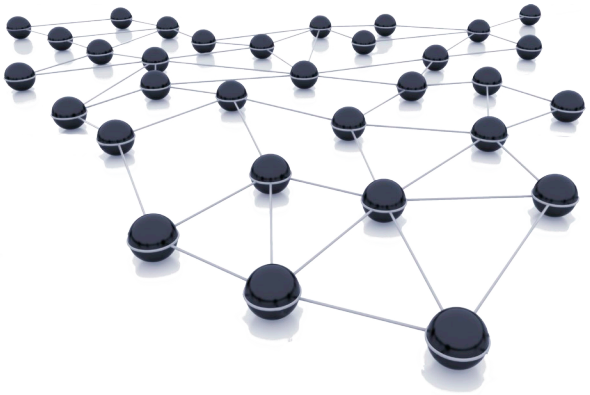
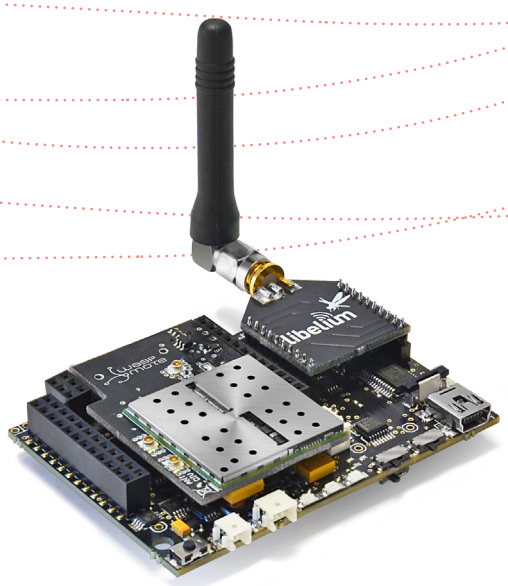
Waspmote Sigfox

Networking Guide





libelium

wasp mote

Document Version: v7.3 - 04/2018

INDEX

1. Introduction

This guide explains the Sigfox features and functions. There are no great variations in this library for 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".

Libelium has added a Sigfox wireless module to its portfolio of Waspmote OEM and Plug & Sense! sensor devices.

The Libelium Sigfox module has been certified as "**Sigfox Ready Class 0**", the designation reserved for maximum range devices on Sigfox Low Power Wide Area (LPWA) networks. This means we can work with our favorite development platform as "Sigfox Ready" when using it in our projects.

Figure: Sigfox module

So now we can use our preferred platform to work on the development of Smart Cities and Internet of Things (IoT) projects that require low-energy, long-range wireless data transmission on the Sigfox global network.

Currently Sigfox operates in most European countries (France, Spain, UK, Germany, Italy, the Netherlands, Belgium, Portugal, Ireland, Sweden, Denmark, Finland, Czech Republic, Slovakia, Croatia, Malta, etc), the Americas (United States, Mexico, Brazil, Argentina, Colombia), Africa (South Africa, Tunisia) and Asia-Pacific (Australia, New Zealand, Japan, Singapore, Taiwan, Hong Kong, Thailand, Iran, Oman). Check the current coverage at Sigfox website.

The Sigfox network is slated for rollout across 60 countries over the next five years, aimed at providing connectivity for low bandwidth IoT applications. Pilot projects are underway in key locations too.

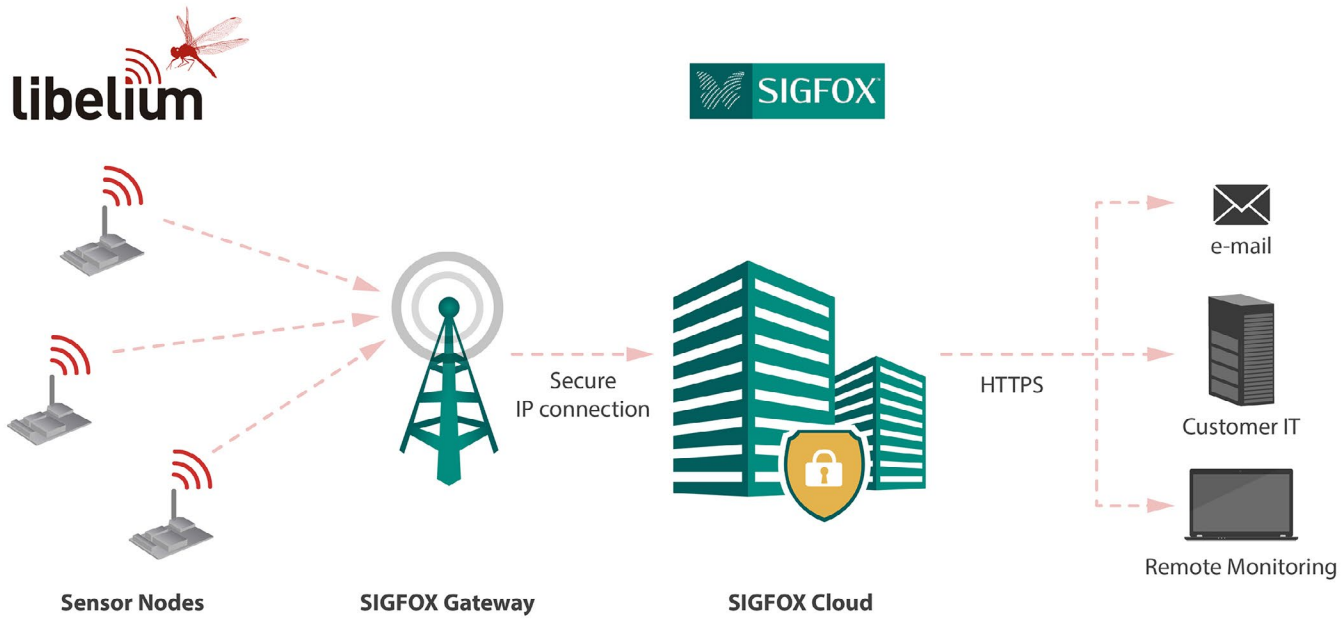


Figure: Sigfox network

Key features	Why it matters
Waspmote Plug & Sense! and Waspmote OEM are certified "Sigfox Ready Class 0."	This level of certification means that all Libelium sensor nodes and OEM products offer the **maximum coverage range** possible on Sigfox Low Power Wide Area (LPWA) networks—up to dozens of kilometers, even in crowded wireless environments. Many Smart Cities and Industrial IoT applications require low bandwidth data transmission over distances, often in areas of high electromagnetic interference.
Embedded radio module in Waspmote sensor nodes send information directly to the Sigfox gateway.	Fast response, and simple, immediate network setup. The network works in a similar way to mobile operator networks – no SIM card is needed.
Two data storage and retrieval options available directly on the network: 1. Trigger actions defined on sensors; 2. Complete API and standard REST calls via HTTP	Because data is stored directly on Sigfox servers, infrastructure costs are reduced for IoT and Smart Cities deployments. Sensor devices can automatically redirect data to a private database or Cloud; Information can be retrieved using standard Web calls and secure API keys.
Secure IP connection assured between devices, Sigfox gateways, and Cloud. Accounts activated with strong, unique API keys.	Secure authentication maintains data integrity, and privacy of the information sent by the sensor nodes and stored on servers, gateways and the Cloud.

As well as the Sigfox mode, the nodes may use two more configurations:

- P2P Mode - Direct Communication between nodes (LAN Interface)
- Hybrid Mode - Sigfox / P2P (P2P + GW to Sigfox Network)

In the **P2P Mode**, nodes may connect directly among them and send messages directly at no cost (as they are not using the Sigfox 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 a Sigfox License** 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 P2P chapter.

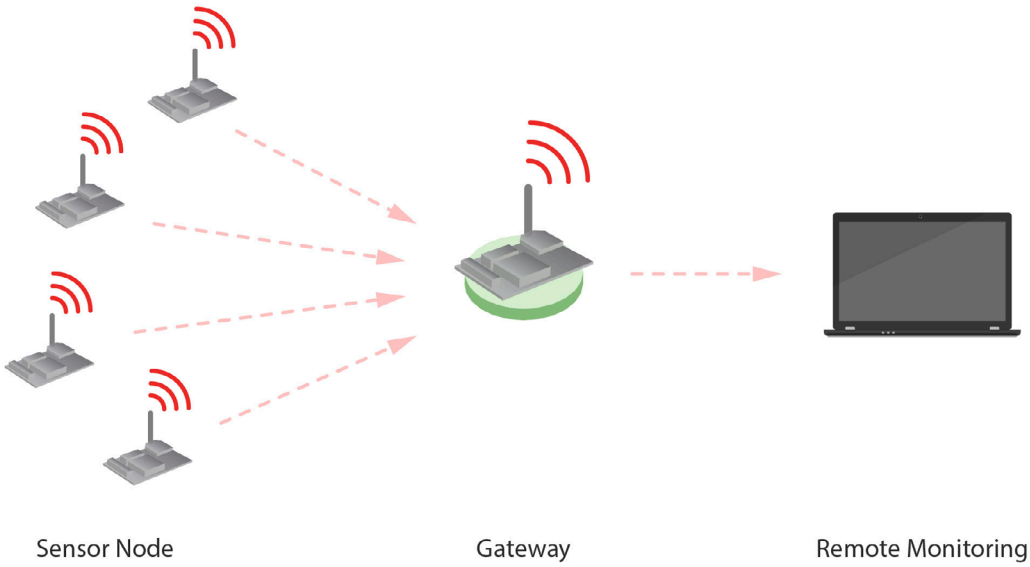


Figure: Sigfox network

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

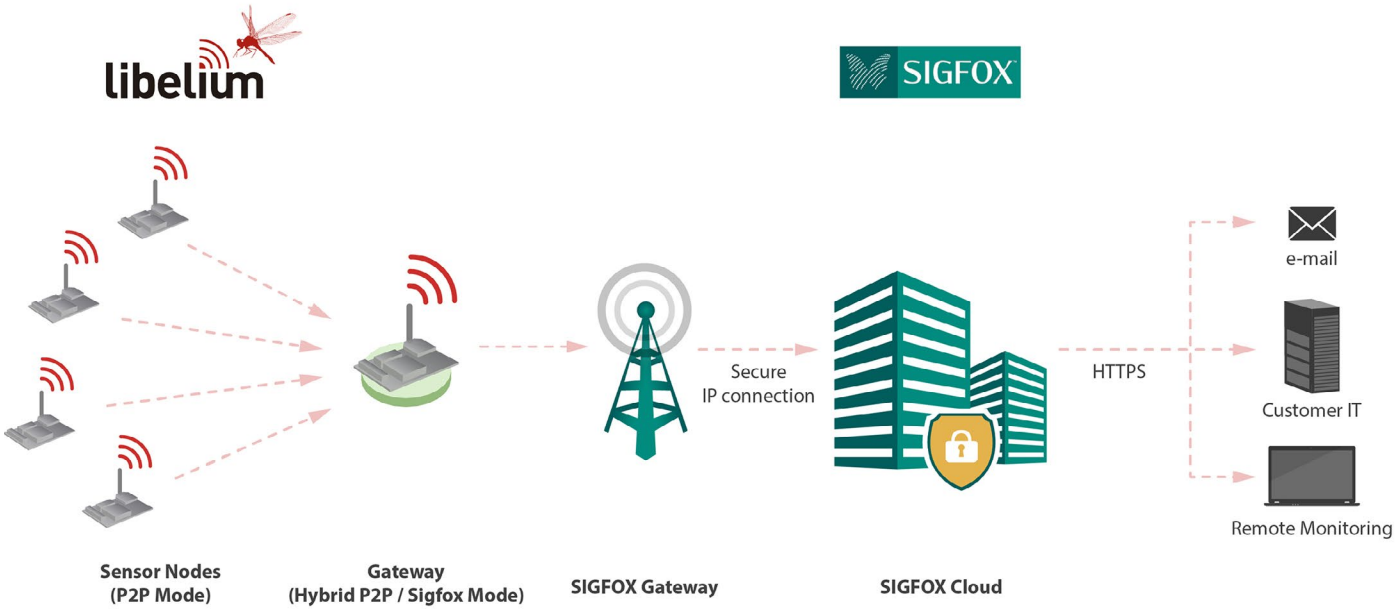


Figure: Hybrid network

FAQ:

Do the Libelium products include a Sigfox license?

No. Sigfox license is NOT included in any of the Libelium items. User must contact Sigfox in order to acquire a license to connect the devices to the Sigfox Network.

Can I buy the Sigfox Ready modules and kits without a Sigfox license?

Yes. You can buy just the hardware kits and acquire a Sigfox license later or use them just in P2P mode without a license.

How many licenses do I need to send to the Sigfox Network?

You need at least one license in order to create a Hybrid Network.

How do I know if I have coverage in my city?

You can check the coverage map here:
http://www.sigfox.com/en/coverage

We do also recommend to contact the Sigfox Network Operator in each country to ensure availability. Complete list of SNO's is included in the link above.

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. Radio technology

Sigfox is a private company that aims to build a worldwide network especially designed for IoT devices. The network is cellular, with thousands of base stations deployed in each country. Sigfox technology offers very long ranges for low-power, battery-constrained nodes. Sigfox is great for very simple and autonomous devices which need to send small amounts of data to this ubiquitous network, taking advantage on the Sigfox infrastructure.

So Sigfox is similar to cellular (GSM-GPRS-3G-4G) but is more energy-efficient, and the annual fees are lower.

Sigfox uses a UNB (Ultra Narrow Band) based radio technology to connect devices to its global network. The use of UNB is key to providing a scalable, high-capacity network, with very low energy consumption, while maintaining a simple and easy to rollout star-based cell infrastructure.

The network operates in the globally available ISM bands (license-free frequency bands) and co-exists in these frequencies with other radio technologies, but without any risk of collisions or capacity problems. Sigfox uses the most popular European ISM band on 868 MHz (as defined by ETSI and CEPT)and local Narrow Band ISM network and the long-distance Ultra Narrow Band in 900 MHz. (American band).

Note 1: *Libelium offers 2 radios, one for the European band (868 MHz) and other one for the US band (900 MHz). See more details on the "Region standards" section.*

Note 2: *Sigfox collaborated with ETSI on the standardization of low throughput networks in Europe. They determined up to 140 messages per object per day. The payload size for each message is 12 bytes. Similar restrictions should be published for USA operation. Sigfox retains the right of cutting off the service and/or applying extra fee if maximum number of packets per day are overpassed.*

1.2. Coverage

Sigfox is being rolled out worldwide. It is the responsibility of the system integrator to consult the catalog of SNOs (Sigfox Network Operators) for checking coverage in the deployment area.

To ensure that your devices are adequately covered, you will need to take the following parameters into account:

- **Geographical coverage.** Please refer to the SNO catalog.
- **Physical surroundings.** As with all radio communications, physical objects (walls, metallic surfaces, etc) may cause the signal to deteriorate.

Although Sigfox uses a more robust technology, the physical laws of radio transmission remain valid, as with any other cellular-based communication technology.

1.3. Sigfox back-end and API

The Sigfox back-end provides a web application interface for device management and configuration of data integration, as well as standards based web APIs to automate the device management and implement the data integration. The APIs are based on HTTPS REST requests, as GET or POST and the payload format is JSON.

However, it is not possible to visualize any kind of graphs. Thus, the user should manage the Sigfox back-end to automatically send notifications or data on Sigfox packets arrival to external devices, clouds or servers.

For further information, refer to the “Sigfox Back-End” chapter.

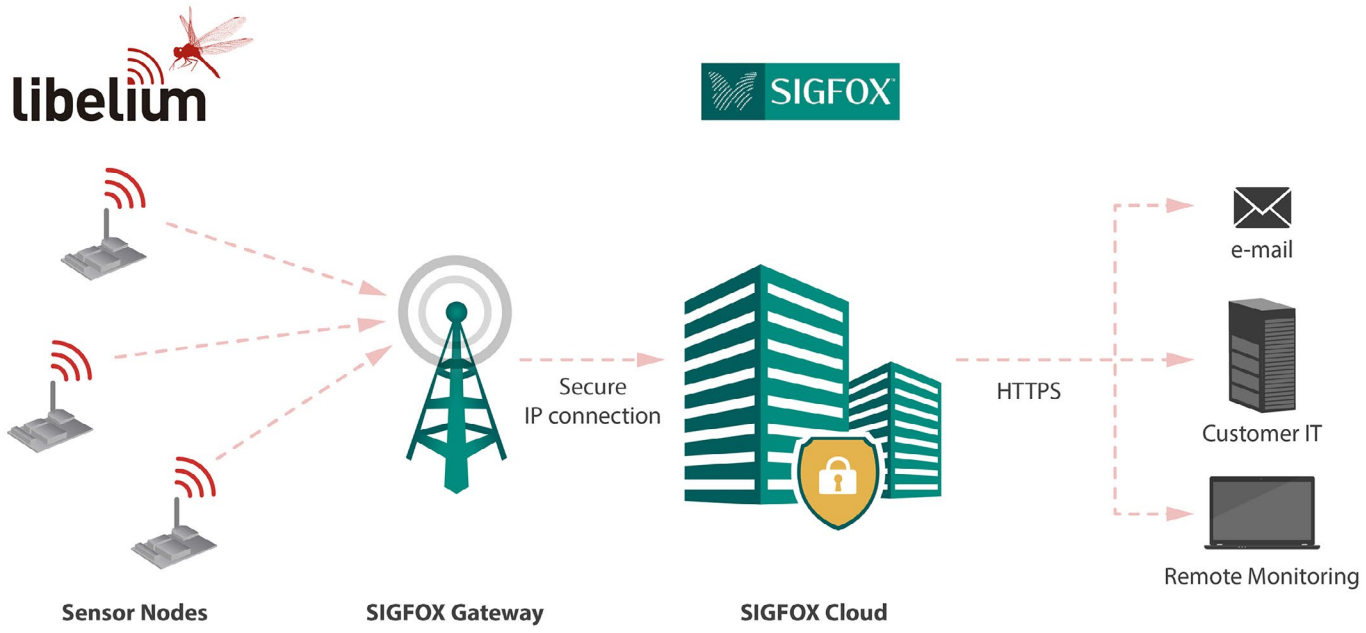


Figure: Sigfox network

2. Hardware

2.1. Specifications

The Sigfox module is managed by UART and it can be connected to SOCKET0 or SOCKET1.

2.1.1. Sigfox EU module

The main features of the module are listed below:

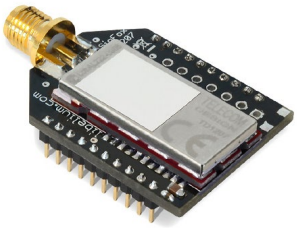


Figure: Sigfox EU

- **Frequency:** ISM 868 MHz
- **TX Power:** 16 dBm
- **Chipset consumption:**
  - TX: 51 mA @ +16 dBm
  - RX: 16 mA
- **Receive sensitivity:** -126 dBm
- **ETSI limitation:** 140 packets of 12 bytes, per module per day
- **Range:** Typically, each base station covers some km. Check the Sigfox Network.
- **Sigfox certified:** Class 0u (the highest level)

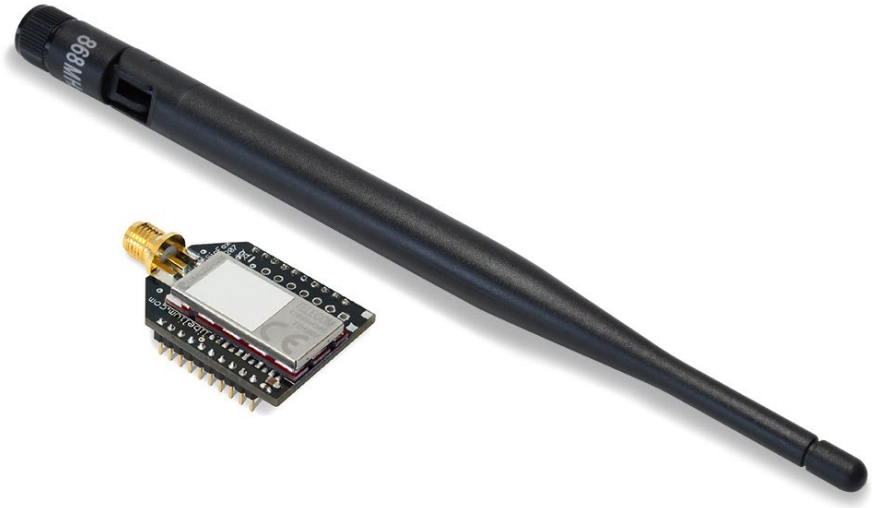


Figure: Sigfox 868 module with 4.5 dBi antenna

2.1.2. Sigfox US module and Sigfox APAC / LATAM module

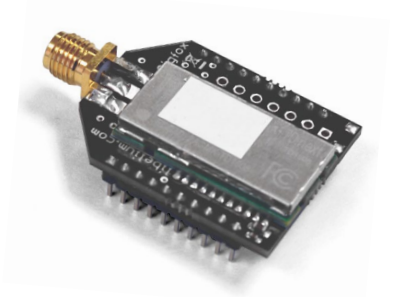


Figure: Sigfox US

The main features of the modules are listed below:

- **Frequency:** ISM 900 MHz
- **TX Power:** 24 dBm
- **Chipset consumption:**
  - TX: 230 mA @ +23 dBm
  - RX: 21 mA
- **Receive sensitivity:** -127 dBm
- **FCC limitation:** Pending
- **Range:** Typically, each base station covers some km. Check the Sigfox Network.

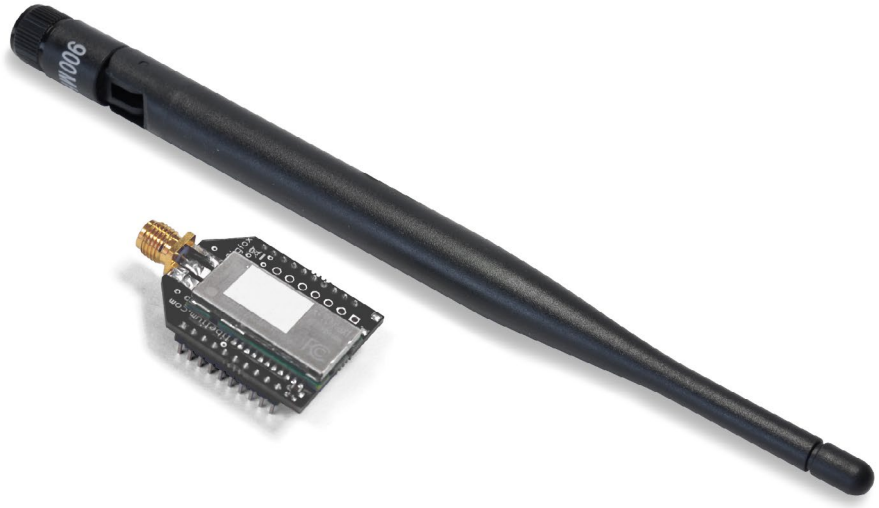


Figure: Sigfox module with 4.5 dBi antenna

2.2. Power consumption

The Sigfox module is powered at 3.3 V. The next table shows the average current consumption in different states of the module.

2.2.1. Sigfox EU

The Sigfox EU 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	0 mA
Transmitting data	~ 52 mA
Receiving data	~ 13 mA

Figure: Power consumption table

2.2.2. Sigfox US and APAC / LATAM

The Sigfox US 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	0 mA
Transmitting data	~ 230 mA
Receiving data	~ 13 mA

Figure: Power consumption table

2.3. Region standards

Due to local regulations, laws or other operating constraints, the configuration of the Sigfox network differs among countries or regions. In other words, there is not a single Sigfox protocol, there are actually several versions. These RF configurations are different in frequency band, radiated power, time-on-air or duty cycle, etc. Check the Sigfox Network.

The next table shows the Sigfox zones and the Libelium's module that supports each zone:

Zone	Supported by
RC1 - Europe, Oman, Iran, South Africa, Tunisia, UAE	Libelium's Sigfox EU module
RC2 - USA, Mexico, Brazil	Libelium's Sigfox US module
RC3 - Japan	**Not** Supported
RC4 - Australia, New Zealand, Singapore, Taiwan, Hong Kong, Colombia, Argentina, Costa Rica, Thailand, Malaysia, Ecuador, Panama, El Salvador, Chile	Libelium's Sigfox APAC / LATAM module

A radio created for one zone will **not** work in other zone. Besides, a radio **cannot** be changed or reprogrammed to work in other zone (with the exception of RC2 and RC4).

Also, remember that any Sigfox transmission must be received by Sigfox base stations. This means that Sigfox (or any of his partners) must have installed their infrastructure in your country. It is not possible for one customer to build his own private Sigfox network, so please check if there is an operating Sigfox network in your country, and specifically if there is Sigfox coverage in the area of your project.

2.4. Time consumption

The client must keep in mind that sending processes take a while to be accomplished. So, this module would fit into a group of high-power consumption modules. In the case that acknowledgements are required, the time consumption increases several seconds.

Transmit mode	Time elapsed
Sending (from ON state)	~ 6 seconds
Sending (from OFF state)	~ 12 seconds
Sending with ACK (from ON state)	~ 39 seconds
Sending with ACK (from OFF state)	~ 45 seconds

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

- Sending (from ON state): The sends the packet to the network (~6 seconds).
- Sending (from OFF state): The module powers on (~6 seconds) and then the packet is sent to the network (~6 seconds).
- Sending with ACK (from ON state): The packet is sent to the network (~6 seconds), then the module enters an idle state waiting for incoming data (~14 seconds) and finally a downlink transmission is performed from the network to the module(~19 seconds).
- Sending with ACK (from OFF state): The module powers on (~6 seconds), the packet is sent to the network (~6 seconds), then the module enters an idle state waiting for incoming data (~14 seconds) and finally a downlink transmission is performed from the network to the module (~19 seconds).

2.5. How to connect the module.

This module can be connected to both SOCKET0 and SOCKET1 on 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.

2.6. 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 the user to plug any kind of radio module through the UART0. On the other hand, SOCKET1 permits to connect a radio module through the UART1.

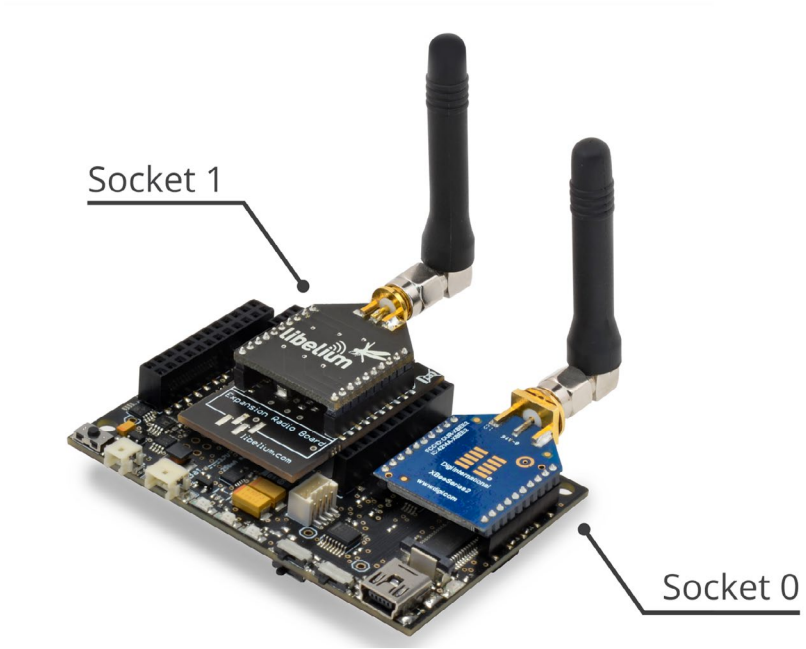


Figure: Use of Expansion Board

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

Selecting `SOCKET0`: `Sigfox.ON(SOCKET0);`
Selecting `SOCKET1`: `Sigfox.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 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:
  - 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

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

3.1. Waspmote libraries

3.1.1. Waspmote Sigfox files

The files related to the Sigfox libraries are:

```
WaspSigfox.h
WaspSigfox.cpp
```

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

```
#include <WaspSigfox.h>
```

3.1.2. Class constructor

To start using the Waspmote Sigfox library, an object from the 'WaspSigfox' class must be created. This object, called `Sigfox`, is already created by default inside Waspmote Sigfox 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
SIGFOX_ANSWER_OK	Successful response to a function
SIGFOX_ANSWER_ERROR	Erratic response to a function
SIGFOX_NO_ANSWER	No response to a command
AT_OK	Static string to check successful responses from the module
AT_ERROR	Static string to check erratic responses from the module
AT_HEADER	Static string to create commands
AT_HEADER_SLASH	Static string to create commands

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 baudrate to be used when the module is switched on
`_uart`	The MCU uart selected (regarding the socket used: SOCKET0 or SOCKET1)
`_command`	The buffer of memory used for creating commands to be sent to the module
`_txFreq`	The carrier frequency for uplink communication
`_rxFreq`	The carrier frequency for downlink communication
`_firmware`	The buffer of memory where the module's firmware is stored
`_id`	The Sigfox module unique ID (given when it is registered)

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 `Sigfox`.

All public functions return three possible values:

- `SIGFOX_ANSWER_OK` = 0
- `SIGFOX_ANSWER_ERROR` = 1
- `SIGFOX_NO_ANSWER` = 2

3.2. Module system management features

Remember you need to purchase Sigfox connectivity to be able to use the Sigfox network.

3.2.1. Switch on

The `ON()` function allows the user to switch on the Sigfox module, open the MCU UART for communicating with the module and automatically enter into command mode. After this step the module will be able to receive more commands to configure it or send packets to the Sigfox network. It is necessary to indicate the socket that it is being used: `SOCKET0` or `SOCKET1`.

Example of use for SOCKET0:

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

Figure: Sigfox module in SOCKET0

3.2.2. Switch off

The `OFF()` function allows the user to switch off the Sigfox 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:

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

3.2.3. Read module ID

The `getID()` function allows the user to get the Sigfox module identifier. This ID is a variable 4- to 8-digit hexadecimal number.The identifier is stored in the `_id` attribute.

Example of use:

```
{
   Sigfox.getID();
}
```

Related variable:

`Sigfox._id` → Stores the module's identifier

Example of getting Sigfox module identifier:
www.libelium.com/development/waspmote/examples/sigfox-01-read-id

3.3. Sigfox features

3.3.1. Send Sigfox packets

The `send()` function allows the user to send Sigfox packets to the network. Sigfox packets maximum payload is 12 bytes. So the user must fit to this restriction when sending new packets.

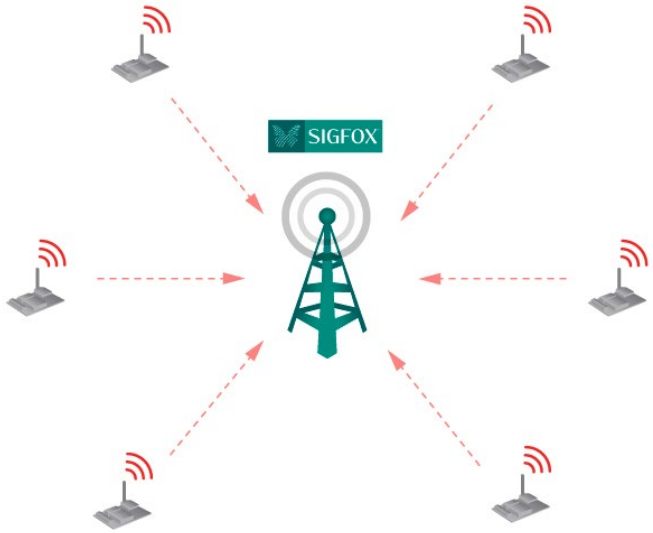


Figure: Sigfox packet sending

There are two sending function prototypes. They are explained below:

- **Send packets defined as strings**

The next function shows how to send packets defined as a string. So every single byte in the packet must be previously converted into the ASCII representation. In the next example, it is possible to see how to send a 12-byte packet composed of 24 hexadecimal digits.

Example of use:

```
{
   char data[] = “010203040506070809 0A0B”;
   Sigfox.send(data);
}
```

Example of sending a packet as string:
www.libelium.com/development/waspmote/examples/sigfox-02-send-data-string

- **Send packets defined as array of bytes**

On the other hand, the next function shows how to send packets defined as array of bytes. It will be necessary to indicate both pointer to the array and the length of the array. This function prototype can be useful for the user if they prefer to define their own packet formats.

Example of use:

```
{
   uint8_t data[] = { 0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B };
   Sigfox.send(data, 12);
}
```

Example of sending a packet as array of bytes:
www.libelium.com/development/waspmote/examples/sigfox-03-send-data-array

3.3.2. Send Sigfox packets with ACK

The `sendACK()` function allows the user to send Sigfox packets to the network acknowledging the packet reception. So it is possible to know when a transmission went well for sure. The user can configure the "device type" in the Sigfox back-end in order to acknowledge the packet. It is even possible to send back information from the server to the module within a downlink transmission when the acknowledge is performed. In the case downlink transmission are set for a specific device, after receiving the downlink data the module answers the server with another acknowledgment which is free of charge. Please, refer to "Sigfox Back-End" section in order to know more.

In the case of using acknowledgements, the transmission of packets takes much longer. The user must keep in mind that ACKs are sent once the packet is received by the Sigfox cloud. So, transmissions processes can take longer than 30 seconds. This will increase the energy consumption of the system. In this case, Sigfox packets also have a maximum payload of 12 bytes. So the user must fit to this restriction when sending new packets.

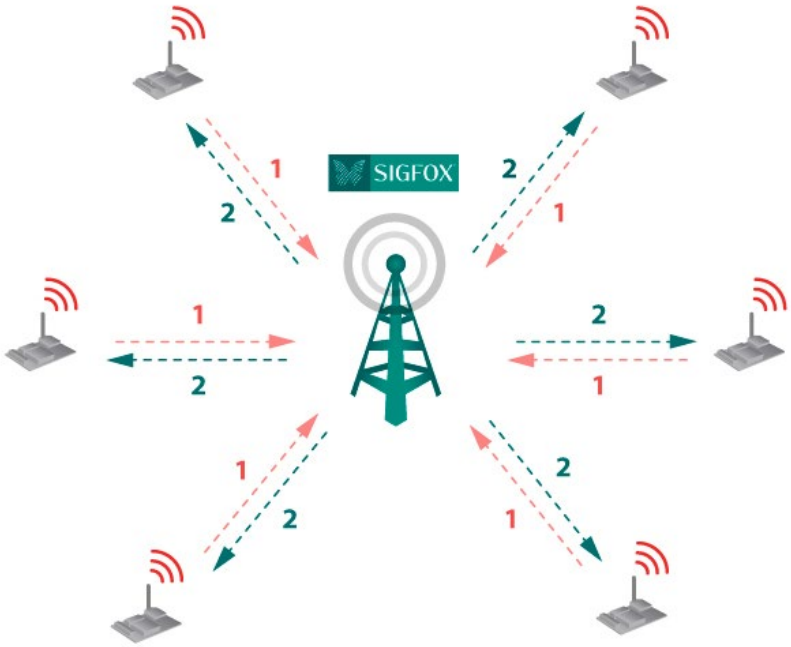


Figure: Sigfox packet sending with ACK

There are two sending function prototypes. They are explained below:

- **Send packets defined as strings with ACK**

The next function shows how to send packets defined as a string waiting for an ACK. So every single byte in the packet must be previously converted into the ASCII representation. In the next example, it is possible to see how to send a 12-byte packet composed of 24 hexadecimal digits.

Example of use:

```
{
   char data[] = "0102030405060708090A0B";
   Sigfox.sendACK(data);
}
```

Example of sending a packet as string with ACK:

www.libelium.com/development/waspmote/examples/sigfox-04-send-data-string-ack

- **Send packets defined as array of bytes with ACK**

On the other hand, the next function shows how to send packets defined as array of bytes waiting for an ACK. It will be necessary to indicate both pointer to the array and the length of the array. This function prototype can be useful for the user if they prefer to define their own packet formats.

Example of use:

```
{
   uint8_t data[] = { 0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B };
   Sigfox.sendACK(data, 12);
}
```

Example of sending packet as array of bytes with ACK:
www.libelium.com/development/waspmote/examples/sigfox-05-send-data-array-ack

3.3.3. Send keep-alive

The `sendKeepAlive()` function allows the user to send a Keep-Alive message to the network. This can be useful to perform daily heart beats, and this way ensure the system will work when most needed in the event of an alarm.

Example of use:

```
{
   Sigfox.sendKeepAlive();
}
```

Example of sending a keep-alive:
www.libelium.com/development/waspmote/examples/sigfox-06-send-keep-alive

3.3.4. Set/Get power level

The `setPower()` function allows the user to set the RF power level in dBm. Sigfox EU power range is from 0 to 14 dBm. The default value is 14 dBm. Remember that you should not transmit at more than 14 dBm in Europe, due to ETSI regulation. Sigfox US power levels are: 15, 20, 22, 23 and 24. The default value is 24 dBm.

Example of use:

```
{
   Sigfox.setPower(14);
}
```

The `getPower()` function allows the user to query the power level.

Example of use:

```
{
   Sigfox.getPower();
}
```

Related variable:

`Sigfox._power` → Stores the module's power level

Example of setting and getting power level:
http://www.libelium.com/development/waspmote/examples/sigfox-07-set-get-power-level

3.3.5. Sigfox TX test

The `testTransmit()` function allows the user to send a TX Sigfox test to the network. The function will ask for three different inputs:

- **Count:** From 0 to 65535, count of Sigfox test RF messages
- **Period:** From 1 to 255, period in seconds between Sigfox test RF messages
- **Channel:** From 0 to 180 or 220 to 400 or -1 (random channel). Channel number to use for Sigfox test RF messages. Channels have a fixed 100 Hz bandwidth, starting at 868.180 MHz for channel 0, ending at 868.198 MHz for channel 180, restarting at 868.202 MHz for channel 220 and ending at 868.220 MHz for channel 400.

Example of use:

```
{
   int16_t   count   = 10;
   uint16_t  period  = 1;
   int16_t   channel = 1;

   Sigfox.testTransmit(count, period, channel);
}
```

Example of TX Sigfox test:

http://www.libelium.com/development/waspmote/examples/sigfox-08-test-sigfox

3.3.6. Show firmware version

The `showFirmware()` function allows the user to print the Firmware version via USB port. The result is stored in the attribute `_firmware`, which is a buffer of the class. The function displays the library version number as follows: SOFTxxxx, where the "xxxx" stands for the version number.

Example of use:

```
{
   Sigfox.showFirmware();
}
```

Example of showing the firmware version:

http://www.libelium.com/development/waspmote/examples/sigfox-09-show-firmware

3.3.7. Configure Sigfox region

This feature is only available for the Sigfox US module and the Sigfox APAC / LATAM module.

The `setRegionRC4()` function allows the user to configure the module to work in the RC4 zone. With the `setRegionRC2()` function the user can configure the module back to the RC2 zone. These functions configure internal variables of the module to the necessary parameters to work in different Sigfox radio configuration zones.

Example of use:

```
{

      Sigfox.setRegionRC4();

}
```

Example of use:

```
{

      Sigfox.setRegionRC2();

}
```

3.4. P2P Mode - Direct communication between nodes (LAN Interface)

Besides using the Sigfox network, it is possible to set up a **LAN network** between several modules so as to establish **P2P communications**.

This feature is only available for the Sigfox EU module (Europe version). The US version module (Sigfox US) can only transmit frames with the Sigfox mode.

The LAN network operates using time division duplexing (TDD), where the device alternately transmits and receives data packets over the same radio channel. Thus, P2P connections can be made between several modules. If only one receiver is used and several modules transmit data, then a **star topology** network is created.

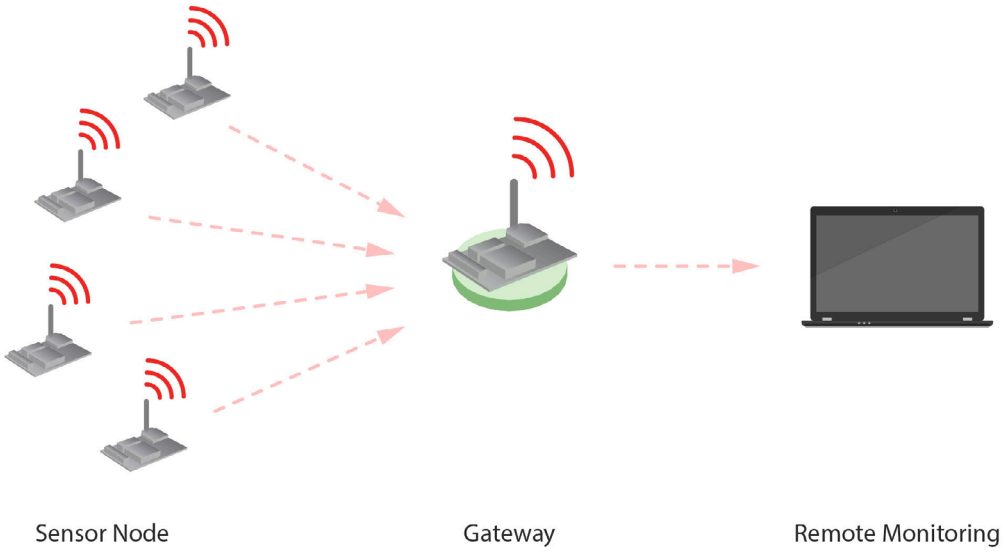


Figure: LAN network with P2P communication between modules

The **frequency** band used for transmissions is the ISM 869 MHz radio band (868.0 to 869.7 MHz). The devices use a single 25 kHz narrow-band channel to transmit data at 9600 bps using a GFSK modulation.

The **transmit power** can be configured using the proper function, in order to reduce the power consumption or increase the radio range when required.

The maximum **payload** is 17 bytes. The packets are acknowledged by the receiver. If this acknowledgement is not received within 2 seconds, the message frame is resent up to 2 times before giving up.

Regarding the addressing of packets, each module is assigned with a logic 24-bit logic **address** and a corresponding 24-bit address **mask**. The logic address is transmitted into the RF frames that are sent and matched by the receiver after applying the address mask to it. The default null address and full address mask (all bits set to 1) ensure that receiving is enabled by default. However, the recommended setup is to have a full address mask (all bits set to 1) for the transmitter and a partial address mask (not all bits are set to 1) for the receiver, both devices having a common address field (i.e. same “subnet”) over the partial address mask bits. This addressing scheme provides a way for transmitters to access the receiver when they are using the same “subnet”, and for the receiver a way to acknowledge a particular frame that has been received from the original transmitter only.

Here is an example for a 4-bit subnet mask:

- Receiver: Address 0x5ED709, Mask 0xF00000

Adress	0	1	0	1	1	1	1	0	1	1	0	1	0	1	1	1	0	0	0	0	1	0	0	1
Mask	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Transmitter: Address 0x55C344, Mask 0xFFFFFF

Adress	0	1	0	1	0	1	0	1	1	1	0	0	0	0	1	1	0	1	0	0	0	1	0	0
Mask	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

3.4.1. LAN address

The `setAddressLAN()` function allows the user to set the module's 24-bit LAN address. The `getAddressLAN()` function allows the user to query the module's LAN address. The attribute `_address` permits to access to the address settings of the module. The range of this variable is from 0x000000 to 0xFFFFFF. The default value is 0x000000.

Example of use:

```
{
   Sigfox.setAddressLAN(0x000001);
   Sigfox.getAddressLAN();
}
```

Related variable:

`Sigfox._address` → Stores the module's LAN address

Example of configuring the LAN settings in the module:

www.libelium.com/development/waspmote/examples/lan-01-configure-module

3.4.2. LAN address mask

The `setMask()` function allows the user to set the module's 24-bit address mask. The `getMask()` function allows the user to query the module's address mask. The attribute `_mask` permits to access to the mask settings of the module. The range of this variable is from 0x000000 to 0xFFFFFF. The default value is 0xFFFFFF.

Example of use:

```
{
   Sigfox.setMask(0xFFFFFF);
   Sigfox.getMask();
}
```

Related variable:

`Sigfox._mask` → Stores the module's LAN mask

Example of configuring the LAN settings in the module:

www.libelium.com/development/waspmote/examples/lan-01-configure-module

3.4.3. Frequency

The `setFrequency()` function allows the user to set the module's frequency value. The `getFrequency()` function allows the user to query the module's frequency value. The attribute `_frequency` permits to access to the frequency settings of the module in Hz units. The range of this variable is from 868000000 to 869700000. The default value is 869312500.

Example of use:

```
{
   Sigfox.setFrequency(869312500);
   Sigfox.getFrequency();
}
```

Related variable:

`Sigfox._frequency` → Stores the module's LAN mask

Example of configuring the LAN settings in the module:

www.libelium.com/development/waspmote/examples/lan-01-configure-module

3.4.4. LAN power

The `setPowerLAN()` function allows the user to set the module's power level for LAN networks. The `getPowerLAN()` function allows the user to query the module's power level. The attribute `_powerLAN` permits to access to the power level settings of the module in dBm units. The range of this variable is from -35 to 14. The default value is 14.

Example of use:

```
{
   Sigfox.setPowerLAN(14);
   Sigfox.getPowerLAN();
}`````
```

Related variable:

`Sigfox._powerLAN` → Stores the module's LAN mask

Example of configuring the LAN settings in the module:

www.libelium.com/development/waspmote/examples/lan-01-configure-module

3.4.5. Send RF messages

The `sendLAN()` function allows the user to send LAN packets to another module. The maximum payload size for LAN packets is **17 bytes**. So the user must fit to this restriction when sending new packets.

There are two sending function prototypes. They are explained below:

- **Send packets defined as strings**

The next function shows how to send packets defined as a string. So every single byte in the packet must be previously converted into the ASCII representation. In the next example, it is shown how to send a 17-byte packet composed of 34 hexadecimal digits.

Example of use:

```
{
   char data[] = "00112233445566778899AABBCCDDEEFF00";
   Sigfox.sendLAN(data);
}
```

Example of sending a packet as string:

www.libelium.com/development/waspmote/examples/lan-02-send-data-string

- **Send packets defined as array of bytes**

On the other hand, the next function shows how to send packets defined as array of bytes. It will be necessary to indicate both pointer to the array and the length of the array.

Example of use:

```
{
   uint8_t data[17];

   data[0]  = 0x00;
   data[1]  = 0x11;
   data[2]  = 0x22;
   data[3]  = 0x33;
   data[4]  = 0x44;
   data[5]  = 0x55;
   data[6]  = 0x66;
   data[7]  = 0x77;
   data[8]  = 0x88;
   data[9]  = 0x99;
   data[10] = 0xAA;
   data[11] = 0xBB;
   data[12] = 0xCC;
   data[13] = 0xDD;
   data[14] = 0xEE;
   data[15] = 0xFF;
   data[16] = 0x00;

   Sigfox.sendLAN(data, 17);
}
```

Example of sending a packet as array of bytes:
www.libelium.com/development/waspmote/examples/lan-05-send-data-array

3.4.6. Receive RF messages in single-packet mode

The `receive()` function allows the user to wait for an incoming LAN packet. The period of time to wait is indicated as input in seconds units.

If a packet is received within the specified timeout, the function returns with ok response and the packet contents are stored in `_packet` structure. The incoming packet will always be a 17-byte length message (represented as a string), independently of the number of bytes sent. If the packet length is less than the maximum size, then the received message is always padded with “00” bytes. If no packet arrives within the given timeout, the function returns with error response.

Anyway, the user must keep in mind that after calling the receiving function, the module exits the receiving mode regardless of the execution response of the function.

Example of use:

```
{
   Sigfox.receive(10);
}
```

Related variable:

`Sigfox._packet` → Stores the received packet inside this string of 35 bytes

Example of receiving a packet as string:
www.libelium.com/development/waspmote/examples/lan-03-receive-single-packet

3.4.7. Receive RF messages in multi-packet mode

The `setMultiPacket()` function allows the user to set up the module for continuous receiving mode. In any moment, a new packet can be received by the module. This function should be called after switching on the module in the case the user wants to prepare the module for receive packets.

Besides, the `getMultiPacket()` function allows the user to wait for an incoming LAN packet. The period of time to wait is indicated as input in seconds units.

If a packet is received within the specified period of time, the function returns with ok response and the packet contents are stored in `_packet` structure. The incoming packet will always be a 17-byte length message (represented as a string), independently of the number of bytes sent. If the packet length is less than the maximum size, then the received message is always padded with "00" bytes. If no packet arrives within the given timeout, the function returns with error response.

In this receiving mode, the module continues in receive mode after calling the `getMultiPacket()` function. So, each time this function is called, a new packet arrival will be treated.

In addition, while the module is in receiving mode, the user is able to transmit packets. After the sending attempt, the module switches back to receiving mode again.

Example of use:

```
{
   // after switching on the module:
   Sigfox.setMultiPacket();

   // every time we listen to a new incoming packet:
   Sigfox.getMultiPacket(10);
}
```

Related variable:

`Sigfox._packet` → Stores the received packet inside this string of 35 bytes

Example of receiving a packet as string:
www.libelium.com/development/waspmote/examples/lan-04-receive-multi-packet

3.4.8. Disable receiving mode

The `disableRX()` function allows the user to disable the receiving mode. So the module stops listening to incoming packets.

Example of use:

```
{
   Sigfox.disableRX();
}
```

3.5. Hybrid Sigfox / P2P mode (P2P + GW to Sigfox Network)

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

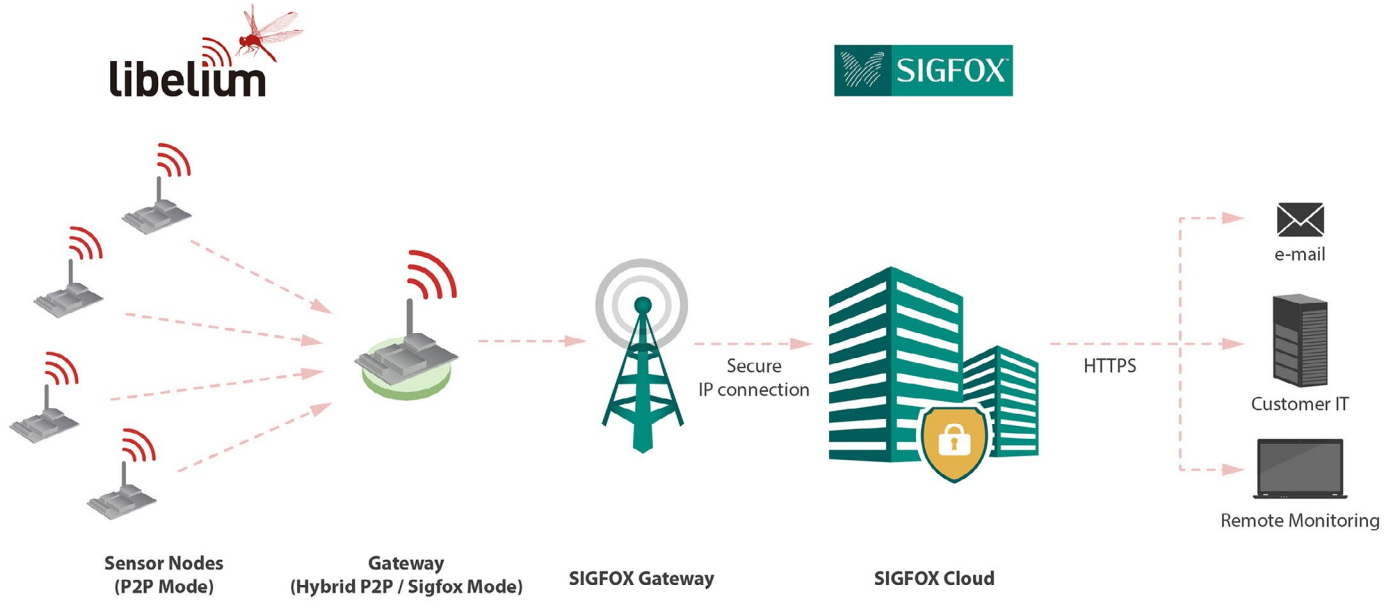


Figure: Hybrid Sigfox / P2P mode

The user must keep in mind that there is a mismatch between the maximum payload in LAN networks (17 bytes) and Sigfox networks (12 bytes). So, the gateway node will need to select the useful 12 bytes from the LAN packet to be sent to the Sigfox network.

The following example shows how to operate as a gateway node sending the first 12 bytes of the incoming LAN packets to the Sigfox network:

www.libelium.com/development/waspmote/examples/lan-06-lan-to-sigfox-gateway

4. Sigfox Back-End

The Sigfox Back-End provides a web application interface for device management and configuration of data integration, as well as standards based web APIs to automate the device management and implement the data integration.

The APIs are based on HTTPS REST requests, as GET or POST and the payload format is JSON.

4.1. Activation process

The first thing you need to do is to contact Sigfox in order to hire the connectivity service for the modules with which you want to transmit to the Sigfox network.

Sigfox will give instructions to you about how to do it. Basically you have to create an account with a valid email address and a password. Then you will create your group, user, device type and device units. You need to pair the Serial Number and PAC number of every module with a license.

4.2. Login

Once you complete the activation process, you will have the credentials (email address and password) needed to connect to the system in:

https://backend.sigfox.com

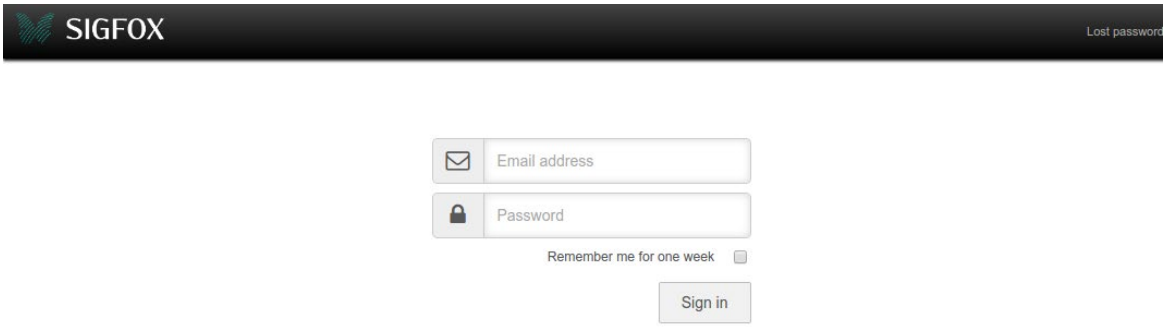


Figure: Sigfox login page

Once logged in, you arrive to the main page of the Sigfox back-end with the latest news and updates, events and coverage map. You can check all this information in the links located on the left menu.

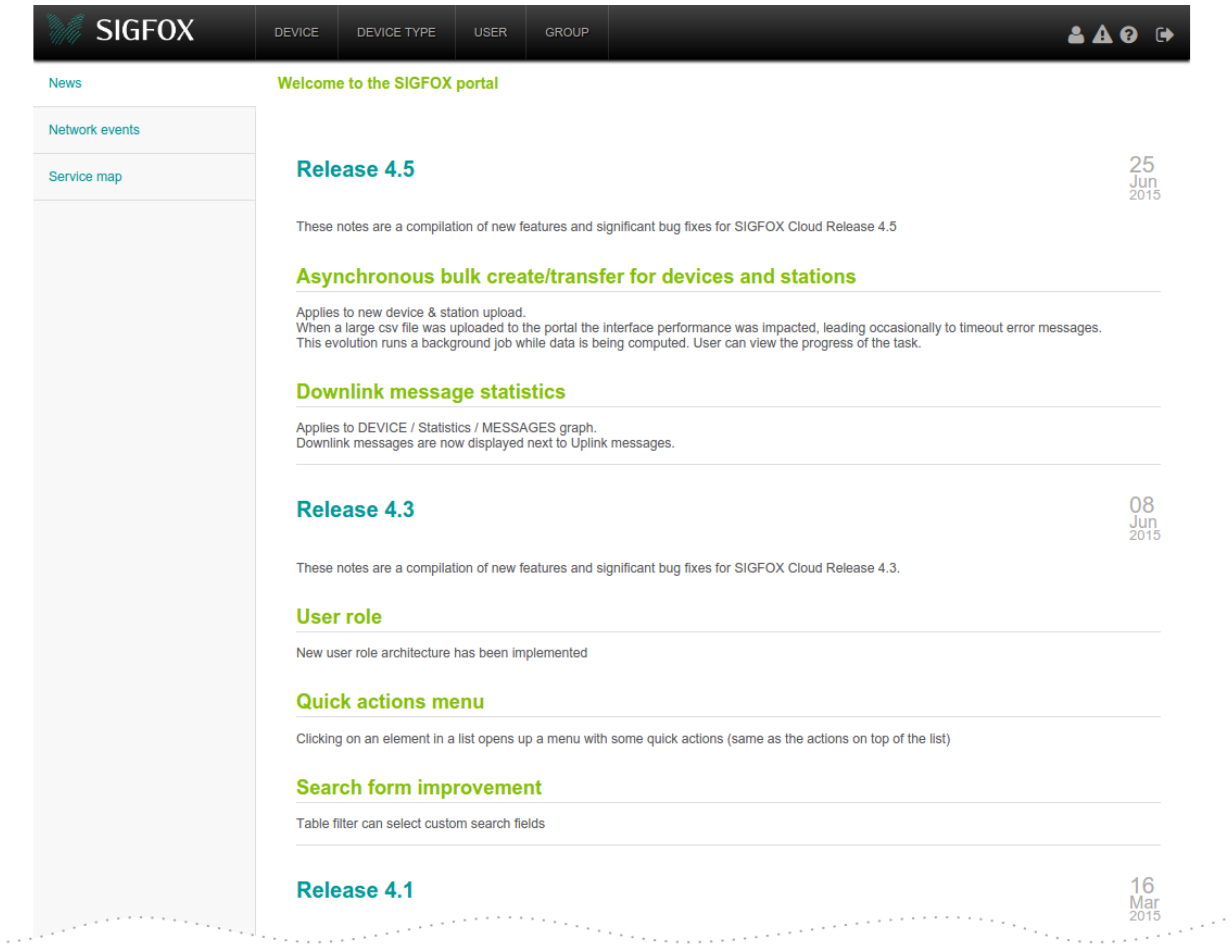


Figure: Sigfox welcome page

4.3. Device

By hitting on “Device” button (top menu), you will get information related to all devices associated to your account. The operations allowed on this list are:

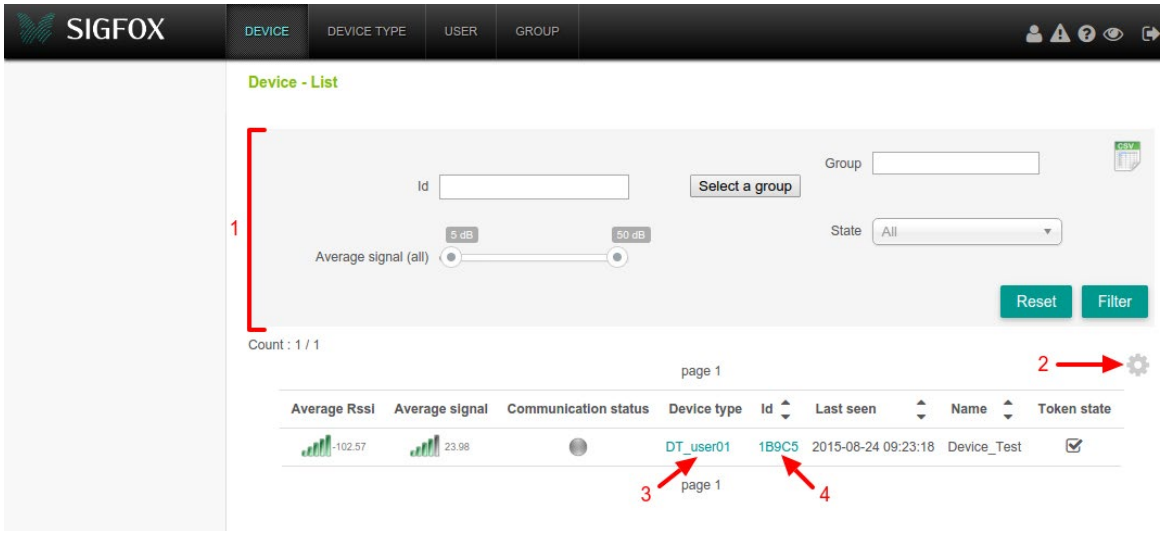


Figure: Sigfox device list page

1. The gray zone is a filter form where you can perform your devices searches.
2. By clicking on this icon you can select the columns you want to see on the list.
3. Device type associated to the device (for further information please refer to Device type chapter).
4. This is the most important operation because it will take you to all the information related to this specific device. On the left side of the window, a menu will appear with all the possibilities available for a single module.

Once a specific device ID is clicked, the back-end leads you to that specific device information area. The next chapters are related to all the options available inside the device area.

4.3.1. Information

Shows all information related to the selected device.

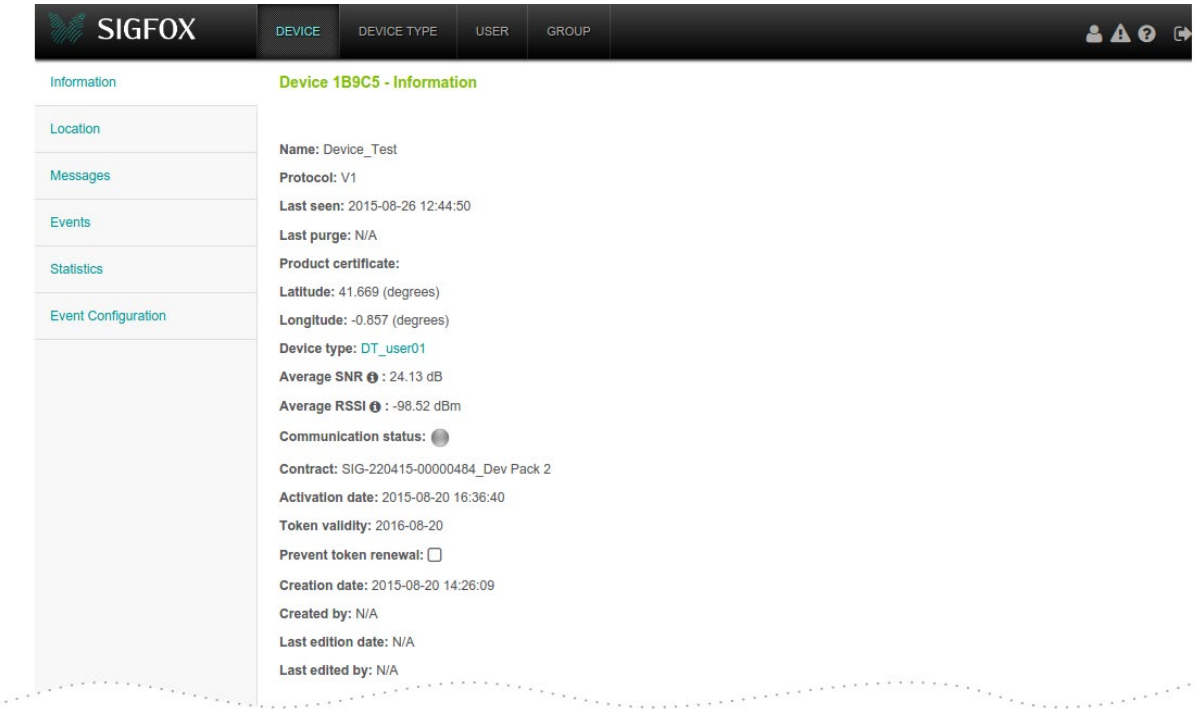


Figure: Sigfox device information page

4.3.2. Location

Displays the device position on a map if the user sets up a specific location in the device's information. If you click on the flag marker you will get more information about the device.

Furthermore, you can select from the drop-down the option "Coverage overlay" to see the coverage information on the map. It is just an approximation based on what base stations received the latest messages.

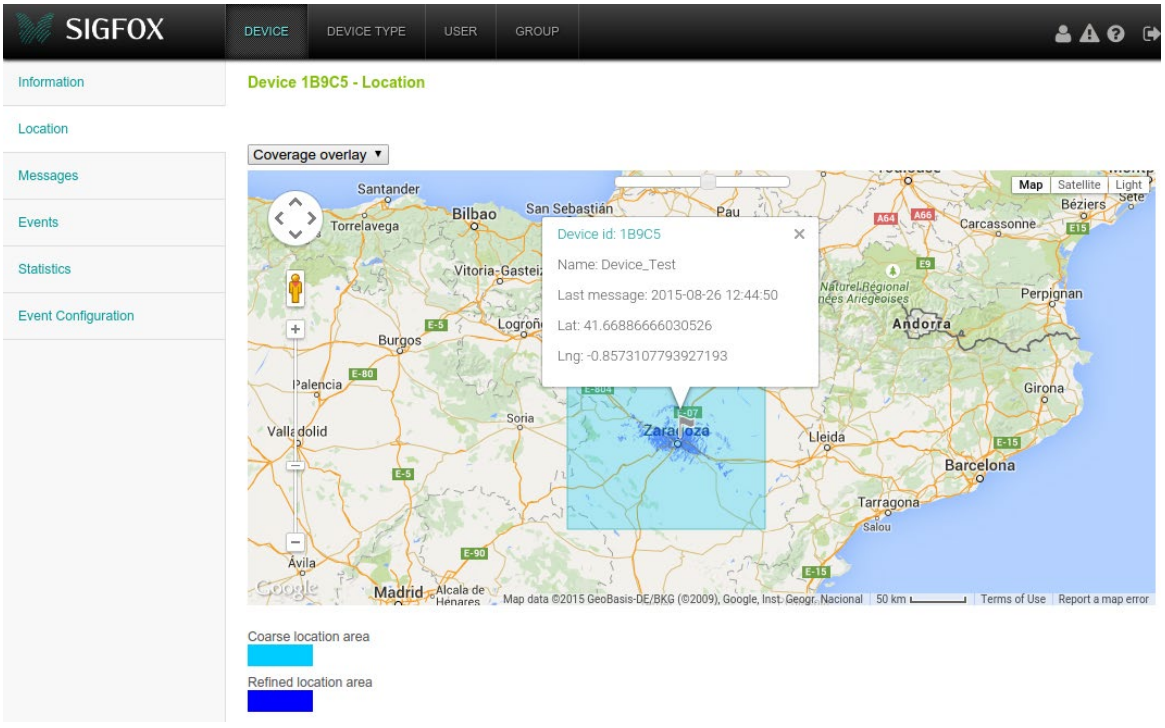


Figure: Sigfox device information page

4.3.3. Messages

In this section you can find all information about all messages received on the Sigfox platform. It is possible to see several columns for:

- The packet arrival timestamp
- The packet data
- Location
- Signal strength
- Packet transmission status depending if callback or ACK is required

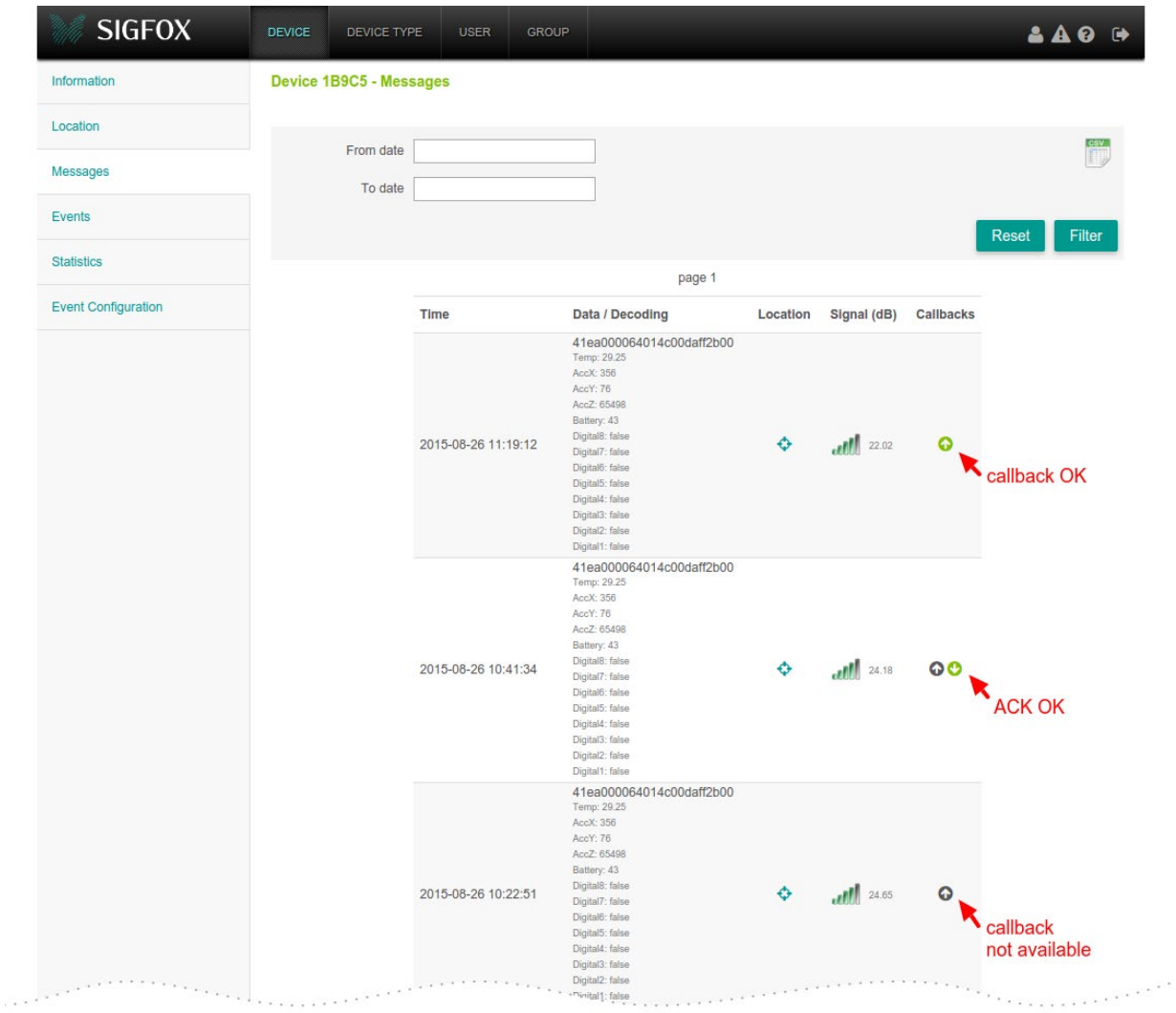


Figure: Sigfox device messages page

4.3.4. Events

A device event is a specific communication parameter change, that can occur during a device lifetime. Device events are used by Sigfox to monitor device activity and to notify administrators of irregular device activity.

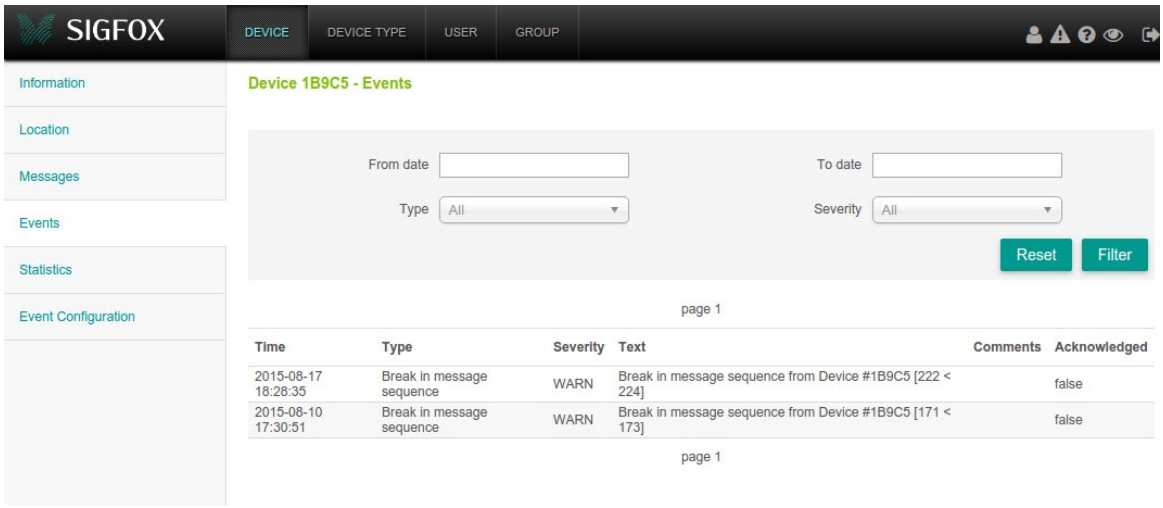


Figure: Sigfox device events page

4.3.5. Statistics

It shows some graphs about the transmitted information:

- Messages
- Bytes
- Received message SNR
- RSSI
- Temperature
- Vdd
- Frequency

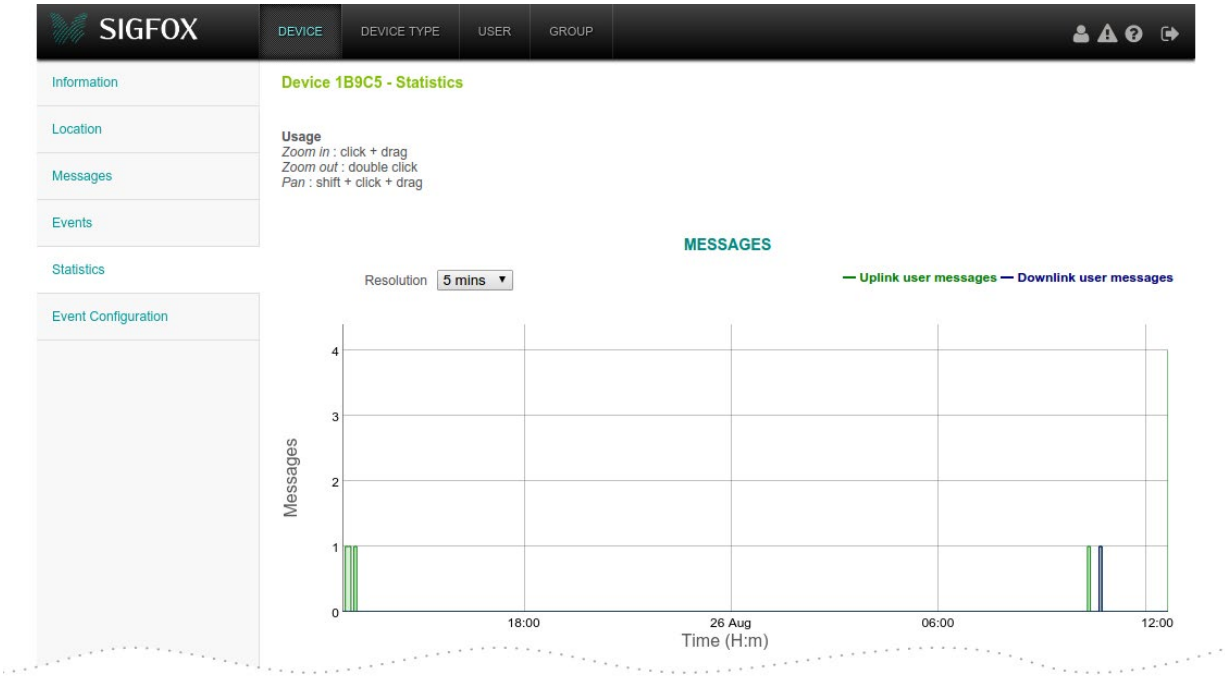


Figure: Sigfox device statistics page

4.3.6. Event configuration

You can configure your own device alerts and be notified by an email when these situations happen. These emails are generic notifications and not customizable. After clicking “New”, the configuration information is shown in order to set the event type and email addresses. For further information, press the “Online help” button.

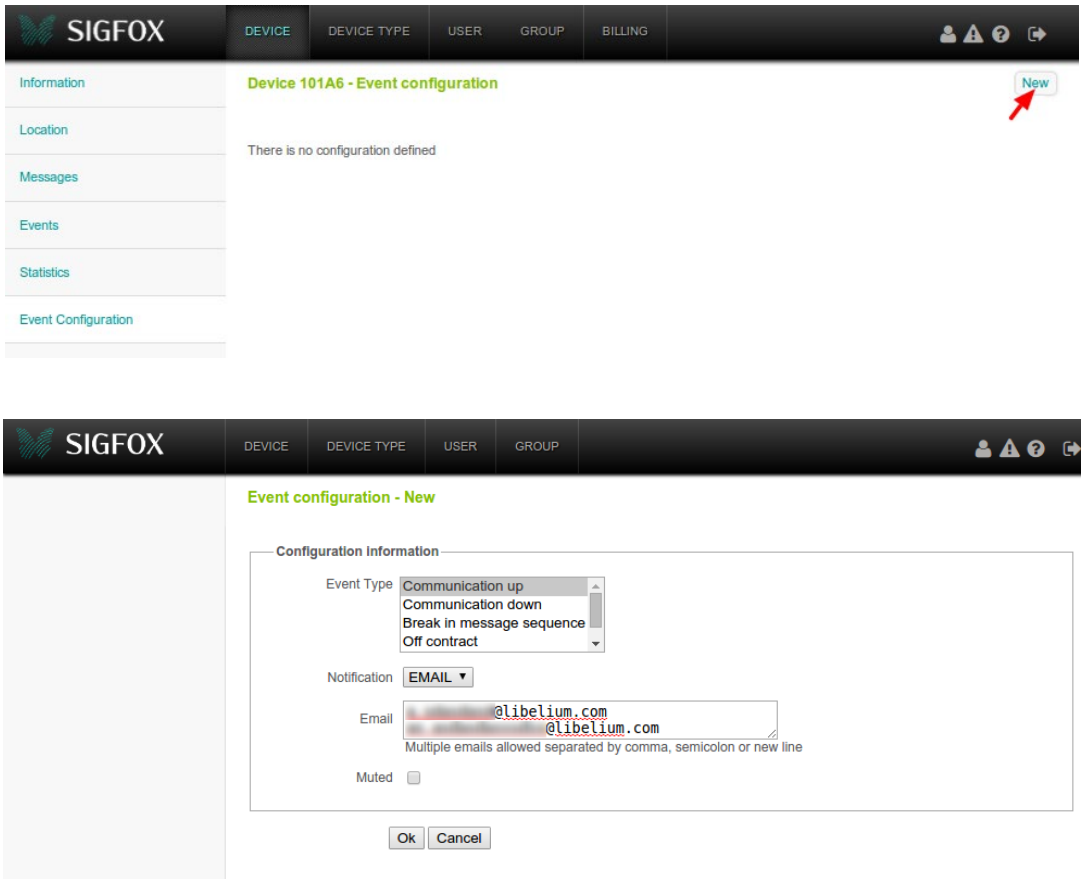


Figure: Sigfox device event configuration process

4.4. Device Type

In the “Device Type” section, you will find information about the characteristics of your device. Firstly, you will find the typical list. Besides there is a “New” button to create new Device Types.

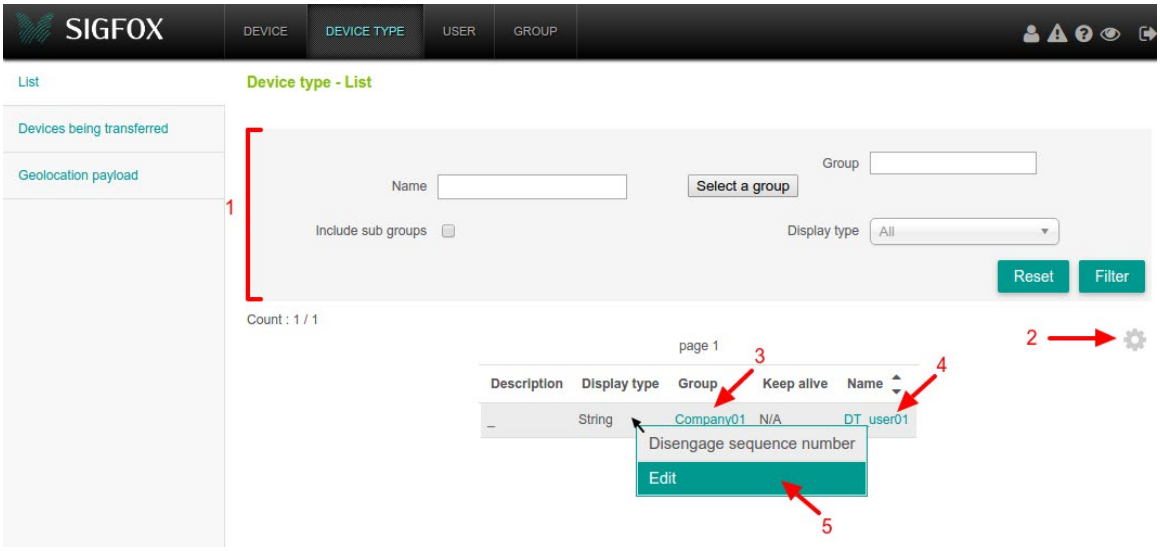


Figure: Sigfox device type list page

1. The gray zone is a filter form where you can perform your devices searches.

2. By clicking on this icon you can select the columns you want to see on the list.

3. This link shows the group information page which the device type selected belongs to.

4. This link takes to the Device Type Information area. Information is shown divided in sections that you can access clicking on the left menu (read the Device Type Information section).

5. The Edit Operation link takes to a form where the user can edit the information of the selected Device Type (read the Edit section).

Once a specific device type is clicked, the back-end leads you to that specific device type information area. The next chapters are related to all the options available inside the device type area.

4.4.1. Edit

On the top right side of the website you find the “Edit” button in order to change the settings of the device type as you consider appropriate for your requirements.

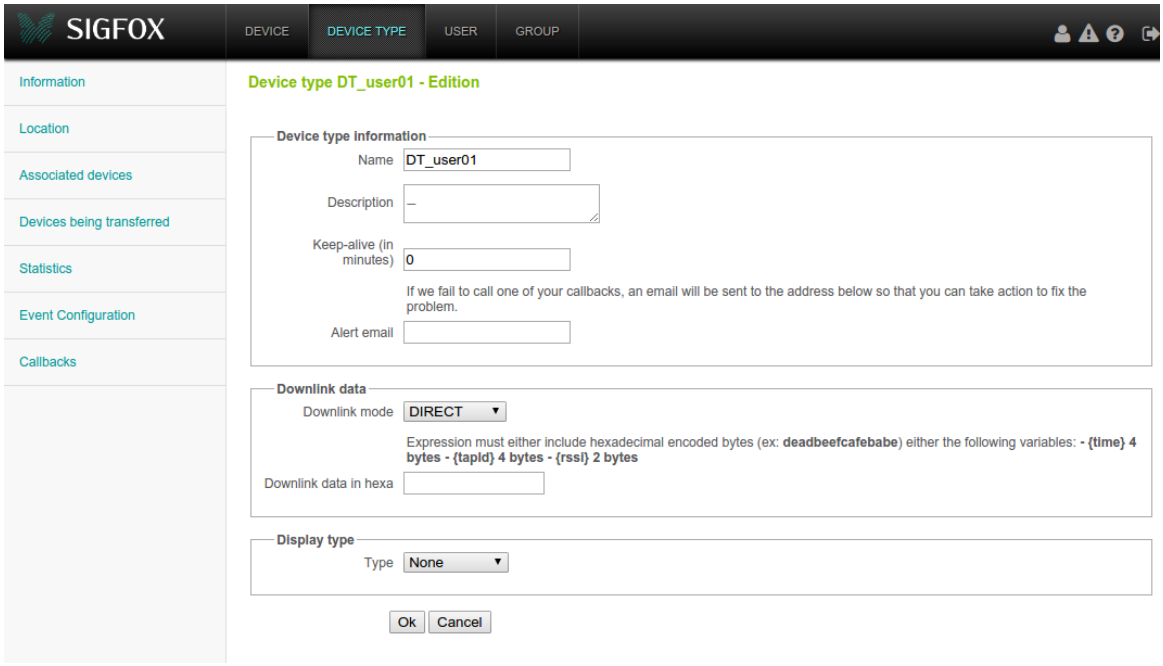


Figure: Sigfox device type edit page

In the “Edition” window it is possible to change:

- Device type information
- Downlink data
- Display type

Device type information block

It is possible to change the “Name” of the device type, add a “Description”, set the “Keep-alive” settings and define the email address where the callbacks are going to be sent in case this device type is enabled for that purpose.

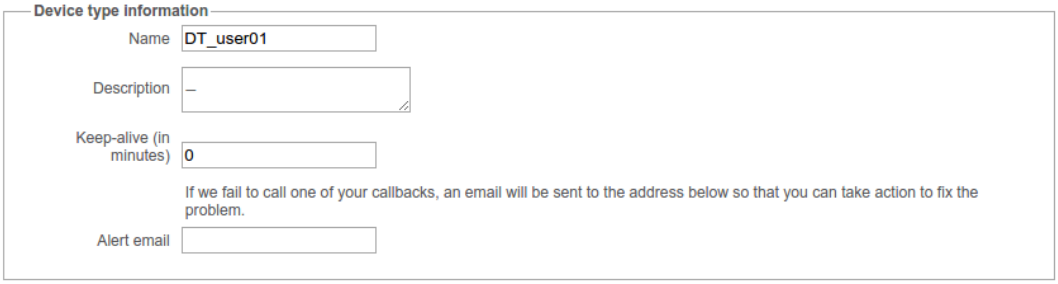


Figure: Sigfox Device Type information block

Downlink data block

In this section it is needed to define the type of downlink data mode used:

- **Direct** (default): If the Sigfox module requires an ACK when a transmission is done, the "Downlink data area" can be filled to answer the module with that information. In the next example, we have used this format: {time}0000{rssi}. When a module sends a frame requiring ACK, it will receive this hexadecimal data as a response. For instance: 55dd7bde0000ffa3. Where 0x55dd7bde is the Unix timestamp (1440578526 → 08/26/2015 @ 8:42am UTC) and 0xffa3 (-93 dBm) is the RRSI value.
- **Callback**: If this mode is enabled, an automatic email is sent upon data arrival to the address defined in the "Device type information" block. The user must keep in mind that when the callback mode is selected in this block, no ACK (downlink data) will be performed to the module. In other words, if the module attempts to send a packet requiring an ACK, the process will fail.

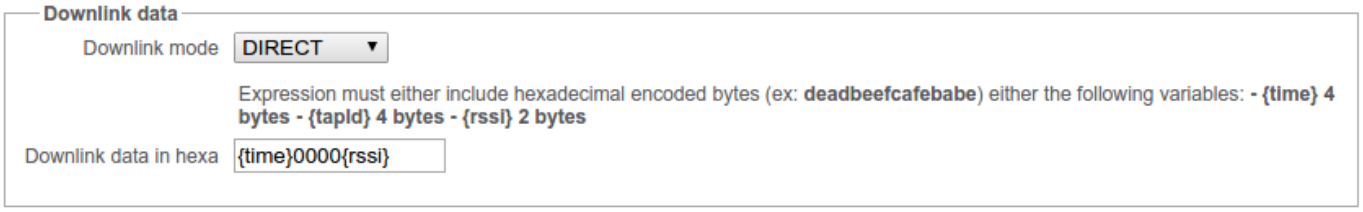


Figure: Sigfox device type downlink data block

Display type block

Here you can create the format of your message. On the list you have to choose:

- None: If you do not want to specify any frame format. It is used for raw data.
- Geolocation: (Not available) This feature is reserved for compatible embedded GPS devices only.
- String: If you want to send a string text.
- Test: Only for testing.
- Custom: You can create your own frame format to send information.

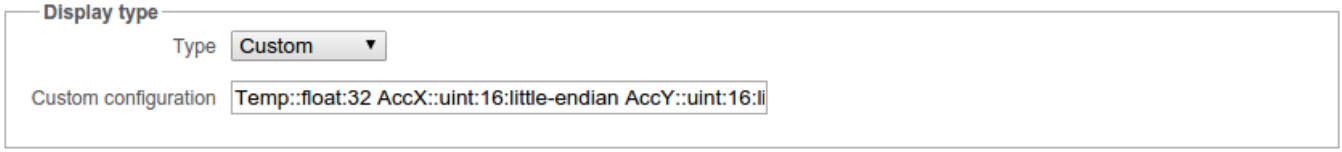


Figure: Sigfox Device Type display type block

In this example, we used this format:

```
Temp::float:32 AccX::uint:16:little-endian AccY::uint:16:little-endian AccZ::uint:16:little-endian Battery::uint:8 Digital8::bool:7 Digital7::bool:6 Digital6::bool:5 Digital5::bool:4 Digital4::bool:3 Digital3::bool:2 Digital2::bool:1 Digital1::bool:0
```

Then in your messages you will see the information like this:

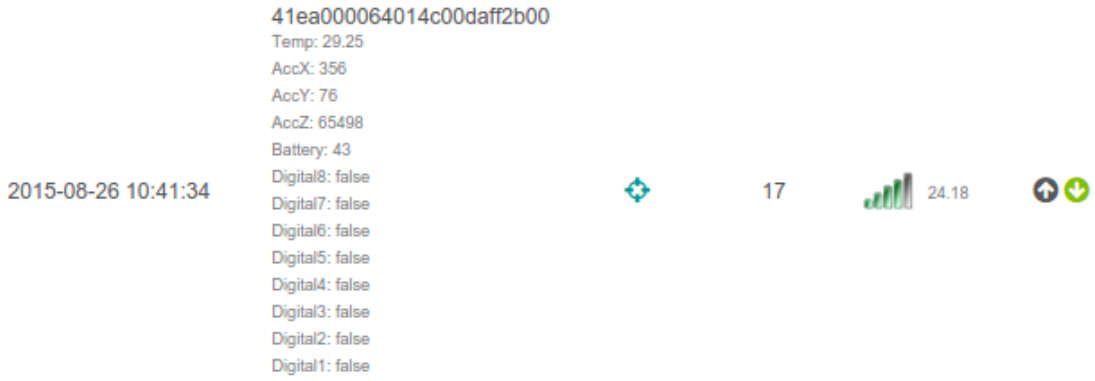


Figure: Sigfox message information received

Here you can find more information about how to customize your own frame:

format = field_def [" " field_def]* ;
field_def = field_name ":" byte_index ":" type_def ;
field_name = (alpha | digit | "-" | "_")* ;
byte_index = [digit*] ;
type_def = bool_def | char_def | float_def | uint_def ;
bool_def = "bool:" ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7") ;
char_def = "char:" length ;
float_def = "float:" ("32") ;
uint_def = "uint:" ("8" | "16" | "24" | "32") [":little-endian" | ":big-endian"] ;
length = number* ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

A field is defined by its name, its position in the message, its length and its type:

- The field name is an identifier including letters, digits and the '-' and '_' characters.
- The byte index is the offset in the message buffer where the field is to be read from, starting from zero. If omitted, the position used is the current byte for boolean fields and the next byte for all other types. For the first field, an omitted position means zero (start of the message buffer).

Next comes the type name and parameters, which varies depending on the type:

- **boolean:** parameter is the bit position in the target byte.
- **char:** parameter is the number of bytes to gather in a string.
- **float:** parameter is the length in bits of the value (only 32 bits). Decoding is done according to the IEEE 754 standard.
- **uint (unsigned integer):** parameters are the number of bits to include in the value, and optionally the endianness for multi-bytes integers. Default is big endian.

Examples:

Format	Message (in hex)	Result
`int1::uint:8 int2::uint:8`	1234	{int1:0x12, int2:0x34 }
`b1::bool:7 b2::bool:6 i1:1:uint:16`	C01234	{b1:true, b2:true, i1:0x1234}
`b1::bool:7 b2::bool:6 i1:1:uint:16:little-endian`	801234	{b1:true, b2:false, i1:0x3412}
`b1::bool:7 b2::bool:6 i1:1:uint:16:little-endian i2::uint:8`	80123456	{b1:true, b2:false, i1:0x3412, i2:0x56}
`str::char:6 i1::uint:16 i2::uint:32`	41424344454601234567890A	{str:"ABCDEF", i1:0x123, i2:0x4567890A}

4.4.2. Information

On the left side of the device type window it is possible to select the information area. It shows all information of the selected Device Type.

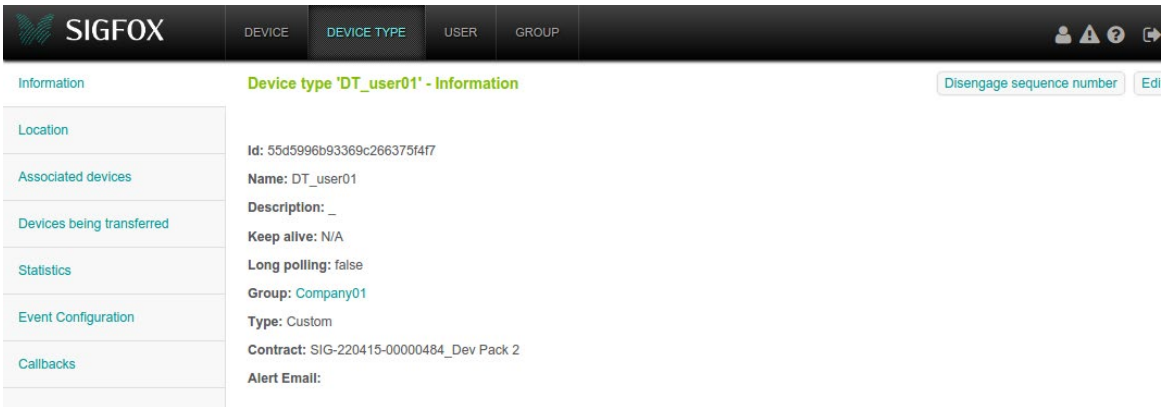


Figure: Sigfox device type information page

4.4.3. Location

Displays all modules assigned to this device type, if they have latitude/longitude data. By hitting on the markers located on the map, another window is opened with more information about the device.

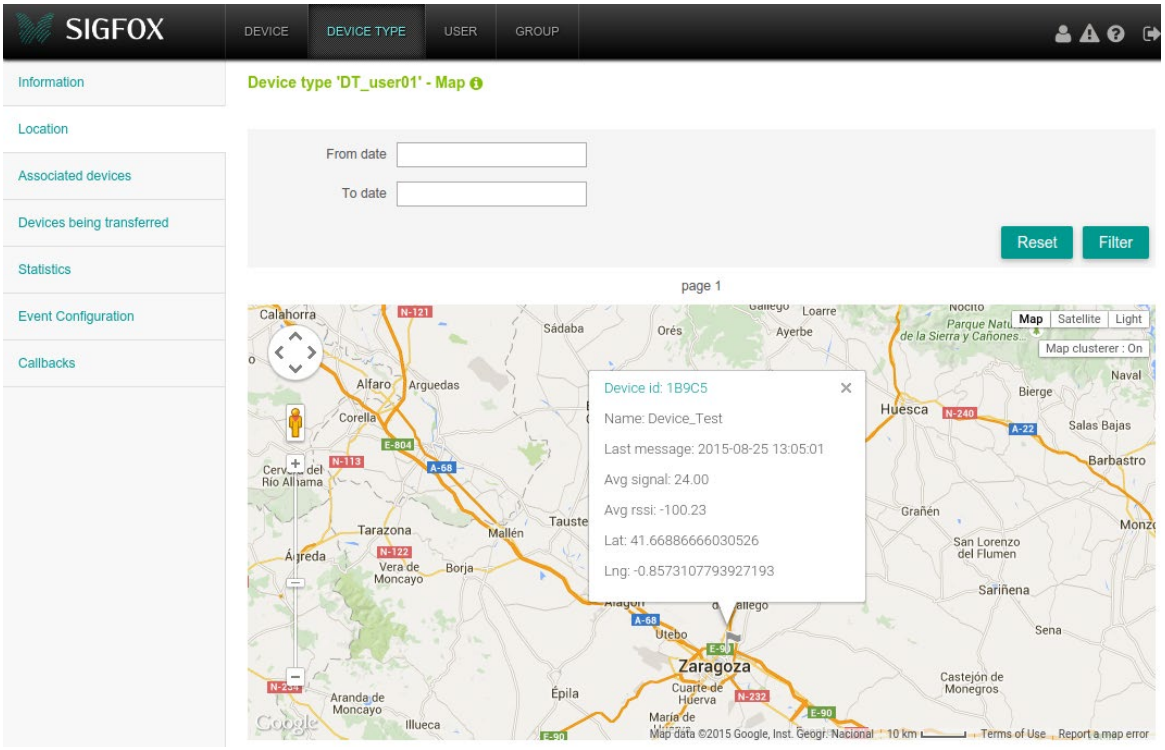


Figure: Sigfox device type location page

4.4.4. Associated devices

Shows a device list (same as Devices section) with all devices associated to this specific device type.

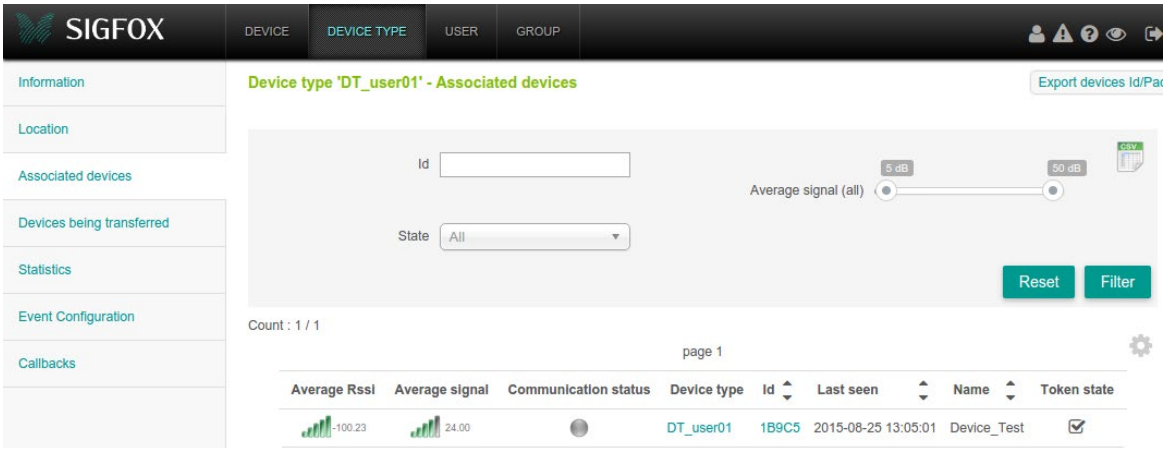


Figure: Sigfox device type associated devices page

4.4.5. Devices being transferred

Here you can find a list with the devices which are being transferred. You have to select the group and confirm the filter form to see the results.

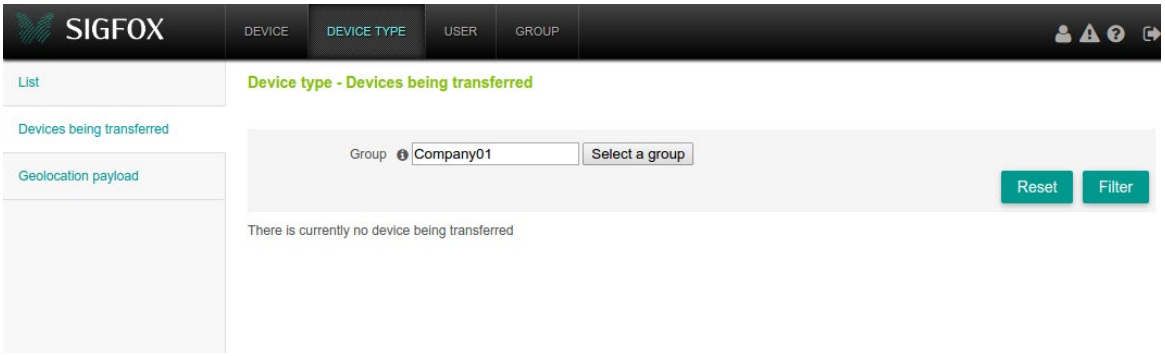


Figure: Sigfox device type devices being transferred page

4.4.6. Statistics

It shows some graphs about the transmitted information.

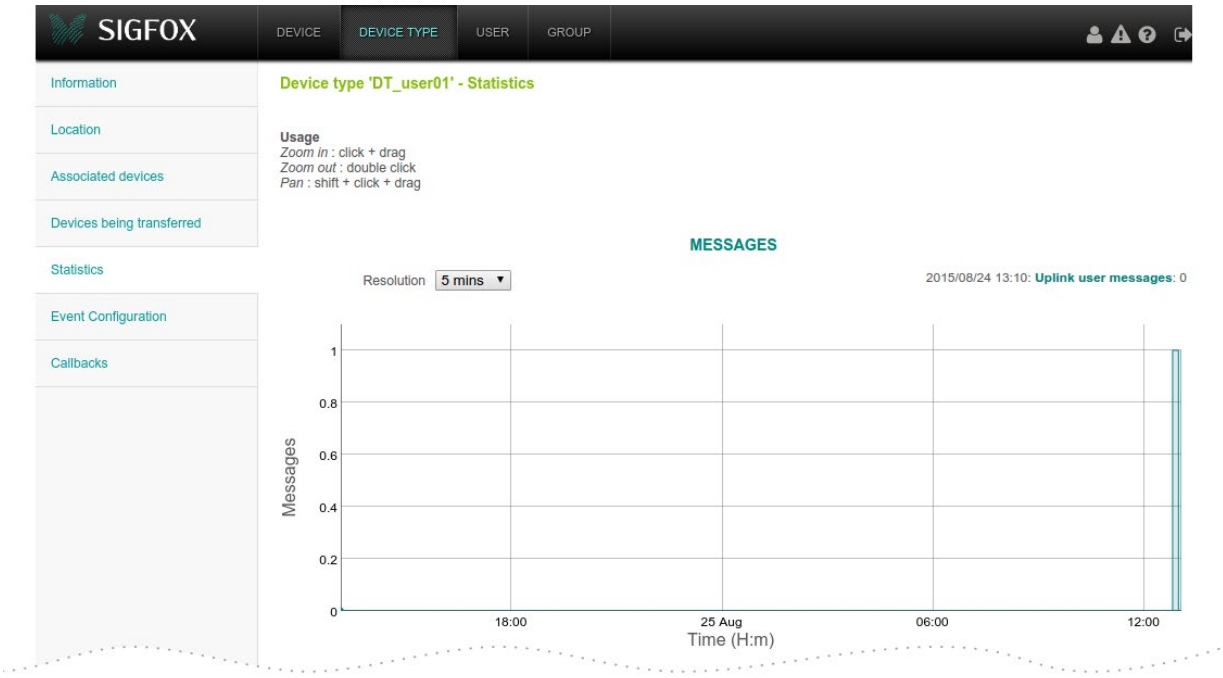


Figure: Sigfox device type statistics page

4.4.7. Event configuration

You can configure alerts which will be sent by email to the addresses you indicate in this section. You have to click on “New” and fill the form with the alert you want to configure.

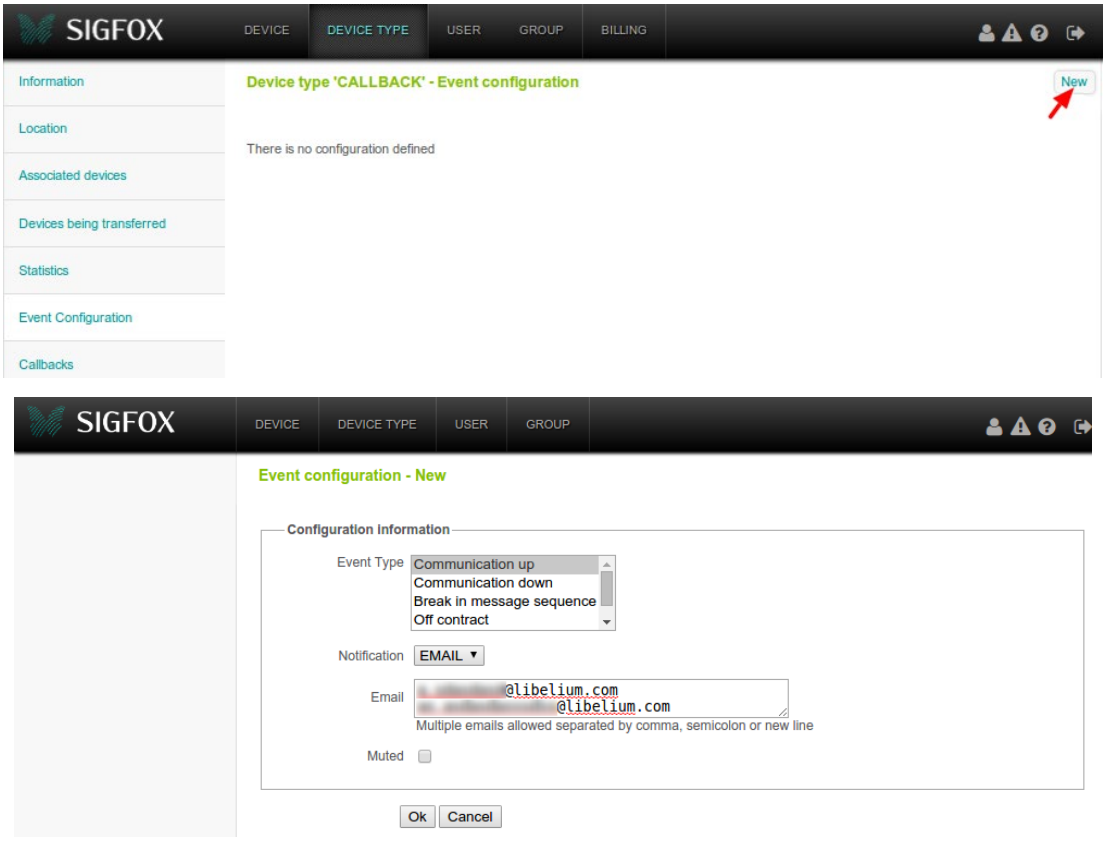


Figure: Sigfox device type event configuration process

4.5. User

In this area you can see all users created in your group. You can create users always associated to your group or edit anyone created before, even your own user.

To create a user, you have to click on the “New” button and fill the form.

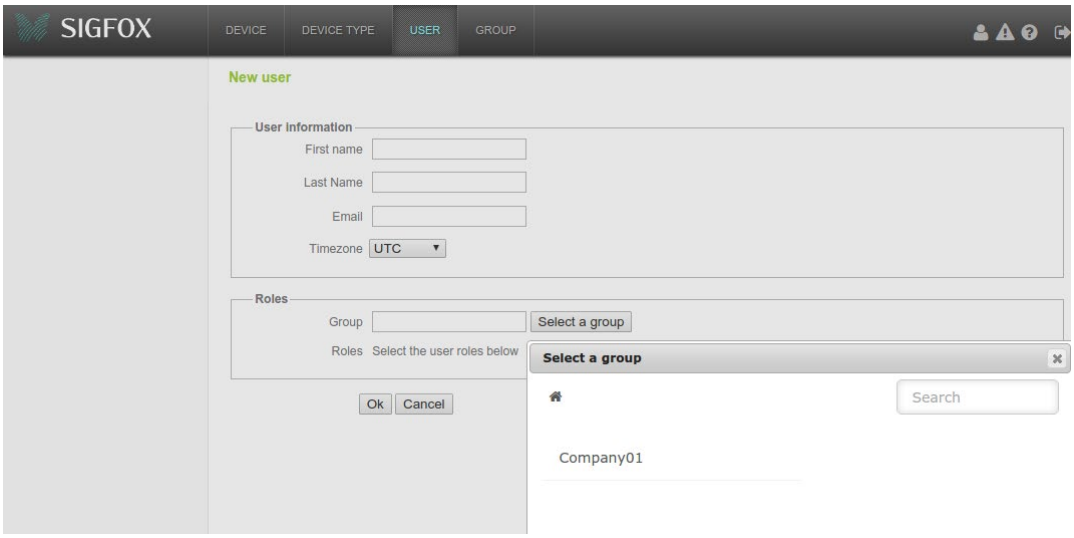


Figure: Sigfox new user page

The roles block is required to set the privileges of the user you want to create. First, you have to select the group, you can only choose one: to the one you belong to. Once the group was selected, you can choose between two roles. If you hit on the info link (the question mark icon near the name), you will get the permissions assigned to the role.

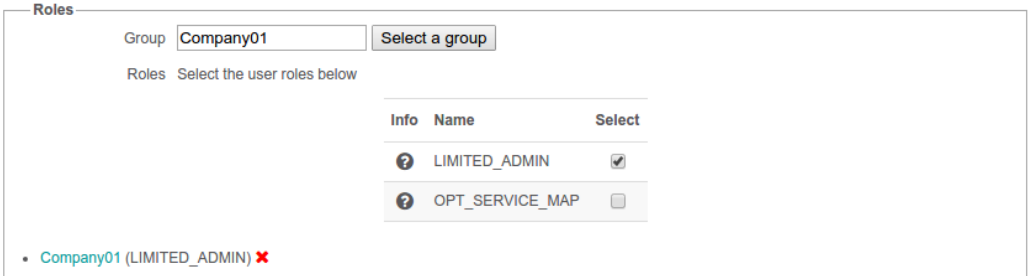


Figure: Sigfox new user roles block

The email field is very important and you must enter an existing account because when you create the user, an email will be sent in order to establish the password. To create the password, you have to follow the link and the instructions sent.

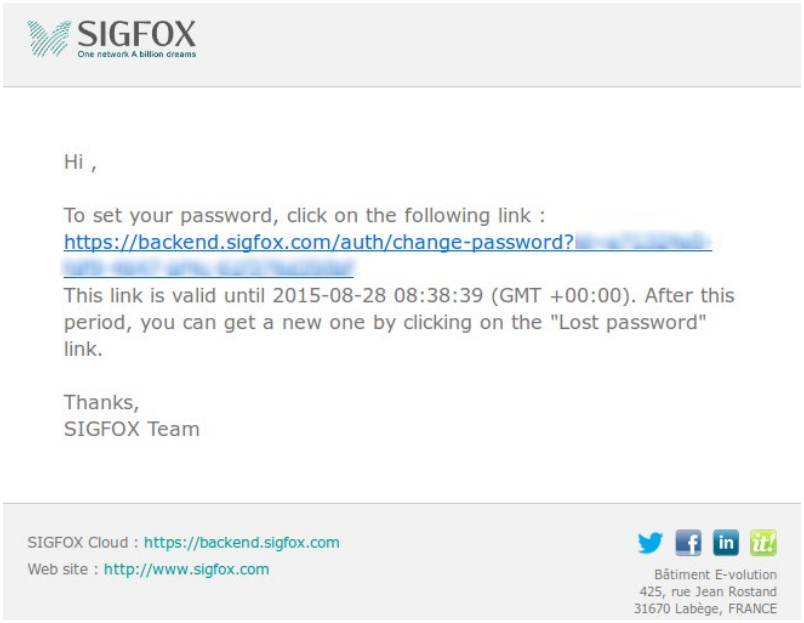


Figure: Sigfox new user roles block

You can edit all users created and delete all, except your own user. If you want to do any of these operations, you have to click on the user register outside of blue links.

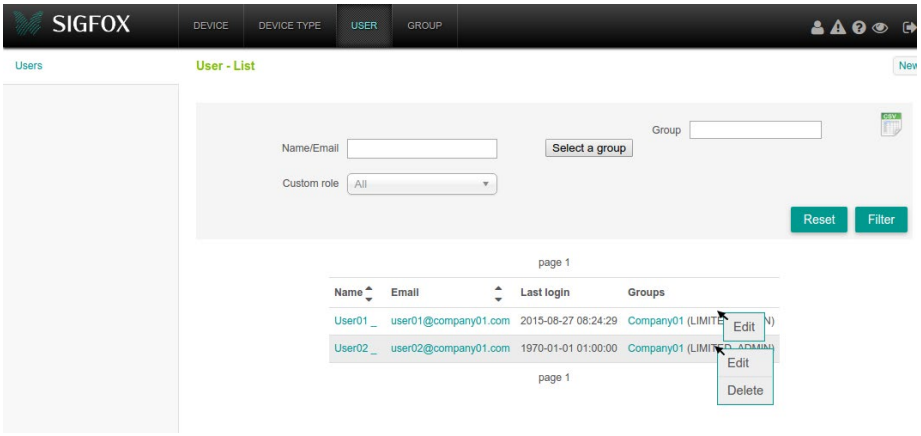


Figure: Sigfox user list options

You can edit all information you can see in the form, even the password, but take into account your password will be required to change other user's password (excluding your own user), as you can see in the next figures.

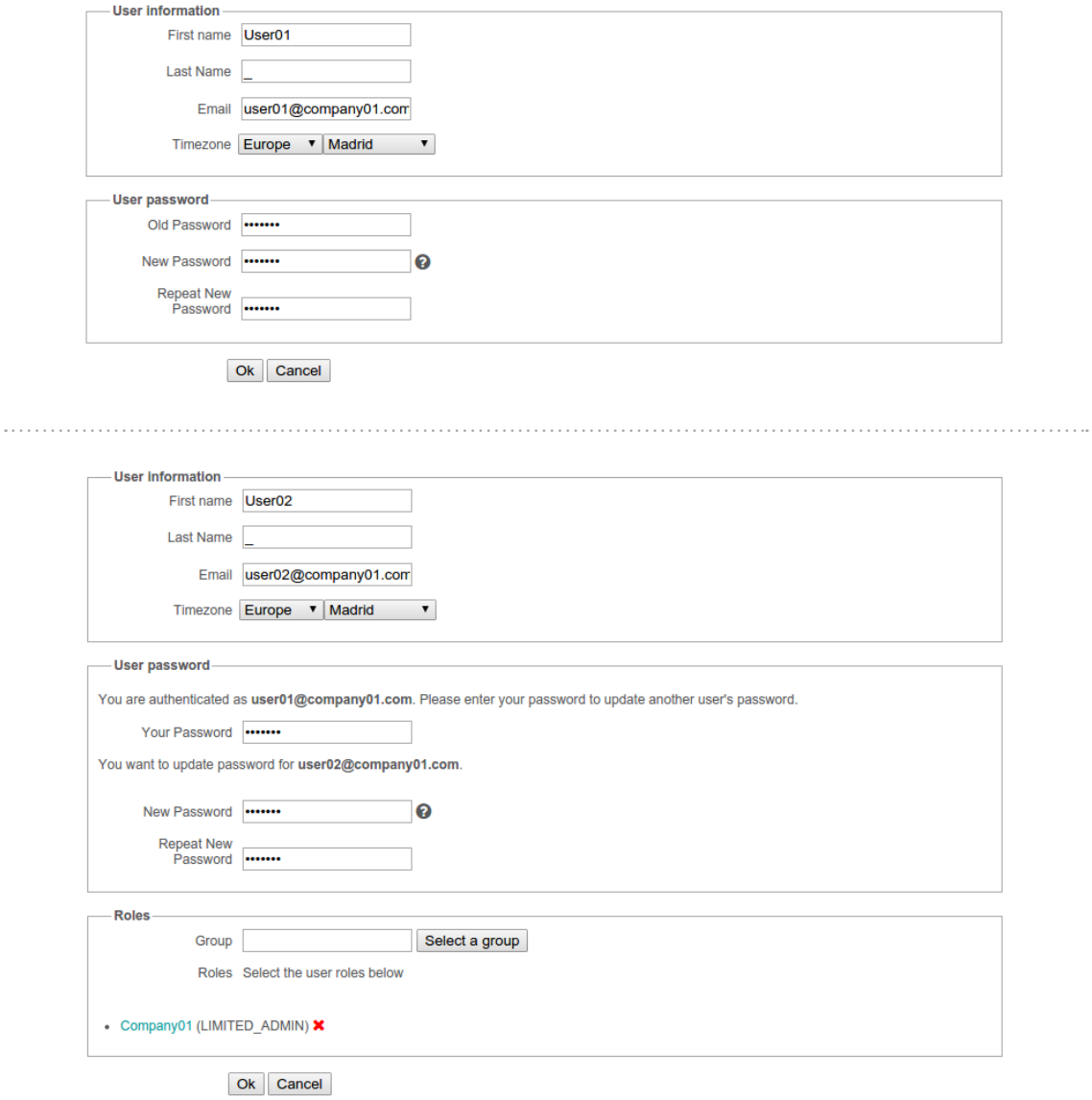


Figure: Sigfox edit user options

4.6. Group

In this section you can see only the group you belong to. You will find more information shown on the left menu.

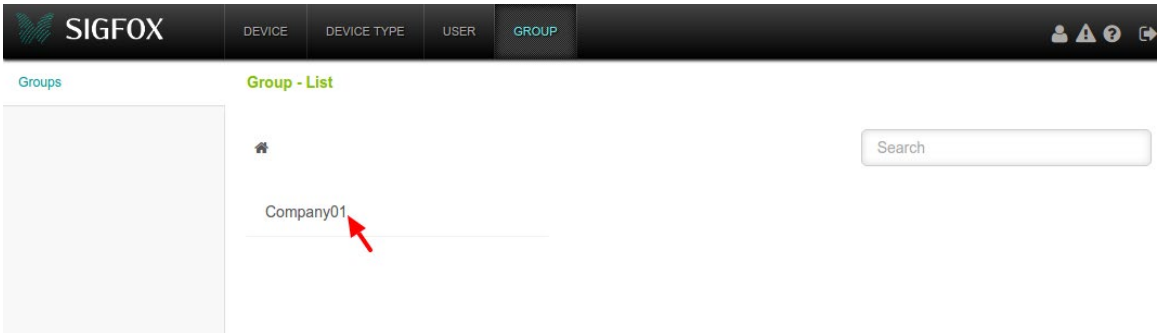


Figure: Sigfox groups page

4.6.1. Information

Shows all information of the group selected previously.

Figure: Sigfox information group page

4.6.2. Associated users

Shows a users list (same as User section) with all users associated to this group.

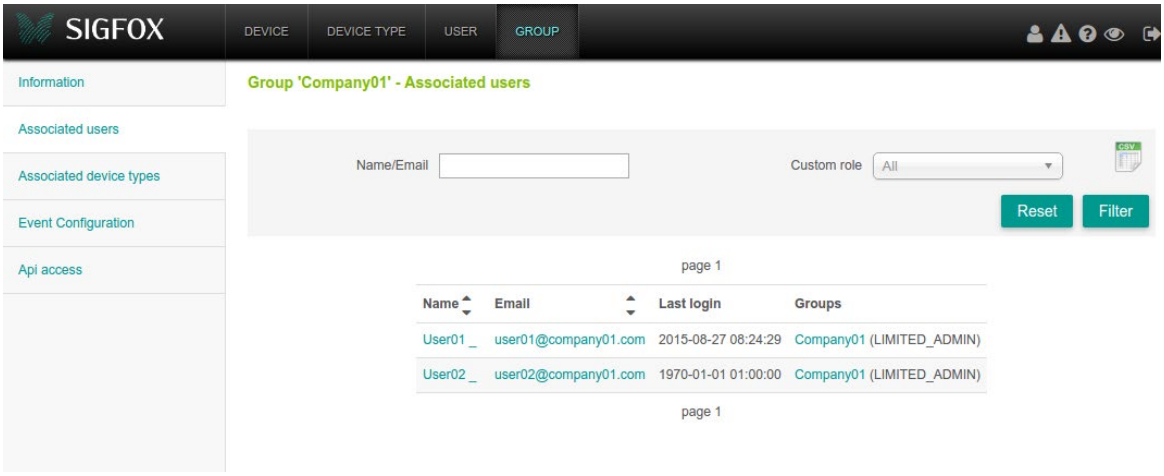


Figure: Sigfox group users associated page

4.6.3. Associated device types

Shows a device types list (same as Device type section) with all device types associated to this group.

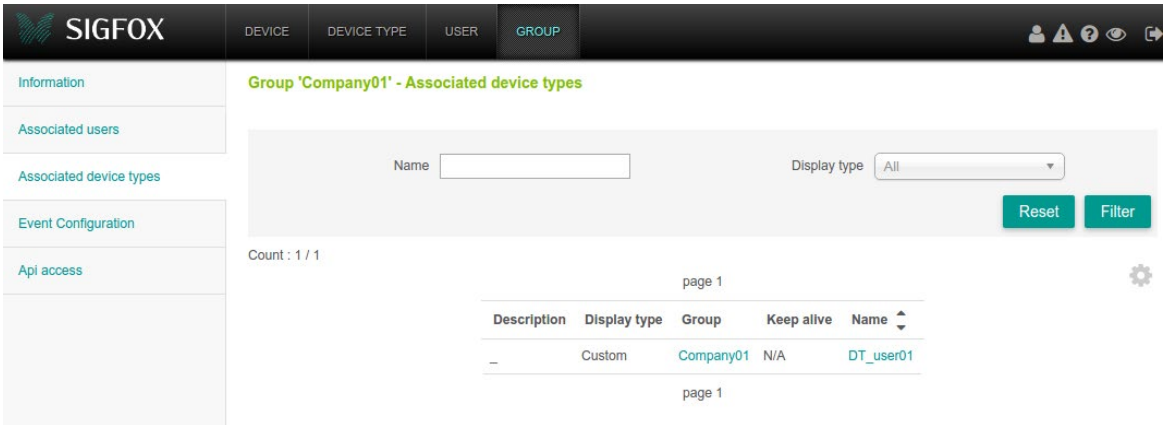


Figure: Sigfox group device types associated page

4.6.4. Event configuration

You will see information similar to the shown in the Device event configuration section, but in this case depending on your group type.

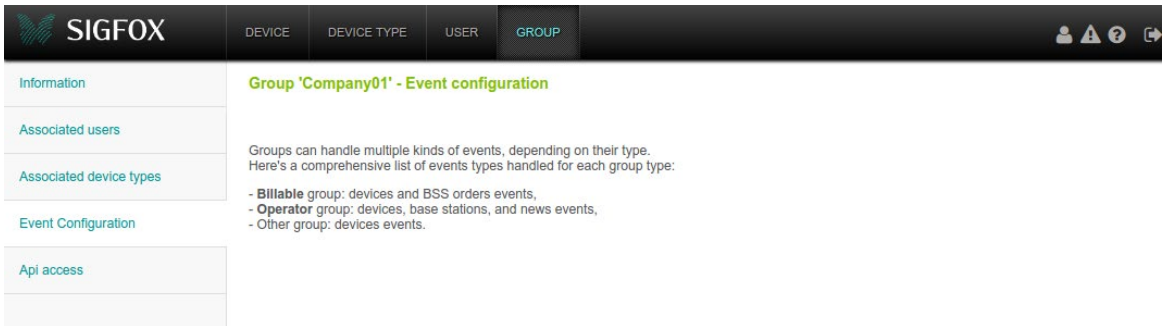


Figure: Sigfox group device types associated page

4.7. How to extract the data from the Sigfox servers

4.7.1. Callbacks

The Back-End can automatically forward some events using the «callback» system. The “Callbacks” button in the “Device Type” section permits to create new callbacks associated to that device type. The callbacks are triggered when a new device message is received or when a device communication loss has been detected.

There are three different callback types:

- DATA
- SERVICE
- ERROR

There is a set of available variables for each type of callback. These variables are replaced by their value when a callback is called.

Firstly, in this page you will see all the callbacks you configured. If you did not configure anything yet, a link with the Sigfox callback documentation link is printed. Anyway, you can access to this manual on this URL: https://backend.sigfox.com/apidocs/callback (you must be logged in the Back-End).

Here we show a callback example: once Sigfox Back-End receives one frame, it will execute the callback which is in charge of writing all data in a file of one server. Furthermore, it will send an ACK frame to the sender to verify the information has been received correctly and it will send data to the device. To create this callback you have to hit on “New” button and fill the following form:

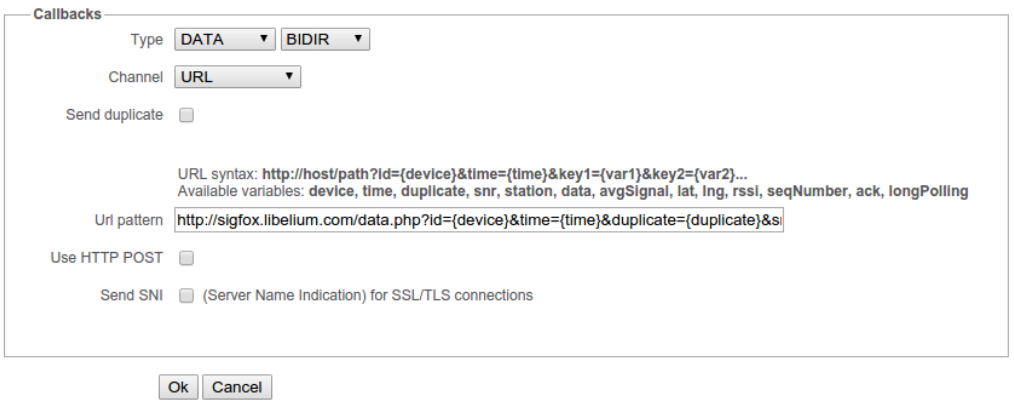


Figure: Sigfox device type callbacks form

The URL pattern would be something like this:

http://YOUR_DOMAIN/data.php?id={device}&time={time}&duplicate={duplicate}&snr={snr}&station={station}&data={data}&avgSignal={avgSignal}&lat={lat}&lng={lng}&rssi={rssi}&seqNumber={seqNumber}&ack=true

In this case, it is necessary to activate the downlink data in order to send information to the device from the created script. So once the callback is saved, we are able to activate it just clicking on the circle located near the downlink table header. It will be marked, indicating it is active.

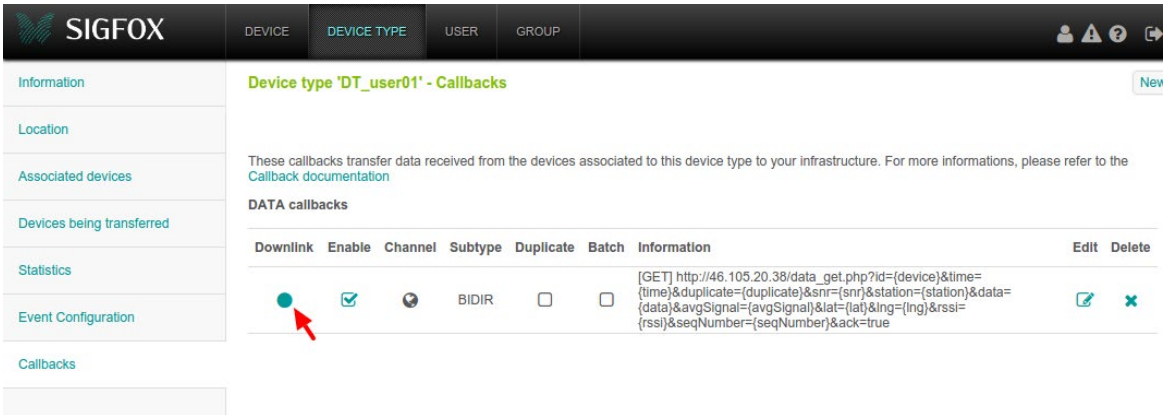


Figure: Sigfox downlink activation

Then in our server we need to create the data.php file, in order to get all data sent from the callback and save it in the final file.

```
<?php
    $data['data'] = array(
  "id"          => $_GET["id"],
        "time"        => $_GET["time"],
        "duplicate"   => $_GET["duplicate"],
        "snr"         => $_GET["snr"],
        "station"     => $_GET["station"],
        "data"        => $_GET["data"],
        "avgSignal"   => $_GET["avgSignal"],
        "lat"         => $_GET["lat"],
        "lng"         => $_GET["lng"],
        "rssi"        => $_GET["rssi"],
  "seqNumber"   => $_GET["seqNumber"],
      );

      if ($fd = fopen('YOUR_PATH/sigfox.json','a')){
        fwrite($fd,json_encode($data));
        fclose($fd);
      }

      header('Content-Type: application/json');
      $downlink= (array(
                          $_GET["id"] => (array('downlinkData' => '0102030405060708'))
/*downlink data must be 8 bytes in hexadecimal format*/
                  ));
      echo json_encode($downlink);
?>
```

Whenever the callback is executed, the PHP will save in the file a JSON string like this:

{"data":{"id":"10186","time":"1440687059","duplicate":"false","snr":"17.66","station":"0CD2","data":"41ea000064014c00daff2b00","avgSignal":"22.65","lat":"42","lng":"-1","rssi":"-129.90","seqNumber":"259"}}

At the same time, in the device messages list, we can find the information received, the callback status, the downlink status and content, and if the ACK was received.

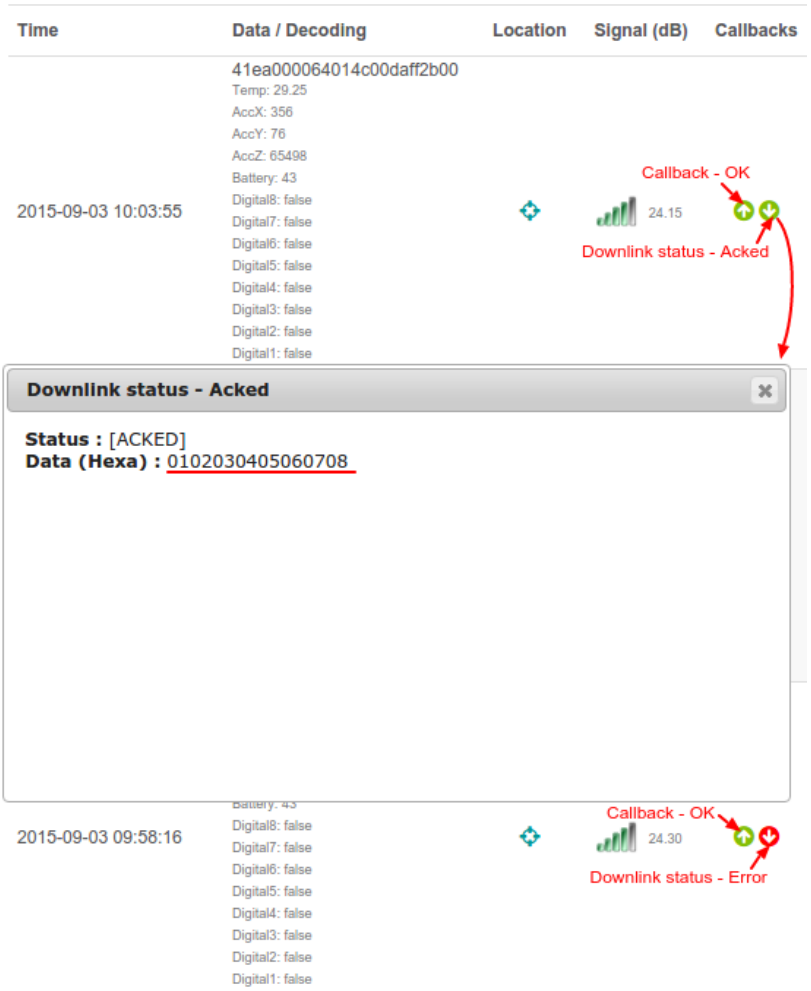


Figure: Sigfox device type callbacks and downlinks

And finally in our device, we will receive the downlink data sent from the callback process.

```
+RX BEGIN
+RX=01 02 03 04 05 06 07 08
+RX END
```

Figure: Sigfox downlink received

4.7.2. API access

Some of the features found on the Sigfox Back-End website can also be accessed programmatically using a webservice API. This API uses the HTTP protocol, following the REST principles.

All API endpoints return data in the JSON format, with the corresponding «application/json» content type header.

Creating an API access is very simple, just click on the “New” button, and fill the form with a name to identify this API access and a role picked from the list.

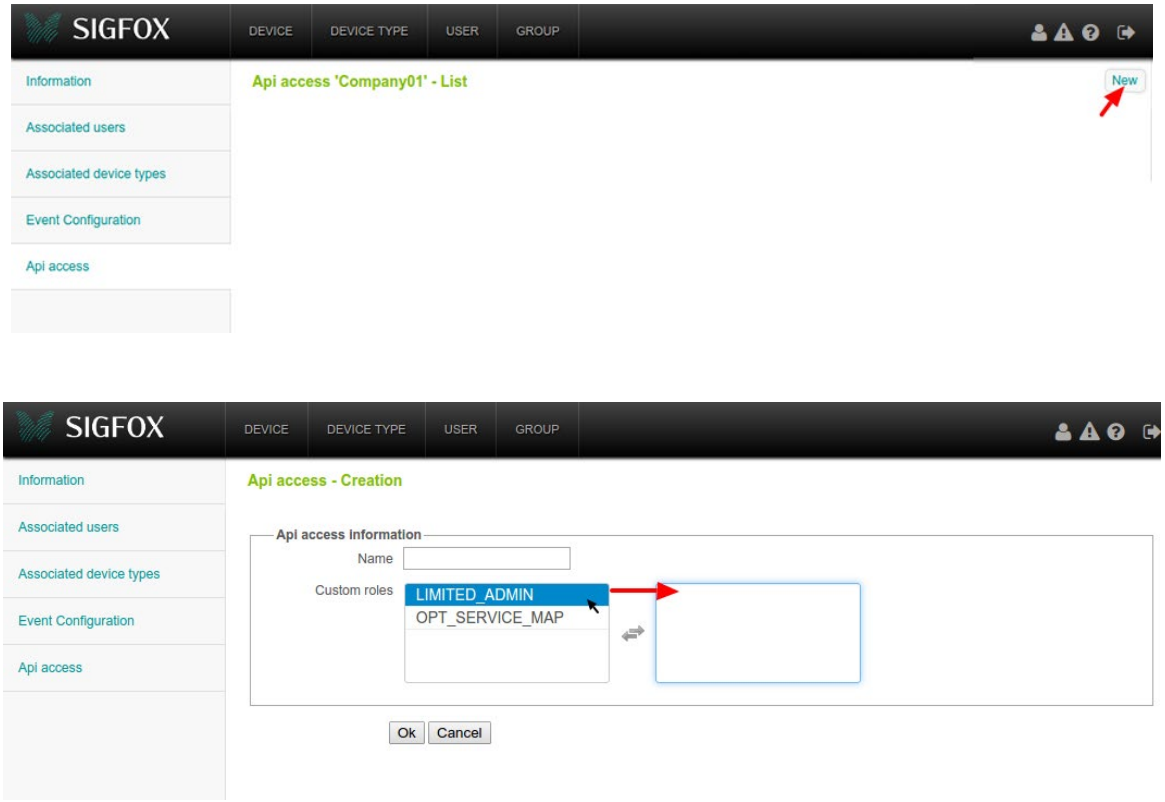


Figure: Sigfox group API creation process

Once it is saved, you will find your user and password to connect to the API service and a link to the API documentation with instructions and examples which are very helpful to create your application.

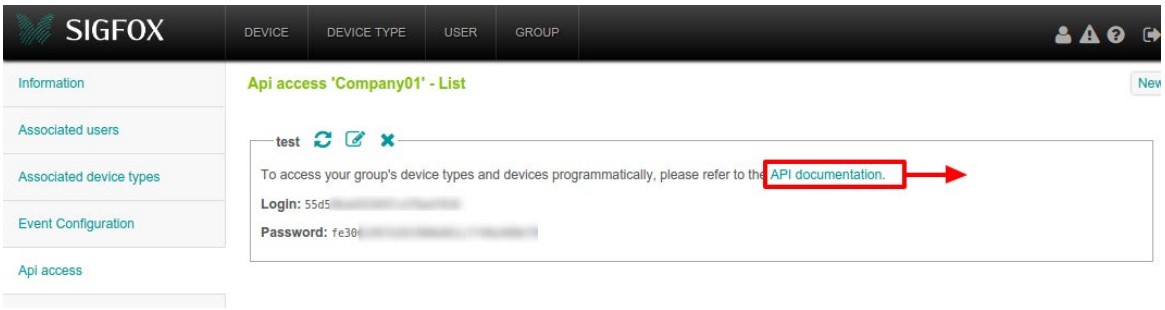


Figure: Sigfox group API access page

An example of this functionality could be getting a list of all messages received by a device. First of all, read and understand the General protocol information you can find in the documentation. This is highly recommended because it explains the functionality and a short example of the API. Then, we find the section Messages sent by a device, needed to develop our example and where we can see the URL to obtain the information we want to use. We are going to use PHP to do the example, using CURL to get the information.

```
<?php

$user = “##YOUR_USER##”;
$password = “##YOUR_PASSWORD##”;
$url = “https://backend.sigfox.com/api/devices/##YOUR_DEVICE_ID##/messages”;

$ch = curl_init();
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, true);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_URL,$url);
curl_setopt($ch, CURLOPT_USERPWD, “$user:$password”);
$result=curl_exec($ch);
curl_close($ch);

$data = (json_decode($result, true));

?>

<html>
  <table>
  <thead>
        <tr>
              <th>device</th>
              <th>time</th>
              <th>data</th>
              <th>snr</th>
        </tr>
  </thead>
  <tbody>
        <?php
              foreach($data[‘data’] as $reg){
        ?>
              <tr>
                    <td><?php echo $reg[‘device’]?></td>
                    <td><?php echo $reg[‘time’]?></td>
                    <td><?php echo $reg[‘data’]?></td>
                    <td><?php echo $reg[‘snr’]?></td>
              </tr>
        <?php
              }
        ?>
  </tbody>
</html>
```

In this example, you have to replace the three variables at the top of the code with your API credentials ($user and $password) and the device ID you want to get the messages ($url). Then a table is printed on your web browser with the messages.

device	time	data	snr
1B9C5	1441211241	41e0000066013200eeff4d00	23.66
1B9C5	1441207554	000102030405060708090a0b	24.23
1B9C5	1441207440	0011223344aabb	23.69
1B9C5	1441193326	000102030405060708090a0b	23.65
1B9C5	1441023751	010203040506070809	24.41

5. When is Sigfox recommended?

Sigfox is a protocol with a good long-range performance. However, it takes long to send a single packet (from 6 to 12 seconds) and has a limitation of 140 packets per day with a payload of 12 bytes due to ETSI regulations for the 868MHz band in Europe. So:

- Sigfox is not advised for projects with a regular duty-cycle, which require sending one frame every few minutes. Exceeding the 140 packets per day limit may lead to the charge of extra fees or the cancellation of the license by Sigfox. Read their contract policy first.
- Sigfox is recommended for long-range device communications in cities where their base stations are deployed.
- Sigfox is NOT recommended for bi-directional communications. It does not exist a real data downlink. Although, packet acknowledgements and callbacks are provided.
- Sigfox is NOT recommended for real-time streaming. Transmission is not done in real time as there is a minimum delay for packet arrival.
- Sigfox is NOT recommended for large amount of data transmissions. Maximum payload is 12 bytes.

6. 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! Sigfox EU" is certified for:

- CE (Europe)

"Plug & Sense! Sigfox US" is certified for:

- FCC (US)
- IC (Canada)

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

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

7. Code examples and extended information

In the Waspmote Development section you can find complete examples:
www.libelium.com/development/waspmote/examples

Example:

```
#include <WaspSigfox.h>

uint8_t socket = SOCKET0;
uint8_t error;

void setup()
{
  USB.ON();
  USB.println(F("Sigfox - Sending example"));
}

void loop()
{
  // 1. switch on
  error = Sigfox.ON(socket);

  // Check sending status
  if( error == 0 )
  {
    USB.println(F("Switch ON OK"));
  }
  else
  {
    USB.println(F("Switch ON ERROR"));
  }


  // 2. send data

  USB.println(F("Sending a packet..."));

  // Send 12 bytes at most
  error = Sigfox.send("00010203040506070809 0A0B");

  // Check sending status
  if( error == 0 )
  {
    USB.println(F("Sigfox packet sent OK"));
  }
  else
  {
    USB.println(F("Sigfox packet sent ERROR"));
  }


  // 3. sleep
  USB.println("\nEnter sleep");
  PWR.deepSleep("00:01:00:00",RTC_OFFSET,RTC_ALM1_MODE1,ALL_OFF);
  USB.println("\n**************************************");
}
```

8. API changelog

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

9. Documentation changelog

From v7.2 to v7.3

- Added references to the 3rd version of the Sigfox radio: Sigfox APAC / LATAM module
- Updated sections for Sigfox APAC / LATAM
- Updated table of supported countries and regions
- New function added to configure the working region

From v7.1 to v7.2

- Changed the official names from 868 to EU and 900 to US
- Updated list of countries with Sigfox service
- Added “Region standards” section to explain differences between Sigfox zones and restrictions of use

From v7.0 to v7.1

- Added a new section to show the user how to connect the module to Waspmote