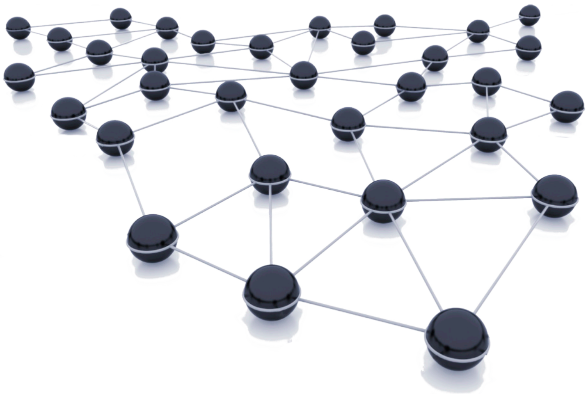
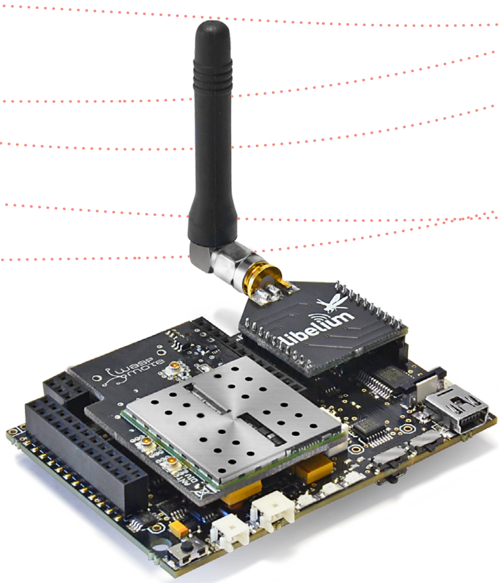
WiFi-PRO Module

Networking Guide





Document version: v7.1 - 07/2017

INDEX

1. Introduction

This guide explains the WiFi PRO features and functions. This module has been integrated into our new product lines Waspmote v15 and Plug & Sense! v15, released on October 2016.

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

You can get more information about the generation change on the document "New generation of Libelium product lines".

The WiFi PRO module offers and supports large variety of features which among them:

- Ten simultaneous TCP/UDP sockets
- DHCP client/server
- DNS client
- HTTP client
- HTTPS client
- FTP client
- NTP client
- Multiple SSIDs
- Roaming mode
- OTA feature. Refer to Over the Air Programming Guide for more information.

The WiFi PRO module supports the SSL3/TLS1 protocol for secure sockets. On the WLAN interface it supports WEP, WPA and WPA2 WiFi encryption.

> ***Note:*** *Working on bands shared with many WiFi devices may cause poor quality of service. Use the WiFi channels less populated so that the sensor nodes may work with no interference.*

Important:

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

1.1. Connect to a standard router

Sensor nodes may connect to any **standard router** which is configured as Access Point (AP) and then send data to other devices in the same network such as laptops and smart phones. 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, so the user needs to specify the key to the WiFi PRO module.

Nodes may also connect to a standard WiFi router equipped with DSL or cable connectivity and send data to a web server located on the Internet. Thus users are allowed to get this data from the Cloud.

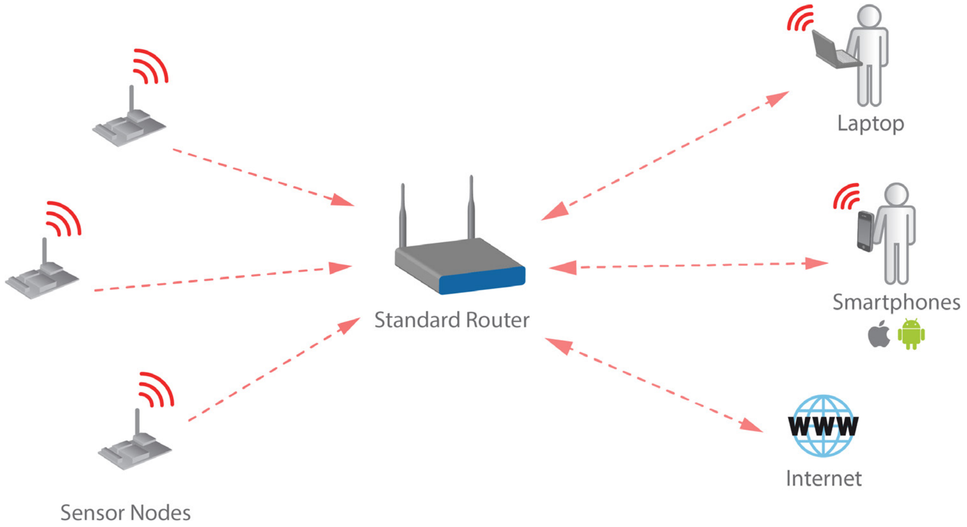


Figure: Standard WiFi router

1.2. Connect to a Meshlium

Instead of using a standard WiFi router as AP, the connection may be performed using a **Meshlium** device as AP. 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.

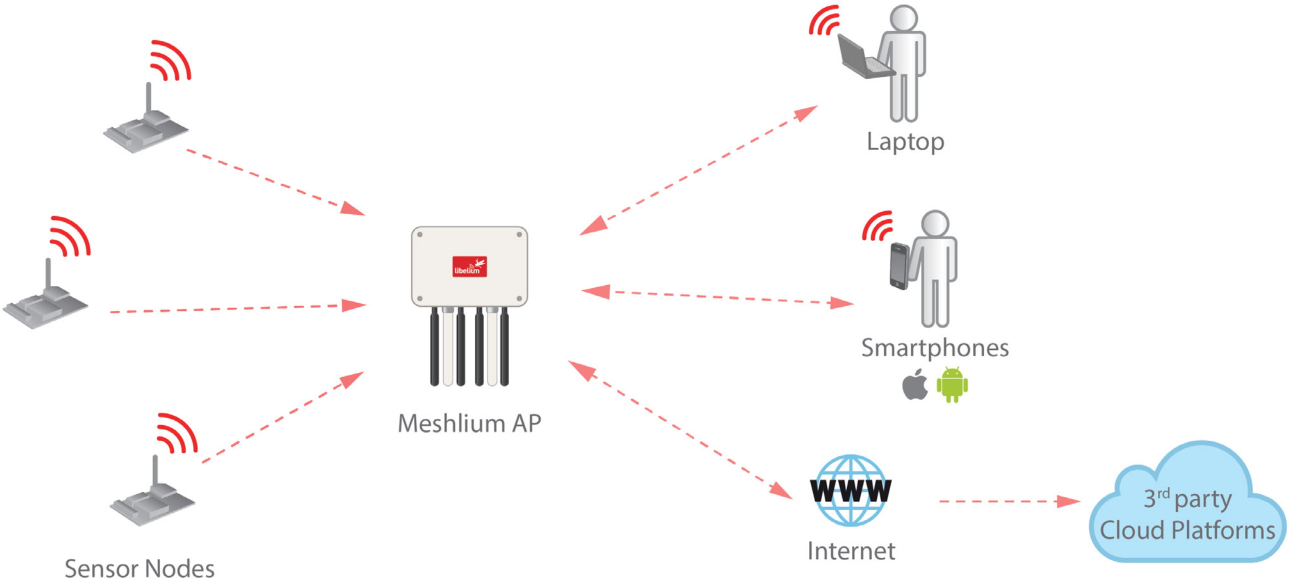


Figure: Meshlium AP

1.3. When is Meshlium recommended?

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

Figure: Meshlium device

Meshlium is also ready to deal with several nodes at the same time, receiving sensor data from all of them and storing it in its internal database automatically. Besides Meshlium is ready to send sensor data to many Cloud software platforms. Just select the most suitable for you, get an account from the provider and configure your Meshlium.

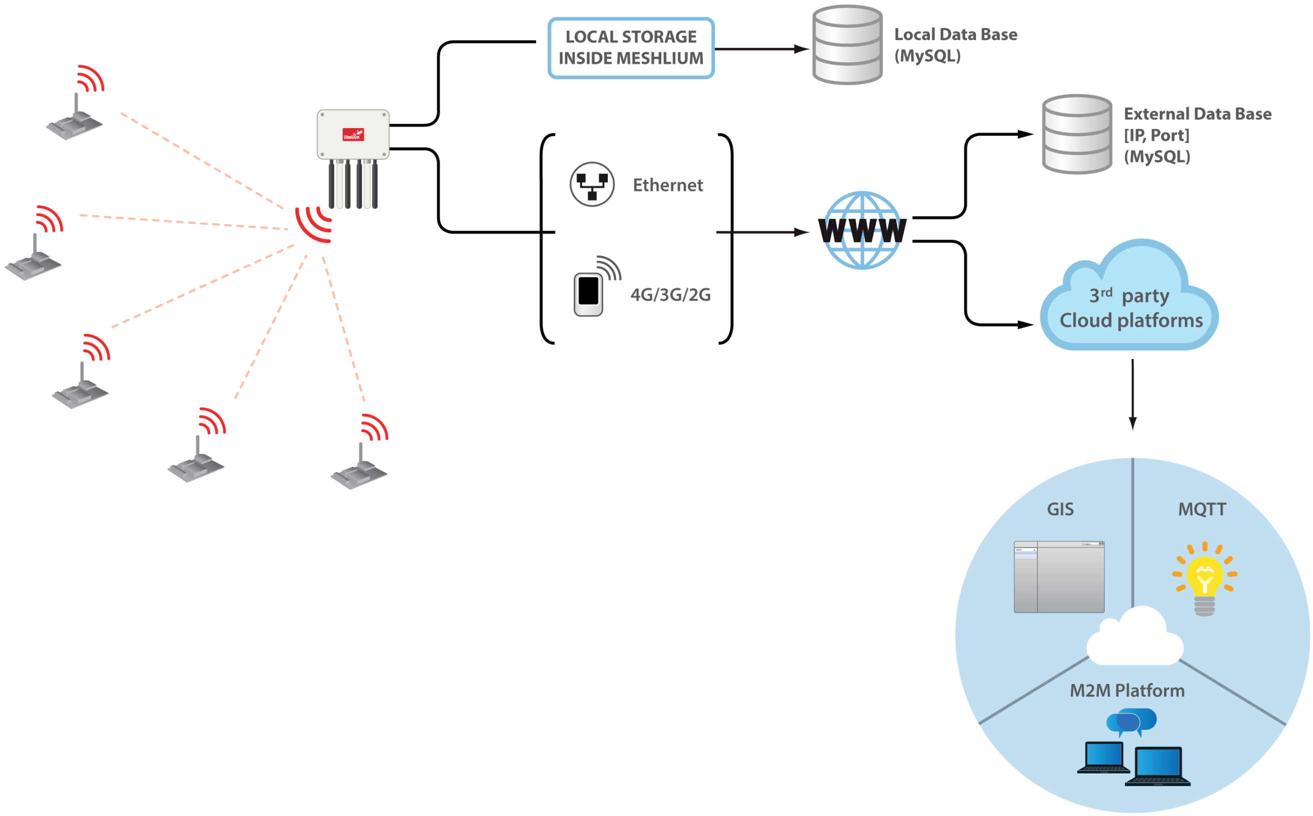

LOCAL STORAGE INSIDE MESHLIUM
Local Data Base (MySQL)
External Data Base [IP, Port] (MySQL)
Ethernet
4G/3G/2G
WWW
3rd party Cloud platforms
GIS
MQTT
M2M Platform

Figure: Cloud connector diagram

As Meshlium is a multiprotocol router, it may work as a WiFi to 4G/3G/GPRS gateway, giving Internet access to all the nodes in the network, using the mobile phones infrastructure. In addition, Meshlium permits to combine WiFi technology with other protocols such as 802.15.4, RF 868 MHz or RF 900 MHz. Meshlium may work as:

- an 802.15.4 to Ethernet/4G/3G/GPRS router for Waspmote nodes
- an RF 868/900 MHz to Ethernet/4G/3G/GPRS router for Waspmote nodes
- a WiFi Access Point
- a WiFi to 4G/3G/GPRS router
- a Bluetooth scanner and analyzer

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

1.4. WiFi vs WiFi PRO

Comparative table between the WiFi and the WiFi PRO module for Waspmote:

	[v12] WiFi	[v15] WiFi PRO
Simultaneous TCP/UDP sockets	1	10
HTTP GET	Yes	Yes
HTTPS POST	No	Yes
HTTPS GET	No	Yes
HTTPS POST	No	Yes
FTP	Yes	Yes
Multiple SSIDs	No	Yes
Roaming mode	No	Yes
Max Tx power	12 dBm	17 dBm
Max Power Consumption	120 mA	350 mA

Figure: Comparative table between WiFi and WiFi PRO

WiFi PRO compatibility:

Item	Compatible	Notes
Waspmote 12	Yes	New Waspmote API needed (v025 or newer)
Waspmote 15	Yes	New Waspmote API needed (v025 or newer)
Old WiFi codes	No	The new WiFi module provides new improved examples and libraries

2. Hardware

2.1. Specifications

The WiFi PRO module is managed by UART and it can be connected to SOCKET0 or SOCKET1. The main features of the modules are listed below:

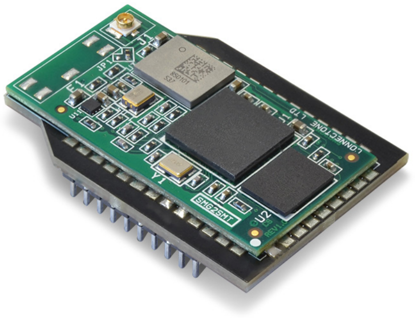


Figure: WiFi PRO module

- **TX power:**
  - 802.11b: 17 dBm
  - 802.11g: 14 dBm
  - 802.11n: 12 dBm
- **RX sensitivity:**
  - 802.11b @11Mbps PER<8%: -87 dBm
  - 802.11b @1Mbps PER<8%: -94 dBm
  - 802.11g @54Mbps PER<10%: -73 dBm
  - 802.11g @6Mbps PER<10%: -86 dBm
  - 802.11n MCS0 PER<10%: -86 dBm
  - 802.11n MCS0 PER<10%: -70 dBm
- **Chipset consumption:**
  - TX mode: 350 mA
  - RX mode: 130 mA
- **Internet protocols:** ARP, ICMP, IP, UDP, TCP, DHCP, DNS, NTP, HTTP, FTP
- **Security protocols:** SSL3/TLS1, HTTPS, RSA, AES-128/256, 3DES, RC-4, SHA-1, MD-5, WEP, WPA and WPA2 Accelerated in HW: AES, 3DEC and SHA
- **Wireless Specifications:**
  - **Standards:** IEEE 802.11b/g/n
  - **Frequency:**
    - Europe: 2.412 – 2.472 GHz
    - USA: 2.412 – 2.462 GHz
    - Japan: 2.412 – 2.484 GHz
  - **Channels:** 1 to 11
- **Antenna:**
  - Plug and Sense!: UFL connector
  - Waspmote OEM: on-chip antenna

Figure: WiFi PRO module for Waspmote OEM

Figure: WiFi PRO module for P&S

2.2. Power consumption

The WiFi PRO module is powered at 3.3 V. The next table shows the module's average current consumption in different states.

State	Power consumption
TX data	350 mA
RX data	130 mA

2.3. Time consumption

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

Action	Time consumption
Power ON	~ 1.8 s
Power ON + join AP	~ 7 s
Perform HTTP GET (already joined AP)	~ 0.5 s
Perform HTTP POST (already joined AP)	~ 0.4 s
Open FTP session (already joined AP)	~ 0.8 s
Perform FTP upload 10KB file (already joined AP)	~ 2.5 s
Perform FTP download 10KB file (already joined AP)	~ 17.6 s

Some of these actions approximately have a fixed elapsed time like powering on the module or performing HTTP/FTP operations. However, the actions related to join AP or open the FTP session are dependent on third parts and have more variability from the mean value. For instance, the joining process can take from few seconds to more than twenty seconds.

2.4. How to connect the module

This module can be connected to both SOCKET0 and SOCKET1 placed in the Waspmote board.

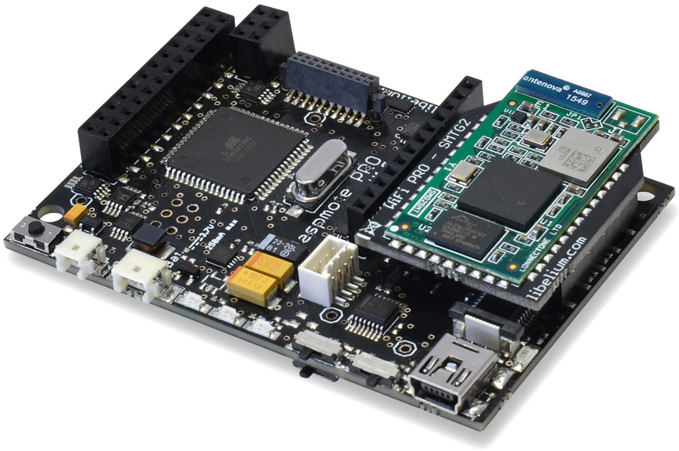


Figure: Module connected to Waspmote in SOCKET0

In order to connect the module to the SOCKET1, the user must use the Expansion Radio Board.

2.5. Expansion Radio Board

The Expansion Board allows to connect two communication modules at the same time in the Waspmote sensor platform. This means a lot of different combinations are possible using any of the wireless radios available for Waspmote: 802.15.4, ZigBee, DigiMesh, 868 MHz, 900 MHz, LoRa, WiFi, GPRS, GPRS+GPS, 3G, 4G, Sigfox, LoRaWAN, Bluetooth Pro, Bluetooth Low Energy and RFID/NFC. Besides, the following Industrial Protocols modules are available: RS-485/Modbus, RS-232 Serial/Modbus and CAN Bus.

Some of the possible combinations are:

- LoRaWAN - GPRS
- 802.15.4 - Sigfox
- 868 MHz - RS-485
- RS-232 - WiFi
- DigiMesh - 4G
- RS-232 - RFID/NFC
- WiFi - 3G
- CAN Bus - Bluetooth
- etc.

Remark: *GPRS, GPRS+GPS, 3G and 4G modules do not need the Expansion Board to be connected to Waspmote. They can be plugged directly in the socket1.*

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.

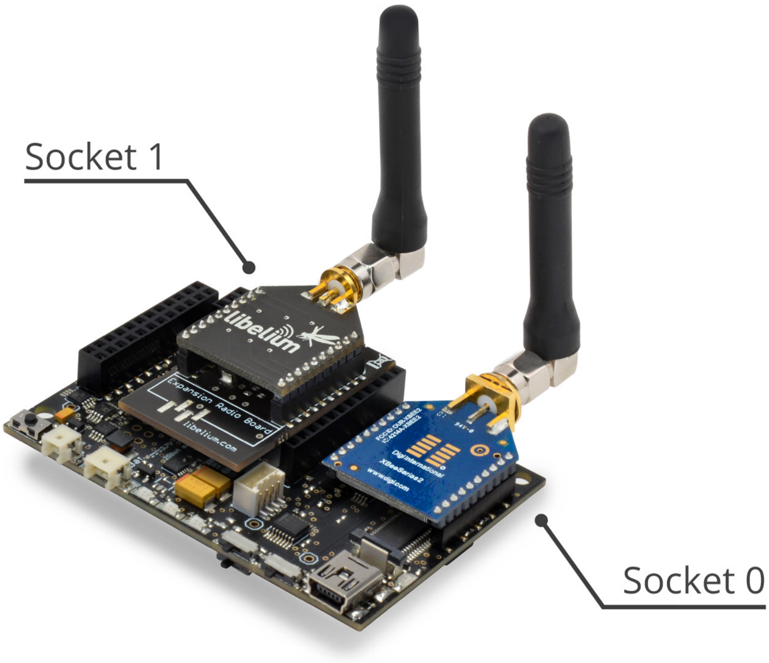


Figure: Use of the Expansion Board

The API provides a function called `ON()` in order to switch the 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_PRO.ON(SOCKET0);`

Selecting SOCKET1: `WIFI_PRO.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.

Warnings:

- Avoid to use DIGITAL7 pin when working with the Expansion Board. This pin is used for setting the XBee into sleep mode.
- Avoid to use DIGITAL6 pin when working with the Expansion Board. This pin is used as power supply for the Expansion Board.
- Incompatibility with Sensor Boards:
    - Agriculture v30 and Agriculture PRO v30: Incompatible with Watermark and solar radiation sensors
    - Events v30: Incompatible with interruption shift register
    - Gases v30: DIGITAL6 is incompatible with CO2 (SOCKET_2) and DIGITAL7 is incompatible with NO2 (SOCKET_3)
    - Smart Water v30: DIGITAL7 incompatible with conductivity sensor
    - Smart Water Ions v30: Incompatible with ADC conversion (sensors cannot be read if the Expansion Board is in use)
    - Gases PRO v30: Incompatible with SOCKET_2 and SOCKET_3
    - Cities PRO v30: Incompatible with SOCKET_3. I2C bus can be used. No gas sensor can be used.

3. Software

3.1. Waspmote libraries

3.1.1. Waspmote WiFi PRO libraries

The files related to the WiFi PRO libraries are:

```
/WIFI_PRO/WaspWIFI_PRO.h
/WIFI_PRO/WaspWIFI_PRO.cpp
/WIFI_PRO/utility/ati_error_codes.h
/WIFI_PRO/utility/ati_generator.h
```

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

```
#include <WaspWIFI_PRO.h>
```

3.1.2. Class constructor

To start using the Waspmote WiFi PRO library, an object from the 'WaspWIFI_PRO' class must be created. This object, called `WIFI_PRO`, is already created by default inside Waspmote WIFI_PRO library. It will be used through this guide to show how Waspmote works.

When using the class constructor, all variables are initialized to a default value.

3.1.3. API constants

The API constants used in functions are:

Constant	Description
DEBUG_WIFI	This definition enables/disables the debug mode via USB port: 0: No debug mode enabled 1: debug mode enabled for error output messages 2: debug mode enabled for both error and OK messages
WIFI_PRO_SCANFILE	This constant defines the file name where the scanned APs are stored
WIFI_PRO_LISTFILE	This constant defines the file name where the FTP listed directories and files are stored
OPEN	Security mode: no security enabled
WEP64	Security mode: WEP 64-bit security enabled
WEP128	Security mode: WEP 128-bit security enabled
WPA	Security mode: WPA security enabled
WPA2	Security mode: WPA2 security enabled
PROFILE_0 PROFILE_1 PROFILE_2 PROFILE_3 PROFILE_4 PROFILE_5 PROFILE_6 PROFILE_7 PROFILE_8 PROFILE_9	Profile definition for multiple SSIDs

3.1.4. API variables

The variables used inside functions and Waspmote codes are:

Variable	Description
_buffer	The buffer of memory used for storing the responses from the module
_length	The useful length of the buffer
_def_delay	The time to wait after sending every command until listen for a response
_baudrate	The baud rate to be used when the module is switched on
_uart	The selected UART (regarding the socket used: SOCKET0 or SOCKET1)
_errorCode	It stores the error code returned by the module when calling a function with error response
_rtt	It stores the last round trip time performed by a ping call
_ip	It stores the module's IP address when the proper function is called
_gw	It stores the gateway's IP address when the proper function is called
_netmask	It stores the netmask's IP address when the proper function is called
_dns1	It stores the DNS #1 server's IP address when the proper function is called
_dns2	It stores the DNS #2 server's IP address when the proper function is called
_socket_handle	It stores the handle number for a new TCP/UDP socket
_ftp_handle	It stores the handle number for a new FTP session
_filesize	It stores the FTP server file size when the proper function is called
_essid	It stores the ESSID of the AP where the module is connected to
_bssid	It stores the BSSID of the AP where the module is connected to
_channel	It stores the channel used by the module in the current connection
_rate	It stores the transmission rate used by the module in the current connection
_level	It stores the signal level of the module in the current connection (%RSSI)
_quality	It stores the link quality of the module in the current connection (%SNR)
_snr	It stores the SNR of the module in the current connection (dBm)
_power	It stores the transmission power level of the module (dBm)

3.1.5. API functions

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

All public functions return one of these possible values:

- 0: OK
- 1: ERROR. See corresponding error code

3.1.6. Error codes

When a function returns error, the _errorCode variable stores the corresponding error meaning. This error value is described by constants as the table below:

Constant	Value	Error code description
ERROR_CODE_0000	0	Waspmote API timeout error
ERROR_CODE_0010	10	SD not present
ERROR_CODE_0011	11	SD file not created
ERROR_CODE_0012	12	SD error opening file
ERROR_CODE_0013	13	SD error setting file offset
ERROR_CODE_0014	14	SD error writing
ERROR_CODE_0020	20	RX buffer full
ERROR_CODE_0021	21	Error downloading UPGRADE.TXT
ERROR_CODE_0022	22	Filename in UPGRADE.TXT is not a 7-byte name
ERROR_CODE_0023	23	No FILE label is found in UPGRADE.TXT
ERROR_CODE_0024	24	NO_FILE is defined as FILE in UPGRADE.TXT
ERROR_CODE_0025	25	No PATH label is found in UPGRADE.TXT
ERROR_CODE_0026	26	No SIZE label is found in UPGRADE.TXT
ERROR_CODE_0027	27	No VERSION label is found in UPGRADE.TXT
ERROR_CODE_0028	28	Version indicated in UPGRADE.TXT is lower/equal to Waspmote's version
ERROR_CODE_0029	29	File size does not match the indicated in UPGRADE.TXT
ERROR_CODE_0030	30	Error downloading binary file
ERROR_CODE_0031	31	Invalid data length
ERROR_CODE_0041	41	Illegal delimiter
ERROR_CODE_0042	42	Illegal value
ERROR_CODE_0043	43	CR expected
ERROR_CODE_0044	44	Number expected
ERROR_CODE_0045	45	CR or ‘,’ expected
ERROR_CODE_0046	46	DNS expected
ERROR_CODE_0047	47	‘:’ or ‘~’ expected
ERROR_CODE_0048	48	String expected
ERROR_CODE_0049	49	‘:’ or ‘=’ expected
ERROR_CODE_0050	50	Text expected
ERROR_CODE_0051	51	Syntax error
ERROR_CODE_0052	52	‘,’ expected
ERROR_CODE_0053	53	Illegal command code
ERROR_CODE_0054	54	Error when setting parameter
ERROR_CODE_0055	55	Error when getting parameter value
ERROR_CODE_0056	56	User abort
ERROR_CODE_0061	61	Internal memory failure
ERROR_CODE_0062	62	User aborted the system
ERROR_CODE_0063	63	CTSH needs to be LOW to change to hardware flow control
ERROR_CODE_0064	64	User aborted last command using ‘---’
ERROR_CODE_0065	65	iChip unique ID already exists
ERROR_CODE_0066	66	Error when setting the MIF parameter

ERROR_CODE_0067	67	Command ignored as irrelevant
ERROR_CODE_0068	68	iChip serial number already exists
ERROR_CODE_0069	69	Timeout on host communication
ERROR_CODE_0070	70	Modem failed to respond
ERROR_CODE_0071	71	No dial tone response
ERROR_CODE_0072	72	No carrier modem response
ERROR_CODE_0073	73	Dial failed
ERROR_CODE_0074	74	WLAN connection lost
ERROR_CODE_0075	75	Access denied to ISP server
ERROR_CODE_0086	86	Writing to internal non-volatile parameters database failed
ERROR_CODE_0087	87	Web server IP registration failed
ERROR_CODE_0088	88	Socket IP registration failed
ERROR_CODE_0094	94	In Always Online mode, connection was lost and re-established
ERROR_CODE_0096	96	A remote host was disconnected
ERROR_CODE_0100	100	Error restoring default parameters
ERROR_CODE_0101	101	No ISP access numbers defined
ERROR_CODE_0102	102	No USRN defined
ERROR_CODE_0103	103	No PWD entered
ERROR_CODE_0104	104	No DNS defined
ERROR_CODE_0111	111	Serial data overflow
ERROR_CODE_0112	112	Illegal command when modem online
ERROR_CODE_0116	116	Error parsing a new trusted CA certificate
ERROR_CODE_0117	117	Error parsing a new Private Key
ERROR_CODE_0118	118	Protocol specified in the USRV parameter does not exist or is unknown
ERROR_CODE_0119	119	WPA passphrase too short has to be 8-63 chars
ERROR_CODE_0125	125	Invalid WEP key
ERROR_CODE_0126	126	Invalid parameters' profile number
ERROR_CODE_0128	128	Product ID already exists
ERROR_CODE_0129	129	HW pin can not be changed after Product-ID was set
ERROR_CODE_0200	200	Socket does not exist
ERROR_CODE_0201	201	Socket empty on receive
ERROR_CODE_0202	202	Socket not in use
ERROR_CODE_0203	203	Socket down
ERROR_CODE_0204	204	No available sockets
ERROR_CODE_0206	206	PPP open failed for socket
ERROR_CODE_0207	207	Error creating socket
ERROR_CODE_0208	208	Socket send error
ERROR_CODE_0209	209	Socket receive error
ERROR_CODE_0210	210	PPP down for socket
ERROR_CODE_0212	212	Socket flush error
ERROR_CODE_0215	215	No carrier error on socket operation
ERROR_CODE_0216	216	General exception
ERROR_CODE_0217	217	Out of memory
ERROR_CODE_0218	218	An STCP (Open Socket) command specified a local port number that is already in use

ERROR_CODE_0220	220	SSL initialization/internal CA certificate loading error
ERROR_CODE_0221	221	Illegal SSL socket handle. Must be an open and active TCP socket
ERROR_CODE_0222	222	Trusted CA certificate does not exist
ERROR_CODE_0224	224	Decoding error on incoming SSL data
ERROR_CODE_0225	225	No additional SSL sockets available
ERROR_CODE_0226	226	Maximum SSL packet size (2KB) exceeded
ERROR_CODE_0227	227	Send command failed because size of stream sent exceeded 2048 bytes
ERROR_CODE_0228	228	Send command failed because checksum calculated does not match checksum sent by host
ERROR_CODE_0229	229	SSL parameters are missing
ERROR_CODE_0230	230	Maximum packet size (4 GB) exceeded
ERROR_CODE_0300	300	HTTP server unknown
ERROR_CODE_0301	301	HTTP server timeout
ERROR_CODE_0303	303	No URL specified
ERROR_CODE_0304	304	Illegal HTTP host name
ERROR_CODE_0305	305	Illegal HTTP port number
ERROR_CODE_0306	306	Illegal URL address
ERROR_CODE_0307	307	URL address too long
ERROR_CODE_0400	400	MAC address exists
ERROR_CODE_0401	401	No IP address
ERROR_CODE_0402	402	Wireless LAN power set failed
ERROR_CODE_0403	403	Wireless LAN radio control failed
ERROR_CODE_0404	404	Wireless LAN reset failed
ERROR_CODE_0405	405	Wireless LAN hardware setup failed
ERROR_CODE_0406	406	Command failed because WiFi module is currently busy
ERROR_CODE_0407	407	Illegal WiFi channel
ERROR_CODE_0408	408	Illegal SNR threshold
ERROR_CODE_0409	409	WPA connection process has not yet completed
ERROR_CODE_0410	410	The network connection is offline (modem)
ERROR_CODE_0411	411	Command is illegal when Bridge mode is active
ERROR_CODE_0501	501	Communications platform already active
ERROR_CODE_0505	505	Cannot open additional FTP session – all FTP handles in use
ERROR_CODE_0506	506	Not an FTP session handle
ERROR_CODE_0507	507	FTP server not found
ERROR_CODE_0508	508	Timeout when connecting to FTP server
ERROR_CODE_0509	509	Failed to login to FTP server (bad username or password or account)
ERROR_CODE_0510	510	FTP command could not be completed
ERROR_CODE_0511	511	FTP data socket could not be opened
ERROR_CODE_0512	512	Failed to send data on FTP data socket
ERROR_CODE_0513	513	FTP shutdown by remote server
ERROR_CODE_0570	570	PING destination not found
ERROR_CODE_0571	571	No reply to PING request

3.2. Switch on

The `ON()` function allows to switch on the WiFi PRO module and it opens the MCU's UART for communicating with the module. After this step the module will be able to receive commands to manage it. It is necessary to indicate the socket that it is being used: `SOCKET0` or `SOCKET1`.

Example of use for SOCKET0:

```
{
   WIFI_PRO.ON(SOCKET0);
}
```

3.3. Restore to factory defaults

The `resetValues()` function allows to restore the module's non-volatile parameter database values to factory defaults. Each one of the module's non-volatile parameters has an associated default value. This function restores all parameters to their factory default values.

Example of use:

```
{
   WIFI_PRO.resetValues();
}
```

3.4. Switch off

The `OFF()` function allows the user to switch off the WiFi PRO module and close the UART. This function must be called in order to keep battery level when the module is not going to be used. It is necessary to indicate the socket that it is being used: `SOCKET0` or `SOCKET1`.

Example of use for SOCKET0:

```
{
   WIFI_PRO.OFF(SOCKET0);
}
```

3.5. How to configure and join an Access Point

In order to configure the module to join an Access Point (AP), it is mandatory to define the ESSID of the AP, the password of the security enabled in that link and finally perform a software reset so as to apply the changes.

Once these parameters have been set, they are permanently stored in the non-volatile memory of the module. So, it is not necessary to re-configure these parameters anymore, unless the user needs to change the AP settings.

3.5.1. Configure ESSID

The `setESSID()` function allows the user to configure the ESSID to join.

The `getESSID()` function allows the user to request the current ESSID setting. The _essid attribute permits to read the settings of the module.

Example of use:

```
{
   WIFI_PRO.setESSID("libelium_AP");
   WIFI_PRO.getESSID();
}
```

Related variable:

`WIFI_PRO._essid` → Stores the current ESSID of the module

3.5.2. Configure the password

The `setPassword()` function allows the user to configure the password to the module. It takes several seconds to generate the keys. This function needs two inputs:

- **Authentication mode:**
  - `OPEN`: No security
  - `WEP64`: WEP 64-bit
  - `WEP128`: WEP 128-bit
  - `WPA`: WPA-PSK
  - `WPA2`: WPA2-PSK

- **Password:**
  - If Security Mode = WPA/WPA2: This is the pass-phrase to be used in generating the PSK encryption key. The allowed value for pass is an ASCII string containing 8 to 63 characters.
  - If Security Mode = WEP64: This key must be defined by 10 hexadecimal digits. Each byte of the 5-byte key is defined by two ASCII characters in the ranges ['0' to '9'], ['A' to 'F'] or ['a' to 'f'].
  - If Security Mode = WEP128: This key must be defined by 26 hexadecimal digits. Each byte of the 13-byte key is defined by two ASCII characters in the ranges ['0' to '9'], ['A' to 'F'] or ['a' to 'f'].

Example of use:

```
{
   WIFI_PRO.setPassword(WPA2, “password”);
}
```

3.5.3. Software reset

Once the module has been set to the correct settings they are kept in the non volatile memory of the module. Besides, it is mandatory to restart the module in order to force the module to use the new settings. For that purpose, the `softReset()` function is used to perform a software reset to the module. After calling this function, the new setting takes effect.

3.5.4. Join the Access Point

Once the module has valid settings in the non volatile memory, it automatically starts searching to join the Access Point. The `isConnected()` function permits to know if the WiFi PRO module is already connected to the Access Point. This function returns true or false values in order to provide the status information.

Examples of configuring the module and joining the AP:

www.libelium.com/development/waspmote/examples/wifi-pro-01-configure-essid

www.libelium.com/development/waspmote/examples/wifi-pro-02-join

3.6. IP addressing

When joining an AP it is possible to use the DHCP client of the module or configure a static IP address.

3.6.1. DHCP client

By default, the WiFi PRO module uses the DHCP client, so when it joins the AP, an IP address is assigned to the module. The `getIP()` function permits to request the current IP address of the module.

Example of use:

```
{
   WIFI_PRO.getIP();
}
```

Related variable:

`WIFI_PRO._ip` → Stores the current IP address assigned to the module

Example of getting the module's IP address:

www.libelium.com/development/waspmote/examples/wifi-pro-03-get-ip

3.6.2. Static IP address

It is possible to set up a default IP address for the WiFi PRO module. Besides, it is possible to set other network parameters: DNS address, Gateway address and Netmask. The functions for all these settings are shown below:

The `setIP()` function allows the user to set the IP address to the module in the network.
The `setDNS()` function allows the user to set the DNS address to the WiFi PRO module.
The `setGateway()` function allows the user to set the Gateway address to the WiFi PRO module.
The `setNetmask()` function allows the user to set the netmask address to the WiFi PRO module.

Remember that a software reset is needed in order to apply all these changes in the WiFi PRO module.

Example of use:

```
{
   WIFI_PRO.setIP(“192.168.5.248”);
   WIFI_PRO.setDNS(“8.8.8.8”);
   WIFI_PRO.setGateway(“192.168.1.2”);
   WIFI_PRO.setNetmask(“255.255.128.0”);
}
```

Example of using static IP address:

www.libelium.com/development/waspmote/examples/wifi-pro-04-static-ip

3.7. Ping

The ping() function sends a two-byte ICMP PING request packet to the remote host defined as input argument. The input of the function can be a logical name of the target host or a host IP address. Upon successfully receiving an ICMP PING reply from the host, the round trip time in milliseconds is returned (RTT) and stored in the _rtt attribute.

Example of use:

```
{
   WIFI_PRO.ping("www.google.com");
}
```

Related variable:

WIFI_PRO._rtt → Stores the last round trip time performed by a ping call

Example of performing a ping from the module:

www.libelium.com/development/waspmote/examples/wifi-pro-05-ping

3.8. Power level

The setPower() function allows the user to configure the transmission power of the chipset. The getPower() function allows the user to request the transmission power of the chipset which is stored in the _power attribute. After a hardware or software reset, the power level parameter returns to its default value. This parameter is in the range 1 to 14 dBm. The default value is 14 dBm.

Example of use:

```
{
   WIFI_PRO.setPower(14);
   WIFI_PRO.getPower();
}
```

Related variable:

WIFI_PRO._power → Stores the power level setting

Example of setting the transmission power level:

www.libelium.com/development/waspmote/examples/wifi-pro-06-set-power

3.9. Certificate management for SSL connections

3.9.1. How SSL works

Secure Sockets Layer (SSL) technology provides data encryption, server authentication and message integrity for a TCP/IP connection. The server authenticates the client using the client's Public Key Certificate (PKC). So, it will be necessary to install the corresponding certificate, created by a CA (Certification Authority), to the module. These CA certificates are usually provided by the browsers.

For more information, refer to the tutorial related to SSL connections:

www.libelium.com/development/waspmote/documentation/how-ssl-works-tutorial

3.9.2. Set the CA certificate

The `setCA()` function sets the certificate of the trusted certificate authority. The WiFi PRO module accepts a server's identity only if its certificate is signed by one of these certificate authorities.

The certificate is a PEM format X509 certificate (DER format, Base-64 encoded with header and footer lines). The certificate is referenced as the trusted certificate authority's certificate during SSL socket connection establishment (handshake). The WiFi PRO module establishes an SSL socket connection only to servers having a certificate authenticated by this certificate authority. The certificate must be defined by multiple lines separated by <CR>, beginning with:`-----BEGIN CERTIFICATE-----` and terminating with: `-----END CERTIFICATE-----`. The certificate should include an RSA encryption public key of 1024 or 2048 bits. The signature algorithm may be MD2, MD5 or SHA1. The maximum size of the certificate is 1500 characters.

Example of valid certificate setting:

```
{
    char TRUSTED_CA[] =\
    “-----BEGIN CERTIFICATE-----\r”\
    “MIICPDCCAaUCEHC65B0Q2Sk0tjjKewPMur8wDQYJKoZIhvcNAQECBQAwXzELMAkG\r”\
    “A1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMTcwNQYDVQQLEy5DbGFz\r”\
    “cyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9yaXR5MB4XDTk2\r”\
    “MDEyOTAwMDAwMFoXDTI4MDgwMTIzNTk1OVowXzELMAkGA1UEBhMCVVMxFzAVBgNV\r”\
    “BAoTDlZlcmlTaWduLCBJbmMuMTcwNQYDVQQLEy5DbGFzcyAzIFB1YmxpYyBQcmlt\r”\
    “YXJ5IENlcnRpZmljYXRpb24gQXV0aG9yaXR5MIGfMA0GCSqGSIb3DQEBAQUAA4GN\r”\
    “ADCBiQKBgQDJXFme8huKARS0EN8EQNvjV69qRUCPhAwL0TPZ2RHP7gJYHyX3KqhE\r”\
    “BarsAx94f56TuZoAqiN91qyFomNFx3InzPRMxnVx0jnvT0Lwdd8KkMaOIG+YD/is\r”\
    “I19wKTakyYbnsZogy1Olhec9vn2a/iRFM9x2Fe0PonFkTGUugWhFpwIDAQABMA0G\r”\
    “CSqGSIb3DQEBAgUAA4GBALtMEivPLCYATxQT3ab7/AoRhIzzKBxnki98tsX63/Do\r”\
    “lbwdj2wsqFHMc9ikwFPwTtYmwHYBV4GSXiHx0bH/59AhWM1pF+NEHJwZRDmJXNyc\r”\
    “AA9WjQKZ7aKQRUzkuxCkPfAyAw7xzvjoyVGM5mKf5p/AfbdynMk2OmufTqj/ZA1k\r”\
    “-----END CERTIFICATE-----”;

    WIFI_PRO.setCA(TRUSTED_CA);
}
```

Note: Firmware versions ID811d15 and greater use SSL3/TLS1.2 protocol only.

3.10. TCP/UDP sockets

3.10.1. TCP client

The `setTCPclient()` function opens a Transmission Control Protocol (TCP) client socket and attempts to connect to the specified port on a server defined as input. Therefore, this function needs three different inputs:

- **Host:** The server name may be any legal Internet server name that can be resolved by module's DNS (Domain Name Server) settings. The server name can also be specified as an absolute IP address given in dot-decimal notation.
- **Remote port:** It is assumed that the server system is listening on the specified port.
- **Local port:** This is the local port when opening the TCP socket.

Upon successfully opening and connecting the TCP socket to the <Host>:<Remote port>, a socket handle is returned. The socket handle is stored in the `_socket_handle` attribute. This handle is in the range 0 to 9. This handle is needed to reference the socket in all following socket commands.

Example of use:

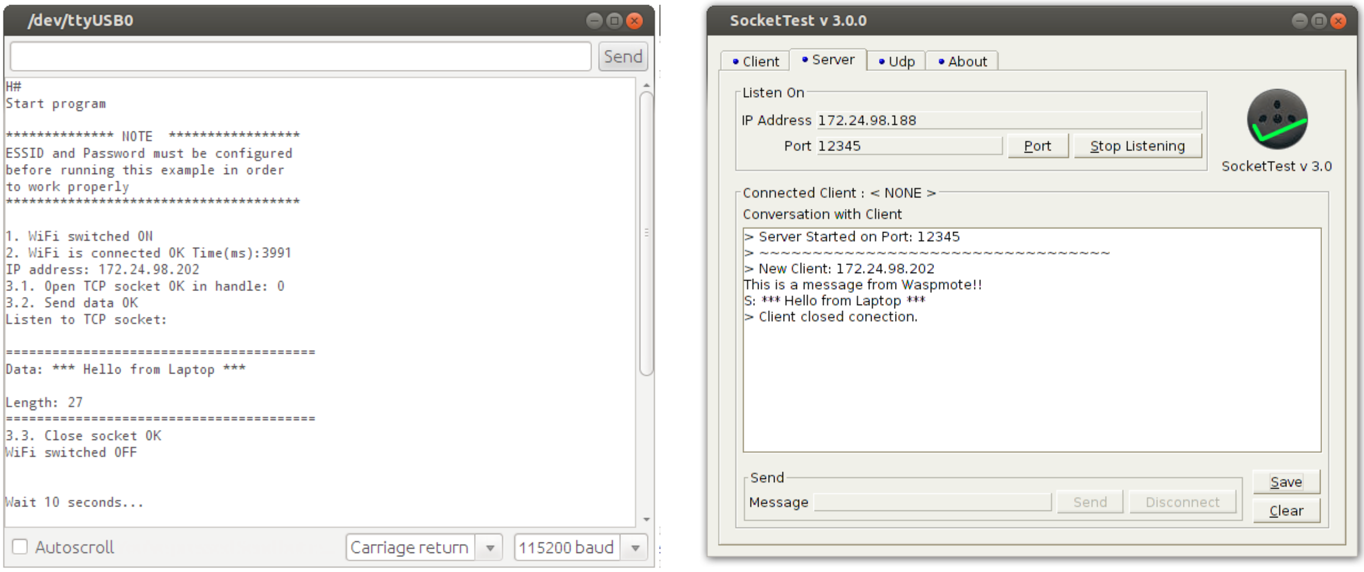
```
{
   char HOST[]        = “192.168.5.152”;
   char REMOTE_PORT[] = “2000”;
   char LOCAL_PORT[]  = “3000”;

   WIFI_PRO.setTCPclient( HOST, REMOTE_PORT, LOCAL_PORT);
}
```

Related variable:

`WIFI_PRO._socket_handle` → Stores the TCP socket handle

Example of use for TCP sockets:

www.libelium.com/development/waspmote/examples/wifi-pro-07-tcp-client



```
H#
Start program

************** NOTE  *****************
ESSID and Password must be configured
before running this example in order
to work properly
**************************************

1. WiFi switched ON
2. WiFi is connected OK Time(ms):3991
IP address: 172.24.98.202
3.1. Open TCP socket OK in handle: 0
3.2. Send data OK
Listen to TCP socket:

========================================
Data: *** Hello from Laptop ***

Length: 27
========================================
3.3. Close socket OK
WiFi switched OFF


Wait 10 seconds...
```

IP Address 172.24.98.188
Port 12345
Connected Client : < NONE >
Conversation with Client

```
> Server Started on Port: 12345
> ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> New Client: 172.24.98.202
This is a message from Waspmote!!
S: *** Hello from Laptop ***
> Client closed conection.
```

Figure: TCP client in Waspmote vs TCP server in laptop

3.10.2. TCP server

The `setTCPserver()` function opens a TCP listening socket on the local IP address and the specified port. The second input parameter specifies the maximum number of remote concurrent connections allowed through the listening socket. Thus, this function needs 2 different inputs:

- **Local port:** The listening port to be used by a remote system when connecting to the WiFi PRO module.
- **Max number of clients:** Specifies the maximum number of active connections that may be concurrently established through the listening socket.

Once the listening socket is open, it automatically accepts remote connect requests up to the maximum allowed. When a remote system connects through the listening socket, a new TCP socket is spawned internally ready to send and receive data.

The `getAllSocketStatus()` function allows the user to update the information of all active sockets connected through a listening socket. The information for each one of the active sockets is stored in a structure called `listen_socket_t`. There are ten structures defined in the Waspmote libraries for permitting up to ten connections simultaneously.

The definition of the array of sockets structures is:

```
listen_socket_t   socket[10];
```

The definition of the structure is:

```
struct  listen_socket_t
{
   uint16_t   handle;       // socket handle
   int8_t     status;       // 0: active;  -1: non-active
   char       ip[16];       // xxx.xxx.xxx.xxx
   uint16_t   port;         // remote connection port
   int        size;         // size of pending bytes
};
```

Example of use for TCP server:

www.libelium.com/development/waspmote/examples/wifi-pro-08-tcp-server

3.10.3. UDP

The `setUDP()` function opens a UDP (User Datagram Protocol) socket and sets the remote system's <Host>:<port> address. Therefore, this function needs three different inputs:

- **Host:** Logical name of the target server or a host IP address. The remote system's name may be any legal Internet server name that can be resolved by module's DNS (Domain Name Server) settings. The server name may also be specified as an absolute IP address given in dot-decimal notation. When the <Host> is defined, the resulting UDP socket is created and connected.

  If <Host>=0.0.0.0, the socket is created but remains unconnected. The first UDP packet to arrive automatically latches the sender's IP port, in effect connecting the socket.

  <Host> may be a subnet directed Broadcast address which allows to broadcast packets to the immediate subnet, not crossing gateways. For example, to broadcast to subnet 192.168.x.x on destination port 1234: Host="192.168.255.255" and Remote port="1234".

  <Host> may be a multicast IP address in the range 224.0.0.0 to 239.255.255.255. IP multicast datagrams will not cross gateways. In this case, data is sent and received on <Remote port>. <Local port> is ignored.
- **Remote port:** Specifies the remote system's port.
- **Local port:** Specifies the local port to use.

Upon successfully opening and connecting the UDP socket to <Host>:<Remote port>, a socket handle is returned. The socket handle is stored in the `_socket_handle` attribute. The socket handle is in the range 0 to 9 and is used to reference the socket in all following socket commands.

> ***Note:*** *The WiFi PRO will only be able to receive UDP packets from the specified <Host> IP address to the specified <Local port>. In the case other connections are needed it is possible to establish new UDP sockets with different hosts.*

Example of use:

```
{
   char HOST[]        = "192.168.5.152";
   char REMOTE_PORT[] = "2000";
   char LOCAL_PORT[]  = "3000";

   WIFI_PRO.setUDP( HOST, REMOTE_PORT, LOCAL_PORT);
}
```

Related variable:

`WIFI_PRO._socket_handle` → Stores the UDP socket handle

Examples of use for UDP sockets:

www.libelium.com/development/waspmote/examples/wifi-pro-09-udp-client

www.libelium.com/development/waspmote/examples/wifi-pro-10-udp-listener

3.10.4. Send data to a TCP/UDP socket

The `send()` function sends a byte stream to the socket specified by the socket handle input. This function needs two different inputs:

- **Socket handle:** A TCP/UDP socket handle of a previously open socket.
- **Data:** This is the stream of data to send to the TCP/UDP socket. This stream of data can be defined as a simple string message. Or an array of bytes, specifying a third input for the length of the array of bytes to send.

Example of use for the string message:

```
{
   WIFI_PRO.send( WIFI_PRO._socket_handle, “this is a message”);
}
```

Example of use for the array of bytes (it is mandatory to specify the length):

```
{
   uint8_t  data[] = {0x31, 0x32, 0x33, 0x34, 0x35}
   WIFI_PRO.send( WIFI_PRO._socket_handle, data, 6);
}
```

3.10.5. Receive data from a TCP/UDP socket

The receive() function receives a byte stream from the TCP/UDP socket specified by the socket handle. Received data is valid only if it already resides in the module’s socket input buffer at the time this command is issued. There are different receiving function prototypes depending on the time the user needs to listen for a new incoming packet. Therefore, this function could need more than one input:

- **Socket handle:** A TCP/UDP socket handle of a previously opened socket. This input is always mandatory.
- **Timeout (optional input):**
  - If no timeout input is specified, the receive function is a non-blocking function which answers if data has been received.
  - If the timeout is inserted as new input, the function will block until a new packet is received or time is up in the case no packet is received. This timeout must be specified in milliseconds units.

Example for instant reception:

```
{
   WIFI_PRO.receive(WIFI_PRO._socket_handle);
}
```

Example for time elapsed reception (i.e. 30 seconds):

```
{
   WIFI_PRO.receive( WIFI_PRO._socket_handle, 30000);
}
```

3.10.6. Closing a socket

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

- **Socket handle**: the socket identifier used for opening the connection.

3.10.7. SSL sockets

The WiFi PRO module includes a software stack for establishing SSL sockets. For using this feature, it is mandatory to insert a certificate of a trusted certificate authority (CA). The user must implement their own secure server and define the certificate to be used with the WiFi PRO module.

The setCA() function sets the certificate of the trusted certificate authority. The WiFi PRO module accepts a server's identity only if its certificate is signed by one of these certificate authorities.

The sslHandshake() function negotiates an SSL connection on a given socket handle. This function requires an input to select the socket handle:

- Socket handle: A TCP/UDP socket handle of a previously opened socket. This input is always mandatory.

Example for SSL connection:

```
{
	WIFI_PRO.sslHandshake(WIFI_PRO._socket_handle);
}
```

Related variable:

WIFI_PRO._socket_handle → Stores the TCP socket handle

Examples of use for SSL sockets:

www.libelium.com/development/waspmote/examples/wifi-pro-26-ssl-sockets

3.11. HTTP client

3.11.1. HTTP GET

The getURL() function retrieves a file from a URL. This function needs two different inputs:

- **Type:** Protocol type must be "http" for simple HTTP or "https" for HTTPS
- **Host:** This is the Host name or IP address
- **Port:** From 0 to 65535. HTTP default port is 80. HTTPS default port is 443.
- **Link:** Absolute link to retrieve on the designated host

Upon the successful retrieving, the answer from the host is stored in the _buffer attribute. Besides, the _length attribute defines the length of the answer stored.

Example of use for HTTP GET to this link:

```
{
   ///////////////////////////////////////////////////////////////////
   char type[] = "http";
   char host[] = "pruebas.libelium.com";
   char port[] = "80";
   char link[] = "getpost_frame_parser.php?counter=1&varA=1&varB=2";
   ///////////////////////////////////////////////////////////////////

   WIFI_PRO.getURL( type, host, port, link);
}
```

Related variable:

WIFI_PRO._buffer → Pointer to the buffer where the answer from host is stored

WIFI_PRO._length → Length of the response stored in _buffer

Example of HTTP GET request:

www.libelium.com/development/waspmote/examples/wifi-pro-12-http-get

3.11.2. HTTP POST

The post() function submits a plain text POST request to a web server defined by the setURL() function. The "Content-type:" field of the POST request is defined by the setContentType() function. A default value of "application/x-www-form-urlencoded" will be used.

This function needs two different inputs:

- **Type:** Protocol type must be "http" for simple HTTP or "https" for HTTPS
- **Host:** This is the host name or IP address
- **Port:** From 0 to 65535. HTTP default port is 80. HTTPS default port is 443.
- **Link:** Absolute link to retrieve on the designated host

Upon the successful posting, the answer from the host is stored in the _buffer attribute. Besides, the _length attribute defines the length of the answer stored.

Example of use for HTTP POST:

```
{
   ///////////////////////////////////////////
   char type[] = “http”;
   char host[] = “pruebas.libelium.com”;
   char port[] = “80”;
   char url[]  = “getpost_frame_parser.php?”;
   ///////////////////////////////////////////

   WIFI_PRO.setURL( type, host, port, link);
   WIFI_PRO.post(“varA=1&varB=2&varC=3&varD=4&varE5=5”);
}
```

Related variable:

`WIFI_PRO._buffer` → Pointer to the buffer where the answer from host is stored

`WIFI_PRO._length` → Length of the response stored in `_buffer`

Example of HTTP POST request:

www.libelium.com/development/waspmote/examples/wifi-pro-13-http-post

3.11.3. HTTPS

It is possible to use HTTPS calls. For that purpose it is mandatory to insert a certificate of a trusted certificate authority. The user must implement their own secure server and define the certificate to be used with the WiFi PRO module.

The `setCA()` function sets the certificate of the trusted certificate authorities. The WiFi PRO module accepts a server’s identity only if its certificate is signed by one of these authorities.

After successfully setting the certificate, the module will be able to perform secure socket connections. Therefore, it will be possible to use the well-known `getURL()` and `post()` functions to perform HTTPS operations.

Example of HTTPS requests:

www.libelium.com/development/waspmote/examples/wifi-pro-14-https-get

www.libelium.com/development/waspmote/examples/wifi-pro-15-https-post

3.11.4. Send Waspmote frames to Meshlium

It is possible to send sensor data from Waspmote to Meshlium using the Waspmote Frame library and HTTP requests. In order to send this kind of data to Meshlium, you can use a Meshlium device as Access Point or use the Internet to access to a remote Meshlium address through a different AP (via a router, for example).

All data sent using the Waspmote Frame to Meshlium is stored in the Meshlium's database using the Frame Parser. Therefore, it is possible to access to this database or synchronize it to an external Cloud Partner.

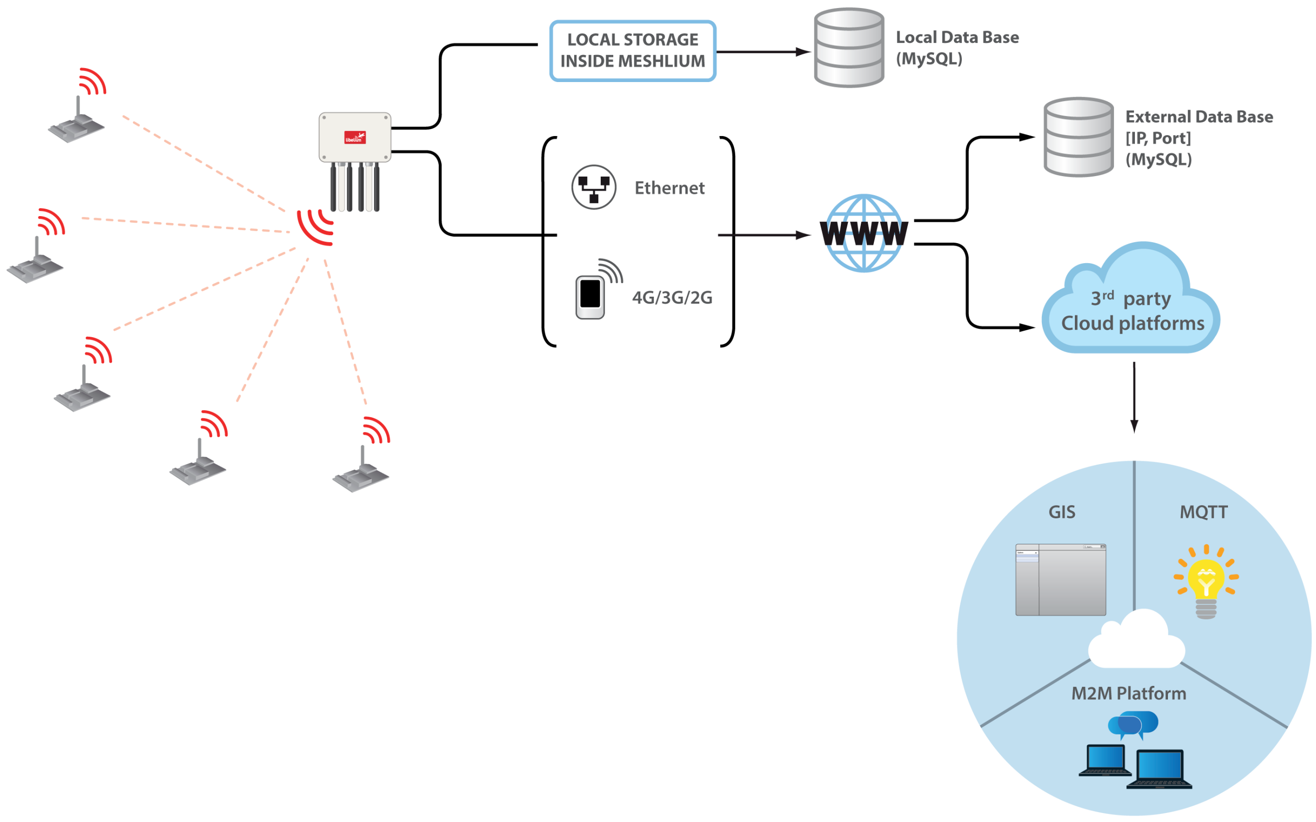

LOCAL STORAGE INSIDE MESHLIUM
Local Data Base (MySQL)
External Data Base [IP, Port] (MySQL)
Ethernet
4G/3G/2G
WWW
3rd party Cloud platforms
GIS
MQTT
M2M Platform

Figure: Send Waspmote frames to Meshlium

The `sendFrameToMeshlium()` function sends the HTTP GET request to the specified host and port. This function needs five inputs:

- **Type:** Protocol type must be "http" for simple HTTP or "https" for HTTPS
- **Host:** This is the Host name or IP address
- **Port:** From 0 to 65535. HTTP default port is 80. HTTPS default port is 443.
- **frame.buffer:** This is the pointer to the Frame structure buffer which contains the sensor data
- **frame.length:** This is the length of the Frame structure buffer

Example of sending a frame to Meshlium:

```
{
   /////////////////////////////
   char type[] = “http”;
   char host[] = “10.10.10.1”;
   char port[] = “80”;
   /////////////////////////////

   WIFI_PRO.sendFrameToMeshlium( type, host, port, frame.buffer, frame.length);
}
```

Example of sending frames to Meshlium:

www.libelium.com/development/waspmote/examples/wifi-pro-16-send-to-meshlium

3.12. FTP client

In order to use the FTP client stack of the WiFi PRO module, different functions must be called. Firstly, the FTP session must be opened. Then it will be possible to upload/download files, list directories, move through directories, etc. Let’s look to all different steps:

3.12.1. Open FTP session

The `ftpOpenSession()` function opens an FTP link to an FTP server. This function needs different inputs:

- **Server:** The server name may be any legal Internet-server name, which can be resolved by the module’s DNS (Domain Name Server) settings. The server name may also be specified as an absolute IP address given in dot-decimal notation.
- **Port:** Optional FTP port in the range 0 to 65535. Default port: 21.
- **User:** FTP user’s name. This must be a registered user on the FTP server.
- **Pass:** FTP user’s password to authenticate the user.

Upon successfully connecting to the FTP Server and authenticating the user, a socket handle is returned. The FTP handle is stored in the `_ftp_handle` attribute. This handle is used to reference the FTP session in all following FTP commands.

Example of use:

```
{
   ///////////////////////////////////////////
   char SERVER[]   = “pruebas.libelium.com”;
   char PORT[]     = “21”;
   char USER[]     = “w@libelium.com”;
   char PASSWORD[] = “ftp1234”;
   ///////////////////////////////////////////

   WIFI_PRO.ftpOpenSession( SERVER, PORT, USER, PASSWORD);
}
```

Related variable:

`WIFI_PRO._ftp_handle` → Stores the FTP handle

3.12.2. FTP directory listing

The ftpListing() function retrieves a full FTP directory listing. The Waspmote SD card is needed for storing all the incoming data on the listing process. There are two function prototypes for this function depending on the inputs:

- **FTP handle:** The FTP handle must be always specified for the FTP session to be considered.
- **Path:** The second input is optional and refers to the directory name or filename wildcard. If <path> is a directory, that directory's files are listed. If it is a filename wildcard, only matching filenames in the current directory are listed. If <path> is not specified, the current directory is listed in full.

Upon successfully retrieving the directory list, the information is stored in an SD file called "LISTFILE.TXT" and is referenced by the WIFI_PRO_LISTFILE label in the libraries. So it is possible to access to this file, display it or extract data from it. Regarding the contents of this file: It contains a list of filenames with file attributes. Each file is listed on a separate line, terminated by <CR/LF>. The file data line syntax is FTP server-dependent.

Example of use for current working directory:

```
{
   WIFI_PRO.ftpListing( WIFI_PRO._ftp_handle);
}
```

Example of use for specific directory path:

```
{
   WIFI_PRO.ftpListing( WIFI_PRO._ftp_handle, “DIRECTORY”);
}
```

Related variable:

`WIFI_PRO_LISTFILE` → The filename where listing info is stored

3.12.3. FTP make directory

The `ftpMakeDir()` function allows the user to create a new directory on the FTP server's file system. This function needs two inputs:

- **FTP handle:** Must have been obtained by a previous execution of a `ftpOpenSession()` function during the current Internet mode session.
- **Path:** Directory name. A new directory will be created under the current directory, as indicated by path. If path includes nonexistent subdirectories, some FTP servers will create them as well.

Example of use:

```
{
   WIFI_PRO.ftpMakeDir( WIFI_PRO._ftp_handle, “DIRECTORY”);
}
```

Example of sending frames to Meshlium:
www.libelium.com/development/waspmote/examples/wifi-pro-19-ftp-make-directory

3.12.4. FTP change working directory

The ftpChangeCWD() function allows the user to change the FTP current working directory. This function needs two inputs:

- **FTP handle:** Must have been obtained by a previous execution of a ftpOpenSession() function during the current Internet mode session.
- **Path:** Absolute or relative path name of the new directory. The special directory ".." means "one directory up".

Example of use:

```
{
   WIFI_PRO.ftpChangeCWD( WIFI_PRO._ftp_handle, "DIRECTORY");
}
```

3.12.5. FTP file size in server

The ftpFileSize() function allows the user to get the size of an FTP server's file. This function needs two inputs:

- **FTP handle:** Must have been obtained by a previous execution of a ftpOpenSession() function during the current Internet mode session.
- **Path:** Absolute or relative path name of the remote file.

Example of use:

```
{
   WIFI_PRO.ftpFileSize( WIFI_PRO._ftp_handle, "FILE.TXT");
}
```

Related variable:

WIFI_PRO._filesize → The size in bytes of the file in the FTP server

3.12.6. FTP upload

The `ftpUpload()` function allows the user to upload a file from the Waspmote's SD card to the FTP server. This function performs different steps: it opens a file in server for storage, uploads a stream of data from the SD card file and finally closes the file in server. This function needs different inputs:

- **FTP handle:** Must have been obtained by a previous execution of a `ftpOpenSession()` function during the current Internet mode session.
- **Server path:** Absolute or relative path name of the remote destination file.
- **SD path:** Absolute or relative path name of the file in Waspmote's SD card.

Example of use:

```
{
   ///////////////////////////////////////////
   uint16_t  handle   =  WIFI_PRO._ftp_handle;
   char SERVER_FILE[] = "/FILE1.TXT";
   char SD_FILE[]     = "/FILE2.TXT";
   ///////////////////////////////////////////

   WIFI_PRO.ftpUpload( handle, SERVER_FILE, SD_FILE);
}
```

Example of FTP upload:

www.libelium.com/development/waspmote/examples/wifi-pro-17-ftp-upload

3.12.7. FTP download

The `ftpDownload()` function allows the user to download a file from the FTP server to Waspmote's SD card. This function needs different inputs:

- **FTP handle:** Must have been obtained by a previous execution of a `ftpOpenSession()` function during the current Internet mode session.
- **Server path:** Absolute or relative path name of the remote destination file.
- **SD path:** Absolute or relative path name of the file in Waspmote's SD card.

Example of use:

```
{
   ///////////////////////////////////////////
   uint16_t  handle   =  WIFI_PRO._ftp_handle;
   char SERVER_FILE[] = "/FILE1.TXT";
   char SD_FILE[]     = "/FILE2.TXT";
   ///////////////////////////////////////////

   WIFI_PRO.ftpDownload( handle, SERVER_FILE, SD_FILE);
}
```

Example of FTP download:

www.libelium.com/development/waspmote/examples/wifi-pro-18-ftp-download

3.12.8. Close FTP session

The `ftpCloseSession()` function closes the FTP link. This function needs the following input:

- **FTP handle:** Must have been obtained by a previous execution of a `ftpOpenSession()` function during the current Internet mode session.

Example of use:

```
{
   WIFI_PRO.ftpCloseSession( WIFI_PRO._ftp_handle);
}
```

3.13. Scan APs

The `scan()` function retrieves a list of all APs available in the surrounding area. The Waspmote SD card is needed for storing all the incoming data on the scanning process.

Upon successfully retrieving the scan list, the information is stored in an SD file called "SCANFILE.TXT" and it is referenced by the `WIFI_PRO_SCANFILE` label in the libraries. So, it is possible to access to this file, display it or extract data from it. Regarding the contents of this file: It contains a list of up to 16 APs available in the surrounding area. Each line contains the following comma-separated fields:

```
<SSID>,ADHOC|AP,<BSSID>,<security-type>,<channel>,<RSSI>
```

Where:

- `<security-type>`=NONE|WEP64|WEP128|WPA|WPA2
- `<RSSI>` = Value between 0-255 which represents (SNR+NoiseFloor). Higher RSSI values indicate weaker signal strength.

For example:

```
libelium_AP,AP,A8:54:B2:9F:46:6E,WPA2,3,55
libelium_teaming,AP,2E:A4:3C:99:2F:B2,WPA2,1,70
libelium_formacion,AP,32:A4:3C:99:2F:C3,WPA2,6,64
Smart_Libelium_Indoor,AP,60:02:B4:68:74:08,WPA2,9,50
libelium_teaming,AP,0E:18:D6:63:E5:2C,WPA2,11,61
I/OK
```

Example of use:

```
{
   WIFI_PRO.scan();
}
```

Related variable:

`WIFI_PRO_SCANFILE` → The filename where scanning info is stored

Example of sending frames to Meshlium:
www.libelium.com/development/waspmote/examples/wifi-pro-20-scan

3.14. Set RTC time from NTP server

It is possible to use Network Time Protocol (NTP) servers to synchronize the Waspmote's RTC settings to the servers settings. For that purpose, different functions must be kept in mind.

3.14.1. Time Server setting

The `setTimeServer()` function allows the user to set the network time server name or IP. The module has two possible network time servers: the primary time server and the alternate time server. So this function has two different inputs:

- **Index:** 1 or 2. Use index=1 to define the primary time server. Use index=2 to define an alternate time server.
- **Server:** A network timeserver name or IP address. The server will be used to retrieve the current time-of-day.

3.14.2. Time activation flag

The `timeActivationFlag()` function sets the network time-of-day activation flag. If this flag is enabled, the module will connect to the time server and retrieve an updated time reading each time it connects to the network. From that point on, the module will maintain time internally. While the module is online, the network time will be refreshed every two hours. The input for this function permits two options in order to enable or disable the flag: `true` or `false`.

3.14.3. GMT

The `setGMT()` function allows the user to permanently set the module location's Greenwich mean time offset, in hours. The range of this parameter is -12 to 12. The default is 0.

3.14.4. Update RTC settings from WiFi PRO module

The `setTimeFromWIFI()` function allows the user to update the Waspmote's RTC settings when the Time Servers is correctly set and the Activation Flag is enabled.

Example of use:

```
{
   WIFI_PRO.setTimeServer(1, “time.nist.gov”);
   WIFI_PRO.setTimeServer(2, “wwv.nist.gov”);
   WIFI_PRO.timeActivationFlag(true);
   WIFI_PRO.setGMT(2);
   WIFI_PRO.setTimeFromWIFI();
}
```

Example of setting the RTC time from NTP server:
www.libelium.com/development/waspmote/examples/wifi-pro-22-time

3.15. Multiple SSIDs

The WiFi PRO module permits to set up to 10 different SSID settings so the module is able to connect to one of them.

The `setESSID()` function allows the user to set the destination Wireless LAN Service Set Identifier (SSID) string into position 'n' in the ten-profile array. The location of an SSID within the list defines its priority, where the first SSID has the top priority. The SSIDs must be configured consecutively. For example, if the first and third SSIDs are set but the second is not, the module ignores the third SSID. For example, if the module is connected to an AP having an SSID value defined by the fourth SSID, and that SSID is set to a different value using this function, the change will take effect immediately and the module will attempt to associate with an AP having the new SSID. On the other hand, if the module is not currently connected to an AP with SSID defined by the fourth SSID and the value of the fourth SSID is changed, the change will take effect only upon the next connection attempt. This function expects two inputs:

- **Profile:** There are ten constants in the library used for indexing the different profiles:
  - `PROFILE_0`
  - `PROFILE_1`
  - `PROFILE_2`
  - `PROFILE_3`
  - `PROFILE_4`
  - `PROFILE_5`
  - `PROFILE_6`
  - `PROFILE_7`
  - `PROFILE_8`
  - `PROFILE_9`

- **ESSID:** The destination SSID. It must be configured in the module to successfully communicate with that AP.

The `setPassword()` function allows the user to configure the security keys for each one of the defined profiles. This function needs three inputs:

- **Profile:** The index of the profile to set.
- **Security Mode:**
  - OPEN: No security
  - WEP64: WEP 64-bit
  - WEP128: WEP 128-bit
  - WPA: WPA-PSK
  - WPA2: WPA2-PSK
- **Password:**
  - If Security Mode = WPA/WPA2: This is the pass-phrase to be used in generating the PSK encryption key. The allowed value for pass is an ASCII string containing 8 to 63 characters.
  - If Security Mode = WEP64: This key can contain up to 10 characters (defining 5 bytes) where each byte is described by two ASCII characters in the range ['0' to '9'], ['A' to 'F'] or ['a' to 'f'].
  - If Security Mode = WEP128: This key can contain up to 26 characters (defining 13 bytes) where each byte is described by two ASCII characters in the range ['0' to '9'], ['A' to 'F'] or ['a' to 'f'].

In the case of using multiple SSIDs it is mandatory to use the `isConnectedMultiple()` function to check if the module has joined any of those APs. The user must keep in mind that this process can take several seconds which implies a great waste of energy.

Example of multiple SSIDs:

www.libelium.com/development/waspmote/examples/wifi-pro-23-multiple-ssid

3.16. Roaming mode

When set to operate in Roaming mode, the module can roam seamlessly among Access Points (APs) sharing the same SSID and the same security configuration without interrupting its IP connectivity. The WiFi PRO module also has a monitoring mechanism that is sensitive to drops in AP signal strength. When the module detects such a drop, it automatically starts searching for APs in its vicinity that have a stronger signal, while remaining connected to the current AP.

The following functions are required to set the module to Roaming mode:

- The `roamingMode()` function enables or disables the Roaming mode.
- The `setScanInterval()` function sets the time interval between consecutive scans that the module performs for APs in its vicinity. The range is 1 to 3600 seconds. The default is 5 seconds.
- The `setLowThreshold()` function sets a low SNR threshold for the module in Roaming mode. The range is 0 to 254 dB. The default is 10 dB.
- The `setHighThreshold()` function sets a high SNR threshold for the module in Roaming mode. The range is 10 to 255 dB. The default is 30 dB.

Example of Roaming mode:

www.libelium.com/development/waspmote/examples/wifi-pro-24-roaming-mode

3.16.1. Behavior following a hardware or software reset

After power-up, hardware or software reset, the module starts scanning for APs in its vicinity at intervals set by the `setScanInterval()` function. The module reads the value set in the ESSID parameter and acts accordingly. The module attempts to connect to an AP whose ESSID is listed first in the array of ESSID profiles. If several APs having that same ESSID exist, the module attempts to connect to the one having the strongest signal. If association succeeds, the module stops scanning and activates its DHCP client. It then monitors the SNR level of the AP it is associated with.

3.16.2. Behavior when AP signal becomes weak

When the beacon signal of the AP with which the module is associated becomes weak (SNR drops below the level set by the `setLowThreshold()` function), the module starts its periodic scan for APs having SNR above the threshold set by the `setHighThreshold()` function.

The WiFi PRO module attempts to connect to the AP that appears first on the list of ESSIDs specified in the ESSID profile array while remaining connected to the current AP. If association with the new AP fails, the module continues scanning until it succeeds connecting to an AP with a stronger signal.

When in Roaming mode, the module does not restart its DHCP client process for new connections.

When the module is not in Roaming mode, it remains connected to an AP as long as it has an open active socket, or until triggered by a Link Lost event. When not in Roaming mode, the module ignores any decrease in AP signal strength while having open active sockets.

When the module is not in Roaming mode and no active sockets are open, it starts periodic scanning for APs having an SNR level above the high SNR threshold. The module attempts to connect to the AP that has the highest priority. After associating with an AP, it starts its DHCP client and monitors the SNR level of the AP it is associated with.

3.16.3. Behavior in the event of a lost link

If the connection is not active, the module starts periodic scanning for APs and attempts to connect to an AP having the highest priority. After associating to an AP, it starts its DHCP client and monitors the SNR level of the AP it is associated with.

If the connection is active, the module waits for an IP activity command from the host. When such a command is sent, it performs a software reset and starts scanning for APs. The module responds with ERROR (074) to indicate that the current connection has been lost.

3.17. Firmware version

The `getFirmwareVersion()` function allows the user to query the device firmware version. The attribute _ firmwareVersion permits to access to the string that stores the firmware version of the module.

Example of use:

```
{

      WIFI_PRO.getFirmwareVersion ();

}
```

Related variable:

`WIFI_PRO._firmwareVersion` → Stores the firmware version

Example of WIFI_PRO firmware version function:

www.libelium.com/development/waspmote/examples/wifi-pro-25-firmware-version/

4. Certifications

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

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

"Plug & Sense! WiFi" is certified for:

- CE (Europe)
- FCC (US)
- IC (Canada)
- ANATEL (Brazil)
- RCM (Australia)

Figure: Certifications of the Plug & Sense! WiFi product line

You can find all the certification documents at:
http://www.libelium.com/legal

5. Code examples and extended information

In the Waspmote Development section you can find complete examples:

www.libelium.com/development/waspmote/examples

```
/*
 *  ------ WIFI Example --------
 *
 *  Explanation: This example shows how to join an access point (AP)
 *
 *  Copyright (C) 2016 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *  Version:           0.1
 *  Design:            David Gascon
 *  Implementation:    Yuri Carmona
 */

#include <WaspWIFI_PRO.h>

uint8_t socket = SOCKET0;
uint8_t error;
uint8_t status;
unsigned long previous;

void setup()
{
  USB.println(F(“Start program”));
}

void loop()
{
  //////////////////////////////////////////////////
  // 1. Switch ON
  //////////////////////////////////////////////////
  error = WIFI_PRO.ON(socket);

  if (error == 0)
  {
    USB.println(F(“1. WiFi switched ON”));
  }
  else
  {
    USB.println(F(“1. WiFi did not initialize correctly”));
  }
  //////////////////////////////////////////////////
  // 2. Join AP
  //////////////////////////////////////////////////

  // get actual time
  previous = millis();

  // check connectivity
  status = WIFI_PRO.isConnected();

  // Check if module is connected
  if (status == true)
  {
    USB.print(F(“2. WiFi is connected OK.”));
    USB.print(F(“ Time(ms):”));
    USB.println(millis()-previous);

    error = WIFI_PRO.ping(“www.google.com”);

    if (error == 0)
    {
      USB.print(F(“3. PING OK. Round Trip Time(ms)=”));
      USB.println( WIFI_PRO._rtt, DEC );
    }
    else
    {
      USB.println(F(“3. Error calling ‘ping’ function”));
    }
  }
  else
  {
    USB.print(F(“2. WiFi is connected ERROR.”));
    USB.print(F(“ Time(ms):”));
    USB.println(millis()-previous);
  }

  //////////////////////////////////////////////////
  // 3. Switch OFF
  //////////////////////////////////////////////////
  WIFI_PRO.OFF(socket);
  USB.println(F(“4. WiFi switched OFF\n\n”));
  delay(10000);
}
```

6. API changelog

Keep track of the software changes on this link:
www.libelium.com/development/waspmote/documentation/changelog/#WiFi-PRO

7. Documentation changelog

From v7.0 to v7.1

- Added new functions to the “Software” section
- Added a new section to show the user how to connect the module to Waspmote
- Improved the “Configure the password” section
- WiFi channels description fixed