

Develop a Gateway with nodeJS

Pre-requisites

In this Lab you will learn.

- How to set up Raspberry Pi with a linux, nodeJS and node-red as gateway
- How to enable BLE device and communicate with them via command line
- How to create easily a gateway with node-red

Setup Raspbian on the Raspberry Pi




In order to perform the Lab instructions, it is required to program the Raspberry Pi platform that will be used as a Gateway in the next project steps. For this, it is required to download the Raspbian Stretch Lite image from this page <https://www.raspberrypi.org/downloads/raspbian/> or simply use the direct download link https://downloads.raspberrypi.org/raspbian_lite_latest

Flash the Operating System

To program the micro SDCard provided with the Raspberry Pi 3 platform, follow the instructions available at this address : <https://www.raspberrypi.org/documentation/installation/installing-images/README.md>



Once done it is mandatory to change the system parameters to first enable the ssh access. From you windows explorer create a file called ssh on the SDCard drive identified as boot.

 start_db.elf	12/11/2018 17:23	Fichier ELF	3 001 Ko
 start_x.elf	12/11/2018 17:25	Fichier ELF	3 963 Ko
 ssh	27/02/2019 11:39	Fichier	0 Ko

Now you are ready to power up the board and identify its IP address.

Connect platform to secured Wifi

The Raspberry Pi can be configured to automatically connect to a secure wireless network. For this you have to create a file on the SDCard called `wpa_supplicant.conf`. The file should contain the following details:

For Raspbian Jessie:

```
network={
    ssid="YOUR_NETWORK_NAME"
    psk="YOUR_PASSWORD"
    key_mgmt=WPA-PSK
}
```

For Raspbian Stretch:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
network={
    ssid="YOUR_NETWORK_NAME"
    psk="YOUR_PASSWORD"
    key_mgmt=WPA-PSK
}
```

With this file in place, Raspbian will move it in `/etc/wpa_supplicant/` when the system is booted.

Connect platform to unsecured Wifi

The Raspberry Pi can be configured to automatically connect to a unsecure wireless network (wireless access with no password). For this you have to create a file on the SDCard called `wpa_supplicant.conf`. The file should contain the following details:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev # Include this
line for Stretch
network={
    ssid="YOUR_NETWORK_NAME"
    key_mgmt=NONE
}
```

With this file in place, Raspbian will move it in `/etc/wpa_supplicant/` when the system is booted.

Connect over ssh to the platform

To connect to the target platform over ssh you have to use the following command :

```
[user@machine ~]$ ssh pi@<Gateway ip>
```

The requested password is the one by default (**raspberry**) for the Raspbian OS.

Update local platform list

Before starting any system package installation it is mandatory to update the local packages repository.

```
[pi@raspberrypi ~]$ sudo apt-get update
```

```
Get:1 http://raspbian.raspberrypi.org/raspbian stretch InRelease [15.0 kB]
Get:2 http://archive.raspberrypi.org/debian stretch InRelease [25.4 kB]
Get:3 http://raspbian.raspberrypi.org/raspbian stretch/main armhf Packages [11.7 MB]
Get:4 http://archive.raspberrypi.org/debian stretch/main armhf Packages [212 kB]
Get:5 http://archive.raspberrypi.org/debian stretch/ui armhf Packages [42.0 kB]
Get:6 http://raspbian.raspberrypi.org/raspbian stretch/non-free armhf Packages [95.5 kB]
Fetched 12.1 MB in 21s (558 kB/s)
Reading package lists... Done
```

Install platform required packages

Install Docker

The hosted services on the gateway will be contained into docker images, so docker is a required package for the next steps.

```
[pi@raspberrypi ~]$ sudo curl -sSL get.docker.com | sh
```

Add current user to the docker user group.

```
[pi@raspberrypi ~]$ sudo usermod -aG docker pi  
[pi@raspberrypi ~]$ sudo reboot now
```

Install, configure and run NodeRed

The local services on the target will be managed within docker containers to simplify the deployment of the solution and the maintainability of the system (instructions from <https://nodered.org/docs/platforms/docker>).

```
[pi@raspberrypi ~]$ docker pull billounet/node-red-rpi
```

Create a folder called nodes in your home folder.

```
[pi@raspberrypi ~]$ mkdir nodes
```

Create a docker container from the docker image.-

```
[pi@raspberrypi ~]$ docker run -it --net host -v /home/pi/nodes:/data --  
name gatewaynode billounet/node-red-rpi  
  
npm info it worked if it ends with ok  
npm info using npm@3.10.10  
npm info using node@v6.11.0  
npm info lifecycle node-red-docker@1.0.0~prestart: node-red-docker@1.0.0  
npm info lifecycle node-red-docker@1.0.0~start: node-red-docker@1.0.0
```

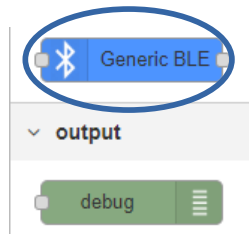
Press Ctrl-p Ctrl-q to close the terminal.

```
# Restart and stop the container  
docker stop gatewaynode  
docker start gatewaynode
```

Bluetooth generic node

Launch node-red from your browser and verify the Generic BLE node is correctly installed. It should appear in the left slide at the bottom of the left.

You can access it in browser at `http://<device IP address>:1880`.



Working with BLE

Some tools are installed with the Bluetooth stack: `hciconfig`, `hcitools`, `gatttool`.

`hciconfig` is the tools as `ifconfig` do for network, for managing Bluetooth interfaces (dongle). Each interface is identified as follow: `hciX`, where X is the interface identified (0,1,2,...). To identify the interfaces that are currently available type:

```
pi@raspberrypi:~$ hciconfig
hci0: Type: BR/EDR  Bus: USB
      BD Address: 5C:F3:70:68:AD:FD  ACL MTU: 1021:8  SCO MTU: 64:1
      UP RUNNING PSCAN
      RX bytes:636 acl:0 sco:0 events:41 errors:0
      TX bytes:1442 acl:0 sco:0 commands:41 errors:0
```

To activate or restart an interface:

```
pi@raspberrypi:~$ sudo hciconfig hciX up # will up
pi@raspberrypi:~$ sudo hciconfig hciX restart # will down and up
```

Once you have the mac address you can communicate with your device.

You can next scan for BLE devices, mac address is before the id of the sensor

```
pi@raspberrypi:~$ sudo hcitool lescan
LE Scan ...
02:80:E1:00:34:12 BlueNRG
02:80:E1:00:34:12 (unknown)
```

Commented [BN1]: Scan ??

1. Connect: before getting data from the remote BLE device you need to connect to.

```
pi@raspberrypi:~$ gatttool -I -b 02:80:E1:00:34:12
[ ] [02:80:E1:00:34:12] [LE]> connect
```

2. Read services and characteristics

```
[CON] [02:80:E1:00:34:12] [LE]> primary
[CON] [02:80:E1:00:34:12] [LE]>
attr handle: 0x0001, end grp handle: 0x0004 uuid: 00001801-0000-1000-
8000-00805f9b34fb
attr handle: 0x0005, end grp handle: 0x000b uuid: 00001800-0000-1000-
8000-00805f9b34fb
attr handle: 0x000c, end grp handle: 0x000e uuid: 01366e80-cf3a-11e1-
9ab4-0002a5d5c51b
attr handle: 0x0013, end grp handle: 0x0019 uuid: 04366e80-cf3a-11e1-
9ab4-0002a5d5c51b
attr handle: 0x001d, end grp handle: 0x001f uuid: 0b366e80-cf3a-11e1-
9ab4-0002a5d5c51b
[CON] [02:80:E1:00:34:12] [LE]> characteristics
[CON] [02:80:E1:00:34:12] [LE]>
handle: 0x0002, char properties: 0x20, char value handle: 0x0003, uuid:
00002a05-0000-1000-8000-00805f9b34fb
handle: 0x0006, char properties: 0x4e, char value handle: 0x0007, uuid:
00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0008, char properties: 0x4e, char value handle: 0x0009, uuid:
00002a01-0000-1000-8000-00805f9b34fb
handle: 0x000a, char properties: 0x0a, char value handle: 0x000b, uuid:
00002a04-0000-1000-8000-00805f9b34fb
handle: 0x000d, char properties: 0x02, char value handle: 0x000e, uuid:
03366e80-cf3a-11e1-9ab4-0002a5d5c51b
handle: 0x0014, char properties: 0x02, char value handle: 0x0015, uuid:
05366e80-cf3a-11e1-9ab4-0002a5d5c51b
handle: 0x0017, char properties: 0x02, char value handle: 0x0018, uuid:
07366e80-cf3a-11e1-9ab4-0002a5d5c51b
handle: 0x001e, char properties: 0x0e, char value handle: 0x001f, uuid:
0c366e80-cf3a-11e1-9ab4-0002a5d5c51b
```



You can refer to Lab01 instructions for the handles descriptions.

3. Read a value (with handle value): here accelerometer value

```
[CON] [02:80:E1:00:34:12] [LE]> char-read-hnd 0x000e
Characteristic value/descriptor: f1 ff 16 00 de 03
```

4. Write a value (with handle value): here led was turned on and turned off

```
[CON] [02:80:E1:00:34:12] [LE]> char-write-cmd 0x001f 01
[CON] [02:80:E1:00:34:12] [LE]> char-write-cmd 0x001f 00
```

Setup Tools on Virtual Machine

VMWare player

1. Before starting, you have to install VMWare player (v6 or higher) on your physical PC regarding its OS (Windows , Linux or MAC) and its CPU architecture (x86 or x64).
2. Then open the provided [Ubuntu 18.04](#) x64 virtual machine.
3. Open the session with login 'gest' and password 'Witekio' (case sensitive and without the quote)

Install MQTT broker



The MQTT broker should already be installed on the Virtual Machine.

For the Lab purpose you need to install a MQTT broker on the Virtual Machine.

1. Add the line below in your `/etc/apt/source.list` file.

```
gest@ubuntuosboxes@osboxes:~$ deb http://ppa.launchpad.net/mosquitto-dev/mosquitto-ppa/ubuntu trusty main
```

2. Then you must update the local packages cache and installed the mosquitto one

```
gest@ubuntuosboxes@osboxes:~$ sudo apt-get update && sudo apt-get install mosquitto mosquitto-clients
```

The mosquitto daemon start at the installation and listen by default on 1880 port on localhost.

For managing its state you can use :

```
gest@ubuntuosboxes@osboxes:~$ sudo service mosquitto stop
gest@ubuntuosboxes@osboxes:~$ sudo service mosquitto start
gest@ubuntuosboxes@osboxes:~$ sudo service mosquitto restart
```

MQTT Spy

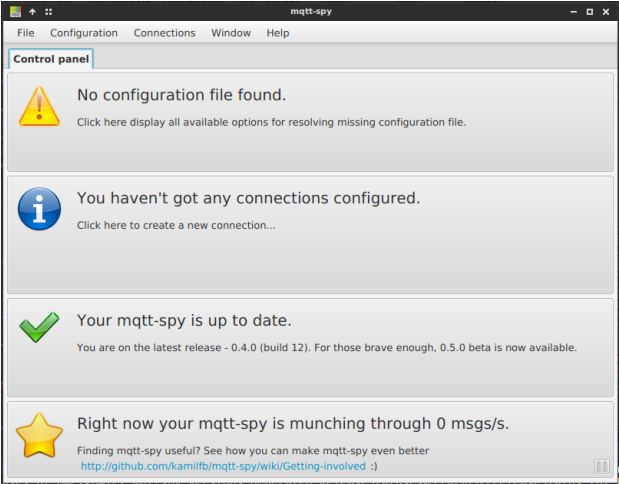
Mqtt spy is a software you can install on your host machine for spy data on the mqtt broker. It will display all traffic on it. You will use it on your host machine. You can download it here :

<https://github.com/kamilfb/mqtt-spy/wiki/Downloads>

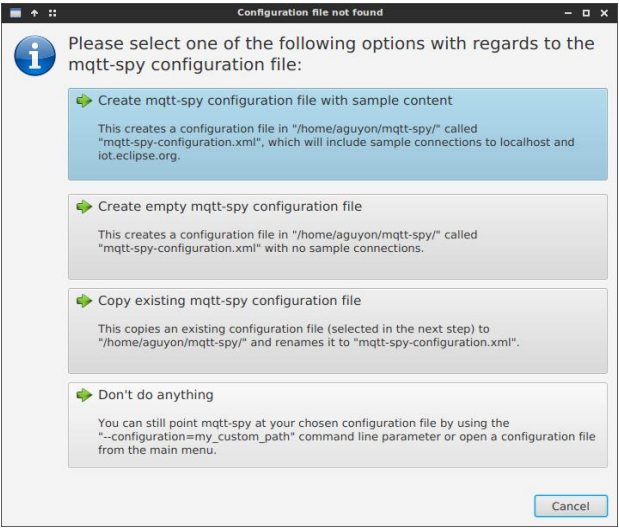
The jar file is in Desktop folder. Launch it with the following command :

```
java -jar mqtt-spy-0.4.0-jar-with-dependencies.jar
```

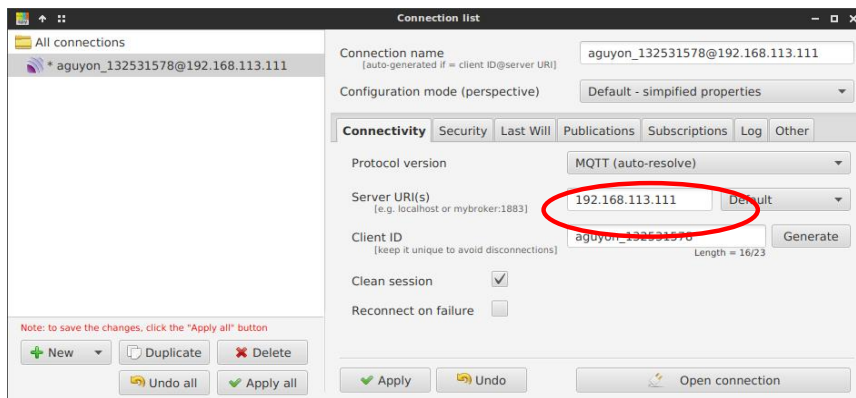

You should see this interface:



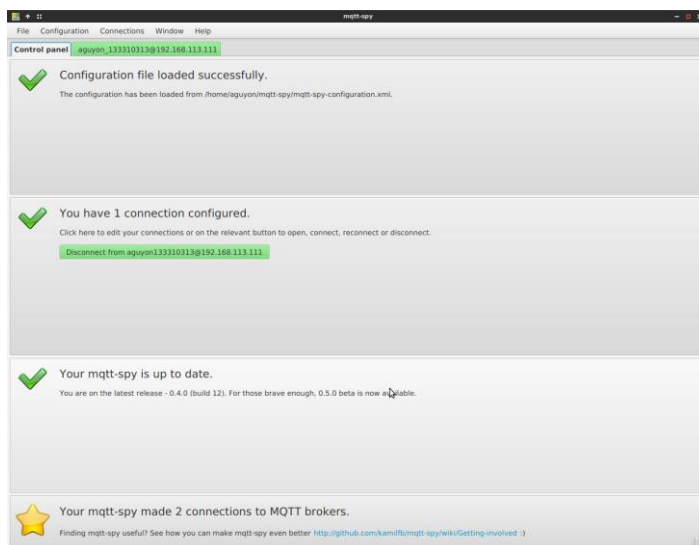
Click on the first big button called "no configuration file found"



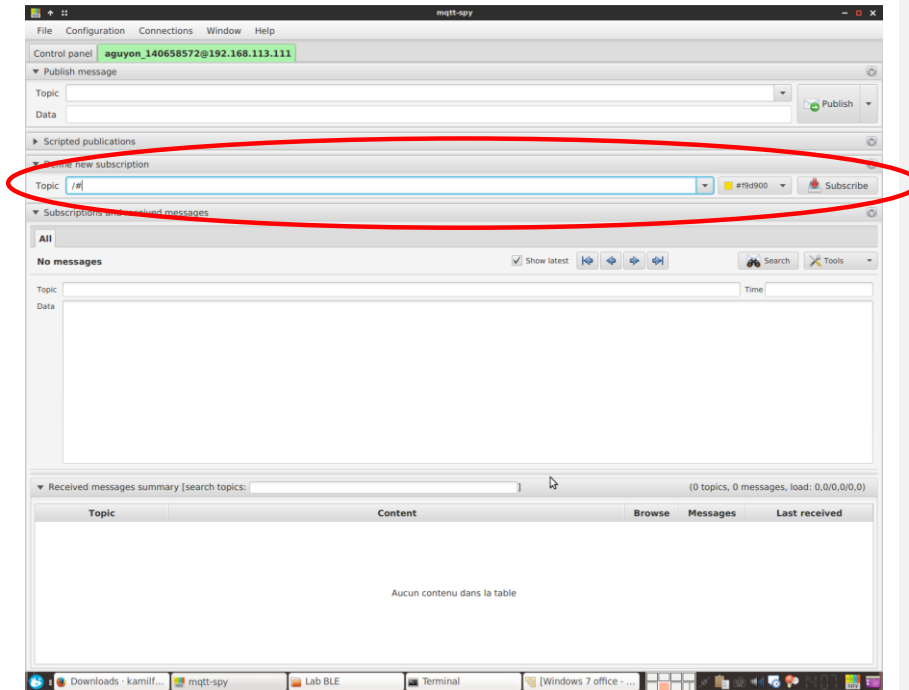
Then click the second button “Create empty mqtt-spy configuration file”. You should come back to the original application screen. You can now add a connection. For this, click on Connections → new connection



Next configure the server URI by setting it to your ip's board (where the mqtt broker is), and click apply and then open connection.



A green tab appeared on the menu screen, click on it.

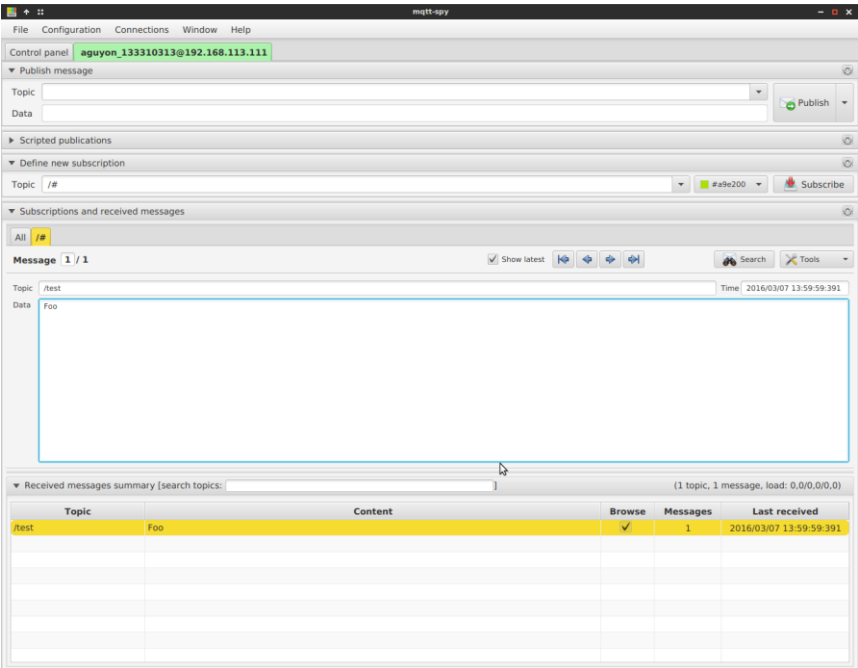


You can now subscribe to topics and publish to them. For subscribing to all topics reproduce information on the previous screenshot and click on subscribe.

On the board publish message by typing:

```
mosquitto_pub -t "/test" -m "Foo"
```

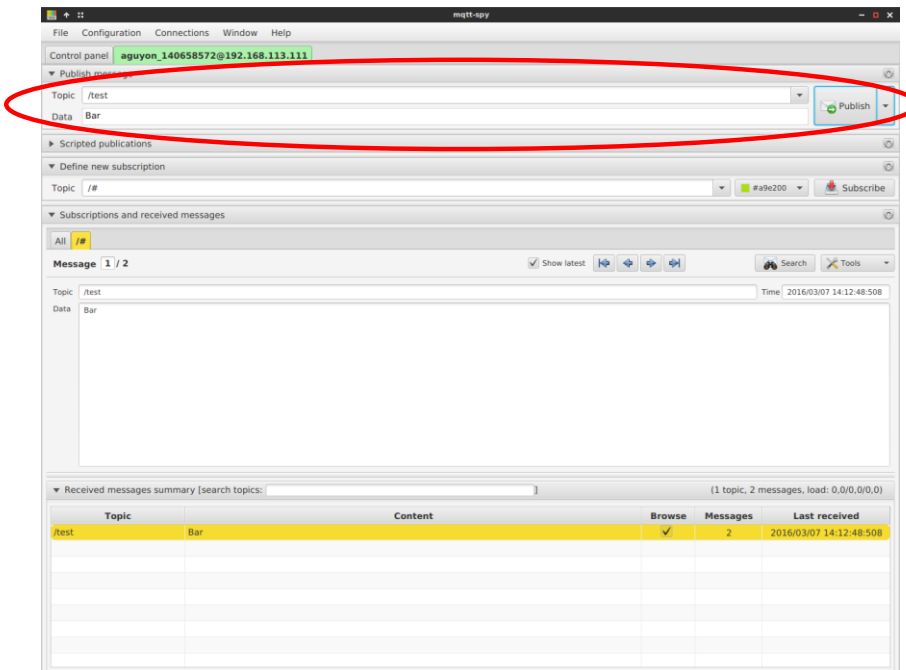
You should see you message payload displayed in the MQTT Spy window, as this:



Next, for subscribing to messages, on the VM launch :

```
gest@ubuntuosboxes@osboxes:~$ mosquitto_sub -v -t "/"#"
```

And publish with mqtt-spy



You should see this on the terminal's board :

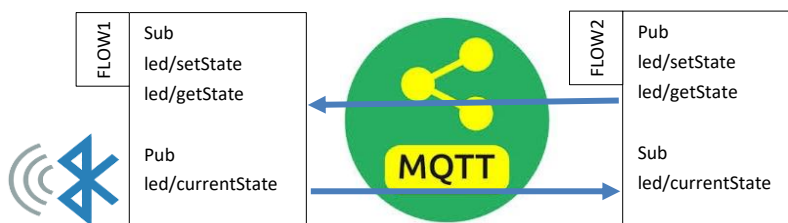
```
gest@ubuntuosboxes@osboxes:~$ mosquitto_sub -v -t "/"#"
/test Bar
```

Gateway Application

Architecture details

The gateway purpose is to provide information to front-end application or databases from sensors. In our case this database is used to retrieve data from the created sensor and make those available to front-end web application which we will develop in Lab 03.

Goal



As MQTT topics are unidirectional, it is required to have a set of dedicated topics for answers to request, like the getState request topic have results answer published on the currentState topic.

The BLE node that will be used in the lab, is a generic BLE node that connect to the STM32 target and will allow to read the various available characteristics and write data.

What is nodeJS ?

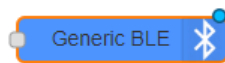
NodeJS is based on V8 Javascript engine by Google. It adds to this web language class and function for doing what system language will do, for instance I/O, websocket, signals, process handling... As this one is still in development, some of the functionalities are unstable and intern implementation can be changed in future version. In our case, we use it mainly because node-red uses it and there is a lot of MIT's licensed existing projects.

Use the node

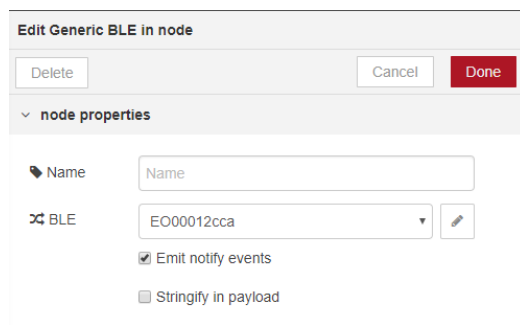
Testing the BLE node

You must close all hci sockets before starting test with node-red because we can only open just one socket on hci device. So stop command gatttool.

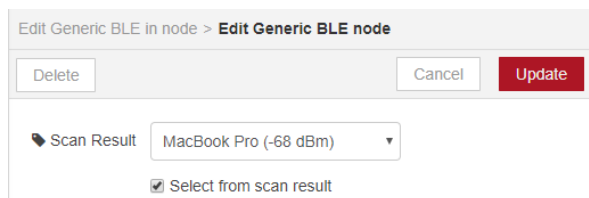
First try the *Generic BLE* node in dragging it in the wide window at the center of node-red. You should have something as below once finished:



You have now to configure the *Generic BLE* node to associate it to your BLE Object. For this, double click on the node.



To select your device, click on the pencil icon located right to the right side of the BLE name field.



Check the Select from scan result check box, to populate the Scan Result list box and select your device from the list.

Edit Generic BLE in node > **Edit Generic BLE node**

Delete Cancel Update

Scan Result BlueNRG (-45 dBm) ▼

☒ Select from scan result

Local Name BlueNRG

MAC 02:80:e1:00:34:13

UUID 0280e1003413

☐ Mute notify events

Operation Timeout (milliseconds, between 300 and 60000 millisecs)

GATT Characteristics

<Unnamed> (03366e80cf3a11e19ab40002a5d5c51b) (Custom Type)	Read
<Unnamed> (05366e80cf3a11e19ab40002a5d5c51b) (Custom Type)	Read
<Unnamed> (07366e80cf3a11e19ab40002a5d5c51b) (Custom Type)	Read
<Unnamed> (0c366e80cf3a11e19ab40002a5d5c51b) (Custom Type)	Write Read WriteWithoutResp

Once selected, then validate and close the sliding element.

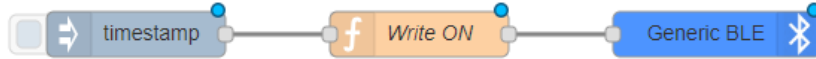
On the flow, the blue dot means that nodes are not deployed yet. For saving and running the flow you must click on **Deploy** button at the top right.

Blink the peripheral led

For this purpose, the BLE node have input. If this input has *characteristic* property value and if this one is set to 0 or 1 it will turn off or turn on the led, respectively.

Rename the current Flow Tab in "Test LED State"

Drag'n drop a function node. Edit the function node (by double clicking on it) in order to modify the provided input message (msg). The goal is to modify the *payload* property with value that will turn on and off the onBoard LED. Then drag and drop an inject node. Connect all this and you should have this below.



The code into the function node can be this one :

```

msg.payload =
{
  "0c366e80cf3a11e19ab40002a5d5c51b": 1
}
return msg;

```

Get the state

Do again the same "architecture" as previous with a inject node and function node to get the current LED State. You should have something like this :



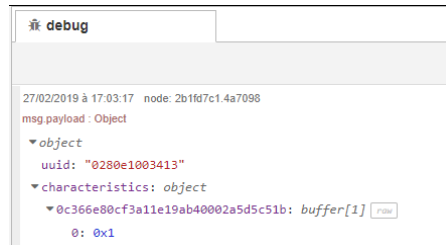
The code into the function node can be this one:

```

msg.topic = "0c366e80cf3a11e19ab40002a5d5c51b"
return msg;

```

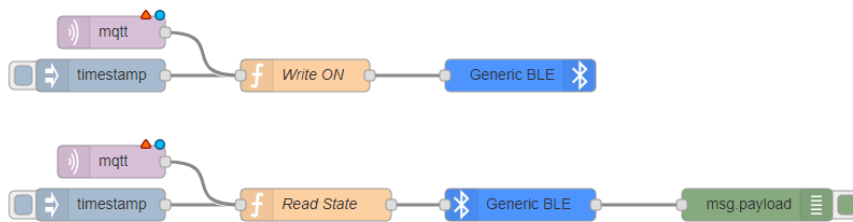
As you can see, if you push the second button, you'll get your LED Status within the debug view.



Transmission (MQTT)

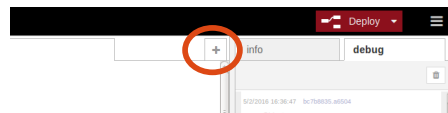
For receiving request via MQTT, you should drag and drop MQTT node in input category. Configure the server (let default option and just click add), and the topic, for instance `led/setState`. Once configuration done, replace your inject node by the MQTT node with the same purpose. And add a second MQTT Node that subscribes to the `led/getState`.

Commented [BN2]: Ajouter une etape pour utiliser MQTTSpy afin de valider que nous envoyons des messages MQTT / \ Ajouter aussi des instructions pour transmettre un message MQTT pour changer l'état de la LED. (deux messages, un pour ON, et un pour OFF)

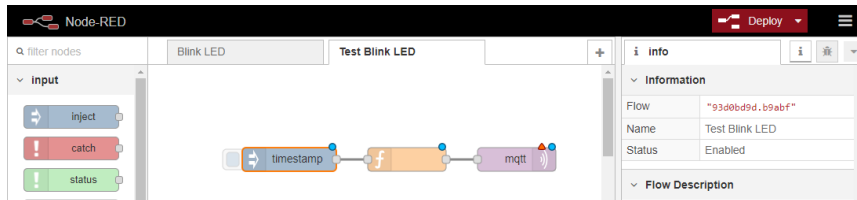


Test flow :

As you can see, we can't test it from node-red for now. We are going to create a new flow to test what we did. To this purpose click on the + button.



Name the new Tab “Test Blink LED”

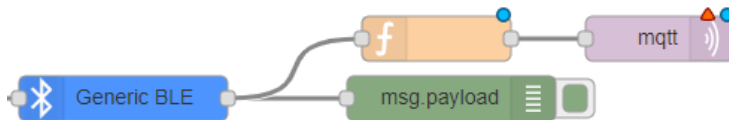


- Add the sequence above with the goal of sending a request over MQTT to turn on the LED.
- Add a second sequence to turn off the LED
- Add a sequence that request the current LED state

You can now deploy your flows and click on buttons for switching the led' state and see the LED's state.

Transmit the led state

To transmit the led' state you can drag and drop output MQTT node in the flow and set it up with this topic *led/currentState*. Connect it at the output of a new function node. This last function node will have as input the output of the BLE node.



This function node aims to filter led state and send it via MQTT, as requested.

- Add a sequence to listen to the LED state transmitted over MQTT

Transmit all sensors data

You are going to develop a new node-red flow. This one, will retrieve from BLE the device sensors values and transmit the values over MQTT. The following information are required:

- Temperature
- Humidity
- Accelerometer

Temperature value

- a. Add all the elements required to retrieve the temperature value from BLE
- b. Add all the nodes required to publish the temperature on the *temp/* topic in decimal
- c. Add a mechanism to trigger a temperature acquisition every 10 secondes

Humidity value

- a. Add all the elements required to retrieve the humidity value from BLE
- b. Add all the nodes required to publish the temperature on the *hum/* topic in decimal
- c. Add a mechanism to trigger a humidity acquisition every 10 secondes

Accelerometer value

- a. Add all the elements required to retrieve the accelerometer value from BLE
- b. Add all the nodes required to publish the accelerometer on the *acc/* topic in JSON format :
{ "x": <value>, "y": <value>, "z": <value> }
- c. Add a mechanism to trigger a temperature acquisition every 1 secondes

Verify with mqtt-spy

Re-open mqtt-spy and you should see all data you sent from the sensor to the cloud machine

- a. Take a screen shot of the application and save it to your repository

Generate more complex data

Average temperature and humidity

Based on the existing data, we are expecting to get an average information of the temperature and humidity values reported every minute.

- a. Add the necessary elements to perform the average computation and transmission in MQTT.
 - Average Temperature in temp/average
 - Average Humidity in humid/average

Tilt detection

Based on the accelerometer values, identify the conditions when the device is heavily checked and raise a dedicated MQTT message.

Gateway connection

When gateway is connected to the MQTT broker, data can be transmitter to listener/reader. We have in our system to detect when device is disconnected to track the current status of the Gateway. Find a solution and implement it.



Do not forget to save you work in the repository.



Do not forget to push ☺