Waspmote

Technical Guide

libelium

waspmote

Document version: v3.4 - 11/2012

INDEX

1. Waspmote Kit 6
1.1. Box content 6
1.2. General and safety information 6
1.3. Conditions of use 7
1.4. Assembly 9

2. Waspmote Plug & Sense! - Encapsulated Line 15
2.1. Quick Overview 15
2.1.1. Features 15
2.1.2. Sensor Probes 15
2.1.3. Solar Powered 16
2.1.4. Programming the Nodes 17
2.1.5. Radio Interfaces 18
2.1.6. Program in minutes 18
2.1.7. Data to the Cloud 19
2.1.8. Models 19
2.1.8.1. Smart Enviroment 20
2.1.8.2. Smart Security 22
2.1.8.3. Smart Metering 24
2.1.8.4. Smart Cities 26
2.1.8.5. Smart Parking 28
2.1.8.6. Smart Agriculture 29
2.1.8.7. Ambient Control 31
2.1.8.8. Radiation Control 33

3. Hardware 34
3.1. Modular Architecture 34
3.2. Specifications 34
3.3. Block Diagram 35
3.4. Electrical Data 36
3.5. I/O 37
3.5.1. Analog 38
3.5.2. Digital 38
3.5.3. PWM 38
3.5.4. UART 39
3.5.5. I2C 39
3.5.6. SPI 39
3.5.7. USB 39

3.6. Real Time Clock - RTC 39
3.7. LEDs 41
3.8. Jumpers 42

4. Architecture and System 43
4.1. Concepts 43
4.2. Timers 44
4.2.1. Watchdog 44
4.2.2. RTC 44
4.3. Interruptions 45

5. Energy System 47
5.1. Concepts 47
5.2. Sleep mode 48
5.3. Deep Sleep mode 48
5.4. Hibernate mode 49

6. Interruptions 50
6.1. Architecture 50
6.2. Watchdog 53
6.3. RTC 53
6.4. Accelerometer 54
6.5. Sensors 55
6.6. GSM/GPRS 55
6.7. Critical battery 56

7. Sensors 58
7.1. Sensors in Waspmote 58
7.1.1. Temperature 58
7.1.2. Accelerometer 59
7.2. Integration of new sensors 62
7.3. Sensor Boards 63
7.4. Power 66

8. 802.15.4/ZigBee 67
8.1. XBee-802.15.4 67
8.2. XBee - ZigBee 70
8.3. XBee - 868 72
8.4. XBee - 900 74
8.5. XBee-XSC 75
8.6. XBee-DigiMesh 76
8.7. RSSI 77

9. Wifi 79
9.1. Wifi Topologies 79
9.1.1. Access Point 79
9.1.2. Ad-hoc mode with iPhone/Android 82
9.1.3. Ad-hoc mode between Waspmotes 82
9.2. Connecting to a Smartphone directly 83
9.2.1. Connecting to an iPhone 83
9.2.1.1. Installation 83
9.2.1.2. iPhone App tutorial 85
9.2.2. Connecting to an Android 88
9.2.2.1. Installation 88
9.2.2.2. Android App tutorial 89

10. Bluetooth 92
10.1. Technical specifications 92
10.2. Scanning for devices 93
10.3. Discovering a device 94
10.4. Security 94
10.5. Services 94
10.6. Adaptive Frequency Hopping (AFH) 95
10.7. Sharing data between ZigBee and Bluetooth Wireless Sensor Networks 96

11. GSM/GPRS 97

12. Bluetooth module for device discovery 99

13. RFID/NFC 101

14. Expansion Radio Board 103

15. Over the Air Programming (OTA) 104
15.1. Overview 104
15.2. Benefits 104
15.3. Concepts 104
15.4. OTA Step by Step 106
15.5. OTA Shell 109

16. GPS 110

17. SD Memory Card 113

18. Energy Consumption 115
18.1. Consumption tables 115

19. Power supplies 117
19.1. Battery 117
19.2. Solar Panel 119
19.3. USB 121

20. Working environment 123
20.1. Linux 123
20.2. Windows 124
20.3. Mac-OS 124
20.4. First steps 125
20.5. Compilation 126
20.6. API 127
20.7. Updating the libraries 130

21. Interacting with Waspmote 131
21.1. Receiving 802.15.4/ZigBee frames with Waspmote Gateway 131
21.1.1. Waspmote Gateway 131
21.1.2. Linux receiver 132
21.1.3. Windows receiver 136
21.1.4. Mac-OS receiver 138
21.2. Meshlium 139
21.2.1. What can I do with Meshlium? 139
21.2.2. How do they work together? 140
21.2.2.1. Meshlium Storage Options 140
21.2.2.2. Meshlium Connection Options 140
21.2.3. Capturing and storing sensor data in Meshlium from a Waspmote sensor network 141
21.2.4. Capturing and storing your own ZigBee frames 154
21.2.5. Sending ZigBee frames from Meshlium to Waspmote 155

22. Certifications 156
22.1. CE 156
22.2. FCC 157
22.3. IC 158
22.4. Use of equipment characteristics 158
22.5. Limitations of use 159

23. Maintenance 160

24. Disposal and recycling 161

1. Waspmote Kit

1.1. Box content

- 5 x Waspmote
- 1 x Waspmote Gateway
- 5 x Lithium Batteries
- 5 x Auxiliary Batteries (button battery)
- 6 x USB cable
- 6 x XBee Radio
- 6 x XBee Antennas (2dBi / 5dBi)
- 0-5 x GPS + 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 **Support 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.

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

http://www.libelium.com/legal

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/legal**. As specified in the Warranty document (**http://www.libelium.com/documentation/condiciones-generales_eng.pdf**), 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.
- 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.
- Jumpers: Handle with care, avoid pushing at right-angles on the pins.
- 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.
- Button battery connection: Connect the battery (CR1200 or 1225) into the available socket respecting the polarity (positive on the upper part, negative pole on the lower part).
- 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.
- 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 - 3G/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 1150, 2300 and 6600 mA·h), 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 = [-20, 60] ºC
  - charge range = [0, 45] ºC

Enclosures:

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. Among other measures:

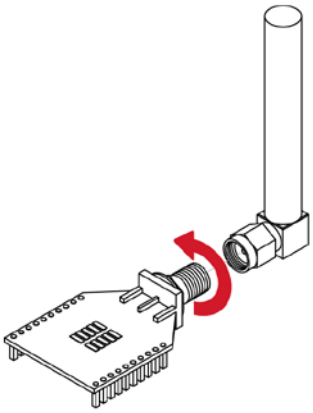
- Before any operation (like disconnecting the battery or connecting the USB) follow these steps:
  - loosen the screws of the enclosure and open it
  - gently disconnect all the cables coming from the sensors (avoid damaging the connectors)
  - unplug the Sensor Board (avoid damaging the solar panel's cable)
  - disconnect the solar panel's cable with care (do not pull the cable)
  - unscrew the XBee's pigtail
  - gently unplug the battery (do not pull the cable, see section "Waspmote battery disconnection", inside chapter 1.4)
  - Waspmote can be taken out from the enclosure now
- The process to install Waspmote back in the enclosure is the same but in the opposite order.
- In any case, cables and connectors must not be under physical stress.
- The probes are designed to protect the sensors in normal conditions of dirt or rain. Make sure the sensor cannot receive liquids, dust or foreign objects from the bottom of the probe: the hole of each hood must be facing downwards so that gravity prevents their entrance.
- Ensure the sensors remain on the upper part of the protective hood.
- The upper part of each probe must be placed inside the enclosure in order to prevent the water penetration into the probe.

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

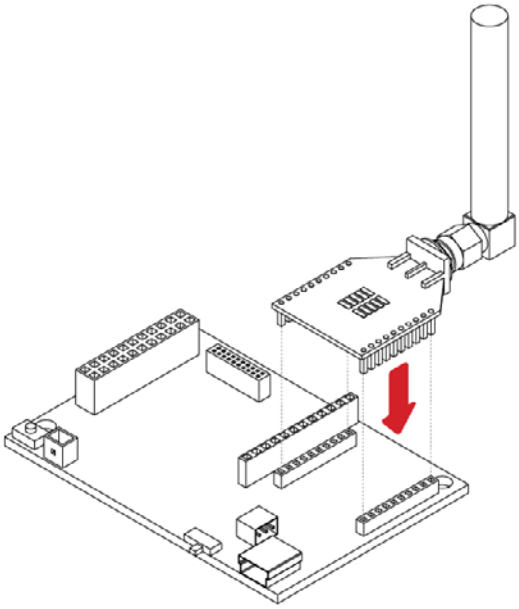
For further information, please visit **http://www.libelium.com/waspmote**

1.4. Assembly

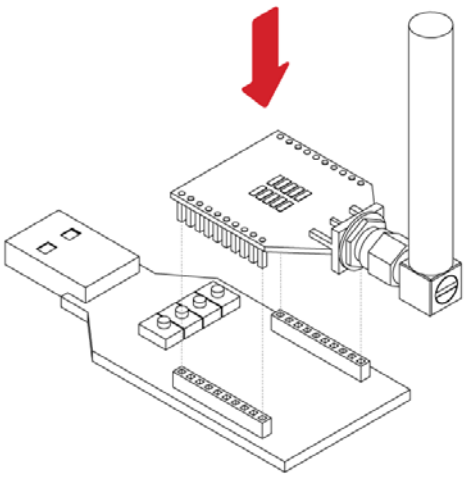
- **Connect the antenna to the XBee module**



- **Place the XBee module in Waspmote**

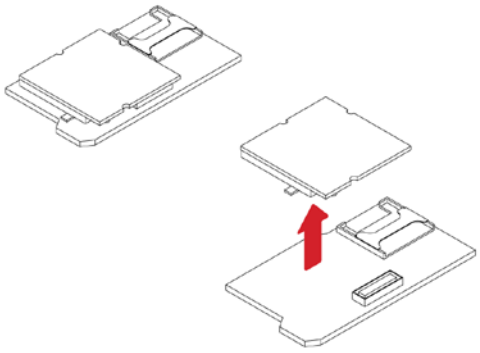


- **Place the XBee module in Waspmote Gateway**

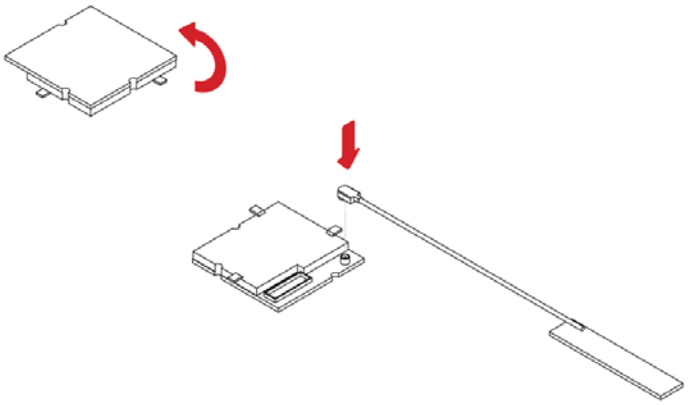


- **Connect the antenna in the 3G/GPRS module**

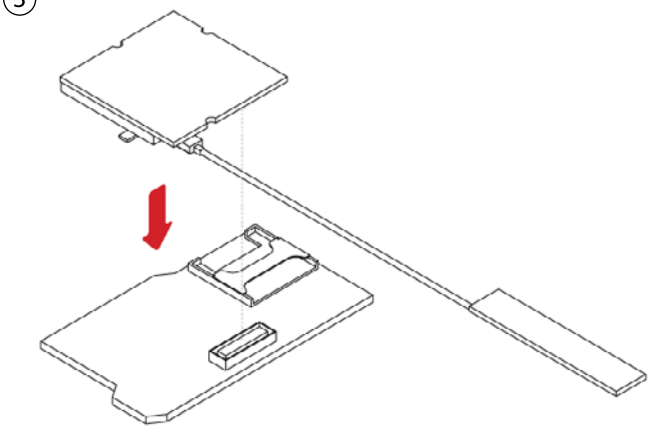
①



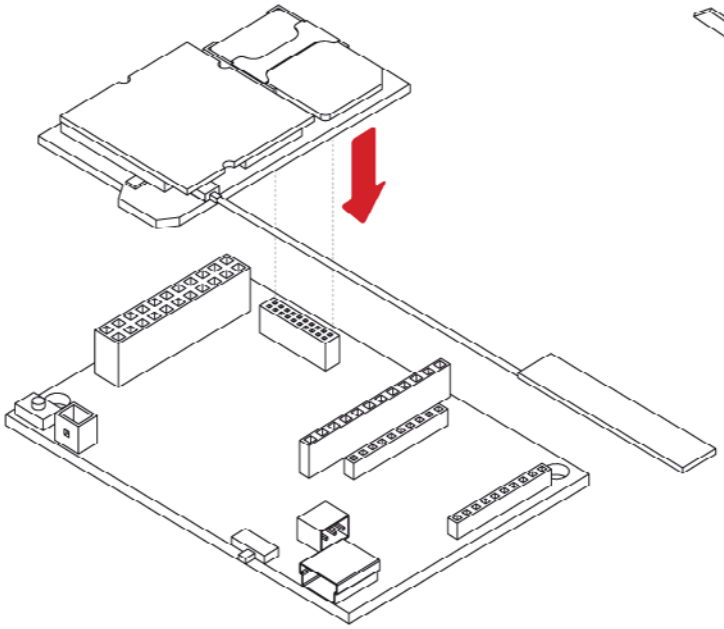
②

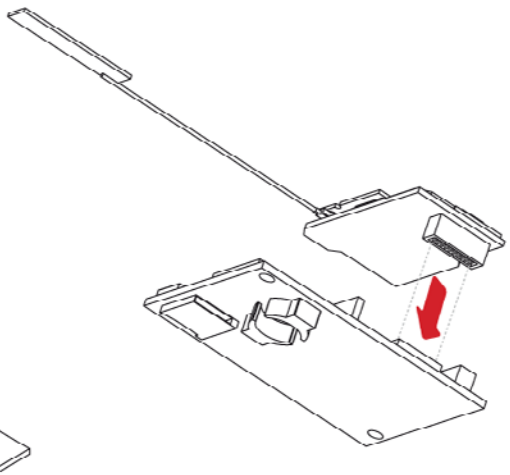


③

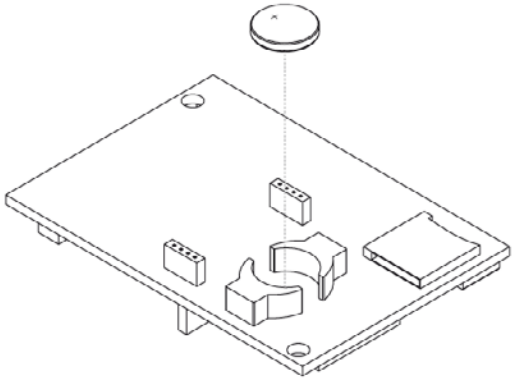


- **Place the 3G/GPRS module in Waspmote**

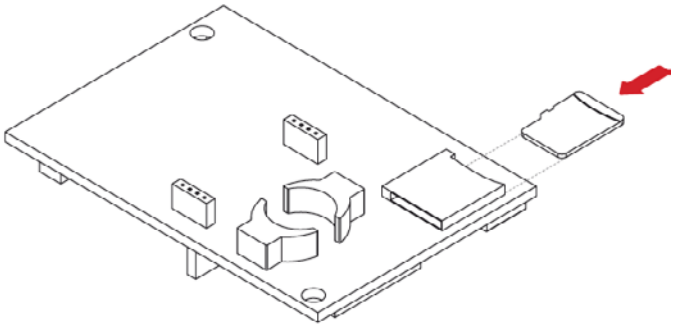




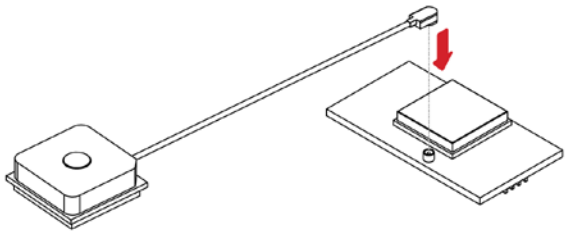
- **Place the button battery in Waspmote**



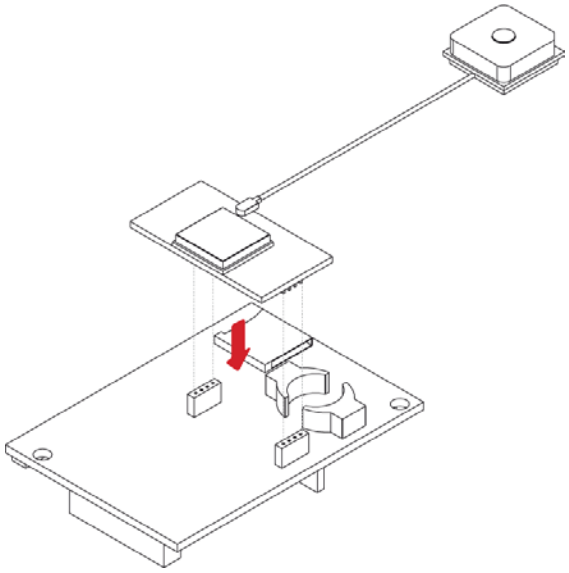
- **Place the SD card in Waspmote**



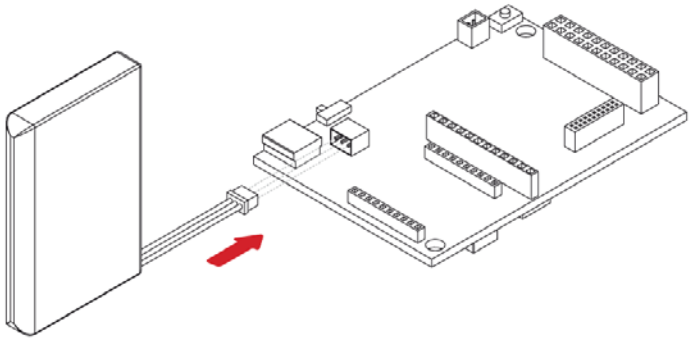
- **Connect the antenna in the GPS module**



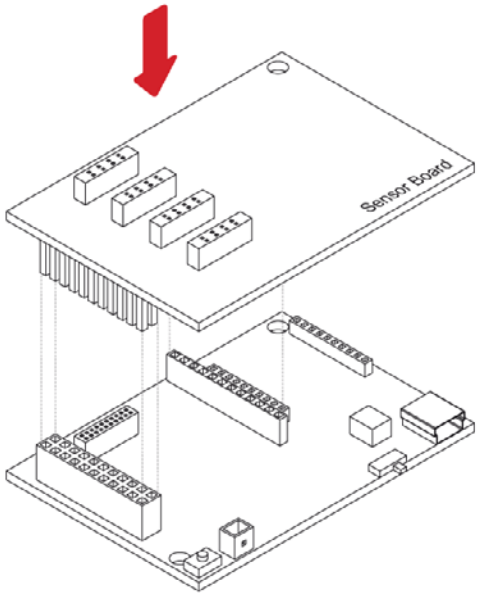
- **Place the GPS module in Waspmote**



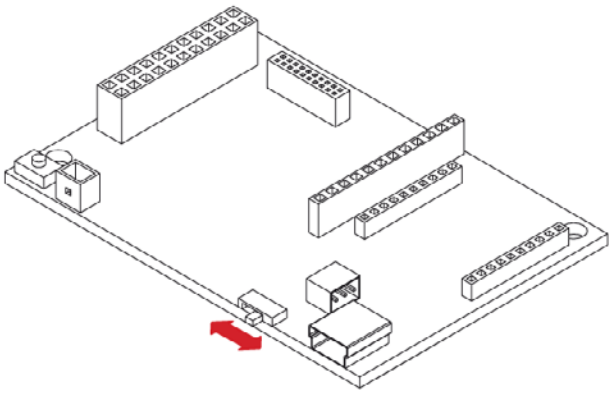
- **Connect the main battery in Waspmote**



- **Connect the sensor board**

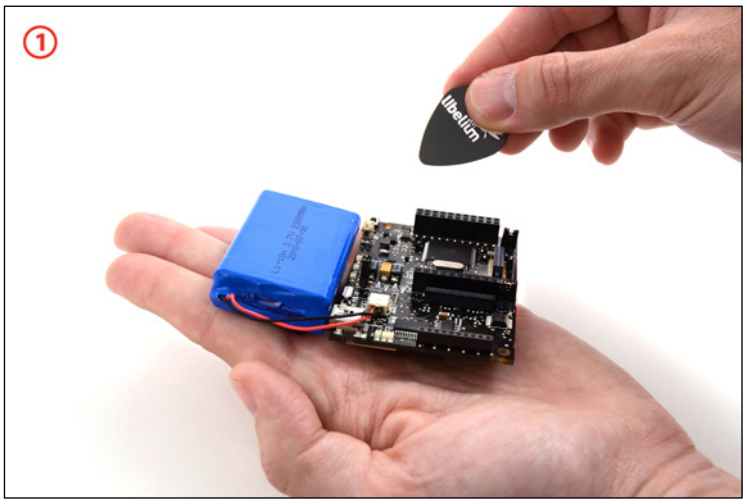


- **Switch it on**

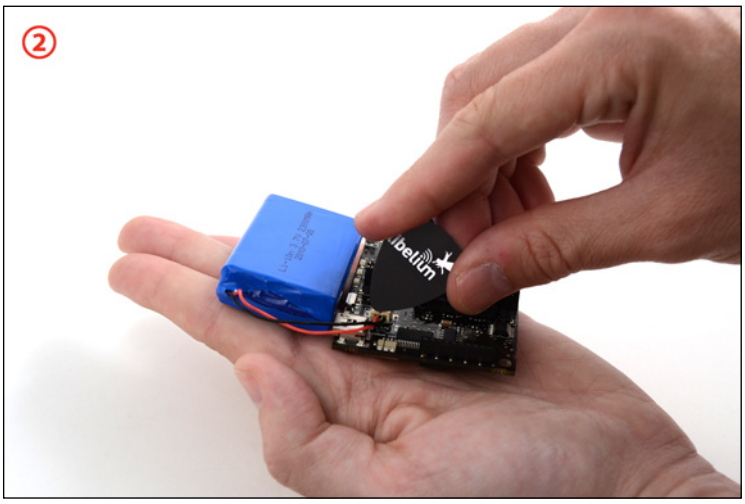


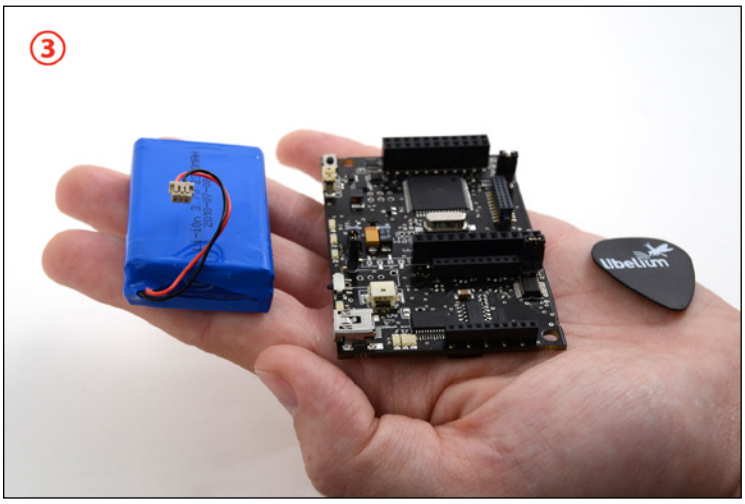
- **Waspmote battery disconnection**

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

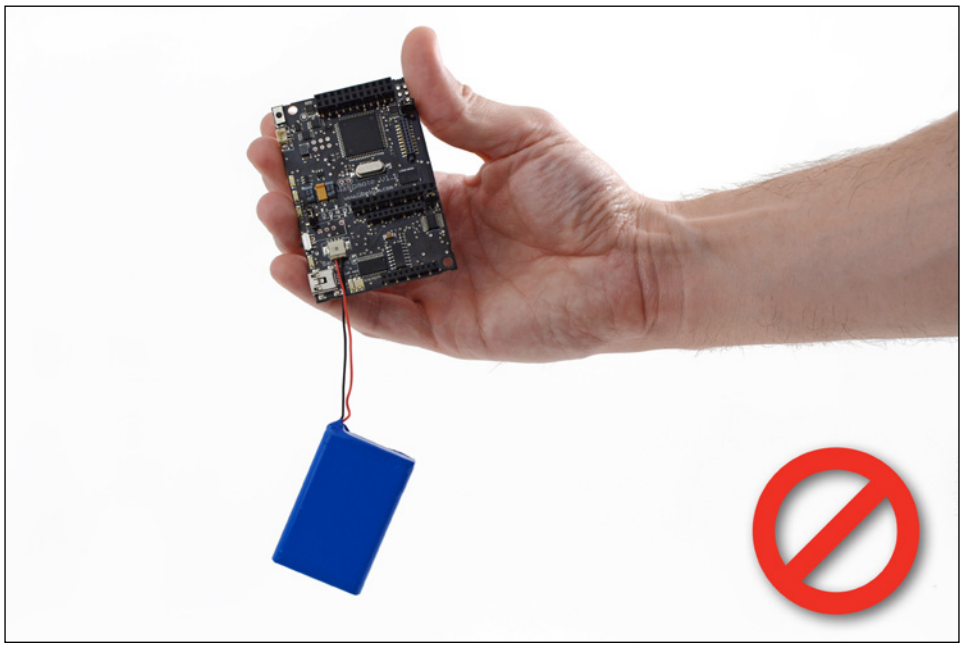


Insert the pick on the slot of the battery connector and pull straight out.
Do not pull the battery cables.

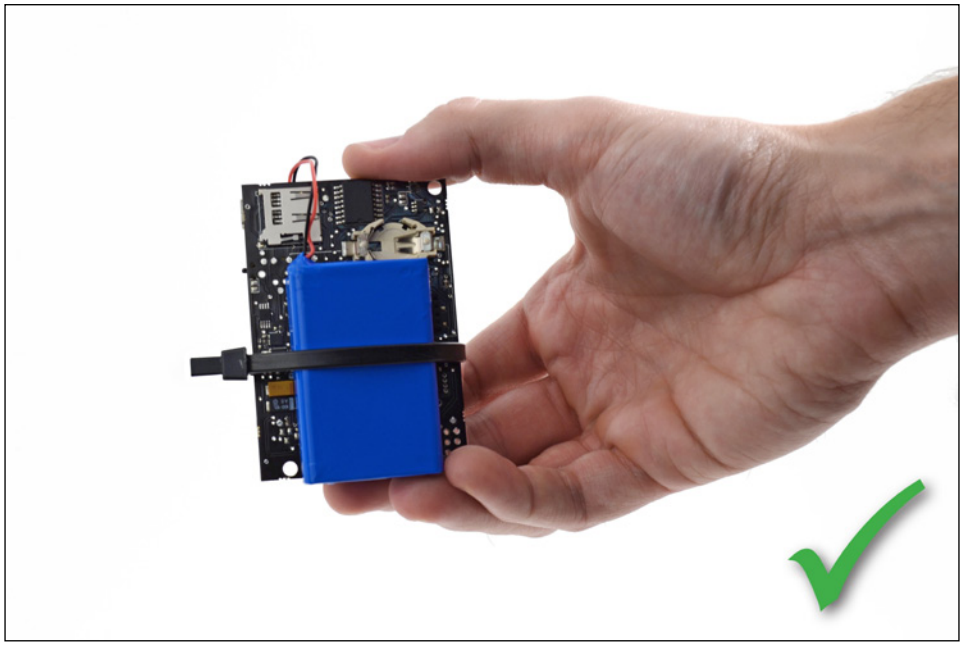




- **Battery handling instructions**
  In order to prevent from cable breaking, avoid leaving battery freely suspended.



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



2. Waspmote 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 **Support section** of the **Libelium website**.

2.1. Quick Overview

2.1.1. 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

2.1.2. 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!

2.1.3. 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

2.1.4. 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.

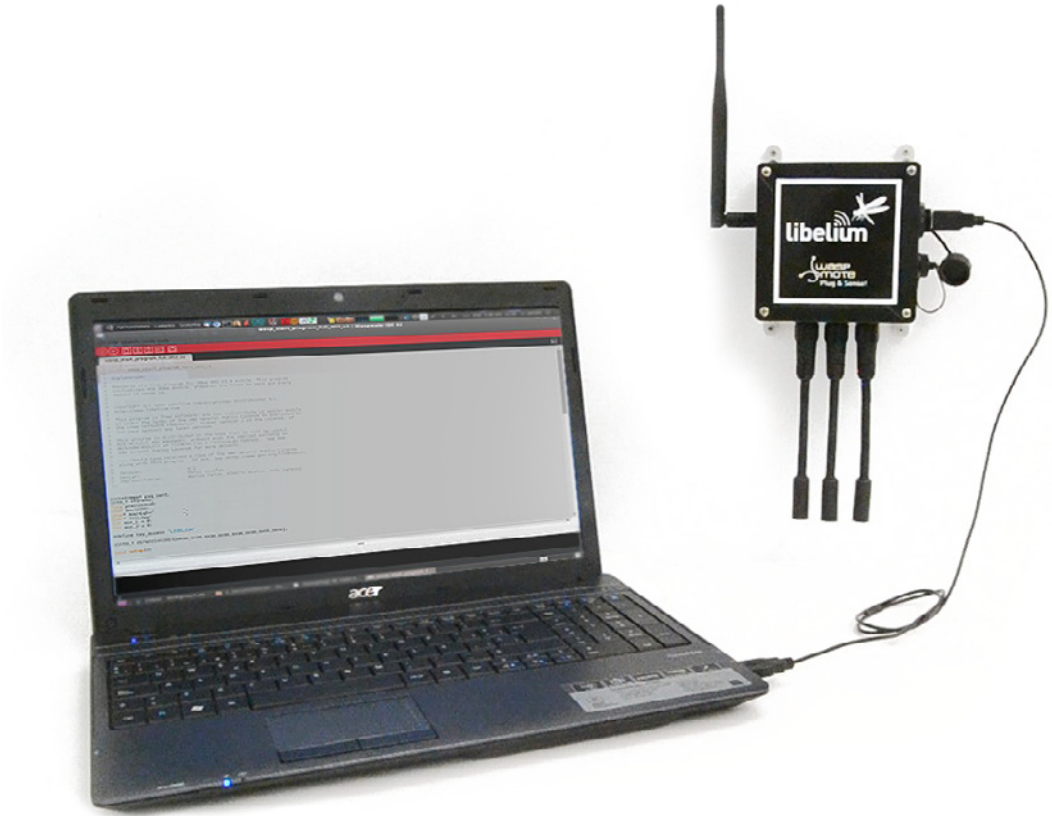


Figure 5: Programming a node

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

2.1.5. 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
XBee-XSC	RF	900MHz	100mW	-106dBm	12Km

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

2.1.6. 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/waspmote/code_generator

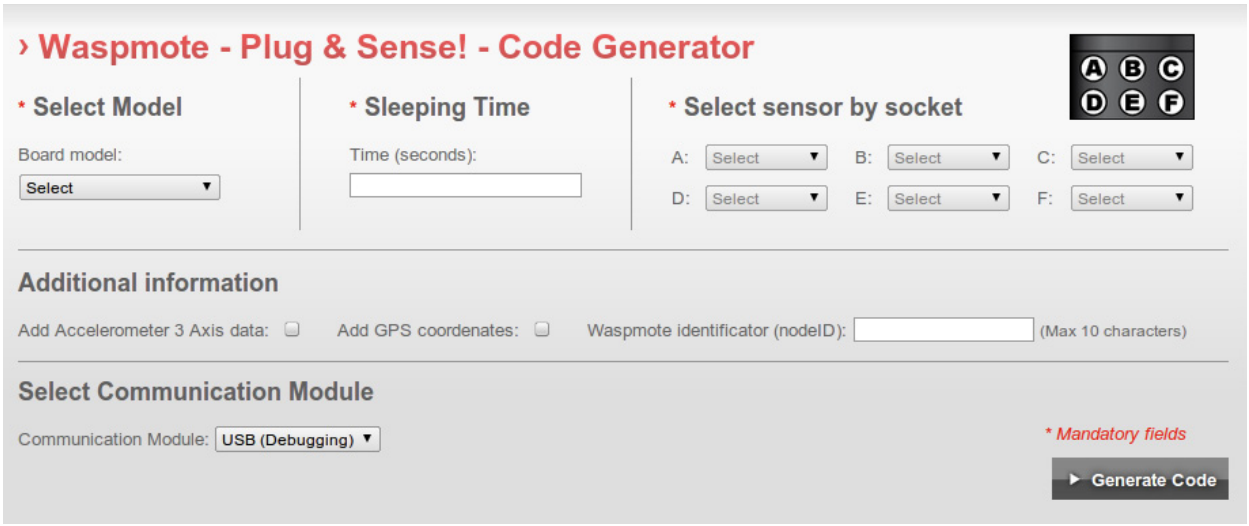


Figure 7: Code Generator

2.1.7. 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

2.1.8. 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.

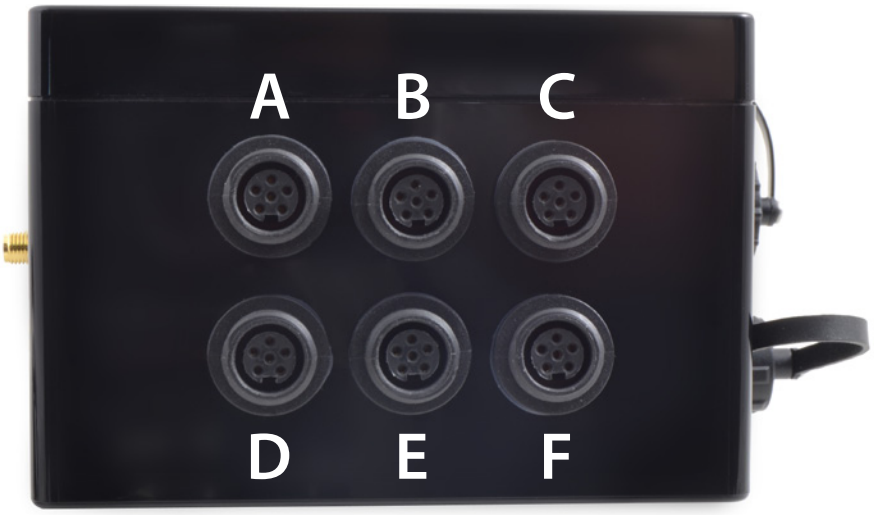


Figure 9: Identification of sensor sockets

2.1.8.1. Smart Enviroment

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 application 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_4H_{10}, CH_3CH_2OH, H_2, CO, CH_4	9235
	Air pollutants 2: $C_6H_5CH_3$, H_2S, CH_3CH_2OH, NH_3, H_2	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_4H_{10}, CH_3CH_2OH, H_2, CO, CH_4	9235
	Air pollutants 2: $C_6H_5CH_3$, H_2S, CH_3CH_2OH, NH_3, H_2	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 **Support section** in Libelium website.

2.1.8.2. 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	9205
	Liquid level	9239, 9240, 9242
	Liquid presence	9243
	Hall effect	9207
E	Luminosity	9205
	Liquid level	9239, 9240, 9242
	Liquid presence	9243
	Hall effect	9207
F	Luminosity	9205
	Liquid level	9239, 9240, 9242
	Liquid presence	9243
	Hall effect	9207

Figure 13: Sensor sockets configuration for Smart Security model

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

2.1.8.3. 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 application section in the **Libelium website** for a complete list of services.

Figure 14: 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	9205

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

Figure 15: Sensor sockets configuration for Smart Metering model

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

2.1.8.4. 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 16: 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	9205
D	Noise sensor	9259
E	Dust sensor	9320
F	Linear displacement	9319

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

Figure 17: Sensor sockets configuration for Smart Cities model

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

2.1.8.5. 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 18: 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.

2.1.8.6. 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 19: 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 20: Sensor sockets configuration for Smart Agriculture model

Note: For more technical information about each sensor probe go to the **Support 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 21: Sensor sockets configuration for Smart Agriculture PRO model

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

2.1.8.7. 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 22: 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	9205
C	Not used	
D	Not used	
E	Not used	
F	Not used	

Figure 23: Sensor sockets configuration for Ambient Control model

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

2.1.8.8. 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 24: 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 **Support section** in Libelium website.

3. Hardware

3.1. 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 categorised in:

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

3.2. Specifications

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

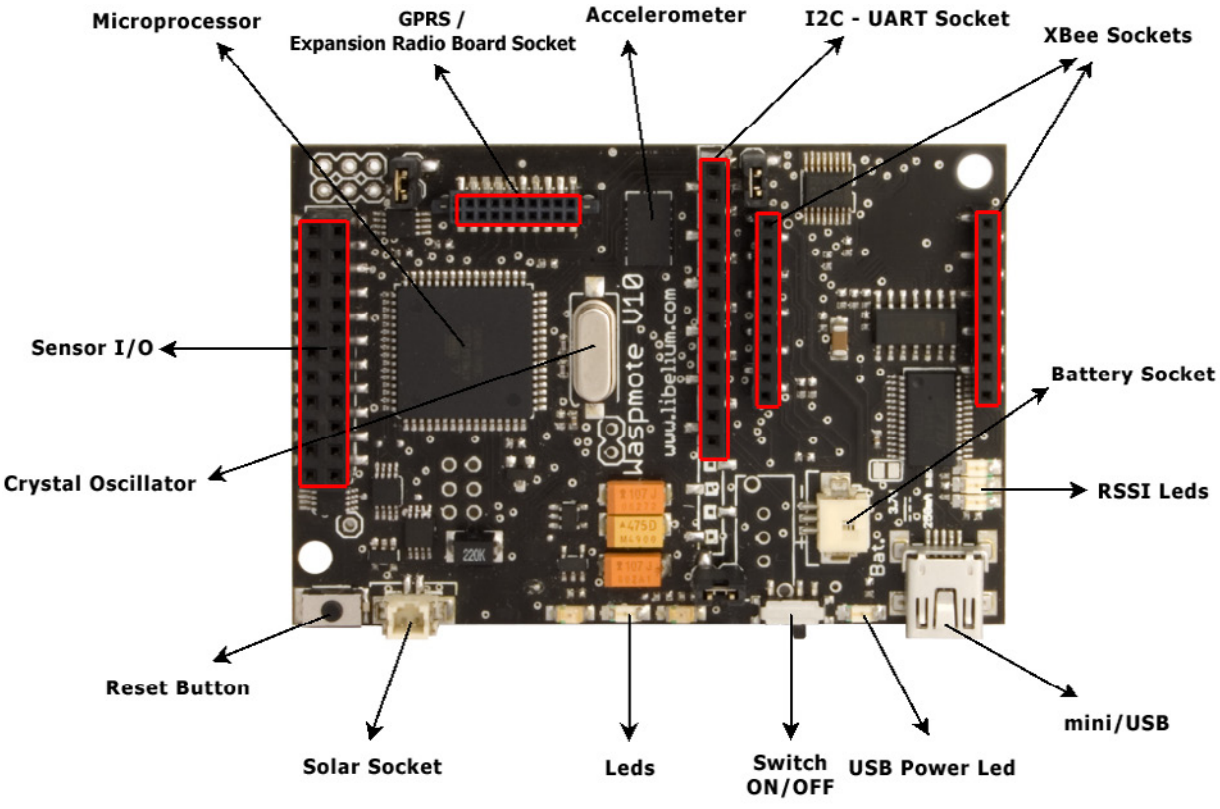


Figure 25: Main Waspmote components – Top side

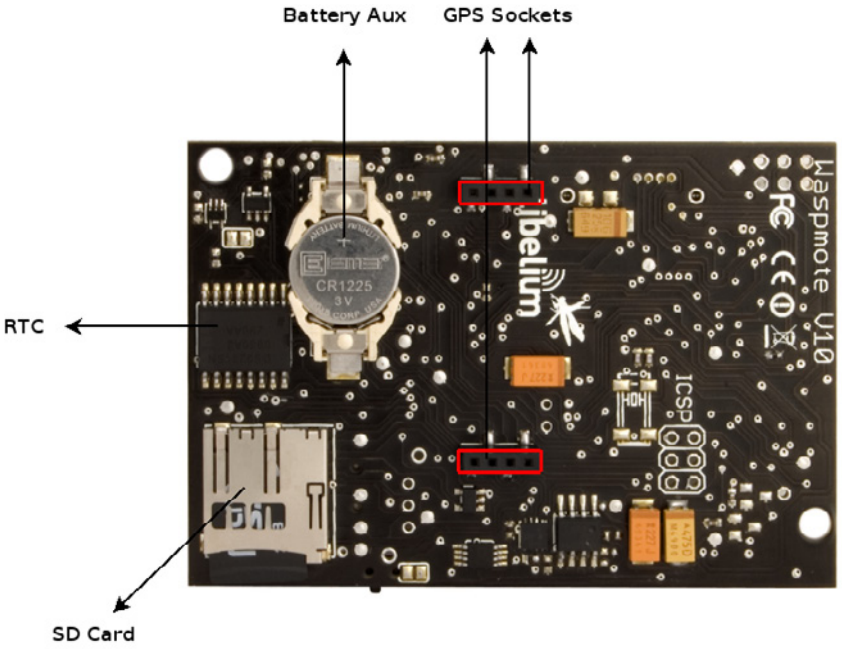


Figure 26: Main Waspmote components – Bottom side

3.3. Block Diagram

Data signals:

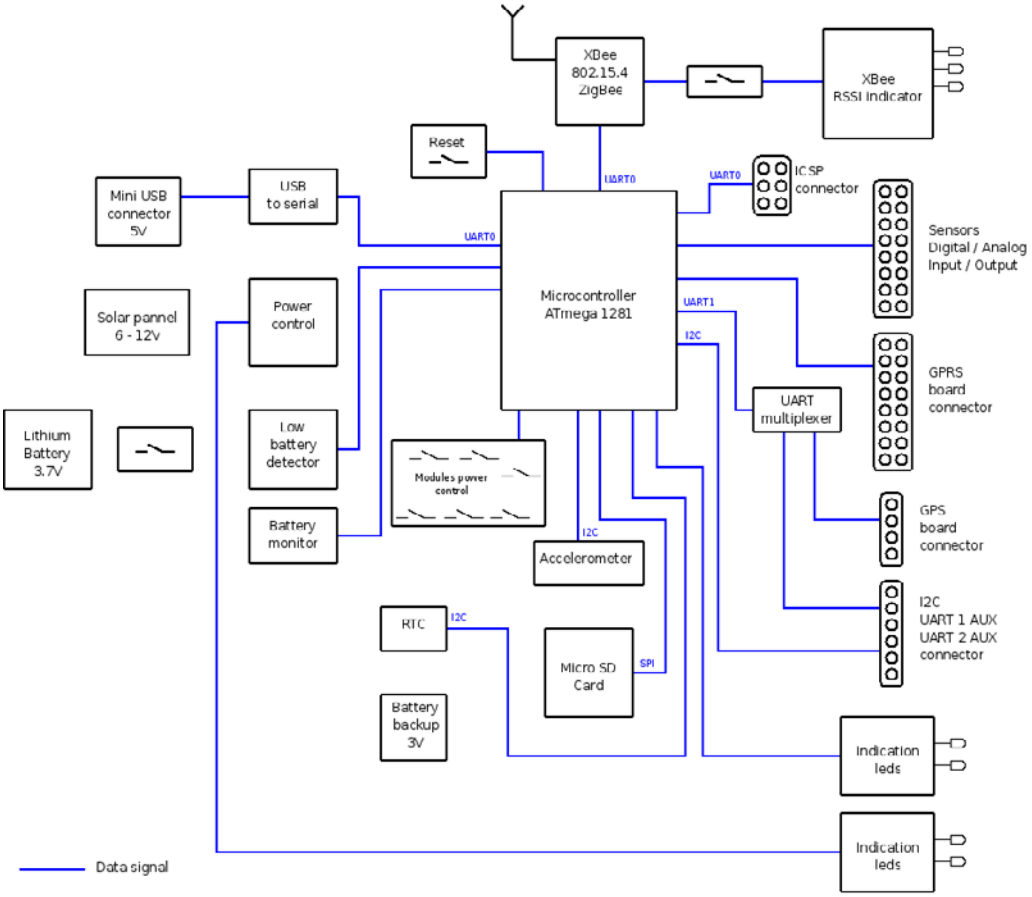


Figure 27: Waspmote block diagrams – Data signals

Power signals:

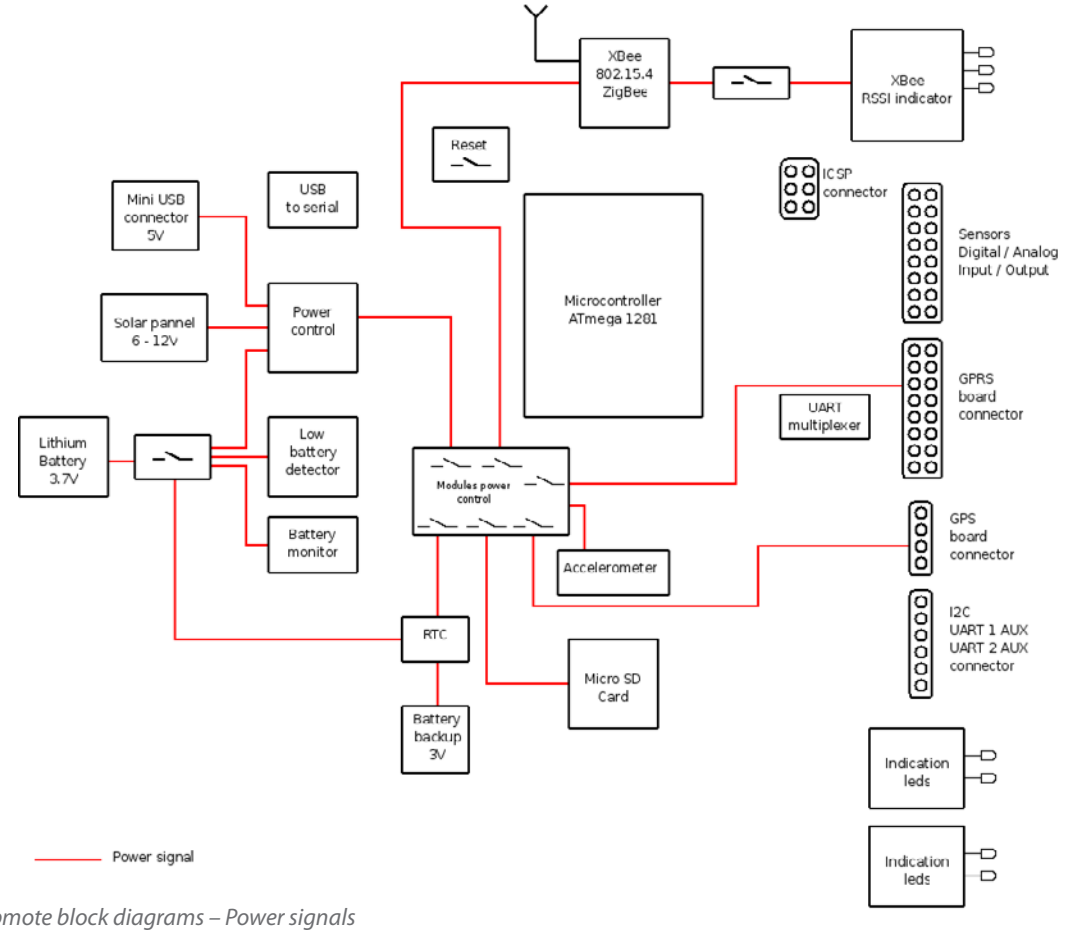


Figure 28: Waspmote block diagrams – Power signals

3.4. 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)
- Button battery voltage 3V

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

3.5. I/O

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

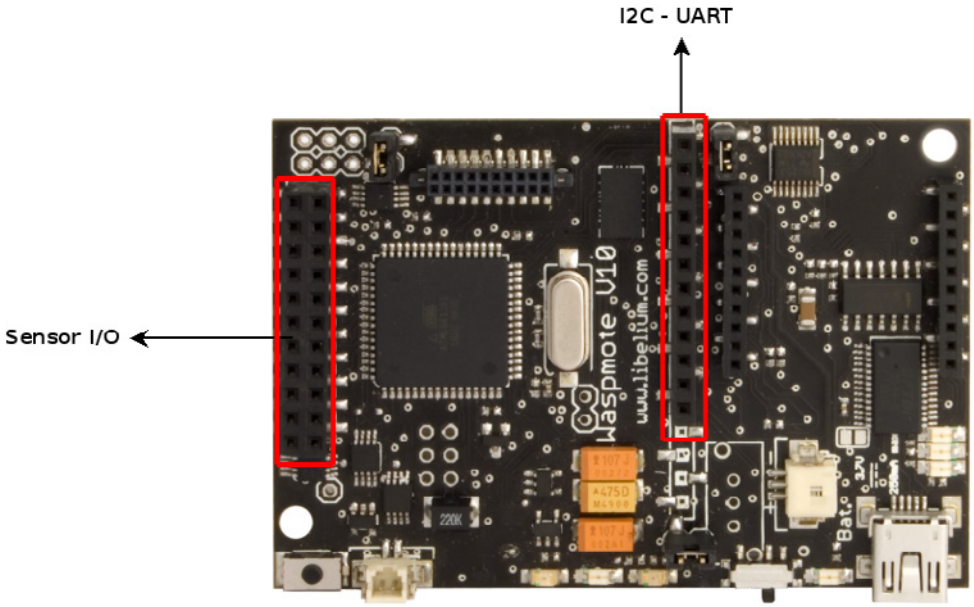


Figure 29: I/O connectors in Waspmote

Sensor connector:

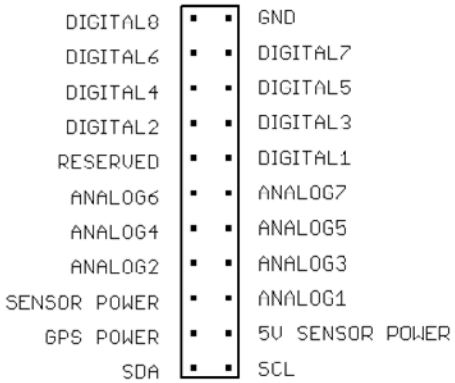


Figure 30: Description of sensor connector pins

Auxiliary I2C-UART connector

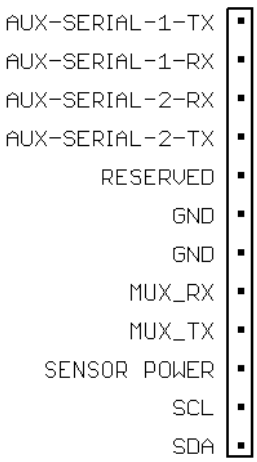


Figure 31: Description of auxiliary I2C-UART connector pins

3.5.1. 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 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);
}
```

3.5.2. 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:

```
{
  pinMode(DIGITAL3,INPUT);
  val=digitalRead(DIGITAL3);
  pinMode(DIGITAL3,OUTPUT);
  digitalWrite(DIGITAL3,LOW);
}
```

3.5.3. 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);
}
```

3.5.4. UART

There are 6 serial ports in Waspmote:

One of the microcontroller's UARTs is simultaneously connected to the XBee communication module and the USB port. The other microcontroller UART is connected to a four channel multiplexer, and it is possible to select in the same program which of the four new UARTs is required to connect to the UART on the microcontroller. These four new UARTs are connected as follows. One is connected to the 3G/GPRS board, the other to the GPS and the other two are accessible to the user in the *auxiliary I2C – UART connector.*

Multiplexer configuration is carried out using the following instructions:

```
{
  setMuxAux1();
  setMuxAux2();
  setMuxGPS();
  setMuxGPRS();
}
```

3.5.5. I2C

The I2C communication bus is also used in Waspmote where three devices are connected in parallel: the accelerometer, the RTC and the digital potentiometer (digipot) which configures the low battery alarm threshold level. In all cases, the microcontroller acts as master while the other devices connected to the bus are slaves.

3.5.6. 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.

3.5.7. USB

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

For USB communication, one of the microcontroller's UARTs is used. The **FT232RL** carries out the conversion to USB standard.

3.6. 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).

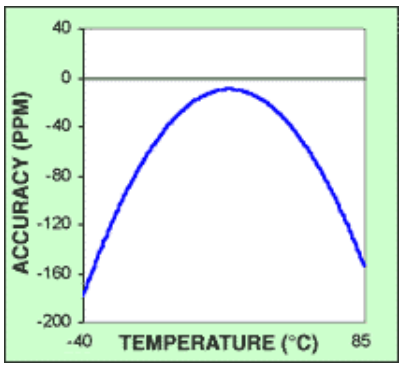


Figure 32: Uncompensated variation curve

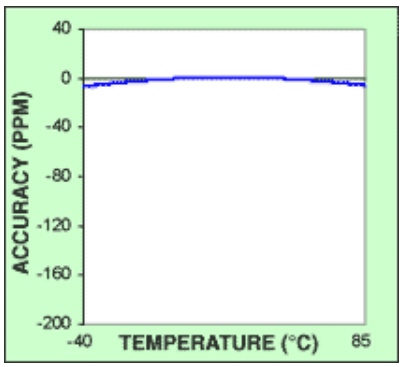


Figure 33: Compensated variation curve

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 section 7.1.1.

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 has 2 power supplies: main battery and **auxiliary battery**. When the mote is connected, the RTC is powered through the main battery. However, to ensure the correct time is always maintained and the information is not deleted when the main battery is changed or is out of load, an auxiliary battery has been included which powers the RTC when the main battery cannot.

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 disconnect its main battery.The RTC (powered from its auxiliary battery) controls when it has to wake Waspmote up and perform a particular action. This allows a consumption of **0.7µA** to be obtained in the Hibernate mode. See sections 5.2 and 5.3.

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 **Support** and **Development sections** in the Libelium website.

3.7. LEDs

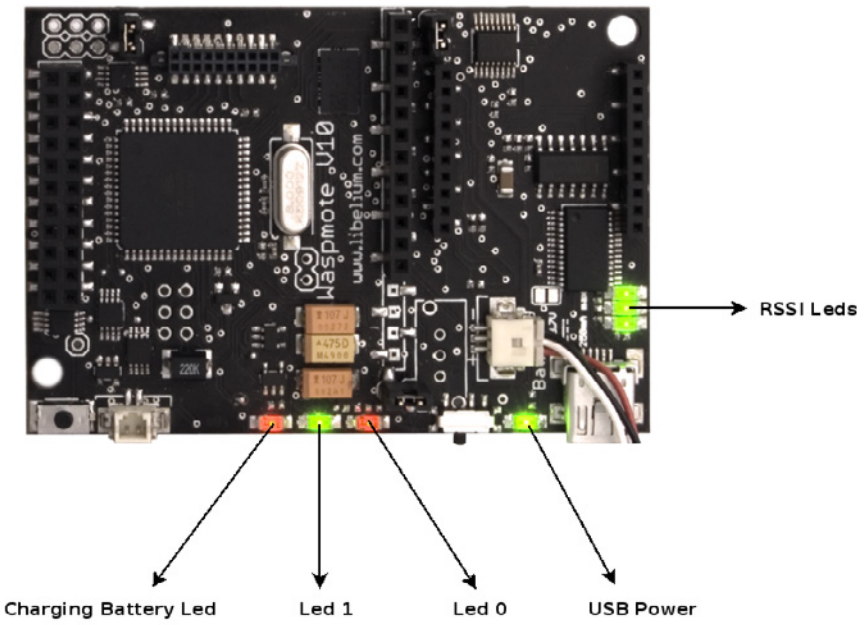


Figure 34: Visual indicator LEDs

- **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.

- **RSSI LEDs**
  3 LEDs have been included to show the **RSSI** (Received Strength Signal Indicator) value of the ZigBee/802.15.4 frames. These LEDs indicate the signal quality of the last packet received. For more information see chapter 7.7.

Programming

Two LEDs are programmable from the program code. The functions for handling these LEDs are *"Utils.setLED() Utils.getLED() 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);
}
```

3.8. Jumpers

There are three jumpers in Waspmote to activate and deactivate certain functions, in the figure below the location of these jumpers is showed.

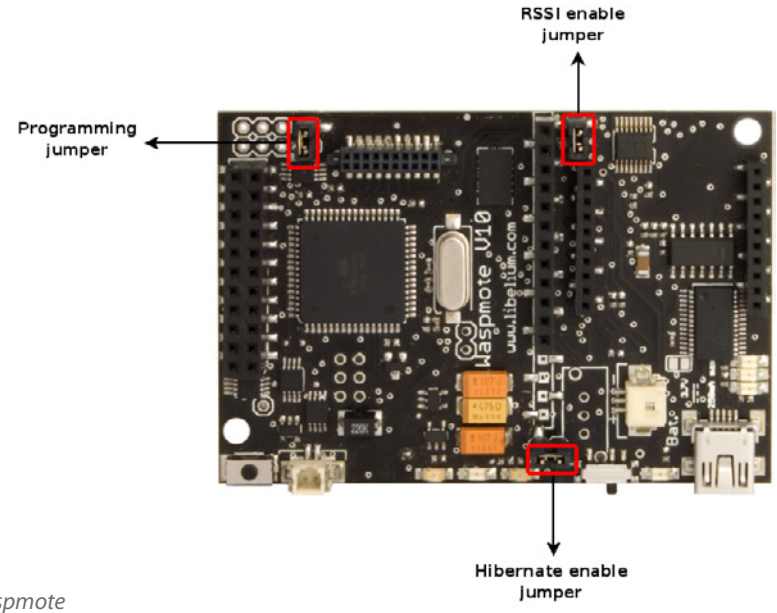


Figure 35: Jumpers in Waspmote

Description of the jumpers:

- **Programming enabling jumper**
  Board programming is only possible if this jumper is set. Once the laboratory phase is completed and before the definitive installation, it is advised not to set the programming jumper in order to reduce the power consumption of Waspmote to the minimum.*

- **RSSI indicator enabling jumper**
  If the jumper is set, the indicating function for the RSSI LEDs is enabled. If it is not being in used, the jumper should be taken out to avoid extra consumption.

- **Hibernate mode enabling jumper**
  If the jumper is not set, programming Waspmote is not possible. Hibernate operation mode needs the jumper to be removed during Waspmote set up. See how to use it in "section 5.4".

  ***Note:** with the last version, the programming jumper is not necessary to enable the upload of code. That means that the code can be written in Waspmote regardless of if the jumper is set or not.

4. Architecture and System

4.1. 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: a part named **setup** and one called **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 recommendable to include 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 behaviour 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 chapter 5).

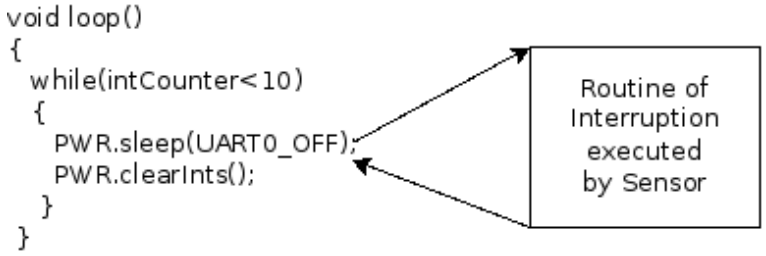


Figure 36: Blocking loop, interruption appears and is dealt with

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 291 are used by Waspmote to save important data, so they must not be over-written. Thus, the available storage addresses go from 292 to 4095. Another option is the use of the high capacity **2GB SD card.**

4.2. Timers

Waspmote uses a quartz oscillator which works at a frequency of **8MHz** 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, 3G/GPRS, USB) a maximum transmission speed of 38400bps has been set for XBee, GRPS and USB, and 4800 for the GPS, so that the success rate in received bits is 100%.

4.2.1. 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 wakes up from a low consumption Sleep mode by generating an interrupt. 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, see chapter 3.2.2

More information about the interruptions generated by the Watchdog can be found in chapter 5.2.

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 **Support** and **Development sections** in the Libelium website.

4.2.2. RTC

As shown in chapter 2.6, Waspmote has a real time clock (RTC) running a 32KHz (32.786Hz) which allows an absolute time base to be set.

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

As well as relative alarms, periodic alarms can be programmed by giving a time measurement, so that it reprograms its alarm automatically each time it goes off.

The RTC allows the microcontroller to be woken from the low consumption state by generating an interrupt. 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 up it by activating an alarm in the RTC. Intervals can go from 8s in Deep Sleep mode, to minutes, hours or even days in Hibernate.

More information about the interruptions generated by the RTC and DeepSleep and Hibernate modes can be found in chapter 5.

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 **Support** and **Development sections** in the Libelium website.

4.3. 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 and makes the sensors inform warn Waspmote when a determined value (threshold) is reached.

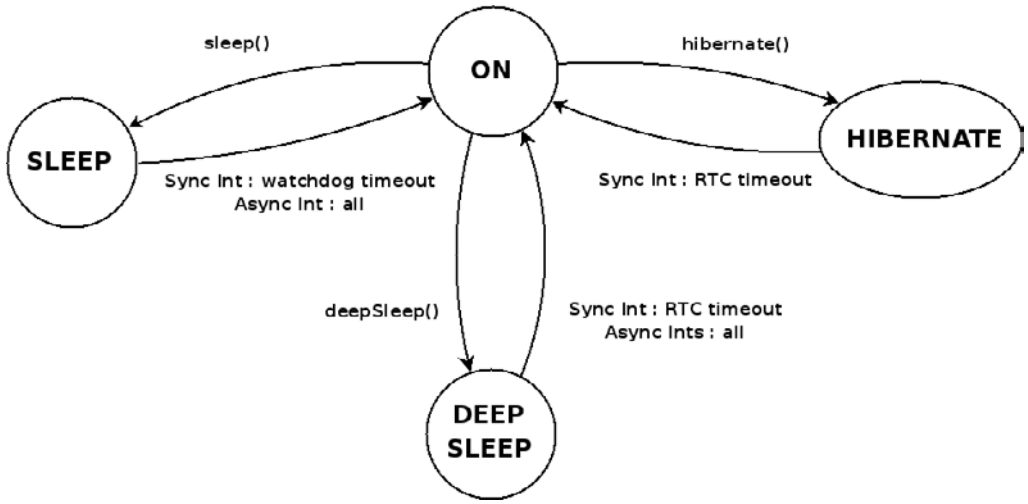


Figure 37: Diagram of mode in Waspmote

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 when a sensor reaches a certain threshold it triggers an alarm.
  - **Low battery**: Waspmote has a circuit which controls the level of battery left at anytime. When a certain threshold is reached an alarm is generated (*"hey! I'm dying!"*) which warns the control centre that one of the motes is running out of battery. This critical battery level will vary depending on the type of modules that Waspmote has, as well as the final application. For this reason, this threshold can be defined dynamically through the programming of a digital potentiometer by the microcontroller.
  - **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 interrupt.
  - **3G/GPRS Module:** receiving a call, an SMS or data trough a TCP or UDP socket generates an interruption.

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 auxiliary 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 in 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.

5. Energy System

5.1. Concepts

Waspmote has 4 operational modes.

- **ON**: Normal operation mode. Consumption in this state is **9mA**.
- **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 **62µA**.
- **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 **8 seconds to minutes, hours, days**. Consumption in this state is **62µA**.
- **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 **8 seconds to minutes, hours, days**. When the device is totally disconnected from the main battery the RTC is powered through an **auxiliary battery** from which it consumes **0.7µA**.

	Consumption	Micro	Cycle	Accepted Interruptions
ON	9mA	ON	-	Synchronous and Asynchronous
Sleep	62µA	ON	32ms - 8s	Synchronous (Watchdog) and Asynchronous
Deep Sleep	62µA	ON	8s – min/hours/days	Synchronous (RTC) and Asynchronous
Hibernate	0.7µA	OFF	8s – min/hours/days	Synchronous (RTC)

On the other hand, each **module** has 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 chapter 5.

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 **Support** and **Development sections** in the Libelium website.

5.2. 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. The duration interval of this state is from **32ms to 8s**. Consumption in this state is **62µA**.

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

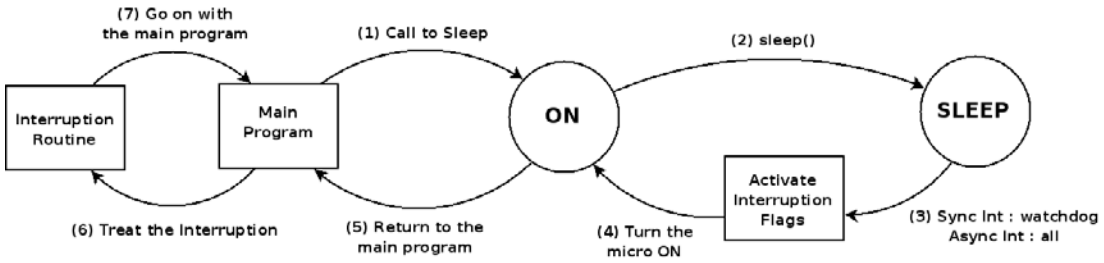


Figure 38: From ON to Sleep

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);
}
```

5.3. 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 be from **8 seconds to minutes, hours, days**. Consumption in this state is **62µA**.

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

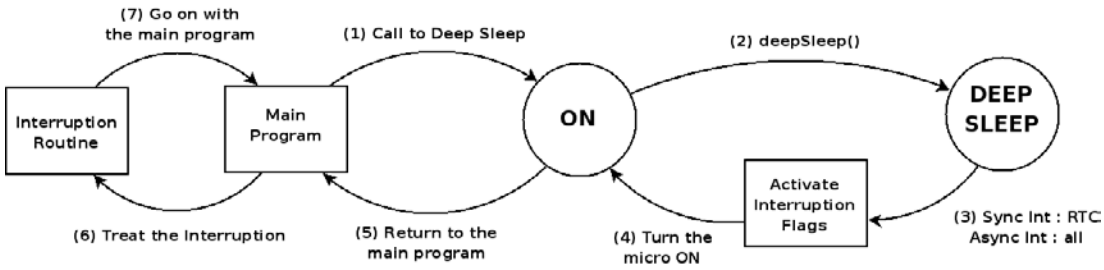


Figure 39: From ON to Deep Sleep

5.4. 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 be from **8 seconds to minutes, hours or days**. When the device is **totally disconnected** from the main battery the RTC is powered through the **auxiliary battery** from which it consumes **0.7µA**.

In this mode the micro 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.

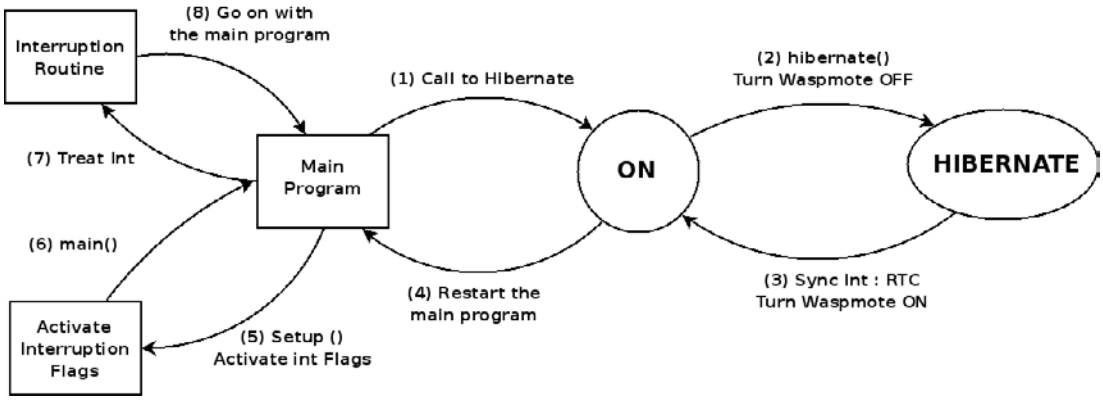


Figure 40: From ON to Hibernate

Hibernate mode requires the hibernate jumper to be removed 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. Connect the button battery
3. Switch Waspmote on.
4. Wait for red led to light on and off
5. Once red led is off, remove hibernate jumper

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

```
{
  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 their programming and operation can be found in the document: **Energy and Power Programming Guide.**

All the documentation is located in the **Support** and **Development sections** in the Libelium website.

6. Interruptions

6.1. Architecture

At the request of the microcontroller, it receives all types of interruption as **hardware interruptions**. For this reason the **UART** (**RXD1 and TXD1)** pins are used.

As stated, Waspmote has 6 possible interruption signals: Watchdog, RTC, accelerometer, 3G/GPRS, sensors and critical battery.

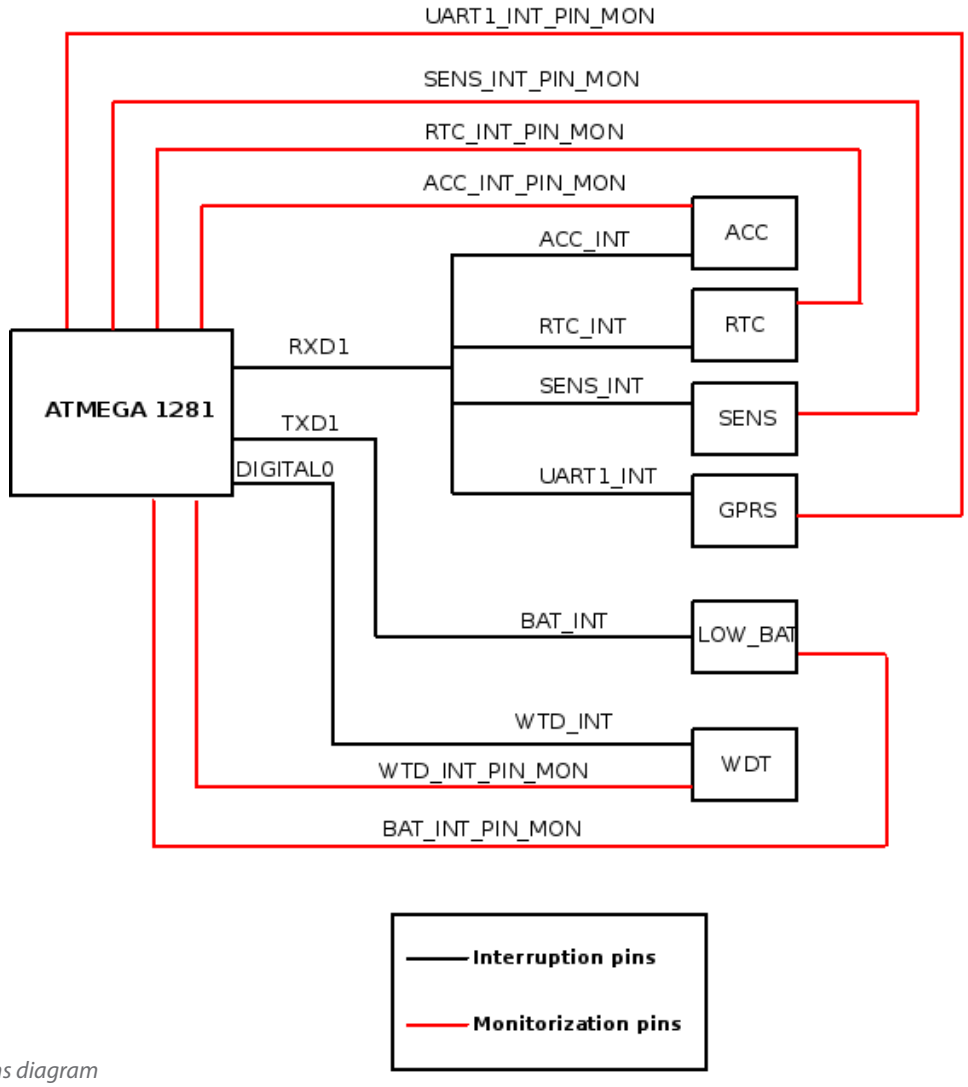


Figure 41: Interruptions diagram

The interruption signals from the RTC, accelerometer, 3G/GPRS and sensors are multiplexed in the **RXD1** pin. The interruption signal from the critical battery is connected on **TXD1** pin, while the internal Watchdog 'simulates' a hardware interruption using the DIGITAL0 pin. This simulation by the Watchdog has been implemented to maintain the same functionality structure in all the interruption sources and can fill in flags in the same way.

The interruption pin in the modules is connected through a resistance to **RXD1** or **TXD1** and a unique **monitoring pin** for each module. In this way when the interruption is captured the monitoring pin of each module is analyzed to see which one generated it.

The definition of the monitoring and interruptions pins can be found in the WaspConstants.h file.

Because of this multiplexing of several interruption signals a series of flag vectors have been created to pull the module which generated the interrupt. These flags are: intConf, intFlag, and intArray.

intConf
This flags vector is used to set which modules are activated to generate interruptions. Only interruptions that are previously activated in this vector will be received.

RES	SENS2	SENS	UART1	UART0	PIN	TIM2	TIM1	TIM0	WTD	RTC	BAT	ACC	LAI	HAI	RES
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 42: Structure of the 'intConf' flag

intFlag
This flag is used to find out which module generated the captured interrupt. This flag marks the position corresponding to the module which generated the interrupt.

HIB	SENS2	SENS	UART1	UART0	PIN	TIM2	TIM1	TIM0	WTD	RTC	BAT	ACC	LAI	HAI	RES
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 43: Structure of the 'intFlag' flag

intCounter
This flag is used to count the total number of interruptions which have been produced. Each time an interruption is generated this counter is increased.

intArray
This flag is used to count the total number of interruptions that have been produced for each module. Each time an interruption is produced the position corresponding to the module which has generated it is increased.

SENS	UART1	UART0	PIN	TIM2	TIM1	TIM0	WTD	RTC	BAT	ACC	LAI	HAI
12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 44: Diagram of the 'intArray' flag

Two subroutines have been associated to the interruption pins that are used **onHAIwakeUP** and **onLAIwakeUP**. These subroutines are run when an interruption is captured, and the flags previously explained are updated. Initially no further action is taken, in an attempt to add the minimum time delay possible in the execution trace of the program's main code. Once these flags are updated we return to the instruction which was being run before capturing the interrupt.

It is in the main code where a function will be then run that will invoke the corresponding interruption routine related to the event whose flag was activated previously. The idea of this design is to avoid pausing actions that Waspmote may be doing at the moment the interruption triggers, an example would be sending a packet through the XBee radio.

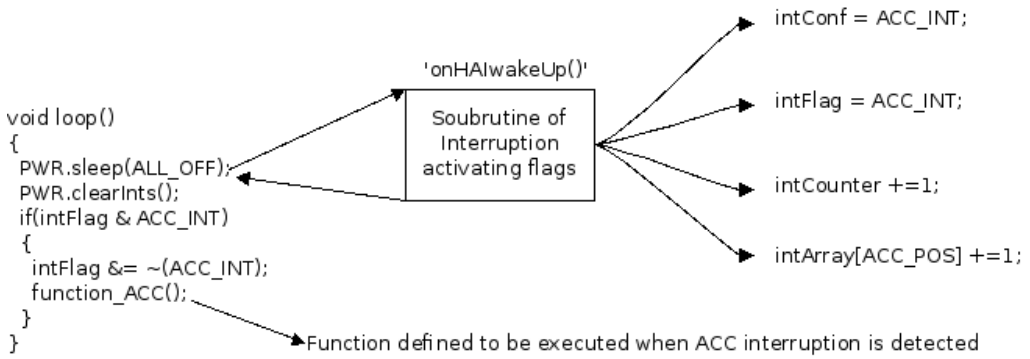


Figure 45: Process of an interrupt

The modules can generate high level, **'1'** logic interruptions, or low level, **'0'** logic interruptions. In this way the subroutine onHAIwakeUP will be run with the modules which generation high level interruptions, while the subroutine onLAIwakeUP will be run with the modules which generate low level interruptions.

These subroutines can be found in Winterruptions.c.

When including a function in the loop's main code that deals with the generated interruptions, it is recommended to use chained comparators which analyse the flags to find out if an interruption has occurred and what module has produced it, and if so, executing the routine of the associated interrupt. To carry out these comparisons, it is recommended to use a treat_ interruptions() function, defined in the main code.

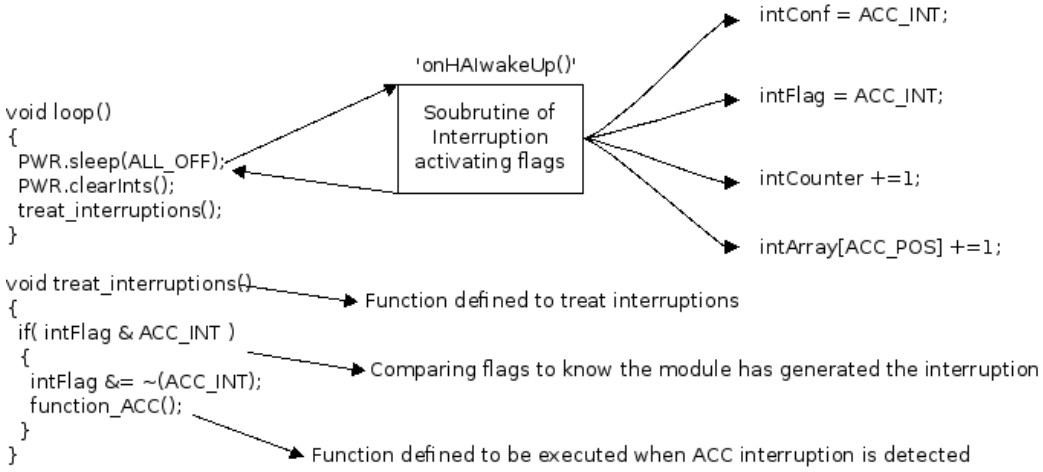


Figure 46: Treatment of an Interruption

The correct use of the flags allows the **chaining of interruptions**; as a result, several interruptions can be triggered before treating them. However, because several interruption signals have been multiplexed, it is necessary to reactivate the interruption reception. For this reason, there is a function called 'PWR.clearInts()' which uses the intFlag flag to clean and reactivate the interruptions which have been captured.

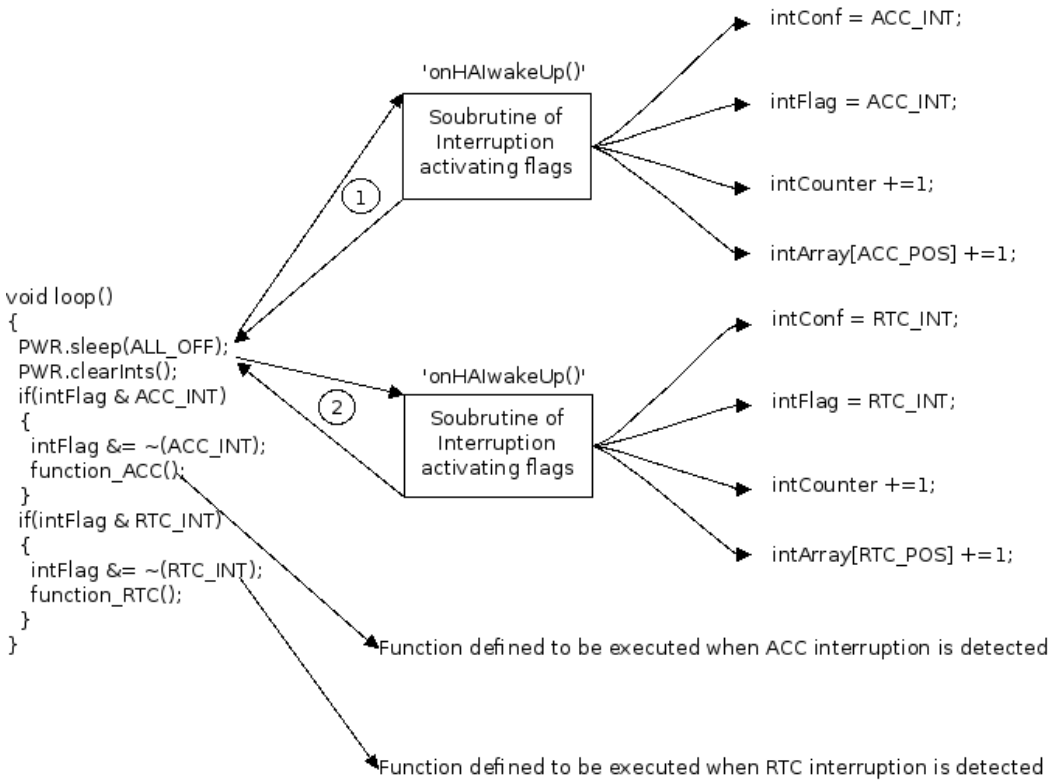


Figure 47: Treatment of chained interruptions

NOTE: The GPS and 3G/GPRS modules use the UART_1 (RXD1 and TXD1) to establish communication with the microcontroller. So that at the moment of communicating with them, the interruptions are deactivated for the milliseconds that the process lasts. It is important to point out that these modules can be activated are running all the time, in the case of the GPS receiving information from satellites and in the case of the 3G/GPRS being connected to the carrier (telephone operator) while the interruptions are activated. Operation in exclusion refers solely to the communication process of the micro with the module.

6.2. Watchdog

The Watchdog allows the microcontroller to be woken up from a low consumption state by generating an interrupt.

This interruption is software generated, the operation being different with regard to the other Waspmote modules. To unify the operation with the rest of the modules a hardware interruption has been simulated generated on receiving the software interrupt.

When the Watchdog interruption occurs, an internal subroutine is run to generated the hardware interruption simulation. The **TXD1** pin is used to detect this simulated interrupt, with the corresponding monitoring pin. For this reason, when this interruption occurs, the subroutine **onLAIwakeUP** is executed, marking the corresponding flags.

The interruption generated by the Watchdog is used to control the Waspmote's Sleep mode, allowing it to put the mote to sleep during the Watchdog's alarm cycle: **16ms, 32ms, 64ms, 128ms, 256ms, 500ms, 1s, 2s, 4s, 8s**.

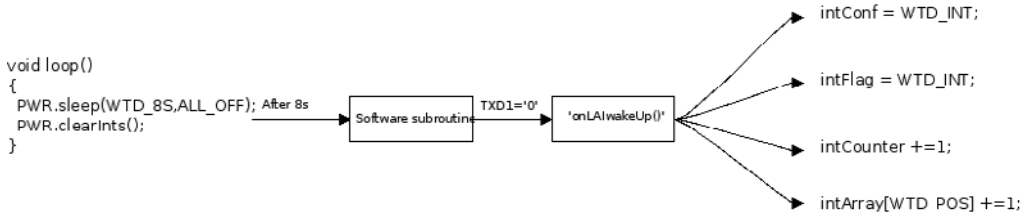


Figure 48: How the simulated interruption is generated

The generated interruptions can also be used by the Watchdog as timed alarms without the need for the microcontroller to be in energy saving mode. The Watchdog alarms can be used for time less or equal to **8s.** For longer times the RTC must be used (see following section).

More information about the Watchdog can be found in section 4.2.1.

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 **Support** and **Development sections** in the Libelium website.

6.3. RTC

The RTC can be used to set a **common time base** in all the Waspmotes in a network. As mentioned in section 3.6 the **DS32321SN** allows maximum accuracy without deviations over time, which makes it possible for the network to remain **synchronised** and for the sending and receiving tasks to be carried out in the same **time window**. The operating paradigm would be that every 'x' seconds the motes would wake up and establish communication between each other enabling data output to the control centre.

The concept is what the **Libelium** team has called ***"the network beat"***.

The RTC is capable of generating **cyclical interruptions** through 2 alarms which can be configured:

Alarm 1 can be configured in **1 second cycles** while alarm 2 can be configured for **1 minute cycles**. Both alarms can be used to generate an interruption and wake Waspmote from a low consumption mode.

The RTC generates the high level interruption so the RXD1 pin is used to capture this interrupt. It also has a single monitoring pin. When the interruption occurs in the RXD1, the subroutine 'onHAIwakeUP' is run marking the corresponding flags.

The use of the RTC has been associated with Waspmote's **Deep Sleep** and **Hibernate** modes, allowing it to put the microcontroller to sleep, activating alarm 1 in the RTC to wake it. In this way, Waspmote can be put into the lowest consumption mode and woken up using the RTC. For more information about these energy saving modes, consult sections 5.2 and 5.3.

It is also possible to use the interruptions generated by the RTC as timed alarms without the need for the microcontroller to be in energy saving mode. The use of RTC alarms is recommended for times longer than 8s, since for shorter timers the microcontroller's internal Watchdog can be used.

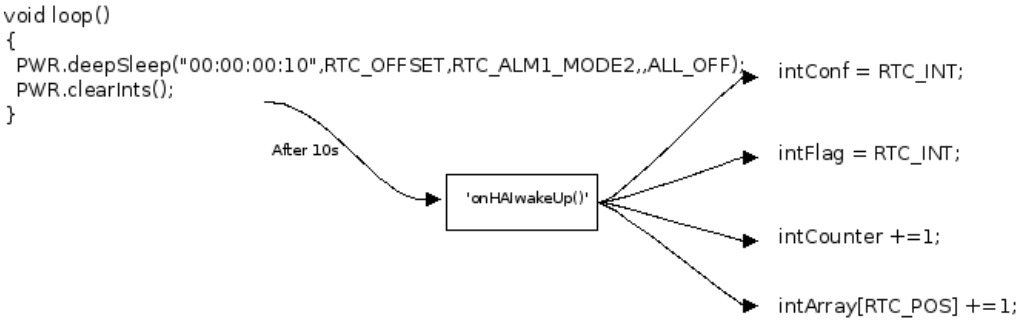


Figure 49: RTC alarm operation

More information about the RTC can be found in chapters 2.6 and 3.2.2.

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 **Support** and **Development sections** in the Libelium website.

6.4. Accelerometer

The Waspmote accelerometer (LIS3LV02DL of STMicroelectronics) can generate 2 types of interrupt: **Free Fall** or **Direction Detection Change**. To be able to manage both types of interruptions, a flag has been created in the accelerometer library, called **ACCevent** for the enabled interruption to be stored in.

The accelerometer generates a high level interrupt, so the **RXD1** pin is used to capture this interrupt. A single monitoring pin is also used. When the interruption occurs in RXD1, the subroutine **onHAIwakeUP** is run, marking the corresponding flags.

The alarms are generated in the accelerometer when certain previously defined thresholds are reached. These thresholds determine what the necessary acceleration is for free fall or to detect a change in direction. These thresholds are defined in 'WaspACC.h'.

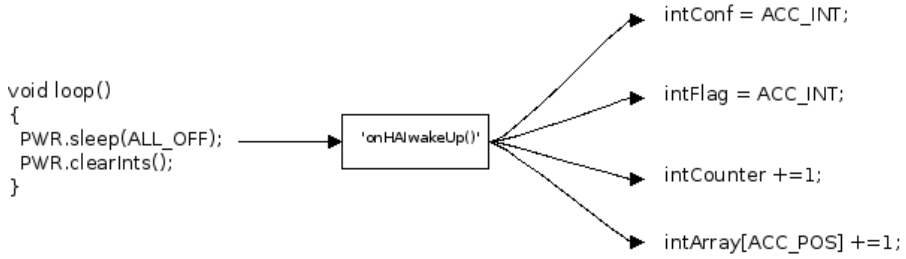


Figure 50: Accelerometer alarm operation

More information about the accelerometer can be found in chapter 6.1.2.

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 **Support** and **Development sections** in the Libelium website.

6.5. Sensors

The sensors are mainly connected to Waspmote through the microprocessor's **analog and digital inputs**. Libelium has developed several sensor integration boards which facilitate their connection with the processing unit. Each one of these sensor boards has several pins used to manage the generation and capture of interruptions. There is a pin which is connected to **RXD1**, as the interruptions generated are **high level**, and another pin that is used for monitoring.

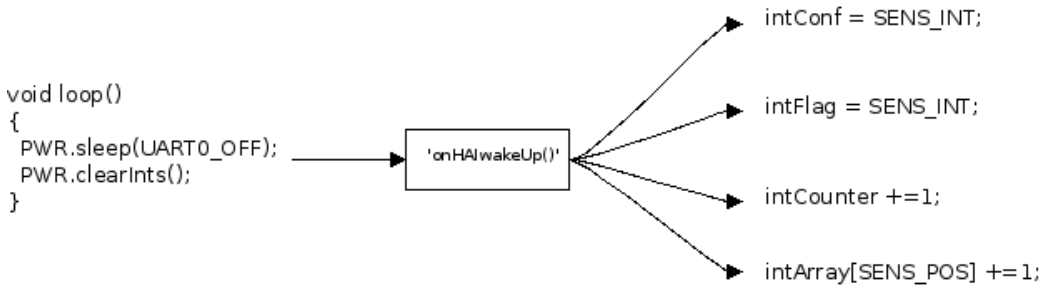


Figure 51: Sensor board alarm operation

More information about the sensors can be found in chapter 6.

Related API libraries: **WaspSensorEvent.h, WaspSensorEvent.cpp, WaspSensorGas.h, WaspSensorGas.cpp, WaspSensorPrototyping.h, WaspSensorPrototyping.cpp**

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

All the documentation is located in the **Support** and **Development sections** in the Libelium website.

6.6. GSM/GPRS

The GSM/GPRS module (SIM900 model from SIMCom) is able of generating interruptions on receiving **data from TCP and UDP sockets**, **calls** and **SMS**.

Because the GSM/GPRS module is connected to the **UART_1**, it uses the **RXD1** and **TXD1** pins to send and receive data, so on receiving data, calls or sms, it generates messages which enable it to generate interruptions.

The GSM/GPRS module generates a high level interrupt, so the **RXD1** pin is used to capture this interrupt. A single monitoring pin is also used. When the interruption occurs in RXD1, the subroutine onHAIwakeUP is run, marking the corresponding flags.

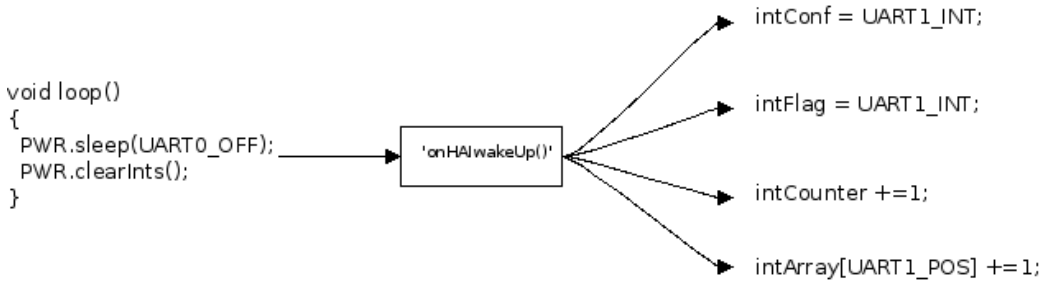


Figure 52: GSM/GPRS module alarm operation

More information about the GSM/GPRS module can be found in chapter 10.

Related API libraries: **WaspGPRS_Pro.h**, **WaspGPRS_Pro.cpp** and **WaspGPRS_Proconstants.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 **Support** and **Development sections** in the Libelium website.

6.7. Critical battery

Waspmote has a **critical battery detector**, which generates an interruption when the battery charge drops below a set voltage threshold. This is based on the batteries having a drop in voltage which is accentuated as the charge decreases. For this reason, a decrease in power voltage corresponds to a variation in the remaining charge.

This threshold can be configured **dynamically** in Waspmote as it is controlled by a **digital potentiometer** (digipot). The range of values which can be configured for this threshold is from **3.4V** to **2.95V**.

Digipot resistance (kohm)	Voltage (V)
200	3.4
180	3.36
160	3.31
140	3.27
120	3.22
100	3.18
80	3.13
60	3.09
40	3.04
20	3.00
0	2.95

Figure 53: Critical battery control levels

The digipot value to change the threshold can be modified in real time in the same program code which Waspmote is running.

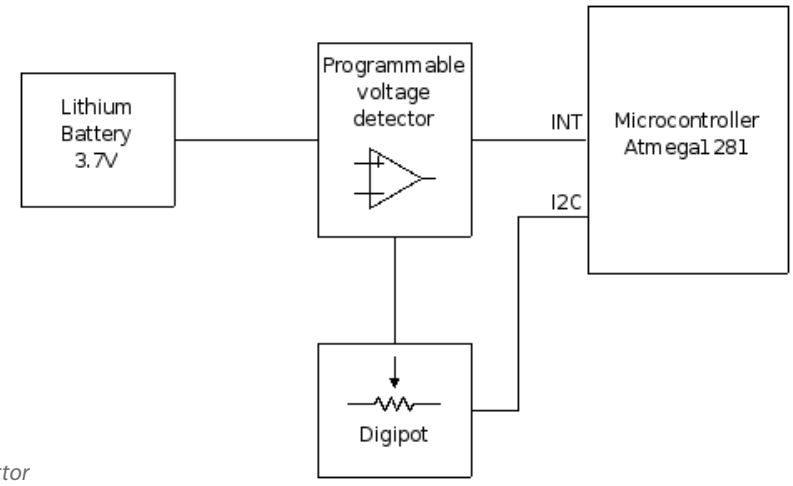


Figure 54: Critical battery detector

Example of programming a critical battery threshold

```
{
  PWR.setLowBatteryThreshold(3.4);
}
```

In the graphic below the range of configurable threshold values can be seen on the battery discharge curve.

The reason that the **low battery threshold** can be configured **dynamically** is because this level is not the same from one application to another, as the components, modules and sensors have different energy requirements in each one. For this reason, a mote does not need the same **minimum charge** level with a 1mW transmission XBee module as one does with the 100mW XBee-Pro model, or some other additional module such as the GSM - 3G/GPRS modem which needs peaks of 1A to be able to send information to the telephone operator.

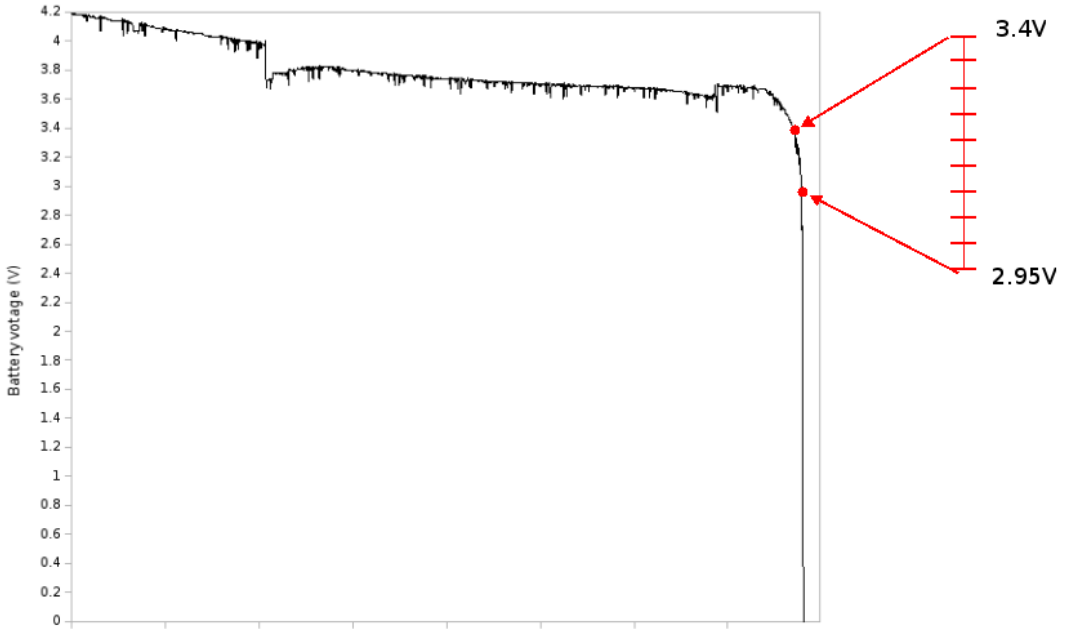


Figure 55: Configurable battery voltage levels for the critical battery detector

The threshold value must be such that when it jumps and enters the interruption treatment routine, it allows communicating with the rest of the brother nodes or the control centre through the communication module containing the message *"I am the node ID=5. My battery is running out!"* so that the network operators can replace or recharge its battery.

Because the critical battery detector maintains a low level on the **TXD1** pin, while the battery charge does not go above a set charge threshold (higher than the discharge threshold), it is not possible to re-enable the interruptions related with the TXD1. If they were re-enabled it would be possible to enter into an infinite loop because the microcontroller would always detect the low battery interrupt, as the level is below the threshold.

In systems in which the charge can be recovered (eg. charging through the solar panel), it is recommended to read the critical battery monitoring pin value every 'x' seconds, so that when a change in the value of this pin is detected, passing from low to high level, the interruptions in the TXD1 are activated again.

More information about the battery discharge curve can be found in chapter 18.

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 **Support** and **Development sections** in the Libelium website.

7. Sensors

7.1. Sensors in Waspmote

7.1.1. 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.

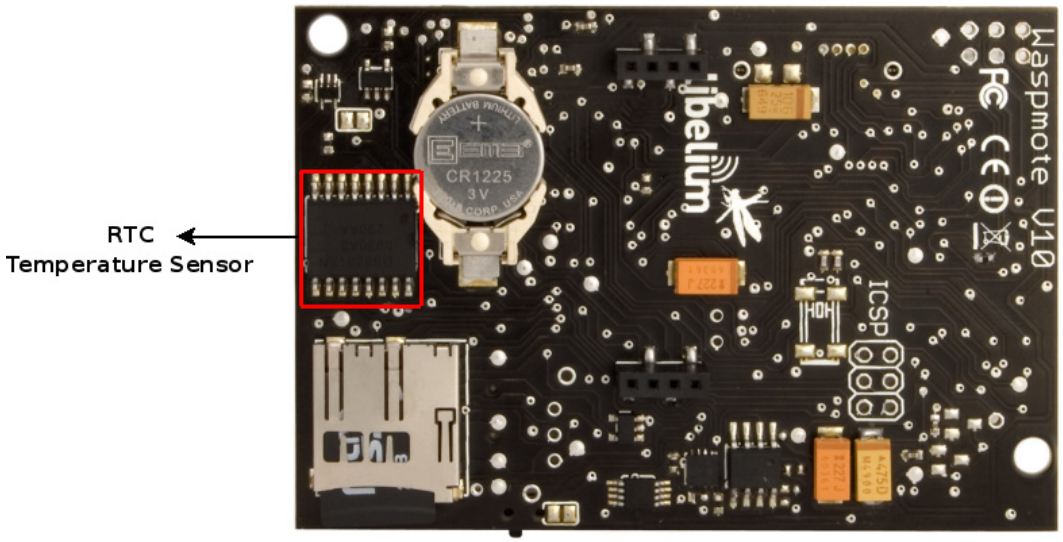


Figure 56: Temperature sensor in the RTC

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 chapter 2.6 and 5.3.

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 **Support** and **Development sections** in the Libelium website.

7.1.2. Accelerometer

Waspmote has a built in acceleration sensor **LIS3LV02DL from 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 2 kind of events: **Free Fall** and **Direction Detection Change** which was mentioned in section 6.4.

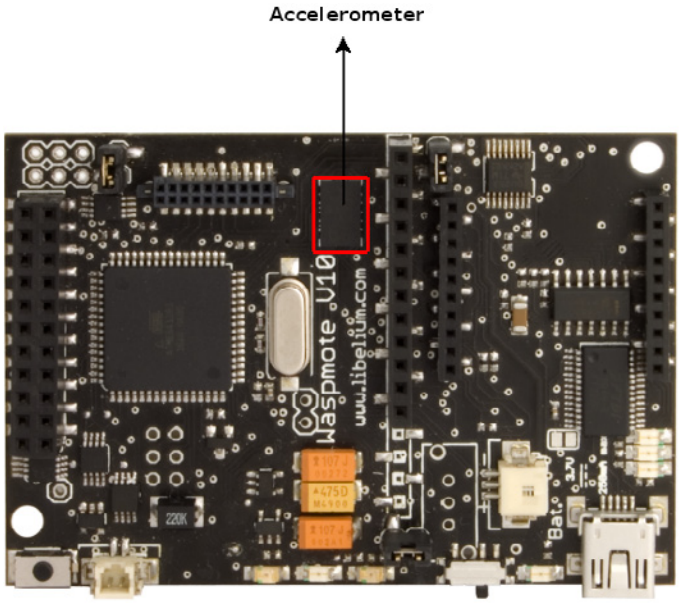


Figure 57: Accelerometer

The LIS3IV02DL is defined as a **3-axis** linear accelerometer which allows measurements to be taken from the sensor through the **I2C** interface. The accelerometer has a **12 bit** resolution (4096 possible values) and includes 2 measurement ranges **±2g** and **±6g**.

In thecase of using the first range ±2g, the accuracy of the reading would be 4096/4g = **1024 LSb/g**

In the case of using the second range of ±6g the accuracy of the reading would be 4096/12g = **340LSb/g**

Another of the parameters that can be adjusted is the accelerometer's **refresh rate**, i.e. how often internal log updates are required. This allows different **bandwidths** to be used. The higher the refresh rate, the higher the consumption, but there is also greater accuracy and more real data are obtained.

By default the accelerometer updates **40** times a second (40Hz), but it can be configured to increase this rate to **160, 640** or even **2560** times a second (2560Hz). In the latter case a value would be obtained every **0.39ms**.

For most applications a rate of 40Hz is sufficient to detect any type of change or programmed interrupt. However, as Waspmote is designed to perform **mobile applications** which can require **maximum accuracy** and **real time** data collection, the use of higher bandwidths is possible. For this reason specific functions have been defined within the programming library.

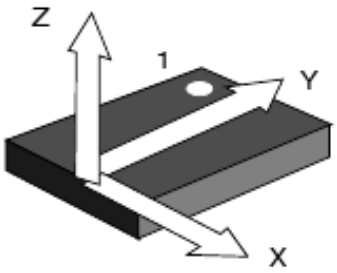


Figure 58: Axes in the LIS3LV02DL accelerometer

This accelerometer has an auto-test for the integrity of the measurements taken, being able to disregard measurements when they are not valid. 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 RDY pin to generate the interruptions.

The accelerometer has 2 types of event which can generate an interrupt: free fall and change of direction.

The free fall interruption is generated when the set threshold is exceeded in any of the axes where the interruption is enabled.

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

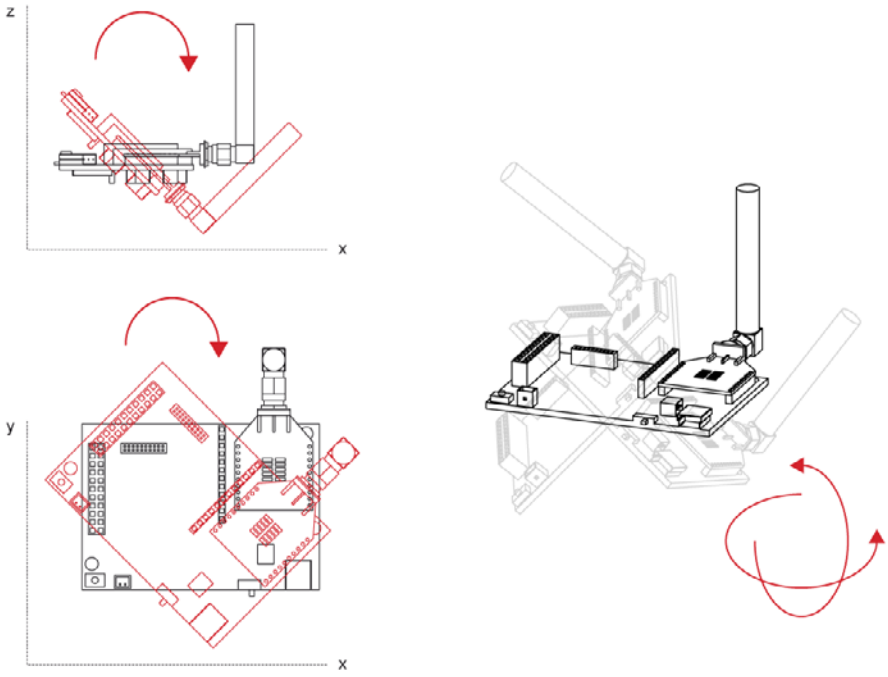
The direction change interruption is generated when 2 set thresholds are exceeded. The aim of this interruption is to detect that a movement in one direction is taking place and a change occurs in that movement, for that reason 2 thresholds are used.

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

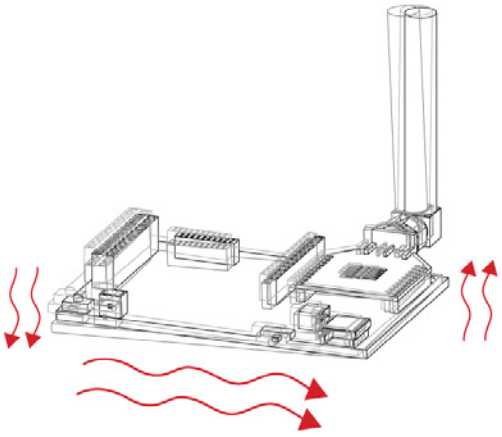
```
{
  acc.getX();
  acc.getY();
  acc.getZ();
}
```

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

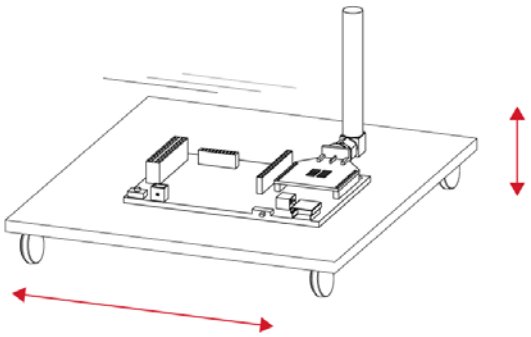
Rotation and Twist:



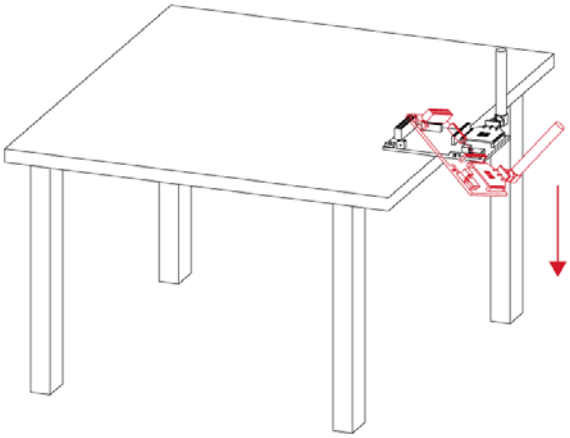
Vibration:



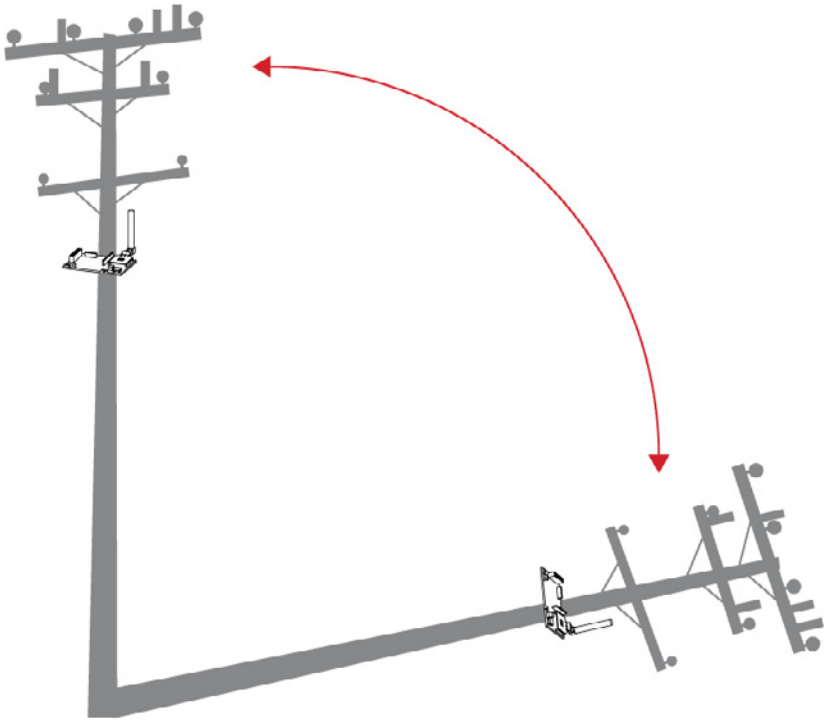
Acceleration:



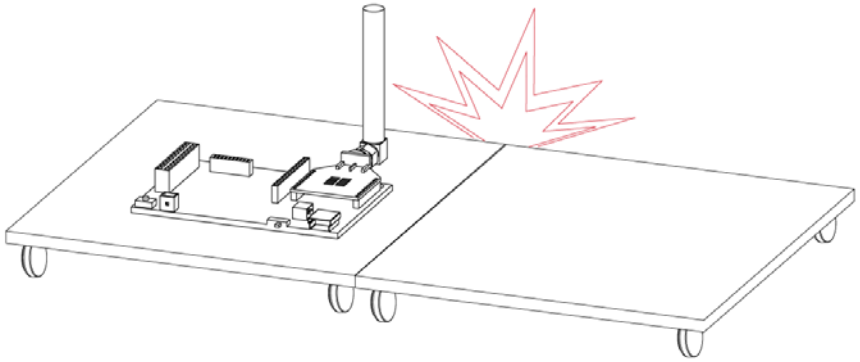
Free fall:



Free fall of objects in which it is installed:



Crash:



More information about interruptions generated by the accelerometer can be found in chapter 5.4.

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 **Support** and **Development sections** in the Libelium website.

7.2. 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 **2x11** 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 section 7.4) 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.

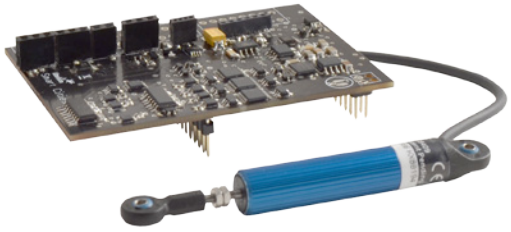
7.3. Sensor Boards

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 section 3.5.1. 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** Vibration, hall effect (doors and windows), person detection PIR • **Emergencies** Presence detection and water level sensors, temperature • **Control of goods in logistics** Vibration and impact sensors	• Pressure/Weight • Bend • Vibration • Impact • Hall Effect • Tilt • Temperature (+/-) • Liquid Presence • Liquid Level • Luminosity • Presence (PIR) • Stretch

SMART CITIES



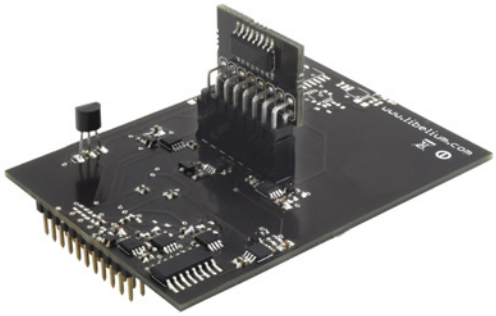
APPLICATIONS

- **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



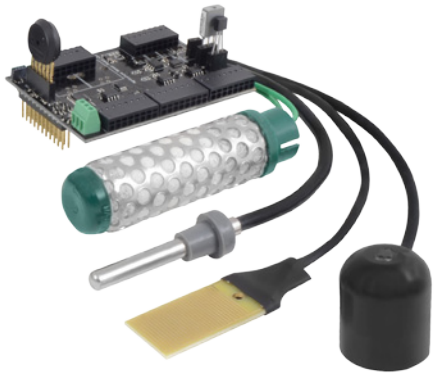
APPLICATIONS

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

SENSORS

- Magnetic Field

AGRICULTURE



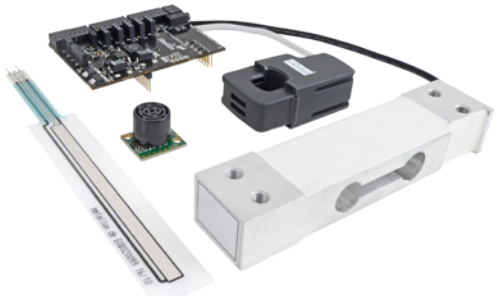
APPLICATIONS

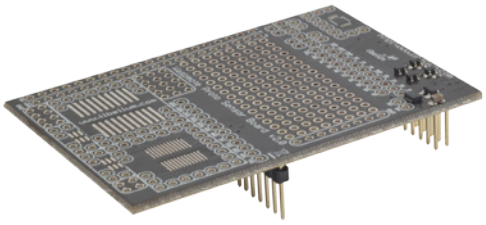
- **Precision Agriculture**
  Lleaf temperature, fruit diameter
- **Irrigation Systems**
  Soil moisture, leaf wetness
- **Greenhouses**
  Solar radiation, humidity, temperature
- **Weather Stations**
  Anemometer, wind vane, pluviometer

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

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/support/waspmote

7.4. 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

8. 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 at a speed of 38400bps.

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
XBee-XSC	RF	900MHz	100mW	-106dBm	12Km

** Line of sight and 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.

8.1. 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 59: XBee 802.15.4

Figure 60: 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.

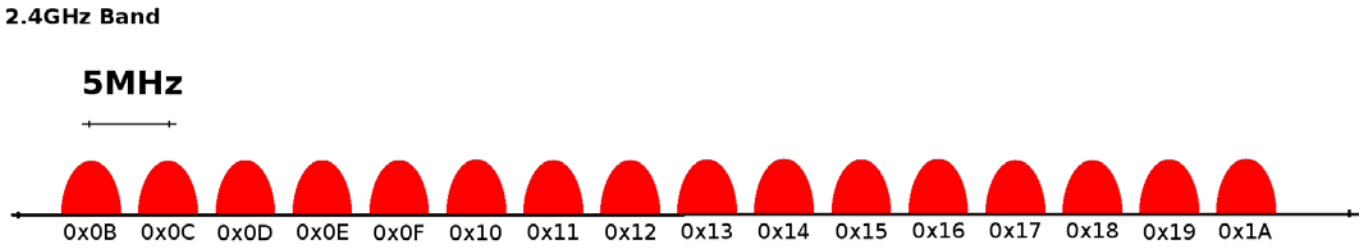


Figure 61: Frequency channels in the 2.4GHz band

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 62: 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.encryptionMode(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 layout of this type of network is P2P, as the nodes establish point to point connections with brother nodes through the use of parameters such as the MAC or network address.

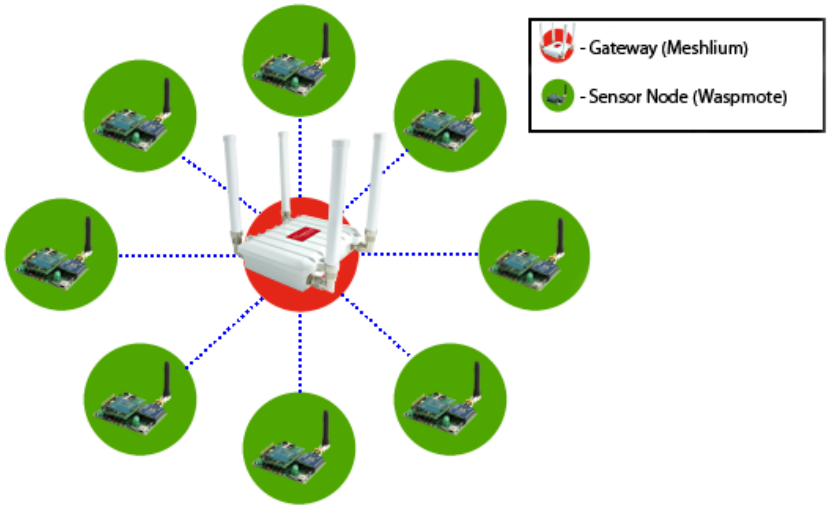


Figure 63: Star topology

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 64: Transmission power values

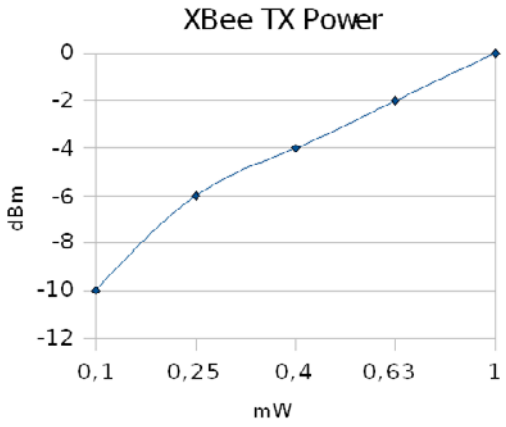


Figure 65: XBee TX Power

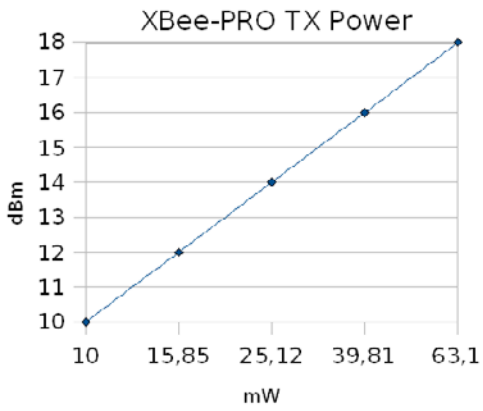


Figure 66: XBee-PRO TX Power

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 **Support** and **Development sections** in the Libelium website.

8.2. 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 67: XBee ZigBee

Figure 68: 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.

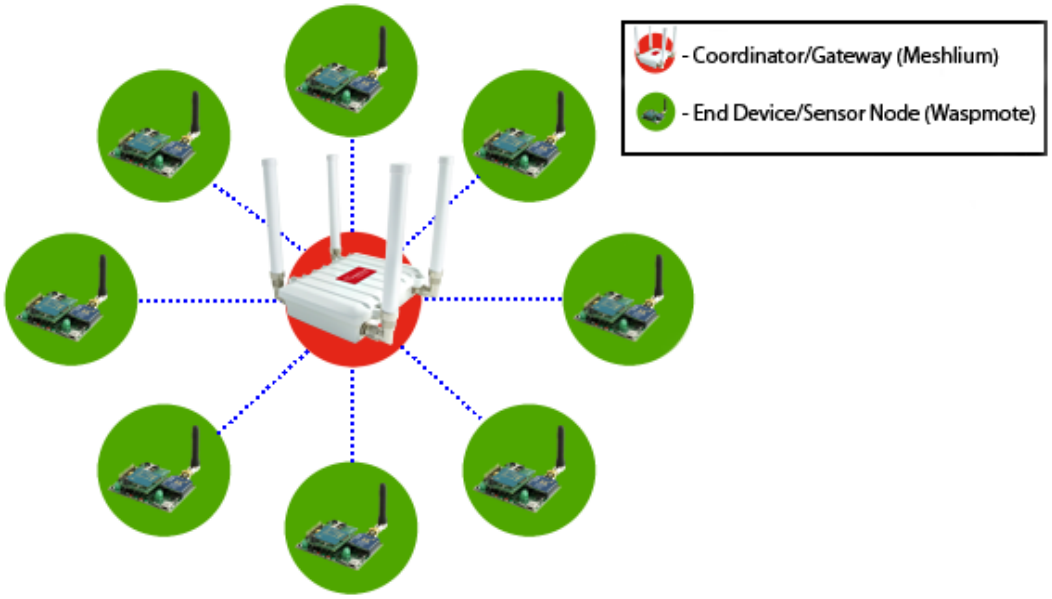


Figure 69: Star topology

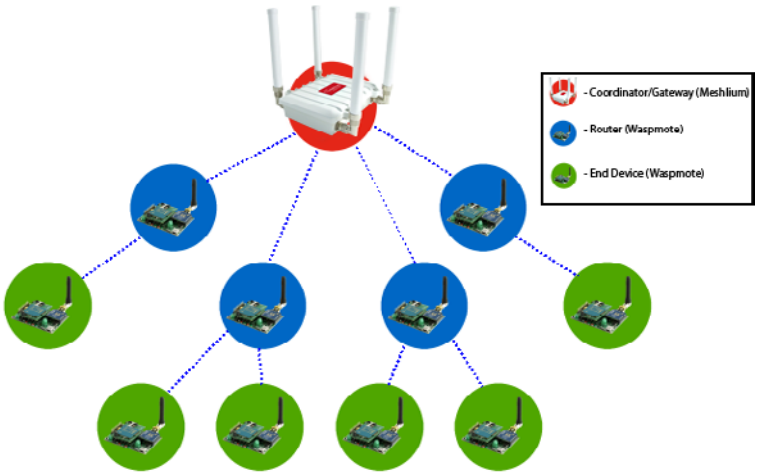


Figure 70: Tree topology

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 71: Transmission power values

A mode called **boost** can be set up which improves reception sensibility by **1dB** and transmission power by **2dB**. Using this mode improves reach, but also increases consumption.

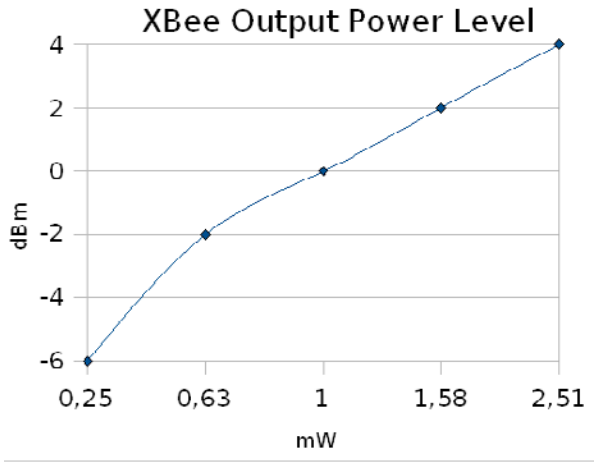


Figure 72: XBee Output Power Level (Boost Mode ON)

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 **Support** and **Development sections** in the Libelium website.

8.3. XBee - 868

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

Figure 73: 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 chapter 21, more information can be obtained about the **Certifications**.

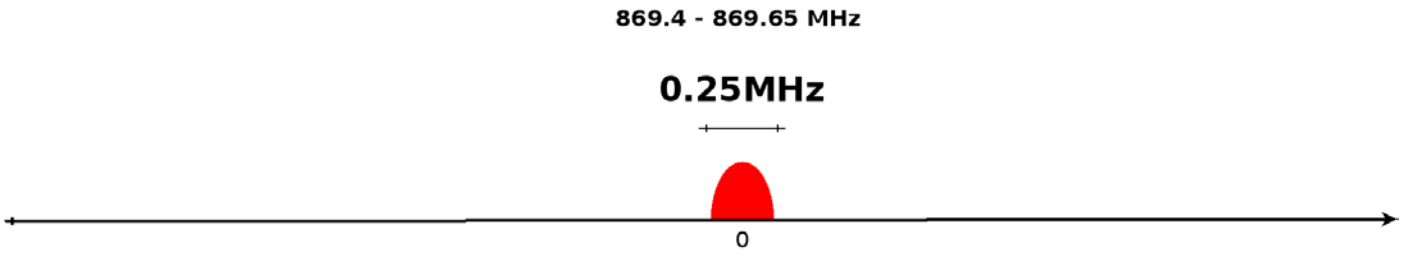


Figure 74: Channel frequency on 869MHz

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 initialisation function and giving it a key to use in the encryption.

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

The classic topology for this type of network is P2P, as the nodes can establish point to point connections with brother nodes through the use of parameters such as the MAC or network address.

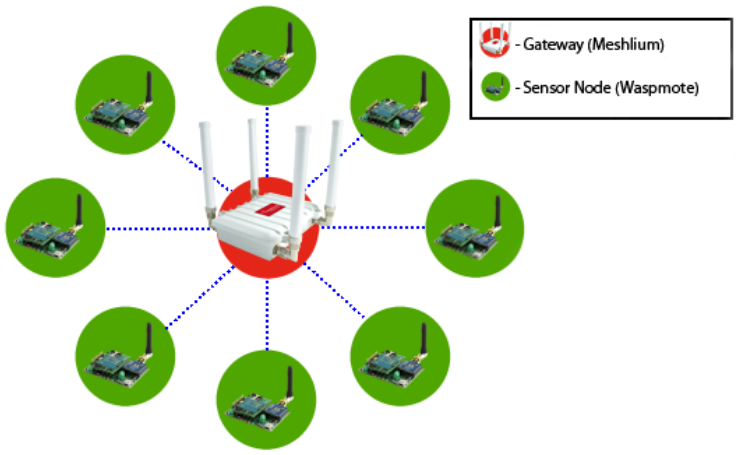


Figure 75: Star topology

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 76: Transmission power values

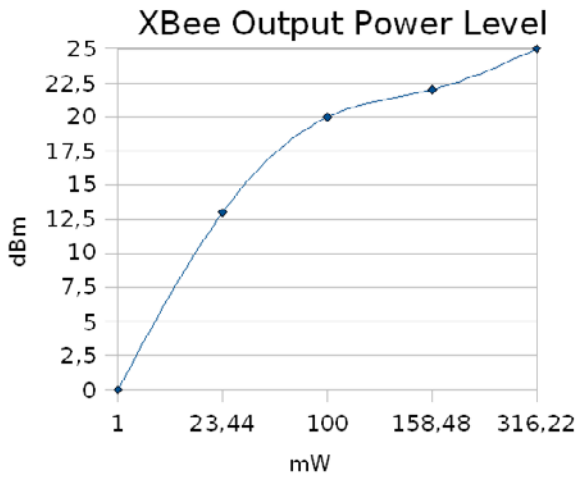


Figure 77: XBee TX Power

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 **Support** and **Development sections** in the Libelium website.

8.4. XBee - 900

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

Figure 78: 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 chapter 21, more information can be obtained about the **Certifications**.

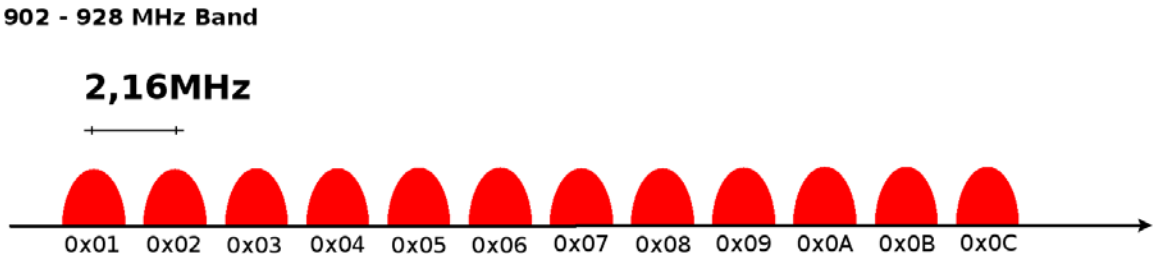


Figure 79: Channel frequencies in the 900MHz band

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 initialisation function and giving it a key to use in the encryption.

```
{
  xbee.encryptionMode(1);
  xbee.setLinkKey(key);
}
```

The classic topology for this type of network is P2P, 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.

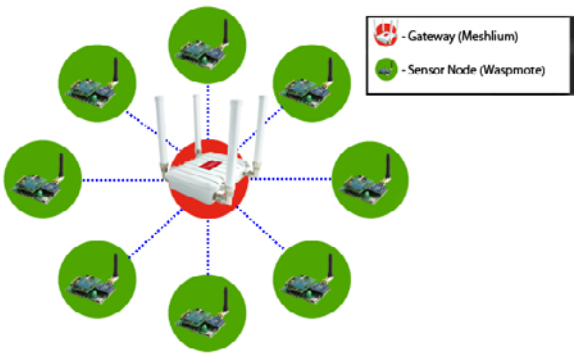


Figure 80: Star topology

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

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

All the documentation is located in the **Support** and **Development sections** in the Libelium website.

8.5. XBee-XSC

Module	Frequency	Tx Power	Sensitivity	Channels	Distance
XBee-XSC	902-928MHz	100mW	-106dBm	7	12km

Figure 81: XBee XSC

The frequency used is the 900MHz band (United States and Canada only), using 7 channels with a transmission rate of 9600bps. In chapter 21, more information can be obtained about the **Certifications**.

The classic topology of this type of network is P2P, as the nodes can establish point to point connections with brother nodes through the use of parameters such as the MAC or network address.

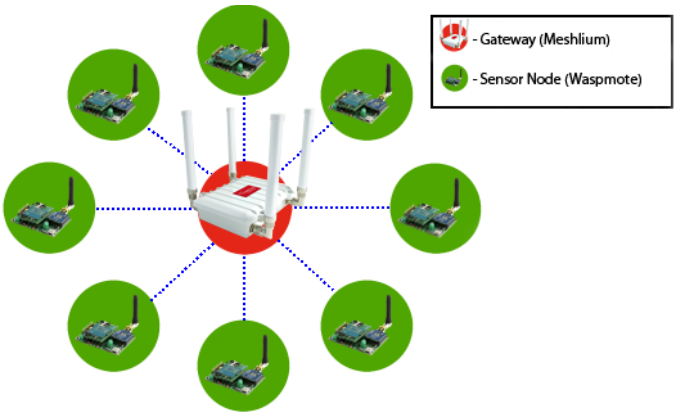


Figure 82: Star topology

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

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

All the documentation is located in the **Support** and **Development sections** in the Libelium website.

8.6. XBee-DigiMesh

The XBee-802.15.4 and XBee-900 modules can use an optional firmware (**DigiMesh**) so that they are able of creating **mesh networks** instead of the usual point to point (p2p) topoplogy. This firmware has been developed by Digi aimed for allowing modules to sleep, synchronise 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.
- **P2P architecture:** 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 synchronisation 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 of parameters such as the MAC or network address or by making **multi-jump connections.**

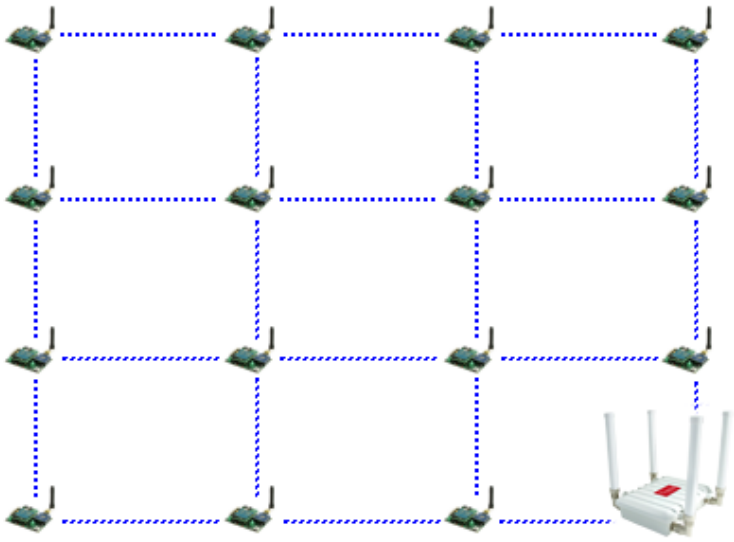


Figure 83: Mesh topology

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, being able to pass from one to the other by changing the firmware. 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, being able to pass from one to the other by changing the firmware. 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 **Support** and **Development sections** in the Libelium website.

8.7. 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:

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

Visual recognition of the quality of the RSSI signal:

Three LEDs have been included to show the **RSSI**. These LEDs indicate the signal quality of the last packet received.

By default this indicator system is disconnected to avoid unnecessary consumption. For this reason, a jumper must initially be connected (see section 3.8) so that this part of the circuit is connected to the XBee module.

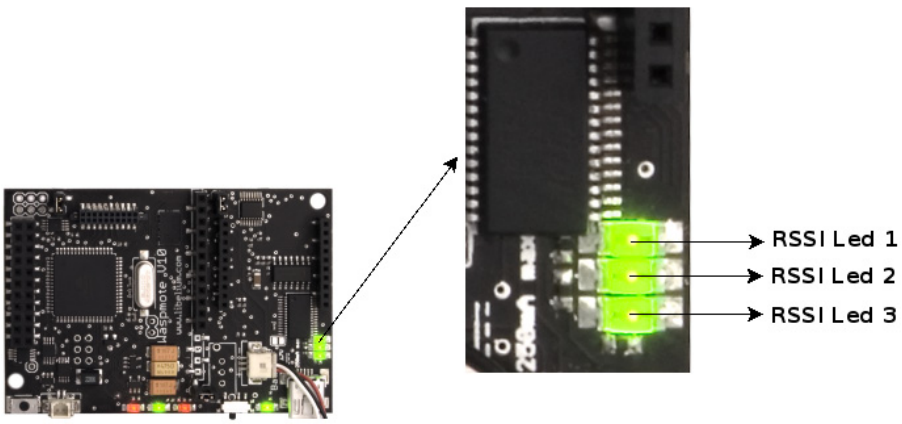


Figure 84: RSSI level LED indicator

N° of LEDs switched on	RSSI received
1 LED	RSSI > (Sensitivity + 11.1dB)
1+2 LEDs	RSSI > (Sensitivity + 21.23dB)
1+2+3 LEDs	RSSI > (Sensitivity + 31.82dB)

Figure 85: Relationship of lit LEDs with RSSI received

In a second step the **XBee.setRSSIOutput()** function must be run to establish how long these LEDs are turned on. This time can vary from 0 (deactivated) to the time a new packet is received (FF).

```
{
  xbee.setRSSItime(0xFF);
  xbee.getRSSItime();
}
```

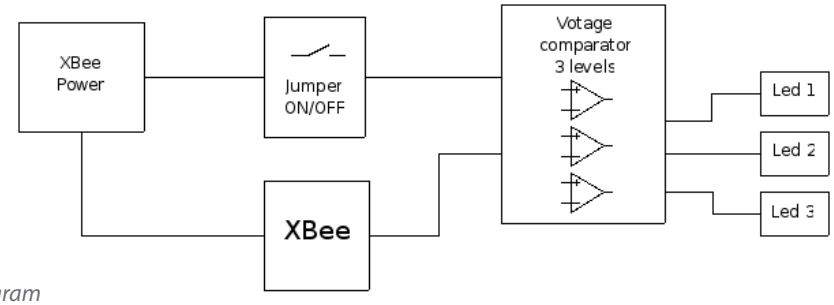


Figure 86: RSSI indicator diagram

9. 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 87: 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.**

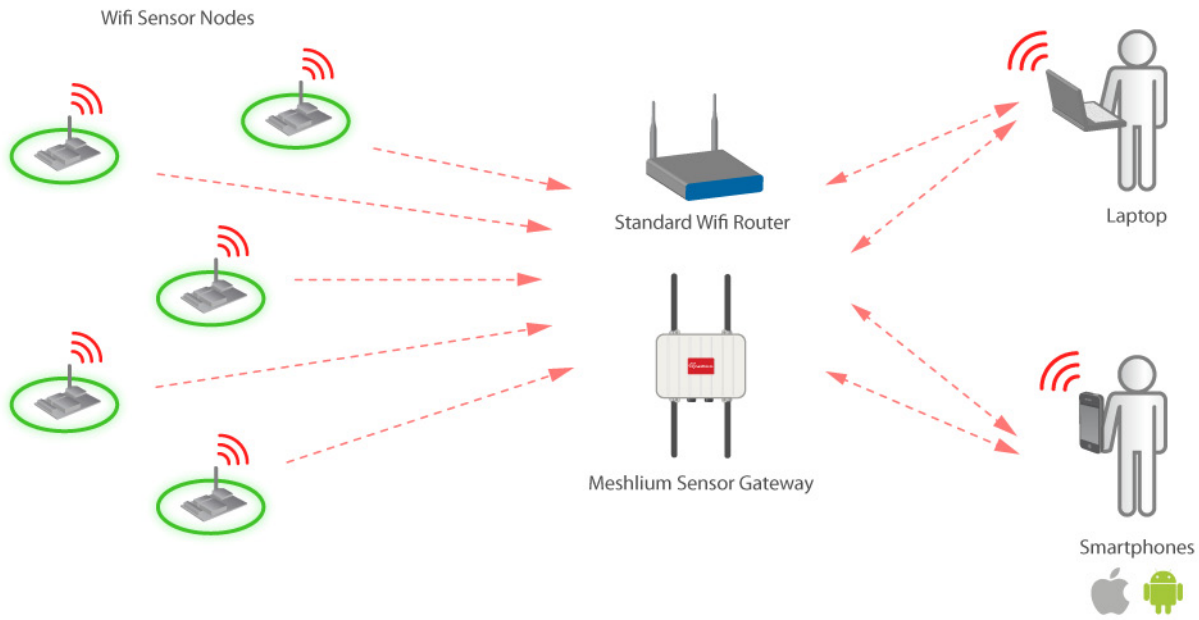
All the documentation is located in the **Support** and **Development sections** in the Libelium website.

9.1. Wifi Topologies

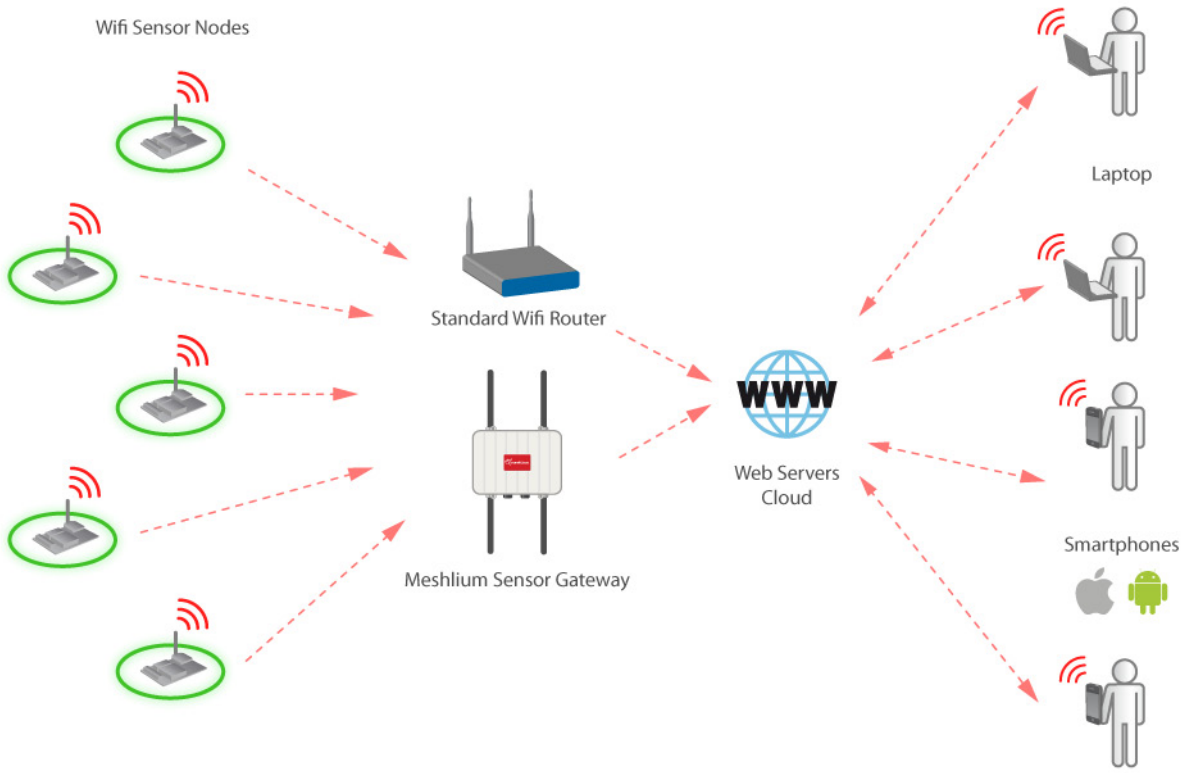
9.1.1. Access Point

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.

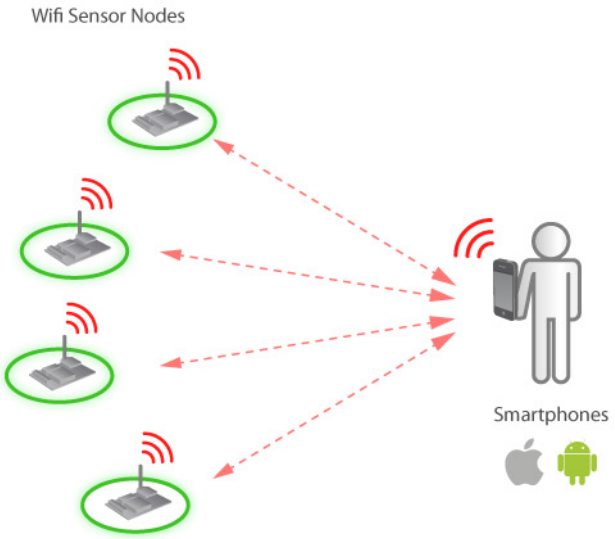
Using Meshlium as Wifi Access Point allows to control and to store the messages received from the Wifi module, or allows to combine Wifi technology with other protocols such as ZigBee. Meshlium may work as:

- 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 SmartPhones scanner (detects iPhone and Android devices)

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

9.1.2. Ad-hoc mode with iPhone/Android

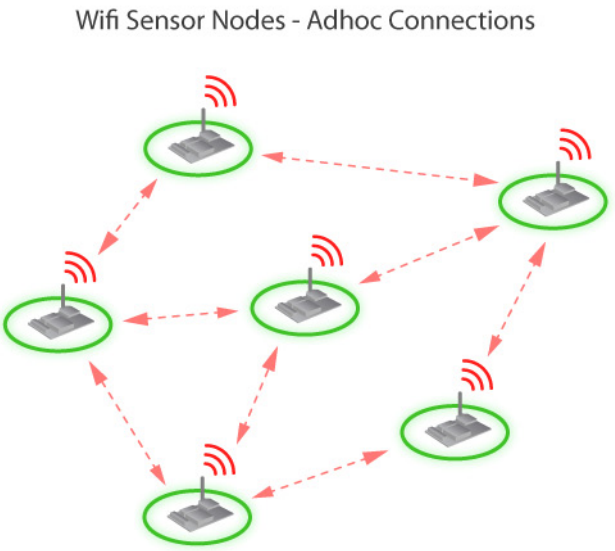
The following diagram shows how Android and iPhone devices can communicate directly with the Wifi integrated in Waspmote through an Adhoc Wifi network without any extra router or gateway.



More information about this topology in section 9.2 "Connecting to a Smartphone directly".

9.1.3. Ad-hoc mode between Waspmotes

The Adhoc topology may also be used between Waspmotes creating a point-to-point network in that each module Wifi is linked directly to every other Wifi device on the Adhoc network. There is not access point.

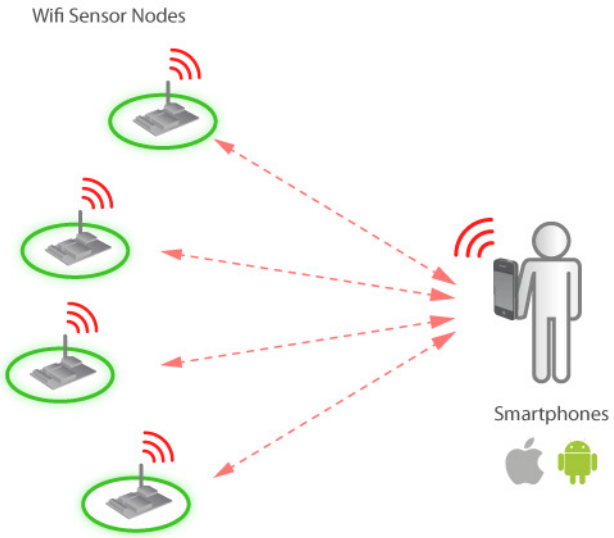


All module Wifi on the Adhoc network participate in keeping the network alive and each keeps track of the other active devices on the network by sending and receiving beacon and probe packets. IP addresses are automatically assigned through the Auto IP function.

Since Auto IP fixes the first two bytes of the IP address you want to use the net-mask of 255.255.0.0 so that other device connecting to the module can be reached. Alternatively you can set the netmask to a smaller subnet if the other device's IP addresses are being statically to he same subnet as the Adhoc device.

9.2. 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_module**

9.2.1. Connecting to an iPhone

9.2.1.1. 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**

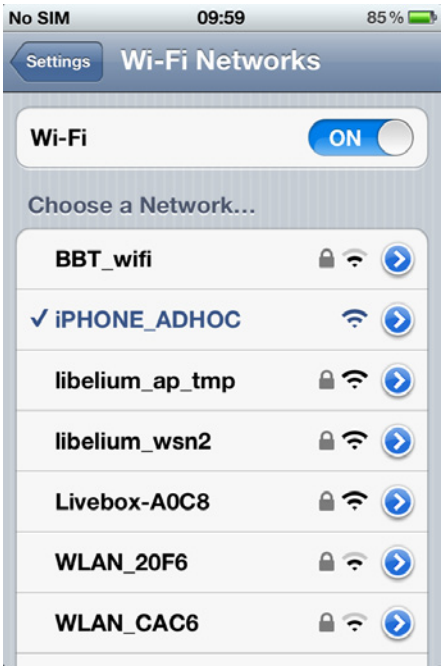
- Then double click on the icon, or right click and open with iTunes.
- Inside iTunes, on the left panel, click on DEVICES->Your Device.
- Select on the top "Apps", and select Sync Apps. Drag into the desired screen Waspmote Wifi app.

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

9.2.1.2. 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.

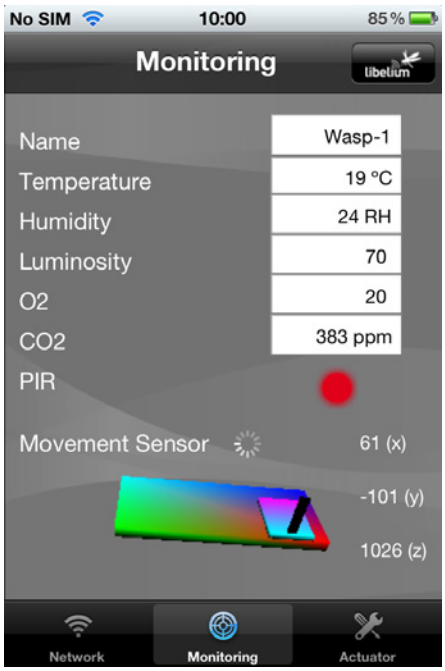


Once connected, you will see a blue Wifi icon in the top of the screen:

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**

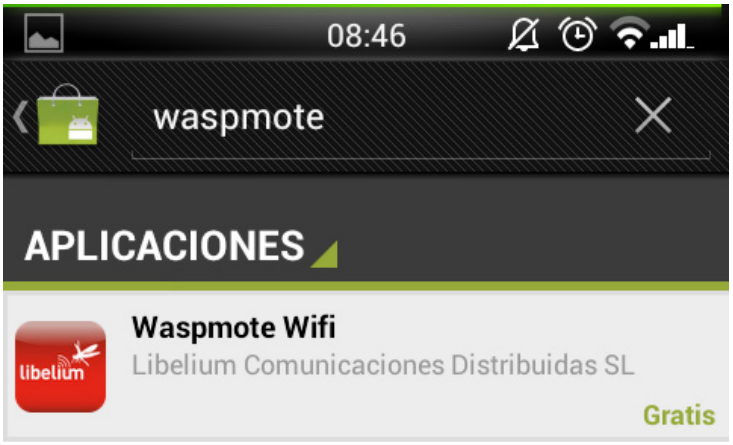
9.2.2. Connecting to an Android

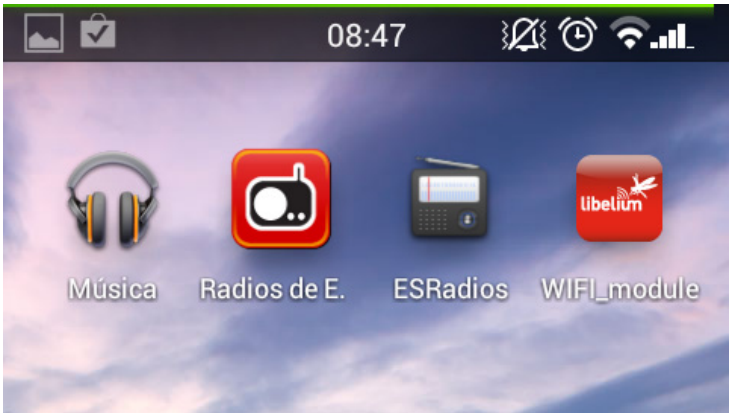
There is a feature since the version Android 2.2 or higher that allows us to create an Adhoc network and allows Waspmote to connect to it.

9.2.2.1. 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**

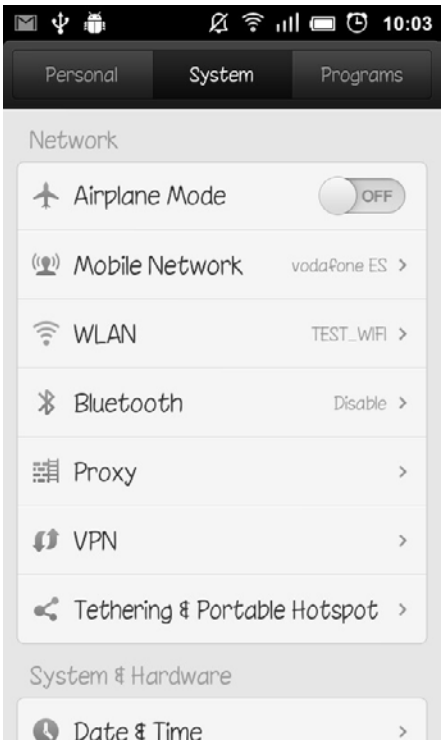
- Insert it to the SD card of your Android device.
- Then explore the SD card in your Android device and install the application. You can explore the SD card with "Astro", "ES Explora", or "File Explorer" applications.

9.2.2.2. 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)



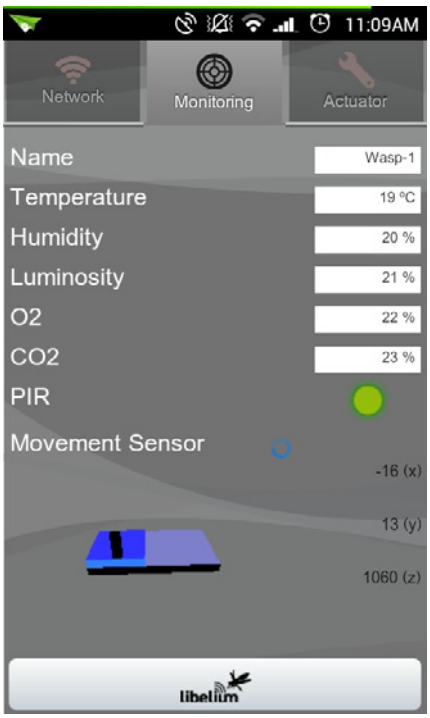
Then configure the WLAN hotspot (name= ANDROID, Security= Open). This settings you can change if you change as well the Waspmote code.

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**

10. 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.

To be able to share information between both ZigBee and Bluetooth Networks please go to the *"Sharing data between ZigBee and Bluetooth Wireless Sensor Networks"* in the Meshlium chapter in the current manual.

10.1. Technical specifications

- **Bluetooth Chip:** eUnistone 31308/2
- **Version:** Bluetooth 2.0 + EDR (Configurable BT 1.2)
- **TX Power:** 2.5dBm
- **RX Sensitivity:** -86dBm
- **Antenna:** 2dBi / 5dBi
- **Antenna Connector:** RPSMA
- **Outdoor Range:** 250m LOS (Line of sight)
- **Indoor Range:** 30m NLOS (Non line of sight)

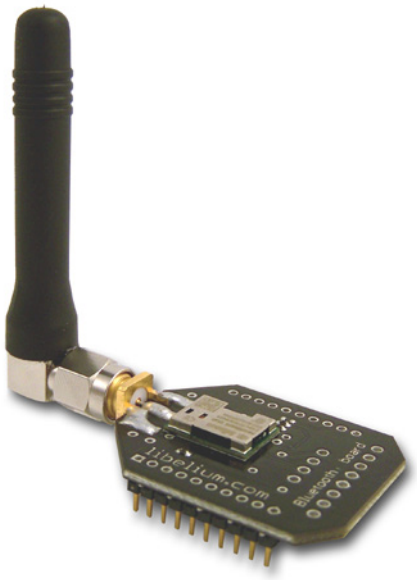


Figure 88: Bluetooth Module

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

This module communicate with the microcontroller through the UART_0 at 38400bps.

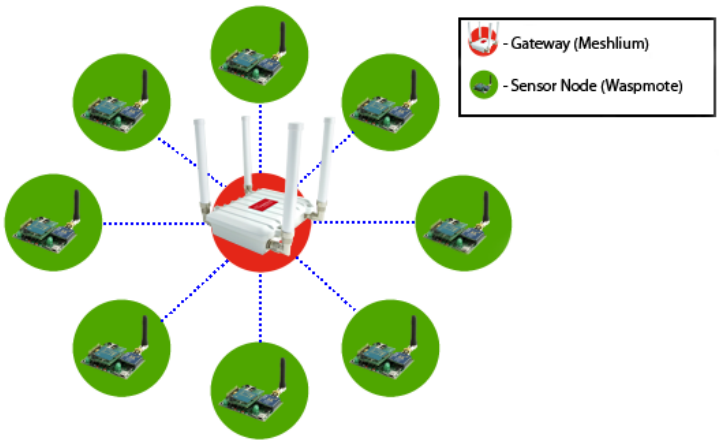


Figure 89: Start topology

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

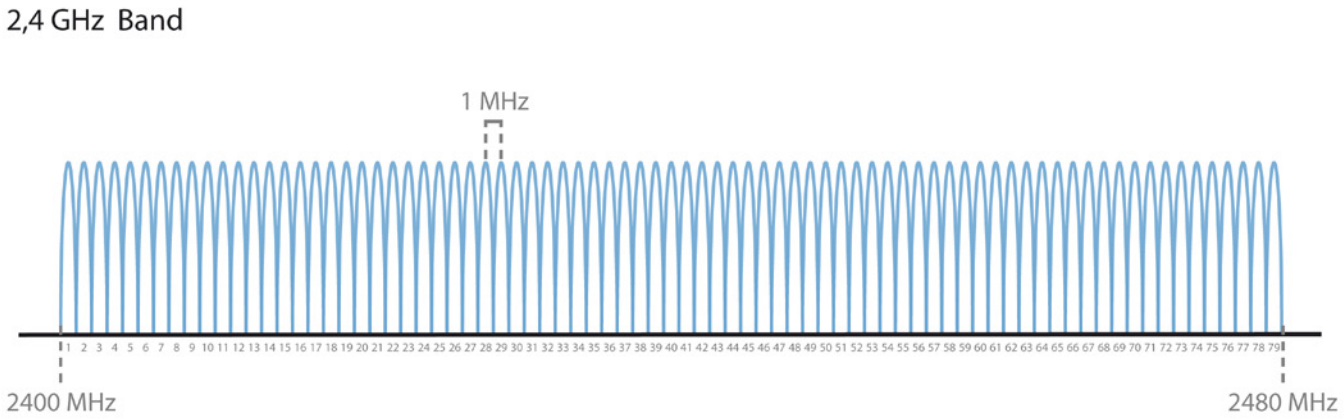


Figure 90: Frequency channels in the 2.4GHz band

Bluetooth modules have some important parameters for their configuration:

- **MAC:** it is the unique address (64b) of each module that is used to send data.
- **Public Name (ESSID):** it is the name that appears when a scan is performed in order to find new devices.
- **Class of Device (CoD):** it is a 6 digit number that allows you to identify different groups of devices. These groups may have different functions and this a way to identify them.
- **Service name:** it is a identifier to differentiate the devices depending on the service they are offering.
- **Service channel:** it is a channel to set the service on it. It may take a value between 0-30 and allows to enable up to 30 different services on the same device.

10.2. Scanning for devices

A basic Bluetooth function is scanning the network looking for new devices. This process lets you find up to **8** devices per scan, with a variable response time depending on the number of devices. This time may vary from **10s** to **30s**. The scan returns the **MAC**, **ESID** and **CoD** of each device.

The API function that handles this scan is:

```
{
        BT.scanNetwork();
}
```

10.3. Discovering a device

Sending data to another device requires the knowledge of some parameters such as the MAC and the service channel. This process is called **'Device discovery'** and returns the **service name** and the **service channel**.

The API function that handles this process is:

```
{
        BT.discoverDevice(MAC);
}
```

Bluetooth devices have different options to become visible or not. A device may be found when performing a scan if this option has been enabled previously. The different visibility options are:

- **No discoverable: the device can not be discovered when performing a scan**
- **Inquiry enabled: the device allows to be discovered in a scan**
- **Page Scan enabled: the device allows to be asked directly about its active services**
- **Discoverable: the device allows to be discovered and asked about its services**

These Bluetooth modules allows an unique connection at the same time, setting the device which creates the connection as master and the device which accepts the connection as slave.

10.4. Security

These Bluetooth modules allows to enable security to maintain encrypted communications. The different security options are:

- **Pre-defined security:** security is enabled on both devices and when a communication is started the PIN is not asked
- **On-demand security:** security is enabled on both devices and when a communication is started the PIN is required. Both devices have to set the same PIN to establish the communication, but this PIN may be different from the one selected when the security was enabled
- **No security:** security is disabled

Enabling security is as easy as using the following function:

```
{
        BT.setSecurity(BT_SECURITY_3, "waspmote_bt_pin");
}
```

Once a secure communication has been established between a pair of devices, these modules become **'Trusted Devices' (TD)**, so the next times they want to establish a communication the PIN will not be required because they are already authenticated. Therefore, delay caused by the authentication process is avoided. The maximum size of the **TD** table is **5**.

10.5. Services

Sending data to another device requires the knowledge of some parameters such as the **MAC** and the **service channel**. Each module may have different service channels (up to 30) and run a different service in each one. These service channels are a software multiplexing of different services.

Bluetooth protocol specifies different profiles such as one for sending files or other to use the devices as virtual serial ports. The only profile that supports this Bluetooth device is the **Serial Port Profile (SPP)**, which allows to send data emulating a virtual serial port. This profile does not allow to send files between a pair of devices.

Therefore, to be able to receive data in a mobile or some other device, this one have to support SPP profile and the data will be received as if they'd have been sent through the serial port.

10.6. Adaptive Frequency Hopping (AFH)

Bluetooth transceivers use Frequency Hopping Spread Sequence (FHSS) as modulation algorithm. The idea is to use a different frequencies when transmitting the signal. This way each bit of information is sent using a different channel in a pseudorandom way. FHSS improves the immunity to interference as the signal is moving continually through the entire defined spectrum (2400-2479MHz). Bluetooth performs 1600 hops by second using this 79 channels.

The Waspmote Bluetooth module uses an improved version of FHSS called **Adaptive Frequency Hopping (AFH)**. Using AFH, the frequency bands which contains interferences caused by other protocols such as ZigBee and Wifi can be detected and 'black listed' in order to avoid them in future transmissions. This way, when we detect some channels are being used by other networks we will stop transmitting on this band and will spread our frequency hops among the rest of the free of interference channels.

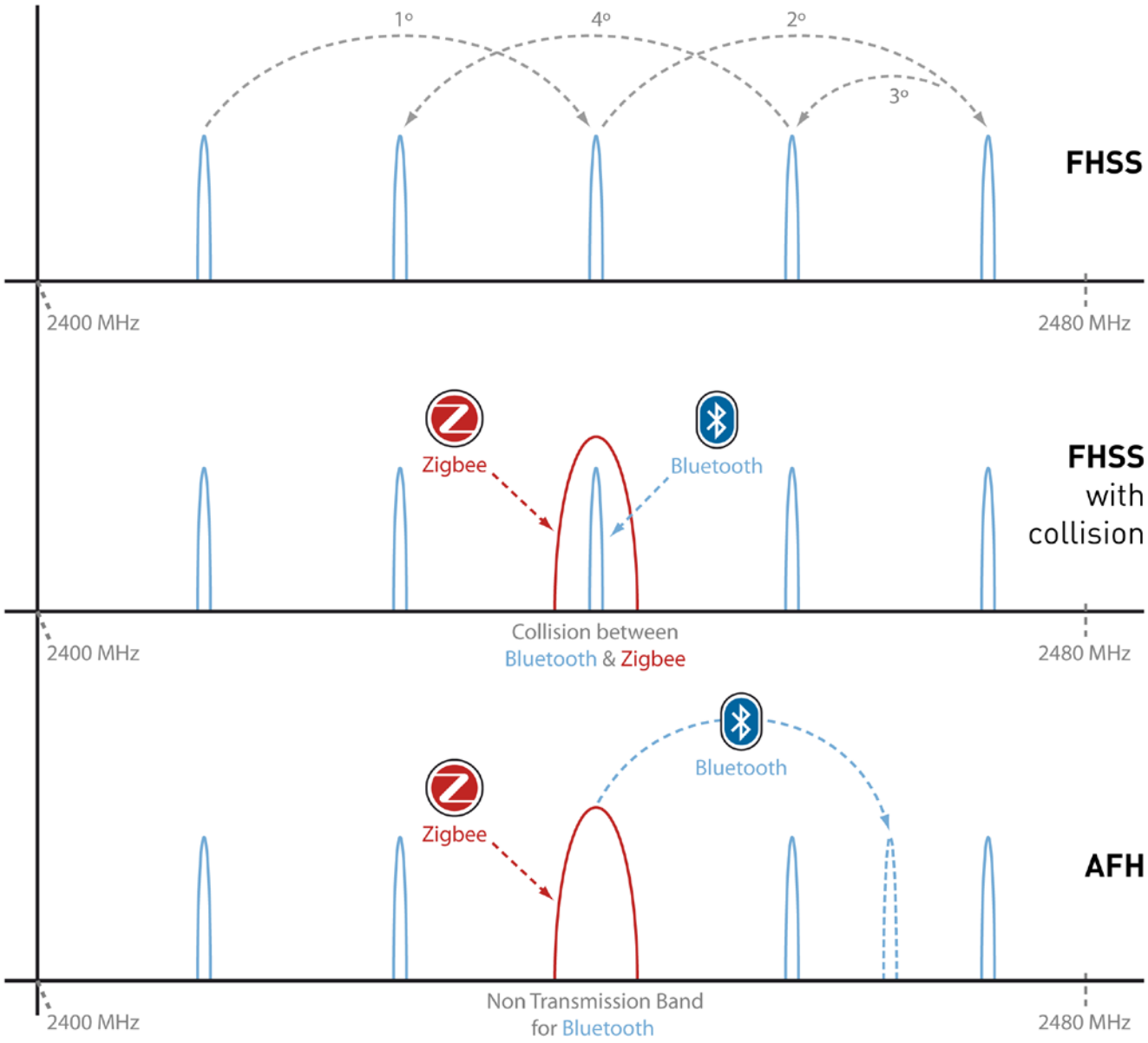


Figure 91: Non Transmission Band for Bluetooth

In order to detect if there is a possible interference in a certain band of frequency the main parameters analyzed are: Packet Loss Ratio (PLR) and Received Signal Strength Indicator (RSSI).

Both master and slave Bluetooth devices can set channels as bad ones and communicate them to the other device in order to have a both sides analysis as the interferences received by the node A may defer from the interferences received by the node B.

AFH is specially oriented to work in heterogeneous environments where there are other networks working at the same time such as **ZigBee** and **Wifi**.

10.7. Sharing data between ZigBee and Bluetooth Wireless Sensor Networks

Read the **Meshlium** chapter in the current manual.

Related API libraries: **WaspBT.h**, **WaspBT.cpp**, **WaspBTConstants.h**

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

All the documentation is located in the **Support** and **Development sections** in the Libelium website.

11. 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

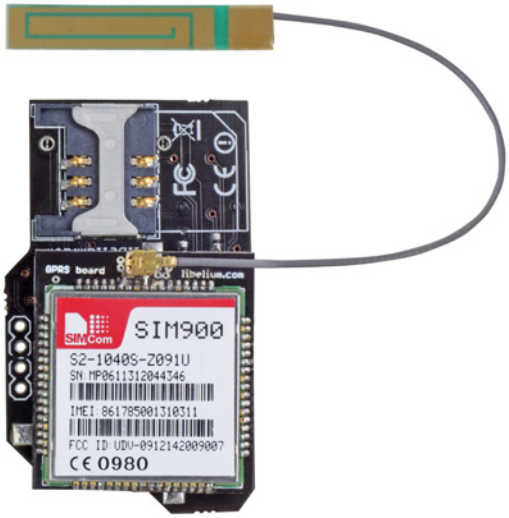


Figure 92: GSM/GPRS module

This module can carry out the following tasks:

- Making/Receiving calls
- Making 'x' tone missed calls
- Sending/Receiving SMS
- Single connection and multiple connections TCP/IP and UDP/IP clients
- TCP/IP server.
- HTTP Service
- FTP Service (downloading and uploading files)

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

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

As the GSM/GPRS module is connected to the UART_1, therefore using the RXD1 and TXD1 pins, it is possible to use the arrival of calls, SMS or data to the socket to generate interruptions. All the information about the module interruptions can be consulted in chapter 5.6.

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

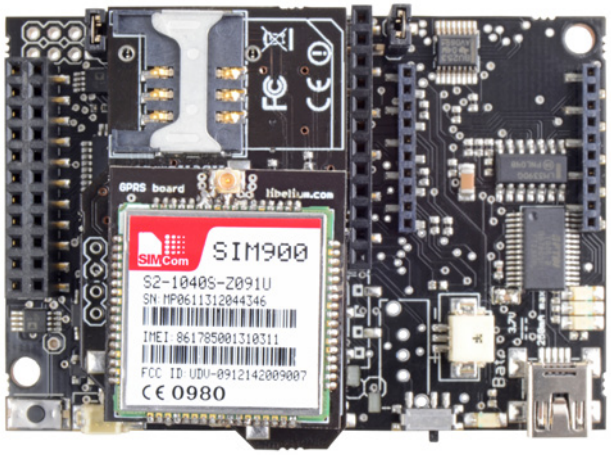


Figure 93: GSM/GPRS module in Waspmote

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 **Support** and **Development sections** in the Libelium website.

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

12. 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.

Figure 94: Bluetooth module for device discovery

Features:

- 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

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 **Support** and **Development sections** 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 "Smartphones Detection" section in the Meshlium website: http://www.libelium.com/meshlium

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.

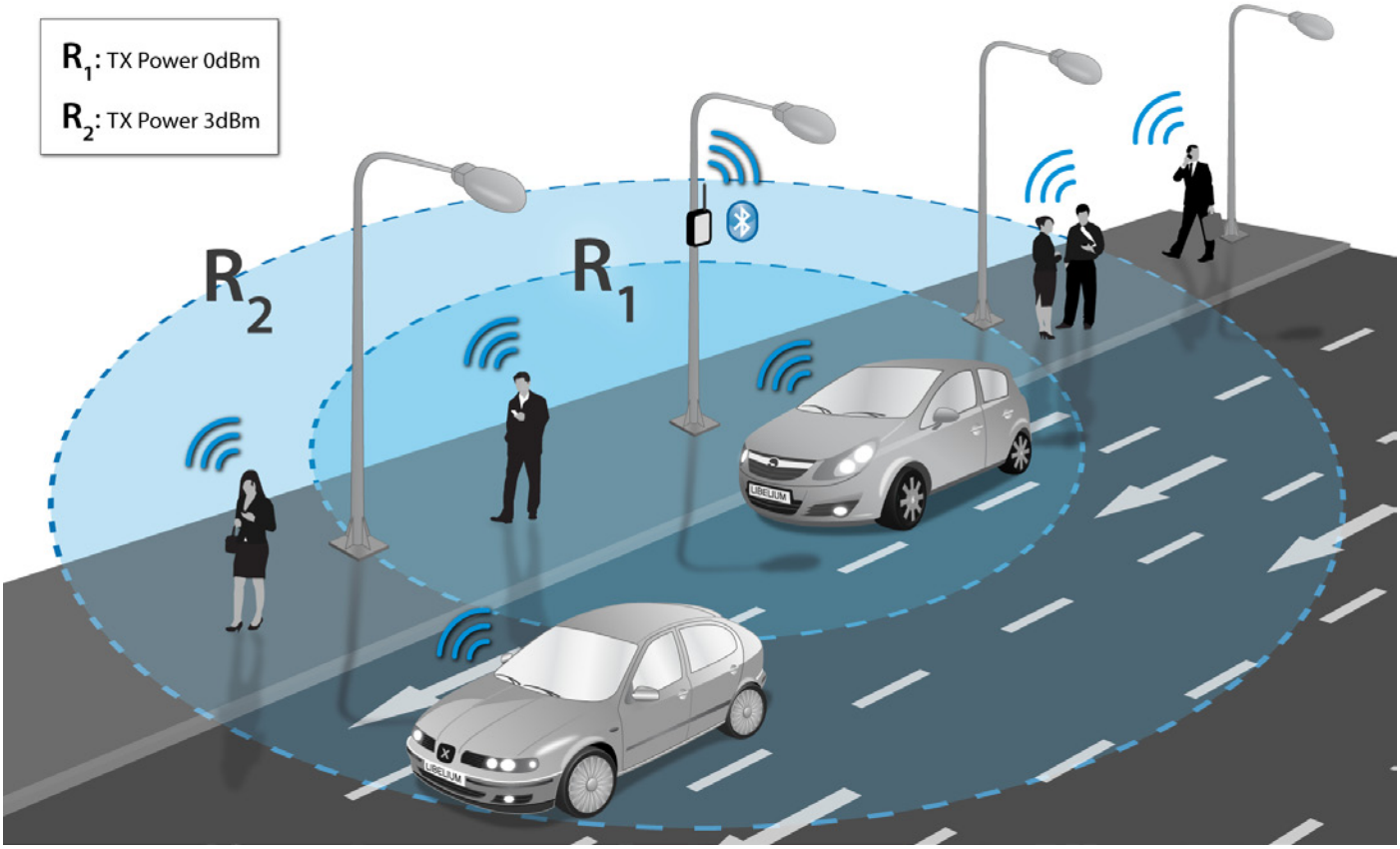


Figure 95: Example of TX power levels

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 9 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.

13. RFID/NFC

13.56MHz

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

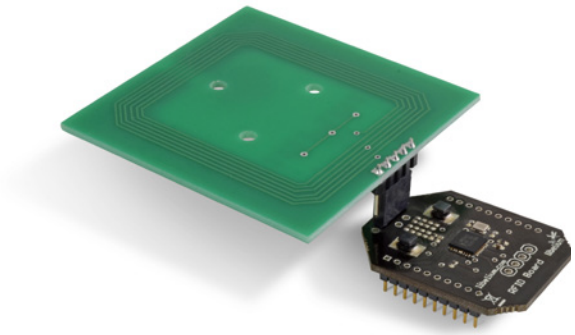


Figure 96: 13.56MHz RFID/NFC module

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

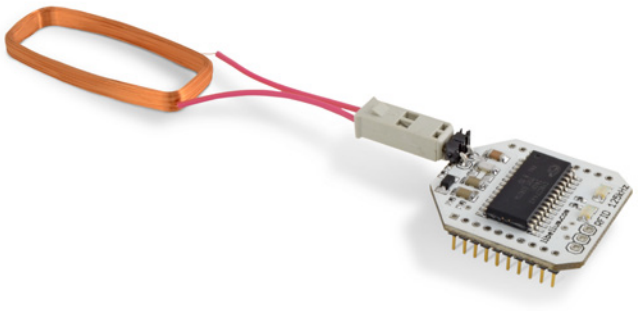


Figure 97: 125KHz RFID module

Applications:

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

Related API libraries: **WaspRFID13.cpp** , **WaspRFID13.h**, **WaspRFID13Constants.h** / **WaspRFID125.cpp**, **WaspRFID125.h**

All information on their programming can be found in documents: **RFID 1356MHz Networking Guide** and **RFID 125KHz Networking Guide**.

All the documentation is located in the **Support** and **Development sections** in the Libelium website.

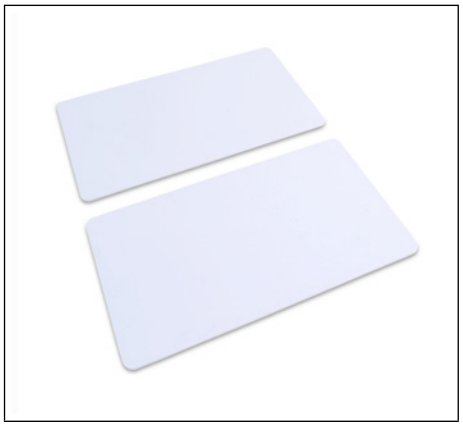
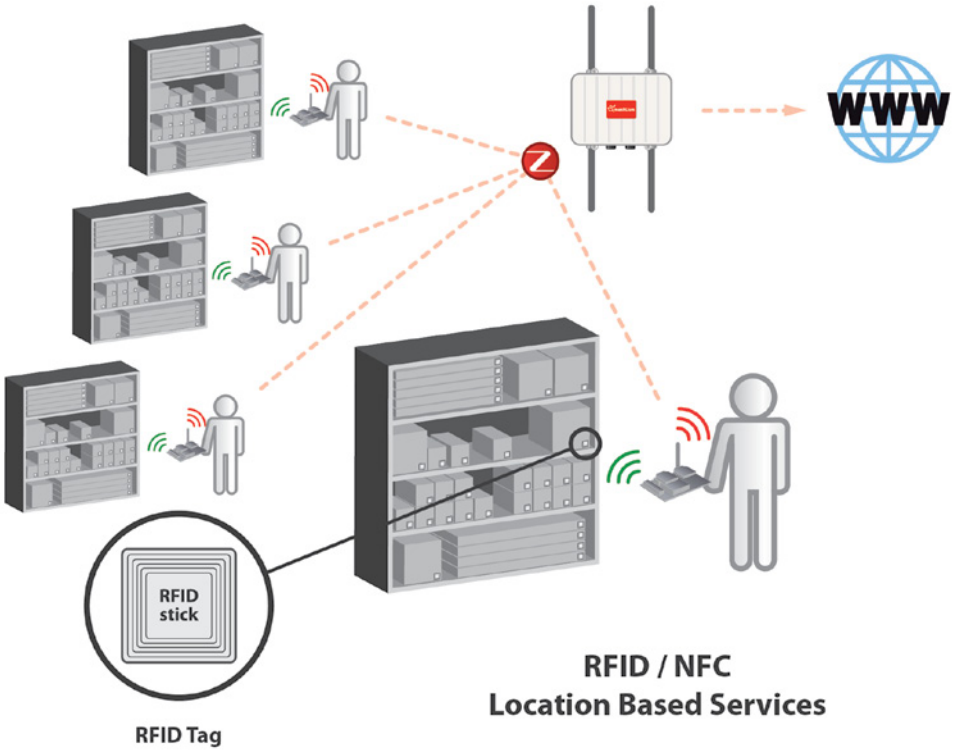
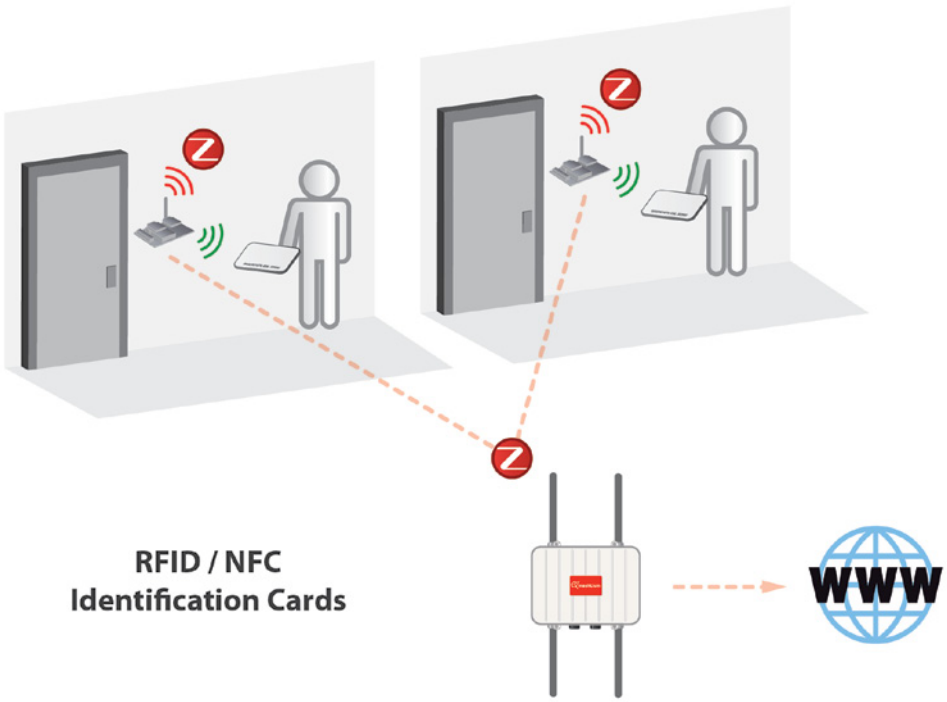


Figure 98: RFID cards

Figure 99: RFID keyrings

Figure 100: RFID sticker


WWW
RFID
stick
RFID Tag
RFID / NFC
Location Based Services


RFID / NFC
Identification Cards
WWW

14. 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 six radios available for Waspmote: **802.15.4**, **ZigBee**, **Bluetooth**, **RFID**, **Wifi**, **3G/GPRS**, **868** and **900**.

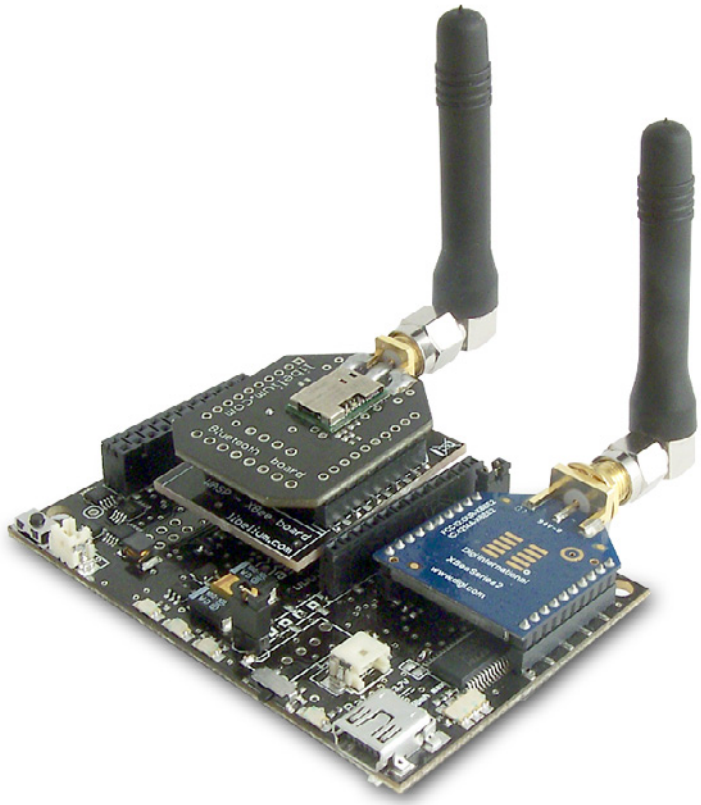


Figure 101: Expansion Radio Board

Some of the possible combinations are:

- ZigBee - Bluetooth
- ZigBee - RFID
- ZigBee - Wifi
- ZigBee - 3G/GPRS
- Bluetooth - RFID
- RFID - 3G/GPRS
- etc.

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

Applications:

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

15. Over the Air Programming (OTA)

15.1. Overview

The concept of Wireless Programming or commonly know as Programming Over the Air (OTA) has been used in the past years overall for the reprogrammation 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.

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)

15.2. Benefits

Libelium OTA Benefits:

- Enables the upgrade or change of firmware versions without physical access
- Enables to recover to any sensor node which gets stuck
- Discover nodes in the area just sending a broadcast discovery query
- Upload new firmware in just a couple of 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

15.3. Concepts

The idea is simple. When the programmer (normally the Gateway) sends a new program it is stored in the SD card. A second command "star_new_firmware" 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)
- 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.

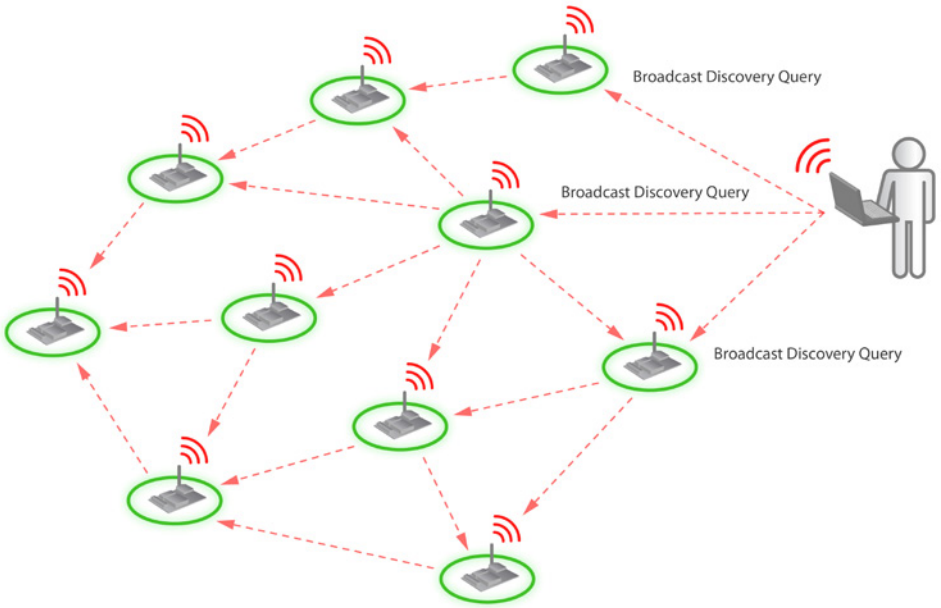
OTA-Shell:

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.

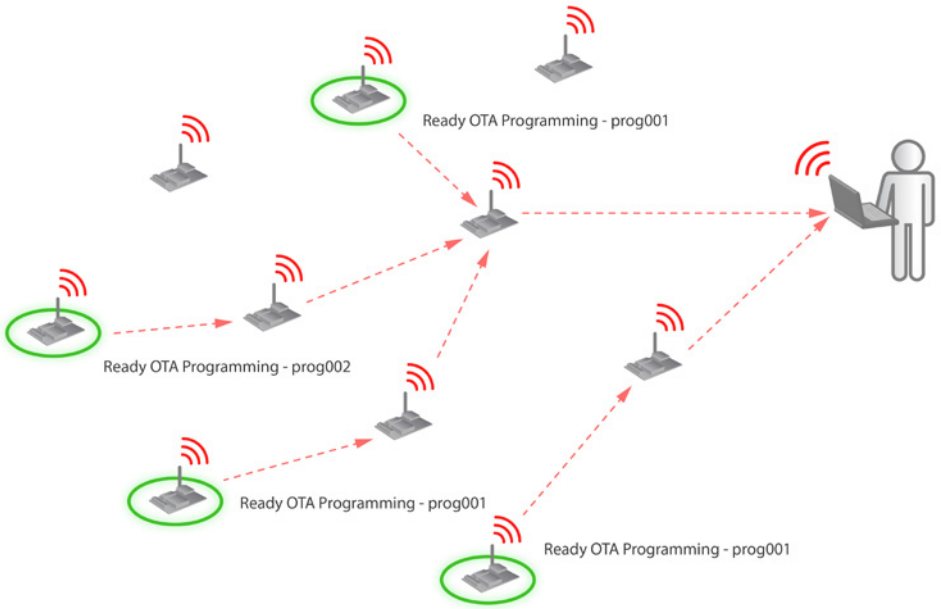
15.4. OTA Step by Step

- **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.

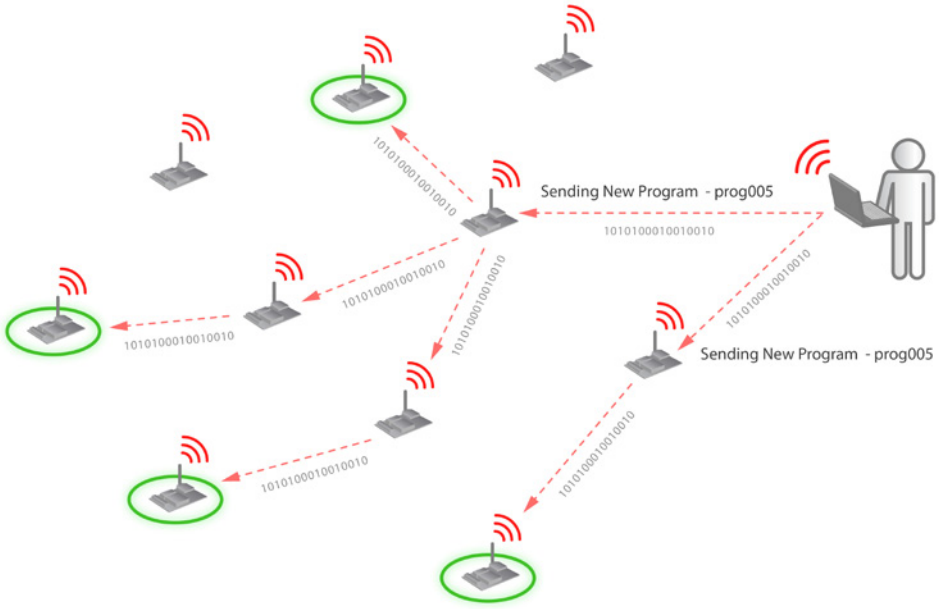


The nodes which are ready at this moment will answer with a "Ready to OTA" frame.

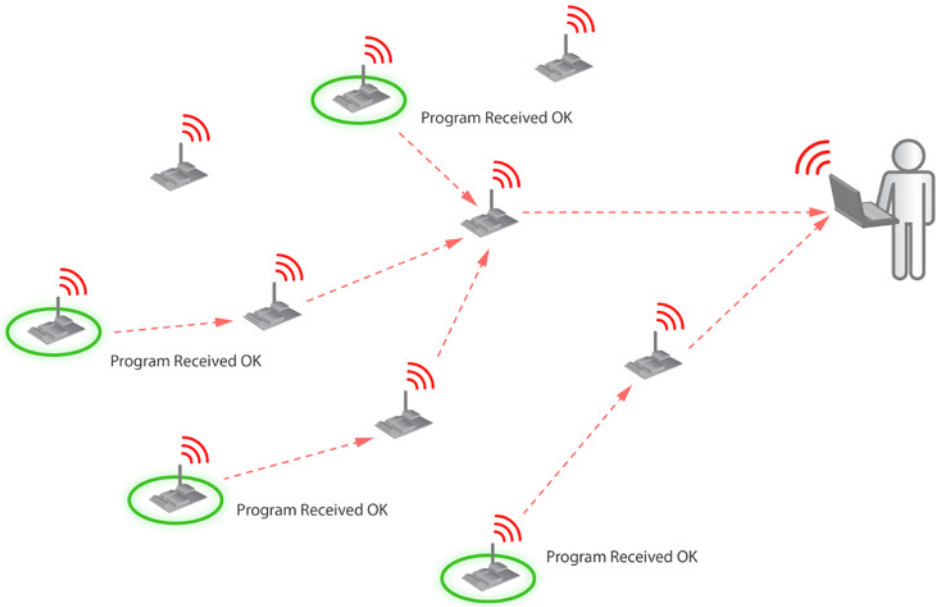


- **Send the new program**

  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.

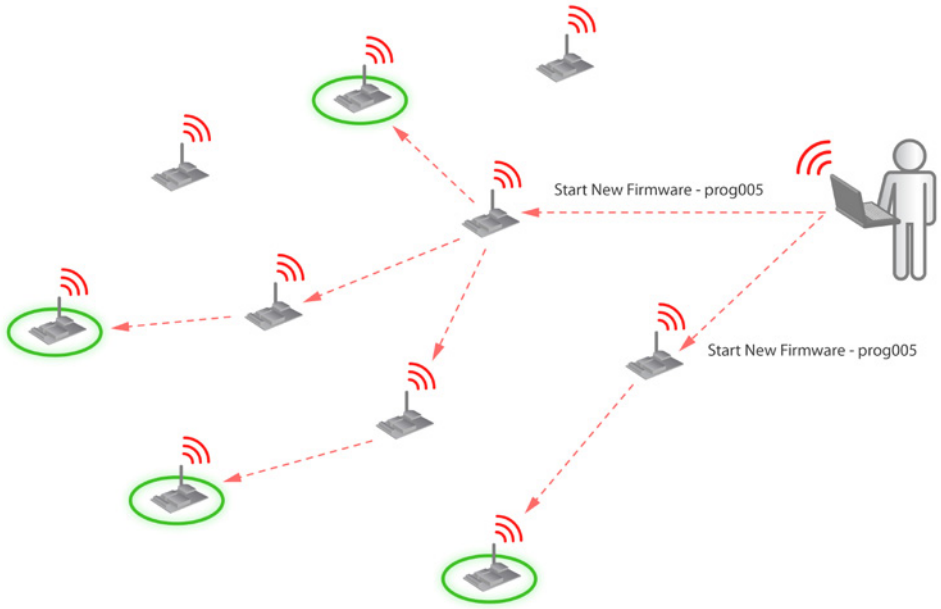


Each node which receives the program sends a message to the gateway to inform of the success of the process.



- **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.



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

15.5. 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 ZigBee Gateway of the sensor network. See chapter 20.2), 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 Support and Development sections 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 Support and Development sections:

http://www.libelium.com/support/waspmote

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

16. GPS

Waspmote can integrate a GPS receiver which allows to know the exact outside location of the mote anytime. In this way 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:** A1084 (Vincotech)
- **Movement sensitivity:** -159dBm
- **Acquisition sensitivity:** -142dBm
- **Hot Start time:** <1s
- **Warm Start Time:** <32s
- **Cold Start Time:** <35s
- **Antenna connector:** UFL
- **External Antenna:** 26dBi

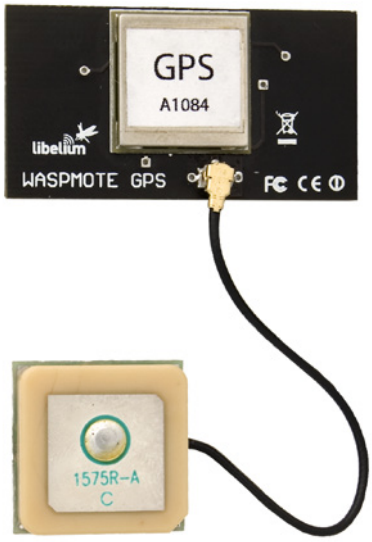


Figure 102: GPS module

The GPS module gives us information about:

- latitude
- longitude
- height
- speed
- direction
- date/time
- ephemerids

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 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.

In the GPS receiver libraries both modes have been used, based on the use of the NMEA statements to obtain the necessary data regarding the **position, date and time**.

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.
- NEMA 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)

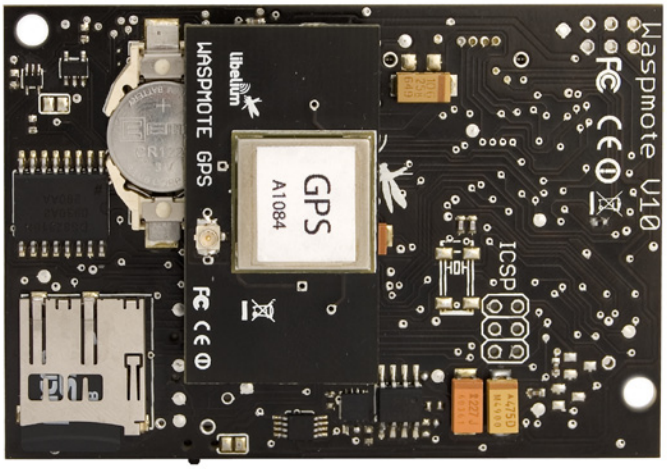


Figure 103: GPS module connected to Waspmote

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 ephemerids. The information that can be found out is relative to the current position of the satellites (ephemerids) 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 ephemerids 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 several types:

- Hot Start: once the time and date are established and the **ephemerids** and valid almanacs are in the memory. Time: **<1s**
- Warm Start: once the time and date are established and the valid **almanacs** are in the memory. Time: **<32s**
- 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 **ephemerids are stored**. For this reason a series of functions have been created in the libraries to **store ephemerids 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 **Support** and **Development sections** in the Libelium website.

17. 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.

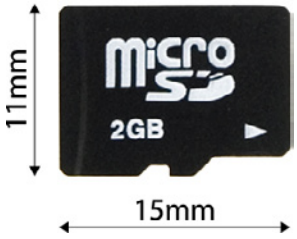


Figure 104: Micro-SD card

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.

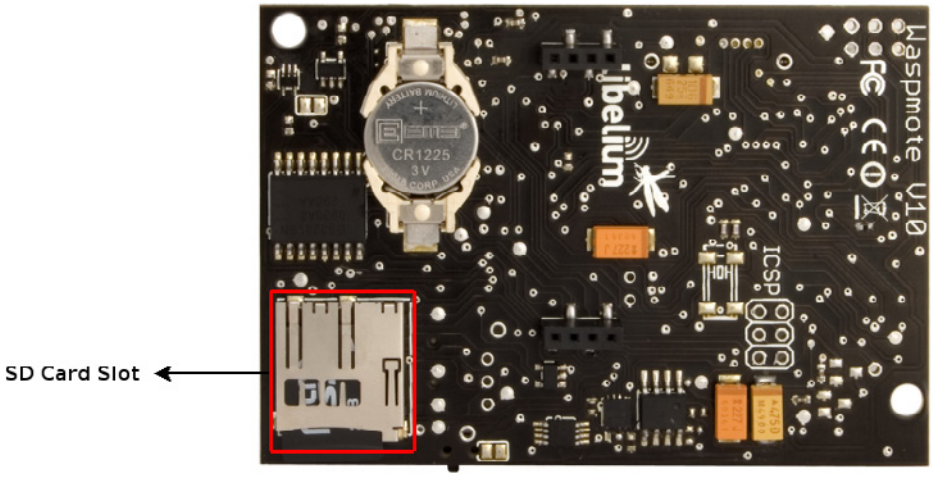


Figure 105: SD Card slot

To communicate with the SD module we use the **SPI** bus. This bus is a communications 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.ls();
  SD.openFile(“data.txt”);
}
```

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 **Support** and **Development sections** 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.*

18. Energy Consumption

18.1. Consumption tables

Waspmote

ON	9mA
Sleep	62µA
Deep Sleep	62µA
Hibernate	0,7µA

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
XBee XSC	48,85mA	0,48mA	0µA	48,35mA	110mA

RSSI(LEDs): 7,6mA

Bluetooth Module

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

GPS

ON	27,5mA
OFF (Waspmote switch)	0µA

GSM - 3G/GPRS

Connecting	~100mA
Calling	~100mA
Receiving Calls	~100mA
Transmitting GPRS	~100mA
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

19. Power supplies

19.1. 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: 1150mA, 2300mA and 6600mA Li-Ion rechargeable and 13000mAH **non - rechargeable.**

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

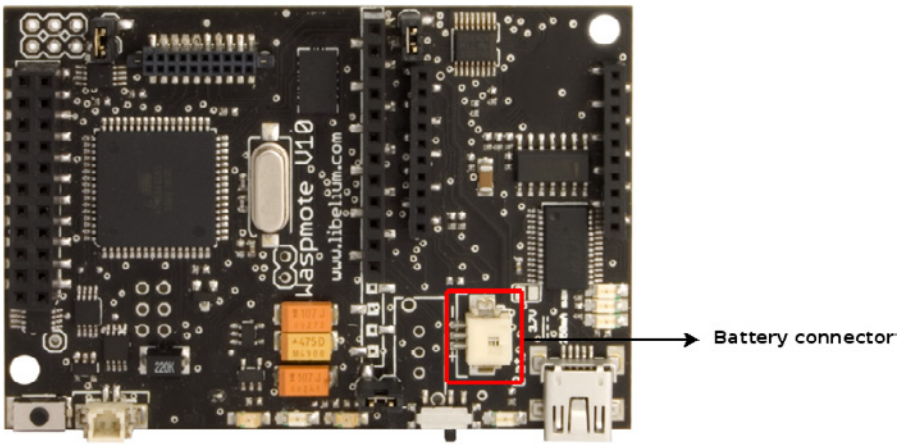


Figure 106: Battery connector

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).

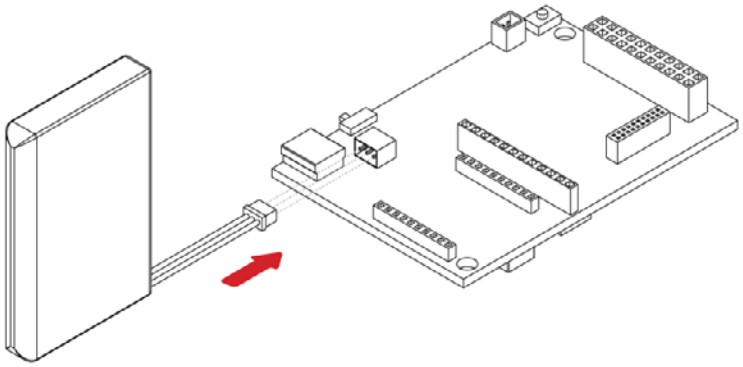


Figure 107: Diagram of the battery connection

Battery discharging and charging curves

The following two images show battery discharging and charging curves.

Battery discharging

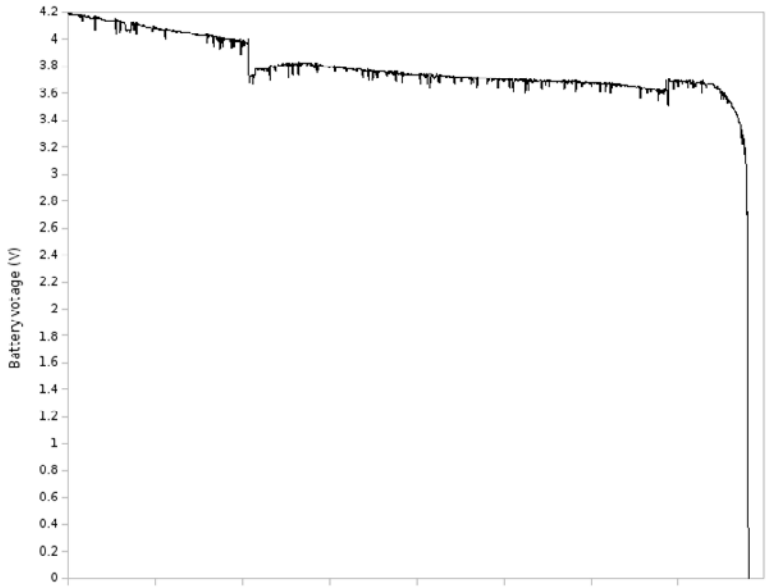


Figure 108: Typical discharging curve for battery

Battery charging using USB

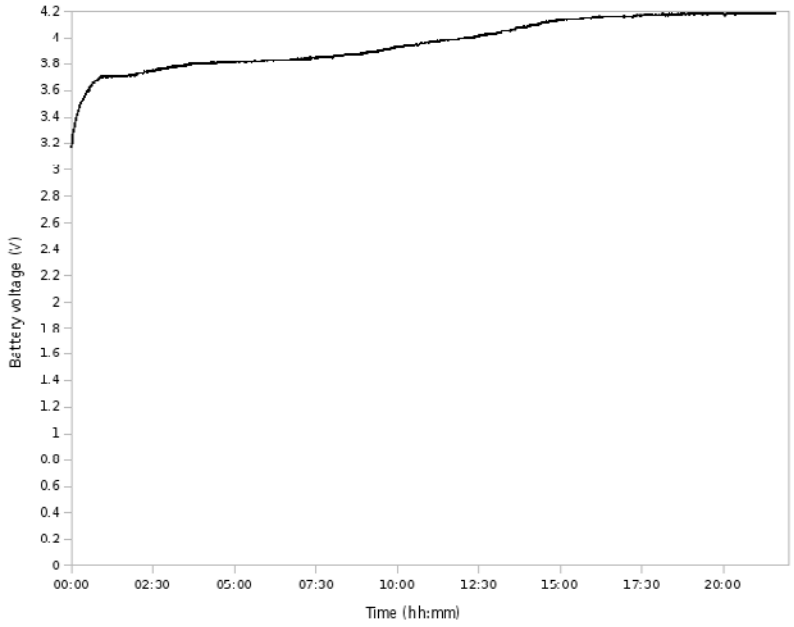


Figure 109: Typical charging curve for battery

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.

19.2. 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**.

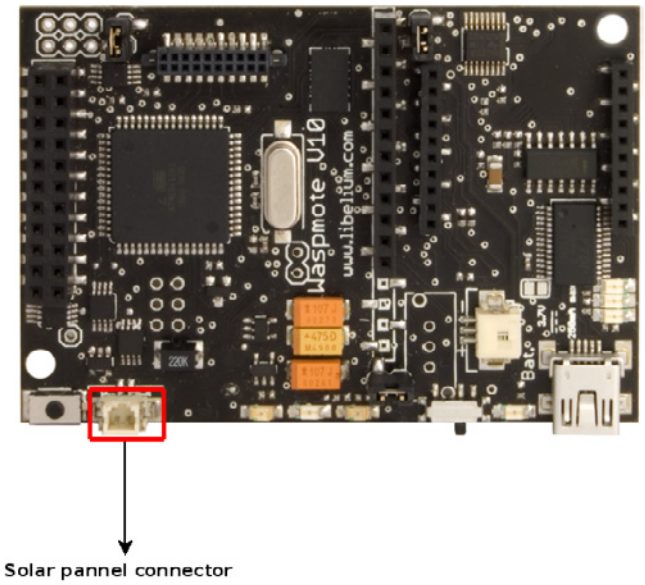


Figure 110: Solar panel connector

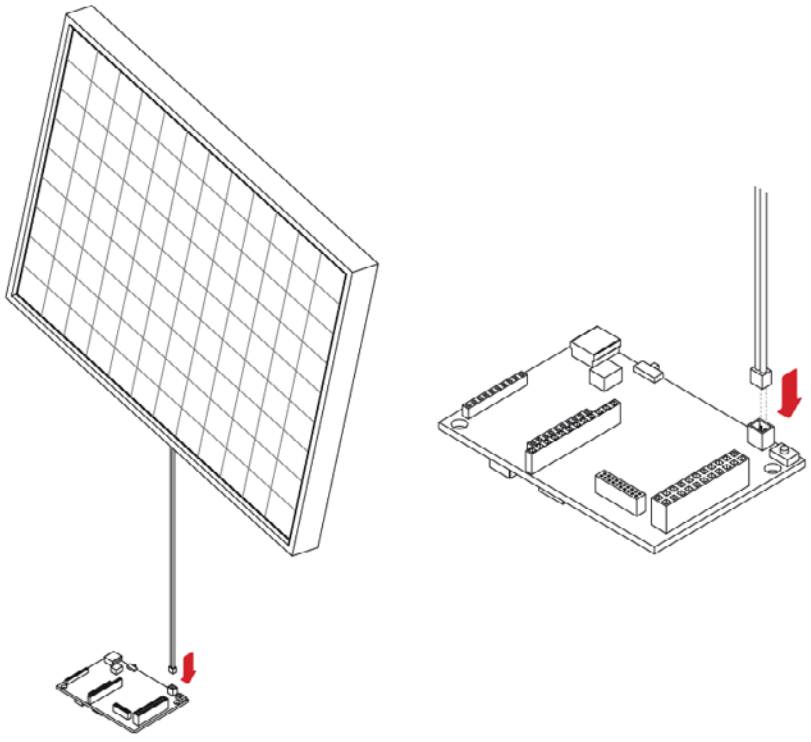


Figure 111: Diagram of solar panel connection

The models supplied by Libelium are shown below:

- **Rigid Solar Panel**

  7V - 500mA

Figure 112: Rigid Solar Panel

- **Flexible Solar Panel**

  7.2V - 100mA

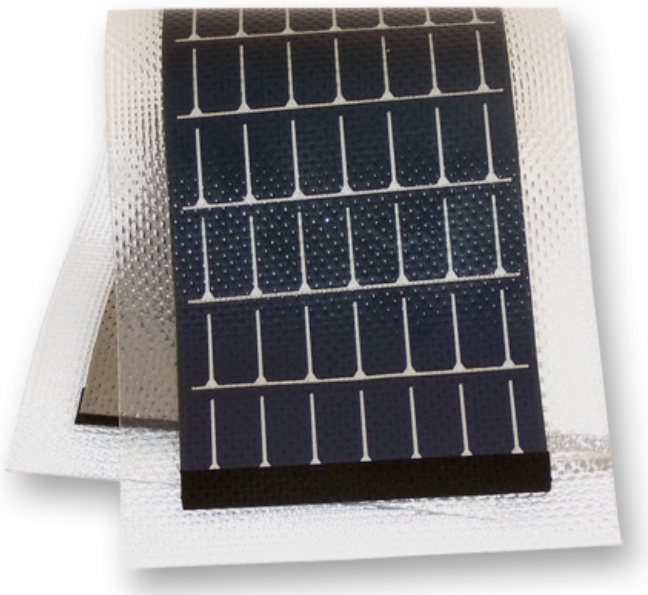


Figure 113: Flexible Solar Panel

19.3. USB

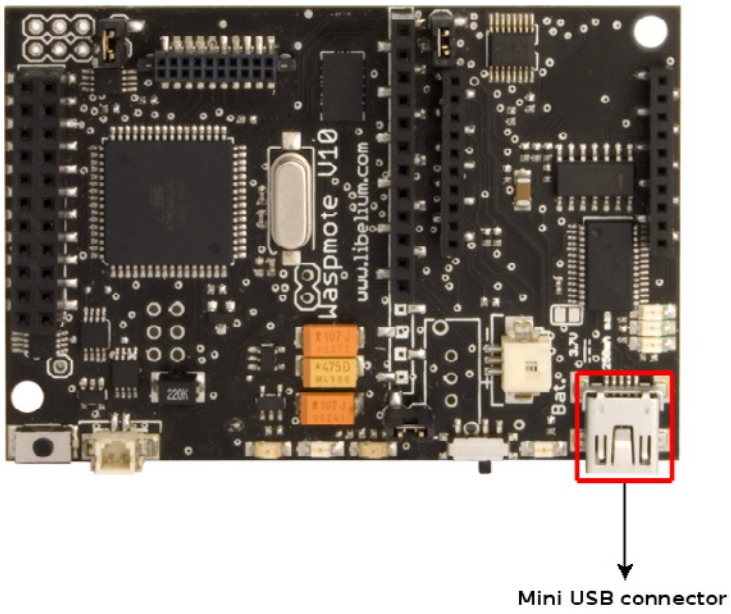


Figure 114: Mini USB connector

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.

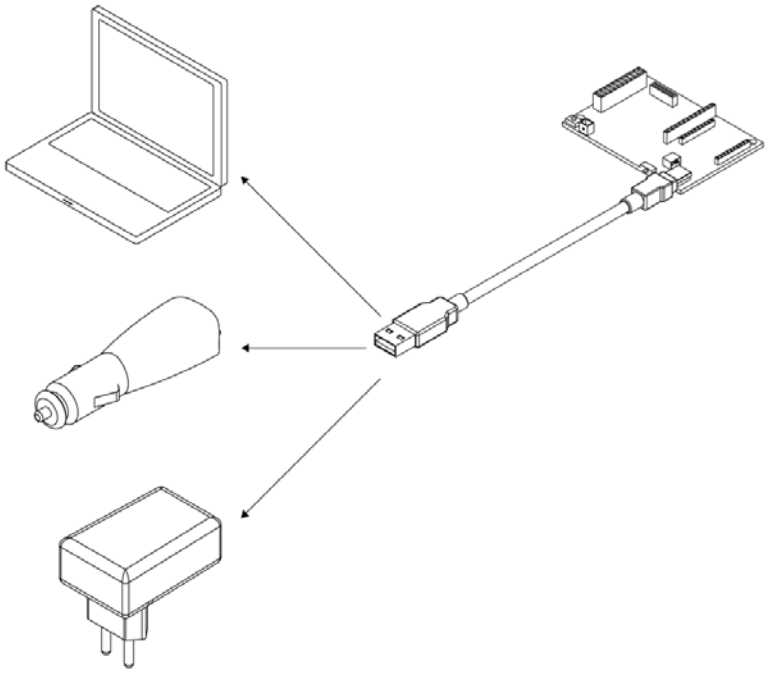


Figure 115: Possible connections for the USB

The models supplied by Libelium are shown below:

Figure 116: 220V AC – USB adapter

Figure 117: 12V DC – USB car lighter adapter

20. 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 and proven and for which we can assure optimum operation.

The Waspmote-IDE includes all the API libraries necessary to compile the programs.

The file which contains the compiler and the libraries is called "Waspmote-environment-vxx.zip" (xx corresponds to the version name). 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.

In order to update to future library versions, the "WaspV10" folder must be modified within the "hardware/cores" folder found inside the previously unzipped folder.

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.

20.1. Linux

Waspmote can be programmed using any of the many available Linux distributions. Libelium ensures stability of the Waspmote-IDE using the Debian and Ubuntu distributions, as they have been used during Waspmote development. The process for Ubuntu 9.04 is explained below.

To be able to correctly compile and use Waspmote it is necessary to install some packets related with the version of the compiler for Atmel microcontrollers and Java environment.

1.- Installing Java

The first step is to install the necessary version of the **Java** environment. We can use the Synaptic package manager or a terminal.

Using the Synaptic package manager, we must look for the "sun-java6-jre" package and install it.

Using the terminal we must use the apt-get command in the following way:

```
$ sudo apt-get install sun-java6-jre
```

2.- Installing AVR-GCC Compiler

The next step is to install the necessary version of the **avr-gcc** compiler to be able to program the Waspmote ATMEGA 1281 microcontroller. We can use the Synaptic package manager or a terminal.

Using Synaptic we must look for the "gcc-avr" package and install it. Using the terminal we must use the apt-get command in the following way:

```
$ sudo apt-get install gcc-avr
```

3.- Installing lib-avc Library

The next step is to install the necessary version of the **lib-avc library**. We can use the Synaptic package manager or a terminal.

Using Synaptic we must look for the "lib-avc" package and install it. Using the terminal we must use the apt-get command in the following way:

```
$ sudo apt-get install avr-libc
```

4.- Installing Waspmote

Waspmote installation entails unzipping the file downloaded in the previous step to the chosen folder. Once the downloaded file has been unzipped, the file called **Waspmote** must be run to launch the IDE.

20.2. Windows

The process for installing Waspmote-IDE on a computer with Window XP SP3 is explained below.

1.- Downloading Waspmote

The first step is to download the file that contains the environment and the libraries. This file can be found at:

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

2.- Installing Waspmote

The next step is to unzip the downloaded file to the chosen folder. This folder includes the drivers needed in the next step to install the USB and FTDI converter.

3.- Connecting a Waspmote board

When connecting a Waspmote board using the mini-USB connector, the message "New device found" will appear. A window will open for the installation of this device.

Select the option "Not right now" and press the 'Next' button.

Next select the path where the drivers for the FTDI converter are. These drivers are in the folder where Waspmote was unzipped.

Then proceed to the installation of the FTDI converter drivers, which shows the following message when finished.

Once installation is finished, the message 'New device found' will appear, referring to the USB. The same process carried out for the FTDI converter must now be followed, choosing the same options in all the windows. The path for the drivers is the same as that previously specified.

Once this installation is finished, a message will appear indicating the correct installation of the USB.

Once both devices are correctly installed, the port on which the Waspmote board has been installed will appear in the "Device Administrator".

20.3. Mac-OS

Waspmote can be programmed using the Mac OS X operating system. The process of install Waspmote-IDE on a computer with a version later than 10.3.9 is explained below.

1.- Downloading Waspmote

The first step is to download the file that contains the environment and libraries. This file can be found on our web page:

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

2.- Installing Waspmote

The next step is to unzip the downloaded file to the chosen folder. The drivers needed in the next step to install the FTDI converter are found in this folder.

3.- Installing FTDI converter drivers

Waspmote requires the installation of the FTDI converter drivers. These drivers are found in the downloaded file.

Once the drivers are installed for the FTDI converter, the Waspmote board can be connected and the system will recognise it correctly.

20.4. First steps

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

Steps:

1. Install the drivers and the serial monitor software on the computer (chapters 19.1, 19.2 and 19.3).
2. Connect the antennas and the rest of the desired components to Waspmote and Waspmote Gateway (chapter 1.4).
3. Plug Waspmote Gateway to the USB port on the computer (chapter 20.1).
4. Launch the serial monitor application and set the next parameters:
   - USB port:38000bps
   - 8bits
   - 1 bit stop
   - no parity setting

5. Connect the batteries to the Waspmotes.
6. Switch Waspmotes to the ON position (chapter 1.4).

When the program starts, it executes sequentially these actions:

- State 1 – Leds ON for 5 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 the MAC address of the XBee module. Example:

~\0x00U\0x80\0x00}3\0xa2\0x00@U3\0xdf=\0x02R\0x01#\0x01\0x00}3\0xa2\0x00@U3\0xdf
--hello, this is Waspmote. MAC address: 0013A200405533DF--\0xc1

Initially there are some hexadecimal characters followed by the message. In the above example the message is

--hello, this is Waspmote. MAC address: 0013A200405533DF--

Where 0013A200405533DF matches with the 64 bits MAC address of the XBee module included in the Waspmote which sent the packet.

The rest of the characters match with the parameters included in the application header established in the WaspmoteXBee library (for extended information see the section 20.6). These parameters are sent in hexadecimal format.

These hexadecimal parameters are used by the Waspmote devices for making control tasks in the network layer. When we make an application for Waspmote these information is extracted and analyzed by each node and only the real message is sent to the next node in the network. However, in this case the Waspmote Gateway shows the whole content in the packet received.

In chapter 15.1.2 and 15.2.1 you can see how to create applications to extract this information and use it in a data base, to trigger alarms and so on.

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

20.5. 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.

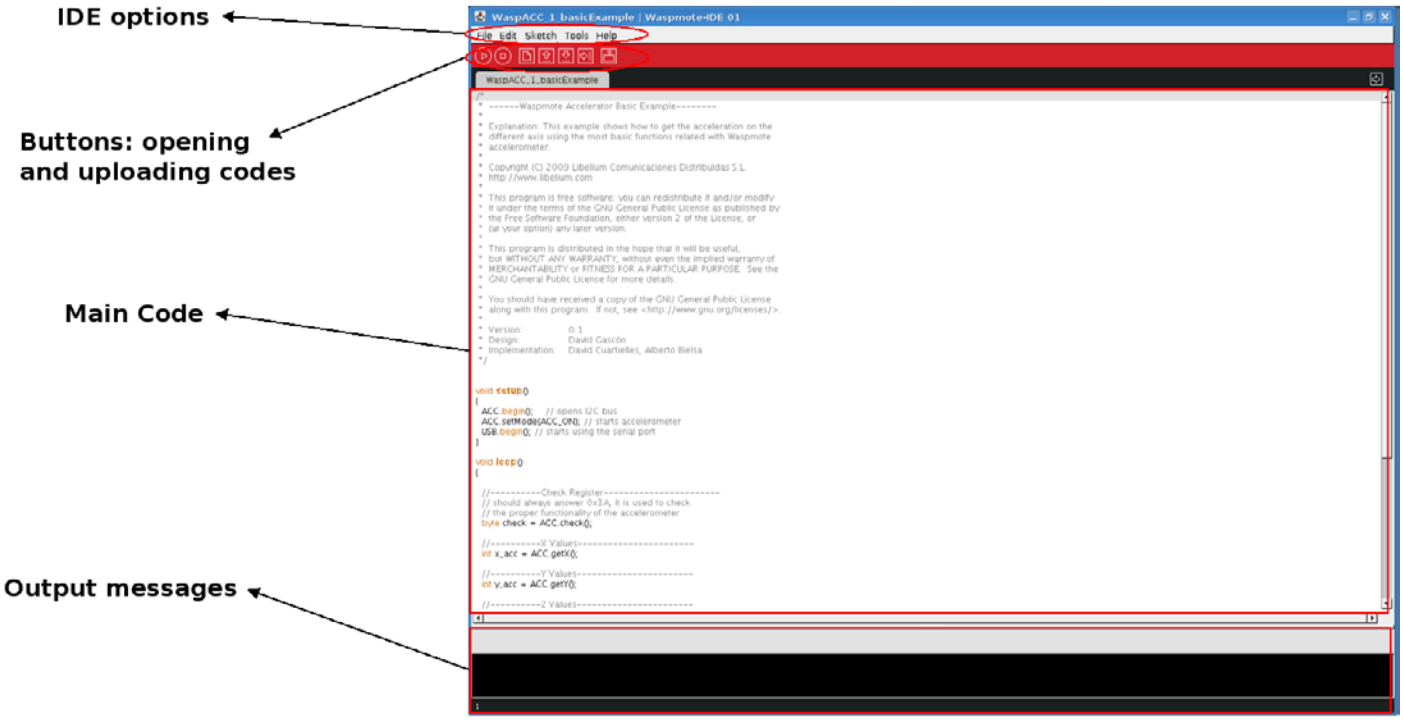


Figure 118: IDE – Waspmote parts

- 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 compilation, opening, saving or loading the selected code on the board.
- The third part contains the main code which will load in Waspmote and 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.

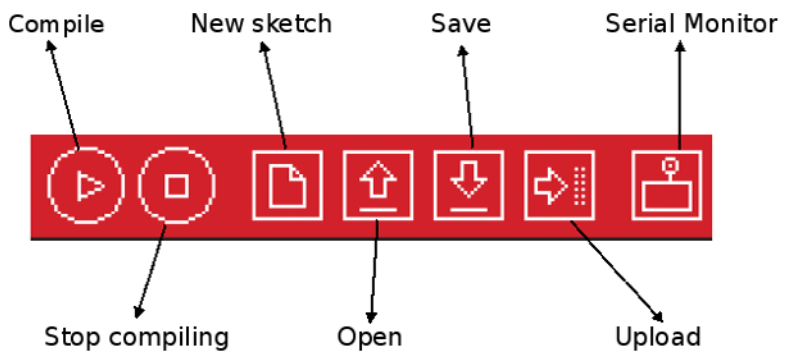


Figure 119: IDE – Waspmote panel of buttons

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. To do this the option 'Wasp v01 must be 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 familiarising yourself with the Waspmote environment have been included in the downloaded file that contains the compiler.

The simplest example is the file called 'prueba.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 'pruebe.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 an LED every 10 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'** clicked. When this button is clicked, it will start to compile 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 chapter 2.

***Note 1:** XBee module has to be removed before uploading a program to Waspmote.*

***Note 2:** If Waspmote is connected to a computer (USB connection), 3G/GPRS has to be connected in a different way. You have to add to your code the following lines and connect the 3G/GPRS module after the first blinking and before the second one.*

Code:

```
        delay(1000);
    Utils.blinkLEDs(500);


// Here you have to connect the module


    delay(5000);
    Utils.blinkLEDs(500);
    Utils.blinkLEDs(500);
```

***Note 3:** 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.*

20.6. 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:

- **General configuration:** WaspClasses.h, WaspVariables.h, WaspConstants.h, Wconstants.h, pins_arduino.h, pins_arduino.c, utils.h, utils.cpp, WProgram.h, Wrandom.cpp
- **Shared:** binary.h, byteordering.h, byteordering.cpp, 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
- **SD Storage:** fat_config.h, fat.h, fat.cpp, partition_config.h, partition.h, partition.cpp, sd_raw_config.h, sd_raw.h, sd_raw.cpp, sd_reader_config.h, WaspSD.h, WaspSD.cpp
- **I2C Communication:** twi.h, twi.c, wire.h, wire.cpp
- **Accelzrometer:** WaspACC.h, WaspACC.cpp
- **GSM - 3G/GPRS:** WaspGPRS.h, WaspGPRS.cpp, WaspGPRSConstants.h
- **GPS:** WaspGPS.h, WaspGPS.cpp
- **Energy Control:** WaspPWR.h, WaspPWR.cpp
- **RTC:** WaspRTC.h, WaspRTC.cpp
- **Sensors:** WaspSensorEvent.h, WaspSensorEvent.cpp, WaspSensorGas.h, WaspSensorGas.cpp, WaspSensorPrototyping.h, WaspSensorPrototyping.cpp
- **USB:** WaspUSB.h, WaspUSB.cpp
- **XBee:** WaspXBee.h, WaspXBee.cpp, WaspXBeeConstants.h, WaspXBeeCore.h, WaspXBeeCore.cpp, WaspXBee802.h, WaspXBee802.cpp, WaspXBeeZB.h, WaspXBeeZB.cpp, WaspXBeeDM.h, WaspXBeeDM.cpp, WaspXBee868.h, WaspXBee868.cpp, WaspXBeeXSC.h, WaspXBeeXSC.cpp
- **Interruptions:** Winterruptions.c

General configuration

Files: WaspClasses.h, WaspVariables.h, WaspConstants.h, Wconstants.h, pins_arduino.h, pins_arduino.c, utils.h, utils.cpp, WProgram.h, Wrandom.cpp

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_arduino.h, pins_arduino.c: the microcontroller's pins and the names to which they are associated are defined.
6. utils.h, utils.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.
8. Wrandom.cpp: functions related to random number generation.

Shared

Files: binary.h, byteordering.h, byteordering.cpp, 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: fat_config.h, fat.h, fat.cpp, partition_config.h, partition.h, partition.cpp, sd_raw_config.h, sd_raw.h, sd_raw.cpp, sd_reader_config.h, WaspSD.h, WaspSD.cpp

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

- *fat_config.h, fat.h, fat.cpp, partition_config.h, partition.h, partition.cpp, sd_raw_config.h, sd_raw.h, sd_raw.cpp, sd_reader_config.h:* 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.

GSM - 3G/GPRS

Files: WaspGPRS.h, WaspGPRS.cpp, WaspGPRSConstants.h

The functions needed for receiving and sending calls, sms or data using the GSM - 3G/GPRS network.

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.

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.

Sensors

Files: WaspSensorEvent.h, WaspSensorEvent.cpp, WaspSensorGas.h, WaspSensorGas.cpp, WaspSensorPrototyping.h, WaspSensorPrototyping.cpp

The functions needed to manage the different sensor boards available on Waspmote.

USB

Files: WaspUSB.h, WaspUSB.cpp

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

XBee

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

1. *WaspXBee.h, WaspXBee.cpp:* the functions needed to control the XBee power switch, as well as to send data directly to the

UART where the XBee module is connected.

2. *WaspXBeeConstants.h:* the constants used in the libraries related to the XBee modules.
3. *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.
4. *WaspXBee802.h, WaspXBee802.cpp:* the specific functions of the XBee 802.15.4 and the shared general library functions are inherited.
5. *WaspXBeeZB.h, WaspXBeeZB.cpp:* the specific functions of the XBee ZigBee modules are defined and the shared general library functions are inherited.
6. *WaspXBeeDM.h, WaspXBeeDM.cpp:* the specific functions of the XBee DigiMesh and 900MHz are defined, and the shared general library functions are inherited.
7. *WaspXBee868.h, WaspXBee868.cpp:* the specific functions of the XB 868MHz modules are defined and the shared general library functions are inherited.
8. *WaspXBeeXSC.h, WaspXBeeXSC.cpp:* the specific functions of the XBee XSC are defined and the shared general library functions are inherited.

Interruptions

Files: Winterruptions.c

The functions needed for interrptions 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.

20.7. 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

These new versions are downloaded in a file similar to "waspmote-api-v0xx.zip" (xx being the current version). This file contains a folder which must substitute the "WaspV10" folder that is currently found in the Waspmote compiler installation.

Once this folder is replaced, the libraries will be updated to the new version.

21. Interacting with Waspmote

21.1. Receiving 802.15.4/ZigBee frames with Waspmote Gateway

21.1.1. Waspmote Gateway

This device allows 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 120: 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**.

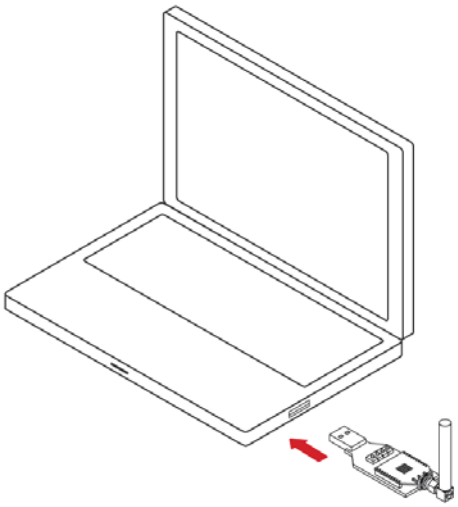


Figure 121: Waspmote Gateway connected in a PC

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.

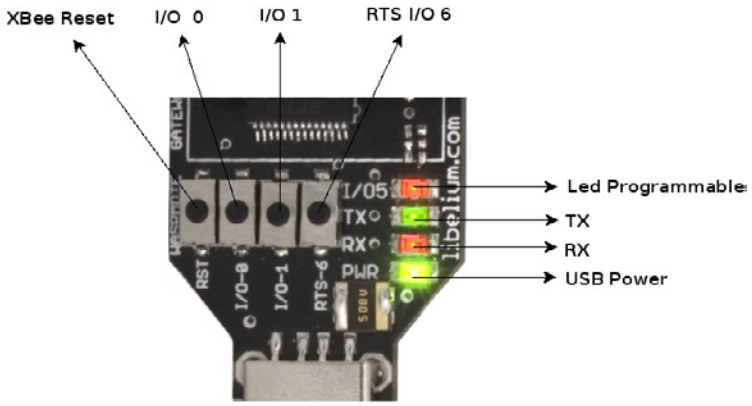


Figure 122: LEDs in Waspmote Gateway

21.1.2. 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 38400 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.

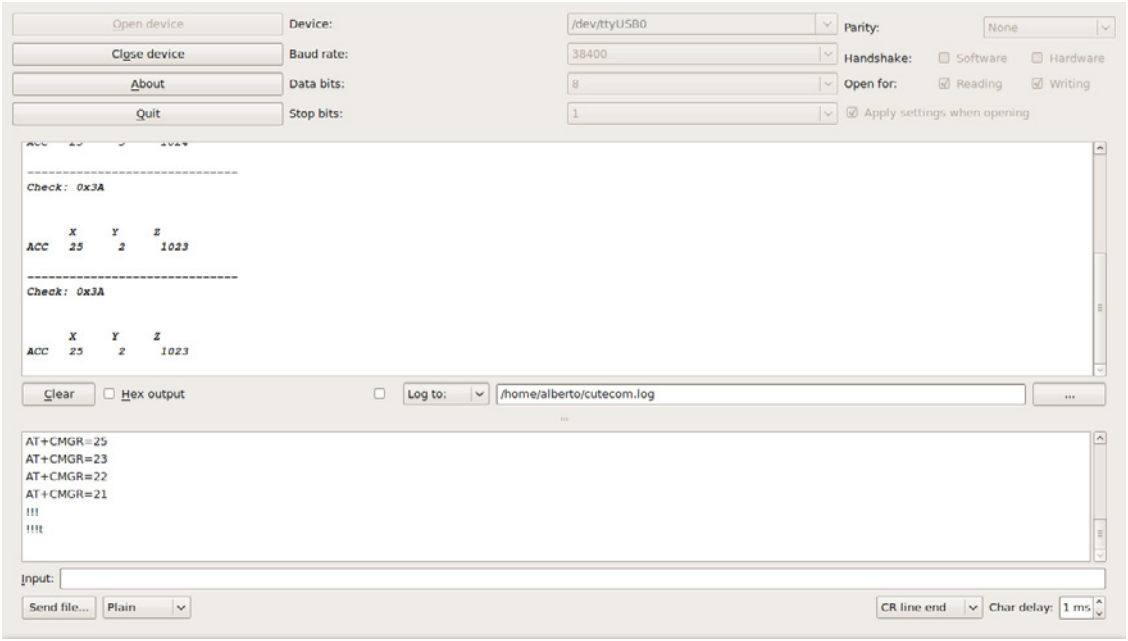


Figure 123: Cutecom application capturing Waspmote's output

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 38900).

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**

21.1.3. 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 124: Step 1 of establishing connection

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

Figure 125: Step 2 of establishing connection

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

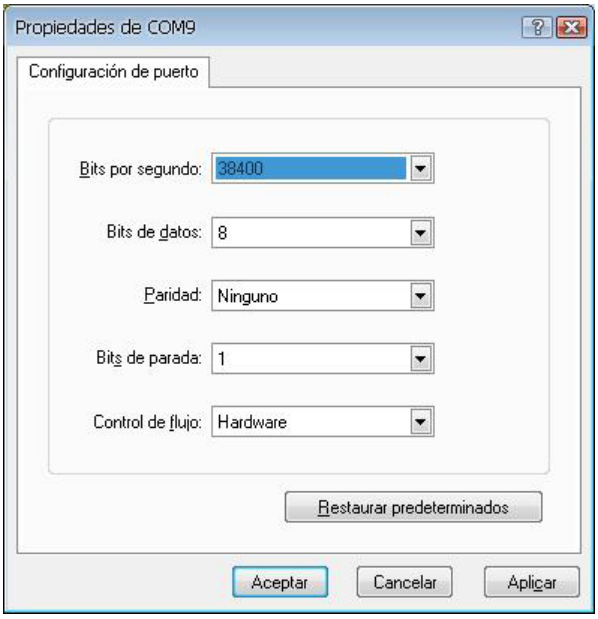


Figure 126: Step 3 of establishing connection

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

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

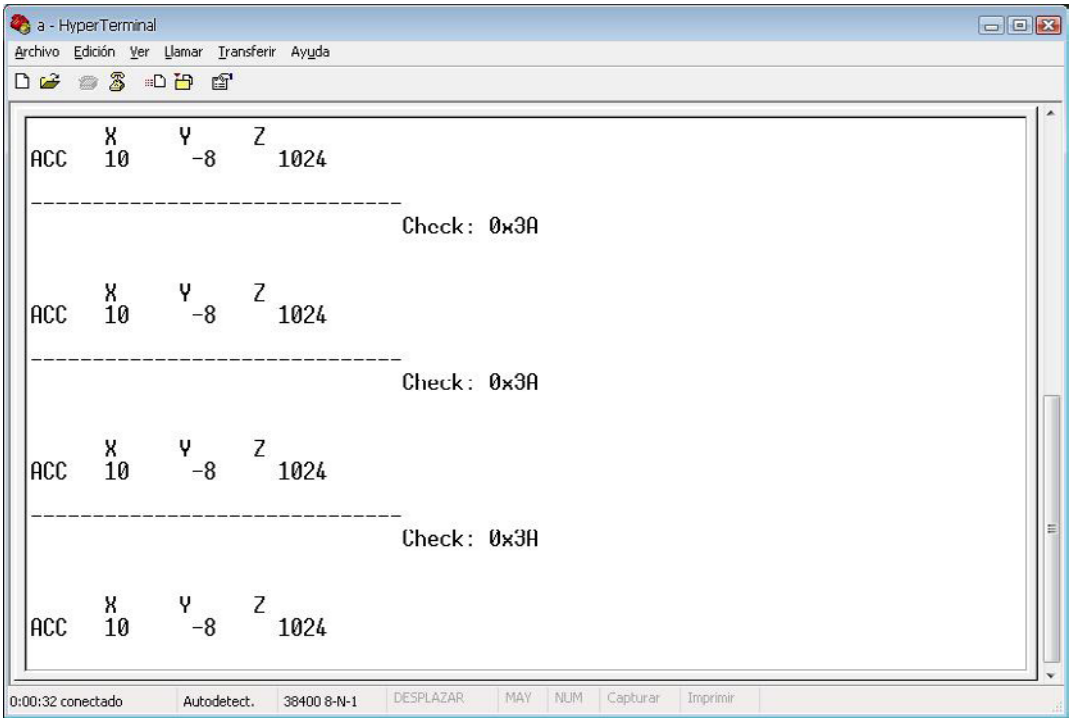


Figure 127: HyperTerminal application capturing Waspmote's output.

21.1.4. 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.

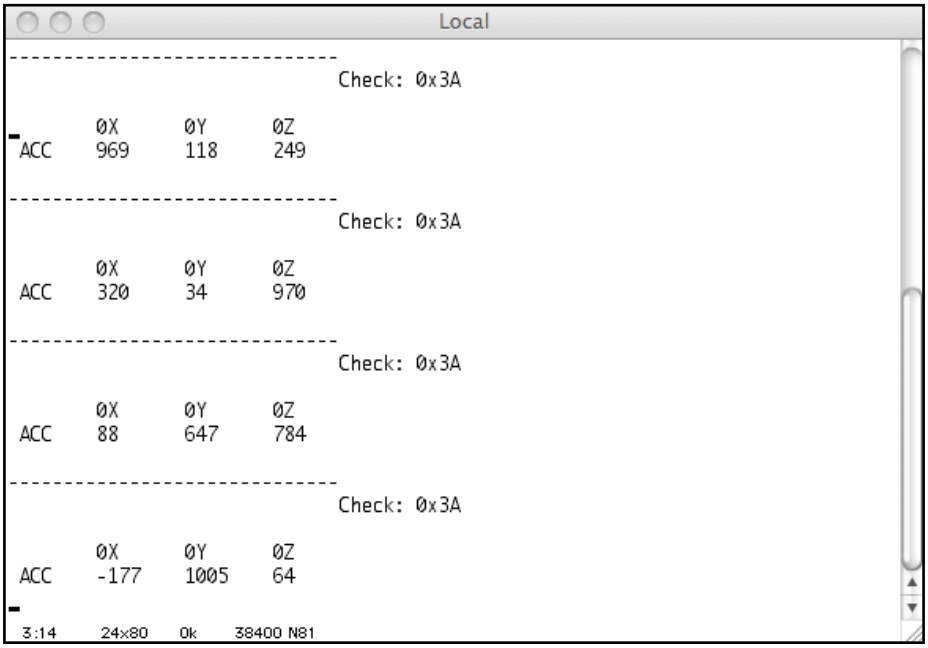


Figure 128: Waspmote's output capture

21.2. Meshlium

Figure 129: 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**

21.2.1. 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

21.2.2. How do they work together?

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

Then 5 possible actions can be performed:

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

21.2.2.1. Meshlium Storage Options

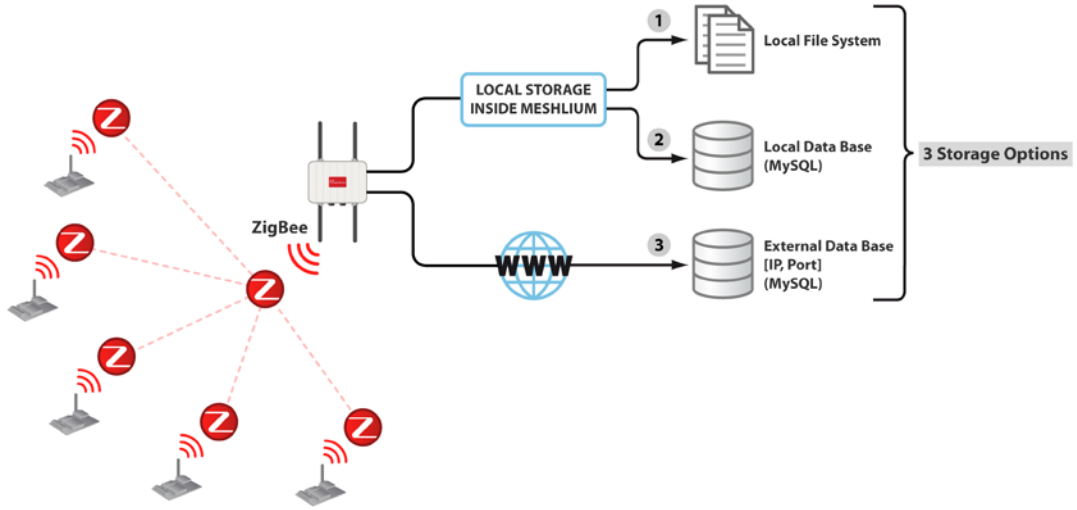


Figure 130: Meshlium Storage Options

- Local File System
- Local Data Base
- External Data Base

21.2.2.2. Meshlium Connection Options

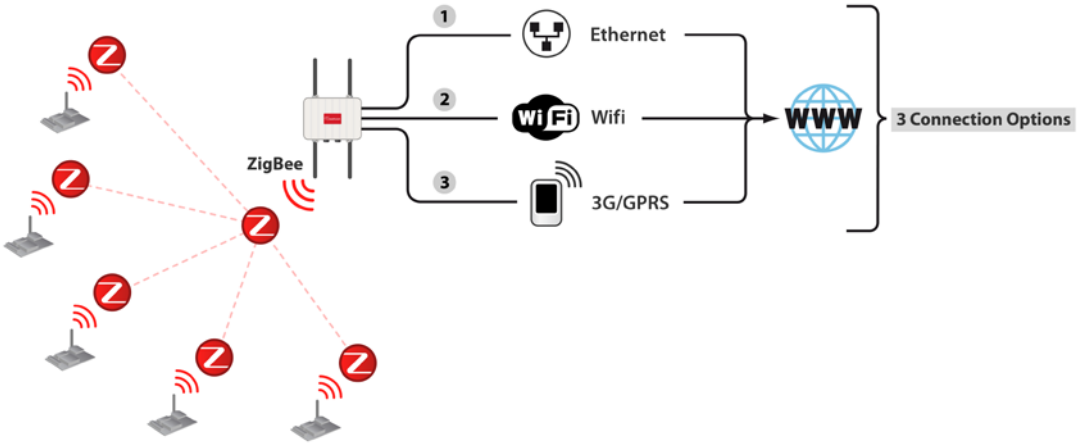


Figure 131: Meshlium Connection Options

- ZigBee -> Ethernet
- ZigBee -> Wifi
- ZigBee -> 3G/GPRS

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

When you buy a kit containing Meshlium and Waspmote, they already come configured to work together. Meshlium will receive the sensor data sent by Waspmote using the ZigBee radio and it will store in the Local File System and in the Local Data Base.

The initial ZigBee frames sent by Waspmote contain the next sequence:

```
[HEADER]-mac:0013a20040307f9c -x:27,y:23,z:1023 -temp:28 -bat:97%
```

They are formed by the MAC address (64b), the acceleration in the three axis (x, y, z) , the temperature and the battery level.

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".

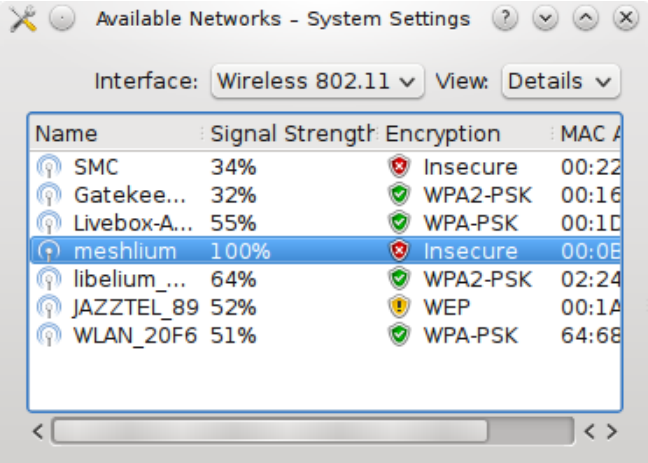


Figure 132: Available Networks screenshot

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 133: Meshlium Manager System Login screen

Now we go to the "Sensor Networks" tab.

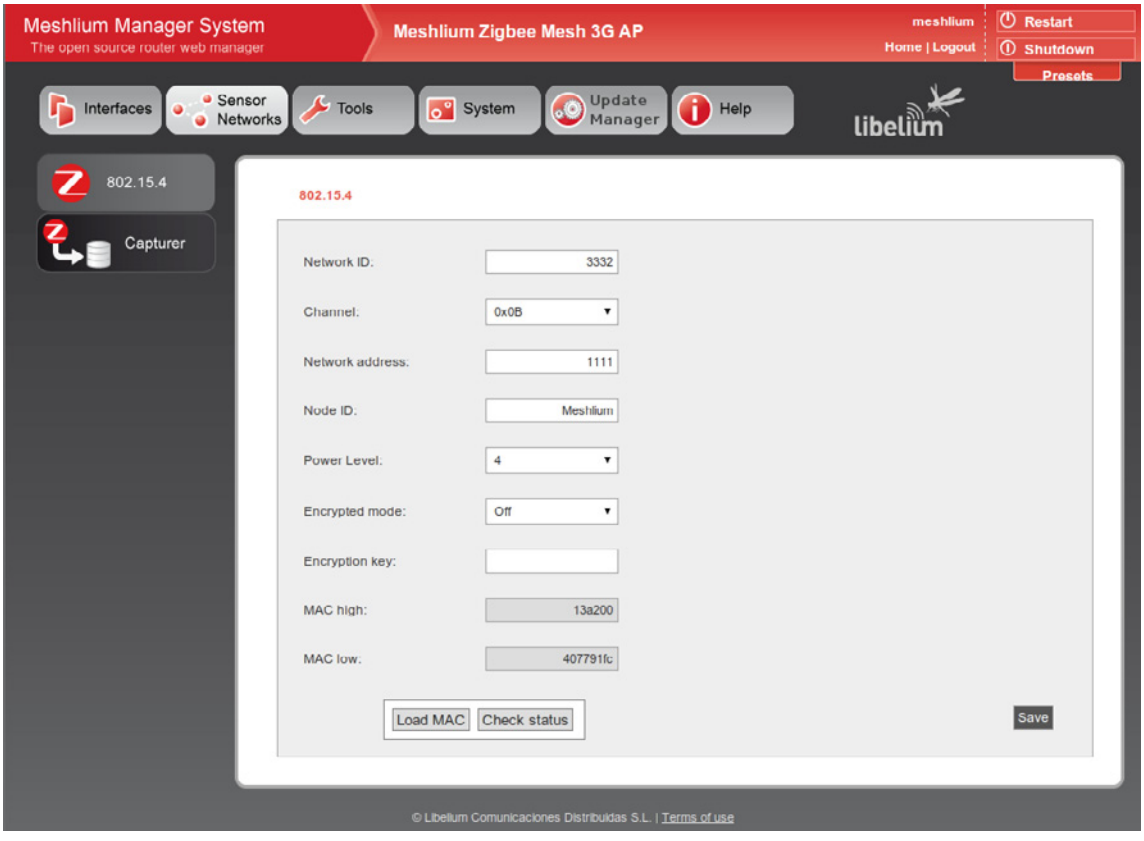


Figure 134: Sensor Networks tab

There are 6 different XBee models can can be configured:

Figure 135: 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**

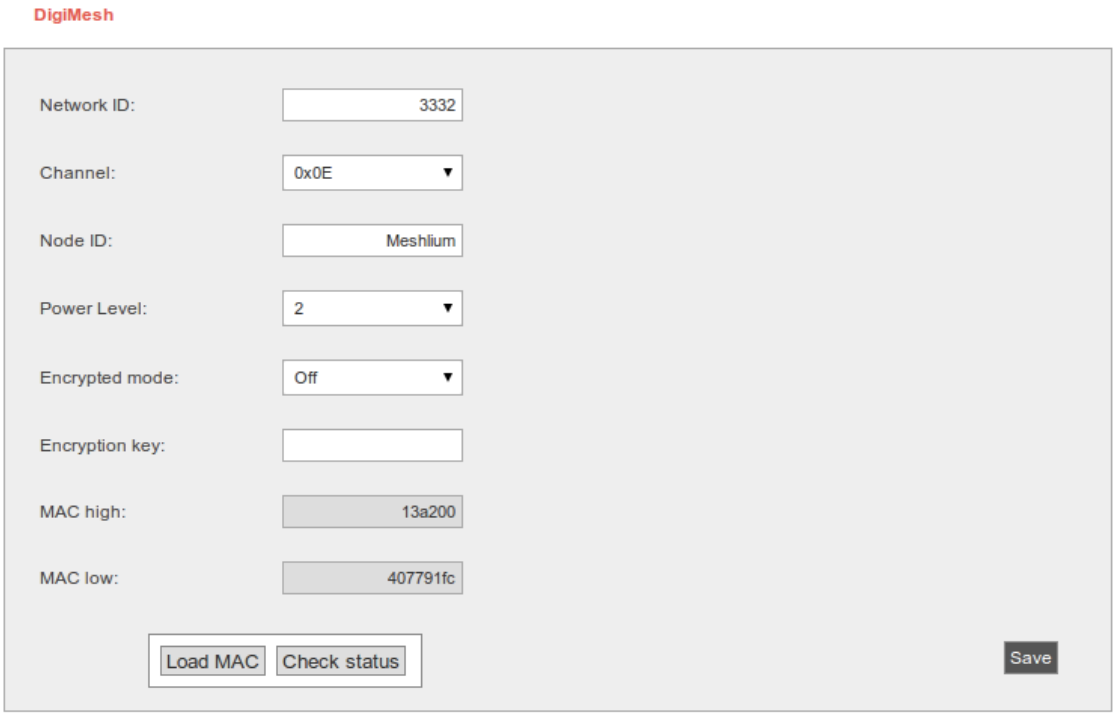


Figure 136: XBee parameters configuration

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.

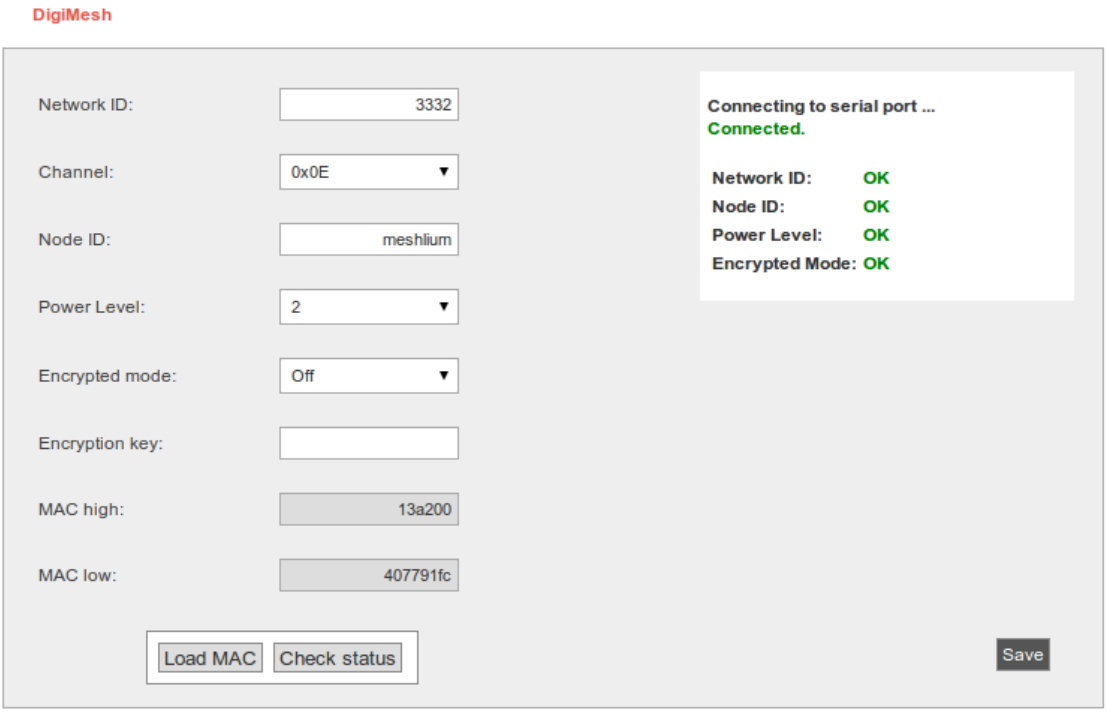


Figure 137: XBee parameters configuration

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

When you buy a kit containing Meshlium and Waspmote, they already come configured to work together. Meshlium will receive the sensor data sent by Waspmote using the ZigBee radio and it will store in the Local File System and in the Local Data Base.

The initial ZigBee frames sent by Waspmote contain the next sequence:

```
[HEADER]-mac:0013a20040307f9c -x:27,y:23,z:1023 -temp:28 -bat:97%
```

They are formed by the MAC address (64b), the acceleration in the three axis (x, y, z), the temperature and the battery level.

In order to add your own sensor frames properly go to the section *"Capturing and storing your own ZigBee frames"*. However if you change the sensor information sent by Waspmote without changing the default capturer daemon it will be saved as a generic "Frame" in the database. See the picture below in order to see different frames types and how they are saved in the database.

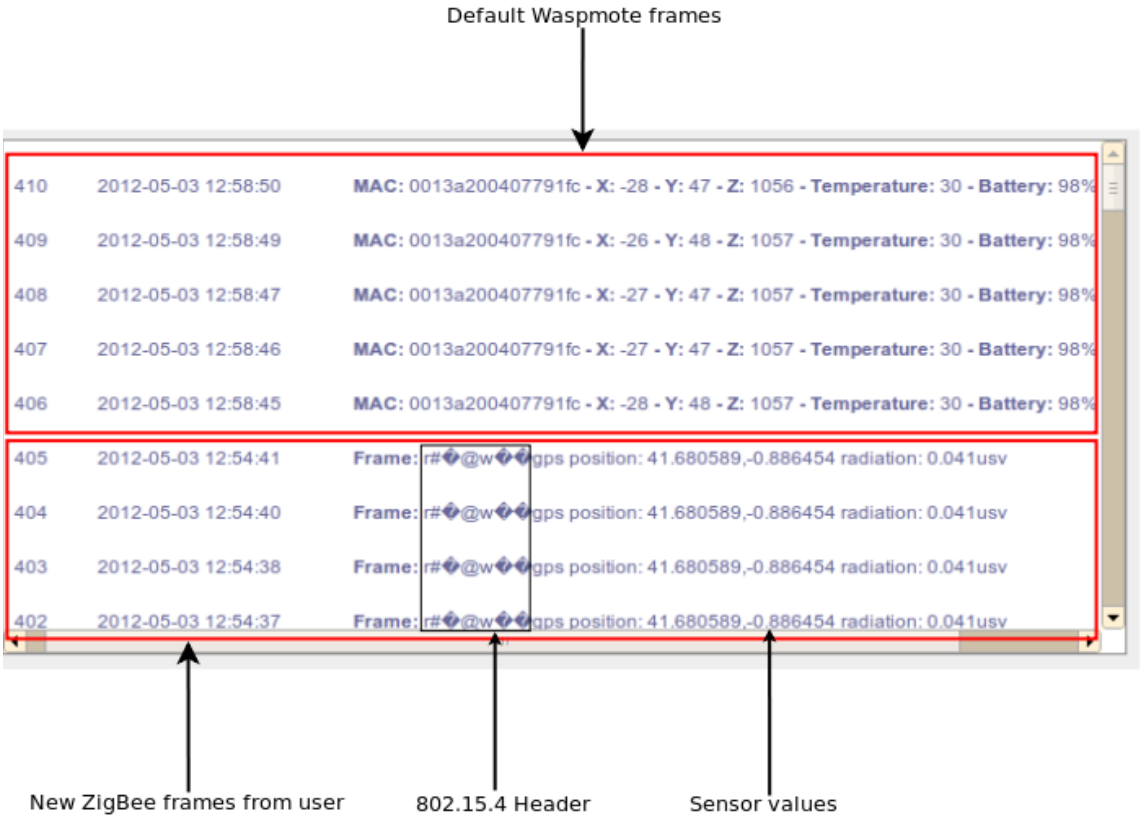


Figure 138: Different frames types

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 three different storage options with the ZigBee frames captured:

- Local File System
- Local Data Base
- External Data Base

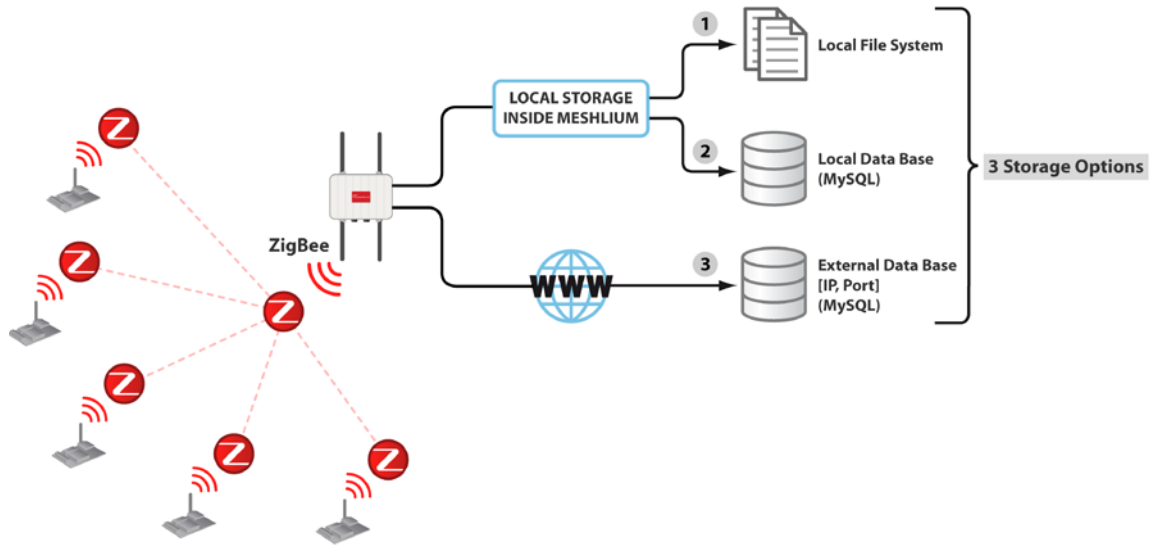


Figure 139: Meshlium Storage options

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

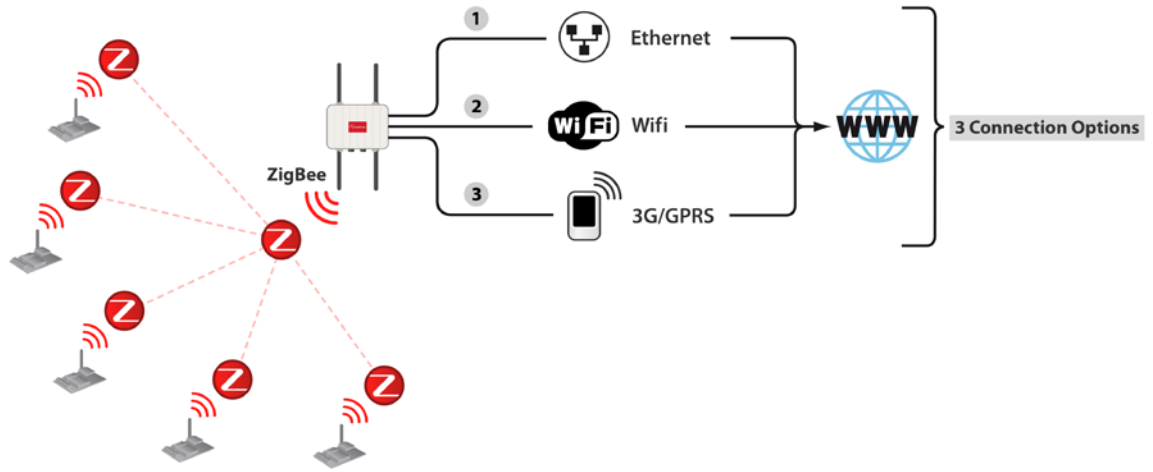


Figure 140: Meshlium Connection options

Local File System

Steps:

1. Give a name to create a new file where the ZigBee data will be saved.
2. Select this file and press the "Select file" button.
3. Set the check box "Store frames in the selected file" and press the "Save" button.

From now Meshlium will automatically capture the ZigBee frames and will store the results in this file. This process will also continue after restarting Meshlium.

The file will be created in the folder "/mnt/user/zigbee_data" and can be downloaded just selecting it and pressing the "Download" button.

At any time you can see the last "x" lines added to the file. Just set how many lines you want to see and press the "Show data" button.

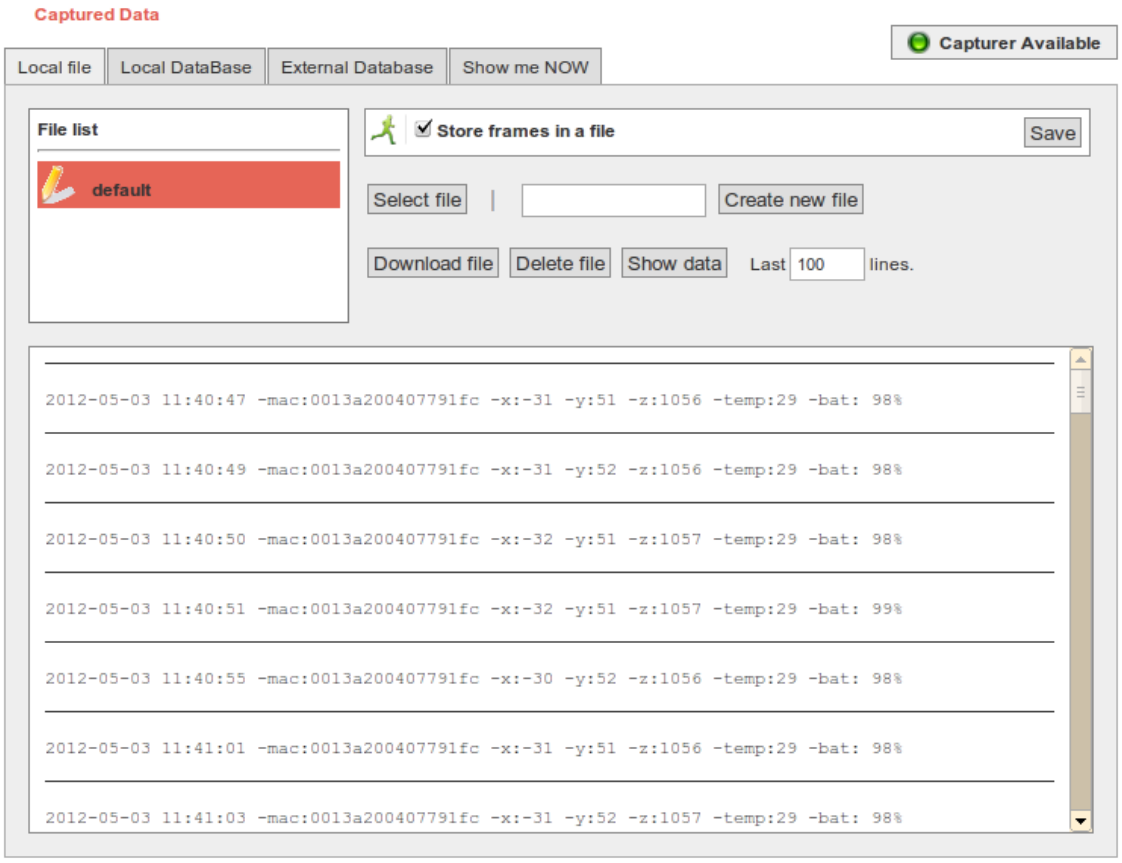


Figure 141: Local File tab

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:** zigbeeData
- **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.

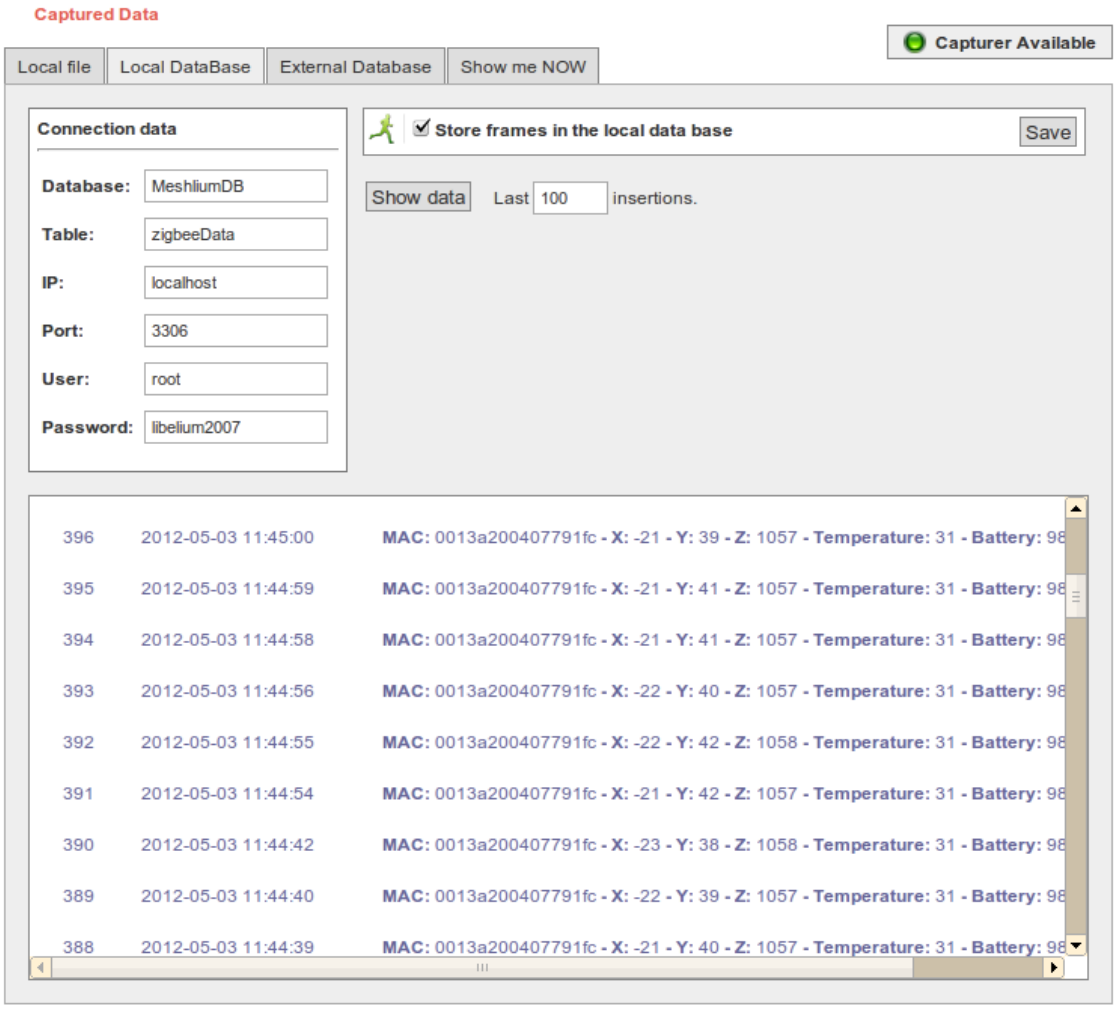


Figure 142: Local Data Base tab

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.

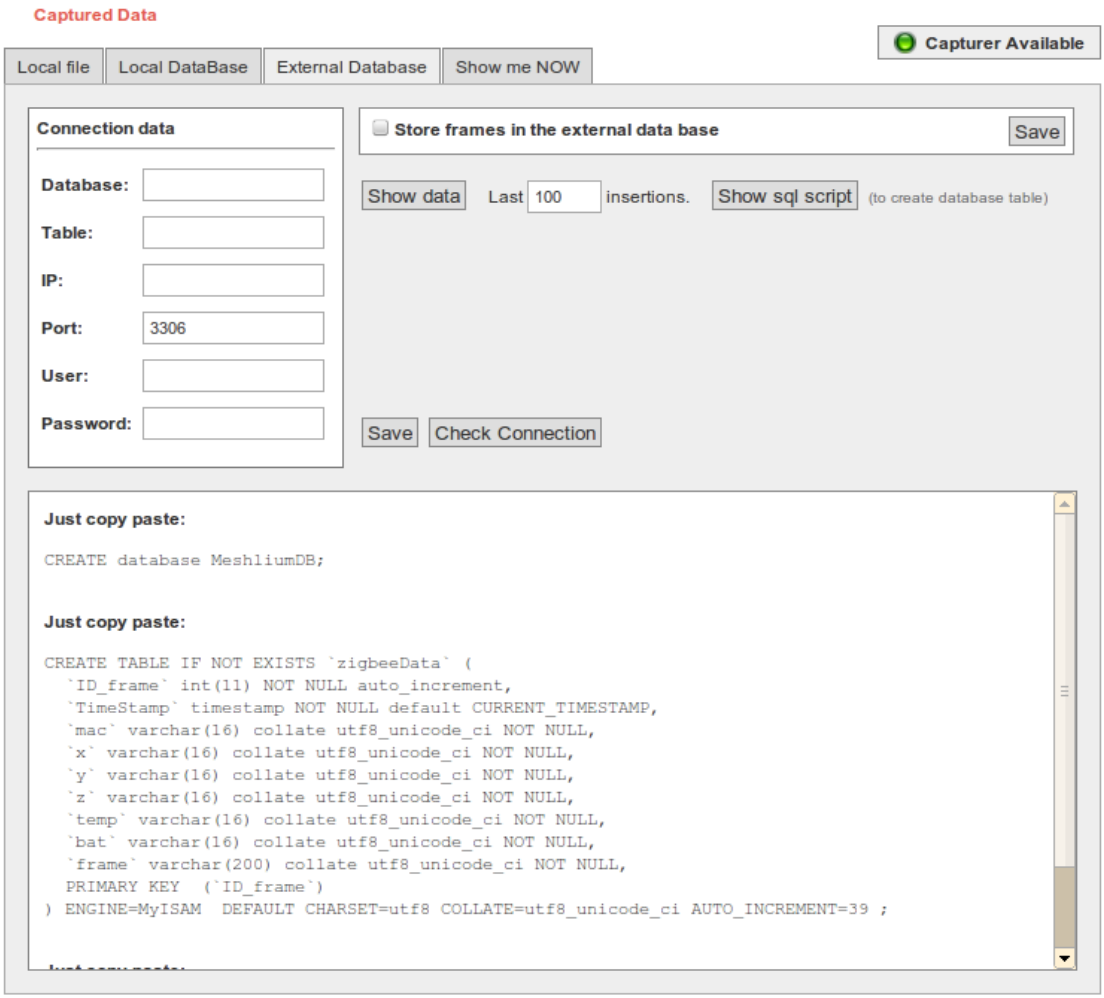


Figure 143: External Database tab - showing SQL Script

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

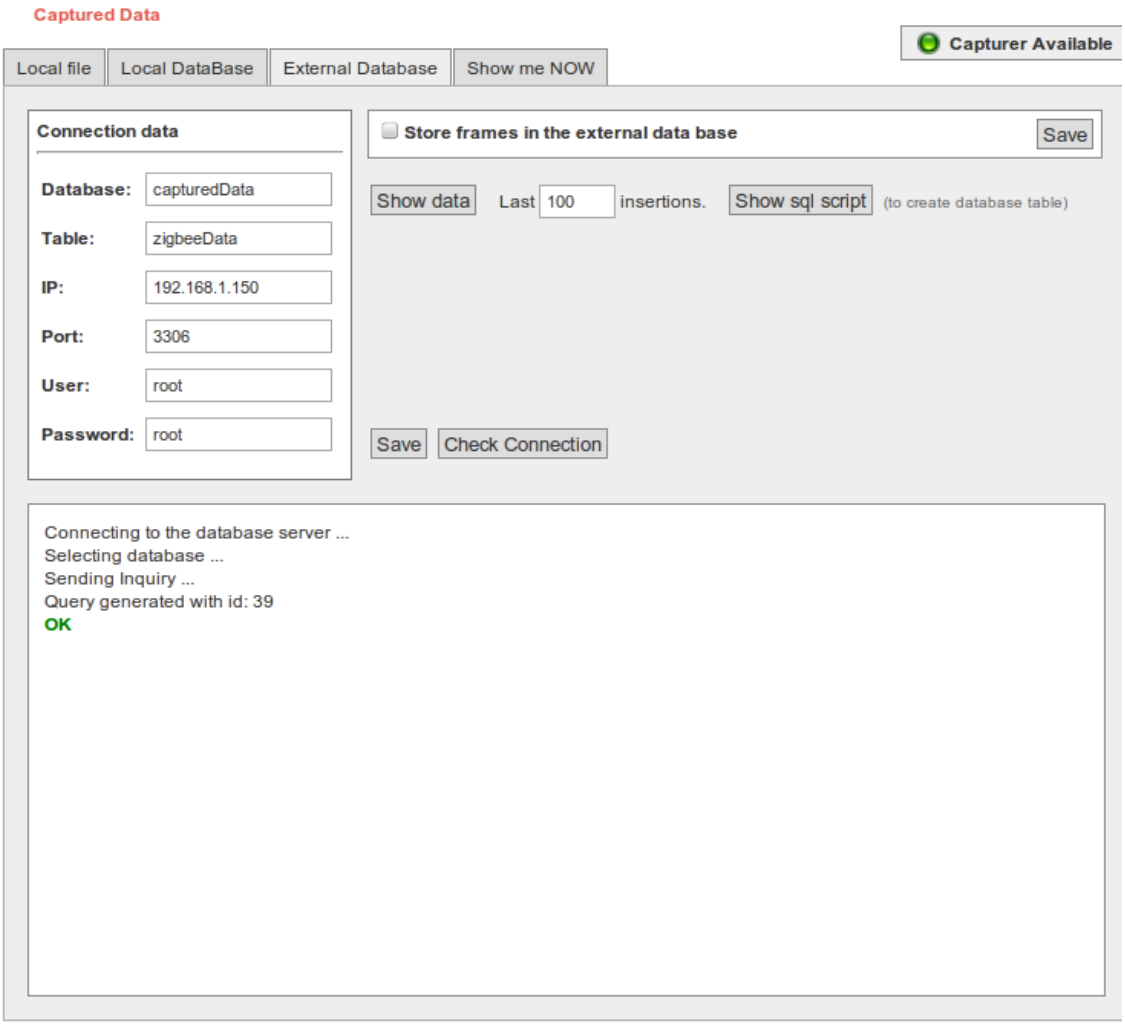


Figure 144: External Database tab - checking connection

5. Set the check box "Store frames in the selected file" 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.

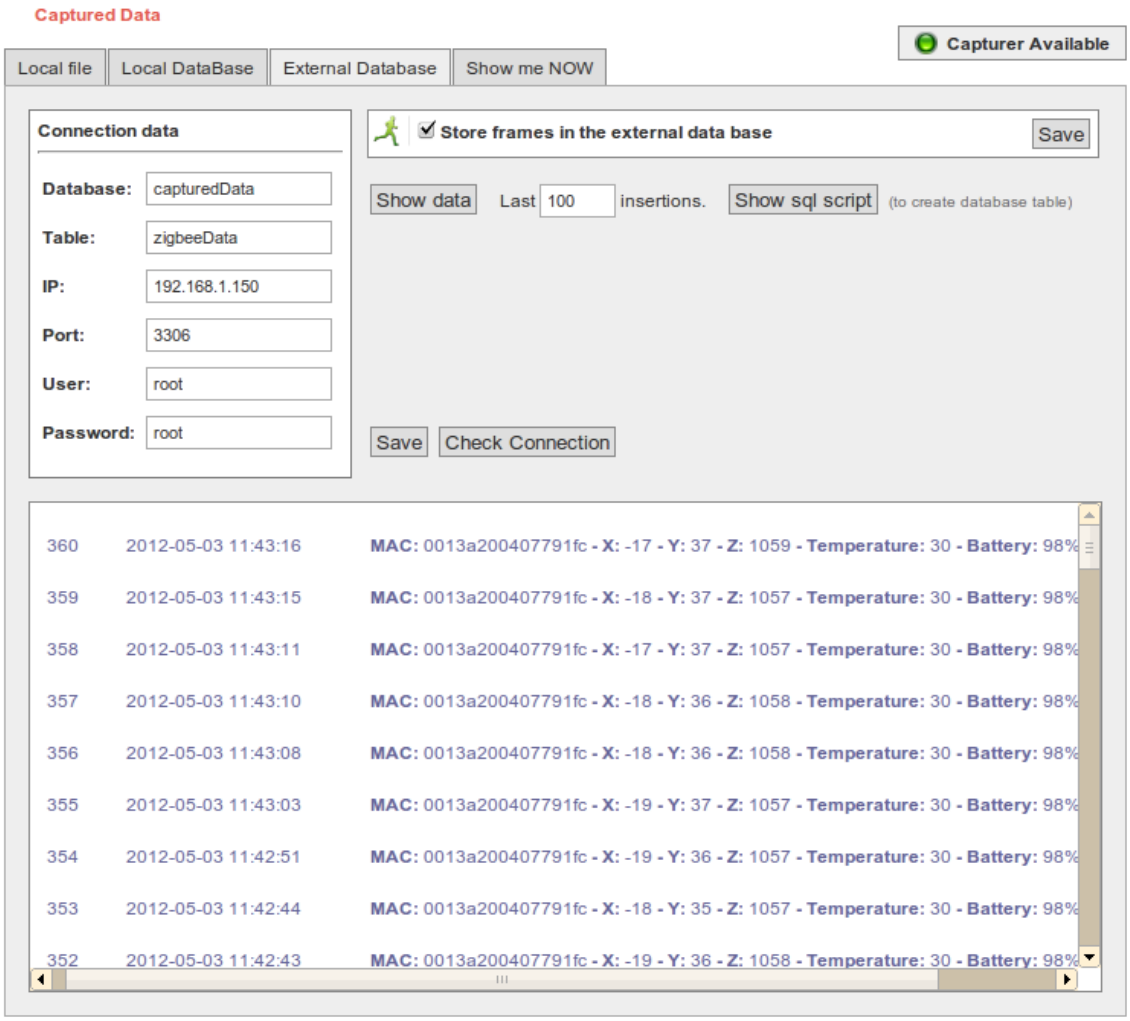


Figure 145: External Database tab - last "x" records stored

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.

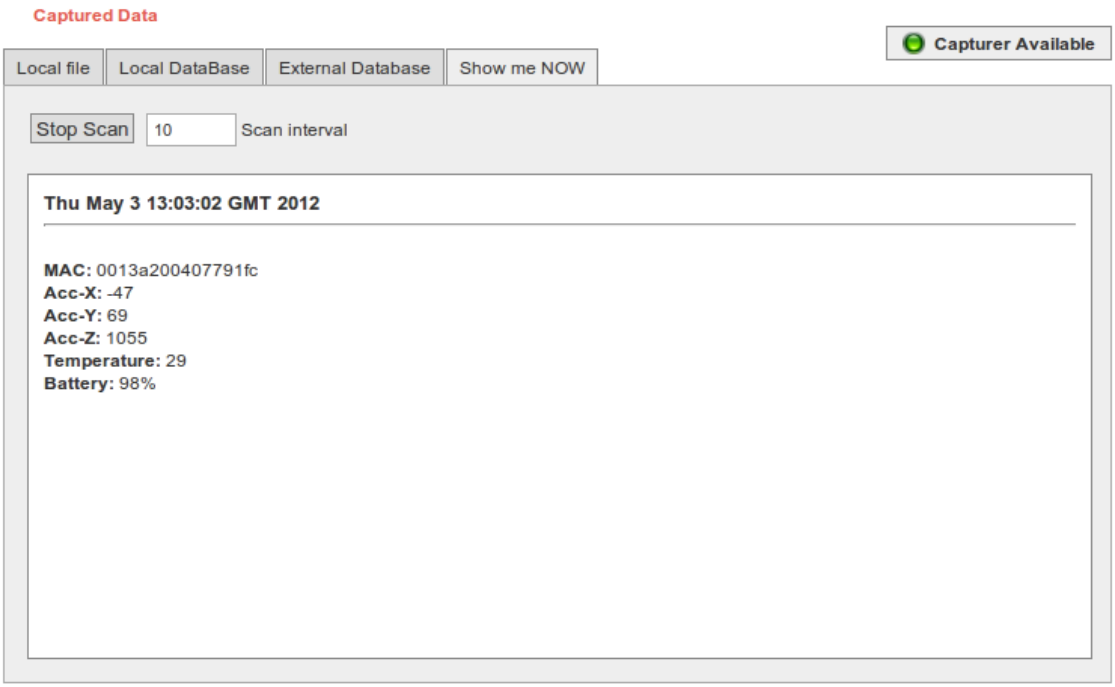


Figure 146: Show me now! tab

21.2.4. Capturing and storing your own ZigBee frames

Their own ZigBee frames are stored in the local file and the databases with the prefix Frame, but you can create your own capture and storage applications.

We have created two template code files in order to help developers to create their own capture and storage applications.

The default ZigBee "capturer and storer" comes with the next structure (pseudocode):

```
main()
{
 init();            //initializes the serial port to communicate with the XBee radio
       while(1)
       {
        getFrame();             //read the frame received from the ZigBee radio
        parseFrame();           //parse the frame in order to extract the data
        storeInAFile(fileName);   //store the data in a file (if enabled)
        storeInADB(LOCAL);        //store the data in the local DB (if enabled)
        storeInADB(EXTERNAL);     //store the data in an external DB (if enabled)
       }
 }
```

The real files (zigbeeStorer.h, zigbeeStorer.c) 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 make it compatible for your specific sensor configuration.

Compilation:

The compilation can be done in the same Meshlium. Just copy these files in a folder accessing by SSH and execute:

```
$ gcc -o zigbeeStorer -I/usr/include/mysql zigbeeStorer.c -L/usr/lib/mysql -lmysqlclient
-Wall
```

Important: before starting to execute the new binary "zigbeeStorer" stop the default ZigBee daemon:

```
$ /etc/init.d/ZigbeeScanD.sh stop
```

If you want your own ZigBee capturer application to execute each time Meshlium starts you have to change the daemon file (/etc/init.d/ZigbeeScanD.sh) in order to execute your binary.

You will find support in the Libelium Forum at: **http://www.libelium.com/forum**

21.2.5. 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:

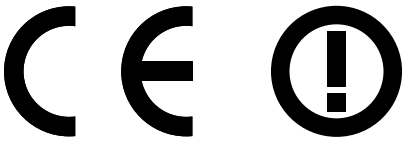
```
$ gcc -o ZigBeeSend ZigBeeSend.c -lpthread
```

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**

22. Certifications

22.1. 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)

22.2. FCC

Waspmote models:

Model 1- FCC (XBee PRO series 1 OEM + 3G/GPRS Hilo)
FCC ID: XKM-WASP01 comprising
- FCC ID: OUR-XBEEPRO
- FCC ID: VW3HILOC

Model 2- FCC (XBee PRO ZB series 2 + 3G/GPRS Hilo)
FCC ID: XKM-WASP02 comprising
- FCC ID: MCQ-XBEEPRO2*
- FCC ID: VW3HILOC

Model 3 - FCC (XBee 900MHz + 3G/GPRS Hilo)
FCC ID: XKM-WASP03 comprising
- FCC ID: MCQ-XBEE09P
- FCC ID: VW3HILOC

Model 4 - FCC (XBee 900MHz XSC + 3G/GPRS Hilo)
FCC ID: XKM-WASP04 comprising
- FCC ID: MCQ-XBEEXSC
- FCC ID: VW3HILOC

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 MCQ-XBEEXSC

For operations in mobile RF exposure conditions, the antenna(s) used for this transmitter must be installed to provide a separation distance of at least 23 cm from all persons with the maximum allowable antenna gains of 9.2 dBi Omni-directional antenna or 15.1 dBi Yagi antenna. Antenna(s) used for this transmitter must not be co-located or operating in conjunction with any other antenna or transmitter, OEM integrators, end-users, and professional installers must be provided with antenna installation instructions and transmitter operating conditions for satisfying RF exposure compliance. End-users are prohibited from access to any programming parameters, professional installation adjustment is required for setting module power and antenna gain to meet EIRP compliance ξ FCC 15.247(b)(4).

FCC ID VW3HILOC

The antenna(s) 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 any other antenna or transmitter. OEM integrators must be provided with antenna installation instructions. OEM integrators and end-Users must be provided with transmitter operation conditions for satisfying RF exposure compliance.

22.3. IC

Waspmote models:

Model 1- IC (XBee PRO series 1 OEM + 3G/GPRS Hilo)
IC: 8472A-WASP01 comprising
- IC: 4214A-XBEEPRO
- IC: 2599H-HILOC

Model 2- IC (XBee PRO ZB series 2 + 3G/GPRS Hilo)
IC: 8472A-WASP02 comprising
- IC: 1846A-XBEEPRO2
- IC: 2599H-HILOC

Model 3- IC (XBee 900MHz + 3G/GPRS Hilo)
IC: 8472A-WASP03 comprising
- IC: 1846A-XBEE09P
- IC: 2599H-HILOC

Model 4 - IC (XBee 900MHz XSC + 3G/GPRS Hilo)
IC: 8472A-WASP04 comprising
- IC: 1846A-XBEEXSC
- IC: 2599H-HILOC

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.

22.4. 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.

22.5. 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 - 3G/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.

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 - 3G/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**

23. 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.

24. 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.