Bluetooth®
SMART

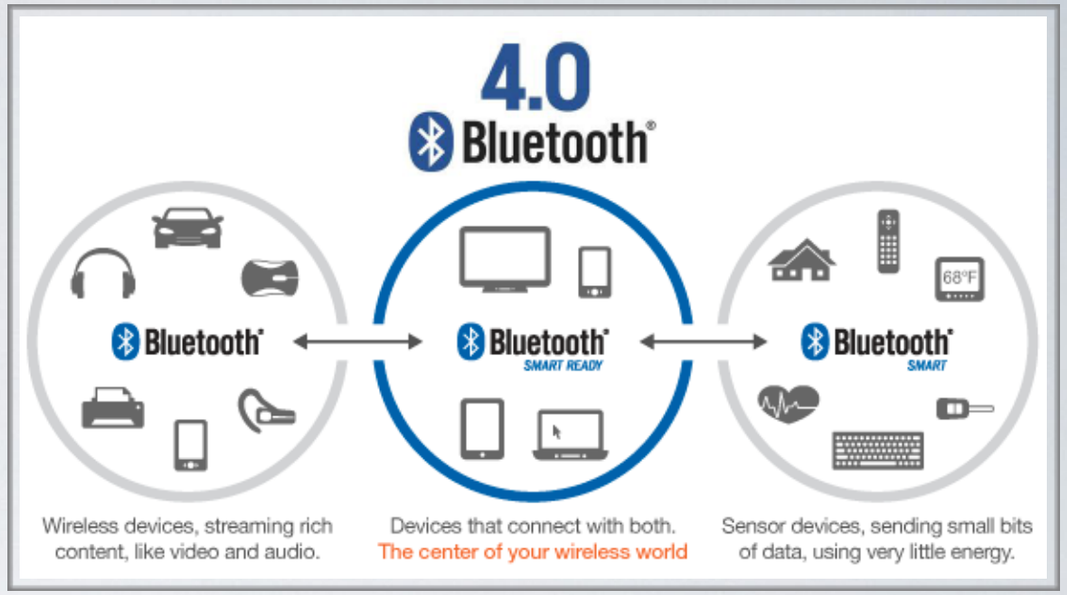
BLUETOOTH LOW ENERGY

IOS, Android examples

iOS

INTRODUCTION

On this presentation you will learn the basics about bluetooth low energy. We're going to learn how to program simple apps in Android and IOS, learn about some modules and how to debug them.



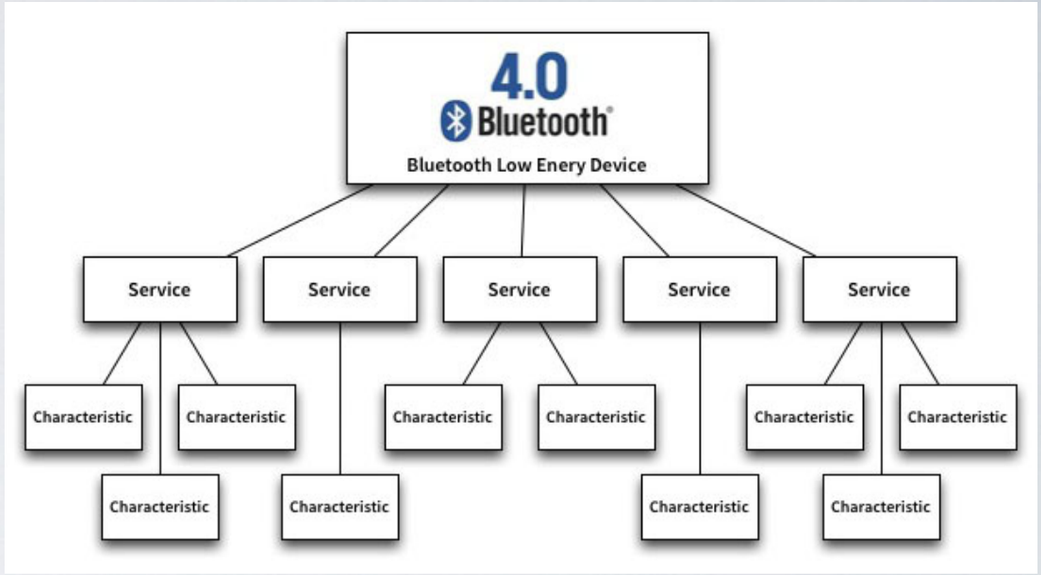
WHAT IT IS

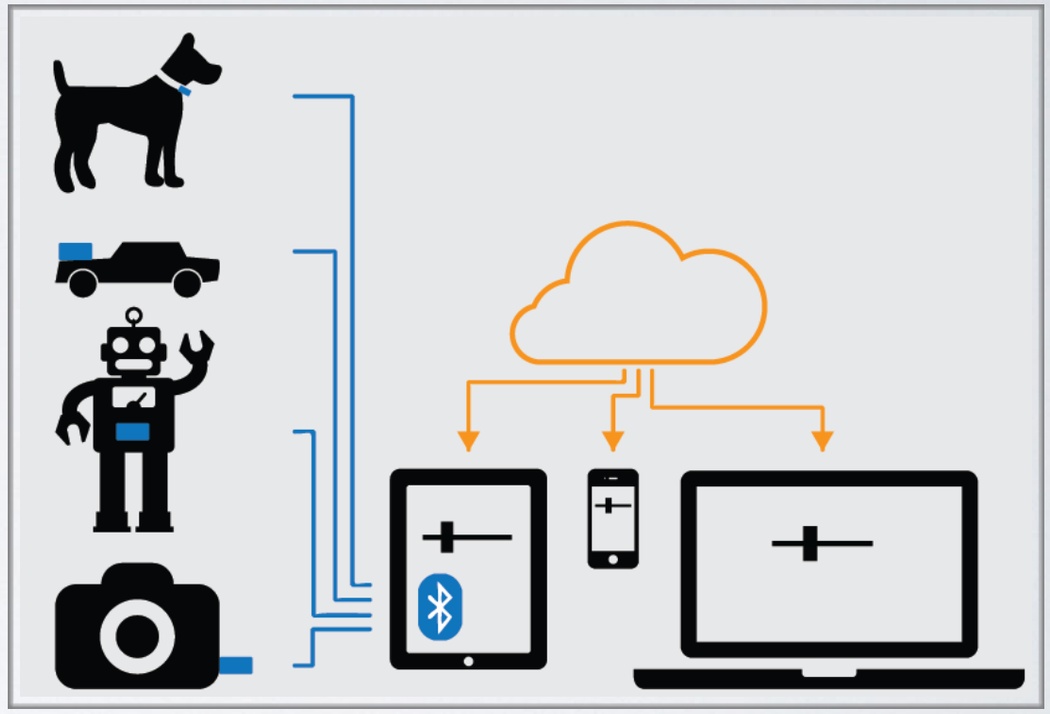
Is a wireless protocol specific for small devices with low bandwith requirements and battery powered devices.

It's throughput is 5Kb to10Kb and it's range from 2m to 5m (30 is possible but uses more battery) and the max ammount of information that you can send at a time is 20 bytes.

Imagine a sensor as a server that provides some services (Battery status), on those services you have some Characteristics (Voltage, current, etc...)

Your phone will be the client, that can read/write information on those characteristics.





Some BLE Terminology

- Device is a BLE **peripheral** (advertiser, protocol slave) running a **server** (generating data).
- Phone is a BLE **central** (protocol initiator) running a **client** (receiving data).

- A device offers a **profile** which has a **service** which consists of **characteristics** (data values) and **descriptors**. They are accessed via a **UUID or handle**.
- A **heart rate** profile might include
  - **heart rate** service with **60** for **HRMeasurement** at UUID **0x2A37.**
  - **battery** service might offer **97** with "**% full**" at UUID **0x2A19**

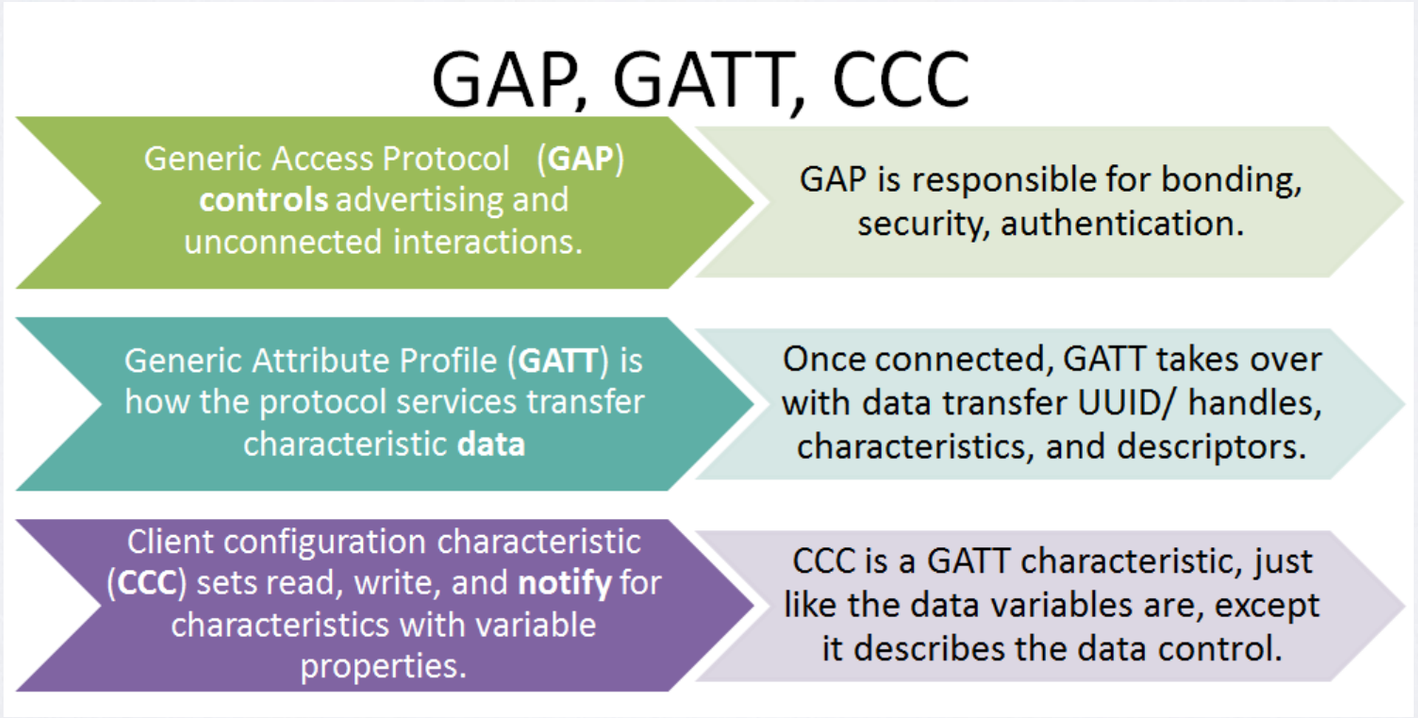
GAP

GAP is an acronym for the Generic Access Profile, and it controls connections and advertising in Bluetooth. GAP is what makes your device visible to the outside world, and determines how two devices can (or can't) interact with each other.

GAP defines various roles for devices, but the two key concepts to keep in mind are Central devices and Peripheral devices.

- Peripheral devices are small, low power, resource contrained devices that can connect to a much more powerful central device. Peripheral devices are things like a heart rate monitor, a BLE enabled proximity tag, etc.

- Central devices are usually the mobile phone or tablet that you connect to with far more processing power and memory.



GATT

GATT is an acronym for the Generic Attribute Profile, and it defines the way that two Bluetooth Low Energy devices transfer data back and forth using concepts called Services and Characteristics. It makes use of a generic data protocol called the Attribute Protocol (ATT), which is used to store Services, Characteristics and related data in a simple lookup table using 16-bit IDs for each entry in the table.

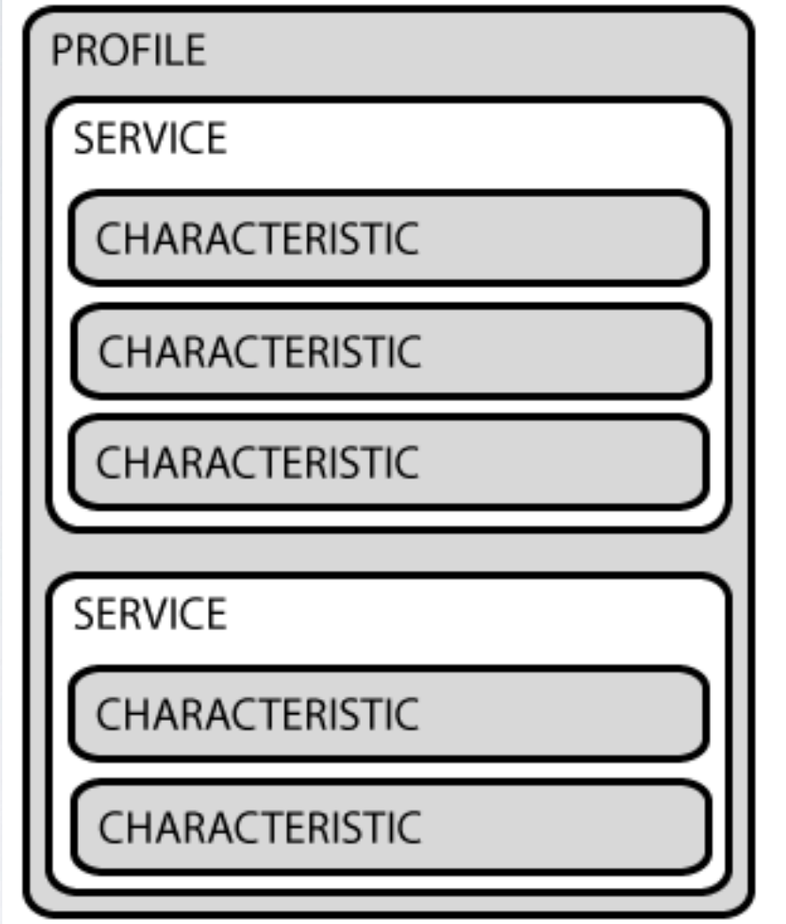
GATT comes into play once a dedicated connection is established between two devices, meaning that you have already gone through the advertising process governed by GAP.

The most important thing to keep in mind with GATT and connections is that connections are exclusive. What is meant by that is that a BLE peripheral can only be connected to one central device (a mobile phone, etc.) at a time! As soon as a peripheral connects to a central device, it will stop advertising itself and other devices will no longer be able to see it or connect to it until the existing connection is broken.

SERVICE AND CHARACTERISTICS

Imagine a BLE sensor as a information server that gives some kind of information, for instance the Battery Service, provides some characteristics (ex: Battery voltage level, current, time of recharge, etc..)

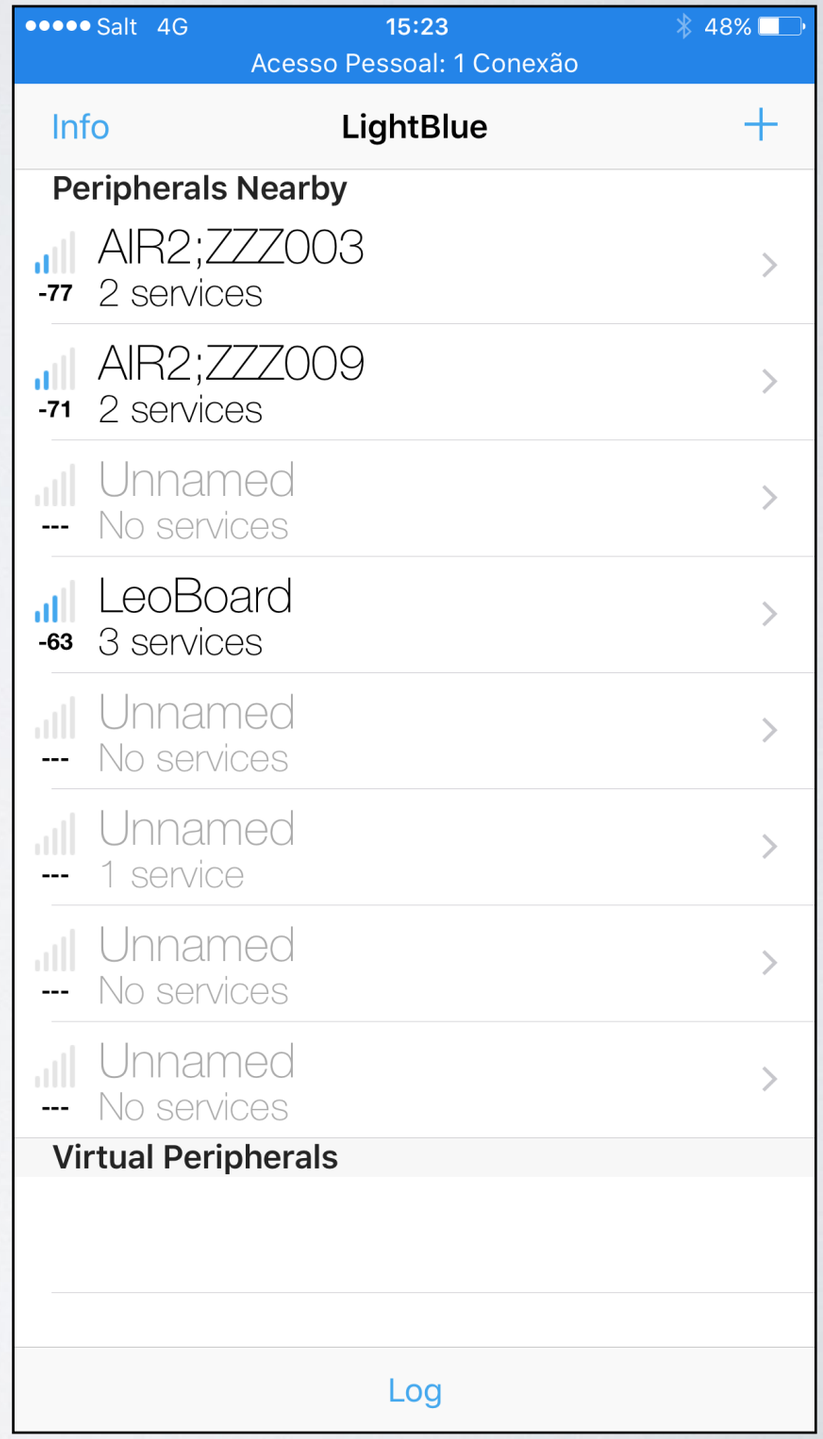
So basically a service will have one or more characteristics that can be read or written by another device (ie the phone)



LIGHTBLUE

It's an IOS app, that used on bluetooth low energy development, it has features that help both the firmware development as the IOS app that will communicate with a device. Some features:

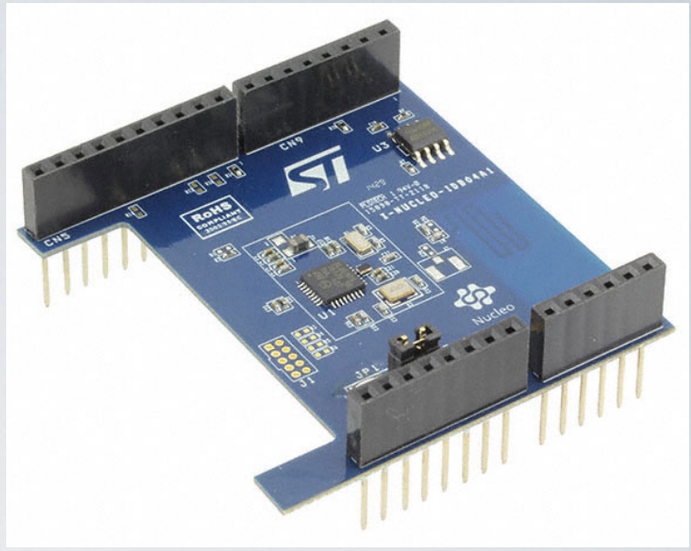
- Show all BLE devices on the phone range
- List all services and characteristics of a device
- Allow you to write or read a characteristic
- Clone a BLE sensor and make the phone become a BLE sensor, so you can develop from another phone and test.
- Create a device with some characteristic and services



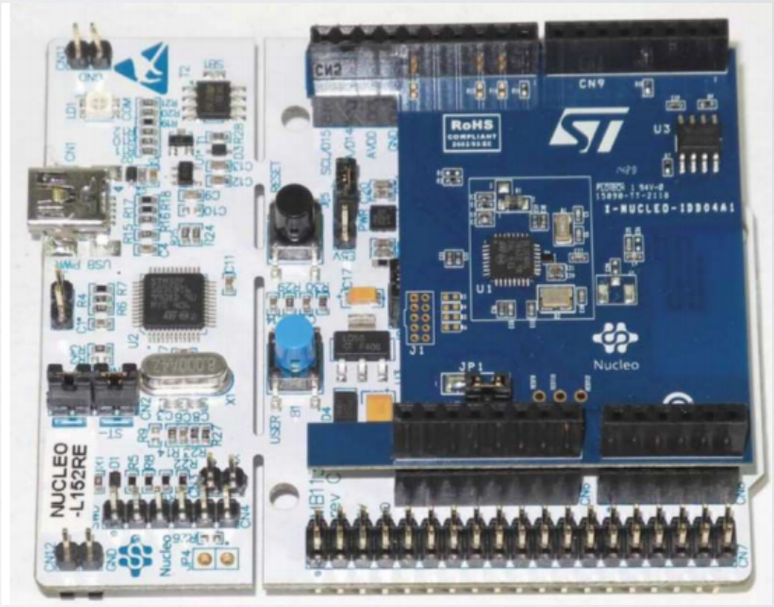
HARDWARE TEST

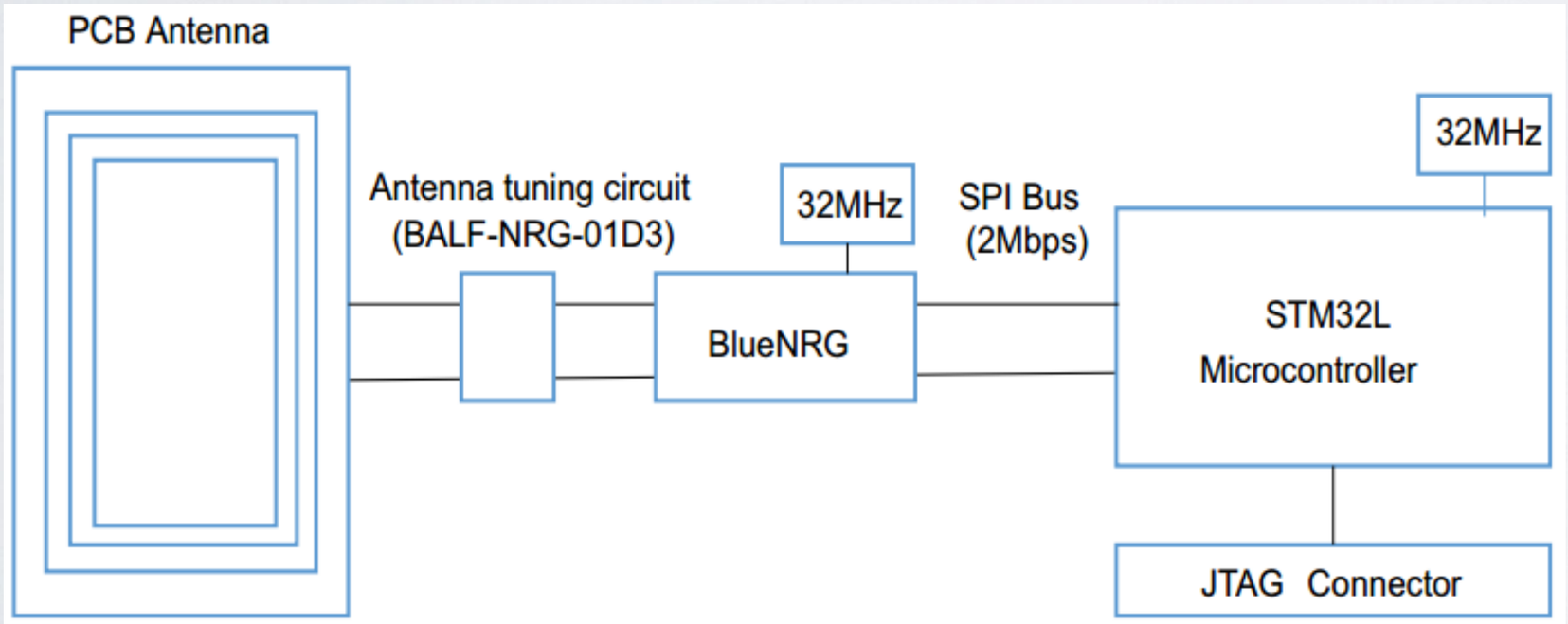
Basically we need the following hardware:

- BLE Nucleo shield(X-NUCLEO-IDB04A1)
- Nucleo STM32F4 (NUCLEO-F411RE, NUCLEO-F401RE)
- Iphone
- Android

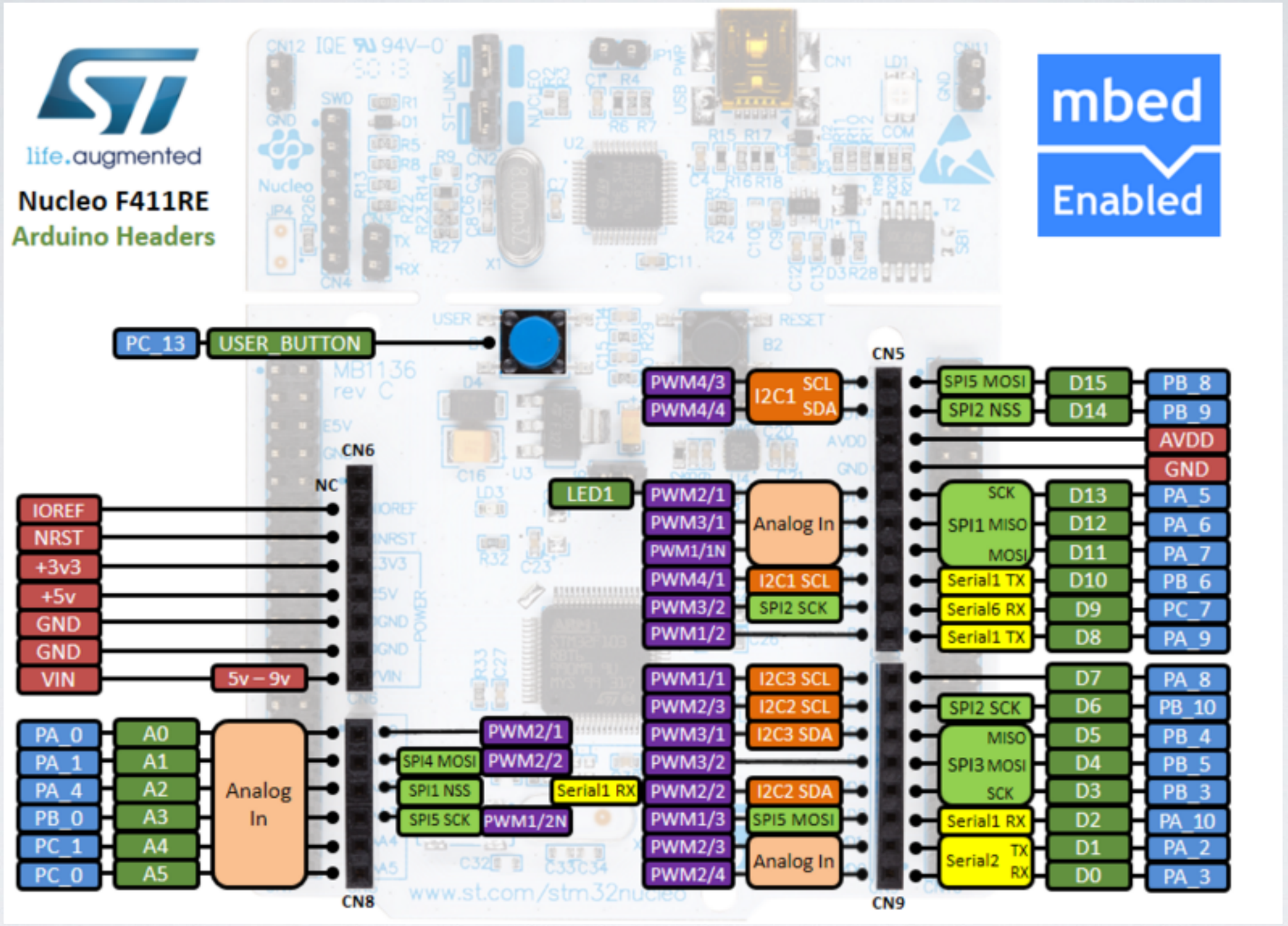


The BLE module and the nucleo board communicate with a SPI interface

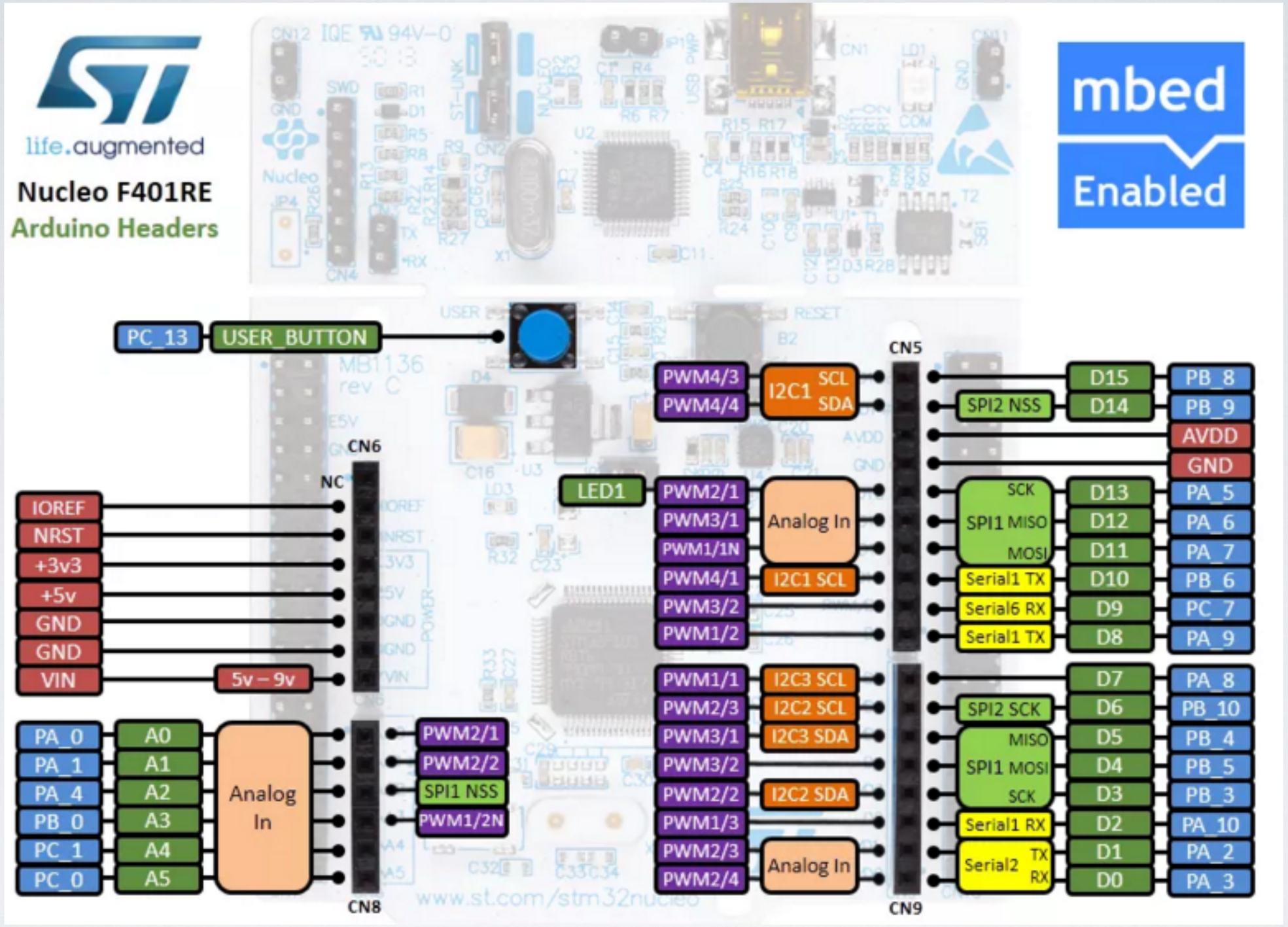




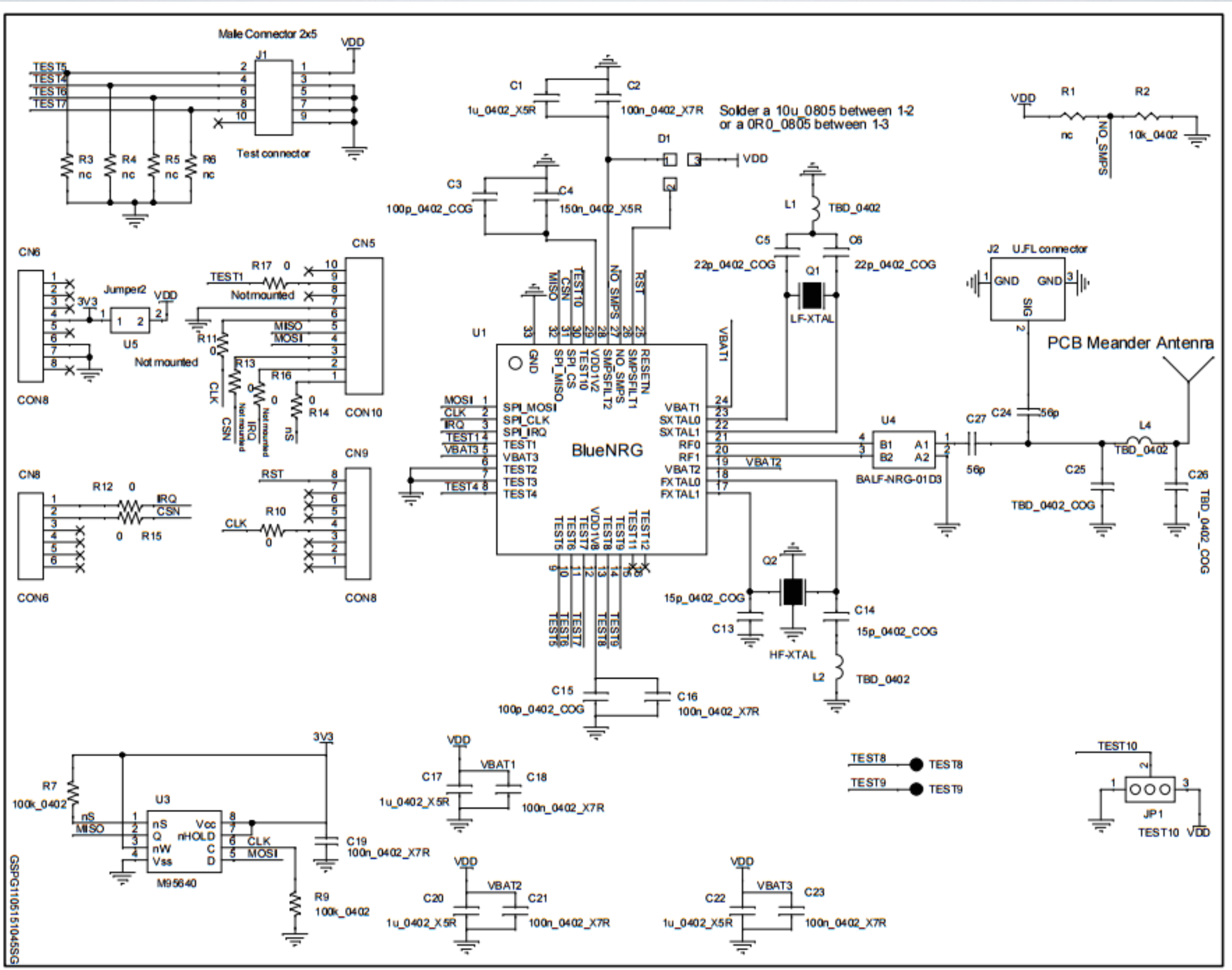
NUCLEO F411RE ARDUINO PINOUT



NUCLEO F401RE ARDUINO PINOUT



MODULE BLE SCHEMATIC



PATCHING TO ARDUINO OR F411RE

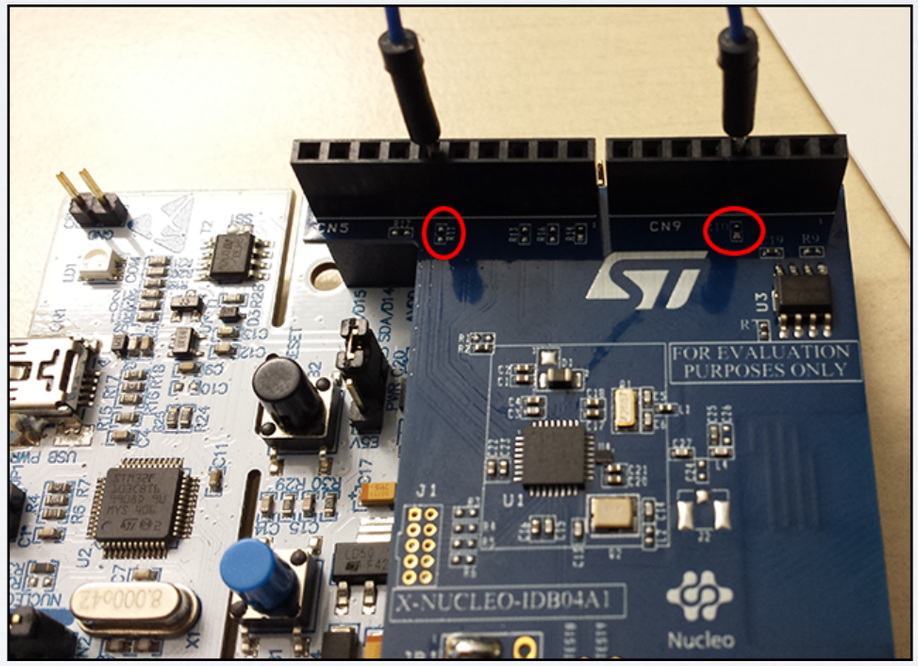
main.cpp | x_nucleo_idb0xa1_targets.h

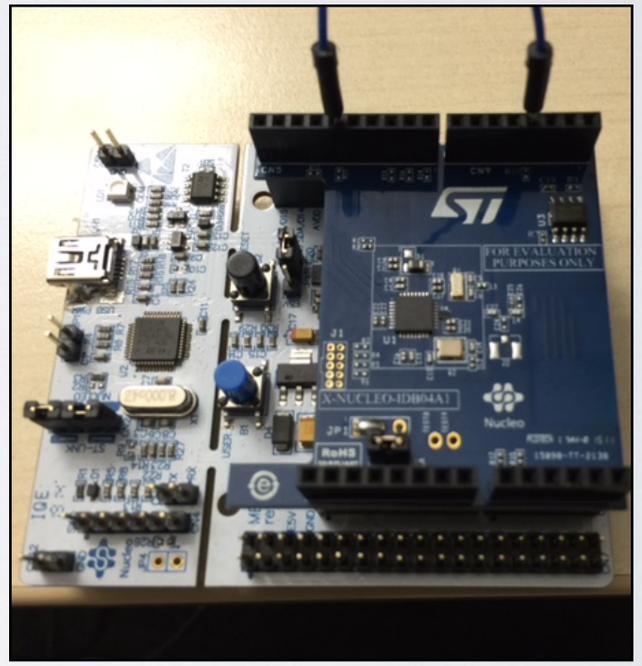
```
36   *
37   ****************************************
38   */
39
40 /* Define to prevent from recursive inclusion
41 #ifndef _X_NUCLEO_IDB0XA1_TARGETS_H_
42 #define _X_NUCLEO_IDB0XA1_TARGETS_H_
43
44 /*** SPI ***/
45 /* Use Arduino I2C Connectors */
46 #define IDB0XA1_PIN_SPI_MOSI   (D11)
47 #define IDB0XA1_PIN_SPI_MISO   (D12)
48 #define IDB0XA1_PIN_SPI_nCS    (A1)
49 #define IDB0XA1_PIN_SPI_RESET  (D7)
50 #define IDB0XA1_PIN_SPI_IRQ    (A0)
51
52 /* NOTE: Define beyond macro if you want to c
53          modified version of the X_NUCLEO_IDB
54          which pin 'D13' (rather than the sta
55          in order to provide the SPI serial c
56       Expansion boards modified in this way al
57       any Arduino-compliant base board.
58 */
59 #define IDB0XA1_D13_PATCH
60
61 #if defined(IDB0XA1_D13_PATCH)
62 #define IDB0XA1_PIN_SPI_SCK    (D13)
63 #else // !defined(IDB0XA1_D13_PATCH)
64 #define IDB0XA1_PIN_SPI_SCK    (D3)
65 #endif // !defined(IDB0XA1_D13_PATCH)
```

In order to use the BLE nucleo board with arduino or the nucleo f411re board we need to change the 0-ohm resistor R10 with the R11, or short with a cable these pins. Just remember that there is also a LED1 on this pin (D13) on the nucleo boards. (So don't use this led)

This pin is responsible for the SPI clock.

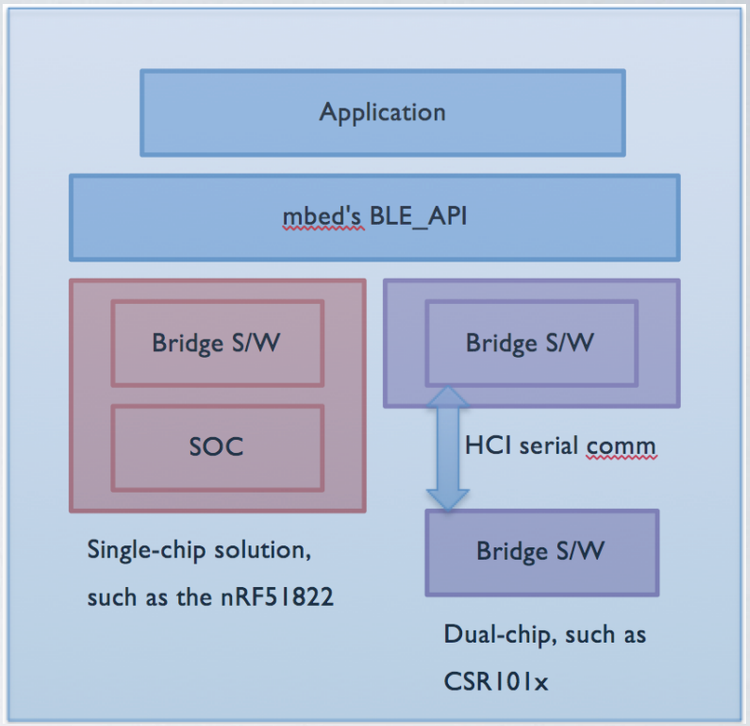
After this we also need to change the header file on the left, this code is part of the HW driver of the BLE nucleo board (X_NUCLEO_IDB0XA1)

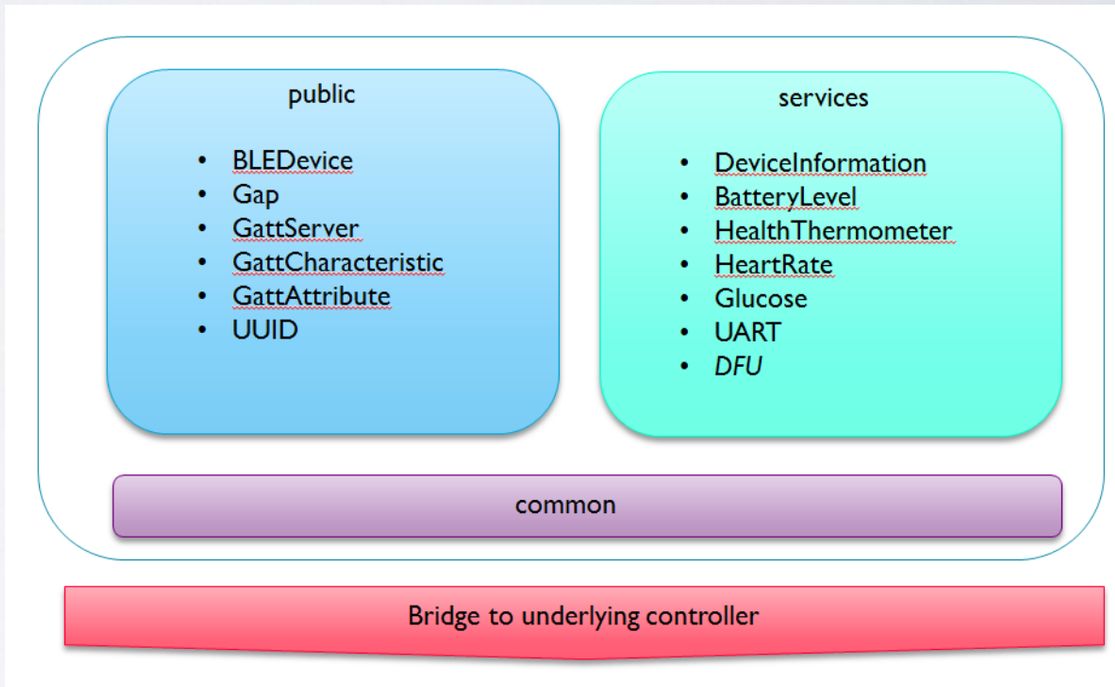




BLE AND MBED

On the Mbed ecosystem, the development of BLE applications is handled by the BLE_API library, which define a SW layer to make the life of the firmware developer easier, this library offer some hardware support for some boards like the RedBearLab BLE Nano, but it also gives a abstraction layer(Bridge S/W) that allow other vendors to port their BLE modules.



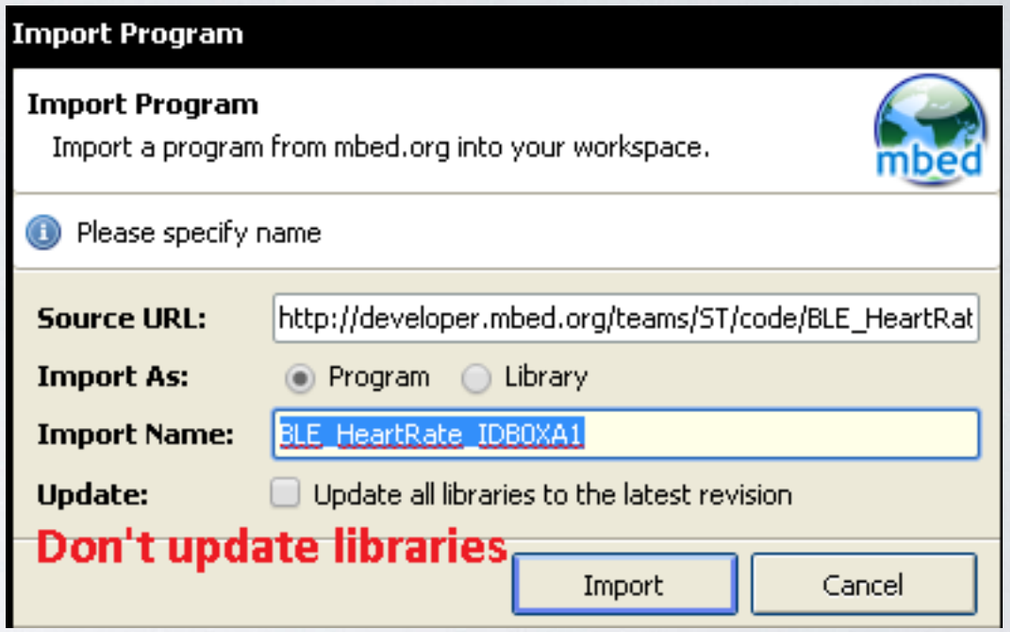


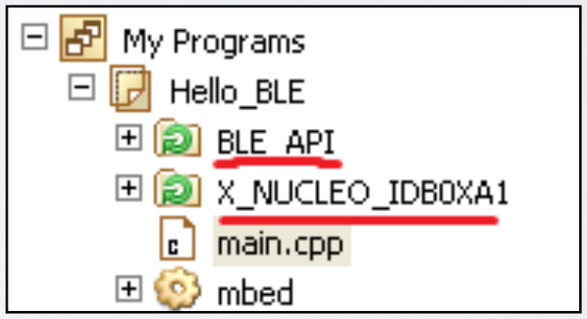
CREATING A BLE PROJECT ON MBED

To give an example on what we can do we will create a BLE periphereal that will serve the following characterisitcs:

- Give the battery level
- Give characteristic to turn on/ off a led

We can start by importing a example project from the BLE module mbed site, basically this will configure a project with BLE_API and the HW layer of the BlueNRG module





CODE, PART 1

```
#include "mbed.h"
#include "ble/BLE.h"
#include "ble/services/BatteryService.h"
#include "ble/services/DeviceInformationService.h"

BLE  ble;

// If you apply the D13 pin patch you cannot use this led anymore
DigitalOut led1(LED1);

const static char     DEVICE_NAME[] = "LeoBoard";
// Has the battery service, device information, and custom service "FFFF" (led control)
static const uint16_t uuid16_list[] = {GattService::UUID_BATTERY_SERVICE,GattService::UUID_DEVICE_INFORMATION_SERVICE, 0xFFFF};

// Custom service and characteristics UUIDS
uint16_t customServiceUUID  = 0xA000;
uint16_t readCharUUID       = 0xA001;
uint16_t writeCharUUID      = 0xA002;

// Set Up custom Characteristics (Package max size is 20bytes)
static uint8_t readValue[20] = {0};
ReadOnlyArrayGattCharacteristic<uint8_t, sizeof(readValue)> readChar(readCharUUID, readValue);
static uint8_t writeValue[20] = {0};
WriteOnlyArrayGattCharacteristic<uint8_t, sizeof(writeValue)> writeChar(writeCharUUID, writeValue);
// Set up custom service
GattCharacteristic *characteristics[] = {&readChar, &writeChar};
```

Here we simply add the BLE_API and the services needed (Battery, Device Information, and Custom for Led control), then we define the custom service and characteristic for the led control.

On the last part we bind the characteristic to variables (Notice the 20 byte array limit)

CODE, PART 2

```
int main(void)
{
    uint8_t batteryLevel = 50; printf("Initialize BLE\r\n");
    ble.init();

    ble.gap().onDisconnection(disconnectionCallback);
    ble.gattServer().onDataWritten(writeCharCallback);

    // Setup battery service
    BatteryService batteryService(ble, batteryLevel);
    // Device Information
    DeviceInformationService deviceInfo(ble, "StarkIndustires", "Quadcopter", "SN1", "hw-rev1", "fw-rev1", "BetaVer");
    // add our custom service
    ble.addService(customService);

    // Setup advertising. Indicate that we only support bluetooth low energy, and our services
    ble.gap().accumulateAdvertisingPayload(GapAdvertisingData::BREDR_NOT_SUPPORTED | GapAdvertisingData::LE_GENERAL_DISCOVERABLE);
    ble.gap().accumulateAdvertisingPayload(GapAdvertisingData::COMPLETE_LIST_16BIT_SERVICE_IDS, (uint8_t *)uuid16_list, sizeof(uuid16_list));
    ble.gap().accumulateAdvertisingPayload(GapAdvertisingData::GENERIC_COMPUTER);
    ble.gap().accumulateAdvertisingPayload(GapAdvertisingData::COMPLETE_LOCAL_NAME, (uint8_t *)DEVICE_NAME, sizeof(DEVICE_NAME));
    ble.gap().setAdvertisingType(GapAdvertisingParams::ADV_CONNECTABLE_UNDIRECTED);
    ble.gap().setAdvertisingInterval(1000); /* 1000ms */
    ble.gap().startAdvertising();

    // Main loop
    while (1) {
        ble.waitForEvent(); // this will return upon any system event (such as an interrupt or a ticker wakeup)
        // the magic battery processing
        batteryLevel++;
        if (batteryLevel > 100) {
            batteryLevel = 20;
        }
        batteryService.updateBatteryLevel(batteryLevel);
    }
}
```

CODE, PART 3

```
void disconnectionCallback(const Gap::DisconnectionCallbackParams_t *params) {
    printf("Start advertising\r\n");
    ble.gap().startAdvertising(); // restart advertising
}

// Handle writes to writeCharacteristic
void writeCharCallback(const GattWriteCallbackParams *params) {
    // check to see what characteristic was written, by handle
    if(params->handle == writeChar.getValueHandle()) {
        // toggle LED if only 1 byte is written
        if(params->len == 1) {
            led1 = params->data[0];
            (params->data[0] == 0x00) ? printf("led off\n\r") : printf("led on\n\r"); // print led toggle
        } else {
            printf("Data received: length = %d, data = 0x",params->len);
            for(int x=0; x < params->len; x++) {
                printf("%x", params->data[x]);
            }
            printf("\n\r");
        }
        // Update the readChar with what the phone write
        ble.updateCharacteristicValue(readChar.getValueHandle(), params->data,params->len);
    }
}
```

BIBLIOGRAPHY

http://www.raywenderlich.com/85900/arduino-tutorial-integrating-bluetooth-le-ios-swift

http://hatemfaheem.blogspot.ch/2014/12/how-would-you-scan-for-nearby-ble.html

http://anasimtiaz.com/?p=201

https://github.com/microbuilder/IntroToBLE

https://labs.ideo.com/2012/07/02/bluetooth-4-0-as-a-prototyping-tool/

http://www.eetimes.com/document.asp?doc_id=1278927

http://www.eetimes.com/document.asp?doc_id=1278966

https://drive.google.com/folderview?id=0B2RH2qnIKMIEfIFPazRhR3d6VWIIaVhsQUR1ZXJxMW5HNTBfeHcwMjZMTnE3bTRfdmFyQVU&usp=sharing

https://www.adafruit.com/products/2269

http://greatscottgadgets.com/ubertoothone/

http://blog.cozybit.com/how-to-crack-bluetooth-le-security-using-crackle/

http://www.st.com/st-web-ui/static/active/en/resource/technical/document/data_brief/DM00114523.pdf

http://www.st.com/st-web-ui/static/active/en/resource/sales_and_marketing/presentation/product_presentation/X-NUCLEO-IDB04A1_quick_guide.pdf

https://developer.mbed.org/blog/entry/Bluetooth-LE-example-on-mbed/

https://developer.mbed.org/forum/team-63-Bluetooth-Low-Energy-community/topic/5262/

BIBLIOGRAPHY

http://ble-intros.readthedocs.org/en/latest/GettingStarted/URIBeacon/

https://developer.mbed.org/components/X-NUCLEO-IDB04A1/

https://developer.mbed.org/teams/Bluetooth-Low-Energy/code/BLE_HeartRate/

https://developer.mbed.org/teams/ST/code/X_NUCLEO_IDB0XA1/wiki/Homepage

http://www.carminenoviello.com/2015/03/09/shield-bluenrg-stm32-nucleo/

https://github.com/cnoviello/stm32-nucleof4/tree/master/stm32-nucleof4-bluenrg-ex1

http://www.st.com/st-web-ui/static/active/en/resource/technical/document/data_brief/DM00114523.pdf

https://developer.mbed.org/teams/Bluetooth-Low-Energy/

https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

https://developer.mbed.org/teams/Bluetooth-Low-Energy/wiki/Architecture-of-mbed-BLE-solution

https://developer.mbed.org/teams/Bluetooth-Low-Energy/code/

https://developer.mbed.org/teams/ST/code/X_NUCLEO_IDB0XA1/

https://vimeo.com/80058454

http://legacy.punchthrough.com/bean/

http://fr.slideshare.net/programmarchy/ble-talk

BIBLIOGRAPHY

http://fr.slideshare.net/StanleyChang13/ble-overview-andimplementation

http://fr.slideshare.net/yeokm1/introduction-to-bluetooth-low-energy?related=1