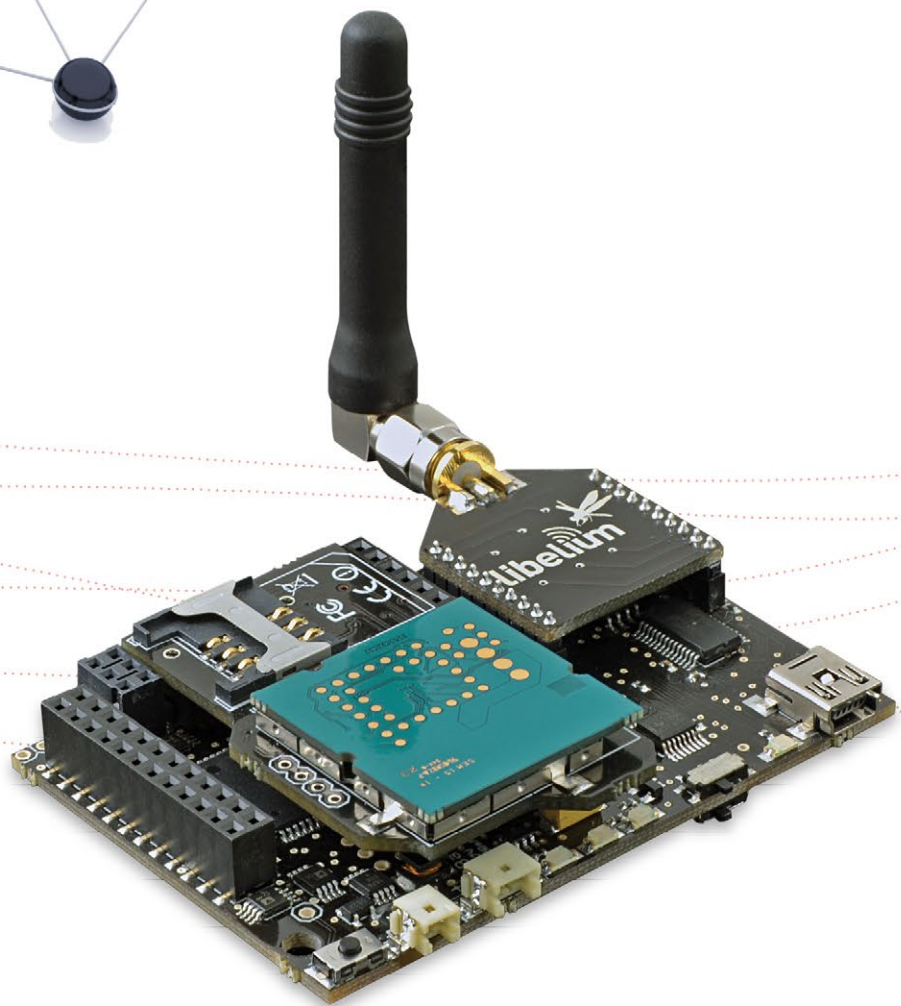
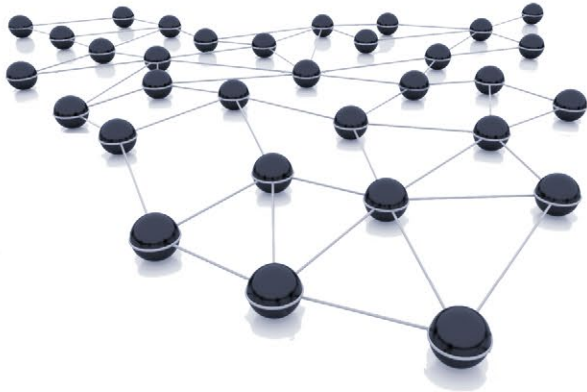


# WiFi Module

## Networking Guide



Document version: v4.4 - 06/2014

© Libelium Comunicaciones Distribuidas S.L.

# INDEX

<b>1. Introduction .....</b>	<b>4</b>
<b>2. Topologies .....</b>	<b>5</b>
2.1. Access Point .....	5
2.2. Ad-hoc mode with iPhone/Android.....	8
<b>3. Hardware .....</b>	<b>9</b>
3.1. Specifications .....	9
3.2. Power consumption.....	9
3.3. Bandwidth .....	10
3.4. Distance ranges .....	10
3.5. Connection times.....	10
<b>4. General considerations .....</b>	<b>11</b>
4.1. Waspote library .....	11
4.1.1. Waspote WiFi files.....	11
4.1.2. Constructor.....	11
4.2. API functions.....	12
4.3. Waspote reboots .....	16
4.4. Constants predefined.....	16
4.5. Class Variables .....	18
<b>5. Initialization .....</b>	<b>19</b>
5.1. Expansion Radio Board .....	19
5.2. Setting ON .....	20
5.3. Setting OFF.....	20
<b>6. Access Point Configuration .....</b>	<b>21</b>
6.1. Manual Mode .....	21
6.2. Automatic Mode.....	21
6.3. Examples .....	22
<b>7. Network Configuration .....</b>	<b>23</b>
7.1. DHCP protocol .....	23
7.2. IP manual .....	23
<b>8. Connection Creation .....</b>	<b>24</b>
8.1. TCP.....	24
8.1.1. TCP Client .....	24
8.1.2. TCP Server .....	25

8.2. UDP .....	25
8.2.1. UDP Client.....	25
8.2.2. UDP Server.....	26
8.3. HTTP .....	26
8.3.1. GET method .....	26
8.3.2. Server response .....	26
8.3.3. Sending HTTP queries .....	27
8.3.4. Sending a frame to Meshlium .....	28
8.4. FTP.....	28
8.5. Broadcast-UDP .....	29
8.6. Sending standard Libelium Frames.....	30
<b>9. Device Options.....</b>	<b>31</b>
9.1. Default settings .....	31
9.2. Improving performance .....	32
9.3. Sleeping the radio WiFi.....	33
<b>10. Status Information .....</b>	<b>34</b>
<b>11. Connecting to Meshlium.....</b>	<b>35</b>
11.1. Setting Meshlium as AP.....	36
11.2. Capturing and storing packets .....	38
<b>12. Connecting to a PC .....</b>	<b>42</b>
<b>13. Connecting to a Smartphone directly .....</b>	<b>44</b>
13.1. Connecting to an iPhone .....	44
13.1.1. Installation.....	44
13.1.2. iPhone App tutorial.....	46
13.2. Connecting to an Android.....	49
13.2.1. Installation.....	49
13.2.2. Android App tutorial .....	50
<b>14. Code examples and extended information .....</b>	<b>54</b>
<b>15. API Changelog .....</b>	<b>56</b>
<b>16. Documentation changelog .....</b>	<b>59</b>

# 1. Introduction

This guide describes all features of the **WiFi module for the Wasp mote sensor platform**, which has been designed to make a connection between a Wasp mote and any other device with a WiFi radio (smart-phones, computers, routers...) or to make a connection directly to an Internet server.

The Wasp mote WiFi module is able to make connections and communicate by using different ways. Connections can be done by Ad-hoc (point to point) or infrastructure mode (connecting to an available Access Point - AP). In the Infrastructure mode we can connect to the AP's by using secure connections such as WEP-128, WPA-PSK (TKIP) and WPA2-PSK (AES).

Furthermore, the WiFi module can manage high level networking applications such as:

- DHCP client
- DNS client
- ARP
- ICMP pings
- FTP
- TELNET
- HTTP
- TCP
- UDP or UDP Broadcast
- OTA feature can be performed now by Wasp mote. Refer to over the air programming guide for more information.  
[www.libelium.com/development/waspote/documentation/over-the-air-programming-guide-otap/](http://www.libelium.com/development/waspote/documentation/over-the-air-programming-guide-otap/)

This guide will try to describe main parts of the WiFi module, its characteristics like consumption and ranges (between them and between a WiFi module and a device like a Meshlium or a smart-phone), and how to carry out many types of inquiries, explaining the API and providing code examples for each case. It also describes the software made to test and connect WiFi module, like PC software and mobile applications.

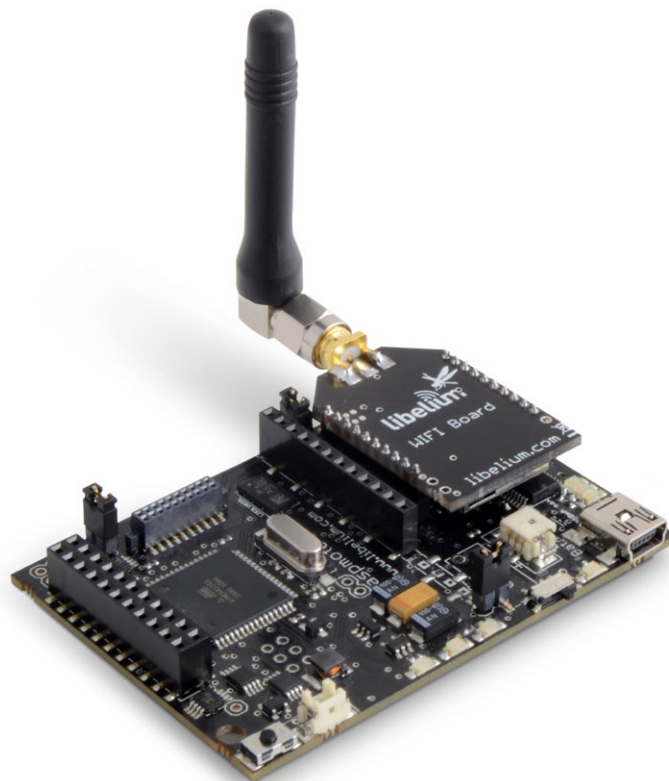


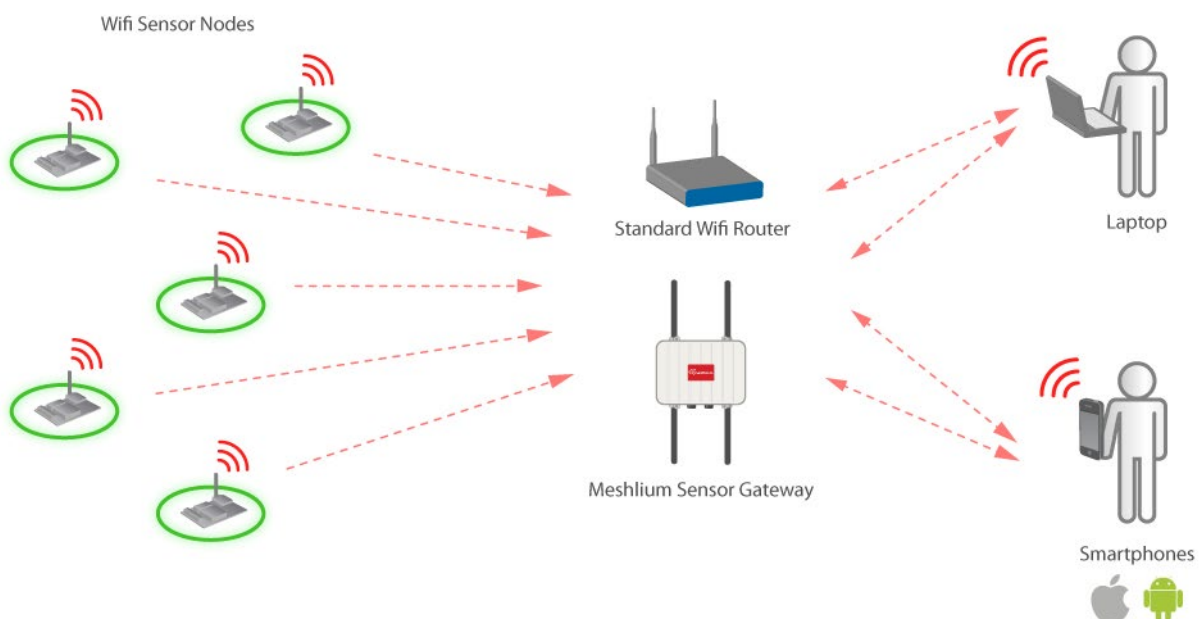
Figure: Image of WiFi module connected to Wasp mote

## 2. Topologies

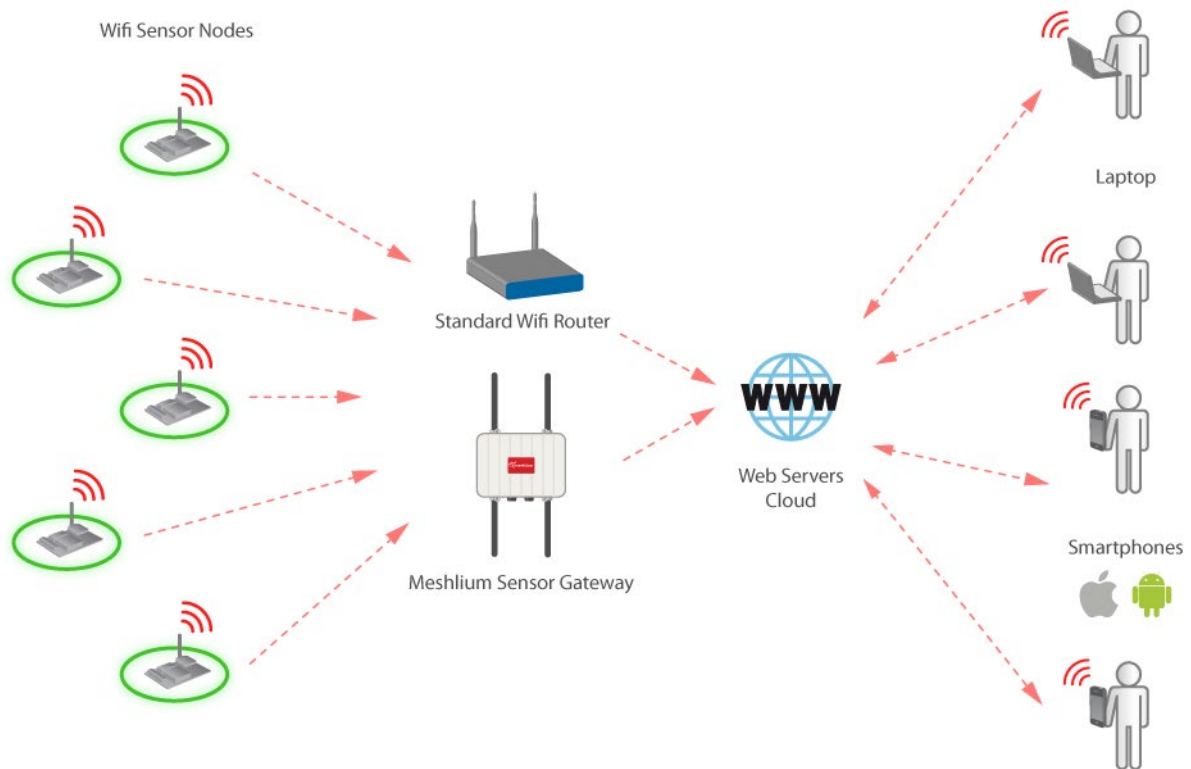
### 2.1. Access Point

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 Wasp mote 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:

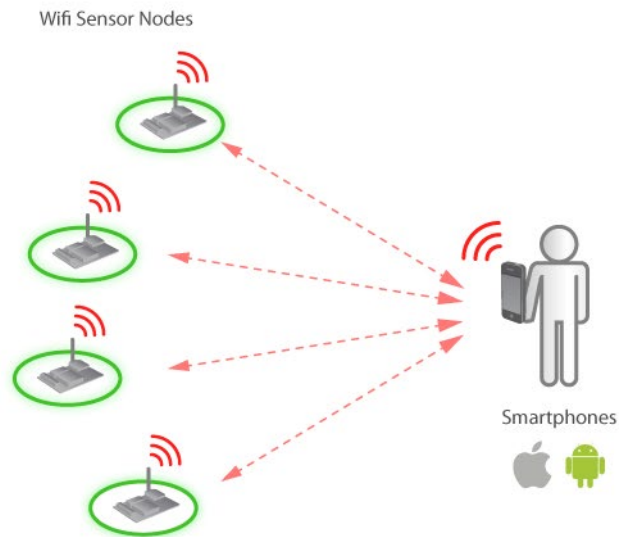
- a ZigBee to Ethernet router for Wasp mote nodes
- a ZigBee to 3G/GPRS router for Wasp mote nodes
- a WiFi Access Point
- a WiFi Mesh node (dual band 2.4GHz-5GHz)
- a WiFi to 3G/GPRS router
- a Bluetooth scanner and analyzer
- a GPS-GPRS real time tracker

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



## 2.2. Ad-hoc mode with iPhone/Android

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



More information about this topology in section “Connecting to a Smartphone directly”.



## 3. Hardware

### 3.1. Specifications

#### Main Features:

- TX Power: 0dBm - 12dBm \* (*variable power controlled by software*)
- RX Sensitivity: -83dBm
- Antenna: 2dBi/5dBi antenna options
- Antenna connector: SMA standard connector for any kind of antenna
- AP and Ad-hoc topologies
- DHCP enabled
- TCP/IP - UDP/IP connections
- HTTP and HTTPS (Secure) connections
- FTP and FTPS (Secure) connections
- FCC / CE/ IC certified 2.4GHz IEEE 802.11b/g



Figure: WiFi module with 2dBi and 5dBi antennas

### 3.2. Power consumption

Next table shows average power consumption in different states of the module.

Estate	Power Consumption
OFF	0 uA
SLEEP	4 uA
ON	33 mA
Receiving Data	38 mA
Transmitting Data	38 mA
Scanning Access Points	34 mA

### 3.3. Bandwidth

Baud-rates	Transmission	Reception
9600	142 ms 5,63 kbps	8 ms 100 kbps
19200	87 ms 9,19 kbps	8 ms 100 kbps
57600	51 ms 15,68 kbps	8 ms 100 kbps

### 3.4. Distance ranges

Distance ranges depend on the gain of the antenna and the TX power selected. The different tests have been done using 2dBi and 5dBi antennas and TX in maximum power (12dBm).

The distance ranges changes in communication between Waspote with WiFi module and connecting it to an Access Point like Meshlium. They also are different for a perfect transmission (100 % UDP packets received) and for the visibility of the AP (longer distance range).

Next table shows the distance ranges of the module depending of the antenna.

Waspote + 2dBi antenna	50m - 100m
Waspote + 5dBi antenna	300m - 500m

The final distance varies on parameters such as if the Fresnel Zone is respected or the humidity in the ambient.

### 3.5. Connection times

It has been measured the connection and join times in some typical cases. This is quite useful for configuring sleep and wake timers. These cases are:

- **Join AP times with different type of encryption**

Encryption	Time (seconds)
Open	3,997 s
WEP	4,077 s
WPA	4,176 s
WPA2	4,293 s

- **Join times in Ad-hoc mode.**

In this case, the test has been made between two WiFi module in Ad-hoc mode.

Type	Time (seconds)
Ad-hoc	10 s

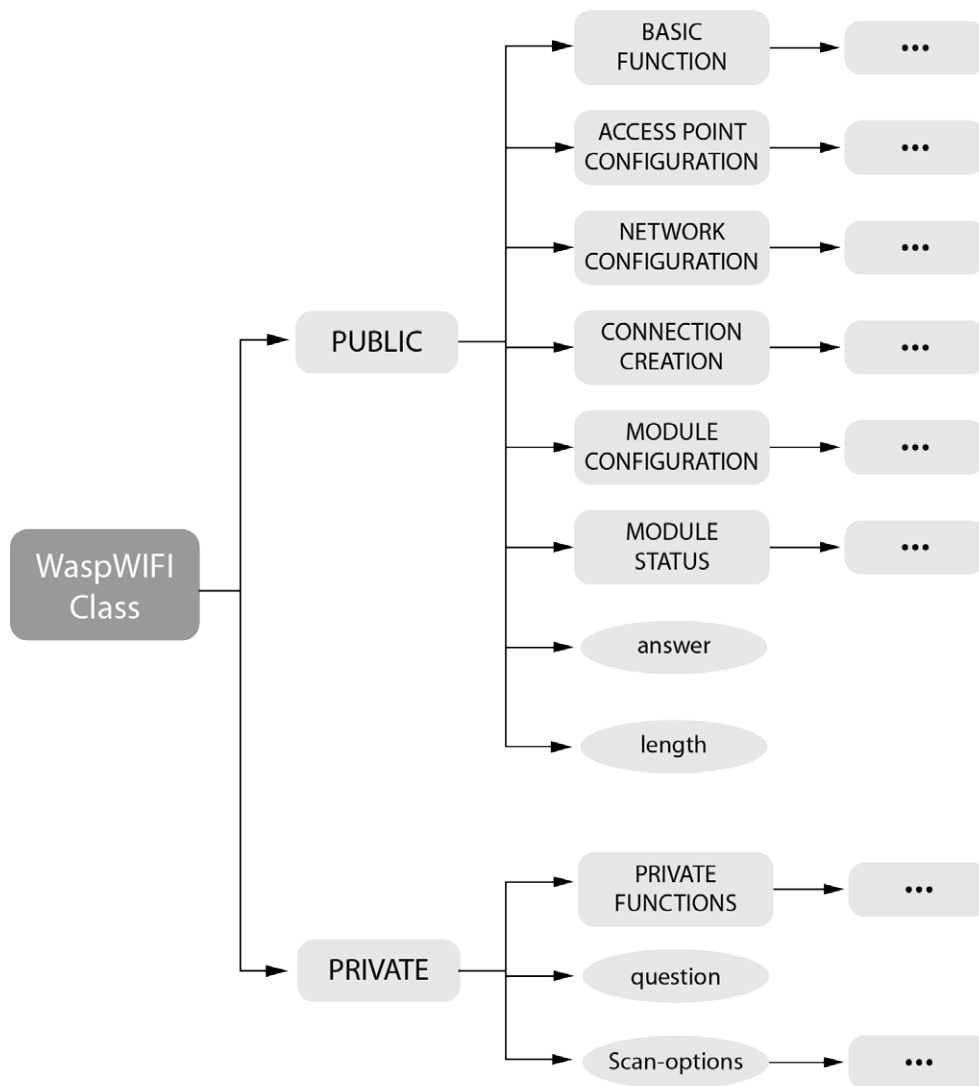
## 4. General considerations

This section will describe the API for the WiFi module. The functions that manage the WiFi module belong to the class WaspWiFi, and the object used to use them is defined as [WIFI](#). All of them are described below including some examples of use.

### 4.1. Waspmote library

#### 4.1.1. Waspmote WiFi files

WaspWiFi.h is the header file of the class, and WaspWiFi.cpp is the class where the functions and variables are implemented. Next diagram shows the organization of the class.



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

```
#include <WaspWiFi.h>
```

#### 4.1.2. Constructor

To start using **Waspmote WiFi library**, an object from class '**WaspWiFi**' must be created. This object, called '**WIFI**', is created inside Waspmote WiFi library and it is public to all libraries. It is used through this guide to show how Waspmote WiFi library works.

When creating this constructor, all the variables are defined with an initial value by default.

## 4.2. API functions

### Private functions:

<code>commandMode();</code>	Switches the module to command Mode.
<code>contains(text, word);</code>	Checks if 'word' is contained in 'text'.
<code>readData(length);</code>	Reads data over the UART.
<code>sendCommand(command);</code>	Sends a command over the UART.
<code>parseBroadcast();</code>	Reads a broadcast message and parses it.
<code>saveReboot();</code>	Saves current configuration and reboots the device in order to new configuration takes effect.
<code>open();</code>	Opens TCP connection, and checks that everything are going good.
<code>openHTTP();</code>	Opens HTTP connection, and checks that everything are going good.

### Public functions:

#### Basic functions

<code>WaspWiFi();</code>	Class constructor.
<code>ON(SOCKET);</code>	Powers on the module in the desired socket ( <code>SOCKET0</code> or <code>SOCKET1</code> ) and enters in command mode.
<code>OFF();</code>	Closes the UART and powers off the module.

#### Access Point Configuration functions

<code>join(SSID);</code>	Joins the chosen SSID.
<code>joinAPnum(value);</code>	Joins a network AP from the scan list.
<code>leave();</code>	If connected, leaves the connected network.
<code>setChannel(value);</code>	Sets the network channel.
<code>setESSID(SSID);</code>	Sets the ESSID of the network.
<code>setAuthKey(mode, pass);</code>	Sets the security mode and the Key or the pass-phrase of the <code>network</code> .
<code>setAutojoinAuth(value);</code>	Sets the authentication mode.
<code>setJoinMode(value);</code>	Sets the policy for automatically or manual joining/associating with <code>network</code> access points.

<code>setAPretries(value);</code>	Sets the link monitor timeout threshold.
<code>setTXRate(value);</code>	Sets the Transmission Rate.
<code>setTXPower(value);</code>	Sets the Transmission Power.
<code>setIPWindow(value);</code>	Sets the IP Window value.
<code>setJoinTime(value);</code>	Join timer is the time in milliseconds the join function will wait for an access point to complete the association process. This timer is also the timeout for the WPA handshaking process.
<code>setScanOptions(time, mask, mode);</code>	Sets the Scan options (the time scanning, the channel mask, and if it is passive or active scan), does not launch the scan.
<code>scan();</code>	Launches the scan.

### Network Configuration functions

<code>sendPing(IP);</code>	Sends a ping to a remote specified host.
<code>resolve(name);</code>	Performs a DNS query on the supplied host-name.
<code>setTimeFromServer(IP, port, interval);</code>	Synchronizes the time from a server. And if interval > 0, the module will synchronize every interval seconds.
<code>setDebugMode(value);</code>	Controls debug print messages.
<code>setGW(IP);</code>	Sets the Gateway address.
<code>setNetmask(net mask);</code>	Sets the net-mask.
<code>setRemoteHost(IP, port);</code>	Sets the Remote Host.
<code>setLocalPort(port);</code>	Sets the Local Port.
<code>setIp(IP);</code>	Sets the IP address.
<code>setDHCPoptions(value);</code>	Sets the DHCP options.
<code>setIPOptions(value);</code>	Sets optional TCP/IP functions/flags.
<code>setConnectionOptions(value);</code>	Sets the connection protocol type.
<code>setTCPpassw(password);</code>	Sets the TCP connection password.
<code>setDNS(type, IP, name);</code>	Sets the IP address, host and backup host of the DNS main or auxiliary server.

### Connection Creation Operations

<code>setAdhocOptions(beacon, probe interval);</code>	Configures for Ad-hoc and opens an Ad-hoc connection by setting the beacon in milliseconds and the probe timeout in seconds.
<code>sendAutoBroadcast(IP, port, interval, id);</code>	Configures and sends auto UDP-broadcast messages each interval seconds.
<code>setUDPclient(IP remote, port remote, port local);</code>	Sets the configuration and prepares it to send a UDP datagram.
<code>setUDPserver(port local);</code>	Sets the configuration and prepares it to receive a UDP datagram.
<code>setTCPclient(IP remote, port remote, port local);</code>	Sets the configuration and opens a TCP connection.
<code>setTCPserver(port local);</code>	Sets the configuration and waits a TCP server connection in the selected port.
<code>getURL(mode, host, request);</code>	Sets the HTTP configuration and opens an HTTP connection.
<code>setFTP(IP, port, mode, timeout);</code>	Sets the FTP parameters.
<code>openFTP(user, password);</code>	Opens the FTP connection using user and password given.
<code>getFile(file, local folder, remote folder);</code>	Gets the file via FTP.
<code>uploadFile(file, local folder, remote folder);</code>	Uploads a file via FTP.
<code>read(val);</code>	Reads data from the opened connection. If val is BLO, WaspMote will be blocked until data is received. In the other case, with val is UNBLO, if nothing is received during 2 seconds, the program skip this function.
<code>send(data);</code>	Sends data to the opened connection.
<code>send(data, length);</code>	Sends standard Libelium frames to the opened connection.
<code>close();</code>	Closes current TCP connection.

### Device configuration

<code>reset();</code>	Reboots the device (ON-OFF-ON).
<code>resetValues();</code>	Restores the default settings of the device.
<code>setSleep(sleep time, sleep cycle);</code>	Sets the sleep time and the wake time (in seconds).
<code>sleep();</code>	Sleeps the device.
<code>wake();</code>	Wakes the device and switches to command mode.
<code>synchronizeTime();</code>	Synchronizes the time.
<code>setBaudRate(value);</code>	Sets the UART baud rate.
<code>storeData();</code>	Saves current configuration.

## Status Information

<code>isConnected();</code>	Checks if the module is correctly connected to the Access Point.
<code>getConnectionInfo();</code>	Displays connection status.
<code>getAPstatus();</code>	Displays current network status, association, authentication, etc.
<code>getRSSI();</code>	Displays current last received signal strength.
<code>getStats();</code>	Displays current statistics, packet Rx/TX counters, etc.
<code>getUpTime();</code>	Displays number of seconds since last power up or reboot.
<code>getAdhocSettings();</code>	Displays all ad-hoc settings.
<code>getBroadcastSettings();</code>	Displays the broadcast settings.
<code>getComSettings();</code>	Displays communication settings.
<code>getDNSsettings();</code>	Displays DNS settings.
<code>getFTPsettings();</code>	Displays FTP settings.
<code>getIP();</code>	Displays IP address and port number settings.
<code>getMAC();</code>	Displays the device MAC address.
<code>getOptionSettings();</code>	Displays the option settings like device ID.
<code>getSystemSettings();</code>	Displays system settings, sleep, wake timers.
<code>getTime();</code>	Displays the time-server UDP address and time-server port number.
<code>getWLANsettings();</code>	Displays the SSID, channel and other WLAN settings.
<code>getUARTsettings();</code>	Displays the UART settings.
<code>getVersion();</code>	Returns the software release version of the WiFi module.

## 4.3. Wasp mote reboots

When Wasp mote is rebooted or it wakes up from a hibernate state (battery is disconnected) the application code will start again, creating all the variables and objects from the beginning, by the way, the user can save some internal configuration by default in the WiFi module calling `storeData`.

## 4.4. Constants predefined

There are some constants predefined in the WiFi library. Internal parameters like module baud-rate, configuration modes or flags are defined here.

### Authentication Modes

OPEN	0	Open Authentication (default)
WEP	1	128-bit WEP encryption.
WPA1	2	WPA-PSK (TKIP) encryption
WPAMIX	3	WPA-PSK mixed mode encryption
WPA2	4	WPA2-PSK (AES) encryption
ADHOC	6	Ad-hoc authentication

### Join Modes

MANUAL	0	Manual policy for joining/associating.
AUTO_STOR	1	Try to join the access point that matches the stored SSID, passkey and channel.
AUTO_BEST	2	Join ANY access point with security matching the stored authentication mode.
CREATE_ADHOC	4	Create an Ad-hoc network.

### DHCP Options

DHCP_OFF	0	Use stored static IP address.
DHCP_ON	1	Get IP address and gateway from AP.
AUTO_IP	2	Generally used with Ad-hoc networks.
DHCP_CACHE	3	Uses previous IP address if lease is not expired.

### Connection Options

CLIENT_SERVER	2	00010 – TCP Server & Client protocol
CLIENT	8	01000 – TCP Client protocol
HTTP	16	10000 – HTTP protocol
UDP	1	00001 – UDP protocol
SECURE	4	00100 – UDP Secure protocol



### Flags Options

TCP_STATUS	1	00000001 – TCP connections are kept open when the connection to the access point is lost.
TCP_NODELAY	2	00000010 – Nagle's algorithm, it is a means of improving the efficiency of TCP/IP networks by reducing the number of packets that need to be sent over the network.
TCP_RETRY	4	00000100 – Enables the TCP retry.
UDP_RETRY	8	00001000 – Retry if no ACK from UDP.
DNS_CACHE	16	00010000 – Enables DNS Cache.
ARP_CACHE	32	00100000 – Enables ARP Table Cache.
UDP_AUTO_PA	64	01000000 – UDP auto pairing enabled.
BYTE_STAMP	128	10000000 - 8 byte time-stamp to UDP or TCP packets.

### DNS Options

MAIN	0	Sets the IP address or name of the DNS server.
AUX	1	Sets the IP address or name of the backup DNS server.

### HTTP Options

IP	1	HTTP connection is specified by an IP address.
DNS	2	HTTP connection is specified by a DNS address.

### READ Options

BLO	0	Blocking read.
NOBLO	1	Read with a timeout of 2 seconds.

### FTP Options

FTP_PASSIVE	0	FTP Passive mode (default).
FTP_ACTIVE	1	FTP Active mode.
FTP_TIMEOUT	30 000	Period of time (in ms) until the process is called off without a response.

### Debug Mode

DEBUG_WIFI		When defined, it enables the Debug mode.
QUIET	0	No messages printed when wakes up or powers up.
ALL	1	All status messages.
CRITICAL	2	Only critical network AP connection level status is output.
IP_STATUS	4	DHCP and IP address status information.

### FTP Options

#define CHECK_VERSION		// when defined it enables the version checking.
OTA_ver_file	"UPGRADE.TXT"	// Server's file.
#define NO_OTA	"NO_FILE"	// No file in server pattern indicator.

## 4.5. Class Variables

WaspWiFi class has a public variable called `answer` that allows the user to check the last message that WiFi module received last time user called `read`. Besides, there is another variable called 'length' which indicates the length of the last answer.

<code>char answer[513];</code>
<code>int length;</code>

It is a vector of 512 char bytes, so if the user is going to receive less or more data, can adjust this vector on the library, in order to save memory.

## 5. Initialization

Before starting to use a module, it needs to be initialized. During this process, configuration parameters are sent to the module. USB UART connection is enabled to check Debug messages and to display information, and SD card is used to store or upload files when using FTP connections.

### 5.1. Expansion Radio Board

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

Some of the possible combinations are:

- ZigBee - Bluetooth
- WiFi – 802.15.4
- WiFi - RFID
- WiFi - 3G/GPRS
- WiFi – Bluetooth
- Etc

**Remark:** the GPRS and 3/GPRS module do not need the Expansion Board to be connected to Wasp mote. It can be plugged directly in the GPRS socket.

In the next photo you can see the sockets available along with the UART assigned. On one hand, SOCKET0 allows to plug any kind of radio module through the UART0. On the other hand, SOCKET1 permits to connect a radio module through the UART1.

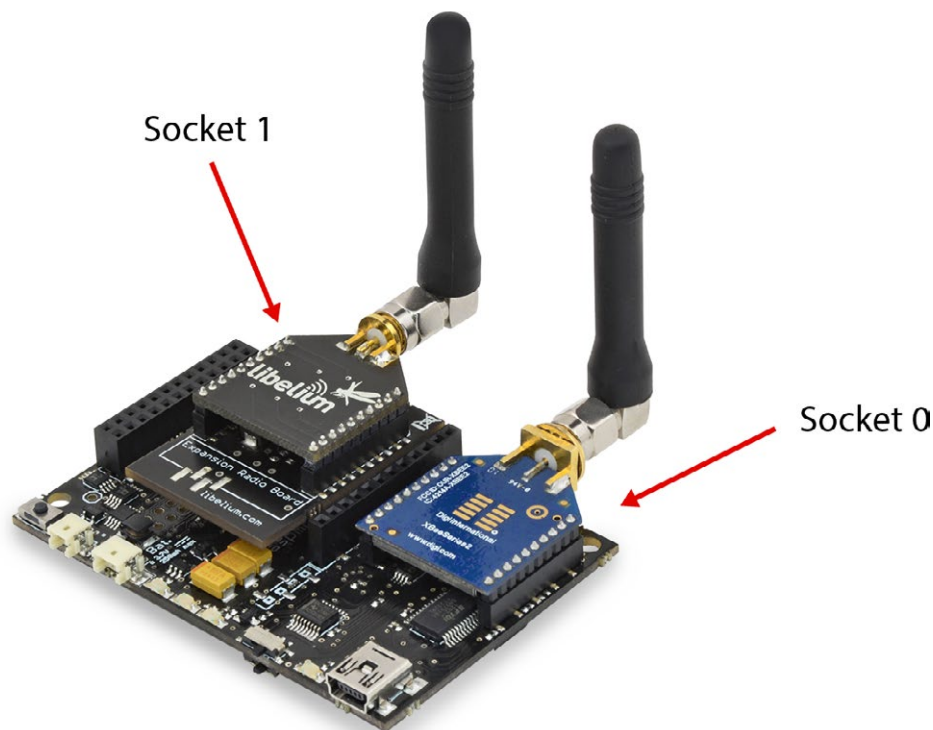


Figure: Use of Expansion Board

The API provides a function called **ON** in order to switch the WiFi module on. This function supports a parameter which permits to select the **SOCKET**. It is possible to choose between **SOCKET0** and **SOCKET1**.

Selecting **SOCKET0**:

```
WIFI.ON(SOCKET0);
```

Selecting **SOCKET1**:

```
WIFI.ON(SOCKET1);
```

The rest of functions are used the same way as they are used with older API versions. In order to understand them we recommend to read this guide.

#### **WARNING:**

- Avoid to use **DIGITAL7** pin when working with Expansion Board. This pin is used for setting the XBee into sleep.
- Avoid to use **DIGITAL6** pin when working with Expansion Board. This pin is used as power supply for the Expansion Board.
- Incompatibility with Sensor Boards:
  - Gases Board: Incompatible with **SOCKET4** and  $\text{NO}_2/\text{O}_3$  sensor.
  - Agriculture Board: Incompatible with Sensirion and the atmospheric pressure sensor.
  - Smart Metering Board: Incompatible with **SOCKET11** and **SOCKET13**.
  - Smart Cities Board: Incompatible with microphone and the CLK of the interruption shift register.
  - Events Board: Incompatible with interruption shift register.

## 5.2. Setting ON

It opens the UART and switches the WiFi ON. The baud rate used to open the UART is defined on the library (9600bps is the factory baudrate; 115200bps is the API baudrate). This function also initializes the USB and SD card connections. USB is used for displaying the data received from the connections (more information on section 8) or for showing debug messages (more on section 10). Finally, it enters in command mode and sets some default configuration.

Example of use:

```
{  
  WIFI.ON(SOCKET0); // Opens the UART and switches the WiFi module on  
}
```

## 5.3. Setting OFF

WiFi API function **OFF** closes the UART and switches the module off.

Example of use:

```
{  
  WIFI.OFF(); // Closes the UART and powers off the module.  
}
```

## 6. Access Point Configuration

This section will describe how an Access Point can be configured in order to make the desired connections. The configurations are saved in the memory of WiFi module, and when the module creates the Access Point, takes those configurations from memory.

It has to be remarked that this information is deleted every time Wasp mote reboots unless the configuration is saved as default in WiFi module using function `storeData` for using automatic associations for example.

Next codes are examples of WiFi Access Point configuration/creation for each type of connection.

### 6.1. Manual Mode

In Manual Mode the user has to specify the name and the security (if exists) of the Access Point.

```
{
  // 1. Configure how to connect the AP.
  WIFI.setJoinMode(MANUAL);

  // 2. If it is manual, call join giving the name of the AP.
  if (WIFI.join("Libelium_AP"))
  {
    ...
  }
}
```

- Connect to Access Point in Manual mode example:  
[www.libelium.com/development/waspmote/examples/wifi-06-connect-ap-manual](http://www.libelium.com/development/waspmote/examples/wifi-06-connect-ap-manual)

### 6.2. Automatic Mode

There are two ways to connect with automatic mode, Auto-Best and Auto-Store.

With **Auto-Best** the WiFi module joins any access point with security matching the stored authentication mode. This ignores the stored SSID and searches for the access point with the strongest signal.

```
{
  // 1. Configure the Authentication mode of the auto-join.
  WIFI.setAutojoinAuth(OPEN);

  // 2. Configure how to connect the AP.
  if (WIFI.setJoinMode(AUTO_BEST))
  {
    ...
  }
}
```

- Connect to Access Point in Auto-Best mode example:  
[www.libelium.com/development/waspmote/examples/wifi-07-connect-ap-auto-best/](http://www.libelium.com/development/waspmote/examples/wifi-07-connect-ap-auto-best/)

And with **Auto-Store**, try to join the access point that matches the stored SSID, passkey and channel. Channel can be set to 0 for scanning.

```
{  
  // 1.1 Configure the authentication mode of the auto-join  
  WIFI.setAutoJoinAuth(OPEN);  
  
  // 1.2 Sets the name of the AP we want to join  
  WIFI.setESSID("libelium_AP");  
  
  // 2. Configure how to connect the AP  
  if (WIFI.setJoinMode(AUTO_STOR))  
  {  
    ...  
  }  
}
```

- Connect to Access Point in Auto-Store mode example:  
**[www.libelium.com/development/waspmote/examples/wifi-08-connect-ap-auto\\_store/](http://www.libelium.com/development/waspmote/examples/wifi-08-connect-ap-auto_store/)**

## 6.3. Examples

- Connect to Access Point in Manual mode:  
**[www.libelium.com/development/waspmote/examples/wifi-06-connect-ap-manual](http://www.libelium.com/development/waspmote/examples/wifi-06-connect-ap-manual)**
- Connect to Access Point in Auto-Best mode:  
**[www.libelium.com/development/waspmote/examples/wifi-07-connect-ap-auto-best/](http://www.libelium.com/development/waspmote/examples/wifi-07-connect-ap-auto-best/)**
- Connect to Access Point in Auto-Store mode:  
**[www.libelium.com/development/waspmote/examples/wifi-08-connect-ap-auto\\_store/](http://www.libelium.com/development/waspmote/examples/wifi-08-connect-ap-auto_store/)**
- Connect to a not encrypted Access Point:  
**[www.libelium.com/development/waspmote/examples/wifi-10-ap-not-encrypted](http://www.libelium.com/development/waspmote/examples/wifi-10-ap-not-encrypted)**
- Connect to a WEP encrypted Access Point:  
**[www.libelium.com/development/waspmote/examples/wifi-11-ap-wep-encrypted](http://www.libelium.com/development/waspmote/examples/wifi-11-ap-wep-encrypted)**
- Connect to a WPA encrypted Access Point:  
**[www.libelium.com/development/waspmote/examples/wifi-12-ap-wpa-encrypted](http://www.libelium.com/development/waspmote/examples/wifi-12-ap-wpa-encrypted)**
- Connect to a WPAMIX encrypted Access Point:  
**[www.libelium.com/development/waspmote/examples/wifi-13-ap-wpamix-encrypted](http://www.libelium.com/development/waspmote/examples/wifi-13-ap-wpamix-encrypted)**
- Connect to a WPA2 encrypted Access Point:  
**[www.libelium.com/development/waspmote/examples/wifi-14-ap-wpa2-encrypted](http://www.libelium.com/development/waspmote/examples/wifi-14-ap-wpa2-encrypted)**

## 7. Network Configuration

This section describes how to configure the network, the WiFi module has to be prepared to join to an access point or to create an Adhoc network.

The most important thing here is to configure the WiFi module address, needed for making connections on the Internet. The WiFi module can set the IP address by calling `WIFI.setDHCPoptions(type)` and the types can be:

- `DHCP_OFF`
- `DHCP_ON`
- `AUTO_IP`
- `DHCP_CACHE`

### 7.1. DHCP protocol

The Dynamic Host Configuration Protocol (DHCP) is a network configuration protocol for hosts on Internet Protocol (IP) networks. Calling this protocol, the WiFi module receives a valid IP address from the access point automatically.

To use this protocol, the module needs to call `WIFI.setDHCPoptions(DHCP_ON)` and then try to join the access point.

- Configure DHCP example:  
[www.libelium.com/development/waspmote/examples/wifi-15-dhcp](http://www.libelium.com/development/waspmote/examples/wifi-15-dhcp)

Another way to solve IP automatically, is calling `WIFI.setDHCPoptions(DHCP_CACHE)`, in this case the WiFi module uses previous IP address if lease is not expired (lease survives reboot).

- Configure auto-cache IP example:  
[www.libelium.com/development/waspmote/examples/wifi-18-auto-cache-ip](http://www.libelium.com/development/waspmote/examples/wifi-18-auto-cache-ip)

### 7.2. IP manual

IP address can also be set manually calling `WIFI.setDHCPoptions(DHCP_OFF)`. In this case you can set as well the gateway address, the net-mask, the DNS address and the local port.

There are functions to configure each address. Next code shows an example of manual network configuration.

```
{
  //////////MANUAL NETWORK CONFIGURATION //////////
  // 1. Configure DHCP off.
  WIFI.setDHCP(DHCP_OFF);
  // 2. Configure the Gateway address
  WIFI.setGW("192.168.4.1");
  // 3. Configure the Net-mask address
  WIFI.setNetmask("255.255.255.0");
  // 4. Configure the local port
  WIFI.setLocalPort(55555);
  // 5. Configure the IP address
  WIFI.setIP("192.168.1.50");
}
```

- Configure static IP:  
[www.libelium.com/development/waspmote/examples/wifi-16-static-ip](http://www.libelium.com/development/waspmote/examples/wifi-16-static-ip)

## 8. Connection Creation

This section will describe how to create/open a connection. There are functions to open or prepare connections for each type of protocol type. Possible connections are:

- Broadcast-UDP
- UDP client-only
- UDP server
- TCP client
- TCP server
- FTP
- HTTP

### 8.1. TCP

TCP is a connection-oriented protocol; the user has to open and close the connection when he finishes to work with it. This protocol is the most reliable, and the user can even configure a password for the connection.

First, the WiFi module configures the protocol with `WiFi.setConnectionOptions(type)`. The types in TCP can be:

- `CLIENT_SERVER`: for TCP server & client
- `CLIENT`: for TCP client only.

And then the user can set the remote parameters in the creation function or with the function:

`WiFi.setRemoteHost(IP, PORT)`.

```
{
  // Configure the transport protocol as TCP server & client
  WiFi.setConnectionOptions(CLIENT_SERVER);
  // Configure the Remote parameters.
  WiFi.setRemoteHost("192.168.1.45", 2000);
}
```

To create the connection, the WiFi module can act as a client, or as a server /client. As a client, the WiFi module specifies the remote address and port and makes the connection. And as a server, the WiFi module specifies the port and waits for a connection on that port.

#### 8.1.1. TCP Client

Next example show how to create a TCP client connection with the function `WiFi.setTCPclient(Remote_IP, Remote_Port, Local_Port)`.

- TCP client example:  
<http://www.libelium.com/development/waspmote/examples/wifi-20a-tcp-client>
- TCP client using Waspote Frame Class to send information:  
<http://www.libelium.com/development/waspmote/examples/wifi-20b-tcp-client-frame>



### 8.1.2. TCP Server

Next example shows how to create a TCP server connection with the function `WIFI.setTCPserver(Local_Port)`.

- TCP server example:

**<http://www.libelium.com/development/waspmote/examples/wifi-21-tcp-server>**

**Note:** *It is not recommended to use this mode in Waspote as the best implementation is to set Waspote as client so as to send Data Frames to a specific server.*

## 8.2. UDP

UDP is a connectionless protocol, for this reason there is not a close method to close the connection. However, with the function `close` the user enters in command mode only and can continue doing other things with the radio WiFi.

There is no initial handshake between the hosts to set up the UDP connection. This makes UDP an unreliable protocol, as there is no guarantee that the data will be correctly delivered. However, UDP is suited for applications that cannot tolerate too much latency but can tolerate some errors in data. Transmission of video and audio would be a good example of UDP application.

### 8.2.1. UDP Client

Next code shows how to create a UDP client connection with the function `WIFI.setUDPclient(Remote_IP, Remote_Port, Local_Port)`.

- UDP client example:

**[www.libelium.com/development/waspmote/examples/wifi-22-udp-client](http://www.libelium.com/development/waspmote/examples/wifi-22-udp-client)**

## 8.2.2. UDP Server

Next code shows how to create a UDP server connection with the function `WIFI.setUDPserver(Local_Port)`.

- UDP server example:  
[www.libelium.com/development/waspmote/examples/wifi-23-udp-server](http://www.libelium.com/development/waspmote/examples/wifi-23-udp-server)

**Note:** It is not recommended to use this mode in Waspote as the best implementation is to set Waspote as client so as to send Data Frames to a specific server.

## 8.3. HTTP

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

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

WiFi module HTTP instances should have the following structure so that our web service can understand.

### 8.3.1. GET method

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

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

Where:

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

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

Some examples:

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

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

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

### 8.3.2. Server response

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

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



Figure: Operating with the web service from a PC browser



Figure: Operating with the web service from a Wasp mote 3G+GPS

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

**[downloads.libelium.com/waspmote-html-get-post-php-parser-tester.zip](https://downloads.libelium.com/waspmote-html-get-post-php-parser-tester.zip)**

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

### 8.3.3. Sending HTTP queries

The Wasp mote WiFi module is capable of sending data to a web server; the user has to specify the address indicating the IP or DNS address, and the get query phrase.

The function `WIFI.getURL(Type, Address, Query)` sends the HTTP get query to the specified server.

**Type:** IP or DNS. To choose between ip address or a DNS name.

**Address:** server address to be specified by a ip address or an url depending on the **Type**.

**Query:** Query sentence with the information to be sent.

Examples:

```
{
  // HTTP query (using IP address)
  WIFI.getURL(IP, "84.20.10.73", "GET$/radioWifitest.php?name=xxx");

  // HTTP query (using DNS)
  WIFI.getURL(DNS, "www.libelium.com", "GET$/radioWifitest.php?name=xxx");
}
```

HTTP query example:

**<http://www.libelium.com/development/waspmote/examples/wifi-26-http-query>**

### 8.3.4. Sending a frame to Meshlium

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

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

The function `WIFI.getURL(Type, Address, Query, frame, length)` sends the HTTP get query to the specified server.

**Type:** IP or DNS. To choose between ip address or a DNS name.

**Address:** server address to be specified by a ip address or an url depending on the **Type**.

**Query:** Query sentence with the information to be sent.

**Frame:** Pointer to the frame buffer

**Length:** Length of the frame to be sent

Examples:

```
{
  // HTTP query (using IP address)
  WIFI.getURL(IP, "10.10.10.1", "GET$/getpost_frame_parser.php?frame=", frame.buffer, frame.length);

  // HTTP query (using DNS)
  WIFI.getURL(DNS, "pruebas.libelium.com", "GET$/getpost_frame_parser.php?frame=", frame.buffer, frame.length);
}
```

HTTP query example with Waspote Frame:

**<http://www.libelium.com/development/waspmote/examples/wifi-26b-http-get-using-frame/>**

## 8.4. FTP

In the FTP protocol, connections are created when uploading or downloading a file from a connection previously configured, and then closed.

FTP connection can be made in 3 easy steps:

- First the WiFi module has to specify the IP address, the port, the mode of the FTP server, and the timeout of the connection with the function `WIFI.setFTP(IP, Port, passive/active, timeout)`.
- Second, the WiFi module has to specify the user name and password of the FTP server with the function `WIFI.openFTP(USER, PASSWORD)`.
- And then, the WiFi module can save a file from the server to the SD card, or upload a file to the server with the functions `WIFI.getFile(filename, folder_sd, folder_ftp)` and `WIFI.uploadFile(filename, folder_sd, folder_ftp)`.

An example of using FTP protocol is showed below.

```
{
  // 1. Configure the Remote parameters (Sets the FTP address, port, etc)
  WIFI.setFTP("62.75.203.94", 21, FTP_PASIVE, 20);

  // 2 Prepares the ftp account with the user name and password
  WIFI.openFTP("USERNAME", "PASSWORD");

  // 3. Now it is able to upload or get a file from the SD card or to a FTP server
  WIFI.uploadFile("FILE", "FOLDER SD CARD", "FOLDER FTP SERVER")
  {
  }
  WIFI.getFile("FILE", "FOLDER SD CARD", "FOLDER FTP SERVER")
  {
  }
}
```

- FTP download example:  
**[www.libelium.com/development/waspmote/examples/wifi-24-ftp-download](http://www.libelium.com/development/waspmote/examples/wifi-24-ftp-download)**
- FTP upload example:  
**[www.libelium.com/development/waspmote/examples/wifi-25-ftp-upload](http://www.libelium.com/development/waspmote/examples/wifi-25-ftp-upload)**

**Note:** Libelium can not secure 100% of file integrity when uploading/downloading using FTP protocol.

## 8.5. Broadcast-UDP

The WiFi module can be setup to automatically generate UDP broadcast packets at certain intervals. This feature can work in parallel with any other protocol before explained. Those packets are sent using UDP protocol.

It is useful for a number of reasons:

- Some AP will disconnect devices that have joined don't send any packets after a time.
- Application programs to auto-discover and auto-connect the WiFi module can use this feature.

The format of the Broadcast-UDP packet is 110 Bytes of data organized as next table shows.

AP MAC ADDR.	CHAN.	RSSI	TCP PORT	REAL TIME CLOCK	TIME OF DAY	DATE CODE	USER DEVICE ID	BOOT TIME
--------------	-------	------	----------	-----------------	-------------	-----------	----------------	-----------

#### Number of Bytes

0-5	6	7	8-9	10-13	18-31	32-59	60-91	92-93
-----	---	---	-----	-------	-------	-------	-------	-------

Next code is an example of Broadcast-UDP connection creation; in the function the user need to specify the broadcast address, the port, the seconds and the User device ID.

- Configure broadcast-UDP messages example:  
[www.libelium.com/development/waspmote/examples/wifi-19-broadcast-udp](http://www.libelium.com/development/waspmote/examples/wifi-19-broadcast-udp)

## 8.6. Sending standard Libelium Frames

This version of WiFi library has the method `WIFI.send(buffer, length)` to send standard Libelium frames in an easy way.

An example is presented next:

```
{
// Create new frame (ASCII)
frame.createFrame(ASCII, "Waspmote_Pro");
// set frame fields (String - char*)
frame.addSensor(SENSOR_STR, (char*) "WiFi frame");

// Sends to the TCP connection
WIFI.send(frame.buffer, frame.length);
}
```

## 9. Device Options

There are some device options when using WiFi module. These options are made to improve speed or to save energy.

### 9.1. Default settings

In terms of settings, the user can restart the module with the default settings with the function [resetValues](#), restart the module with the last saved settings with [reset](#), or save current settings with [storeData](#).

Next table shows default configuration of the WiFi module to be used with the provided API.

Ad-hoc Beacon	100 milliseconds
Ad-hoc probe	5 seconds
IP Broadcast	255.255.255.255
Port Broadcast	55555
Interval Broadcast	7 seconds
DHCP	ON
IP address	0.0.0.0
Net mask	255.255.255.0
Local port	2000
Remote port	2000
Protocol	TCP server and client
Device ID	WiFly-GSX
Join timer	1000 milliseconds
UART Baud rate	9600 (Factory Default) 115200 (Waspote API Default)
SSID	roving1
Channel	0 (automatic)
Join mode	AUTO_STOR
Rate	12 (24 Mbit)
TX Power	0 (12 dBm)
Pass phrase	rubygirl
Key	000000000000

## 9.2. Improving performance

To improve the features or performance of WiFi module it has been created some functions to set the speed of WiFi radio communication with the Waspote, the transmission rate and transmission power.

```
{
  // Sets the baud rate between the radio WiFi and Waspote
  WIFI.setBaudRate(115200);

  // Sets the transmission rate with the network
  WIFI.setTXRate(15);

  // Sets the power of transmission
  WIFI.setTXPower(0);

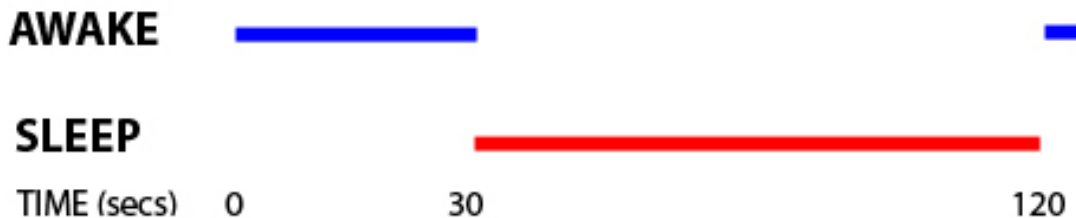
  // Sets the timeout in milliseconds of joining to a network
  WIFI.setJoinTime(100);
}
```

## 9.3. Sleeping the radio WiFi

The Radio WiFi uses the Real Time Clock (RTC) to generate timers. The RTC is active even when radio WiFi is asleep. This makes it possible to set the module to sleep and wake up from sleep based on timer intervals. During the sleep time, any sent to radio WiFi over the UART will not be processed.

That feature may be done automatically with `setSleep( sTime, sCycle)` function. When radio WiFi reboots, this configuration will remain the radio WiFi awake during `sCycle` seconds and then it will sleep the radio during `sTime` seconds. The user can configure default connections when waking up.

The next example shows how the module works when `sTime=90` and `sCycle=30`.



- Sleep and Wake up the module automatically:  
[www.libelium.com/development/waspote/examples/wifi-04-sleep-auto](http://www.libelium.com/development/waspote/examples/wifi-04-sleep-auto)

**Note:** It is not recommended to use this mode. It is better to switch on and off the module in order to save battery as it is not possible to know when the module is awake or not.



## 10. Status Information

Finally, the last group of functions are the ways for watching the current status and the current configuration of WiFi module. This information is displayed through the USB connection and stored in the `WIFI.answer` variable.

These options are: connection, network, signal quality, statistics, delays, ad-hoc, broadcast, communication, DNS, FTP, IP, MAC, device ID, timers, UDP and UART settings.

- Display module status example:  
[www.libelium.com/development/waspmote/examples/wifi-03-status](http://www.libelium.com/development/waspmote/examples/wifi-03-status)

## 11. Connecting to Meshlium

Meshlium is a Linux router which works as the Gateway of the Waspote 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 waterproof** (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.



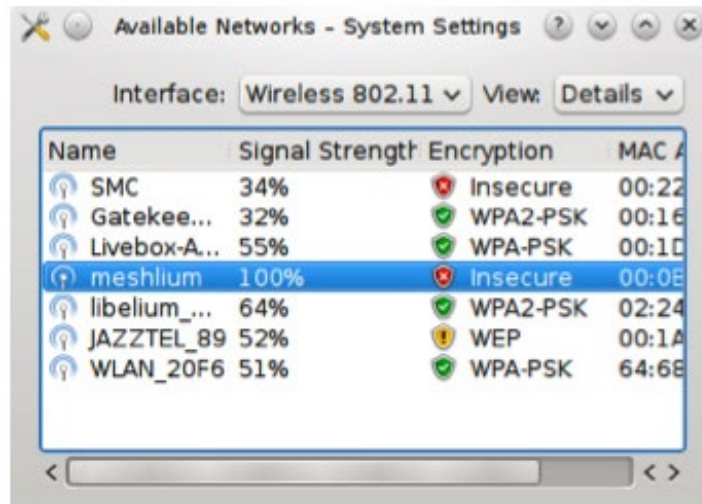
More info: <http://www.libelium.com/meshlium>

Focusing in WiFi technologies, there are two main ways of using Meshlium and Waspote with the WiFi module. First is using Meshlium to create an Access Point and give the WiFi module access to the Internet. And the other is configuring Meshlium for receiving, capturing and sending messages from and to Waspote with WiFi module.

## 11.1. Setting Meshlium as AP

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 \*.

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

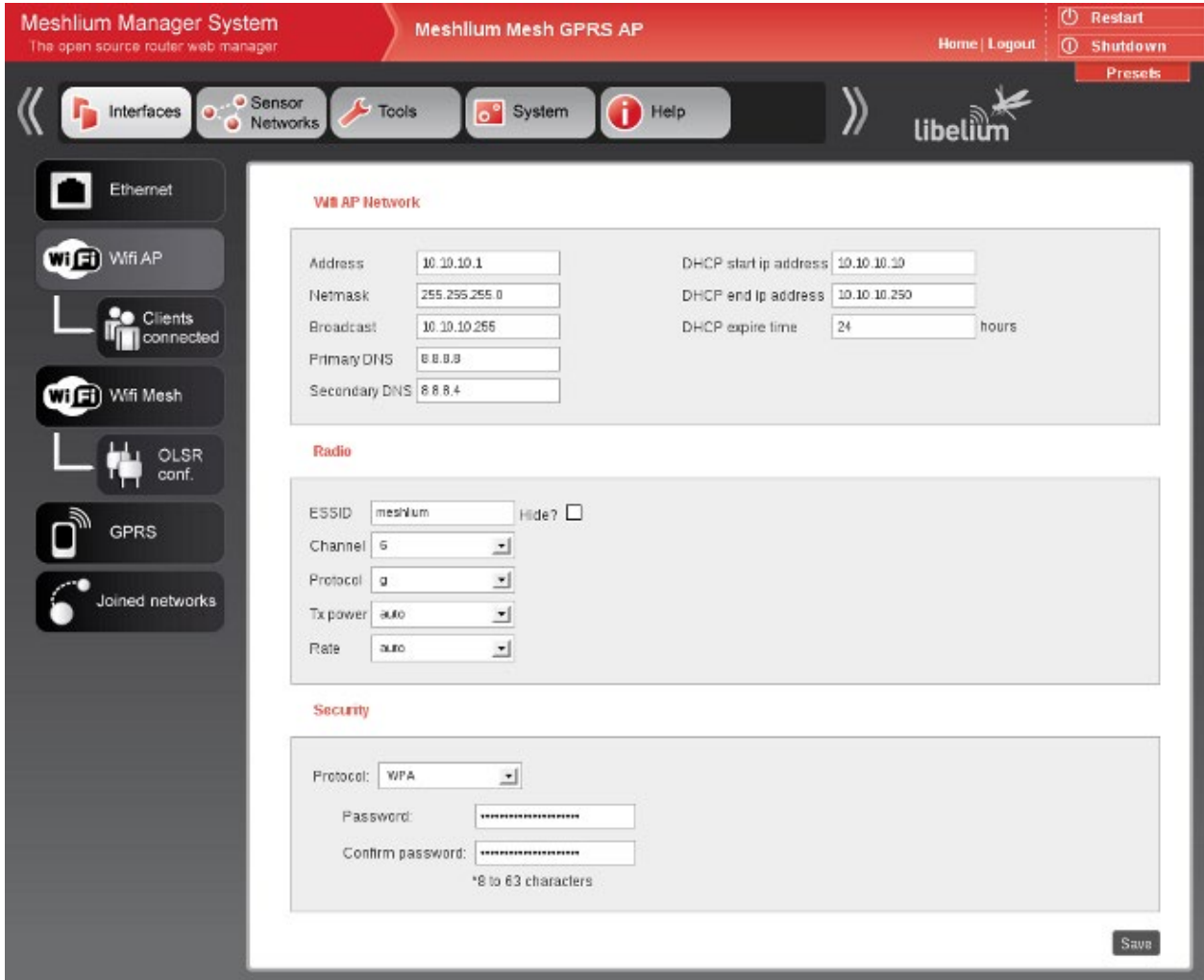


No password is needed as the network is public (you can change it later in the WiFi AP Interface options). When you select it, Meshlium will give an IP from the range 10.10.10.10 - 10.10.10.250.

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

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

In the manager system, Meshlium AP can be configured by giving it a name and providing it the security desired. The user can also configure a DHCP server to give IP addresses to the radio WiFi. To do this, click in Interfaces, and then in WiFi AP. Now you will see next image:



The screenshot shows the Meshlium Manager System web interface. The top navigation bar includes 'Home | Logout', 'Restart', 'Shutdown', and 'Presets' buttons. The main navigation menu on the left includes 'Interfaces', 'Sensor Networks', 'Tools', 'System', and 'Help'. The 'Interfaces' section is expanded, showing 'Ethernet', 'WiFi AP', 'Clients connected', 'WiFi Mesh', 'OLSR conf.', 'GPRS', and 'Joined networks'. The 'WiFi AP' option is selected, leading to the 'WiFi AP Network' configuration page. This page has three main sections: 'WiFi AP Network', 'Radio', and 'Security'. The 'WiFi AP Network' section contains fields for Address (10.10.10.1), Netmask (255.255.255.0), Broadcast (10.10.10.255), Primary DNS (8.8.8.8), Secondary DNS (8.8.8.4), DHCP start ip address (10.10.10.10), DHCP end ip address (10.10.10.250), and DHCP expire time (24 hours). The 'Radio' section contains fields for ESSID (meshlium), Channel (6), Protocol (g), Tx power (auto), and Rate (auto). The 'Security' section contains a Protocol dropdown (WPA), Password, and Confirm password fields, with a note '\*8 to 63 characters'. A 'Save' button is located at the bottom right of the form.

If your network does not offer DHCP service Meshlium starts with a default IP (192.168.1.100). In this case you can connect Meshlium through the WiFi connection (which is always available) or with the crossover cable provided with Meshlium. If you want to access to the Manager System using the crossover ethernet cable go to:

- **URL:** <http://192.168.1.100/ManagerSystem>
- **user:** root
- **password:** libelium

Then, when the WiFi radio connects this AP, the Meshlium can act like an Internet Router and provide Internet access to the Waspote network, by the Ethernet or the 3G configuration. Once connected, Waspote can create any kind of connection to the Internet (see section 8) through Meshlium using external IP addresses.

Meshlium is a complete Linux system. This means you can prepare it to execute any kind of application including PHP, Python, Perl, Ajax, Ruby, C, C++, Java, etc and services: http, ftp(not installed by default), mysql, postgre, etc.

Meshlium counts with a file system up to 32GB which can be used by developers to store the data generated in their own applications and with the sensor data coming from the ZigBee, Bluetooth and GPS capturer tools. You can access through SSH using the same user and password of the Manager System.

- [illegible]

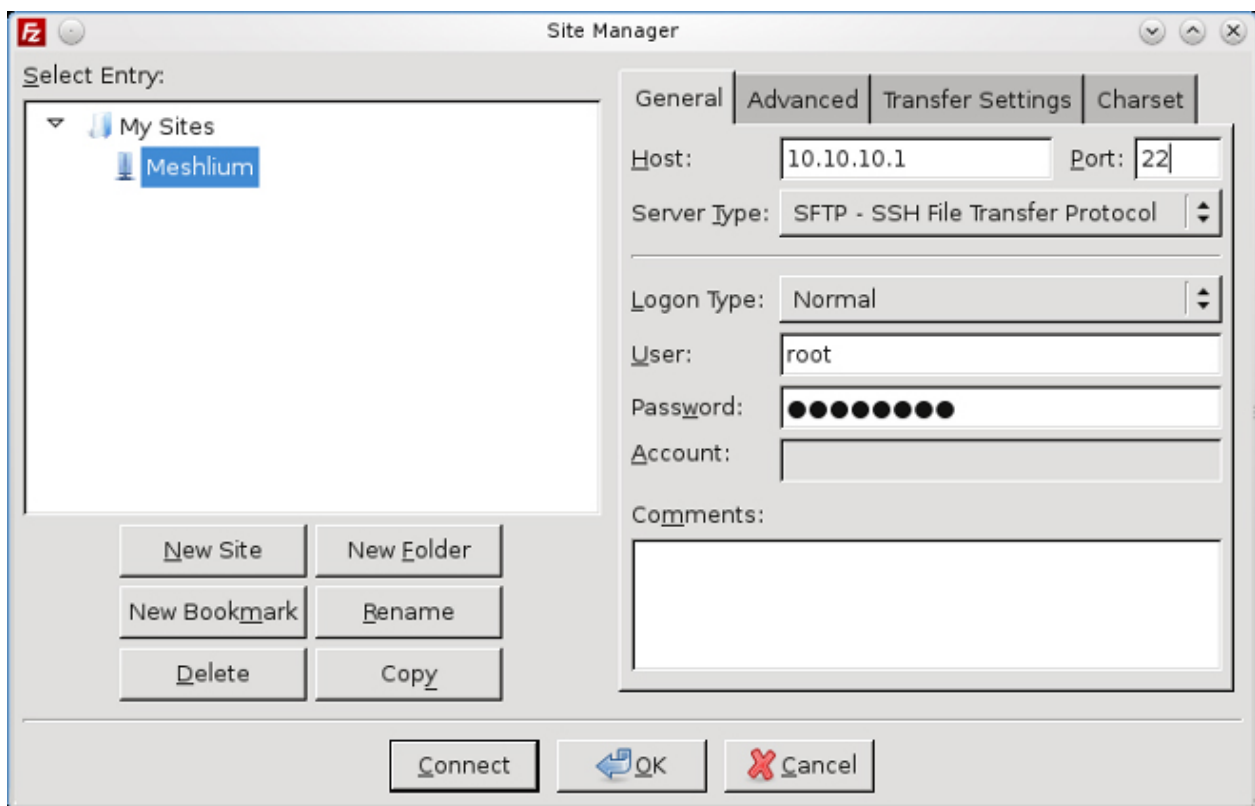
#remountrw

You can use “**scp**” (available by default in most linux distributions) in order to copy your own files to Meshlium.

```
#scp myFile root@10.10.10.1:/mnt/user/.
```

Alternatively you can use a sftp client program, like **Filezilla**, (available for windows and mac) and use it from you pc to upload files to Meshlium using the following data:

- **Host:** 10.10.10.1
- **Server Type:** SFTP
- **Port:** 22
- **User:** root
- **Passw:** libelium



Last you must remount the system in read only mode. To do so execute as root: “**remountro**”.

```
#remountro
```

**Important:** Installing new software is a delicate task which has to be done for experts. If you are not sure about adding new files to the system path do not perform any action including remounting the system as read and write. If you have any doubt ask to our Development team at: <http://www.libelium.com/forum>

All the information coming from all the interfaces (ZigBee, Bluetooth, 3G/GPRS, **WiFi** and from the GPS module) can be stored in the Local Data Base as explained in the “Storage Options” section or even exported to an external Data Base connected to the internet.

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

```
main()
{
    /* file descriptors */
    int fd;
    int i;
    struct sockaddr_in server;
    struct sockaddr_in client;
    int sin_size;
    int numbytes;
    char buf[MAXDATASIZE];

    /* create a socket() */
    if ((fd=socket(AF_INET, SOCK_DGRAM, 0)) == -1 )
    {
        printf("socket error\n");
        exit(-1);
    }
    /* create the struct sockaddr_in with the addresses/settings */
    sin_size = sizeof(client);
    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);
    server.sin_addr.s_addr = INADDR_ANY;
    bzero(&(server.sin_zero), 8);
    /* bind() the socket to a port */
    if (bind(fd, (structsockaddr*)&server, sizeof(structsockaddr)) == -1)
    {
        printf("bind error \n");
        exit(-1);
    }
    //HERE WHATEVER YOU WANT TO DO: STORE, RECEIVE, ANSWER...
    /* receive data from the socket */
    recvfrom(fd, buf, MAXDATASIZE, 0, (structsockaddr*)&client, &sin_size);
    /* send data to the socket */
    sendto(fd, ...);
    /* closes the socket */
    close(fd);
}
```

This example is for UDP connection, if the user wants to make a TCP connection, there are some changes in the code, for example, function socket changes and the functions `listen()` and `accept()` are needed.

```
/* creates TCP socket */
fd = socket(AF_INET, SOCK_STREAM, 0) ;
/* bind() the socket to a port */
bind(fd, (structsockaddr*)&server, sizeof(structsockaddr));
/* listen for connections */
listen(fd, BACKLOG) == -1);
/* accept and establish a TCP connection */
fd2 = accept(fd, (structsockaddr *)&client, &sin_size));
/* receive data from the socket */
recv(fd2, buf, MAXDATASIZE, 0));
/* send data to the socket*/
send(fd2, "whatever.");
```

The real files (*wifi\_UDP.c*, *wifi\_TCP.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 configuration.

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

```
#gcc wifi_UDP.c -o wifiUDP  
#gcc wifi_TCP.c -o wifiTCP
```

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

Finally, **execute** one of these programs.

```
#./wifiUDP  
#./wifiTCP
```

And next image shows what you should see:

```
meshlium:~/pruebasjoki# ./wifiTCP  
Connection from: 192.168.1.170  
Wasmote Message: *HELLO*  
Wasmote Message: TCP - Hi from Wasmote through WIFI!
```



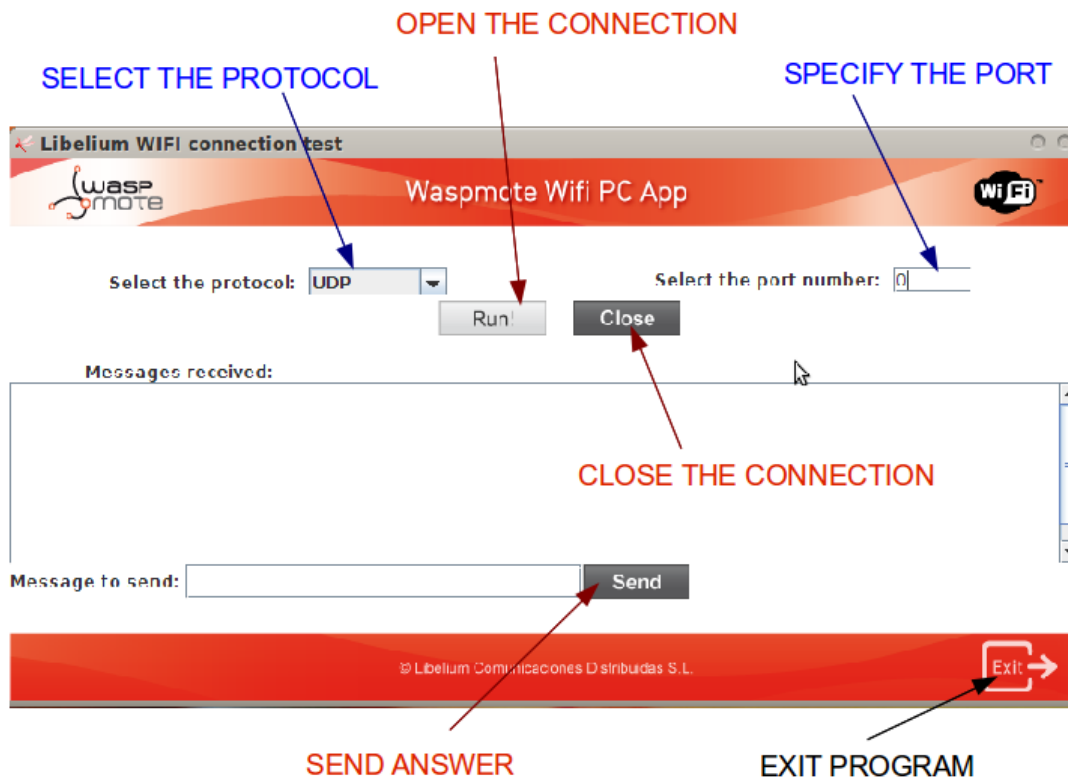
## 12. Connecting to a PC

In this section we will see how to use the **Waspote WiFi PC App** to communicate with the WiFi radio easily from any PC, and test your WiFi Waspote programs. There are three versions of the program: Windows (executable program), Mac (application program) and Linux (Java executable program).

For the **installation**, you just need to download the latest version of the **Radio WiFi PC-test** from the Development section: <http://www.libelium.com/development/waspote>

The screen of the program is divided in 3 parts:

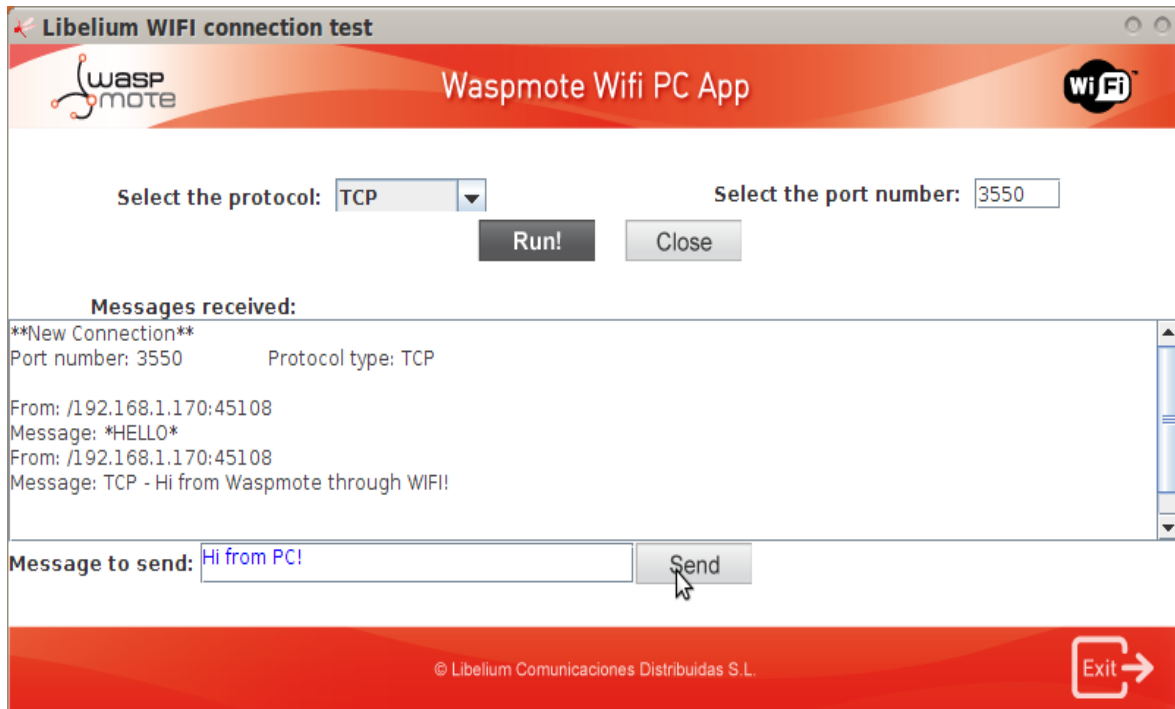
- First part corresponds to the configuration and the creation/close of the connection.
- In the second part there are two text areas, first one displays information and messages received, and the second text area corresponds to the message that will be send when the user presses send button.
- Finally, in the bottom of the screen there is a button to finish the program.



Use of the program:

First, launch the program: double click on the icon, or right click and open.

- First select TCP protocol, introduce port number 3550 and click **Run!**
- Once pressed run button, messages from the WiFi module will be received
- When the first message is received, **Send** button is enabled and the user can send a message back to the radio WiFi
- Fill the text area in the bottom of the window with the message to send, and press **Send**
- Press **Close** to finish the connection securely



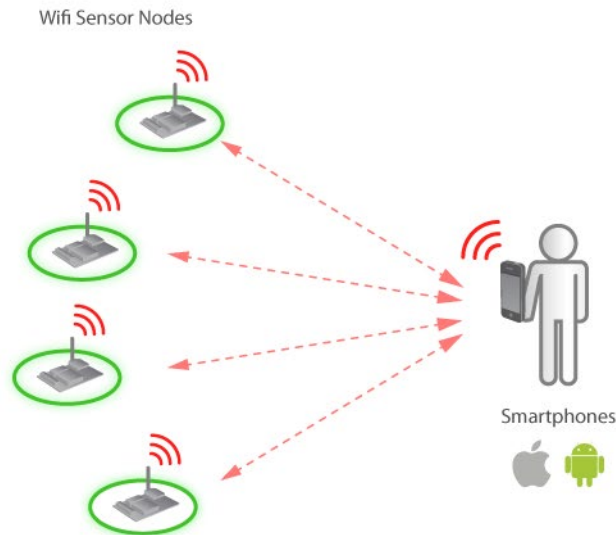
-Message to send specified-

Waspmote code example:

- PC App example (UDP):  
[www.libelium.com/development/waspote/examples/wifi-30a-pc-app-udp](http://www.libelium.com/development/waspote/examples/wifi-30a-pc-app-udp)
- PC App example (TCP):  
[www.libelium.com/development/waspote/examples/wifi-30b-pc-app-tcp](http://www.libelium.com/development/waspote/examples/wifi-30b-pc-app-tcp)
- PC App example (broadcast-UDP):  
[www.libelium.com/development/waspote/examples/wifi-30c-pc-app-broadcast-udp](http://www.libelium.com/development/waspote/examples/wifi-30c-pc-app-broadcast-udp)

## 13. 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 **Waspote 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/waspote-wifi/id501974230>
- Android: [https://play.google.com/store/apps/details?id=com.libelium.WIFI\\_module2](https://play.google.com/store/apps/details?id=com.libelium.WIFI_module2)



### 13.1. Connecting to an iPhone

#### 13.1.1. Installation

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

- From the iPhone, go to App Store. Go to Search screen and search "Waspote WiFi".
- Select Waspote 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 "Waspote WiFi":  
<http://itunes.apple.com/us/app/waspote-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.



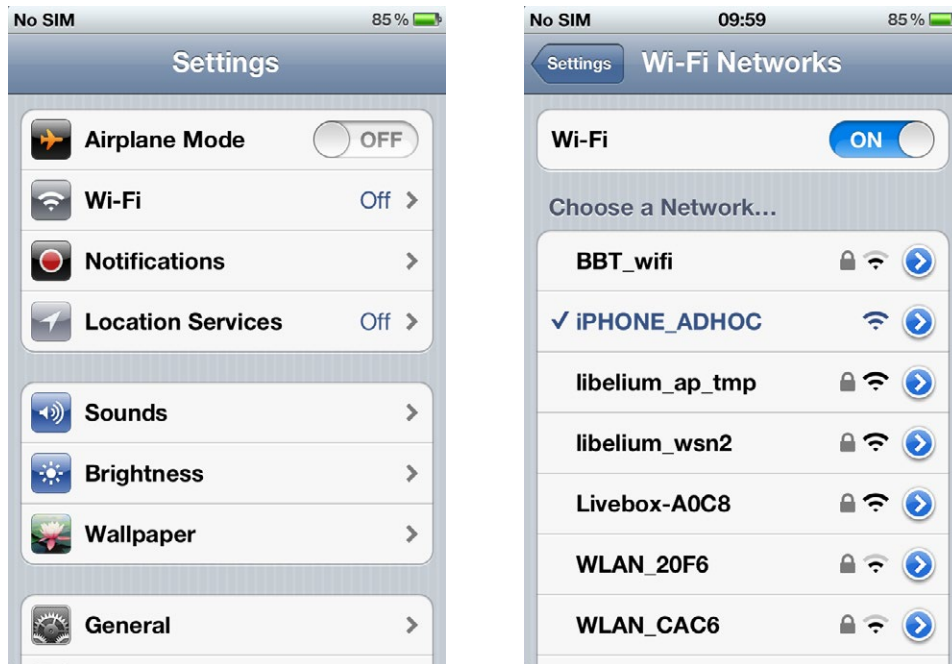
Once installed, the app appears in your iPhone/iPod screen:



### 13.1.2. iPhone App tutorial

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

To connect to the network created by the WiFi module of Wasp mote: Go to Settings → Wi-Fi and select iPhone\_ADHOC.

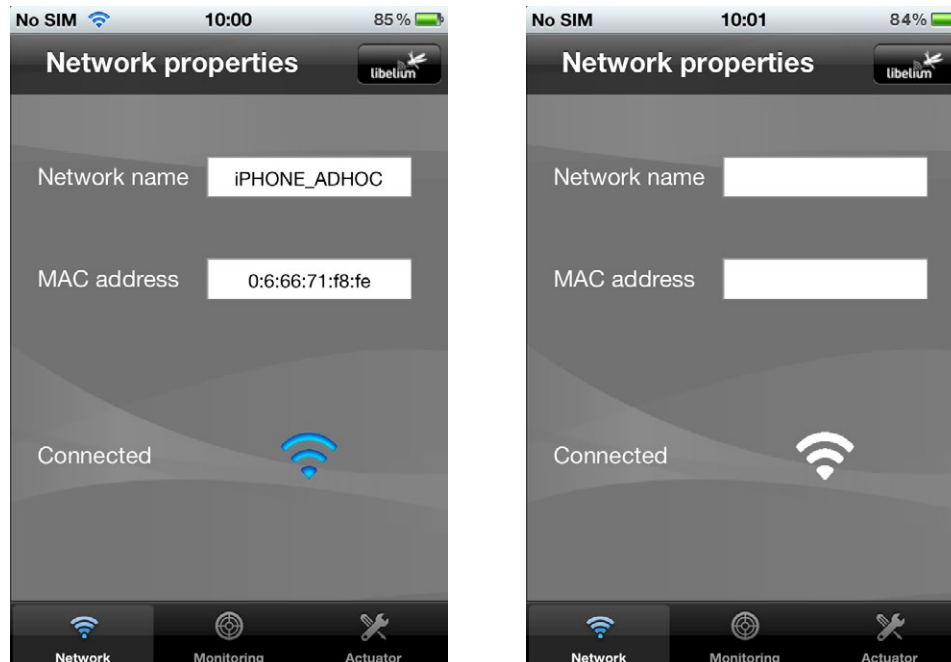


Once connected, you will see a blue WiFi icon in the top of the screen:

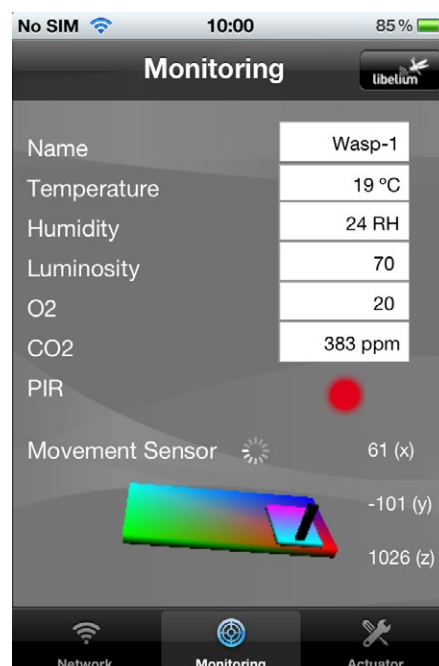


Inside the app, the first tab "Network" shows the information of the connection:

- 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<sub>2</sub>, CO<sub>2</sub>, PIR and a 3D model of Wasp mote 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 Waspote code used in this program can be downloaded from the Libelium website:  
<http://www.libelium.com/development/waspote>

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 Sales Department at: [commercial@libelium.com](mailto:commercial@libelium.com)

## 13.2. 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 Waspote to connect to it.

### 13.2.1. Installation

a) Download the application from Google Play:

- From the Android device, go to Google Play.
- Search "Libelium" or "Waspote WiFi" and press enter.



b) Download the application (WaspoteWifi.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.

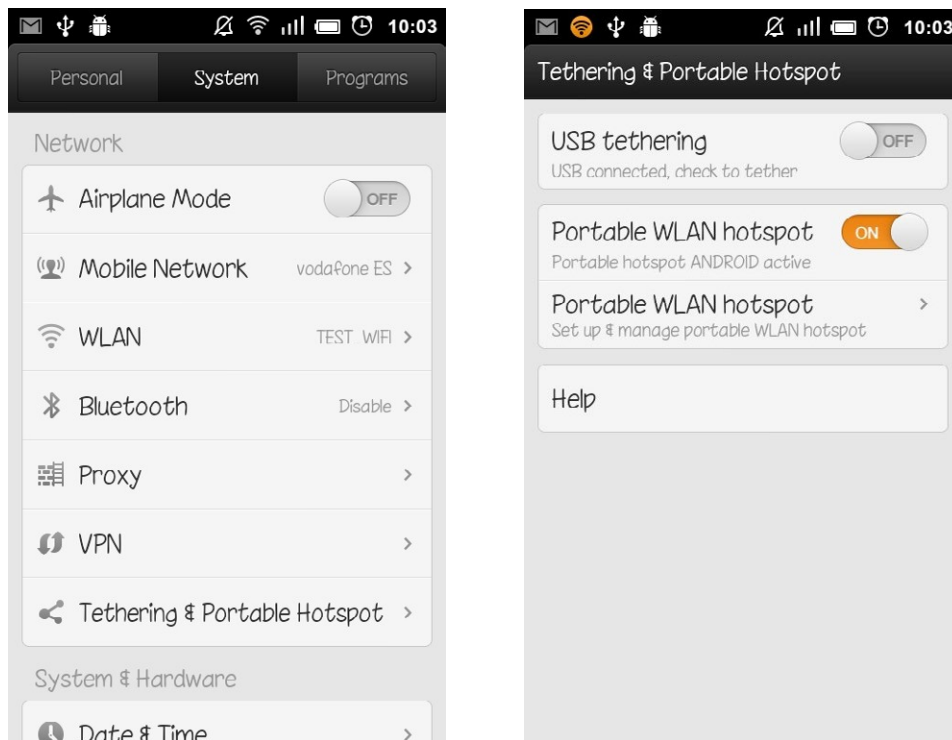


### 13.2.2. Android App tutorial

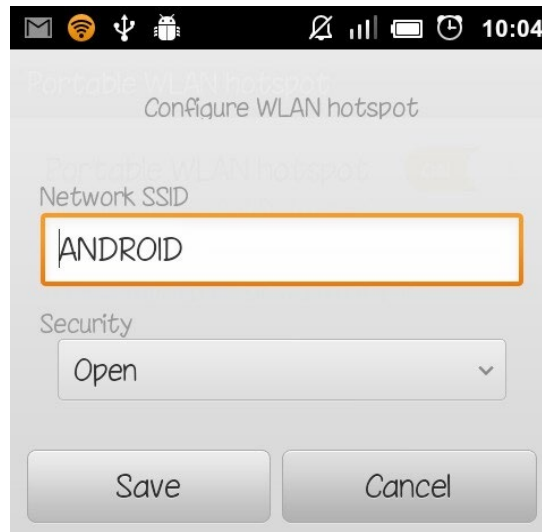
The use of the app is very simple, first you have to create an AP from your Android device and then set Waspote to connect to it.

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)



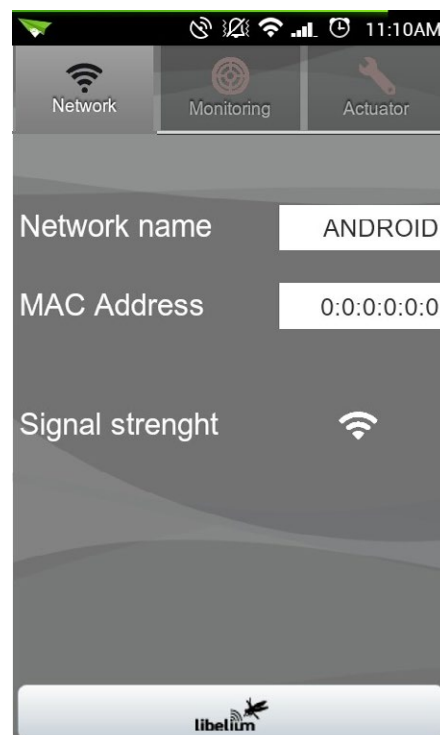
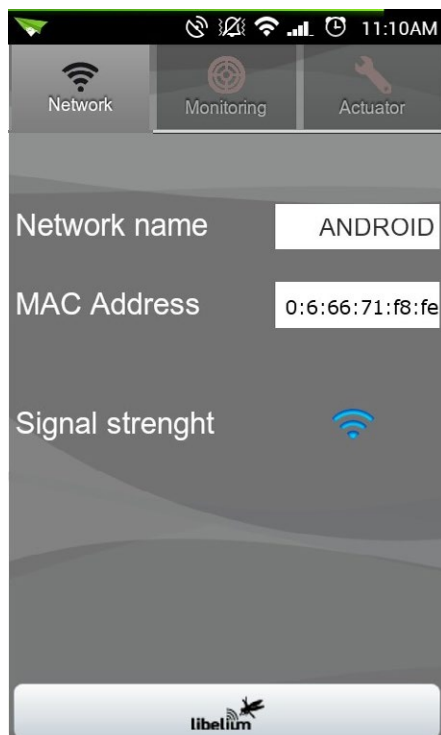
Then configure the WLAN hotspot (name= ANDROID, Security= Open). This settings you can change if you change as well the Wasp mote code.



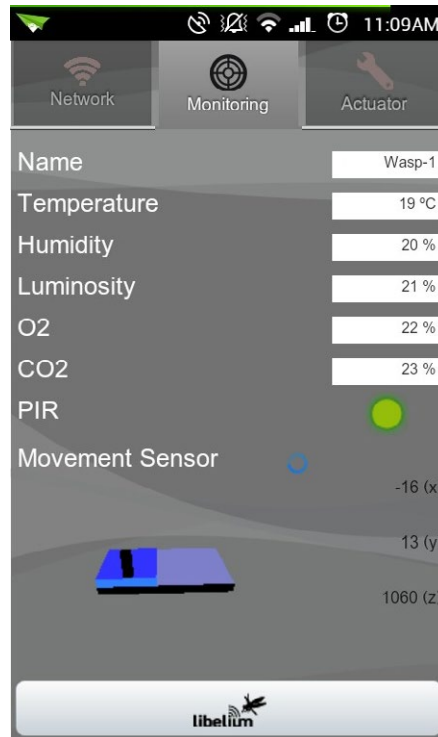
Finally, enable Portable WLAN hotspot (or My WiFi Zone), and Wasp mote will connect to the Android device. Once connected, you can launch the Wasp mote WiFi Demo app.

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

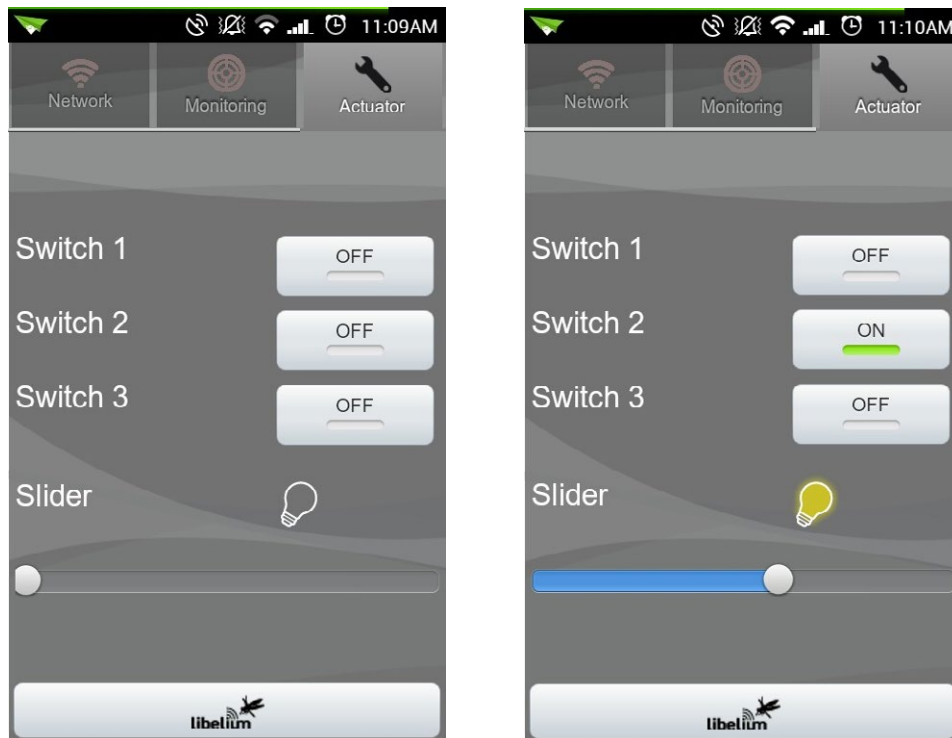
- Name of the Adhoc network
- MAC address of the node acting as gateway
- Connection status



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<sub>2</sub>, CO<sub>2</sub>, PIR and a 3D model of Wasp mote 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 Waspote code used in this program can be downloaded from the Libelium website:

**<http://www.libelium.com/development/waspote>**

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 Sales Department at: **[commercial@libelium.com](mailto:commercial@libelium.com)**

*Figure: Examples*

- iPhone App example:  
**[www.libelium.com/development/waspote/examples/wifi-29a-iphone](http://www.libelium.com/development/waspote/examples/wifi-29a-iphone)**
- Android App example:  
**[www.libelium.com/development/waspote/examples/wifi-29b-android](http://www.libelium.com/development/waspote/examples/wifi-29b-android)**

## 14. Code examples and extended information

- For more information about the Waspote hardware platform go to:  
<http://www.libelium.com/waspmote>  
<http://www.libelium.com/development/waspmote>
- In the Waspote Development section you can find complete examples:  
<http://www.libelium.com/development/waspmote/examples>

### Example:

```
#include <WaspWiFi.h>          // Include WiFi library
#include <WaspFrame.h>         // Include Frame library

// choose socket (SELECT USER'S SOCKET)
uint8_t socket=SOCKET0;

// TCP server settings
#define IP_ADDRESS "???.???.???.???"
#define REMOTE_PORT 55557
#define LOCAL_PORT 2000

// WiFi AP settings (CHANGE TO USER'S AP)
#define ESSID "libelium_AP"
#define AUTHKEY "password"

void setup()
{
  // Switch ON the WiFi module on the desired socket
  WIFI.ON(socket);

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(CLIENT);

  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);

  // 3. Configure how to join the AP
  WIFI.setJoinMode(MANUAL);

  // 4. Set Authentication key
  WIFI.setAuthKey(WPA1, AUTHKEY);

  // 5. Store values
  WIFI.storeData();
}

void loop()
{
  // 1. Switch WiFi ON
  WIFI.ON(socket);

  // 2. Join AP
  if (WIFI.join(ESSID))
  {
    USB.println("Connected to AP");
  }

  // 3. Call the function to create a TCP connection
  if (WIFI.setTCPclient(IP_ADDRESS, REMOTE_PORT, LOCAL_PORT))
```

```
{
  // Connection is open, get values
  USB.println(F("TCP client set"));
  RTC.ON();
  RTC.getTime();

  // Create new frame (ASCII)
  frame.createFrame(ASCII,"Waspmote_Pro");
  // set frame fields
  frame.addSensor(SENSOR_BAT, PWR.getBatteryLevel() );
  frame.addSensor(SENSOR_TIME, RTC.hour, RTC.minute, RTC.second );

  // Sends to the TCP connection
  WIFI.send(frame.buffer,frame.length);

  // Closes the TCP connection.
  USB.println(F("Close TCP socket"));
  WIFI.close();
}
else
{
  USB.println(F("TCP client NOT set"));
}
}
else
{
  USB.println(F("NOT Connected to AP"));
}

// Switch off the module
WIFI.OFF();
USB.println(F("-----"));
delay(3000);
}
```

## 15. API Changelog

Function / File	Changelog	Version
WaspWiFi::commandMode	internal change in function	v009 → v010
WaspWiFi::setChannel	internal change in function	v009 → v010
WaspWiFi::setJoinMode	internal change in function	v009 → v010
WaspWiFi::setAuthKey	internal change in function	v009 → v010
WaspWiFi::setAPretries	internal change in function	v009 → v010
WaspWiFi::scan	internal change in function	v009 → v010
WaspWiFi::setESSID	internal change in function	v009 → v010
WaspWiFi::setTXRate	internal change in function	v009 → v010
WaspWiFi::setTXPower	internal change in function	v009 → v010
WaspWiFi::setIPWindow	internal change in function	v009 → v010
WaspWiFi::join	internal change in function	v009 → v010
WaspWiFi::joinAPnum	internal change in function	v009 → v010
WaspWiFi::setJoinTime	internal change in function	v009 → v010
WaspWiFi::sendPing	internal change in function	v009 → v010
WaspWiFi::setTimeFromServer	internal change in function	v009 → v010
WaspWiFi::setSleep	internal change in function	v009 → v010
WaspWiFi::setDebugMode	internal change in function	v009 → v010
WaspWiFi::setGW	internal change in function	v009 → v010
WaspWiFi::setNetmask	internal change in function	v009 → v010
WaspWiFi::setRemoteHost	internal change in function	v009 → v010
WaspWiFi::setLocalPort	internal change in function	v009 → v010
WaspWiFi::setIp	internal change in function	v009 → v010
WaspWiFi::setDHCPoptions	internal change in function	v009 → v010
WaspWiFi::setIPOptions	internal change in function	v009 → v010
WaspWiFi::setConnectionOptions	internal change in function	v009 → v010
WaspWiFi::setTCPpassw	internal change in function	v009 → v010
WaspWiFi::setDNS	internal change in function	v009 → v010
WaspWiFi::setFTP	internal change in function	v009 → v010
WaspWiFi::openFTP	internal change in function	v009 → v010
WaspWiFi::getFile	internal change in function	v009 → v010
WaspWiFi::uploadFile	internal change in function	v009 → v010
WaspWiFi::getURL	internal change in function	v009 → v010
WaspWiFi::setTCPclient	internal change in function	v009 → v010
WaspWiFi::setTCPserver	internal change in function	v009 → v010

<code>WaspWiFi::setUDPClient</code>	internal change in function	v009 → v010
<code>WaspWiFi::setUDPserver</code>	internal change in function	v009 → v010
<code>WaspWiFi::sendAutoBroadcast</code>	internal change in function	v009 → v010
<code>WaspWiFi::setAdhocOptions</code>	internal change in function	v009 → v010
<code>WaspWiFi::resolve</code>	internal change in function	v009 → v010
<code>WaspWiFi::setBaudRate</code>	internal change in function	v009 → v010
<code>WaspWiFi::setCommSize</code>	internal change in function	v009 → v010
<code>WaspWiFi::setCommTimer</code>	internal change in function	v009 → v010
<code>checkAnswer()</code>	internal change in function	v006 → v007
<code>readData()</code>	internal change in function	v006 → v007
<code>openHTTP()</code>	internal change in function	v006 → v007
<code>scan()</code>	internal change in function	v006 → v007
<code>sendPing()</code>	internal change in function	v006 → v007
<code>uploadFile()</code>	internal change in function	v006 → v007
<code>getURL()</code>	internal change in function	v006 → v007
<code>read()</code>	internal change in function	v006 → v007
<code>reset()</code>	internal change in function	v006 → v007
<code>resolve()</code>	internal change in function	v006 → v007
<code>isConnected()</code>	internal change in function	v006 → v007
<code>getConnectionInfo()</code>	internal change in function	v006 → v007
<code>getAPstatus()</code>	internal change in function	v006 → v007
<code>getRSSI()</code>	internal change in function	v006 → v007
<code>getStats()</code>	internal change in function	v006 → v007
<code>getUpTime()</code>	internal change in function	v006 → v007
<code>getAdhocSettings()</code>	internal change in function	v006 → v007
<code>getBroadcastSettings()</code>	internal change in function	v006 → v007
<code>getComSettings()</code>	internal change in function	v006 → v007
<code>getDNSsettings()</code>	internal change in function	v006 → v007
<code>getFTPsettings()</code>	internal change in function	v006 → v007
<code>getIP()</code>	internal change in function	v006 → v007
<code>getMAC()</code>	internal change in function	v006 → v007
<code>getOptionSettings()</code>	internal change in function	v006 → v007
<code>getSystemSettings()</code>	internal change in function	v006 → v007
<code>getTime()</code>	internal change in function	v006 → v007
<code>getWLANsettings()</code>	internal change in function	v006 → v007
<code>getUARTsettings()</code>	internal change in function	v006 → v007
<code>getVersion()</code>	internal change in function	v006 → v007



<code>int length;</code>	Class variable created	v002 → v003
<code>#define WIFI_BAUDRATE 115200</code>	New baudrate definition. Now the module works at 115200bps	v002 → v003
<code>#define FTP_TIMEOUT 30000</code>	New FTP timeout definition	v002 → v003
<code>#define CHECK_VERSION</code>	New check OTA version definition	v002 → v003
<code>#define OTA_ver_file "UPGRADE.TXT"</code>	New OTA default server file name definition	v002 → v003
<code>#define NO_OTA "NO_FILE"</code>	New OTA default no file in server definition	v002 → v003
<code>WaspWiFi::wake</code>	Function deleted	v002 → v003
<code>WaspWiFi::sleep</code>	Function deleted	v002 → v003
<code>uint8_t waitForData(int numBytes, unsigned long timeout);</code>	New function prototype	v002 → v003
<code>int findPattern(uint8_t* input, char* pattern, int len);</code>	New function to seek strings inside array of bytes	v002 → v003
<code>int read(uint8_t blo, unsigned long time);</code>	New function prototype with timeout input	v002 → v003
<code>int8_t requestOTA();</code>	New function to request Over The Air Programming to an FTP server	v002 → v003
<code>WaspWiFi::begin()</code>	Method deleted	V0.31 → v001
<code>#define USB_RATE</code>	Define deleted	v0.31 → v001
<code>#define UART_RATE</code>	Define deleted	v0.31 → v001
<code>int baud_rate</code>	New private variable that specifies the current baud_rate	v0.31 → v001
<code>WaspWiFi::ON(uint8_t sock)</code>	This method has been changed due to changes in the hardware.	v0.31 → v001
<code>uint8_t _usb</code>	Variable deleted	v0.31 → v001
<code>uint8_t _uartWiFi</code>	New variable that specifies the socket the module is plugged in.	v0.31 → v001
All public functions	All public methods call <code>beginSerial(baud_rate, _uartWiFi);</code> method at the beginning due to changes in the hardware.	v0.31 → v001
<code>WaspWiFi::send(uint8_t* data, uint16_t length)</code>	New method added to handle the new Libelium standard frame.	v0.31 → v001

## 16. Documentation changelog

### **From v4.3 to v4.4:**

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

### **From v4.2 to v4.3:**

- API changelog updated to API v007
- Added section to explain Libelium's HTTP web service

### **From v4.1 to v4.2:**

- Added reference to OTA with WiFi module via FTP server

### **From v4.0 to v4.1:**

- Added references to 3G/GPRS Board in section: Expansion Radio Board