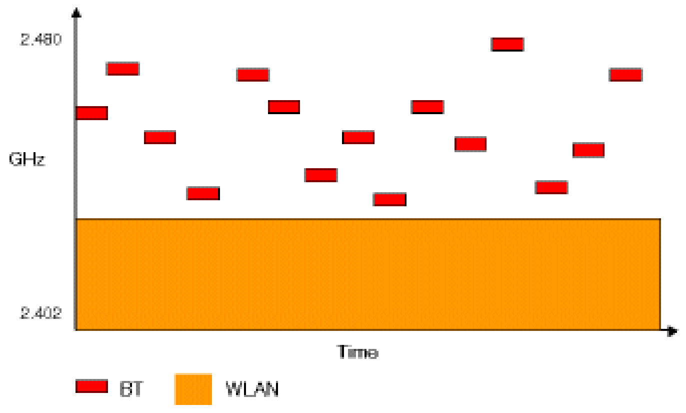
How Bluetooth is working ?

Bluetooth is a wireless solution running around 2,4 Ghz. It shares ISM (Industrial Scientific Medical) band and divides it into 40 channels on 2 MHz spacing from 2,4000GHz to 2,4835GHz. Wall these 40 channels, 3 are for advertiser (use for asking connection for example) and 37 for data transfer (Use once connected).

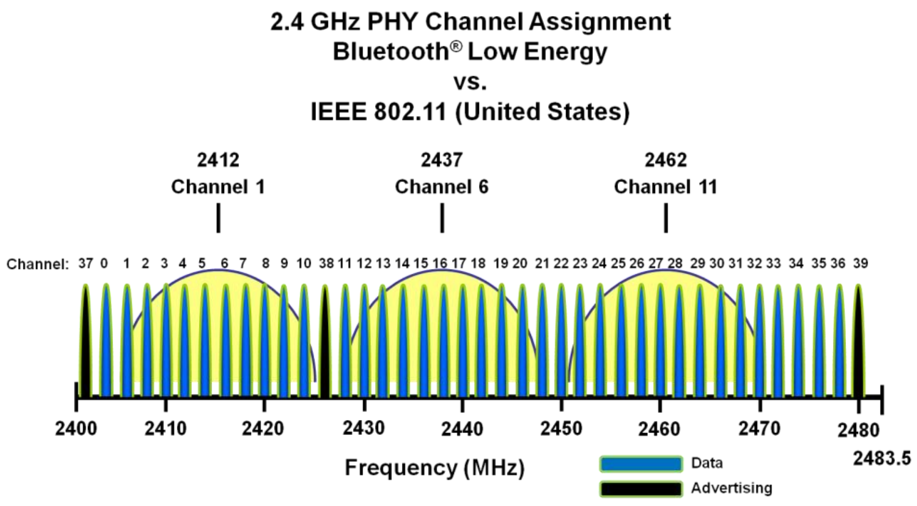


Data Rate is about 1Mbps. Bluetooth communication is done thank to adaptative frequency hopping (figure above). It permits to change frequency every 625 µs (1600 hop per seconds).

It follows this: Fn+1 = (Fn + hop) %37.

This method permits two things:

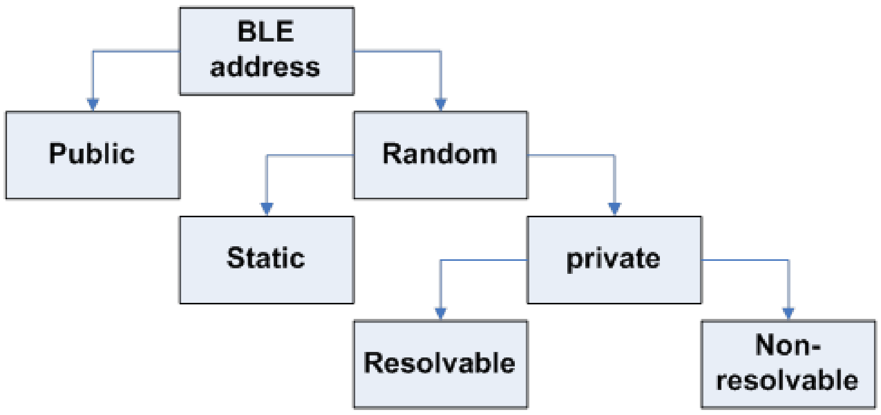
- Avoid to be listen, indeed you change frequently frequency so it's difficult to catch specific data. (Use by army previously)
- Avoid to lost data. Bluetooth shared his band with Wi-Fi and Wi-Fi band is larger than Bluetooth band)


2.4 GHz PHY Channel Assignment
Bluetooth® Low Energy
vs.
IEEE 802.11 (United States)
2412 Channel 1
2437 Channel 6
2462 Channel 11
Channel: 37 0 1 2 3 4 5 6 7 8 9 10 38 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 39
2400
2410
2420
2430
2440
2450
2460
2470
2480
2483.5
Frequency (MHz)
Data
Advertising

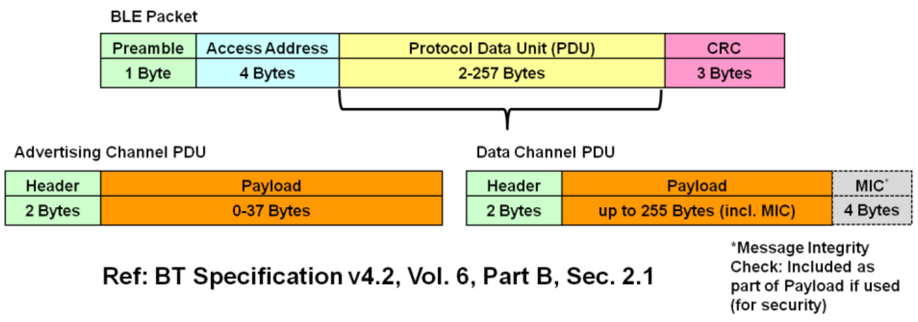
Both devices in a Bluetooth communication could have different role:

- Advertiser / Scanner (Use for connection for example)
- Master / Slave (Between a phone and a device for example)
- Broadcast / Observer (Beacon mode for example)

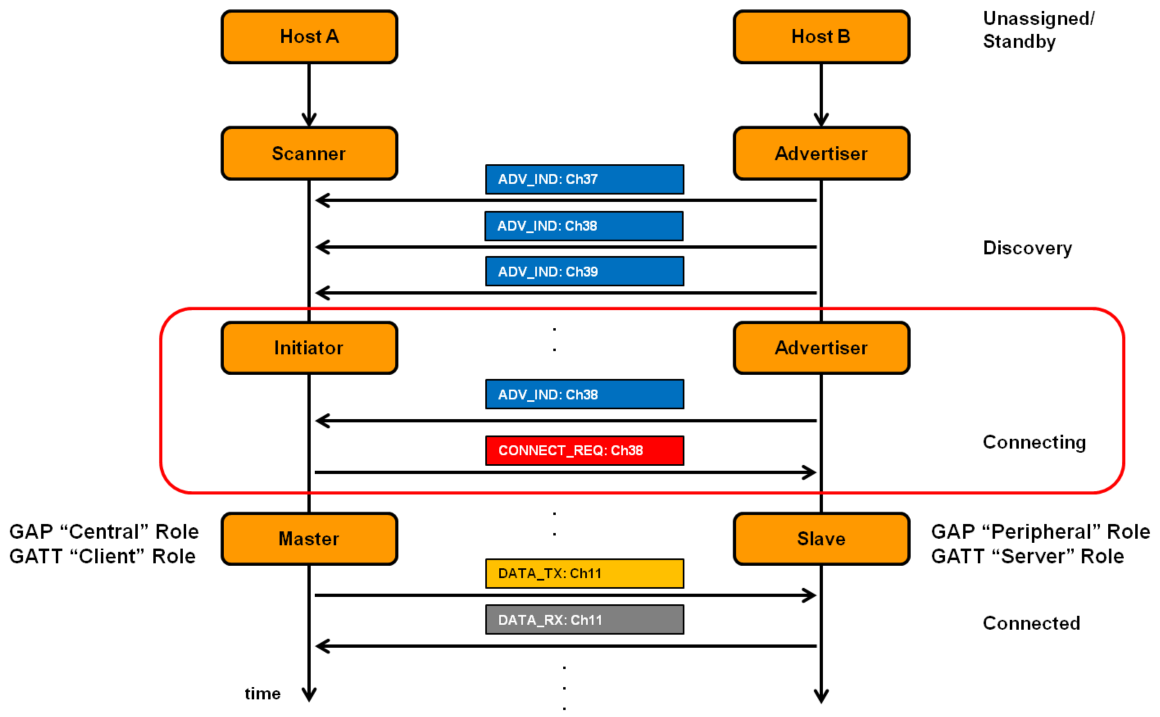
Each device has an address. This one could be the original or not at all:

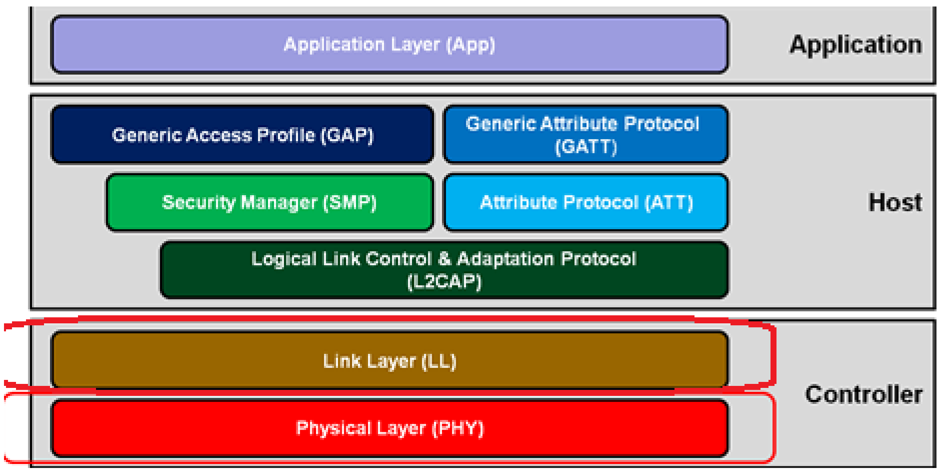


The BLE Link Layer has only one packet format for Advertiser or data channel packets



To be connected to a device, a master should advertise the slave (in scanner mode) by sending advertising packets switching continuously of advertise channel:





That was the physical layer and link layer

Now we will talk about Host Layer (Generic Access Profile GAP and Generic Attribute Profile GATT).

GAP describes procedures between two devices to be connected. There are 3 main operations;

- Discover and connect with peers
- Broadcast Data
- Establish secure connections

Different GAP Roles are:

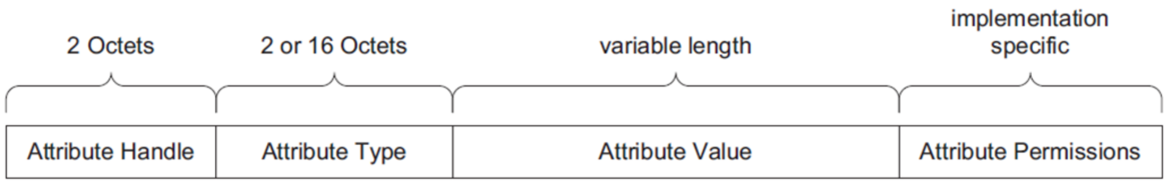
- Broadcaster (Advertise for example)
- Observer (Scan mode)
- Peripheral (Slave Device)
- Central (Master device)

There is only two mode: Discovery & Connection

This layer is used for connection procedure. It permits to do bonding too. What is bonding? It permits to store security keys of a connected device to accelerate future pairing. In the RN487x, all bonding device are added in a “While list” for example.

Then we have the GATT layer that establish how data is organized for a Bluetooth communication.

We will talk about attribute. An attribute has the following structure:



Attribute handle is in fact the principal UUID, a unique 16-bit identifier.

Then you have Attribute type which is a 128-bit ID composed of service UUID, characteristic UUID, profile UUID and vendor specified UUID.

Then you have the data to finish with attribute permissions (Read/write/both/encryption, …).

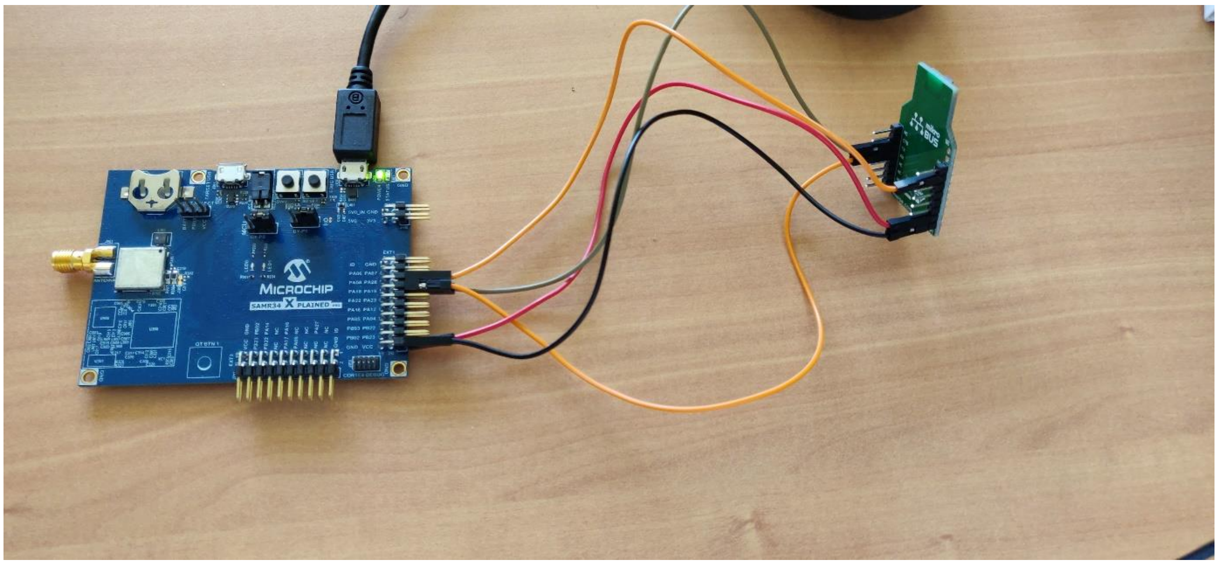
With this layer, there is only two roles possible Client and Server.

I talked about service UUID. What is it? Certain use case specific profile are standardized. You can choose among 40 different services like ‘hearth rate’ or ‘http proxy’. Click on this link to know more about this 40 services : https://www.bluetooth.com/specifications/gatt/services/

Demo Application

Hardware setup

Before use RN4871 you need to update firmware version. Indeed, this update correct important problem on this board. To do that, follow this link https://microchipdeveloper.com/ble:rn4870-app-example-fw-upgrade.



We will use a SAMR34 Xplained pro board and a RN4871 (It works for RN4870 too) mounted on a click Board named “RN4871 click”.

Pin table

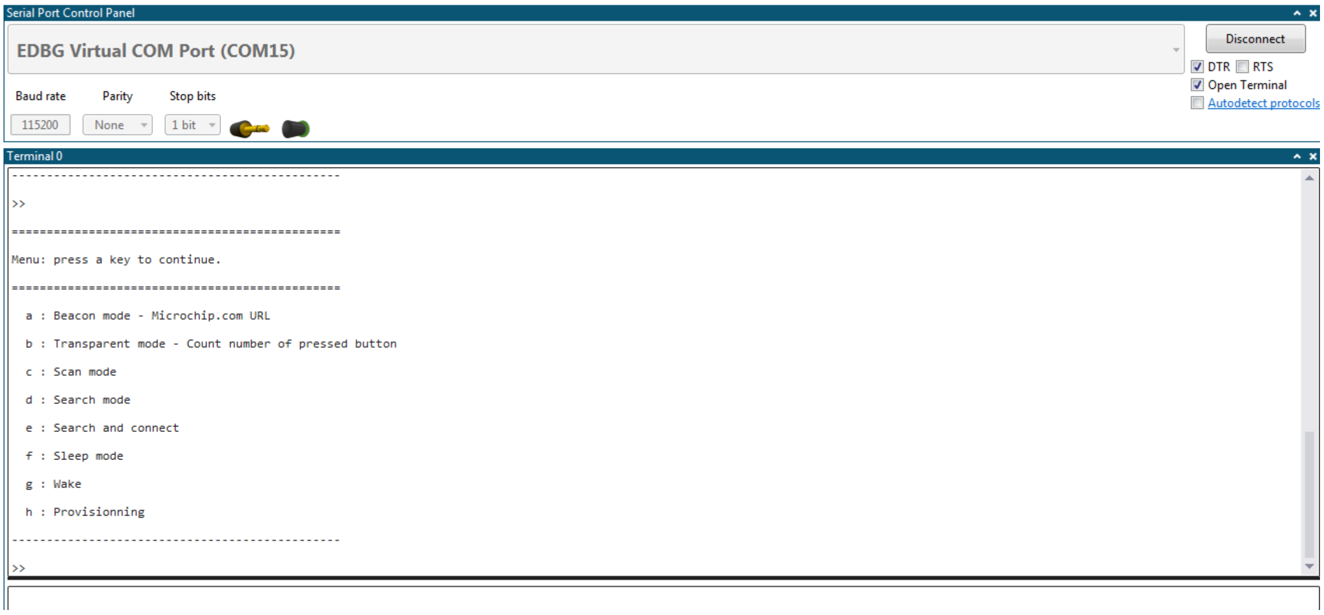
Pin / Board	SAMR34	RN487x
UART	PA18	Rx
UART	PA19	Tx
VCC	Vcc	3.3
GND	GND	GND
RESET	PA08	RST

I didn’t choose Extension header bus because it uses PA04 and PA05 pin. To use “printf” function as debug on EDBG UART, we need to use these pins so I had to change UART pin for RN. The switch on Click Board must be on ‘0’.

Software presentation

Once the hardware setup done, open the project on Atmel Studio and run it on SAMR34 Board. You need to open Data Visualizer and choose serial porn on EDBG Virtual COM Port with 115200 bauds.

Then you will see this menu:



```
-----------------------------------------------

>>

===============================================

Menu: press a key to continue.

===============================================

  a : Beacon mode - Microchip.com URL

  b : Transparent mode - Count number of pressed button

  c : Scan mode

  d : Search mode

  e : Search and connect

  f : Sleep mode

  g : Wake

  h : Provisionning

-----------------------------------------------

>>
```

You have the choice of which mode you want to try.

A – Beacon Mode

<u>What is a Beacon?</u>

The beacon is a small Bluetooth Low Energy beacon. It emits a short-range radio wave (~100 meters) and consumes very little energy. This wave is picked up by smartphone that can receive different content like push notification, message, URL, etc … Generally, use for publicity or push notification when the user pass near to a shop.

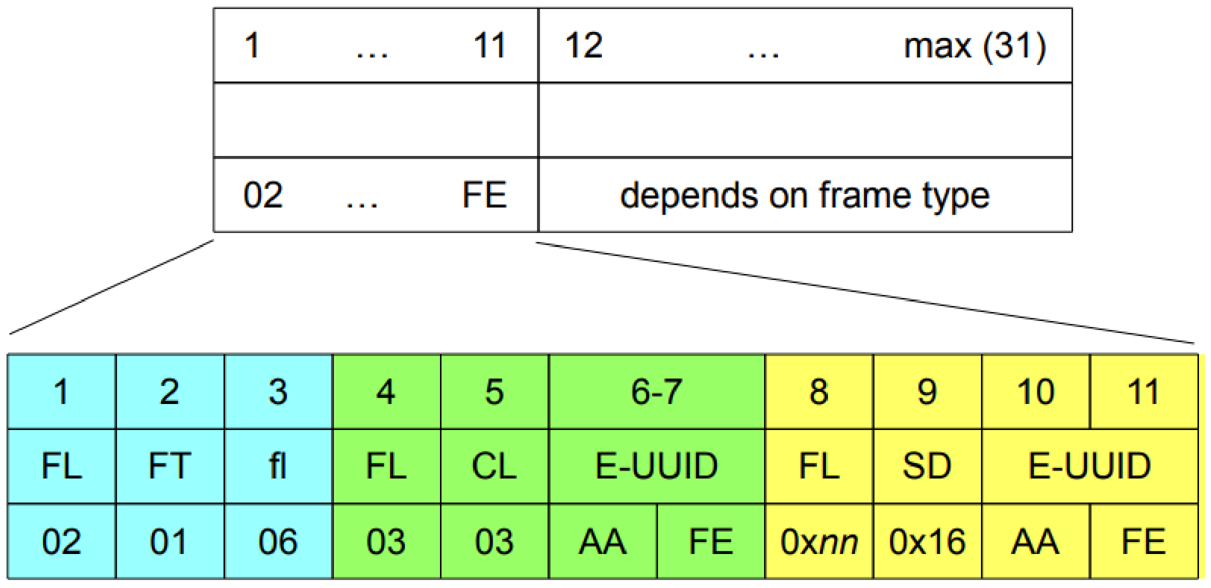
It permits to have an idea of distance between transmitter and receiver thank to the change of RSSI.

First one has been created by Apple in 2013, this is iBeacon, then was created the Altbeacon by Radius network in 2014.

The last one has been created by google in 2015, the Eddystones. This is the one we chose for the example.

Thank to Microchip Bluetooth Data app we can see our beacon. (Available on android and Iphone)

Let see Frame format:



1 … 11	12 … max (31)
02 … FE	depends on frame type

1	2	3	4	5	6-7		8	9	10	11
FL	FT	fl	FL	CL	E-UUID		FL	SD	E-UUID	
02	01	06	03	03	AA	FE	0x*nn*	0x16	AA	FE

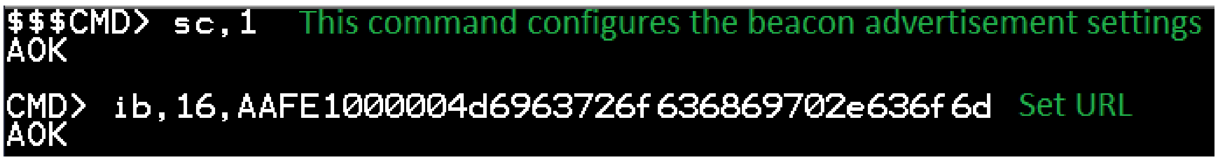
FL = Field Length
FT = Field Type
fl = LE and BR/EDR flag
CL = Complete list of service UUID
SD = Service Data
E-UUID = Eddystone UUID (0xFEAA)

Nine first bytes are ignored by module, probably because there is abstract degree thank to the Bluetooth stack. So, your frame has to begin with AAFE (This is the Eddystone UUID). Then you have to precise a frame type. There are only 3 possibilities:

- 0X00 for set Eddystone UID
- 0X01 for set an URL
- 0X02 for set Eddystone TLM (Vbatt, temp, and other)

If you want more precision read this : https://www.blueupbeacons.com/docs/WorkshopBeacons.pdf

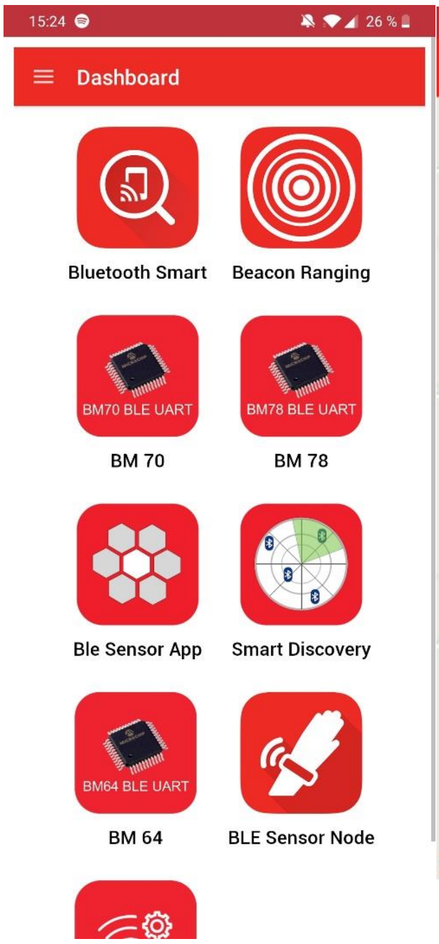
That is the specific command to set beacon:

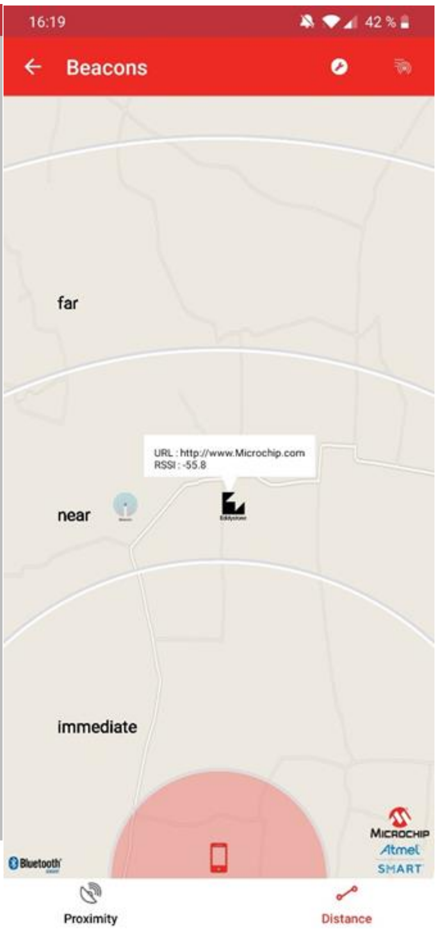


```
$$$CMD> sc,1   This command configures the beacon advertisement settings
AOK

CMD> ib,16,AAFE1000004d6963726f636869702e636f6d  Set URL
AOK
```

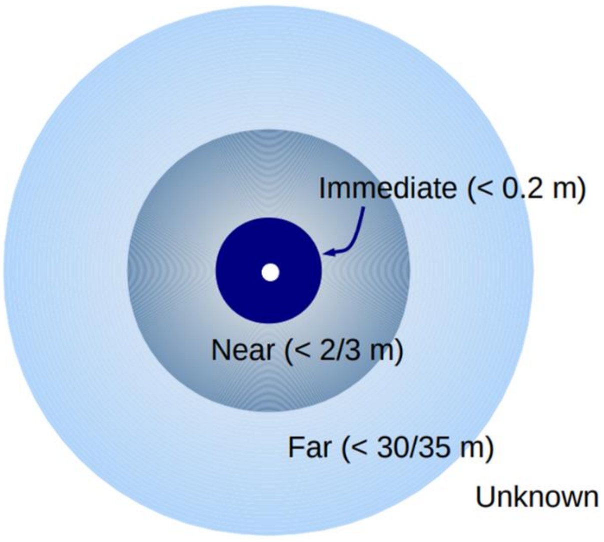
Once send a ‘a’ character, you will wait few second. Once you see ‘done.’ appears on HyperTerminal, your beacon is visible. Launch the app and click on Beacon Ranging.





URL : http://www.Microchip.com
RSSI : -55.8

Concerning proximity, of course it is not precise but it can help to have a idea. Let see the picture below:



But like I said it is not precise. For example, distance between my phone and RN4870 never exceeds 20cm and as you can see in my first screenshot, app indicates that my BLE module is in the near area.

B- Transparent mode

This mode permits, once connected to a peer device, to stream data through UART interface. This example will show you the number of pressed button (On SW0) sending by Bluetooth on your smartphone thank to Microchip Bluetooth Data.

After send a ‘b’ char, process to connect your phone will be explained on Hyperterminal.

The software only set support device information and UART Transparent as Service. Indeed, you must set a service among this table:

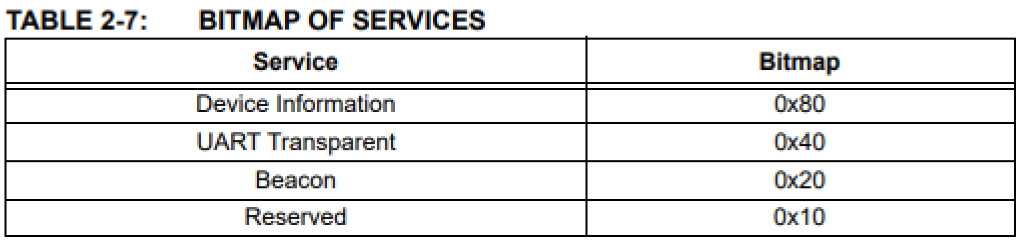
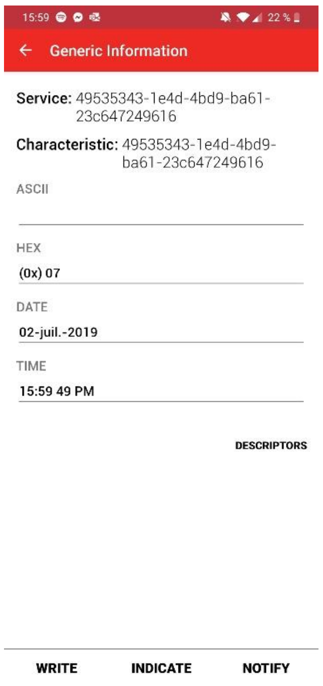


TABLE 2-7: BITMAP OF SERVICES

Service	Bitmap
Device Information	0x80
UART Transparent	0x40
Beacon	0x20
Reserved	0x10

Then it will reboot to save configuration.

There is the result



15:59 22 %

Generic Information

Service: 49535343-1e4d-4bd9-ba61-23c647249616

Characteristic: 49535343-1e4d-4bd9-ba61-23c647249616

ASCII

HEX
(0x) 07

DATE
02-juil.-2019

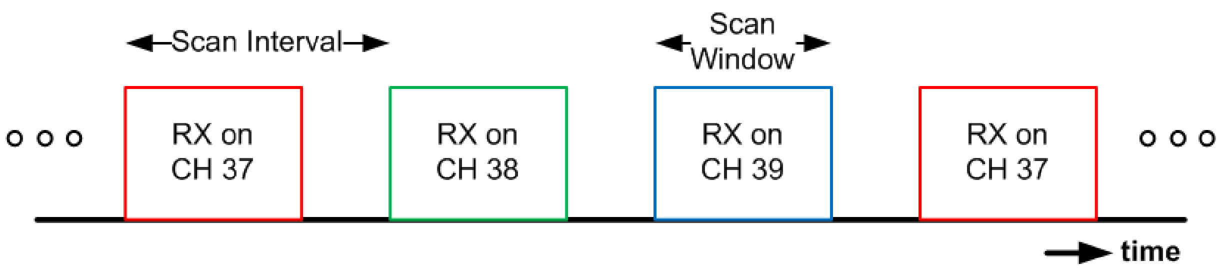
TIME
15:59 49 PM

C – Scan Mode

Choice ‘c’, ‘d’ and ‘e’ use the same function, but thank to a flag provide as parameter, it does not do the same thing. Indeed, first choice only show all devices scanned. Second one will search for a specific MAC address. Third will scan, search and try to connect to the specify address.

This function specifies if the address scanned is connectable or not.

For a scan you can choose 2 parameters: Scan interval and scan window



I set it to have 1100 ms for scan interval and 200 ms for scan window.

```
-----------------------------------------------

>>
You input :c
Wait for scan ...

5B8C548AE1C2 - Connectable
FADF301CAC4F - Non-connectable
5E46F8ABD3AB - Non-connectable
```

For last choice, the device will scan around him every mac address until it doesn’t find the good one comparing input parameter and each receiving address. Once found, it will send to him a connect request.

D – Wake and sleep mode

For this part, it is simple. The module will sleep after receive ‘0,o’ as command. To wake up, the only way is to do an hardware reset. (Command provide by driver).

E – Provisioning

In this example everything is explain when you make your choice by sending ‘h’ character.

It will ask you to send 3 different data:

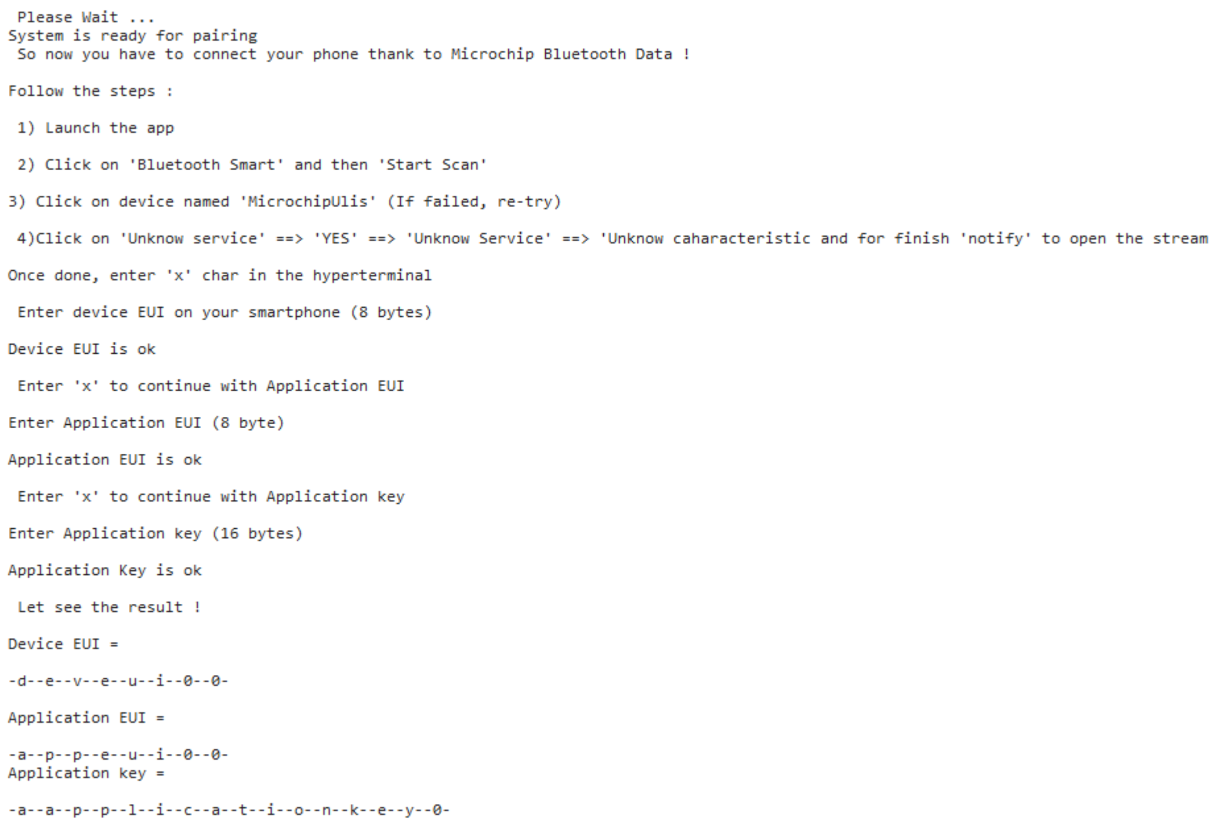
Device EUI (8 bytes)

Application EUI (8 bytes)

Application key (16 bytes)

And then it will display it on the HyperTerminal.

This is an example:



```
 Please Wait ...
System is ready for pairing
 So now you have to connect your phone thank to Microchip Bluetooth Data !

Follow the steps :

 1) Launch the app

 2) Click on 'Bluetooth Smart' and then 'Start Scan'

3) Click on device named 'MicrochipUlis' (If failed, re-try)

 4)Click on 'Unknow service' ==> 'YES' ==> 'Unknow Service' ==> 'Unknow caharacteristic and for finish 'notify' to open the stream

Once done, enter 'x' char in the hyperterminal

 Enter device EUI on your smartphone (8 bytes)

Device EUI is ok

 Enter 'x' to continue with Application EUI

Enter Application EUI (8 byte)

Application EUI is ok

 Enter 'x' to continue with Application key

Enter Application key (16 bytes)

Application Key is ok

 Let see the result !

Device EUI =

-d--e--v--e--u--i--0--0-

Application EUI =

-a--p--p--e--u--i--0--0-
Application key =

-a--a--p--p--l--i--c--a--t--i--o--n--k--e--y--0-
```

F – Check_response()

I made a function to be sure that the RN487x answer AOK, CMD and factory reset. This function will return STATUS_OK if a good answer is detected.