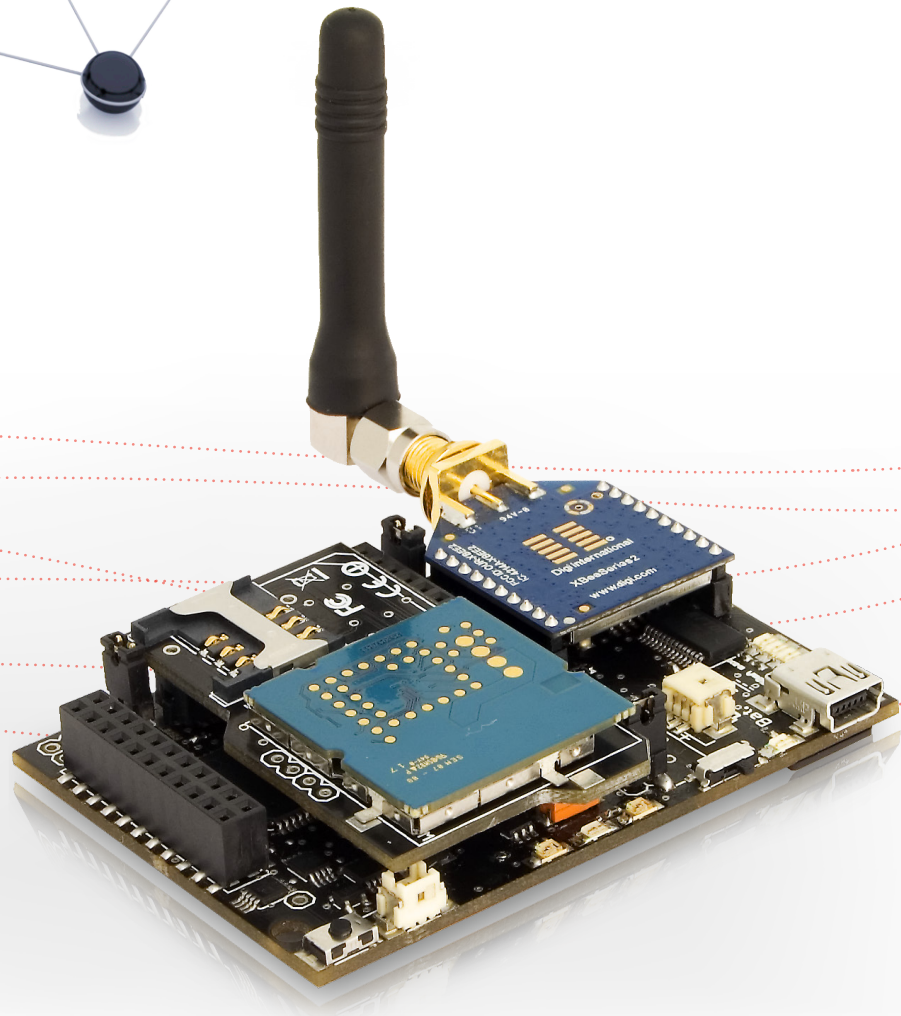
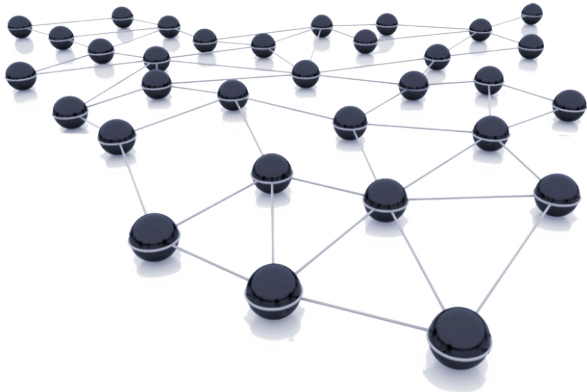


Bluetooth for device discovery

Networking Guide



Document Version: v0.5 - 04/2012
© Libelium Comunicaciones Distribuidas S.L.

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. General considerations	6
3.1 Wasmote libraries.....	6
3.1.1 Wasmote bluetooth files.....	6
3.1.2 Constructor.....	6
3.2 API functions.....	6
3.3 Wasmote reboot	7
3.4 Constants predefined.....	7
4. Initialization	8
4.1 Setting ON	8
4.2 Setting OFF.....	8
4.3 Initializing	8
5. Bluetooth Device parameters.....	9
6. Searching for devices	10
7. Code examples.....	12
8. Code examples and extended information	15

1. Introduction

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, leaving for future developments other applications like data exchange.

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

This guide will try to describe main parts of bluetooth module and how to carry out many types of inquiries, providing code examples for each case.

1.1 General description

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

Detection area is variable from 10 to 50 meters, just changing the radio power of the inquiry. This action is carried out easily just changing a defined API parameter called TX_POWER which have seven allowed values. Values can be checked in the header file.

There are some parameters that can be inquired from devices inside detection area. The mos 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 digit and it allows distinguish if the detected bluetooth device is a vehicle, a pedestrian, and so on.

RSSI (Received Signal Strength Indicator): This parameters 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 MAC address.

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

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



Figure 1.1: Example of TX power levels

2. Hardware

2.1 Specifications

Main features of bluetooth module are listed below:

- Bluetooth v2.1 + EDR. Class 2
- TX Power: 3dBm
- Antenna: 2dBi
- 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 [-27dBm, +3dBm]
- Scan devices with maximum inquiry time
- Scan devices with maximum number of nodes
- Scan devices looking for a certain user by MAC address



Figure 2.1: 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. General considerations

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

3.1 Waspmote libraries

3.1.1 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_1 to 1024.

3.1.2 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.

3.2 API functions

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

Public functions:

ON()	Turns on bluetooth module. Opens UART
OFF()	Turns off bluetooth module. Closes UART
init()	Initializes some parameters of the module
reset()	Resets the module
getTemp()	Reads temperature from internal sensor
ScanNetwok (time , power)	Normal scan
ScanNetwokLimited (Max_DEVICES, power)	Time limited scan
ScanDevice (MAC, maxtime, power)	Scan for specific device
ScanNetworkName (time, power)	Scan showing also friendly name
PrintInquiry();	Prints last inquiry results

Private functions:

set Mode()	Sets module ON or OFF
lookForAnswer()	Searches a string in a text
sendCommand()	Send command to bluetooth module
ReadCommandAnswer()	Reads command answer and saves it
changeInquiryPower()	Changes Inquiry TX power
parseBlock()	Parses received data from module
waitInquiryAnswer()	Reads UART while inquiry is being answered
waitScanDeviceAnswer()	Reads UART while inquiry is being answered
getSetDateID()	Read nodeID and date and stores it.
parseNames()	Looks for friendly names and stores them.

setInquiryTime()	Sets time to be waiting for inquiry answer
ScanNetworkCancel()	Cancels currently inquiry

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

3.3 Wasp mote reboot

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

3.4 Constants predefined

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

BT_BLUEGIGA_RATE	Preconfigured Baudrate of bluetooth module
PORT_USED	UART used by bluetooth module
DEFAULT_MAX_INQUIRY_RESULTS	Maximum inquiry results defined by internal firmware
RX_BUFFER	Size of UART 1 RX buffer. Set in wiring_serial.c
BLOCK_SIZE	Block size used to parse data
BLOCK_MAC_SIZE	Block size used to store Mac
BT_ON and BT_OFF	BLOCK_SIZE
INQFILE	File name where inquiry is stored
BT_PW	Bluetooth pin
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

4. Initialization

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

4.1 Setting ON

With the function ON() module is powered and the UART is opened to communicate with the module. It returns nothing.

Example of use

```
{  
  BT_Pro.ON(); // Powers the module and opens UART  
}
```

Note: For API versions V0.23 and older, It is necessary to turn on SD card before turning on bluetooth module. Otherwise it will not work properly.

4.2 Setting OFF

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

Example of use

```
{  
  BT_Pro.OFF(); // Closes UART and switch off the module.  
}
```

4.3 Initializing

Using bluetooth function init(), some configuration parameters are sent to the module and SD card is prepared to save inquiry data. It returns nothing

Example of use

```
{  
  BT_Pro.init(); // Configuration parameters.  
}
```

5. Bluetooth Device parameters

In section 2.1 some device parameters are described. This section will describe how they are saved and how they can be managed.

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

Remember that the bluetooth module requires a SD card to save all data of discovered devices. Please be sure that a 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 when inquiry is done, identifier of the Waspote node and scan type according to executed function.

Following header all discovered devices are shown, being each line a different device with MAC address, CoD and RSSI. At the end of inquiry, the number of discovered devices during the inquiry is also shown.

Moreover, 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 Waspote 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 till 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".

```
void setup(){
  // setup for Serial port over USB
  USB.begin();
  USB.println("USB port started...");

  // setup for RTC
  RTC.ON();
  // Insert your local time here, only necessary one time
  RTC.setTime("11:10:05:04:11:15:00");
  USB.println(RTC.getTime());

  // Setup for SD
  USB.println("Starting SD...");
  SD.ON();
  if (SD.isSD()==0) USB.println("Card not present. Please insert Card");

  // setup for Bluetooth module
  BT_Pro.ON();
  BT_Pro.init(nodeID); // saves nodeID into EEPROM
}

void loop(){
  // Normal scan
  int a=0;
  USB.println("Scan Network 10s.");
  a=BT_Pro.scanNetwork(10,TX_POWER_6);
  USB.print("discovered: ");
  USB.println(a);
}
```

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.

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 take 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 with their friendly name. When 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.

```
// Name Scan

USB.println("Scan Network with NAME 20s.");
a = BT_Pro.scanNetworkName(10,TX_POWER_6);
USB.print("discovered: ");
USB.println(a);
```

ScanNetorkLimited(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, inquiry stops. However, if maximum is not reached, the module continues inquiring till maximum time and at the end the message "Maximum not reached" is printed on USB.

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

```
// Limited scan

USB.println("Scan Network till find 4 devices at max TX power.");
BT_Pro.scanNetworkLimited(4,TX_POWER_6);
```

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 device. In addition, class of device has 6 hexadecimal digits and RSSI value is shown in dBm.

Maximum scanning time is defined by "maxtime". This function scans network till find desired device. Once device is found, it is saved into SD card file. However, if device is not found, inquiry will continue till reach "maxtime".

Returning value of this function is 1 if device is found. Otherwise it returns 0.

```
// Specific device can

USB.print("Scan 10s looking for mac: ");
USB.println(mac);
if (BT_Pro.scanDevice(mac,10,TX_POWER_6)) USB.println("Device found.");
else USB.println("Device not found.");
```

7. Code examples

Next lines show a complete example of use for scanning functions. Also the response for this example is included.

```
char* nodeID="00000001"; // WASPMOTE ID. It must have 8 characters "
int a=0;

void setup(){
  // setup for Serial port over USB
  USB.begin();
  USB.println("USB port started...");

  // setup for RTC
  RTC.ON();

  // Insert your local time here, only necessary one time
  RTC.setTime("11:10:05:04:11:15:00");
  USB.println(RTC.getTime());

  // Setup for SD
  USB.println("Starting SD...");
  SD.ON();
  if (SD.isSD()==0) USB.println("Card not present. Please insert Card");

  // setup for Bluetooth module
  BT_Pro.ON();
  BT_Pro.init(nodeID); // saves nodeID into EEPROM
}

void loop(){

  // Mac for module must be defined as follows
  char mac[18] = "00:1a:16:e8:c2:01";

  // Normal scan

  USB.println("Scan Network 10s.");
  a=BT_Pro.scanNetwork(10,TX_POWER_6);
  USB.print("discovered: ");
  USB.println(a);

  // Name Scan

  USB.println("Scan Network with NAME 20s.");
  a = BT_Pro.scanNetworkName(10,TX_POWER_6);
  USB.print("discovered: ");
  USB.println(a);

  // Limited scan

  USB.println("Scan Network limited to 4.");
  a = BT_Pro.scanNetworkLimited(4,TX_POWER_6);
  USB.print("discovered: ");
  USB.println(a);

  // Specific device can

  USB.print("Scan 10s looking for mac: ");
  USB.println(mac);
  if (BT_Pro.scanDevice(mac,10,TX_POWER_6)) USB.println("Device found.");
  else USB.println("Device not found.");

  // get temp
```

```

long b= BT_Pro.getTemp();
USB.print("Module Temp: ");
USB.println(b);
delay(5000);
}

```

One possible response for previous code is shown below. By default, discovered devices are not shown through USB because they are only saved on SD card. If the user wants to view discovered devices just use `printInquiry()` function after the scan function and discovered devices in last inquiry will be shown.

```

USB port started...
Wednesday, 11/10/05 - 11:15.00
Starting SD...

Scan Network 10s.
discovered: 20

Scan Network with NAME 20s.
discovered: 17

Scan Network limited to 4.
discovered: 4

Scan 10s looking for mac: 00:1a:16:e8:c2:01
Device found.

Module temperature: 35

```

And the archive on SD card (by default "devices.txt") will contain the text below.

```

05-10-11;11:15; 00000001; ScanNetwork
00:1a:70:90:b3:3b; 4a0000; -76;
00:1e:8c:dd:8a:95; 18010c; -83;
00:1a:70:90:b3:2b; 4a0000; -73;
00:1a:70:90:b3:44; 4a0000; -73;
00:03:19:0d:0e:b8; 111111; -73;
00:03:19:0d:0e:b7; 111111; -83;
00:03:19:0d:0e:c0; 111111; -64;
00:18:f8:89:82:45; 4a0000; -83;
00:80:5a:46:3b:28; 4a0100; -74;
00:25:d3:b0:5f:2e; 4a010c; -77;
00:1a:16:e8:c2:01; 5a0204; -53;
00:80:5a:20:eb:9c; 4a0000; -44;
00:07:80:49:58:b5; 001f00; -30;
00:1a:70:90:b3:28; 4a0000; -46;
00:03:19:0d:0e:b9; 111111; -70;
00:22:a9:3c:09:92; 5a0204; -83;
00:03:19:0d:0e:ba; 111111; -66;
00:03:19:0d:0e:bb; 111111; -73;
00:03:19:0d:0e:d7; 111111; -73;
Total: 19
;
05-10-11;11:15; 00000001; ScanNetworkName
00:18:f8:89:82:45; 4a0000; -75;
00:1e:8c:dd:8a:95; 18010c; -85;
00:1a:70:90:b3:2b; 4a0000; -73;
00:07:80:49:58:b5; 001f00; -29;
00:1a:70:90:b3:3b; 4a0000; -83;
00:1a:70:90:b3:28; 4a0000; -46;
00:1a:70:90:b3:44; 4a0000; -69;
00:03:19:0d:0e:b6; 111111; -78;
00:03:19:0d:0e:b7; 111111; -83;
00:25:d3:b0:5f:2e; 4a010c; -69;
00:03:19:0d:0e:ba; 111111; -64;

```

```
00:22:a9:3c:09:92; 5a0204; -88;
00:1a:16:e8:c2:01; 5a0204; -52;
00:80:5a:46:3b:28; 4a0100; -66;
00:03:19:0d:0e:c0; 111111; -64;
00:03:19:0d:0e:b8; 111111; -80;
00:80:5a:20:eb:9c; 4a0000; -40;
00:03:19:0d:0e:d7; 111111; -77;
Friendly names:
00:18:f8:89:82:45 "Broadcom BCM2035"
00:1e:8c:dd:8a:95 "MADERILO"
00:1a:70:90:b3:2b "Broadcom BC
00:07:80:49:58:b5 "javi2"
00:1a:70:90:b3:3b "Broadcom BC
00:1a:70:90:b3:28 "Broadcom BC
00:1a:70:90:b3:44 "Broadcom BC
00:03:19:0d:0e:b6 NONAME
00:03:19:0d:0e:b7 "waspmote_bt"
00:25:d3:b0:5f:2e "matrix-0"
00:03:19:0d:0e:ba "waspmote_bt"
00:22:a9:3c:09:92 "Teo"
00:1a:16:e8:c2:01 "Sisko"
00:80:5a:46:3b:28 "libelium-de
00:03:19:0d:0e:c0 "waspmote_bt"
00:03:19:0d:0e:b8 "waspmote_bt"
00:80:5a:20:eb:9c "Sisko-PC"
00:03:19:0d:0e:d7 "waspmote_bt"
Total: 18
;
05-10-11;11:16; 00000001; ScanNetworkLimited.
00:1a:70:90:b3:3b; 4a0000; -80;
00:18:f8:89:82:45; 4a0000; -84;
00:1a:70:90:b3:44; 4a0000; -76;
00:03:19:0d:0e:bb; 111111; -72;
;
05-10-11;11:16; 00000001#; ScanDevice: 00:1a:16:e8:c2:01
00:1a:16:e8:c2:01; 5a0204; -58;
```

8. Code examples and extended information

For more information about the WaspMote hardware platform go to:

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

<http://www.libelium.com/support/waspmote>

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