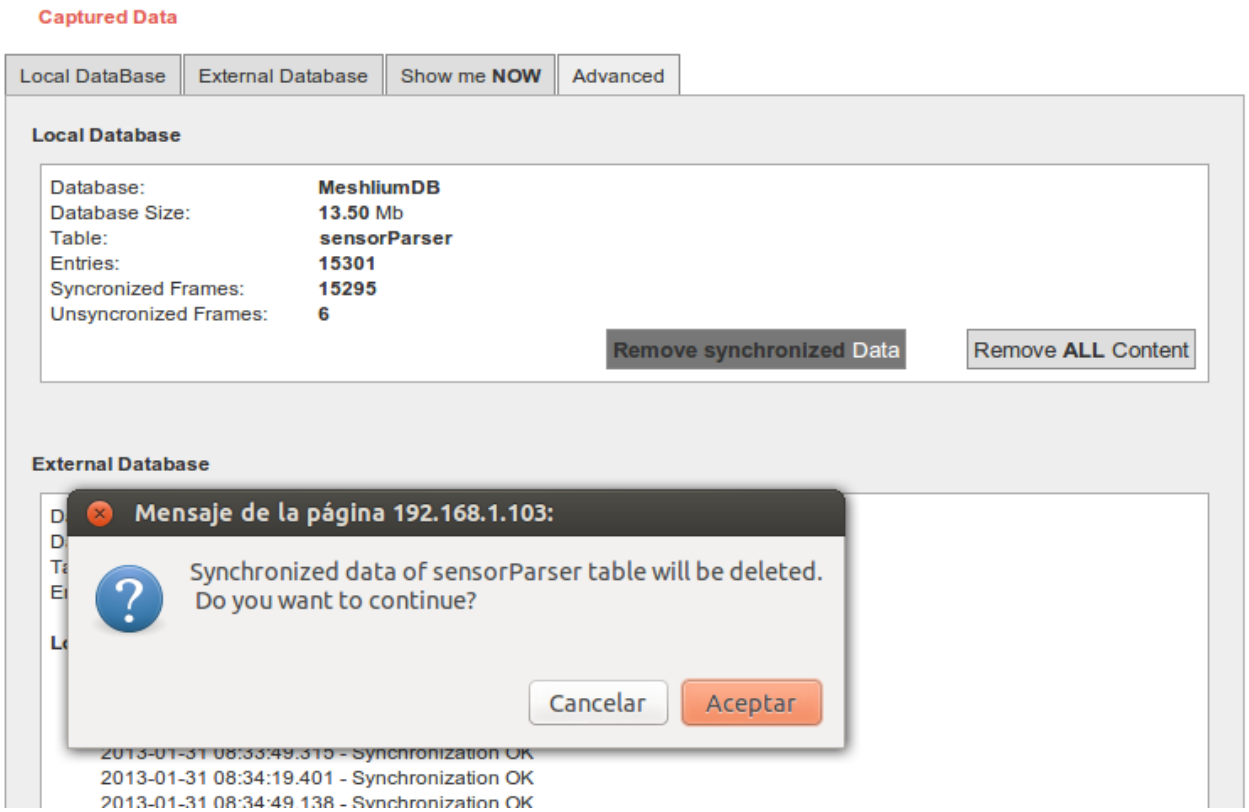


Figure 145: Advanved Tab – Remove data

In addition can display a log of the date of the last synchronization between the local database and external database was successful.

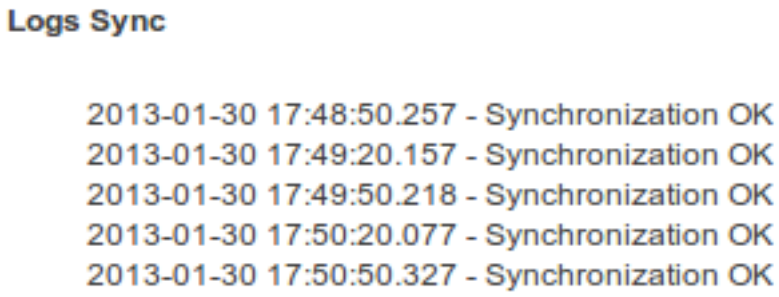


Figure 146: Advanved Tab – Synchronization log

23.2.4. Capturer logs

Inside "Sensor Networks" exists the section **Logs**, in this section you can see the last frames received on Meshlium.

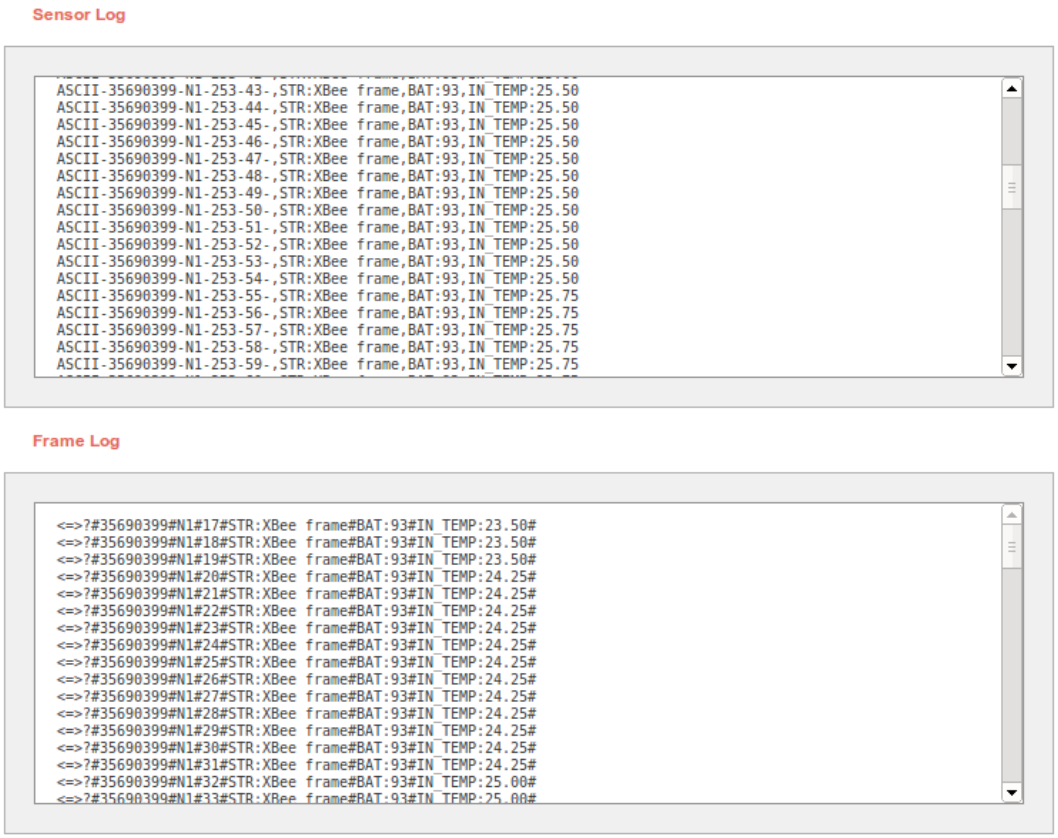


Figure 147: Sensor log

First show the "sensor log", in this logs shows the frames are stored after being processed.

```
ASCII-35690399-N1-253-198-,STR:XBee frame,BAT:93,IN_TEMP:31.50
```

secondly shown "Frame Log", in this logs shows the frames stored as the arrive to Meshlium.

```
<=>?#35690399#N1#198#STR:XBee frame#BAT:93#IN_TEMP:31.50#
```

23.2.5. Sensors

In section "Sensor List", the user can **add new sensors or delete sensors.**

By default Meshlium recognize all Libelium official sensors frames. All sensors frames that Meshlium can capture and store must be specified in an XML file.

The file with official sensors of Libelium is localed in `/mnt/lib/cfg/parser/sensors.xml`

The button "update sensors" update the Libelium official sensor. User sensors remaining unchanged.

Users can add and remove sensors in an easy and simple from ManagerSystem.

To add a new sensor the user must complete the fields:

- ASCII ID: sensor id for ASCII frame.
- Fields: This field specifies the number of sensor fields sent in the frame. This helps to calculate the frame length.
- Type: type of fields
  - uint8_t
  - int
  - float
  - string
  - ulong
  - array(ulong)

Once all fields are filled in, click on the button "Add sensor"

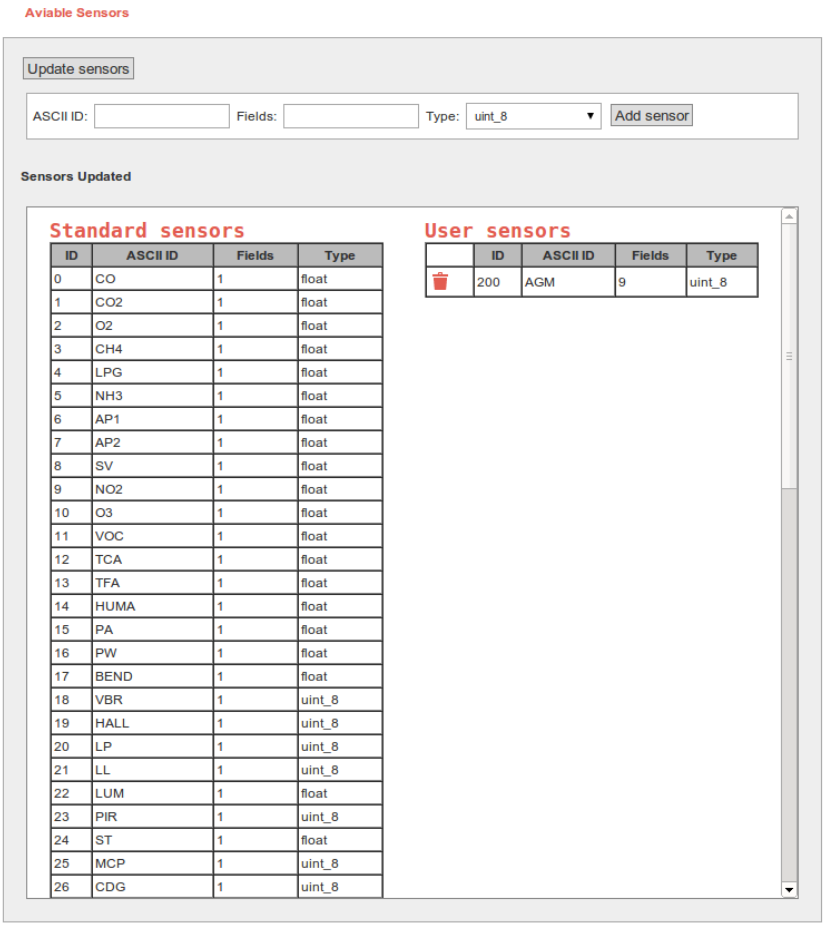


Figure 148: Sensor List – Addition

The new user sensors will be added to the new XML file, the file with user sensors is localed in /mnt/lib/cfg/parser/user_sensors.xml

> ***Note:*** *In "Waspmote data frame guide" document is located more extensive information about how to build the frame.*

To delete sensor the user must press the garbage can that appears to the left of the description of the sensor. To complete the action should accept a confirmation message.

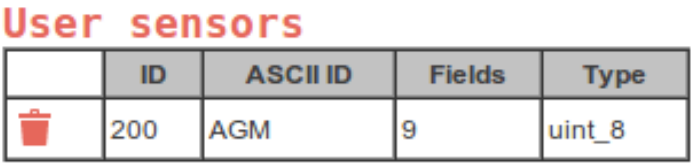


Figure 149: Sensor List – Remove

23.2.6. Sending ZigBee frames from Meshlium to Waspmote

Meshlium can also send ZigBee frames to the Waspmote nodes. In order to use this feature you have to stop the "capturing and storing" daemon which is running in the system.

To do so access by SSH to Meshlium and stop the default ZigBee daemon::

```
$ /etc/init.d/ZigbeeScanD.sh stop
```

Now you can execute the ZigBeeSend command. There are several ways to send information to a node:

- Using its 802.15.4 MAC address (64b)
- Using its Network address (MY) (16b)
- Performing a broadcast transmission

Sending to Waspmote using its MAC address (64b):

```
$ ./ZigBeeSend -mac 0013a2004069165d "Hello Waspmote!"
```

Sending to Waspmote using its Net address (MY - 16b):

```
$ ./ZigBeeSend -net 1234 "hello Waspmote!"
```

Send to all the Waspmote devices at the same time - Broadcast mode:

```
$ ./ZigBeeSend -b "hello everybody!"
```

The source code "ZigbeeSend.c" and the reception program to be installed in Waspmote can be downloaded from the Meshlium Development section: **http://www.libelium.com/development/meshlium**

You can download these files and change them in order to get new features and sending options.

Compilation:

The compilation can be done in the same Meshlium. Just copy these files in a folder accessing by SSH and execute:

```
$ gcc -o ZigBeeSend ZigBeeSend.c -lpthread
```

Important: *If you want to create a "ZigBee sending" daemon that is executed each time Meshlium starts you have to deactivate the "ZigBee Capturer" daemon (/etc/init.d/ZigbeeScanD.sh) as the ZigBee radio has to be used by one process at a time.*

You will find support in the Libelium Forum at: **http://www.libelium.com/forum**

24. Documentation Changelog

From v4.4 to v4.5

- Added "Non-Rechargeable Battery" warning

From v4.3 to v4.4

- Deleted references to OTA reset

From v4.2 to v4.3

- Added references to OTA with 3G/GPRS/WiFi via FTP
- Magnet reset reference in Plug & Sense!
- Note about consumption in sleep modes (XBee + Waspmote + SD card)
- Reference to next line of calibrated Gas Board
- Some changes in Recommendations of Use
- Update for the new WiFi library
- Errata correction

From v4.1 to v4.2

- Added references to 3G module
- Better IDE explanation on Linux
- Some errata and better explanations

25. Certifications

25.1. CE

In accordance with the 1999/05/CE directive, Libelium Comunicaciones Distribuidas S.L. declares that the Waspmote device conforms to the following regulations:

EN 55022:1998

EN 55022:1998/A1:2000

EN 55022:1998/A2:2003

EN 61000-4-3:2002

EN 61000-4-3/A1:2002

EN 61000-4-3:2006

UNE-EN 60950-1:2007

Compliant with ETSI EN 301 489-1 V1.6.1, EN 300 328, Date: March 26, 2009

If desired, the Declaration of Conformity document can be requested using the Contact section at:

http://www.libelium.com/contact

Waspmote is a piece of equipment defined as a wireless sensor capture, geolocalization and communication device which allows:

- short and long distance data, voice and image communication
- capture of analog and digital sensor data directly connected or through probes
- wireless access enablement to electronic communication networks as well as local networks allowing cable free connection between computers and/or terminals or peripheral devices
- geospatial position information
- interconnection of wired networks with wireless networks of different frequencies
- interconnection of wireless networks of different frequencies between each other
- output of information obtained in wireless sensor networks
- use as a data storage station
- capture of environmental information through interface interconnection, peripherals and sensors
- interaction with the environment through the activation and deactivation of electronic mechanisms (both analog and digital)

25.2. FCC

Waspmote models:

Model 1- FCC (XBee PRO series 1 OEM + SIM900 GSM/GPRS module)
FCC ID: XKM-WASP01 comprising
- FCC ID: OUR-XBEEPRO
- FCC ID: UDV-0912142009007

Model 2- FCC (XBee PRO ZB series 2 + SIM900 GSM/GPRS module)
FCC ID: XKM-WASP02 comprising
- FCC ID: MCQ-XBEEPRO2*
- FCC ID: UDV-0912142009007

Model 3 - FCC (XBee 900MHz + SIM900 GSM/GPRS module)
FCC ID: XKM-WASP03 comprising
- FCC ID: MCQ-XBEE09P
- FCC ID: UDV-0912142009007

Installation and operation of any Waspmote model must assure a separation distance of 20 cm from all persons, to comply with RF exposure restrictions.

Module Grant Restrictions

FCC ID OUR-XBEEPRO

The antenna(s) used for this transmitter must be installed to provide the separation distances, as described in this filing, and must not be co-located or operating in conjunction with any other antenna or transmitter. Grantee must coordinate with OEM integrators to ensure the end-users of products operating with this module are provided with operating instructions and installation requirements to satisfy RF exposure compliance. Separate approval is required for all other operating configurations, including portable configurations with respect to 2.1093 and different antenna configurations. Power listed is continuously variable from the value listed in this entry to 0.0095W

FCC ID MCQ-XBEEPRO2

OEM integrators and End-Users must be provided with transmitter operation conditions for satisfying RF exposure compliance. The instruction manual furnished with the intentional radiator shall contain language in the installation instructions informing the operator and the installer of this responsibility. This grant is valid only when the device is sold to OEM integrators and the OEM integrators are instructed to ensure that the end user has no manual instructions to remove or install the device.

FCC ID: UDV-0912142009007

This device is to be used in mobile or fixed applications only. For other antenna(s) not described in this filing the antenna gain including cable loss must not exceed 7.3 dBi in the 850 MHz Cellular band and 12.7 dBi in the PCS 1900 MHz band, for the purpose of satisfying the requirements of 2.1043 and 2.1091. The antenna used for this transmitter must be installed to provide a separation distance of at least 20 cm from all persons, and must not be co-located or operating in conjunction with other antennas or transmitters within a host device, except in accordance with FCC multi- transmitter product procedures. Compliance of this device in all final product configurations is the responsibility of the Grantee. OEM integrators and end-users must be provided with specific information required to satisfy RF exposure compliance for all final host devices and installations.

25.3. IC

Waspmote models:

Model 1- IC (XBee PRO series 1 OEM + SIM900 GSM/GPRS module)
IC: 8472A-WASP01 comprising
- IC: 4214A-XBEEPRO
- IC: 8460A-20100108007

Model 2- IC (XBee PRO ZB series 2 + SIM900 GSM/GPRS module)
IC: 8472A-WASP02 comprising
- IC: 1846A-XBEEPRO2
- IC: 8460A-20100108007

Model 3- IC (XBee 900MHz + SIM900 GSM/GPRS module)
IC: 8472A-WASP03 comprising
- IC: 1846A-XBEE09P
- IC: 8460A-20100108007

The term "IC:" before the equipment certification number only signifies that the Industry Canada technical specifications were met.

Installation and operation of any Waspmote model must assure a separation distance of 20 cm from all persons, to comply with RF exposure restrictions.

25.4. Use of equipment characteristics

- Equipment to be located in an area of restricted access, where only expert appointed personnel can access and handle it.
- The integration and configuration of extra modules, antennas and other accessories must also be carried out by expert personnel.

25.5. Limitations of use

The ZigBee/IEEE 802.15.4 module has a maximum transmission power of 20dBm.

It is regulated according to EN 301 489-1 v 1.4.1 (202-04) and EN 301 489-17 V1.2.1 (2002-08). The configuration software must be used to limit to a maximum power of 12'11dBm (PL=0).

The 868MHz XBee module has a maximum transmission power of 27dBm. This module is regulated only for use in Europe.

The 900MHz XBee module has a maximum power of 20dBm. This module is regulated only for use in the United States.

The GSM/GPRS module has a power of 2W (Class 4) for the 850MHz/900MHz band and 1W (Class 1) for the 1800MHz and 1900MHz frequency band.

The 3G/GPRS module has a power of 0,25W for the UMTS 900MHz/1900MHz/2100MHz band, 2W for the GSM 850MHz/900MHz band and 1W DCS1800MHz/PCS1900MHz frequency band.

Important: *In Spain the use of the 850MHz band is not permitted. For more information contact the official organisation responsible for the regulation of power and frequencies in your country.*

The cable (pigtail) used to connect the radio module with the antenna connector shows a loss of approximately 0.25dBi for GSM/GPRS.

The broadcast power at which the WiFi, XBee 2.4GHz, XBee 868MHz, XBee 900MHz operate can be limited through the configuration software. It is the responsibility of the installer to choose the correct power in each case, considering the following limitations:

The broadcast power of any of the modules added to that of the antenna used minus the loss shown by the pigtail and the cable that joins the connector with the antenna (in the event of using an extra connection cable) must not exceed 20dBm (100mW) in the 2.4GHz frequency band and 27dBm for the 868MHz band, according to the ETSI/EU regulation.

It is the responsibility of the installer to configure the different parameters of the equipment correctly, whether hardware or software, to comply with the pertinent regulation of each country in which it is going to be used.

Specific limitations for the 2.4GHz band.

- In Belgium, outdoor use is only on channels 11(2462MHz), 12(2467MHz) and 13(2472MHz) only. It can be used without a licence if it is for private use and at a distance less than 300m. Over longer distances or for public use, an I'IBPT licence is required.
- In France the use of channels 10(2457MHz), 11(2462MHz), 12(2467MHz) and 13(2472MHz) is restricted. A licence is required for any use both indoors and outdoors. Contact ARCEP (**http://www.arcep.fr**) for further information.
- In Germany a licence is required for outdoor use.
- In Italy a licence is required for indoor use. Outdoor use is not permitted.
- In Holland a licence is required to outdoor use.
- In Norway, use near Ny-Alesund in Svalbard is prohibited. For further information enter Norway Posts and Telecommunications (**http://www.npt.no**).

Specific limitations for the 868MHz band.

- In Italy the maximum broadcast power is 14dBm.
- In the Slovakian Republic the maximum broadcast power is 10dBm.

IMPORTANT

It is the responsibility of the installer to find out about restrictions of use for frequency bands in each country and act in accordance with the given regulations. Libelium Comunicaciones Distribuidas S.L does not list the entire set of standards that must be met for each country. For further information go to:

CEPT ERC 70-03E - Technical Requirements, European restrictions and general requirements: **http://www.ero.dk**

R&TTE Directive - Equipment requirements, placement on market: **http://www.ero.dk**

26. Maintenance

- In this section, the term "Waspmote" encompasses both the Waspmote device itself as well as its modules and sensor boards.
- Take care when handling Waspmote, do not let it fall, knock it or move it suddenly.
- Avoid having the devices in high temperature areas as it could damage the electronic components.
- The antennas should be connected carefully. Do not force them when fitting them as the connectors could be damaged.
- Do not use any type of paint on the device, it could harm the operation of the connections and closing mechanisms.

27. Disposal and recycling

- In this section, the term "Waspmote" encompasses both the Waspmote device itself as well as its modules and sensor boards.
- When Waspmote reaches the end of its useful life, it must be taken to an electronic equipment recycling point.
- The equipment must be disposed of in a selective waste collection system, and not that for urban solid residue. Please manage its disposal properly.
- Your distributor will inform you about the most appropriate and environmentally friendly disposal process for the used product and its packaging.