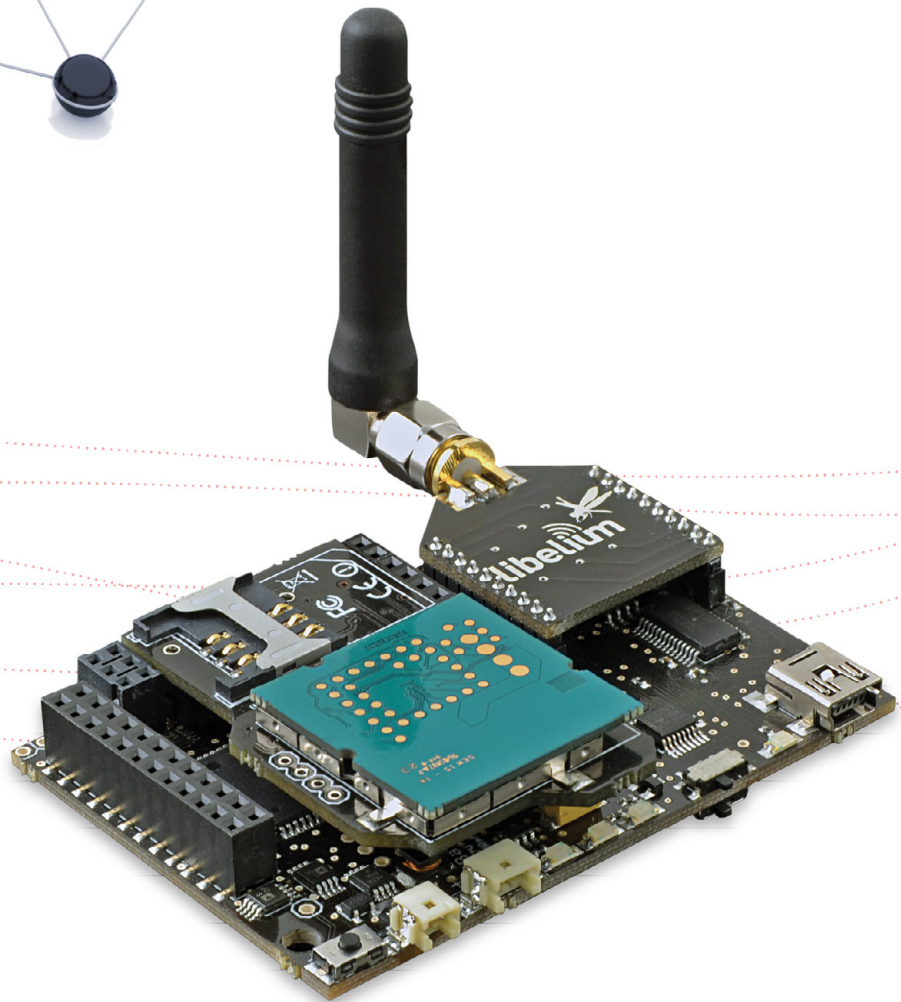
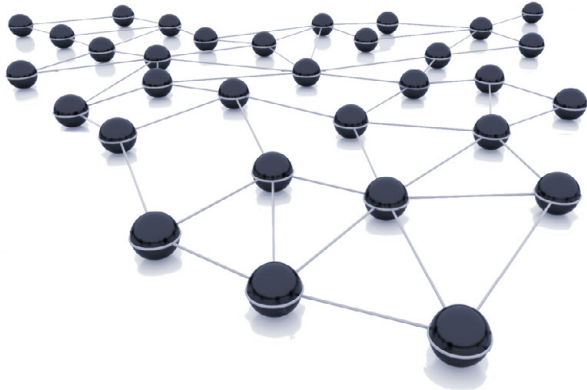


Wasp mote LoRaWAN Networking Guide



Document Version: v4.3 - 05/2016
© Libelium Comunicaciones Distribuidas S.L.

INDEX

1. Introduction	5
1.1. Technology overview	8
2. Hardware	9
2.1. Specifications	9
2.1.1. LoRaWAN 868	9
2.1.2. LoRaWAN 900	10
2.2. Power consumption	11
2.2.1. LoRaWAN 868	11
2.2.2. LoRaWAN 900	11
2.3. Time consumption	11
2.4. Expansion Radio Board	12
3. Software	14
3.1. Wasp mote libraries	14
3.1.1. Wasp mote LoRaWAN files	14
3.1.2. Class constructor	14
3.1.3. API constants	14
3.1.4. API variables	15
3.1.5. API functions	16
3.2. Module system management features	16
3.2.1. Switch on	16
3.2.2. Switch off	17
3.2.3. Module software reset	17
3.2.4. Module factory reset	17
3.2.5. Preprogrammed unique identifier (EUI)	18
3.3. LoRaWAN mode features	18
3.3.1. Operational ISM bands	18
3.3.2. Send data to a LoRaWAN gateway	19
3.3.3. Receiving data from a LoRaWAN gateway	21
3.3.4. Join a network	21
3.3.5. Save configuration	22
3.3.6. Enable LoRaWAN mode	22
3.3.7. Device EUI	23
3.3.8. Device address	23
3.3.9. Application session key	24
3.3.10. Network session key	25
3.3.11. Application key	26
3.3.12. Power level	26

3.3.13. Adaptive data rate (ADR).....	27
3.3.14. Data rate.....	27
3.3.15. Transmission retries	28
3.3.16. Uplink counter	28
3.3.17. Downlink counter.....	29
3.3.18. Channel parameters	29
3.3.19. Duty cycle prescaler.....	32
3.3.20. Margin.....	33
3.3.21. Gateway number	33
3.4. P2P mode – Direct communication between nodes	33
3.4.1. Enable P2P mode	33
3.4.2. Send data	33
3.4.3. Receive data	34
3.4.4. Power level	34
3.4.5. Spreading Factor	35
3.4.6. Frequency deviation	35
3.4.7. Preamble length	35
3.4.8. CRC header	36
3.4.9. Coding Rate.....	36
3.4.10. Bandwidth	36
3.4.11. Frequency.....	37
3.4.12. Signal to noise ratio (SNR)	37
3.5. Hybrid LoRaWAN / P2P mode.....	38

4. LoRaWAN back-ends 39

4.1. Wasp mote recommended configuration.....	39
4.1.1. LoRaWAN 868	39
4.1.2. LoRaWAN 900	39
4.2. Actility	40
4.2.1. Device registration.....	40
4.2.2. Wasp mote programming	40
4.3. OrbiWise	41
4.3.1. Device registration.....	41
4.3.2. Data downlink	43
4.3.3. Wasp mote programming	43
4.4. LORIoT	44
4.4.1. Device registration.....	44
4.4.2. Data downlink	46
4.4.3. Wasp mote programming	46

5. When is LoRaWAN recommended?	47
6. Certifications.....	48
7. Code examples and extended information	50
8. API changelog	55
9. Documentation change	56

1. Introduction

The Libelium LoRaWAN module has been integrated into the main sensor lines Waspote OEM and Plug & Sense!, so now you can create your own **Low Power Wide Area Network (LPWAN)**.

LoRaWAN is a new, private and spread-spectrum modulation technique which allows sending data at extremely low data-rates to extremely long ranges. The low data-rate (down to few bytes per second) and LoRaWAN modulation lead to very low receiver sensitivity (down to **-136 dBm**), which combined to an output power of **+14 dBm** means extremely large link budgets: up to **150 dB**, what means more than **22 km (13.6 miles)** in **LOS** links and up to **2 km (1.2 miles)** in **NLOS** links in urban environment (going through buildings).



Figure: Waspote LoRaWAN

Libelium's LoRaWAN 868 module works in both **868** and **433 MHz** ISM bands and the LoRaWAN 900 module works in **900 MHz** ISM band, which makes them suitable for virtually any country. Those frequency bands are lower than the popular 2.4 GHz band, so path loss attenuation is better in LoRaWAN. In addition, 433, 868 and 900 MHz are bands with much fewer interference than the highly populated 2.4 GHz band. Besides, these low frequencies provide great penetration in possible materials (brick walls, trees, concrete), so these bands get less loss in the presence of obstacles than higher band.

With the LoRaWAN modules we can send the data directly to any **Base Station (BS)** that is LoRaWAN compatible. Some companies already offering solutions are: Kerlink, Link-Labs, Multitech, Cisco, Augtek, Manthink, Gupsy, Gemteck, ExpEmb, Embedded Planet, Calao, RFI, etc. In order to visualize the information we will need also a Cloud platform where the data has to be sent. Normally when you acquire a BS you can install your preferred SW packet in order to make it work against the Cloud platform. We tested the LoRaWAN radios with three Cloud platforms: Activity, Orbiwise and Loriot, you can find more information about the configuration in this tutorial.

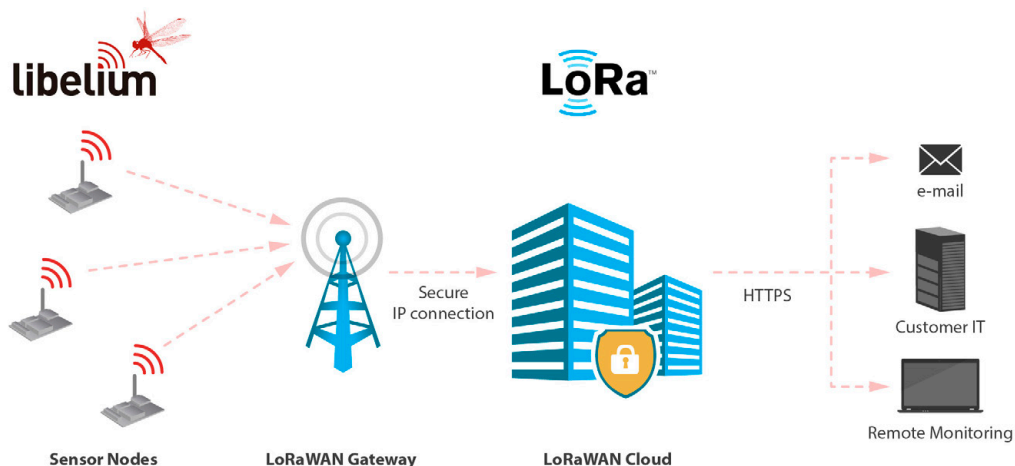


Figure: LoRaWAN network

Libelium currently offers two options of this type of radio technology: LoRa and LoRaWAN

- **LoRa** contains only the link layer protocol and is perfect to be used in P2P communications between nodes. LoRa modules are a little cheaper than the LoRaWAN ones.
- **LoRaWAN** includes the network layer too so it is possible to send the information to any Base Station already connected to a Cloud platform. LoRaWAN modules may work in different frequencies by just connecting the right antenna to its socket.

As well as the LoRaWAN to Base Station mode, the modules may be used in two different more configurations.

- P2P Mode - Direct Communication between nodes (LAN Interface)
- Hybrid Mode - LoRaWAN / P2P (P2P + GW to LoRaWAN Network)
- In the **P2P Mode** nodes may connect directly among them and send messages directly at no cost (as they are not using the LoRaWAN Network but just direct radio communication). This is useful as we can create secondary networks at any time as we don't need to change the firmware but just use specific AT Commands in the current library. **This mode works without the need of a Base Station or a Cloud account** so in case you don't want to purchase any license (or renew the license after the initial period) you will be able to keep on using the modules this way. For more info go to the section P2P Mode.

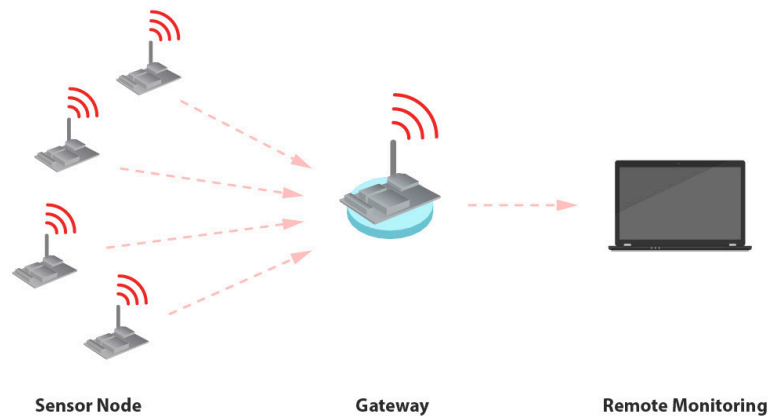


Figure: P2P mode

In the **Hybrid Mode** we use a combination of the LoRaWAN and P2P modes allowing to send just certain messages using the LoRaWAN Network. In this case we use one node as GW of the network (P2P + LoRaWAN mode) and the rest of the nodes in P2P mode. **This mode may work using just one LoRaWAN License.** For more info go to the section Hybrid Mode.

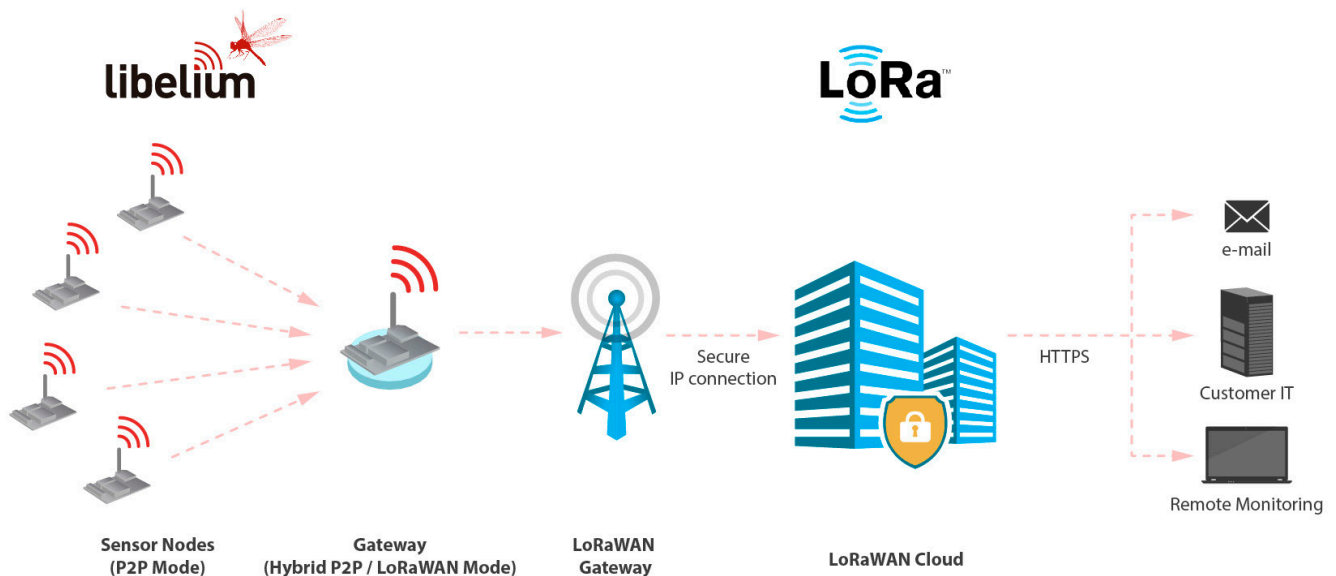


Figure: P2P mode

Market Ready Technology

Libelium offers two types of sensor platforms: Wasp mote OEM and Plug & Sense!.

- Wasp mote 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/wasp-mote/>
- 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. <http://www.libelium.com/products/plug-sense/>

In accordance with the 1999/5/EC (R&TTE) and the EU Directive 2011/65/EU (RoHS2), Libelium Comunicaciones Distribuidas S.L. declares that the "Plug & Sense! LoRaWAN 868" device conforms to the following regulations:

- EN 301 489-1(1.9.2)- (1.6.1)
- EN 55022:2010
- EN 60950-1: 2006
- A11: 2009 + A1: 2010 + A12: 2011 + Ac: 2011 + A2: 2013

Plug & Sense! Sigfox 900 / 915 is certified for the US (FCC) and Canada (IC). Specifically under the Part 15 Spread Spectrum Transmitter (FCC Rule Part 15 C) and the Industry Canada Regulation (IC) - (RSS-210) respectively.



1.1. Technology overview

LoRaWAN is a Low Power Wide Area Network (LPWAN) specification intended for wireless battery operated devices in regional, national or global network. LoRaWAN target key requirements of Internet of things such as secure bi-directional communication, mobility and localization services. This standard will provide seamless interoperability among smart Things without the need of complex local installations and gives back the freedom to the user, developer, businesses enabling the role out of Internet of Things.

LoRaWAN network architecture is typically laid out in a star-of-stars topology in which **gateways** is a transparent bridge relaying messages between **end-devices** and a central **network server** in the back-end. Gateways are connected to the network server via standard IP connections while end-devices use single-hop wireless communication to one or many gateways.

Communication between end-devices and gateways is spread out on different **frequency channels** and **data rates**. The selection of the data rate is a trade-off between communication range and message duration. Due to the spread spectrum technology, communications with different data rates do not interfere with each other and create a set of “virtual” channels increasing the capacity of the gateway. To maximize both battery life of the end-devices and overall network capacity, the LoRaWAN network server is managing the data rate and RF output for each end-device individually by means of an **adaptive data rate** (ADR) scheme.

National wide networks targeting internet of things such as critical infrastructure, confidential personal data or critical functions for the society has a special need for secure communication. This has been solved by several layer of encryption:

- Network Session Key (128-bit key) ensures security on network level
- Application Session Key (128-bit key) ensures end-to-end security on application level
- Application Key (128-bit key) ensures end-to-end security on application level (only OTAA procedure)

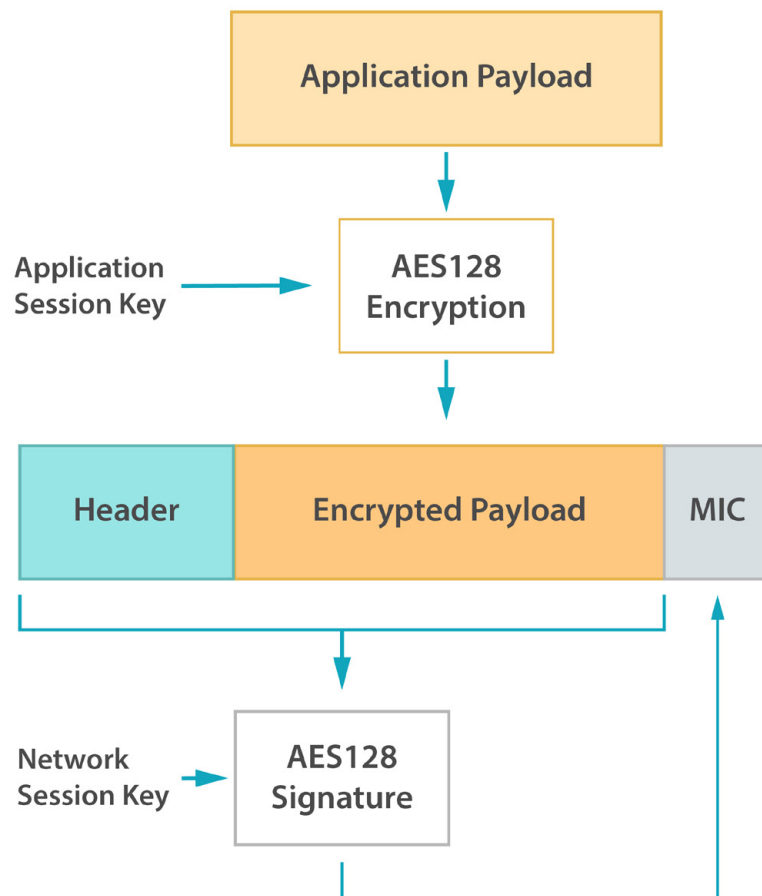


Figure: LoRaWAN security overview

2. Hardware

2.1. Specifications

The LoRaWAN module is managed via UART and it can be connected to SOCKET0 or SOCKET1.

2.1.1. LoRaWAN 868

The main features of the module are listed below:

- **Protocol:** LoRaWAN 1.0, Class A
- **LoRaWAN-ready**
- **Frequency:** 868 MHz and 433 MHz ISM frequency bands.
- **TX Power:** up to +14 dBm
- **Sensitivity:** down to -136 dBm
- **Range:** >15 km at suburban and >5 km at urban area. Typically, each base station covers some km. Check the LoRaWAN Network in your area.
- **Chipset consumption:** 38.9 mA
- **Radio Bit Rate:** from 250 to 5470 bps
- **Receiver:** purchase your own base station or use networks from LoRaWAN operators

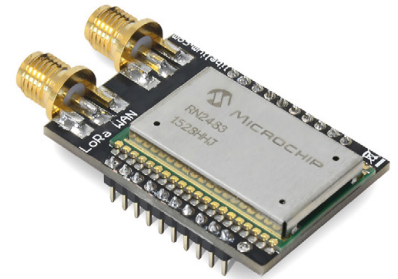


Figure: LoRaWAN 868 module



Figure: LoRaWAN 868 module with normal antenna



Figure: LoRaWAN 868 module with 0 dBi antenna

2.1.2. LoRaWAN 900

The main features of the module are listed below:

- **Protocol:** LoRaWAN 1.0, Class A
- **LoRaWAN-ready**
- **Frequency:** 900-930 MHz ISM band
- **TX Power:** up to +18.5 dBm
- **Sensitivity:** down to -136 dBm
- **Range:** >15 km at suburban and >5 km at urban area. Typically, each base station covers some km. Check the LoRaWAN Network in your area.
- **Chipset consumption:** 124.4 mA
- **Radio Bit Rate:** from 250 to 12500 bps
- **Receiver:** purchase your own base station or use networks from LoRaWAN operators



Figure: LoRaWAN 900 module



Figure: LoRaWAN 868 module with normal antenna



Figure: LoRaWAN 868 module with 0 dBi antenna

Note: The 900 MHz version does not support operation on the 433 MHz band.

Note: The user must check the allowed bands, channels and transmission power, in order to respect the regulations in the operation country.

Libelium commercializes different items depending on the band the user wants to use. In the case of 868 and 433, the module is the same, but the antenna is different for each band. The module for 868 and 433 MHz includes 2 RP-SMA connectors for the antenna. One is for the 868 band and the other for the 433 band. A sticker on the bottom of the modules specifies clearly where to screw the antenna.

Note: Any LoRaWAN module is provided with a special antenna (for 433 **or** for 868 **or** for 900 MHz), which enables maximum range. The only exception is Smart Parking; in this case the antenna is smaller, to fit inside the enclosure.

Note: Due to the propagation characteristics of the sub-GHz bands, the near field effect could make that 2 modules cannot communicate if they are placed very close ($< 1\text{ m}$). We suggest to keep a minimum distance of 3 or 4 meters between modules.

2.2. Power consumption

2.2.1. LoRaWAN 868

The LoRaWAN 868 module is powered at 3.3 V. The next table shows the module's average current consumption in different states of the module.

State	Power Consumption
OFF	2.8 mA
Transmitting data	38.9 mA
Receiving data	14.2 mA

Figure: Power consumption table

2.2.2. LoRaWAN 900

The LoRaWAN 900 module is powered at 3.3 V. The next table shows the module's average current consumption in different states of the module.

State	Power Consumption
ON	2.7 mA
Transmitting data	124.4 mA
Receiving data	13.5 mA

Figure: Power consumption table

2.3. Time consumption

The elapsed periods defined in this chapter take into account the following steps depending on the case:

- Join to a network and send unconfirmed data
- Join to a network and send confirmed data

These periods of time depend on the data rate set which is defined by the spreading factor and signal bandwidth configured.

Transmit mode	Time elapsed
Send unconfirmed at 5470 bps	~ 2.8 seconds
Send unconfirmed at 250 bps	~ 4.2 seconds
Send confirmed at 5470 bps	~ 1.7 seconds
Send confirmed at 250 bps	~ 4.2 seconds

Note: When transmitting in ISM frequency bands, the user must ensure that the communication is not exceeding the permitted time using the chosen frequency channel (for example, 1% of time). This depends on the local regulations (CE, FCC, etc). It is the responsibility of the user to know the allowed time of use in the occupied frequency band and respect it. Ignoring this, could lead to considerable penalties. Also, a LoRaWAN back-end operator could cut off the service or apply extra fees, if they detect that the user is exceeding the maximum number of frames or data in a period of time.

2.4. Expansion Radio Board

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

Remark: GPRS Pro, GPRS+GPS and 3G/GPRS modules do not need the Expansion Board to be connected to Waspote. 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.

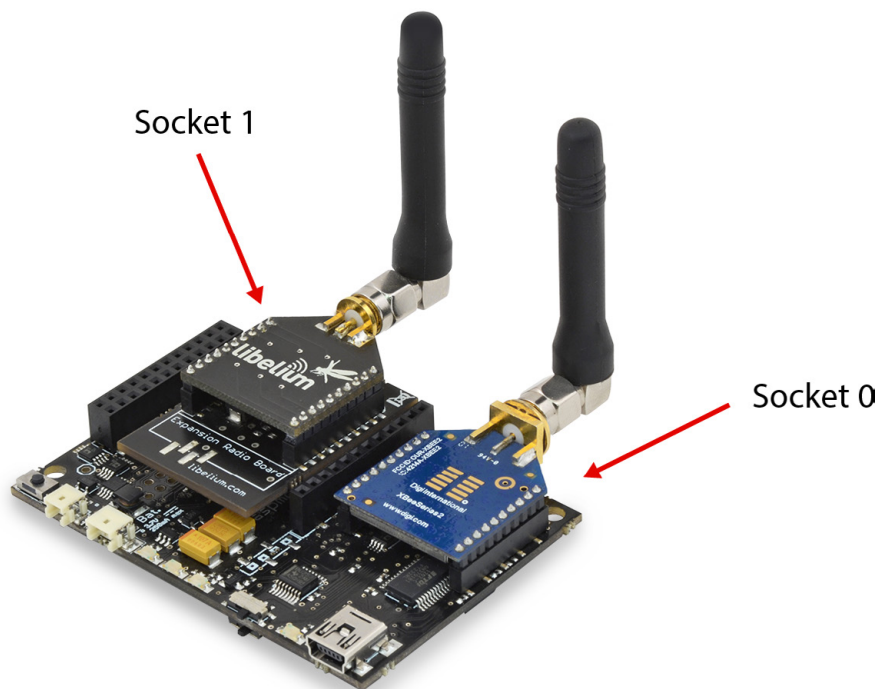


Figure: Use of the Expansion Board

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**: `LoRaWAN.ON(SOCKET0);`

Selecting **SOCKET1**: `LoRaWAN.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 NO₂/O₃ 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.

3. Software

The Waspote device communicates with the LoRaWAN module via UART. So different commands are sent from the microcontroller unit to the module so as to perform different tasks.

3.1. Waspote libraries

3.1.1. Waspote LoRaWAN files

The files related to the LoRaWAN libraries are:

```
WaspLoRaWAN.h  
WaspLoRaWAN.cpp
```

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

```
#include <WaspLoRaWAN.h>
```

3.1.2. Class constructor

To start using the Waspote LoRaWAN library, an object from the 'WaspLoRaWAN' class must be created. This object, called [LoRaWAN](#), is already created by default inside Waspote LoRaWAN library. It will be used through this guide to show how Waspote 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
LORAWAN_ANSWER_OK	Successful response to a function
LORAWAN_ANSWER_ERROR	Erratic response to a function
LORAWAN_NO_ANSWER	No response to a function
LORAWAN_INIT_ERROR	Required keys to join to a network were not initialized
LORAWAN_LENGTH_ERROR	Data to be sent length limit exceeded
LORAWAN_SENDING_ERROR	Server did not response
LORAWAN_NOT_JOINED	Module has not joined a network
LORAWAN_INPUT_ERROR	Invalid input parameter
LORAWAN_VERSION_ERROR	The module does not support this function
RN2483_MODULE	LoRaWAN module plugged is LoRaWAN 868
RN2903_MODULE	LoRaWAN module plugged is LoRaWAN 900

3.1.4. API variables

The variables used inside functions and Wasmote codes are:

Constant	Description
<code>_buffer</code>	The buffer of memory used for storing the responses from the module
<code>_length</code>	The useful length of the buffer
<code>_def_delay</code>	The time to wait after sending every command until listen for a response
<code>_baudrate</code>	The baudrate to be used when the module is switched on
<code>_uart</code>	The selected UART (regarding the socket used: SOCKET0 or SOCKET1)
<code>_adr</code>	The adaptive data rate state (on or off)
<code>_eui</code>	The buffer used for storing the preprogrammed globally unique identifier from the module's hardware
<code>_devEUI</code>	The buffer used for storing the globally unique identifier for the module (software programmable)
<code>_appEUI</code>	The buffer used for storing the application identifier for the module
<code>_nwksKey</code>	The buffer used for storing the network session key for the module
<code>_appSKey</code>	The buffer used for storing the application session key for the module
<code>_appKey</code>	The buffer used for storing the application key for the module
<code>_devAddr</code>	The buffer used for storing network device address for module
<code>_band</code>	The buffer used for storing the frequency band
<code>_margin</code>	The demodulation margin received in the last Link Check Answer frame
<code>_gwNumber</code>	The number of gateways successfully received the last Link Check Answer frame
<code>_freq</code>	The buffer used for storing the operating frequency for every channel
<code>_radioFreq</code>	The transceiver operating frequency
<code>_radioFreqDev</code>	The transceiver frequency deviation
<code>_preambleLength</code>	The preamble length for transceiver
<code>_dCycle</code>	The buffer used for storing the operating duty cycle for every channel
<code>_drMin</code>	The minimum operating data rate range for every channel
<code>_drMax</code>	The maximum operating data rate range for every channel
<code>_dCyclePS</code>	The duty cycle prescaler (which can only be configured by the server)
<code>_crcStatus</code>	The CRC status to determine if it is to be included during operation
<code>_powerIndex</code>	The output power to be used on LoRaWAN transmissions
<code>_dataRate</code>	The data rate to be used on LoRaWAN transmissions
<code>_retries</code>	The number of retransmissions for an uplink
<code>_upCounter</code>	The value of the uplink frame counter
<code>_downCounter</code>	The value of the downlink frame counter
<code>_radioPower</code>	The output power level used by the transceiver
<code>_radioSF</code>	The spreading factor to be used by the transceiver
<code>_radioRxBW</code>	The receiving bandwidth used by the transceiver
<code>_radioCR</code>	The coding rate used by the transceiver
<code>_radioWDT</code>	The time to be used by the transceiver watchdog timer
<code>_radioBW</code>	The value used for the transceiver bandwidth

<code>_radioSNR</code>	The SNR value for the last received packet by the transceiver
<code>_status</code>	The status of every LoRaWAN channel
<code>_data</code>	The buffer of memory used for storing data received from the back-end
<code>_port</code>	The port where data was received
<code>_dataReceived</code>	The flag used to inform if any data was received
<code>_version</code>	The version of the module plugged into Waspote

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 `LoRaWAN`.

All public functions return one of these possible values:

- `LORAWAN_ANSWER_OK` = 0
- `LORAWAN_ANSWER_ERROR` = 1
- `LORAWAN_NO_ANSWER` = 2
- `LORAWAN_INIT_ERROR` = 3
- `LORAWAN_DATA_LENGTH_ERROR` = 4
- `LORAWAN_SENDING_ERROR` = 5
- `LORAWAN_NOT_JOINED` = 6
- `LORAWAN_INPUT_ERROR` = 7
- `LORAWAN_VERSION_ERROR` = 8

3.2. Module system management features

3.2.1. Switch on

The `ON()` function allows to switch on the LoRaWAN module, it opens the MCU UART for communicating with the module and it automatically enters into command mode.

In addition, when the module has been powered and communication is opened, this function checks whether a module is plugged to the socket and which type of module has been plugged. This check is done within every function that reboots the module such as "ON", "reset" or "factoryReset".

After this step the module will be able to receive commands to configure it or send packets. It is necessary to indicate the socket that it is being used: `SOCKET0` or `SOCKET1`.

Example of use for `SOCKET0`:

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

Related variable:

`LoRaWAN._version` → Stores the module's version



Figure: LoRaWAN module in SOCKET0

3.2.2. Switch off

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

Example of use for SOCKET0:

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

3.2.3. Module software reset

The `reset()` function allows the user to reset and restart the LoRaWAN module. The stored internal configurations will be loaded automatically upon reboot.

Example of use:

```
{
  LoRaWAN.reset();
}
```

Related variable:

`LoRaWAN._version` → Stores the module's version

3.2.4. Module factory reset

The `factoryReset()` function allows the user to reset the module's configuration data and user EEPROM to factory default values and restart the module.

Example of use:

```
{  
    LoRaWAN.factoryReset();  
}
```

Related variable:

`LoRaWAN._version` → Stores the module's version

Examples of LoRaWAN configuration:

www.libelium.com/development/waspmote/examples/lorawan-01a-configure-lorawan-868

www.libelium.com/development/waspmote/examples/lorawan-01b-configure-lorawan-900

3.2.5. Preprogrammed unique identifier (EUI)

The `getEUI()` function allows the user to query the preprogrammed EUI node address from the module. The preprogrammed EUI node address is a read-only value and cannot be changed or erased. It is a global unique 64-bit identifier.

Example of use:

```
{  
    LoRaWAN.getEUI();  
}
```

Related variable:

`LoRaWAN._eui` → Stores the module's EUI

3.3. LoRaWAN mode features

3.3.1. Operational ISM bands

The `resetMacConfig()` function allows the user to reset the software LoRaWAN stack and initialize it with the parameters for the selected band: 433 MHz, 868 MHz or 900 MHz.

The `getBand()` function allows the user to query the current frequency band of operation. This function is not available for the LoRaWAN 900 module since it can only work in the 902-928 MHz ISM band. The attribute `saveConfig()` function due to keep the band configuration after a reboot. Value can be either 433 or 868.

Example of use:

```
{  
    LoRaWAN.resetMacConfig(433);  
    LoRaWAN.getBand();  
}
```

Related variable:

`LoRaWAN._band` → Stores the current frequency band of operation

3.3.2. Send data to a LoRaWAN gateway

- **Sending unconfirmed packets**

The `sendUnconfirmed()` function allows the user to transmit data on a specified port number. This function will not expect any acknowledgement back from the server. It is necessary to indicate the port to use. The range is from 1 to 223. The second input of the sending function is the payload of the packet to send. The payload must be specified in hexadecimal format as a string. The length of the payload capable of being transmitted is dependent upon the set data rate.

Please refer to the [LoRaWAN Specification](#) for the payload length values.

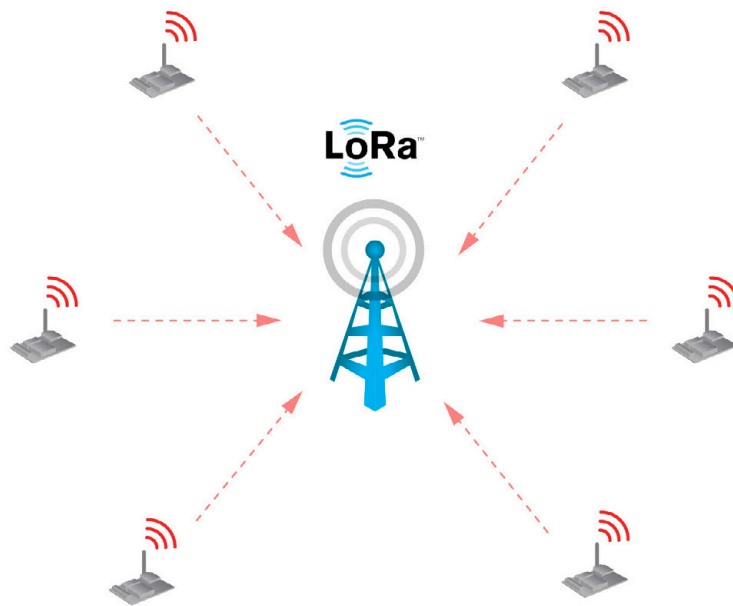


Figure: Sending unconfirmed packets without ACK

Example of use:

```
{
  uint8_t port = 1;
  char data[] = "010203040506070809";

  LoRaWAN.sendUnconfirmed( port, data);
}
```

Example of sending a packet without ACK:

www.libelium.com/development/waspmote/examples/lorawan-02-send-unconfirmed
www.libelium.com/development/waspmote/examples/lorawan-08-join-otaa-send-unconfirmed

- **Sending confirmed packets (with ACK)**

The `sendConfirmed()` function allows the user to transmit data on a specified port number. This function will expect an acknowledgement from the server. If no ACK is received, the message will be retransmitted automatically up to a maximum of times specified by the `setRetries()` function. It is necessary to indicate the port to use. The range is from 1 to 223. The second input of the sending function is the payload of the packet to send. The payload must be specified in hexadecimal format as a string. The length of the payload capable of being transmitted is dependent upon the set data rate.

Please refer to the [LoRaWAN Specification](#) for the payload length values.

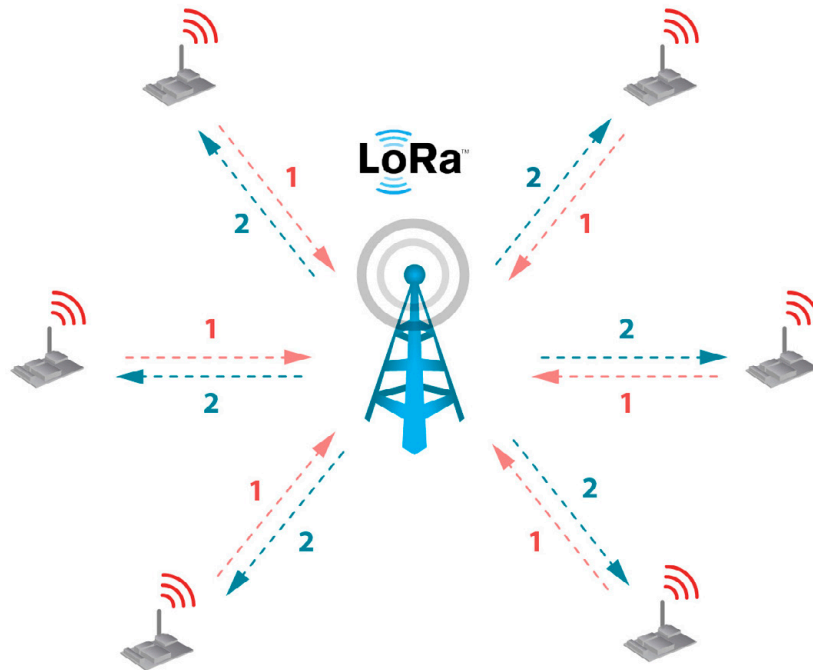


Figure: Sending confirmed packets with ACK

Example of use:

```
{
  uint8_t port = 1;
  char data[] = "010203040506070809";

  LoRaWAN.sendConfirmed( port, data);
}
```

Example of sending a packet with ACK:

www.libelium.com/development/waspmote/examples/lorawan-03-send-confirmed

www.libelium.com/development/waspmote/examples/lorawan-09-join-otaa-send-confirmed

3.3.3. Receiving data from a LoRaWAN gateway

Note that not every back-end is able to send data to end devices. The back-ends which are able to perform data downlink use the ACK process to send a special ACK frame containing data. So once the user programs a downlink process, this downlink data will be received by the target module the next time it performs a transmission.

There is no specific function to receive data from back-ends in this library. The `sendUnconfirmed()` and the `sendConfirmed()` functions check if any special ACK is received and store data if any has been received. In case data has been received, `_dataReceived` flag will be set to true.

Related variables:

`LoRaWAN._port` → Stores the port used by the back-ends to send data

`LoRaWAN._data` → Stores the received data

`LoRaWAN._dataReceived` → Data received flag

3.3.4. Join a network

Before sending packets to a gateway, the node must join a network first. The `joinABP()` function allows the user to attempt joining the network using the Activation By Personalization mode (ABP). The `joinOTAA()` function allows the user to attempt joining the network using the Over the Air Activation mode (OTAA). Before joining the network, the specific parameters for activation should be configured depending on the joining procedure.

- Join ABP

Before joining the network, the specific parameters for activation should be configured: device EUI, device address, network session key and application session key.

Example of use:

```
{  
    LoRaWAN.joinABP();  
}
```

- Join OTAA

Before joining the network, the specific parameters for activation should be configured: device EUI, application EUI and application key.

Example of use:

```
{  
    LoRaWAN.joinOTAA();  
}
```

3.3.5. Save configuration

The `saveConfig()` function allows the user to save LoRaWAN Class A protocol configuration parameters to the module's non-volatile memory. This function must be issued after the configuration parameters have been appropriately set.

The LoRaWAN Class A protocol configuration savable parameters are:

- Band
- Uplink Frame Counter
- Downlink Frame Counter
- Data Rate
- Adaptive Data Rate state
- Device EUI
- Application EUI
- Application Key
- Network Session Key
- Application Session Key
- Device Address
- All Channel Parameter
 - Frequency
 - Duty Cycle
 - Data Rate Range
 - Status

There are some exceptions for the LoRaWAN 900 module, it does **not** save:

- Band, since it does not use this parameter
- Data rate
- Channel frequency, preset according to specification
- Duty cycle, since it does not use this parameter

Example of use:

```
{  
  LoRaWAN.saveConfig();  
}
```

Examples of LoRaWAN configuration:

www.libelium.com/development/waspmote/examples/lorawan-01a-configure-lorawan-868

www.libelium.com/development/waspmote/examples/lorawan-01b-configure-lorawan-900

3.3.6. Enable LoRaWAN mode

The `macResume()` function allows the user to enable LoRaWAN mode. After switching on the module it is not mandatory to call this function because the default mode upon reboot is LoRaWAN mode. However, if P2P mode was previously set and the user needs to switch back to LoRaWAN mode, then this function must be called.

Example of use:

```
{  
  LoRaWAN.macResume();  
}
```

3.3.7. Device EUI

The `setDeviceEUI()` function allows the user to set the 64-bit hexadecimal number representing the device EUI. There are two function prototypes which are explained below:

No input device EUI is specified, then the preprogrammed EUI is used as the device EUI.

A user-provided device EUI is specified as input.

The `getDeviceEUI()` function allows the user to query the device EUI which was previously set by the user. The attribute `_devEUI` permits to access to the settings of the module. The default value is 0000000000000000.

Depending on the network to join, it is needed to configure a random device EUI or a fixed device EUI provided by the back-end. This matter relies on the registering process for new devices in each back-end. For further information please go to “LoRaWAN back-ends” chapter.

Example for preprogrammed EUI:

```
{  
  LoRaWAN.setDeviceEUI();  
  LoRaWAN.getDeviceEUI();  
}
```

Example for user-provided device EUI:

```
{  
  LoRaWAN.setDeviceEUI("0102030405060708");  
  LoRaWAN.getDeviceEUI();  
}
```

Related variable:

`LoRaWAN._devEUI` → Stores the previously set device EUI

Examples of LoRaWAN configuration:

www.libelium.com/development/waspmote/examples/lorawan-01a-configure-lorawan-868

www.libelium.com/development/waspmote/examples/lorawan-01b-configure-lorawan-900

3.3.8. Device address

The `setDeviceAddr()` function allows the user to set the 32-bit hexadecimal number representing the device address. This address must be unique to the current network. There are two function prototypes which are explained below:

No input device address is specified, then the last 4 bytes of the preprogrammed EUI are set as device address.

A user-provided device address is specified as input.

The `getDeviceAddr()` function allows the user to query the device address which was previously set by the user. The attribute `_devAddr` permits to access to the settings of the module. The range goes from 00000000 to FFFFFFFF. The default value is 00000000.

Depending on the network to join, it is possible configure a random device address or a fixed device address. This matter depends on the back-end and the registering process for new devices. For further information please go to “LoRaWAN back-ends” chapter.

Example for using the preprogrammed EUI:

```
{
  LoRaWAN.setDeviceAddr();
  LoRaWAN.getDeviceAddr();
}
```

Example for user-provided device address:

```
{
  LoRaWAN.setDeviceAddr("01020304");
  LoRaWAN.getDeviceAddr();
}
```

Related variable:

`LoRaWAN._devAddr` → Stores the previously set device address

Examples of LoRaWAN configuration:

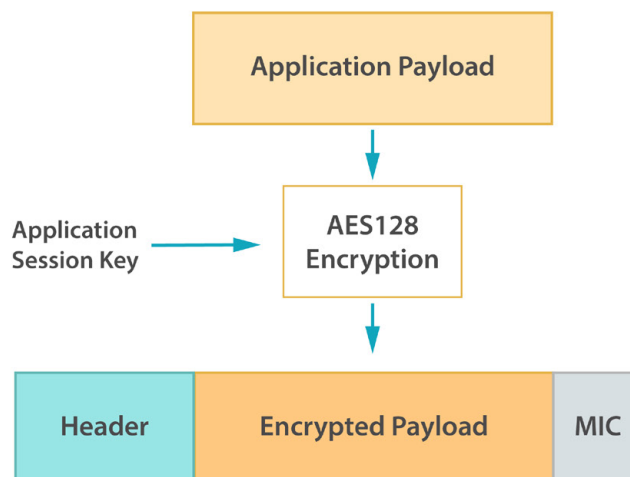
www.libelium.com/development/waspmote/examples/lorawan-01a-configure-lorawan-868

www.libelium.com/development/waspmote/examples/lorawan-01b-configure-lorawan-900

3.3.9. Application session key

The `setAppSessionKey()` function allows the user to set the 128-bit hexadecimal number representing the application session key.

All payloads are encrypted using an AES algorithm with a 128-bit secret key, the Application Session Key. Each end-device has its own unique Application Session Key only known by the end-device and the application server



Use of application session key

The attribute `_appSKey` stores the application session key previously set by the user. The range goes from 0000000000000000 to FFFFFFFFFFFFFFFFFFFFFFFFFF.

Example of use:

```
{
  LoRaWAN.setAppSessionKey("00102030405060708090A0B0C0D0E0F");
}
```


Related variable:

`LoRaWAN._appSKey` → Stores the previously set application session key

Examples of LoRaWAN configuration:

www.libelium.com/development/waspmote/examples/lorawan-01a-configure-lorawan-868

www.libelium.com/development/waspmote/examples/lorawan-01b-configure-lorawan-900

3.3.10. Network session key

The `setNwkSessionKey()` function allows the user to set the 128-bit hexadecimal number representing the network session key.

All frames contain a 32-bit cryptographic Message Integrity Check (MIC) signature computed using the AES algorithm with a 128-bit secret key, the Network Session Key. Each end-device has its own Network Session Key only known by the end-device and the network server.

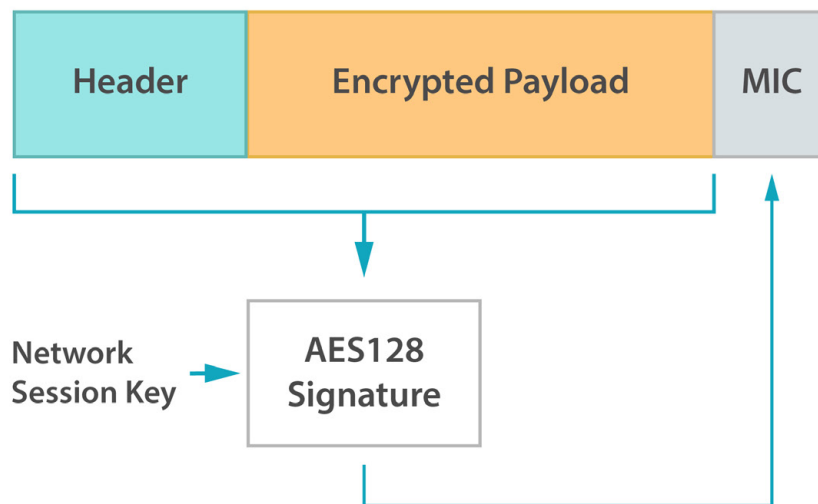


Figure: Use of network session key

The attribute `_nwkSKey` stores the network session key previously set by the user. The range goes from 00000000000000000000 to FFFFFFFF00000000000000000000.

Example of use:

```
{
  LoRaWAN.setNwkSessionKey("00102030405060708090A0B0C0D0E0F");
}
```

Related variable:

`LoRaWAN._nwkSKey` → Stores the previously set network session key

Examples of LoRaWAN configuration:

www.libelium.com/development/waspmote/examples/lorawan-01a-configure-lorawan-868

www.libelium.com/development/waspmote/examples/lorawan-01b-configure-lorawan-900

3.3.11. Application key

The `setAppKey()` function allows the user to set the 128-bit hexadecimal number representing the application key. Whenever an end-device joins a network via OTAA, the Application Key is used to derive the session keys, Network Session Key and Application Session Key, which are specific for that end-device to encrypt and verify network communication and application data.

The attribute `_appKey` stores the application session key previously set by the user. The range goes from 00000000000000000000000000000000 to FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF.

Example of use:

```
{
  LoRaWAN.setAppKey("00102030405060708090A0B0C0D0E0F");
}
```

Related variable:

`LoRaWAN._appKey` → Stores the previously set application session key

Examples of LoRaWAN configuration:

www.libelium.com/development/waspmote/examples/lorawan-01a-configure-lorawan-868

www.libelium.com/development/waspmote/examples/lorawan-01b-configure-lorawan-900

3.3.12. Power level

The `setPower()` function allows the user to set the output power to be used on the next transmissions.

The `getPower()` function allows the user to query the device power index which was previously set by the user. The attribute `_powerIndex` permits to access to the settings of the module. The range goes from 0 to 5 for the 433 MHz frequency band and from 1 to 5 for the 868 MHz frequency band. LoRaWAN 900 power index values can be: 5, 7, 8, 9 or 10.

Power Index	Power Level (868 MHz)	Power Level (433 MHz)
0	N / A	10 dBm
1	14 dBm	7 dBm
2	11 dBm	4 dBm
3	8 dBm	1 dBm
4	5 dBm	-2 dBm
5	2 dBm	-5 dBm

Figure: Power levels table (LoRaWAN 868/433 module)

Power Index	Power Level (900 MHz)
5	20 dBm
7	16 dBm
8	14 dBm
9	12 dBm
10	10 dBm

Figure: Power levels table (LoRaWAN 900 module)

Example of use:

```
{
  LoRaWAN.setPower(1);
  LoRaWAN.getPower();
}
```

Related variable:

`LoRaWAN._powerIndex` → Stores the previously set power

Example of setting power level:

www.libelium.com/development/waspmote/examples/lorawan-04-power-level

3.3.13. Adaptive data rate (ADR)

The `setADR()` function allows the user to enable or disable the adaptive data rate (ADR). The server is informed about the status of the module's ADR in every uplink frame it receives from ADR field in uplink data packet. If ADR is enabled, the server will optimize the data rate and the transmission power based on the information collected from the network: the RSSI / SNR of the last received packets.

The `getADR()` function allows the user to query the device adaptive data rate status. The attribute `_adr` permits to access to the settings of the module. This attribute is set to 'true' if ADR is enabled or 'false' if ADR is disabled.

Example of use:

```
{
  LoRaWAN.setADR("on");
  LoRaWAN.setADR("off");

  LoRaWAN.getADR();
}
```

Related variable:

`LoRaWAN._adr` → Stores the previously set ADR status

Example of setting adaptive data rate:

www.libelium.com/development/waspmote/examples/lorawan-06-adaptive-data-rate

3.3.14. Data rate

The `setDataRate()` function allows the user to set the data rate to be used for the next transmission. The following encoding is used for Data Rate (DR):

Data Rate	Configuration	Indicative physical bit rate [bits/s]
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470

Figure: Data rates table for the LoRaWAN 868 module

Data Rate	Configuration	Indicative physical bit rate [bits/s]
0	LoRa: SF10 / 125 kHz	980
1	LoRa: SF19 / 125 kHz	1760
2	LoRa: SF18 / 125 kHz	3125
3	LoRa: SF7 / 125 kHz	5470
4	LoRa: SF8 / 500 kHz	12500

Figure: Data rates table for the LoRaWAN 900 module

The `getDataRate()` function allows the user to query the device data rate. The attribute `_dataRate` permits to access to the settings of the module.

Example of use:

```
{
  LoRaWAN.setDatarate(0);
  LoRaWAN.getDataRate();
}
```

Related variable:

`LoRaWAN._dataRate` → Stores the previously set data rate

Example of setting data rate:

www.libelium.com/development/waspmote/examples/lorawan-05-data-rate

3.3.15. Transmission retries

The `setRetries()` function allows the user to set the number of retransmissions to be used for an uplink confirmed packet, if no downlink acknowledgment is received from the server.

The `getRetries()` function allows the user to query the number of retransmissions which was previously set by the user. The attribute `_retries` permits to access to the settings of the module. The attribute range is from 0 to 255.

Example of use:

```
{
  LoRaWAN.setRetries (3);
  LoRaWAN.getRetries ();
}
```

Related variable:

`LoRaWAN._retries` → Stores the previously set number of retransmissions

Examples of LoRaWAN configuration:

www.libelium.com/development/waspmote/examples/lorawan-01a-configure-lorawan-868

www.libelium.com/development/waspmote/examples/lorawan-01b-configure-lorawan-900

3.3.16. Uplink counter

The `setUpCounter()` function allows the user to set the uplink frame counter that will be used for the next uplink transmission.

The `getUpCounter()` function allows the user to query the uplink frame counter that will be used for the next uplink transmission. The attribute `_upCounter` permits to access the settings of the module. The attribute range is from 0 to 4294967295.

If the back-end's sequence number check is set to strict, this uplink counter must be synchronized with the back-end uplink counter. The `_upCounter` is saved into the module's memory after every transmission.

Example of use:

```
{  
    LoRaWAN.setUpCounter(10);  
    LoRaWAN.getUpCounter();  
}
```

Related variable:

`LoRaWAN._upCounter` → Stores the previously set uplink frame sequence number

3.3.17. Downlink counter

The `setDownCounter()` function allows the user to set the downlink frame counter that will be used for the next downlink reception.

The `getDownCounter()` function allows the user to query the downlink frame counter that will be used for the next downlink reception. The attribute `_downCounter` permits to access the settings of the module. The attribute range is from 0 to 4294967295.

If the back-end check sequence number function is set to strict, this downlink counter must be synchronized with the back-end downlink counter. The `_downCounter` is saved into the module's memory after every reception.

Example of use:

```
{  
    LoRaWAN.setDownCounter(10);  
    LoRaWAN.getDownCounter();  
}
```

Related variable:

`LoRaWAN._downCounter` → Stores the previously set downlink frame sequence number

3.3.18. Channel parameters

The LoRaWAN 868 module has 16 channels available to be configured. The channel parameters are:

- Frequency
- Duty cycle
- Data rate range
- Status

Channel Number	Parameters	Frequency band	
		868	433
Channel 0	Frequency (Hz)	868100000	433175000
	Duty cycle	302	302
	Data rate range	0-5	0-5
	Status	On	On
Channel 1	Frequency (Hz)	868300000	433375000
	Duty cycle	302	302
	Data rate range	0-5	0-5
	Status	On	On
Channel 2	Frequency (Hz)	868500000	433575000
	Duty cycle	302	302
	Data rate range	0-5	0-5
	Status	On	On
Channel 3 - 15	Frequency (Hz)	0	0
	Duty cycle	65535	65535
	Data rate range	15 -15	15- 15
	Status	Off	Off

Figure: Channel parameters table for LoRaWAN 868

The LoRaWAN 900 module has 72 channels with a preset fixed frequency for every channel. Data rate range and channel status are the only settable parameters.

Channel Number	Parameters	Default Values
Channel 0-63	Frequency (Hz)	$902300000 + 200000 * \text{channel Index}$
	Data rate range (min - max)	0 - 3
	Status	ON
Channel 64-71	Frequency (Hz)	$903000000 + 1600000 * \text{channel Index}$
	Data rate range (min - max)	4 - 4
	Status	ON

Figure: Channel parameters table for LoRaWAN 900

Below you can see how the parameters can be configured.

• Channel frequency

The first three channels have a fixed frequency value. The rest of them can be configured in the following ranges: from 863250000 to 869750000 Hz for the 868 MHz band, and from 433050000 to 434790000 Hz for the 433 MHz band. Though frequency is not a settable parameter in LoRaWAN 900 module frequency can be queried with ranges from 902300000 to 914900000 Hz.

The `setChannelFreq()` function allows the user to set the operational frequency on the given channel number (from 3 to 15). The default channels (0-2) cannot be modified in terms of frequency. This function is not available for LoRaWAN 900 module because channels have a fixed frequency.

The `getChannelFreq()` function allows the user to query the channel frequency which was previously set by the user. This function can query channels from 0 to 15 when using LoRaWAN 868 module and channels from 0 to 71 when using LoRaWAN 900 module. The attribute `_freq` permits to access to the settings of the module.

Example of use:

```
{  
    LoRaWAN.setChannelFreq(3, 868000000);  
    LoRaWAN.getChannelFreq(3);  
}
```

Related variable:

`LoRaWAN._freq[n]` → Stores the previously set frequency for channel 'n'

Example of channel settings configuration:

www.libelium.com/development/waspmote/examples/lorawan-07-channels-configuration

• Channel duty cycle

The `setChannelDutyCycle()` function allows the user to set the operational duty cycle on the given channel number (from 0 to 15). The duty cycle value that needs to be used as input argument can be obtained from the wanted duty cycle X (in percentage) using the following formula: $\text{duty cycle} = (100/X) - 1$. The default settings consider only the three default channels (0-2), and their default duty cycle is 0.33%. If a new channel is created either by the server or by the user, all the channels (including the default ones) must be updated by the user in terms of duty cycle to comply with the applicable regulations in the country. This function is not available for LoRaWAN 900 module.

The `getChannelDutyCycle()` function allows the user to query the channel duty cycle which was previously set by the user. The attribute `_dCycle` permits to access to the settings of the module. The attribute range goes from 0 to 65535. The `_dCycle` value that needs to be configured can be obtained from the actual duty cycle X (in percentage) using the following formula: $X = 100/(_dCycle + 1)$. This function is not available for LoRaWAN 900 module.

Example of use:

```
{  
    LoRaWAN.setChannelDutyCycle(3, 9);  
    LoRaWAN.getChannelDutyCycle(3);  
}
```

Related variable:

`LoRaWAN._dCycle[n]` → Stores the previously set duty cycle for channel 'n'

Example of channel settings configuration:

www.libelium.com/development/waspmote/examples/lorawan-07-channels-configuration

• Channel data rate range (DRR)

The `setChannelDRRRange()` function allows the user to set the operational data rate range, from minimum to maximum values, for the given channel number.

The LoRaWAN 868 module supports data rate ranges from 0 to 7 on channels 0 to 15.

The LoRaWAN 900 module supports data rate ranges from 0 to 4 on channels 0 to 63. Channels from 64 to 71 have a fixed data rate range.

The `getChannelDRRRange()` function allows the user to query the data rate range which was previously set by the user. The attributes to store the maximum and minimum data rates are `_drrMax` and `_drrMin` respectively.

Example of use:

```
{  
    LoRaWAN.setChannelDRRRange(3, 0, 6);  
    LoRaWAN.getChannelDRRRange(3);  
}
```

Related variable:

`LoRaWAN._drrMax[n]` → Stores the previously set maximum data rate for channel 'n'

`LoRaWAN._drrMin[n]` → Stores the previously set minimum data rate for channel 'n'

Example of channel settings configuration:

www.libelium.com/development/waspmote/examples/lorawan-07-channels-configuration

- **Channel status**

The `setChannelStatus()` function allows the user to set the operation of the given channel, either "on" or "off".

LoRaWAN 868 allows to configure the channel status on channels from 0 to 15.

LoRaWAN 900 allows to configure the channel status on channels from 0 to 71.

The `getChannelStatus()` function allows the user to query the operation channel status. The attribute `_status` permits to access to the settings of the module.

Example of use:

```
{
  LoRaWAN.setChannelStatus(3, "on");
  LoRaWAN.setChannelStatus(3, "off");

  LoRaWAN.getChannelStatus(3);
}
```

Related variable:

`LoRaWAN._status[n]` → Stores the previously set status for channel 'n'

Example of channel settings configuration:

www.libelium.com/development/waspmote/examples/lorawan-07-channels-configuration

3.3.19. Duty cycle prescaler

The `getDutyCyclePrescaler()` function allows the user to query the duty cycle prescaler. The value of the prescaler can be configured only by the server through use of the Duty Cycle Request frame. Upon reception of this command from the server, the duty cycle prescaler is changed for all enabled channels. The attribute `_dCyclePS` permits to access to the settings of the module.

Example of use:

```
{
  LoRaWAN.getDutyCyclePrescaler();
}
```

Related variable:

`LoRaWAN._dCyclePS` → Stores the duty cycle prescaler established by the server

3.3.20. Margin

The `getMargin()` function allows the user to query the demodulation margin as received in the last Link Check Answer frame. The attribute `_margin` permits to access to the settings of the module.

Example of use:

```
{  
    LoRaWAN.getMargin();  
}
```

Related variable:

`LoRaWAN._margin` → Stores the margin received in the last Link Check Answer frame

3.3.21. Gateway number

The `getGatewayNumber()` function allows the user to query the number of gateways that successfully received the last Link Check Request frame command, as received in the last Link Check Answer. The attribute `_gwNumber` permits to access to the settings of the module.

Example of use:

```
{  
    LoRaWAN.getGatewayNumber();  
}
```

Related variable:

`LoRaWAN._gwNumber` → Stores the number of gateways that received the last Link Check Request frame

3.4. P2P mode – Direct communication between nodes

3.4.1. Enable P2P mode

The `macPause()` function allows the user to disable LoRaWAN mode and enable P2P mode. After power reboot, the module's default mode is LoRaWAN. So, it is mandatory to call this function in order to work with the P2P mode. After calling this function, all P2P functions explained in this section will be able to be run.

Example of use:

```
{  
    LoRaWAN.macPause();  
}
```

3.4.2. Send data

The `sendRadio()` function allows the user to transmit data using the radio transceiver. This function will not expect any acknowledgement back from the receiver. The maximum length of the frame is 255 bytes (510 ASCII digits).

Example of use:

```
{
  char data[] = "010203040506070809";

  LoRaWAN.macPause();
  LoRaWAN.sendRadio(data);
}
```

3.4.3. Receive data

The `receiveRadio()` function allows the user to receive data using the radio transceiver. This function needs a timeout parameter to keep the module listening for any data. The range for this timeout input parameter is from 0 ms to 4294967295 ms. If any data frame is received, it is stored in `_buffer`. The length of the buffer is specified in `_length`. The user must keep in mind that this buffer structure is used for all functions in the API. So, the packet contents should be stored in a program buffer for being used after reception.

Example of use:

```
{
  uint32_t time = 10000;

  LoRaWAN.macPause();
  LoRaWAN.receiveRadio(time);
}
```

Related variable:

`LoRaWAN._buffer` → Stores data received through radio transceiver

`LoRaWAN._length` → Stores length of the data stored in `_buffer`

3.4.4. Power level

The `setRadioPower()` function allows the user to set the operating output power in P2P mode.

The `getRadioPower()` function allows the user to query the operating output power level in P2P mode which was previously set by the user. The attribute `_radioPower` permits to access to the settings of the module.

The range of this attribute goes from -3 to 15 for the LoRaWAN 868 module.

The range of this attribute goes from 2 to 20 for the LoRaWAN 900 module.

The output power level in dBm can be consulted in the radio transceiver module datasheet, RN2483 to see about LoRaWAN 868 and RN2903 to see about LoRaWAN 900.

Example of use:

```
{
  LoRaWAN.setRadioPower(3);
  LoRaWAN.getRadioPower();
}
```

Related variable:

`LoRaWAN._radioPower` → Stores the previously set output power level

3.4.5. Spreading Factor

The `setRadioSF()` function allows the user to set the operating spreading factor (SF) in P2P mode.

The `getRadioSF()` function allows the user to query the operating Spreading Factor (SF) in P2P mode which was previously set by the user. The attribute `_radioSF` permits to access to the settings of the module. The spreading factor can take the following values: "sf7", "sf8", "sf9", "sf10", "sf11" and "sf12"

Example of use:

```
{  
  LoRaWAN.setRadioSF("sf7");  
  LoRaWAN.getRadioSF();  
}
```

Related variable:

`LoRaWAN._radioSF` → Stores the previously set Spreading Factor

3.4.6. Frequency deviation

The `setRadioFreqDeviation()` function allows the user to set the frequency deviation during operation in P2P mode.

The `getRadioFreqDeviation()` function allows the user to query the operating frequency deviation which was previously set by the user. The attribute `_radioFreqDev` permits to access to the settings of the module. The frequency deviation range goes from 0 to 200000.

Example of use:

```
{  
  LoRaWAN.setRadioFreqDeviation(5000);  
  LoRaWAN.getRadioFreqDeviation();  
}
```

Related variable:

`LoRaWAN._radioFreqDev` → Stores the previously set frequency deviation

3.4.7. Preamble length

The `setRadioPreamble()` function allows the user to set the preamble length for transmit/receive in P2P mode.

The `getRadioPreamble()` function allows the user to query the preamble length which was previously set by the user. The attribute `_preambleLength` permits to access to the settings of the module. The preamble length range goes from 0 to 65535.

Example of use:

```
{  
  LoRaWAN.setRadioPreamble(8);  
  LoRaWAN.getRadioPreamble();  
}
```

Related variable:

`LoRaWAN._preambleLength` → Stores the previously set preamble length

3.4.8. CRC header

The `setRadioCRC()` function allows the user to set the Cyclic Redundancy Check (CRC) header status for transmit/receive in P2P mode.

The `getRadioCRC()` function allows the user to query the CRC status which was previously set by the user. The attribute `_crcStatus` permits to access to the settings of the module. This attribute is set to "on" if CRC is enabled or "off" if CRC is disabled.

Example of use:

```
{
  LoRaWAN.setRadioCRC("on");
  LoRaWAN.setRadioCRC("off");

  LoRaWAN.getRadioCRC();
}
```

Related variable:

`LoRaWAN._crcStatus` → Stores the previously set CRC status

3.4.9. Coding Rate

The `setRadioCR()` function allows the user to set the Coding Rate (CR) for communications in P2P mode.

The `getRadioCR()` function allows the user to query the CR which was previously set by the user. The attribute `_radioCR` permits to access to the settings of the module. The CR can take the following values: "4/5", "4/6", "4/7" and "4/8".

Example of use:

```
{
  LoRaWAN.setRadioCR("4/5");

  LoRaWAN.getRadioCR();
}
```

Related variable:

`LoRaWAN._radioCR` → Stores the previously set coding rate

3.4.10. Bandwidth

The `setRadioBandwidth()` function allows the user to set the operating radio bandwidth (BW) for LoRa operation.

The `getRadioBandwidth()` function allows the user to query radio bandwidth which was previously set by the user. The attribute `_radioBW` permits to access to the settings of the module. The radio bandwidth can take the following values: 125 kHz, 250 kHz and 500 kHz.

Example of use:

```
{
  LoRaWAN.setRadioBandwidth(250);

  LoRaWAN.getRadioBandwidth();
}
```

Related variable:

`LoRaWAN._radioBw` → Stores the previously set radio bandwidth

3.4.11. Frequency

The `setRadioFrequency()` function allows the user to set the communication frequency of the radio transceiver.

The `getRadioFrequency()` function allows the user to query radio frequency which was previously set by the user. The attribute `_radioFreq` permits to access to the settings of the module.

When using the LoRaWAN 868 module, the operation frequency can take values from 433250000 to 434550000 or from 863250000 to 869750000, for the 433 and 868 MHz bands.

When using the LoRaWAN 900 module, the operation frequency can take values from 902000000 to 928000000.

Example of use:

```
{
  LoRaWAN.setRadioFrequency(868100000);

  LoRaWAN.getRadioFrequency();
}
```

Related variable:

`LoRaWAN._radioFreq` → Stores the previously set communication frequency

3.4.12. Signal to noise ratio (SNR)

The `getRadioSNR()` function allows the user to query the Signal to Noise Ratio (SNR) for the last received packet. The attribute `_radioSNR` permits to access to the settings of the module. The SNR can take values from -127 to 128.

Example of use:

```
{
  LoRaWAN.getRadioSNR();
}
```

Related variable:

`LoRaWAN._radioSNR` → Stores the previously set communication frequency

3.5. Hybrid LoRaWAN / P2P mode

It is possible to set up hybrid networks using both Radio and LoRaWAN protocols. Therefore, several nodes can use a P2P star topology to reach a central node which will access to the LoRaWAN network to route the information. The basis of this operation is that the central node listens to P2P packets and sends them to the LoRaWAN infrastructure. See the following diagram to understand this hybrid network:

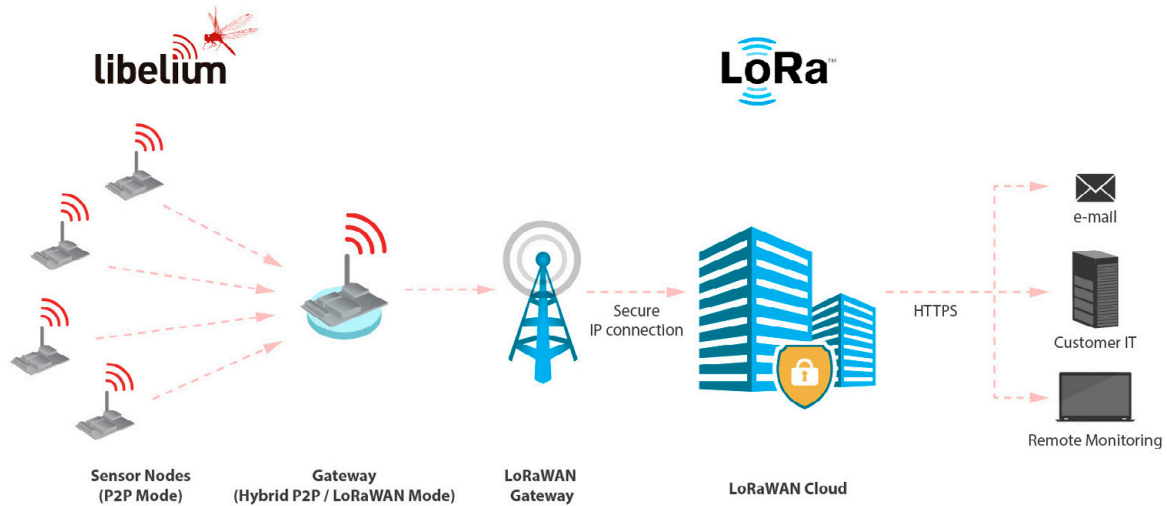


Figure: Hybrid LoRaWAN / P2P mode

The user must keep in mind that there is a mismatch between the maximum payload in P2P networks (255 bytes) and LoRaWAN networks (292 bytes). Therefore, the central node will be able to resend all received frames from the other P2P nodes. The following example shows how to operate as a central node sending data of the incoming P2P packets to the LoRaWAN network:

www.libelium.com/development/waspmote/examples/lorawan-p2p-04-hybrid-p2p-to-lorawan

4. LoRaWAN back-ends

LoRaWAN network architecture is typically laid out in a star-of-stars topology in which a **gateway** is a transparent bridge relaying messages between **end-devices** and a central **network server** in the back-end.

4.1. Wasp mote recommended configuration

Before using a LoRaWAN module with the back-ends explained below or any other back-end, it is strongly recommended to know the configuration that the LoRaWAN gateway is using so we can configure Wasp mote the same way.

We will need to pay attention to some of these parameters:

- Number of channels supported
- Receiving windows configurations
- Gateway's firmware version

Depending on the LoRaWAN module version we work with, it will be necessary to configure different parameters.

Make sure that the gateway's firmware is always up to date. If there is no certain about it, contact the gateway provider or the back-end provider.

4.1.1. LoRaWAN 868

As we can see in the section "Channel parameters", the LoRaWAN 868 module supports up to 16 channels to transmit information. According to the *LoRa Alliance: LoRaWAN Specification* document, it has 3 channels configured by default with fixed frequencies.

Depending on the gateway manufacturer, the number of supported channels may vary. To get the best performance it is recommended to configure the module to use as many channels as the gateway may support.

There is a set of functions to configure these channels parameters, and to turn on channels so the module can use them. Before turning on channels we must configure them, otherwise the module will not allow the user to activate them.

So set frequencies for every channel according to the gateway's configuration. Set the data rate (max range depends on that) as desired. Keep in mind that the duty cycle must be set and modified for the existence of other channels so it complies with the applicable regulations in the country as shown in the "Channel Duty Cycle" section.

Once the configuration has been correctly done, channels can be activated.

Window reception parameters may also vary from a gateway to another. They should be configured according to the parameters given by the back-end provider.

You can see an example of configuration in this code:

www.libelium.com/development/waspote/examples/lorawan-01a-configure-lorawan-868

4.1.2. LoRaWAN 900

As we can see in the section "Channel parameters", LoRaWAN 900 module supports up to 64 channels to transmit uplink messages. By default, the whole set of channels is activated when the module starts working.

Depending on the gateway manufacturer, the number of supported channels may vary. To get the best performance it is recommended to configure the module to use as many channels as gateway may support. It is strongly recommended to deactivate channels that are not supported by the gateway so the module does not try to send information on these channels.

Before setting on any channel, it is necessary to configure the data rate (max range depends on that) as desired.

Window reception parameters may also vary from a gateway to another. They should be configured according to the parameters given by the back-end provider.

You can see an example of configuration in this code:

www.libelium.com/development/waspote/examples/lorawan-01b-configure-lorawan-900

4.2. Actility

The [ThingPark Wireless Device Manager](#) is the back-end User Interface (UI) which allows you to manage all of your LoRaWAN devices.

This Guide will provide the guidelines on the entire GUI, the device provisioning, device configuration and management, alarm and routing profile management and connectivity plan association.

The Device Manager can be fully integrated into a third party customer UI, through all the ThingPark Wireless OSS REST API.

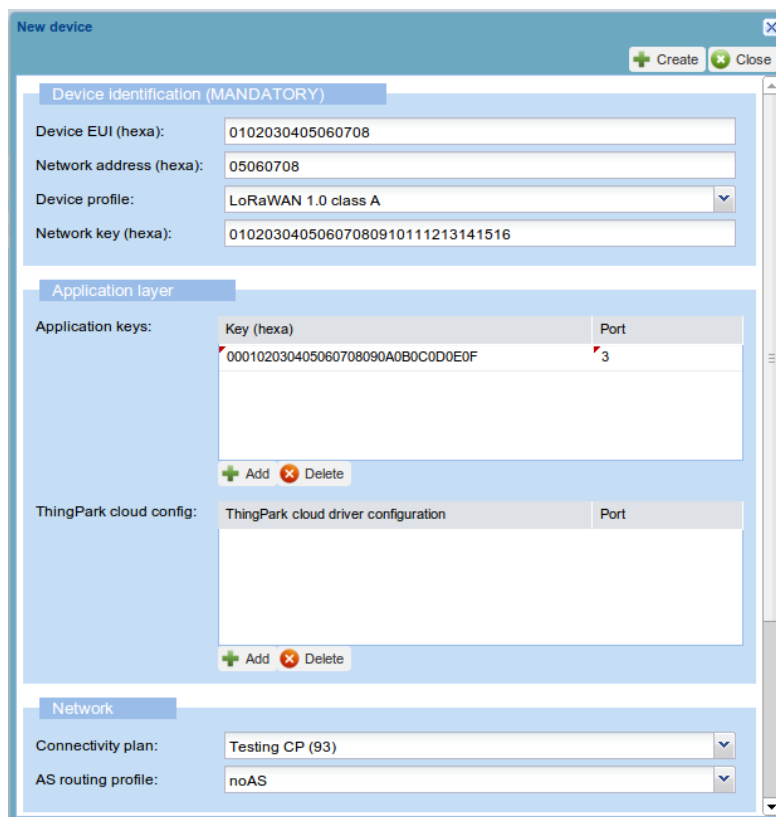
4.2.1. Device registration

The device provisioning is the process that allows users to create devices and register them on the network through Activation By Personalization (ABP).

Information required to create a new device:

- Device EUI (**DevEUI**)
- Device Address (**DevAddr**)
- Network Session Key (**NwkSKey**)
- Application Session Key (**AppSKey**)

These parameters match with the ones which must be configured in the ThingPark Actility Portal (back-end).



New device [Create] [Close]

Device Identification (MANDATORY)

Device EUI (hexa): 0102030405060708

Network address (hexa): 05060708

Device profile: LoRaWAN 1.0 class A

Network key (hexa): 01020304050607080910111213141516

Application layer

Application keys:

Key (hexa)	Port
000102030405060708090A0B0C0D0E0F	3

[Add] [Delete]

ThingPark cloud config:

ThingPark cloud driver configuration	Port
--------------------------------------	------

[Add] [Delete]

Network

Connectivity plan: Testing CP (93)

AS routing profile: noAS

Figure: ThingPark Wireless Device Manager (new device creation)

4.2.2. Wasp mote programming

Activity Portal lets the user configure all parameters needed to connect into a network. **NwkSKey** and **AppSKey** are 128-bit keys specific for the end-device, used to calculate and verify an application-level MIC (message integrity code) and also used by both network server and end-device to encrypt and decrypt the payload field of application-specific data messages.

Both **NwkSKey** and **AppSKey** will be configured by the user in the Activity Portal and set into the end-device. These keys are not shown in the ThingPark Wireless Device Manager once they have been configured so it is strongly recommended to set them into the end-device before configuring the end-device creation in the Activity Portal.

About **DevEUI** and **DevAddr**, there are two ways to configure it into the module with Waspote.

- On one hand they may be set by user manually. When using this method always keep in mind they must be unique for every module.
- On the other and they can be configured automatically by Waspote. This method uses the preprogrammed by manufacturer module's identifier, that matches **DevEUI** length, ensuring it will be unique. **DevAddr** will be extracted from this manufacturer module's identifier taking its last 32 bit. E.g., Microchip Module: EUI: 0004A30B001A836D, **DevEUI**: 0004A30B001A836D, **DevAddr**: 001A836D.

Examples of setting configuration necessary to connect into a network and send packets:

www.libelium.com/development/waspote/examples/lorawan-02-send-unconfirmed

www.libelium.com/development/waspote/examples/lorawan-03-send-confirmed

4.3. OrbiWise

OrbiWise introduces its comprehensive UbiQ core network solution for Low-Power Wide-Area (LPWA) networks taking full advantage of LoRaTM bidirectional communication technology.

LPWA networks are of special interest to wireless operators, utilities and industrial companies that want to serve many applications in the Internet-of-Things domain with a single low-cost infrastructure.

OrbiWise's UbiQ solution enables the deployment of such infrastructure from city-scale to nation-wide coverage with seamless expansion and support for millions of connected objects. Furthermore, UbiQ's O&M tool suite allows operations and maintenance of the entire network from a single web-based graphical user interface.

4.3.1. Device registration

The Add New Device process allows users to create devices and register them on the network through Activation By Personalization (ABP).

Information required to create a new device:

- Device EUI (**DevEUI**)
- Device Address (**DevAddr**)
- Network Session Key (**NwkSKey**)

Optional information:

- Application EUI (**AppEUI**)
- Comment (Cloud API device's identifier)
- Application Session Key (**AppSKey**)

- Always have in mind to select the Personalized Registration Type. Despite **AppSKey** is an optional parameter to create the device in the server, it is mandatory to be configured in the module. In any case, it is always better to set an **AppSKey** both in server and module (same for both) so data can be decrypted by server.

Add New Device

DevEUI

The DevEUI is a 8-byte unique identified based on IEEE EUI-64. Mandatory.

AppEUI

The AppEUI identify the associated application. Optional.

Comment

The device comment is for convinience only. Optional.

Keys

QoS

Packet Storage

LoRa Parameters

Miscellaneous

Registration type

Join Procedure

Personalized

Personalised devices have pre-generated session keys and will not perform the JOIN procedure.

DevAddr

The DevAddr (device address) is a 4 byte value. Mandatory.

NwkSKey

The NwkSKey is a 16-byte encryption key used to encrypt the LoRaWAN protocol frames. Mandatory.

AppSKey

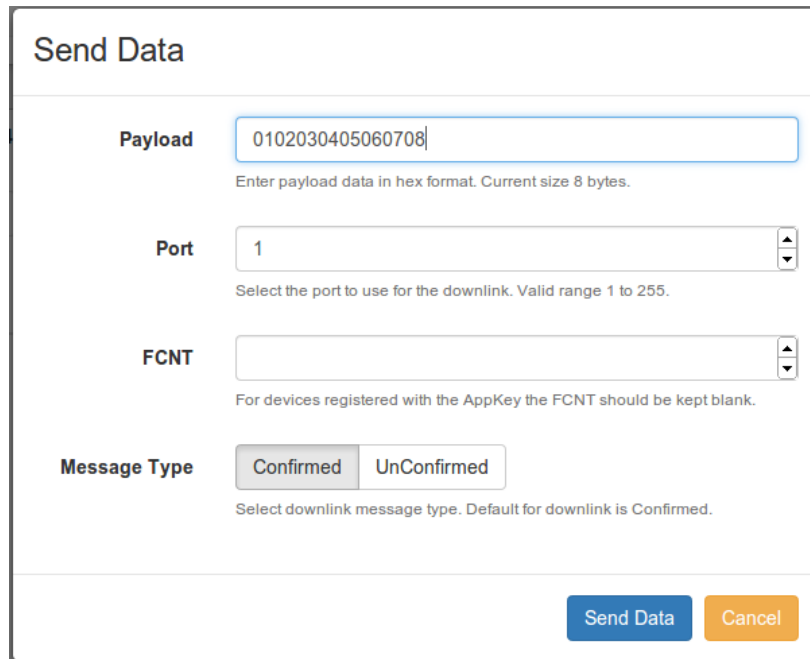
The AppSKey is a 16-byte encryption key used to encrypt the data payloads. If provided all encryption is managed by the network. If not provided, the payload encryption must be managed by the application. Optional.

Add Device

Cancel

4.3.2. Data downlink

To perform data downlinks, a button called “Send Data” can be found in Data section. Clicking on it will open a pop-up that the user can fill with data for the payload and select the message type. Data will be received by the module after the next data uplink. It does not matter if the next uplink process is confirmed or unconfirmed, the downlink will be performed.



The "Send Data" dialog box contains the following fields and controls:

- Payload:** A text input field containing the hex string "0102030405060708". Below it, a note states: "Enter payload data in hex format. Current size 8 bytes."
- Port:** A numeric input field with the value "1" and up/down arrow controls. Below it, a note states: "Select the port to use for the downlink. Valid range 1 to 255."
- FCNT:** A numeric input field that is currently empty and has up/down arrow controls. Below it, a note states: "For devices registered with the AppKey the FCNT should be kept blank."
- Message Type:** Two radio buttons labeled "Confirmed" and "UnConfirmed". The "Confirmed" button is selected. Below it, a note states: "Select downlink message type. Default for downlink is Confirmed."
- Buttons:** At the bottom right, there are two buttons: "Send Data" (blue) and "Cancel" (orange).

Figure: OrbiWise Wireless Device Manager (sending data downlink)

4.3.3. Wasp mote programming

Once the new device was created in the OrbiWise portal only **DevEUI** will be shown, so user should set the device address, network session key and application session key in both portal and module at the same time.

Examples of setting configuration necessary to connect into a network and send packets:

www.libelium.com/development/waspote/examples/lorawan-02-send-unconfirmed

www.libelium.com/development/waspote/examples/lorawan-03-send-confirmed

4.4. LORIIOT

LORIIOT.io is a provider of a LoRaWAN Network Server and Application Server software, which is commercially offered through a set of business models.

They provide:

- Software for the supported LoRa gateways
- Cloud-based LoRaWAN Network Server
- Programming interface (APIs) for Internet of Things applications to access the end node data
- Output of end node data to number of 3rd party services

As a gateway owner, users can use LORIIOT.io software on gateways to connect them to their cloud. From then on, all data received by the gateways will be relayed to the user through the LORIIOT.io APIs or 3rd party services.

The network server components fulfills to role of protocol processor. It is a TLS connection end-point for the gateways and the customer applications. It is responsible for processing the incoming end node data according to the LoRaWAN protocol.

The specific roles of LORIIOT.io Network Server are:

- Gateway population management
- Application population management
- Device population management
- Collection of billing records
- Security management
- Data distribution

4.4.1. Device registration

Some parameters must be set into the module so it can join to a network through Activation By Personalization (ABP).

- Device EUI (**DevEUI**)
- Device Address (**DevAddr**)
- Network Session Key (**NwkSKey**)
- Application Session Key (**AppSKey**)

When a new device is generated in the LORIIOT.io portal these parameters are provided. The user will have to configure the module according to these parameters.

Add New Device

DevEUI

The DevEUI is a 8-byte unique identified based on IEEE EUI-64. Mandatory.

AppEUI

The AppEUI identify the associated application. Optional.

Comment

The device comment is for convinience only. Optional.

Keys

QoS

Packet Storage

LoRa Parameters

Miscellaneous

Registration type

Join Procedure

Personalized

Personalised devices have pre-generated session keys and will not perform the JOIN procedure.

DevAddr

The DevAddr (device address) is a 4 byte value. Mandatory.

NwkSKey

The NwkSKey is a 16-byte encryption key used to encrypt the LoRaWAN protocol frames. Mandatory.

AppSKey

The AppSKey is a 16-byte encryption key used to encrypt the data payloads. If provided all encryption is managed by the network. If not provided, the payload encryption must be managed by the application. Optional.

Add Device

Cancel

Figure: LORIIOT.io Wireless Device Manager (new device creation)

Once the device is created you can access to its configuration and data clicking over the Device EUI field with the mouse. Data messages can be seen there.

4.4.2. Data downlink

Data sent to the back-end can be found inside every device page. Data could also be found in the WebSocket API section.

LORIIOT.io									
Connected to BE7A002A									
Disconnect Decode data Send data									
Device EUI	Timestamp	Freq	SF	RSSI	SNR	Seq #	Port	Payload	
BE7A000000000798	10:22:01	868.500	SF10 BW125 4/5	-40	6.2	31 [H]	3	0123123231	
BE7A000000000798	10:14:21	868.500	SF12 BW125 4/5	-51	7	30 [H]	3	0123123231	
BE7A00000000072C	18:24:59	868.300	SF8 BW125 4/5	-48	8.5	17 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:24:51	868.500	SF8 BW125 4/5	-45	8	16 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:24:42	868.300	SF8 BW125 4/5	-47	12	15 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:24:33	868.300	SF7 BW125 4/5	-43	10.2	14 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:24:21	868.500	SF7 BW125 4/5	-43	6.5	13 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:24:13	868.300	SF7 BW125 4/5	-45	11	12 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:24:05	868.300	SF7 BW125 4/5	-43	9.5	11 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:23:56	868.300	SF7 BW125 4/5	-43	8.8	10 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:23:48	868.100	SF7 BW125 4/5	-45	11	9 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:23:40	868.500	SF7 BW125 4/5	-45	7.5	8 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:23:31	868.300	SF7 BW125 4/5	-43	10.8	7 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:23:22	868.100	SF7 BW125 4/5	-46	7.2	6 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:23:14	868.500	SF7 BW125 4/5	-47	7.8	5 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:23:05	868.300	SF7 BW125 4/5	-47	9.2	4 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:22:57	868.100	SF7 BW125 4/5	-42	10	3 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:22:49	868.100	SF7 BW125 4/5	-41	8.5	2 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:22:41	868.300	SF7 BW125 4/5	-39	10	1 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:22:33	868.100	SF7 BW125 4/5	-39	8.5	0 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:22:22	868.100	SF8 BW125 4/5	-43	8.2	22 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:22:13	868.300	SF8 BW125 4/5	-43	9.5	21 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:22:05	868.300	SF8 BW125 4/5	-45	10.2	20 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:21:57	868.500	SF8 BW125 4/5	-45	8.5	19 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:21:49	868.100	SF8 BW125 4/5	-45	11.8	18 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:21:41	868.100	SF8 BW125 4/5	-45	10.5	17 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:21:32	868.300	SF8 BW125 4/5	-45	10.5	16 [H]	14	0102030405060708090a0b0c0d0e0f	
BE7A00000000072C	18:21:24	868.500	SF8 BW125 4/5	-43	8.5	15 [H]	14	0102030405060708090a0b0c0d0e0f	

Figure: LORIIOT WebSocket API

To perform data downlinks click on the “Send data” button and the “Send to device” section will be displayed. In this section, Device EUI, port number and data payload to send must be filled. Finally, the “Send to device” button will enqueue the data to be downloaded. Data will be received by the module after the next data uplink. It does not matter if the next uplink process is confirmed or unconfirmed, the downlink will be performed.

BE7A00000000072C

21

0102030405060708|

Send to device

Figure: LORIIOT WebSocket API (sending data downlink)

4.4.3. Wasp mote programming

LORIIOT.io portal provides by default the whole configuration required by the module to connect to a network.

The mandatory field to field are the ones which where mentioned in the previous section. They can be copied as is to the code so module is configured the same way it was created in the portal.

Inside the device description in the LORIIOT.io portal they warn to copy **EUI** and **Address** in little endian format, but it won't be necessary for the Wasp mote code, it can be copied big endian format.

Examples of setting configuration necessary to connect into a network and send packets:

www.libelium.com/development/waspote/examples/lorawan-02-send-unconfirmed

www.libelium.com/development/waspote/examples/lorawan-03-send-confirmed

5. When is LoRaWAN recommended?

LoRaWAN is a protocol with a good long-range performance. It is achieved thanks to the excellent receiver sensitivity of the LoRa modulation, which is possible due to very low data rates (few bps). The main drawback of LoRaWAN is the low data rate of the transmission modes with high Spreading Factor. The better range, the worse bitrate. This may be a problem in crowded networks, because the shared channel could be too busy because it can take several seconds to send each frame. The base station and the back-end server will try to find an efficient balance in the network thanks to the LoRaWAN algorithms. So:

- LoRaWAN is not advised for projects with a duty-cycle which require sending one frame every few minutes.
- The downlink data rate is very low too, so OTA programming is not possible.
- LoRaWAN is NOT recommended for real time streaming. Transmission is not done in real time as there is a minimum delay for packet arrival.
- LoRaWAN is recommended for long-range device communications in cities, where base stations by some LoRaWAN operator are deployed, so the user can take advantage of this infrastructure.
- Otherwise, the user can purchase his own LoRaWAN-compliant gateways and deploy a private LoRaWAN network for a specific project.

6. Certifications

Libelium offers two types of sensor platforms: Waspote OEM and Plug & Sense!.

- Waspote 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/waspote/>
- 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. <http://www.libelium.com/products/plug-sense/>

CE

In accordance with the 1999/5/EC (R&TTE) and the EU Directive 2011/65/EU (RoHS2), Libelium Comunicaciones Distribuidas S.L. declares that the "Plug & Sense! LoRaWAN 868" device conforms to the following regulations:

- EN 301 489-1(1.9.2)- (1.6.1)
- EN 55022:2010
- EN 60950-1: 2006
- A11: 2009 + A1: 2010 + A12: 2011 + Ac: 2011 +A2: 2013



If desired, the Declaration of Conformity document can be requested using the Contact section at:

<http://www.libelium.com/contact>

FCC

Plug & Sense! LoRaWAN 900 / 915 is ready for US. It has been certified under the Part 15 Spread Spectrum Transmitter (FCC Rule Part 15 C). If you want to know more contact the [Libelium Sales Department](#).

**IC**

Plug & Sense! LoRaWAN 900 / 915 is ready for Canada. It has been certified under the Industry Canada Regulation (IC) - (RSS-210). If you want to know more contact the [Libelium Sales Department](#).



7. Code examples and extended information

In the Wasp mote Development section you can find complete examples:

www.libelium.com/development/waspmote/examples

Example:

```
/*
 * ----- LoRaWAN Code Example -----
 *
 * Explanation: This example shows how to configure the module
 * and all general settings related to back-end registration
 * process.
 *
 * Copyright (C) 2015 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:   Luismi Marti
 */

#include <WaspLoRaWAN.h>

////////////////////////////////////
uint8_t socket = SOCKET0;
////////////////////////////////////

// Device parameters for Back-End registration
////////////////////////////////////
char DEVICE_EUI[] = "0102030405060708";
char DEVICE_ADDR[] = "05060708";
char NWK_SESSION_KEY[] = "01020304050607080910111213141516";
char APP_SESSION_KEY[] = "000102030405060708090A0B0C0D0E0F";
char APP_KEY[] = "000102030405060708090A0B0C0D0E0F";
////////////////////////////////////

// variable
uint8_t error;

void setup()
{
  USB.ON();
  USB.println(F("LoRaWAN example - Module configuration"));

  ///////////////////////////////////
  // 1. switch on
  ///////////////////////////////////
}
```

```
error = LoRaWAN.ON(socket);

// Check status
if( error == 0 )
{
    USB.println(F("1. Switch ON OK"));
}
else
{
    USB.print(F("1. Switch ON error = "));
    USB.println(error, DEC);
}

////////////////////////////////////
// 2. Reset to factory default values
////////////////////////////////////

error = LoRaWAN.factoryReset();

// Check status
if( error == 0 )
{
    USB.println(F("2. Reset to factory default values OK"));
}
else
{
    USB.print(F("2. Reset to factory error = "));
    USB.println(error, DEC);
}

////////////////////////////////////
// 3. Set/Get Device EUI
////////////////////////////////////

// Set Device EUI
error = LoRaWAN.setDeviceEUI(DEVICE_EUI);

// Check status
if( error == 0 )
{
    USB.println(F("3.1. Set Device EUI OK"));
}
else
{
    USB.print(F("3.1. Set Device EUI error = "));
    USB.println(error, DEC);
}

// Get Device EUI
error = LoRaWAN.getDeviceEUI();

// Check status
if( error == 0 )
{
    USB.print(F("3.2. Get Device EUI OK. "));
    USB.print(F("Device EUI: "));
    USB.println(LoRaWAN._devEUI);
}
else
{
    USB.print(F("3.2. Get Device EUI error = "));
    USB.println(error, DEC);
}
```

```
////////////////////////////////////
// 4. Set/Get Device Address
////////////////////////////////////

// Set Device Address
error = LoRaWAN.setDeviceAddr(DEVICE_ADDR);

// Check status
if( error == 0 )
{
    USB.println(F("4.1. Set Device address OK"));
}
else
{
    USB.print(F("4.1. Set Device address error = "));
    USB.println(error, DEC);
}

// Get Device Address
error = LoRaWAN.getDeviceAddr();

// Check status
if( error == 0 )
{
    USB.print(F("4.2. Get Device address OK. "));
    USB.print(F("Device address: "));
    USB.println(LoRaWAN._devAddr);
}
else
{
    USB.print(F("4.2. Get Device address error = "));
    USB.println(error, DEC);
}

////////////////////////////////////
// 5. Set Network Session Key
////////////////////////////////////

error = LoRaWAN.setNwkSessionKey(NWK_SESSION_KEY);

// Check status
if( error == 0 )
{
    USB.println(F("5. Set Network Session Key OK"));
}
else
{
    USB.print(F("5. Set Network Session Key error = "));
    USB.println(error, DEC);
}

////////////////////////////////////
// 6. Set Application Session Key
////////////////////////////////////

error = LoRaWAN.setAppSessionKey(APP_SESSION_KEY);

// Check status
if( error == 0 )
{
    USB.println(F("6. Set Application Session Key OK"));
}
```

```

}
else
{
    USB.print(F("6. Set Application Session Key error = "));
    USB.println(error, DEC);
}

////////////////////////////////////
// 7. Set retransmissions for uplink confirmed packet
////////////////////////////////////

// set retries
error = LoRaWAN.setRetries(7);

// Check status
if( error == 0 )
{
    USB.println(F("7.1. Set Retransmissions for uplink confirmed packet OK"));
}
else
{
    USB.print(F("7.1. Set Retransmissions for uplink confirmed packet error = "));
    USB.println(error, DEC);
}

// Get retries
error = LoRaWAN.getRetries();

// Check status
if( error == 0 )
{
    USB.print(F("7.2. Get Retransmissions for uplink confirmed packet OK. "));
    USB.print(F("TX retries: "));
    USB.println(LoRaWAN._retries, DEC);
}
else
{
    USB.print(F("7.2. Get Retransmissions for uplink confirmed packet error = "));
    USB.println(error, DEC);
}

////////////////////////////////////
// 8. Set application key
////////////////////////////////////

error = LoRaWAN.setAppKey(APP_KEY);

// Check status
if( error == 0 )
{
    USB.println(F("8. Application key set OK"));
}
else
{
    USB.print(F("8. Application key set error = "));
    USB.println(error, DEC);
}

////////////////////////////////////
// 9. Save configuration
////////////////////////////////////

error = LoRaWAN.saveConfig();

```

```
// Check status
if( error == 0 )
{
    USB.println(F("9. Save configuration OK"));
}
else
{
    USB.print(F("9. Save configuration error = "));
    USB.println(error, DEC);
}

USB.println(F("-----"));
USB.println(F("Now the LoRaWAN module is ready for"));
USB.println(F("joining networks and send messages.));
USB.println(F("Please check the next examples..."));
USB.println(F("-----\n"));

}

void loop()
{
    // do nothing
}
```

8. API changelog

Keep track of the software changes on this link:

[www.libelium.com/development/waspmote/documentation/changelog/#LoRaWAN 23. Documentation changelog](http://www.libelium.com/development/waspmote/documentation/changelog/#LoRaWAN%2023.Documentation%20changelog)

9. Documentation change

From v4.2 to v4.3:

- Updated with the new LoRaWAN 900 module information
- Added examples and recommended configurations for better operation

From v4.1 to v4.2:

- Updates about certification status

From v4.0 to v4.1:

- New API constants, API variables and function responses were added
- Added section for the operational ISM bands
- The data downlink process is explained, from the module point of view and from some back-ends point of view
- More details about the ABP and OTAA processes
- More details about the Application Session Key, the Network Session Key and the Application Key
- Added sections for the downlink counter and the uplink counter