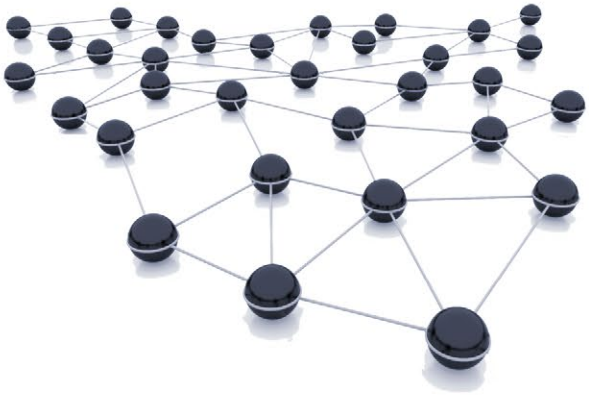
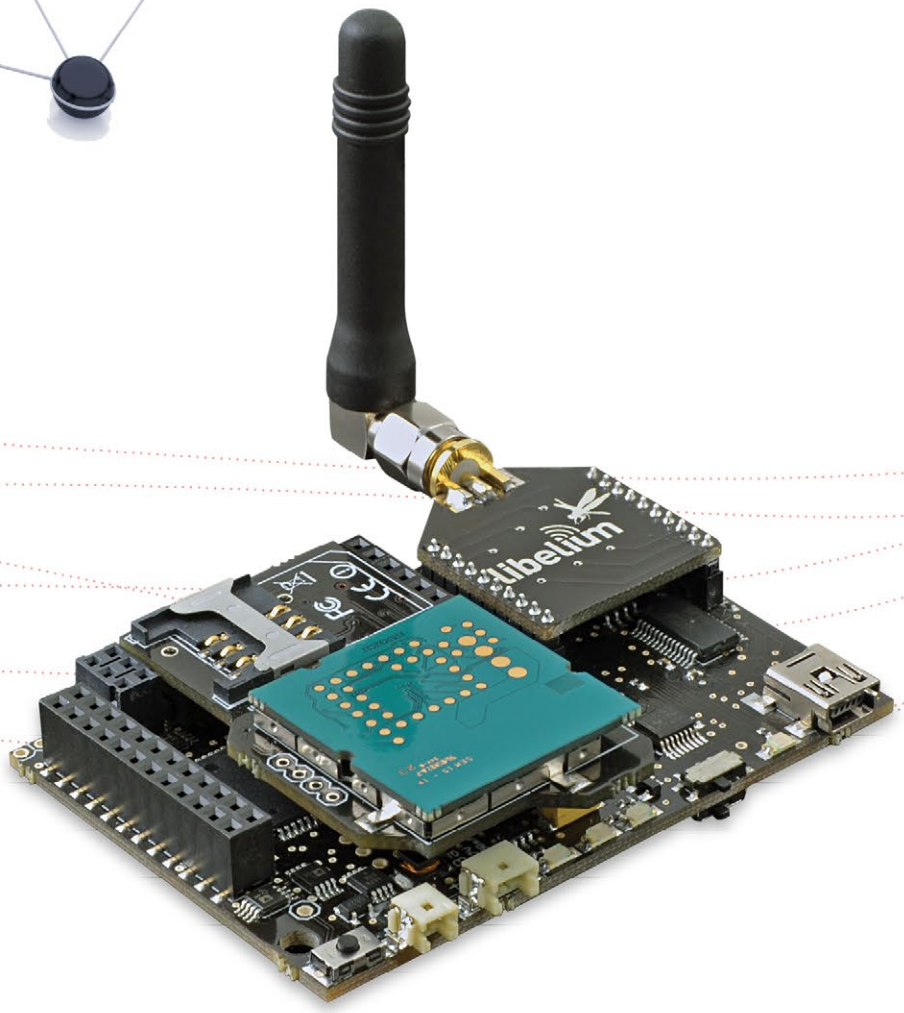
RS-232 Module

Communication Guide





libelium

waspmote

Document version: v4.4 - 01/2016

INDEX

1. Introduction

1.1. The standard

The RS-232 standard was first introduced in 1962 by the Radio Sector of the EIA. The original DTEs (data terminal equipment) were electromechanical teletypewriters, and the original DCEs (data circuit-terminating equipment) were usually modems. When electronic terminals began to be used, they were often designed to be interchangeable with teletypewriters, and so supported RS-232. The C revision of the standard was issued in 1969 in part to accommodate the electrical characteristics of these devices. For many years, an RS-232-compatible port was a standard feature for serial communications, such as modem connections, on many computers. The RS-232 standard is still used to connect industrial equipment (such as PLCs), console ports and special purpose equipment.

The IEEE RS-232 standard defines electrical, signal timing, and size connectors. Use of a common ground, limits RS-232 to applications with relatively short cables. RS-232 connection consisting only of transmit data, receive data, and ground. RS-232 protocol uses bipolar signal. Valid signals are ±3 to +15 volts, the ±3V range is not a valid RS-232 level. Data signals between -3V and -15V represents a logic 1. The logic 0 is represented by a voltage of between +3V and +15V.

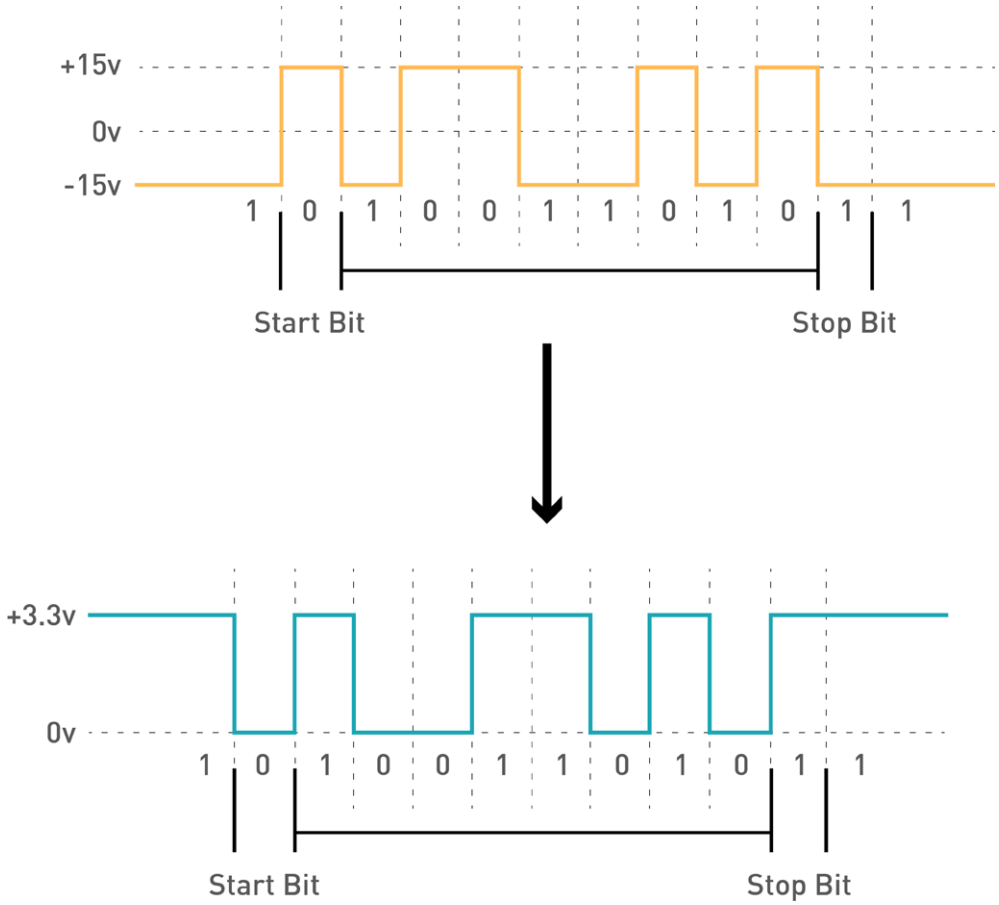


Figure : RS-232 voltage conversion

Serial transmission requires synchronization. A logic 0 is sent as a start bit for the synchronization, followed by normally eight bits. After the data itself, a parity bit is sent. It is optional and it can be chosen to have even or odd parity. Finally a stop bit is sent. This is normally one bit long and is used to signify the end of a particular byte. Sometimes two stop bits are required. This is an option that can be set on some equipments.

This list includes some of the more common uses of the standard:

- Dial-up modems
- GPS receivers (typically NMEA 0183 at 4,800 bit/s)
- Bar code scanners and other point of sale devices
- LED and LCD text displays
- Satellite phones, low-speed satellite modems and other satellite based transceiver devices
- Flat-screen (LCD and Plasma) monitors to control screen functions by external computer, other AV
- components or remotes
- Test and measuring equipment such as digital multimeters and weighing systems
- Updating Firmware on various consumer devices
- Some CNC controllers
- Uninterruptible power supply
- Stenography or Stenotype machines
- Software debuggers that run on a 2nd computer
- Industrial field buses

2. Hardware

The RS-232 / Modbus module has been tested with various devices and is compatible with the majority of commercial modules, but this does not ensure the working with all of them. Be sure that the RS-232 module fits your technical requirements. The final user is the responsable to perform the task of communicatiing the RS-232 module with other commercial devices.

2.1. Electrical features

- **Board power voltages:** 3.3 V
- **Maximum admitted voltage:** -0.3 V to +6 V
- **Typical data rate:** 115200 bps
- **Temperature Range:** [0 ºC, 70 ºC]
- **Dimensions:** 33 x 31.5 mm

2.2. Connection diagram

The RS-232 Serial / Modbus module uses the UART pins, RX and TX, for communication. The Expansion Board allows to connect two boards at the same time in the Waspmote sensor platform. This means a lot of different combinations are now possible using any of the radios available for Waspmote (802.15.4, ZigBee, 868, 900, LoRaWAN, LoRa, Sigfox, Bluetooth Pro/Low Energy, NFC/RFID, WiFi and GPRS/3G) and the RS-232 module.

In the next photo you can see the available sockets along with the UART assigned. The RS-232 module can be connected normally on socket 0, and in socket 1 with the Expansion Board.

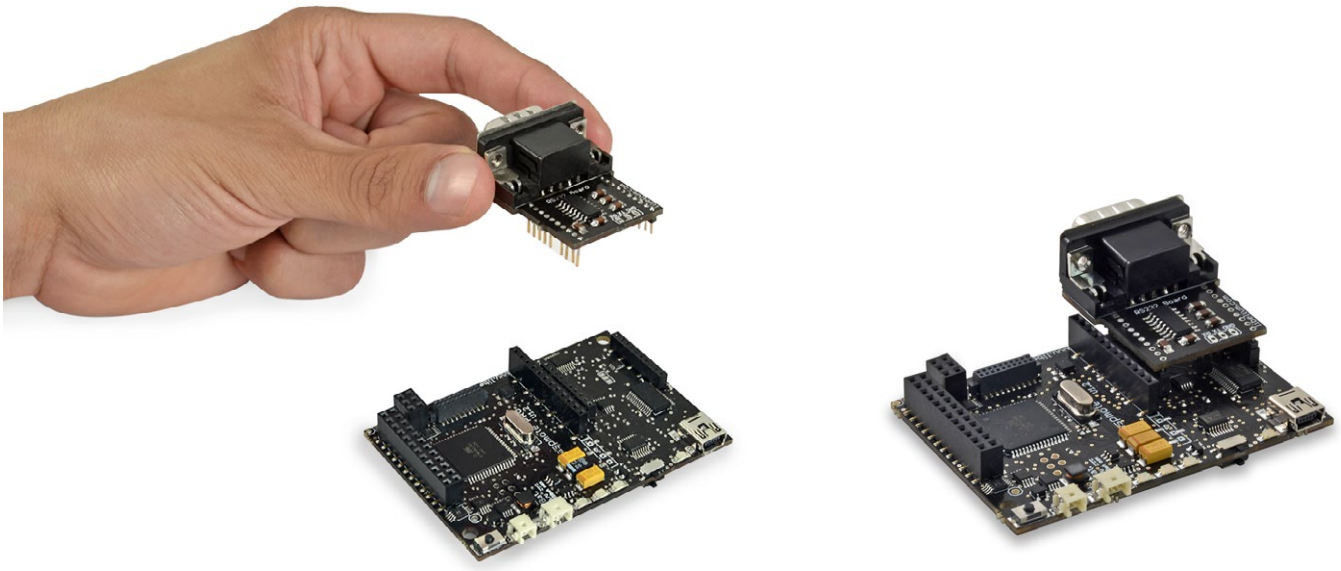


Figure : RS-232 in socket0

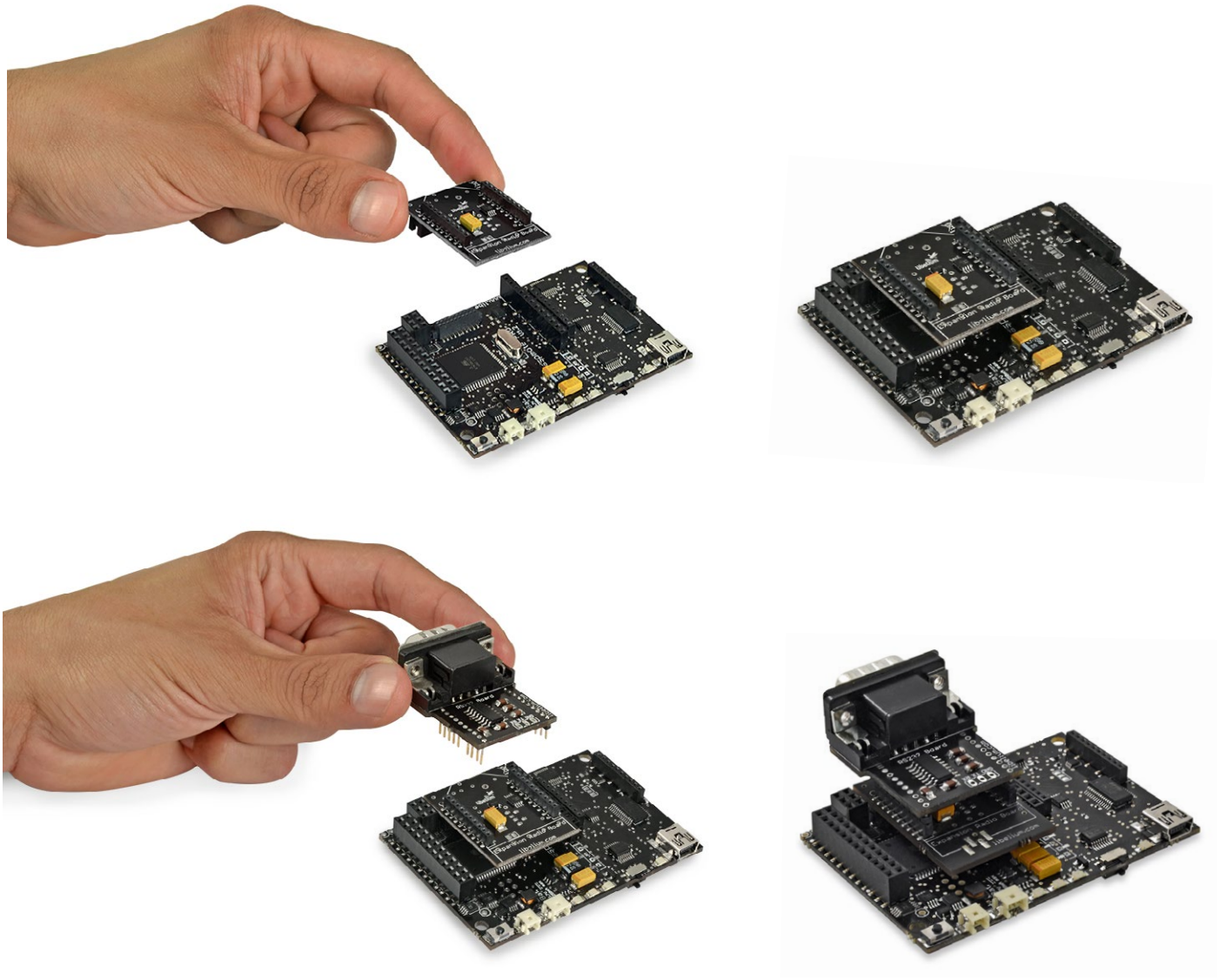


Figure : RS-232 in socket1

The RS-232 Serial / Modbus module can be used with two different protocols:

1. RS-232 Serial standard (this is the scope of this guide)
2. Modbus protocol (which adds some features; see the Modbus Communication Guide for more details)

2.3. Consumption

RS-232 module uses a low power transceiver. The board is guaranteed to run at data rates of 120 kbps while maintaining RS-232 output levels. The next table shows the consumption at various baud rates.

Baud Rate (bps)	Consumption (mA)
300	1,65
600	1,65
1200	1,65
2400	1,67
4800	1,67
9600	1,72
19200	1,8
38400	1,9
57600	2,0
115200	2,26

Figure : RS-232 consumption table

2.4. Connector

The standard recommends but does not make mandatory the D25-pin connector. In the RS-232 module, the connector is a DB9 female. The RS-232 module uses pin 2 to receive data, 3 to transmit and ground pin. The DB9 connector is used in many applications; for example, any PC has a serial DB9 connector. It provides size and cost benefits. Also the RS-232 9-pin configuration is sufficient in most circumstances, because many of the lines available for RS-232 signaling are rarely used. This means that the DB9 connector is able to provide all the required connectivity for most applications.

Figure : DB9 connector

The RS-232 module comes with a standard male-female DB9 cable. This cable is useful for connecting the module to other RS-232 devices which have a DB9 male connector.

Figure : Male-female DB9 cable

3. Dual Radio with Expansion Board

3.1. Expansion Radio Board

The RS-232 module can use the 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, 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 Waspmote. They can be plugged directly in the socket1.

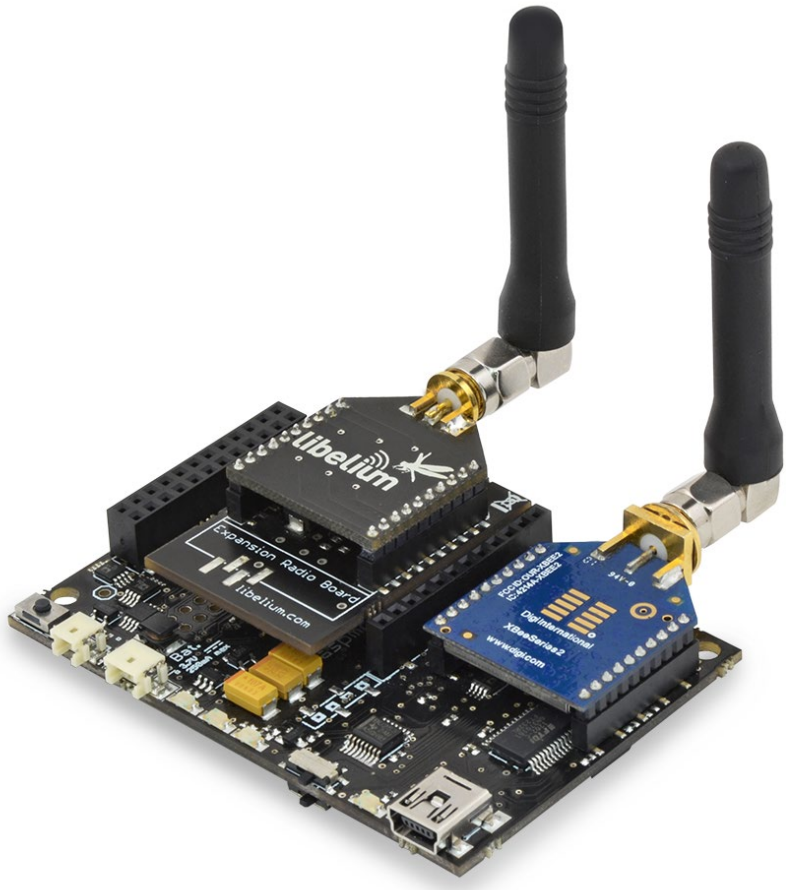


Figure : Waspmote with XBee radio on socket 0 and WiFi module on socket 1

The API provides a function to initialize the RS-232 module, it is called `ON(socket)`. This function supports a new parameter which permits to select the socket. It is possible to choose between socket0 and socket1.

An example of use the initialization function is the following:

Selecting socket0: `W232.ON(SOCKET0);`

Selecting socket1: `W232.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 SOCKET3B and NO_2 sensor.
- Agriculture Board: Incompatible with Sensirion and the atmospheric pressure sensor.
- Smart Metering Board: Incompatible with SOCKET2, SOCKET3, SOCKET4 and SOCKET5.
- Smart Cities Board: Incompatible with microphone and the CLK of the interruption shift register.
- Events Board: Incompatible with interruption shift register.

4. Applications

This module allows the user to interface the Waspmote ecosystem with RS-232 systems. Waspmote allows to perform three main applications:

1º- Connect any sensor to an existing RS-232 device/network

Waspmote can be configured to work as a node in the network, inserting sensor data into the RS-232 bus already present. Waspmote can obtain information from more than 70 sensors which are currently integrated in the platform by using specific sensor boards (e.g: CO, CO_2, temperature, humidity, acceleration, pH, IR, luminosity, etc). This way, the sensor information can be read from any RS-232 device connected to the bus.

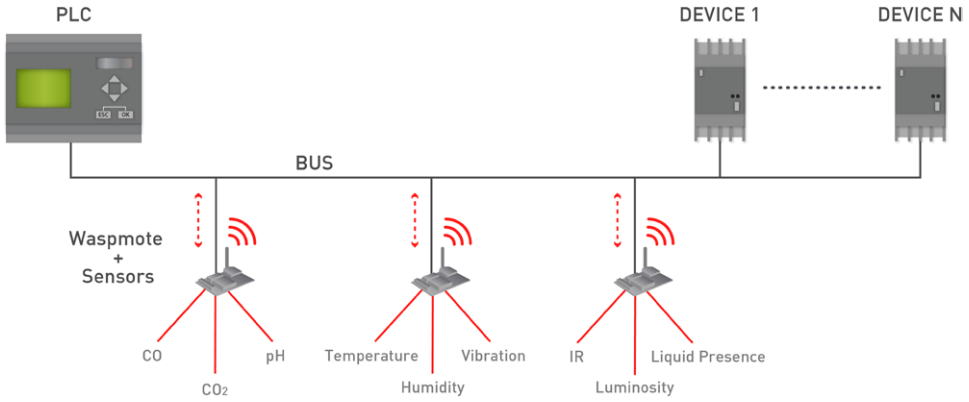


Figure : RS-232 in wireless sensor network applications

2º- Add wireless connectivity to RS-232 devices

Waspmote can be configured to read the information coming from the RS-232 bus and send it wirelessly using any of the wireless modules available in the platform to a base station or even directly to a Cloud server. The available wireless technologies are: WiFi, 3G, GPRS, 802.15.4, ZigBee, LoRaWAN, LoRa, Sigfox, Bluetooth, Bluetooth Low Energy, RF-868MHz, RF-900MHz.

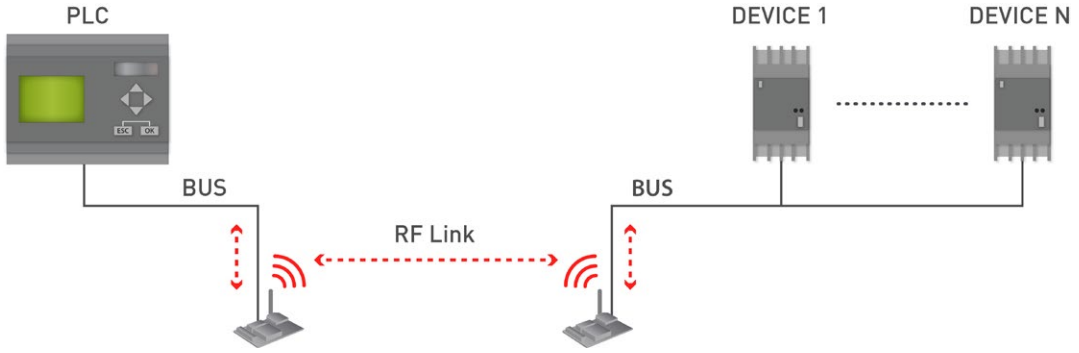


Figure : RS-232 wire replacement

3º- Connect to the Cloud RS-232 devices

Waspmote can be configured to read the information coming from the RS-232 bus and send it wirelessly directly to the Cloud using WiFi, 3G and GPRS radio interfaces.

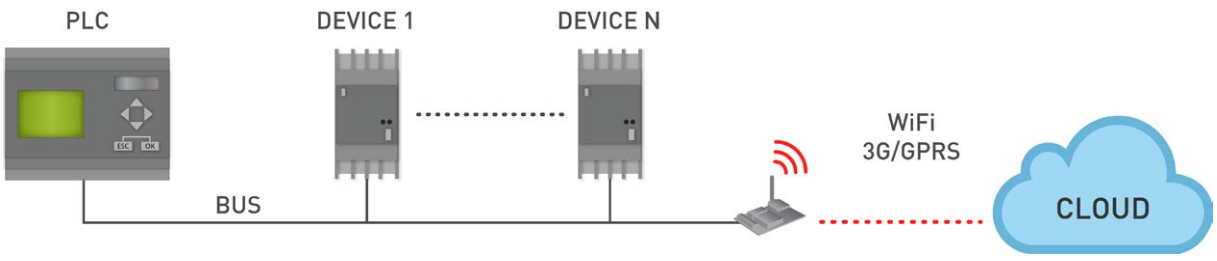


Figure : Cloud connection

5. Libelium's API

It is mandatory to include the RS-232 library when using this module. The following line must be introduced at the beginning of the code:

```
#include <Wasp232.h>
```

Waspmote's API RS-232 files:

- Wasp232.cpp
- Wasp232.h

APIs functions

- Private functions:

The following functions are executed inside the API functions. In normal conditions, the user must NOT manage or use them.

`print(char c, uint8_t uart);`	Sends the data through the selected UART
`printNumber(unsigned long n, uint8_t base, uint8_t uart);`	Sends the data with a base through the selected UART

Figure : RS-232 private functions

- Public functions:

`ON(char socket);`	Powers the 232 module and opens the UART
`OFF(void);`	Switches off the module and closes the UART
`baudRateConfig(unsigned long speed);`	Sets the speed of asynchronous communication
`char read(void);`	Receives data through the UART
`send(n);`	Sends data through the UART
`int available(void);`	Gets the number of bytes (characters) available for reading
`flush(void);`	Flushes the buffer of incoming serial data
`parityBit(bool state);`	Enables or disables the parity bit
`stopBitConfig(uint8_t numStopBits);`	Selects the number of stop bits to be inserted by the transmitter
`uint8_t error(void);`	Detects a transmission error

Figure : RS-232 public functions

6. Library functions

6.1. Library constructor

To start using Waspmote RS-232 library, an object from class 'Wasp232' must be created. This object, called `W232`, is created inside Waspmote RS-232 library and it is public to all libraries. It is used through this guide to show how Waspmote RS-232 library works. When creating this constructor, all the variables are defined with an initial value by default.

6.2. Switching the module ON

The `ON()` function powers the RS-232 module and assigns the UART. The default baud rate is 115200 and it can be modified by the function `baudRateConfig()`.

Example of use:

```
{
  W232.ON(SOCKET0);
}
```

The same example but in the socket1:

```
{
  W232.ON(SOCKET1);
}
```

See an example of use here:

http://www.libelium.com/development/waspmote/examples/rs-232-01-send-data

6.3. Switching the module OFF

Switches off the RS-232 module and stops sending data frames. When you switch the module off, it is not necessary to pass the socket as parameter. This information is already stored in an internal variable when you switch the module on.

Example of use:

```
{
  W232.OFF();
}
```

6.4. Configuring communication speed

RS-232 speed communication can be configured with the values of the RS-232 standard. The RS-232 module has been tested at 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600 and 115200 bps. The maximum supported datarate in the RS-232 module is 115200 bps.

Example of use:

```
// Include always this library when you are using the Wasp232 functions
#include <Wasp232.h>

void setup()
{
  // Power on the USB for viewing data in the serial monitor
  USB.ON();
  delay(100);

  // Powers on the module and assigns the UART in socket1
  W232.ON(SOCKET1);
  delay(100);

  // Configure the baud rate of the module
  W232.baudRateConfig(9600);
}
```

See an example of use here:

http://www.libelium.com/development/waspmote/examples/rs-232-01-send-data

6.5. Configuring the number of stop bits

The most common configuration used between computers is 8N1: eight bit characters, with one stop bit and no parity bit. With the RS-232 module you can configure the communication mode with one or two stop bits using the `stopBitConfig()` function.

Example of use with one stop bit:

```
{
  W232.ON(SOCKET0);
  W232.baudRateConfig(9600);
  W232.stopBitConfig(1);
}
```

Example of use with two stop bits:

```
{
  W232.ON(SOCKET0);
  W232.baudRateConfig(9600);
  W232.stopBitConfig(2);
}
```

See an example of use here:

http://www.libelium.com/development/waspmote/examples/rs-232-02-receive-data

6.6. Configuring the parity bit

Parity is used in many hardware applications to detect frame errors and is usually generated and checked by the interface hardware. The RS-232 module uses:

- `NONE`: No parity bit
- `EVEN`: Even parity
- `ODD`: Odd parity

Parity can be enabled or disabled depending on communication requirements. For that purpose, the `parityBit()` function permits to change this parameter.

Example of use with no parity:

```
{
  W232.ON(SOCKET0);
  W232.baudRateConfig(9600);
  W232.parityBit(NONE);
  W232.stopBitConfig(1);
}
```

Example of use with even parity enabled:

```
{
  W232.ON(SOCKET0);
  W232.baudRateConfig(9600);
  W232.parityBit(EVEN);
  W232.stopBitConfig(1);
}
```

Example of use with odd parity enabled:

```
{
  W232.ON(SOCKET0);
  W232.baudRateConfig(9600);
  W232.parityBit(ODD);
  W232.stopBitConfig(1);
}
```

See an example of use here:

http://www.libelium.com/development/waspmote/examples/rs-232-02-receive-data

6.7. Sending data

6.7.1. Sending a char

The most common way to send a character is to send it between single quotes (e.g. 'p').

Example of use:

```
{
  char data = 'p';
  W232.send(data);
}
```

You can also send char variables by declaring the corresponding ASCII code. For example, if you initialize a char with "123", it sends the corresponding ASCII "{".

Example of use:

```
{
  char data = 123;
  W232.send(data);
}
```

6.7.2. Sending an integer

You can send an unsigned or signed integer using the same function. If an unsigned int is declared you can assign a value from 0 to 65535, and if the variable is signed the value can be between -32768 and 32767.

Examples of use:

```
{
  unsigned int data = 12345;
  W232.send(data);
}

{
  int data = -12345;
  W232.send(data);
}
```

6.7.3. Sending a string

The same function can be used for sending string of characters.

Example of use:

```
{
  W232.send("Hello world");
}
```

See an example of use here:

http://www.libelium.com/development/waspmote/examples/rs-232-01-send-data

6.7.4. Sending with base

The `send()` function allows to represent the variable in a specific base. You can select binary, octal, byte, decimal, and hexadecimal representation.

Example of use:

```
{
  int data = 12345;

  // Send 12345 in binary base. It prints 0110 000 0011 1001.
  W232.send(data, BIN);

  // Send 12345 in octal base. It prints 30071.
  W232.send(data, OCT);

  // Send 12345 in decimal base. It prints 12345.
  W232.send(data, DEC);

  // Send 12345 in hexadecimal base. It prints 3039.
  W232.send(data, HEX);
}
```

6.7.5. Sending a long

Long variables can be sent also with the function `send()`. Long variables represents 32 bits or 4 bytes and and the `send()` function allows signed and unsigned declaration.

Example of use:

```
{
  unsigned long data = 1234567;
  W232.send(data);
}
```

6.8. Receiving data

All received bytes are stored in the UART buffer of the micro-controller. These bytes can be read with the function `read()`. This function returns the read value from the buffer.

Example of use:

```
{
  if (W232.available())
  {
        while (W232.available())
        {
              char data = W232.read();
              USB.print(data);
        }
  }
}
```

The `receive()` function permits to receive all information in the reception buffer so as to manage it. The contents are stored in a buffer that belongs to the 'WaspRS232' class. The buffer is called `_buffer` and the length of the buffer is stored in `_length`. The function returns the number of bytes stored in the buffer.

Example of use:

```
{
  if(W232.receive() > 0)
  {
        USB.println(W232._buffer, W232._length);
  }
}
```

See an example of use here:

http://www.libelium.com/development/waspmote/examples/rs-232-02-receive-data

6.9. Knowing when new data is available

The `available()` function returns the number of bytes available in the UART buffer.

```
{
 if (W232.available())
 {
        while (W232.available())
        {
               char data = W232.read();
               USB.print(data);
        }
 }
}
```

See an example of use here:

http://www.libelium.com/development/waspmote/examples/rs-232-02-receive-data

6.10. Flushing buffer

Flushes the buffer of incoming serial data. The function `flush()` waits for outgoing data to transmit before clearing the buffer contents.

Example of use:

```
{
 W232.flush();
}
```

7. Code examples and extended information

For more information about the Waspmote hardware platform go to:

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

In the Waspmote Development section you can find complete examples:

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

Example:

```
/*
 *  ------ [RS-232_01] RS-232 Send Data --------
 *
 *  This sketch shows how to send data through RS-232 standard.
 *  This standard defines the electrical characteristics of drivers
 *  and receivers for use in digital systems. It does not specify
 *  or recommend any communications protocol. For a complete
 *  communication protocol, please see the Modbus examples.
 *
 *  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 2 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:          0.2
 *  Design:           David Gascon
 *  Implementation:   Ahmad Saad
 */

//Include always this library when you are using the Wasp232 functions
#include <Wasp232.h>


void setup() {

  // Power on the USB for viewing data in the serial monitor
  // Note : if you are using the socket 0 for communication,
  // for viewing data in the serial monitor, you should open
  // the USB at the same speed
  USB.ON();
  delay(100);
```

```
  // Powers on the module and assigns the UART in socket0
  W232.ON(SOCKET0);
  delay(100);


  // Configure the baud rate of the module
  W232.baudRateConfig(115200);

  // Configure the parity bit as disabled
  W232.parityBit(NONE);

  // Use one stop bit configuration
  W232.stopBitConfig(1);

  delay(100);

  // Hello message
  USB.println("RS-232 serial communication properly initialized.");
  USB.println("Hello, this is RS-232 communication send example! ");

  delay(100);

}

void loop() {

  // Reading the analog input 1
  int analog1 = analogRead(ANALOG1);
  // Reading the analog input 2
  int analog2 = analogRead(ANALOG2);

  // Send data through RS-232 line
  W232.send("Data from analog1 input: ");
  W232.send(analog1);
  W232.send("\n");

  W232.send("Data from analog2 input: ");
  W232.send(analog2);
  W232.send("\n");

  delay(2000);

}
```

8. API changelog

Keep track of the software changes on this link:

www.libelium.com/development/waspmote/documentation/changelog/#RS_232

9. Documentation changelog

From v4.3 to v4.4

- References to the new LoRaWAN module

From v4.2 to v4.3

- General improvements in document to clarify how to use the module
- References to the new Sigfox module

From v4.1 to v4.2

- References to the new LoRa module
- Created new chapter "API changelog"

From v4.0 to v4.1

- Expansion Radio Board section updated
- Reference to the standard male-female DB9 cable supplied