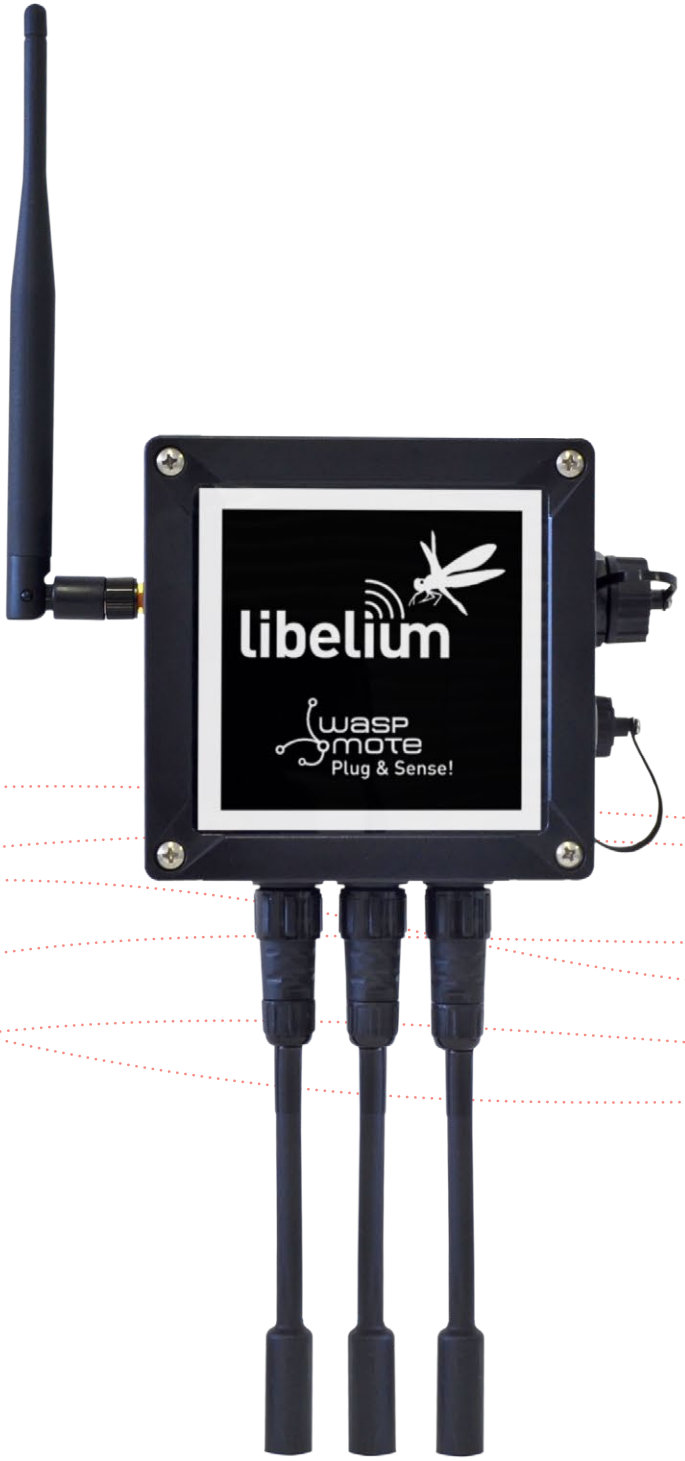
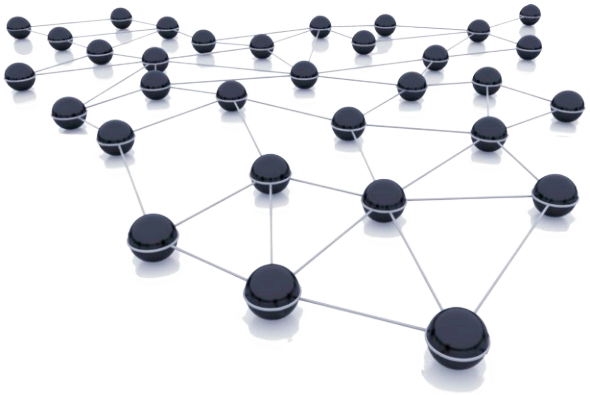
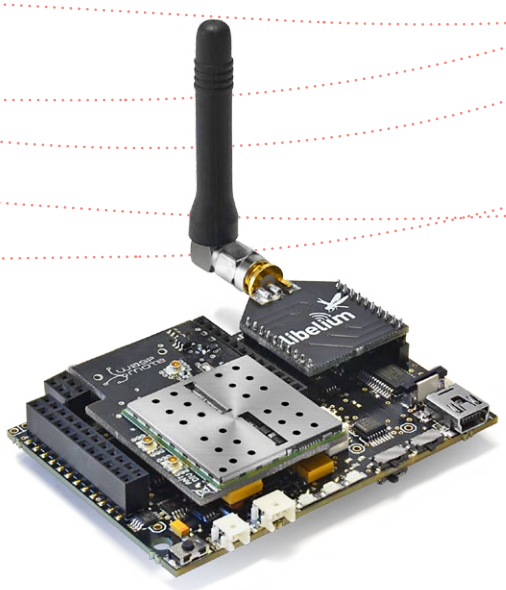
Bluetooth for device discovery

Networking Guide







libelium

wasp mote

Document Version: v7.0 - 02/2017

INDEX

1. Introduction 3
1.1. General description 3

2. Hardware 5
2.1. Specifications 5
2.2. Electrical characteristic and power consumption 5

3. Dual radio with the Expansion Board 6
3.1. Expansion Radio Board 6
3.2. Setting on 7
3.3. Setting off 7

4. General considerations 8
4.1. Waspmote libraries 8
4.1.1. Waspmote bluetooth files 8
4.1.2. Constructor 8
4.2. API functions 8
4.3. Waspmote reboot 9
4.4. Constants predefined 10

5. Bluetooth device parameters 11

6. Searching for devices 12

7. Connecting to other Bluetooth module 14
7.1. Pairing with devices 15
7.2. Using WaspFrame class to create sensor data frames 15

8. Code examples and extended information 16

9. Certifications 17

10. API Changelog 18

1. Introduction

This guide explains the Bluetooth module features and functions. This product was designed for Waspmote v12 and continues with no changes for Waspmote v15. There are no great variations in this library for our new product lines Waspmote 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".

This guide describes all features of the Libelium bluetooth module which has been mainly designed to discover up to 250 devices in a variable area. The module belongs to the Smart Cities solution created by Libelium, allowing applications like vehicle and pedestrian traffic monitoring.

Moreover, a dedicated API has been also created to manage inquiries of the bluetooth module.This API is designed only for discovery device purposes and basic data exchanges, leaving for the future other applications like data complex exchange in a Bluetooth network.

It has to be mentioned that inquiry processes of bluetooth module are anonymous due to only the MAC address is obtained from the bluetooth remote device. No account or phone numbers are obtained. This fact allows saving privacy of bluetooth users.

1.1. General description

The Bluetooth module has two main parts which are a previous designed module and an external antenna. The last one uses a RP-SMA connector in case it has to be replaced.

There are 7 different power levels which go from -27 dBm to 3 dBm in order to set different inquiry zones from 10 to 50 m. These zones can also be increased or decreased by using a different antenna for the module as it counts with an standard SMA connector. The user can choose between a 2 dB and a 5 dB antenna.

There are some parameters that can be inquired from devices inside the detection area. The most important ones are described below.

- **MAC address:** It is the unique identification number of the bluetooth device. It has 12 hexadecimal digits separated by ":". One example could be "12:34:56:aa:bb".

- **CoD (Class of Device):** Bluetooth devices are classified according to the device which they are integrated. Therefore, a vehicle hands free device will belong to a different class than a pedestrian mobile phone. This parameter has 6 hexadecimal digits and it allows to distinguish if the detected bluetooth device is a vehicle, a pedestrian, and so on.

- **RSSI (Received Signal Strength Indicator):** This parameter shows quality of the radio link. It can be used to know the distance between the bluetooth module and the inquired device. It is shown as a negative value between -40 dBm (close devices) and -90 dBm (far devices).

In addition, there is another parameter that can be inquired from bluetooth devices. This parameter is called "Friendly name" and it is defined by the owner of the bluetooth device. It is just a "friendlier" way to name a bluetooth device instead of the MAC address.

Note: *The bluetooth module requires an SD card to save all data of discovered devices. Please be sure that an SD card is inserted before using the module.*

Next figure shows a typical application for bluetooth modules where vehicles and pedestrians can be detected. Also, different detection areas are shown.

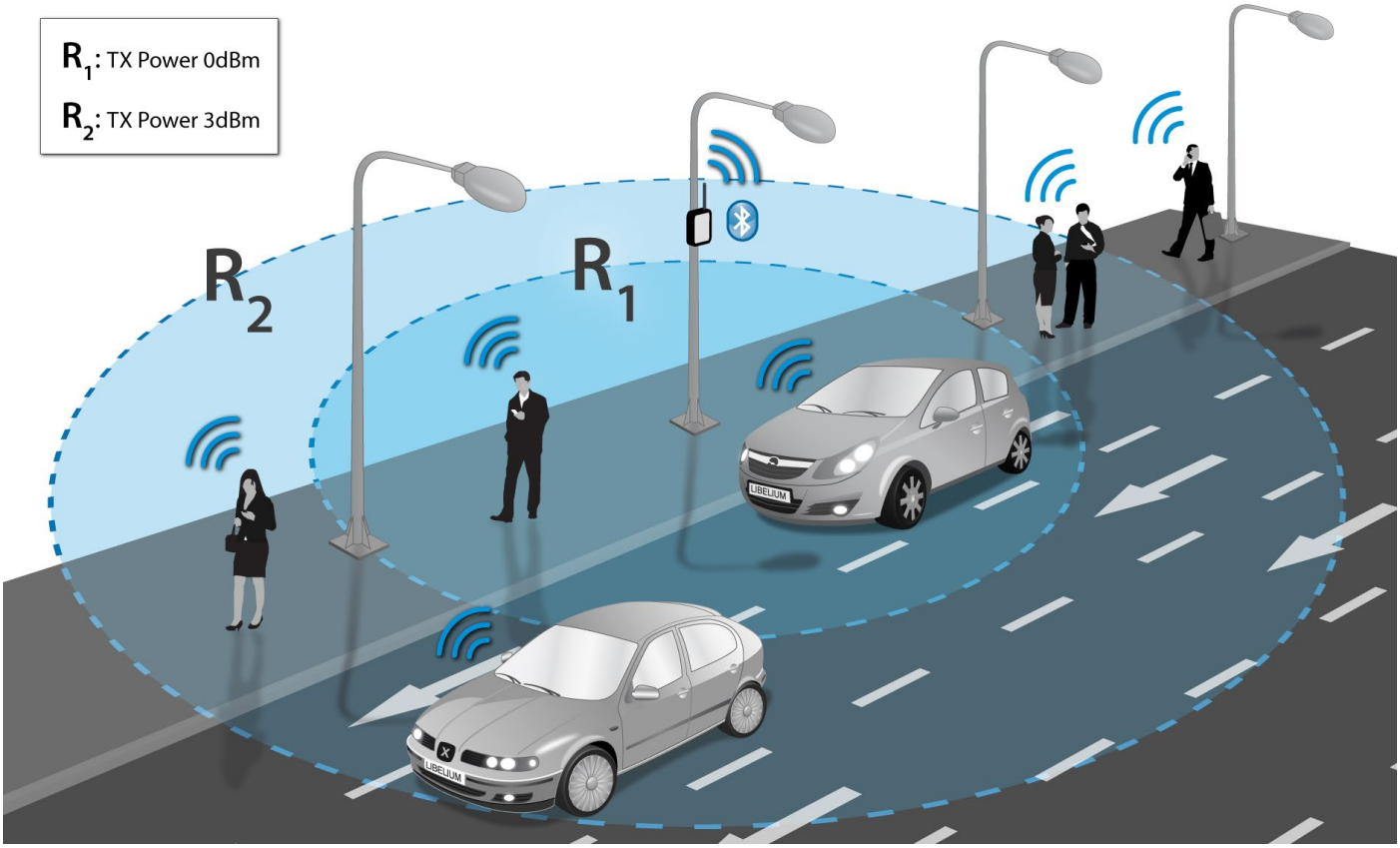


Figure : Example of TX power levels

How do the Bluetooth and ZigBee radios coexist without causing interferences with each other?

ZigBee and Bluetooth work in the 2.4 GHz frequency band (2.400 – 2.480 MHz). However, the Bluetooth radio integrated in Waspmote uses an algorithm called Adaptive Frequency Hopping (AFH) which improves the common algorithm used by Bluetooth (FHSS) and enables the Bluetooth radio to dynamically identify channels already in use by ZigBee and WiFi devices and to avoid them.

2. Hardware

2.1. Specifications

Main features of the bluetooth module are listed below:

- Bluetooth v2.1 + EDR. Class 2.
- TX Power: 3 dBm
- Antenna: 2 or 5 dBi
- Up to 250 unique devices in each inquiry
- Received Strength Signal Indicator (RSSI) for each scanned device
- Class of Device (CoD) for each scanned device
- 7 Power levels [-27 dBm, 3 dBm]
- Scan devices with maximum inquiry time
- Scan devices with maximum number of nodes
- Scan devices looking for a certain user by MAC address
- Classification between pedestrians and vehicles

Figure : Libelium bluetooth module

2.2. Electrical characteristic and power consumption

The Libelium Bluetooth module is powered from 3.3V. Next table shows average power consumption in different states of the modules.

Estate	Power consumption
Off	0
Sleep	<0.5 mA
On (Idle state)	2 mA
Inquiry at minimum power	33.5 mA
Inquiry at maximum power	36.5 mA

3. Dual radio with the Expansion Board

Before starting to use a module, it needs to be initialized. During this process, configuration parameters are sent to the module. USB and SD card are also initialized.

3.1. 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.*

Next image shows the sockets available along with the UART assigned. On one hand, SOCKET0 allows to plug any kind of radio module through the UART0. On the other hand, SOCKET1 permits to connect a radio module through the UART1.

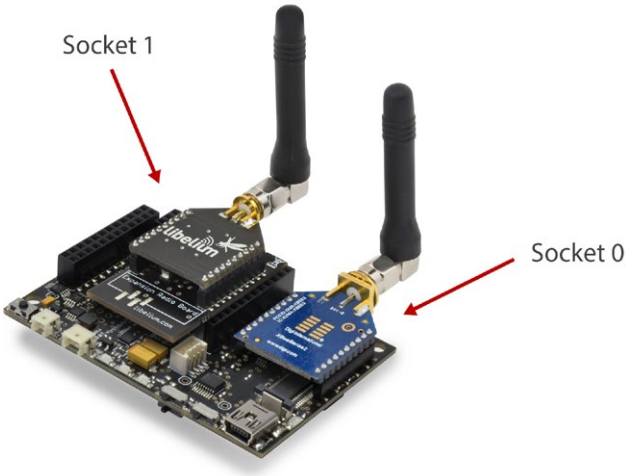


Figure : Waspmote with XBee radio on socket 0 and Bluetooth module on socket 1

This API provides a function in order to initialize the Bluetooth module called `BT_Pro.ON(socket)`. This function supports a new parameter which permits to select the socket. It is possible to choose between socket0 or socket1.

An example of use the initialization function is the following:

- Selecting socket0: `BT_Pro.ON(SOCKET0);`
- Selecting socket1: `BT_Pro.ON(SOCKET1);`

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

Warnings:

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

3.2. Setting on

With the function `ON()` module is powered and the UART is opened to communicate with the module. It also enters in command mode and sets some default configurations.

```
// switches ON the Bluetooth module using expansion board
BT_Pro.ON(SOCKET1);
```

3.3. Setting off

The Bluetooth API function `OFF()` closes the UART and switches the module off.

```
// closes the UART and powers off the module
BT_Pro.OFF();
```

4. General considerations

This section will describe the bluetooth module API. The functions which manage bluetooth module belong to the class **WaspBT_Pro**, and the object created to use them is defined as **BT_Pro**. All of them are described below including some examples of use.

4.1. Waspmote libraries

4.1.1. Waspmote bluetooth files

WaspBT_Pro.h, WaspBT_Pro.cpp.

Note: *If you are planning to detect a big amount of bluetooth devices, it is highly recommended to increase RX buffer of Waspmote UART. To do that, go to file wiring_serial.c and change value of RX_BUFFER_SIZE to 1024. More information can be obtained in the Libelium forum.*

4.1.2. Constructor

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

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

4.2. API functions

Next tables show all functions and variables contained in class "BT_Pro", including a brief description. In the next section, main functions are described in deep. However, not relevant ones will be ignored.

In the Waspmote Development section you can find complete examples about using this module:
http://www.libelium.com/development/waspmote/examples

Public functions:

`ON()`	Turns on Bluetooth module
`OFF()`	Turns off Bluetooth module
`checkActiveConnections()`	Checks if there are active connections
`createConnection()`	Creates a transparent connection with other Bluetooth module using serial port profile
`enterCommandMode()`	Enters command mode of Bluetooth module
`getNodeID()`	Reads Bluetooth Node Identifier
`GetOwnMac()`	Gets Bluetooth module MAC address
`getOwnName()`	Gets public name of Bluetooth device
`getRSSI()`	Returns the Receiver Signal Strength Indication of a link
`getTemp()`	Reads internal temperature sensor of Bluetooth module
`init()`	Initializes some parameters of the module
`printInquiry();`	Prints last inquiry results. Use only for debugging purposes
`removeConnection()`	Removes a transparent connection
`reset()`	Resets the module
`returnToDataMode()`	Switches from command mode to data mode

Public functions:

`scanNetwok (time , power)`	Normal scan
`scanNetwokLimited (Max_DEVICES, power)`	Time limited scan
`scanDevice (MAC, maxtime, power)`	Scans for specific device
`scanNetworkName ( time, power)`	Scans showing also friendly name
`sendData()`	Sends data trough a transparent connection with other Bluetooth module using serial port profile
`setNodeID(ID)`	Saves Bluetooth Node identifier into the EEPROM
`setOwnName(publicName)`	Sets public name on Bluetooth device module. Maximum length is 20 chars
`sleep()`	Turns Bluetooth module into Sleep mode
`wakeUp()`	Wakes up Bluetooth module from Sleep mode

Private functions:

`changeInquiryPower()`	Changes Inquiry TX power
`createSDFiles()`	Creates required files by Bluetooth module into SD card
`eraseSDFiles()`	Erases files created by Bluetooth module on SD card
`getSetDateID()`	Reads nodeID and date and stores it
`lookForAnswer()`	Searches a string in a text
`parseBlock()`	Parses received data from module
`parseNames()`	Looks for friendly names and stores them
`ReadCommandAnswer()`	Reads command answer and saves it
`ScanNetworkCancel()`	Cancels currently inquiry
`sendCommand()`	Sends command to Bluetooth module
`setInquiryTime()`	Sets time to be waiting for inquiry answer
`waitInquiryAnswer()`	Reads UART while inquiry is being answered
`waitScanDeviceAnswer()`	Reads UART while inquiry is being answered

Variables

Public variables are described below. Private variables are not relevant.

`uint16_t numberOfDevices`	Stores number of discovered devices in last inquiry
`char temperature`	Stores module temperature
`uint16_t numberOfDevices`	uint16_t numberOfDevices
`char temperature`	char temperature
`uint16_t numberOfDevices`	uint16_t numberOfDevices
`char temperature`	char temperature

4.3. Waspmote reboot

When Waspmote is rebooted or it comes up from a hibernate state (battery is disconnected), the application code will start again, creating all the variables and objects from the beginning.

4.4. Constants predefined

There are some constants predefined in the bluetooth library. Internal parameters like module baudrate, file names or transmission powers are defined here.

BLOCK_MAC_SIZE	Block size used to store MAC address
BLOCK_SIZE	Block size used to parse data
BT_BLUEGIGA_RATE	Default Baudrate of Bluetooth module
BT_NODE_ID_ADDR	EPROM address where node ID is stored.
BT_PW_1	Bluetooth power pin when UART1 is used.
BT_PW_0	Bluetooth power pin when UART0 is used.
COMMAND_SIZE	Reserved bytes for Bluetooth module commands.
DEFAULT_MAX_INQUIRY_RESULTS	Maximum inquiry results defined by internal firmware
ENABLE_DATE_AND_TIME	Uncomment to enable date and time in each device
ENABLE_FRIENDLY_NAME	Uncomment to enable friendly name features
ENDSTRING	File definition. By default "---"
ERRORSD1, ERRORSD2, ERRORSD2	SD card error codes for debugging.
INQFILE	File name where inquiry is stored
INQFILEHEAD	"Discovered devices". Format of inquiry file.
RX_BUFFER	Buffer for received data
TOTAL	File definition. By default "total:"
TX_POWER_0....6	TX power values accepted by module in dBm. 0 is minimum.
TX_POWER_DEFAULT_WT12	By default module is at this TX power
TX_POWER_MAX_WT12	Max TX power. Is equal to TX_POWER_6

5. Bluetooth device parameters

Some Bluetooth device parameters have been already described. This section will describe how they are saved and how the can be managed.

Due to limited memory of Waspmote microcontroller, all discovered devices are stored in an SD card. By default, a file called "DEVICES.TXT" is created in the root directory of the SD card containing data of all inquiry. This file is deleted every Waspmote reset.

Remember that the bluetooth module requires an SD card to save all data of discovered devices. Please be sure that an SD card is inserted before using the module.

The way of storing inquiry data is quite simple. First of all, a header is written to provide information of the inquiry carried out. The header contains date and time, identifier of the Waspmote node and scan type according to executed function.

After the header, all discovered devices are shown, being each line a different device with MAC address, CoD, RSSI and device type. At the end of the inquiry, the number of discovered devices during the inquiry is also shown. Next lines show an example of an inquiry saved into the Waspmote SD card.

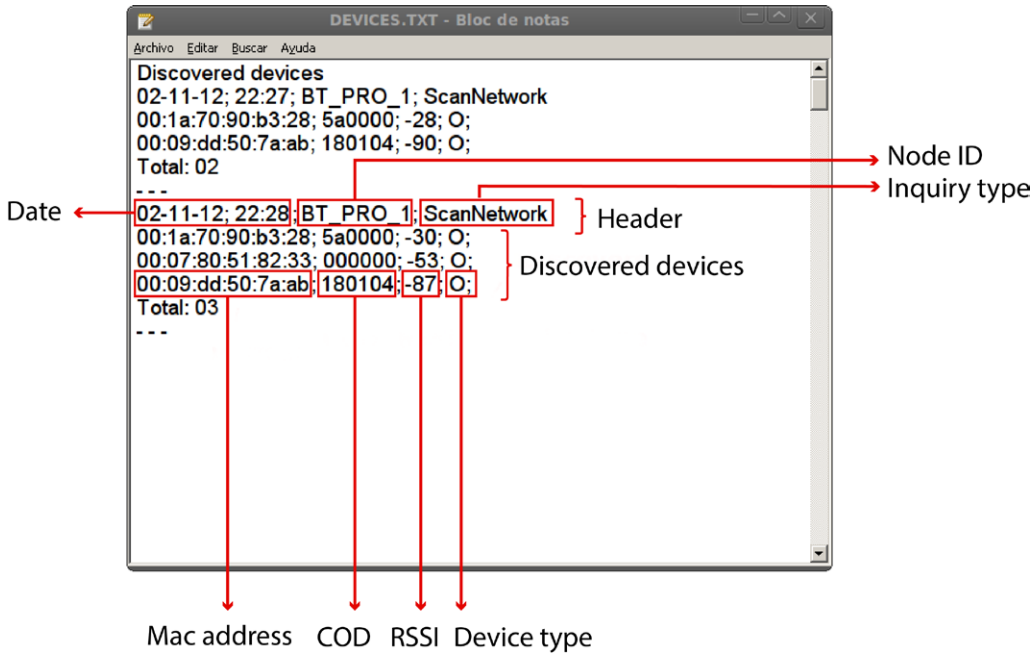


Figure : Inquiries saved into the SD card

As shown in previous figure, the header contains date and time, followed by the Bluetooth node identifier and the inquiry type (normal, limited, specific or with friendly name). Besides that, each discovered device is show as a MAC address, followed by the Class of Device (CoD), the RSSI value and the device type (pedestrian as "P", car as "C" and other as "O").

Moreover, the next inquiry will be stored below the previous one in the same way, allowing to save data of a big amount of different inquiries. This data can be managed later for other purposes.

It has to be remarked that data file is deleted every Waspmote reset.

6. Searching for devices

There are four different ways of scanning the network which corresponds with four defined functions in the bluetooth module API. In this functions, some parameters like inquiry power, inquiry time, number of devices, etc, have to be specified. Each function is described below.

ScanNetwork(Time, Power)

This function allows a simple scan of network. Inquiry time and TX power must be specified. Inquiry results are limited to 250 and they are stored in SD card file. The total of discovered devices is returned.

The usual time to discover about 20 devices is rounding 10 seconds, but time can be set from 1 to 48. Keep in mind that this time is approximated because time value is internally multiplied by 1.28 (predefined firmware value). That means that you can do inquiries until 61 seconds long. Besides that, some time is spent to process and save data.

Next example makes an inquiry of 10 seconds, but the real time spent is 12.8 plus parsing and saving data processes. Number of discovered devices is stored in public variable numberOfDevices.

```
{
  BT_Pro.scanNetwork(10, TX_POWER_6);
}
```

TX power values are predefined as constants because bluetooth module only has the capability of transmit at seven values. Any integer value can be entered here but module will round it to the nearest valid value.

In the Waspmote Development section you can find a complete example about using this function:

http://www.libelium.com/development/waspmote/examples/bt-pro-01-normal-scan

ScanNetworkName(Time, Power)

This function is equal to scanNetwork but it also includes friendly names, when they are available. User should take into account that friendly names have to be asked to each discovered device and this fact takes some extra time. By default this time is limited to 60 seconds but it can be changed modifying parseNames() in WaspBT_Pro.cpp.

Device names are saved after the inquiry data, showing each MAC address with their friendly name. When the device friendly name is not available, default value for name is "NONAME".

This function is very slowly and most of time "friendly names" are not available so keep it in mind when using this function.

```
{
  BT_Pro.scanNetworkName(10, TX_POWER_6);
}
```

In the Waspmote Development section you can find a complete example about using this function.

http://www.libelium.com/development/waspmote/examples/bt-pro-04-scan-with-friendly-name

ScanNetworkLimited(MAX_DEVICES, power)

This function scans the network in the same way of scanNetwork, but now you can limit your scan to a specific number. If this number is reached,the inquiry stops. However, if maximum is not reached, the module continues inquiring until maximum time.

Next example shows an inquiry at maximum power until find four devices. If four devices are present, they will be stored on the SD card.

```
{
  BT_Pro.scanNetworkLimited(4, TX_POWER_6);
}
```

In the Waspmote Development section you can find a complete example about using this function:

http://www.libelium.com/development/waspmote/examples/bt-pro-02-limited-scan

ScanDevice(MAC, maxtime, power)

This function is used to know if a specific device is present inside detection area of bluetooth module. Devices are commonly identified by its MAC address. When defining variables containing MAC addresses, be sure that they are defined with the format "00:1a:16:e8:c2:01", otherwise scanDevice will not find specified the device. In addition, the class of device has 6 hexadecimal digits and RSSI value is shown in dBm.

The maximum scanning time is defined by maxtime. This function scans network until the desired device is found. Once the device is found, it is saved into the SD card file. However, if the device is not found, the inquiry will continue until maxtime is reached. The returned value of this function is 1 if the device is found. Otherwise, it returns 0.

```
{
  BT_Pro.scanDevice("00:1a:70:90:b3:28", 10, TX_POWER_6));
}
```

In the Waspmote Development section you can find a complete example about using this function:
http://www.libelium.com/development/waspmote/examples/bt-pro-03-scan-specific-device

PrintInquiry()

By default, discovered devices are not shown through USB because they are only saved on the SD card. If the user wants to view discovered devices just use printInquiry() function after the scan function and discovered devices in the last inquiry will be shown. Use this function only for debugging purposes due to it can become slow when the SD card files are big.

7. Connecting to other Bluetooth module

Besides than scanning functions, the Bluetooth API contains some functions to open a transparent connection to other Bluetooth module.

CreateConnection (MAC)

This function is used to open a transparent connection with other Bluetooth module using the Serial Port Profile. The MAC address of the remote Bluetooth device must be specified.

```
{
  if (BT_Pro.createConnection(mac) == 1)
    {
      USB.println(F("Connected."));
    }
  else
    {
      USB.println(F("Not conencted"));
    }
}
```

SendData (data)

This function sends data directly to other Bluetooth module using a previously created connection.

```
{
  if (BT_Pro.sendData("Hello, This is Bluetooth module.") == 1)
  {
    USB.println(F("Data sent"));
  }
}
```

RemoveConnection()

This function removes a transparent connection with other Bluetooth module.

```
{
  if (BT_Pro.removeConnection() == 1)
  {
    USB.println(F("Connection removed"));
  }
  else
  {
    USB.println(F("Not removed"));
  }
}
```

In the Waspmote Development section you can find a complete example about using these functions:

http://www.libelium.com/development/waspmote/examples/bt-pro-10-creating-a-transparent-connection

7.1. Pairing with devices

If the user wants to connect one device with security, both devices have to be paired before. A pin code can be used also to avoid man-in-the-middle intrusions. There are three functions to manage paired devices saved by the Bluetooth module.

Pair(macAddress, pincode)

This function pairs the Bluetooth module with the specified Bluetooth device. If the pin code is not specified, default code will be "123456".

```
{
  if (BT_Pro.pair("00:1a:70:90:b3:28") == 1)
  {
    USB.println(F("Paired OK"));
  }
  else
  {
    USB.println(F("Not paired"));
  }
}
```

IsPaired(deviceMac)

This function is used to check if a device is already paired.

```
{
  if (BT_Pro.isPaired("00:1a:70:90:b3:28") == 1)
  {
    USB.println(F("Already paired"));
  }
  else
  {
    USB.println(F("Not paired"));
  }
}
```

removePairedDevices()

This function removes all paired devices from the list.

```
{
  BT_Pro.removePairedDevices();
}
```

In the Waspmote Development section you can find a complete example about using these

functions: **http://www.libelium.com/development/waspmote/examples/bt-pro-12-pairing-example**

7.2. Using WaspFrame class to create sensor data frames

WaspFrame is a class that allows the user to create data frames with a specified format. It is a very useful tool to build the packet in the same way as other Waspmote examples. It is recommended to read the Waspmote Programming Guide available here:

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

8. Code examples and extended information

In the Waspmote Development section you can find complete examples about using this module:
http://www.libelium.com/development/waspmote/examples

Next lines show a complete example of use for scanning functions.

```
/*
 *  ------------------ [BT PRO.01] - Normal scan -------------------
 *
 *  Explanation: This example shows how to make a normal scan with
 *  Bluetooth module Pro, printing number of discovered devices
 *  and storing them into SD card.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 *  Version:          0.1
 *  Design:           David Gascón
 *  Implementation:   Javier Siscart
 */

#include “WaspBT_Pro.h”

void setup(){

  // Setup for Serial port over USB
  USB.ON();
  USB.println(F(“USB port started...”));

  // Turn On Bluetooth module
  BT_Pro.ON(SOCKET1);

}

void loop(){

  // 1. Normal scan
  USB.println(“Scan 5s.”);
  BT_Pro.scanNetwork(5, TX_POWER_6);
  USB.print(“discovered devices=”);
  USB.println(BT_Pro.numberOfDevices, DEC);

  // 2. Print data of last inquiry (only debug purposes)
  BT_Pro.printInquiry();

}
```

9. Certifications

Libelium offers 2 types of IoT 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: 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: www.libelium.com/products/plug-sense

Besides, Meshlium, our multiprotocol router for the IoT, is also certified with the certifications below. Get more info at:

www.libelium.com/products/meshlium

List of certifications for Plug & Sense! and Meshlium:

- CE (Europe)
- FCC (US)
- IC (Canada)
- ANATEL (Brazil)
- RCM (Australia)
- PTCRB (cellular certification for the US)
- AT&T (cellular certification for the US)

Figure : Certifications of the Plug & Sense! product line

You can find all the certification documents at:

www.libelium.com/certifications

10. API Changelog

Keep track of the software changes on this link:

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