Mise en oeuvre d’un périphérique Bluetooth BLE sous Qt/Android

Thierry Vaira <tvaira@free.fr>
2018-2019 (rev. 1.1)
tvaira.free.fr

Magic Blue LED

C’est une ampoule LED E27 à intensité variable pilotable en Bluetooth :

Achat :

- GearBest : 11.21 EUR + Livraison
- Amazon : 19.75 EUR + Livraison

Applications gratuites :

- LED Magic Blue (Android)
- LED Magic Blue (iOS)

Pré-requis

Lire :

- Bluetooth BLE
- Qt pour Android
- Cours QML
- Mise en oeuvre du Bluetooth BLE sous Qt

Documentation :

- Qt Bluetooth LE
- Qt Bluetooth Low Energy Scanner Example

Bluetooth LE et Qt5

Pour utiliser l'API Qt Bluetooth, il faudra commencer par ajouter le module dans le fichier de projet `.pro` :

```
QT += bluetooth
```

L'API fournit les classes Qt suivantes :

- QBluetoothDeviceDiscoveryAgent : pour découvrir les périphériques Bluetooth à proximité
- QBluetoothDeviceInfo : des informations (nom et adresse) sur un périphérique Bluetooth
- QLowEnergyController : un controleur BLE local
- QLowEnergyService : un service BLE
- QLowEnergyCharacteristic : une caractéristique d'un service BLE
- QLowEnergyAdvertisingData : pour les données à diffuser lors de l'*advertising*

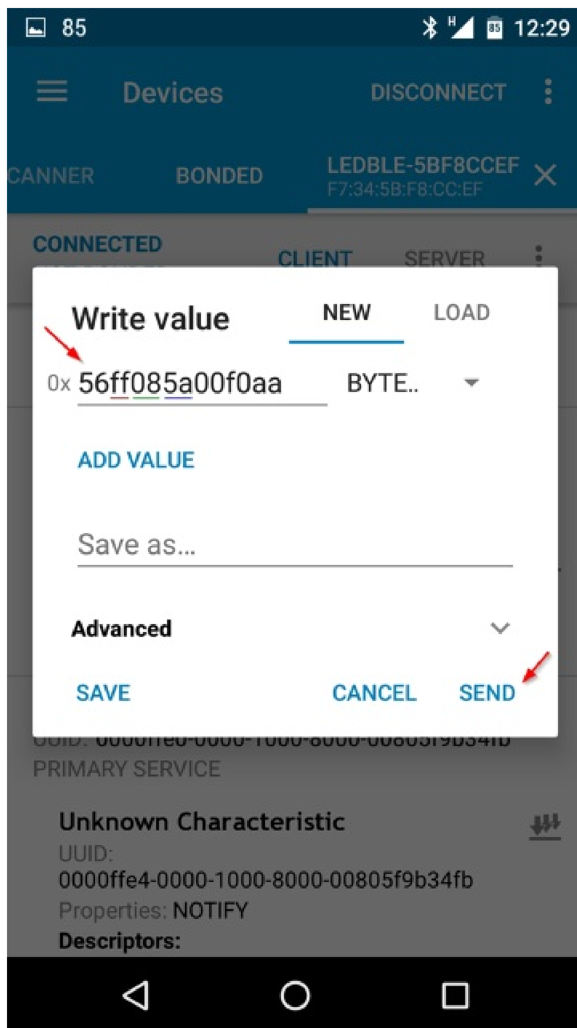
- QLowEnergyAdvertisingParameters : pour les paramètres utilisés pour l'*advertising*
- QLowEnergyServiceData : pour configurer les données de service du GATT
- QLowEnergyCharacteristicData : pour configurer les données des caractéristiques de service du GATT
- QLowEnergyDescriptorData : pour fournir un descripteur "Configuration des caractéristiques du client"

Communication Bluetooth LE et Magic Blue LED

Reverse engineering :

- Reverse engineering an RGB bulb
- Reverse Engineering the Bulbs

Test :



nRF Connect for Mobile :

Devices

DISCONNECT

SCANNER BONDED ADVERTISER LEDBLE-78630D96 F8:1D:78:63:0D:96

CONNECTED
NOT BONDED

CLIENT SERVER

Generic Access
UUID: 0x1800
PRIMARY SERVICE

Unknown Service
UUID: 0000fff0-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE

This service is empty.

Unknown Service
UUID: 0000ffe5-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE

Unknown Characteristic
UUID: 0000ffe9-0000-1000-8000-00805f9b34fb
Properties: WRITE, WRITE NO RESPONSE

Unknown Service
UUID: 0000ffe0-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE

Unknown Characteristic
UUID: 0000ffe4-0000-1000-8000-00805f9b34fb
Properties: NOTIFY
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: (0x) 00-00

Wireless by Nordic

Outil `gatttool` :

```
$ sudo gatttool -i hci0 -b F8:1D:78:63:0D:96 --primary
attr handle = 0x0001, end grp handle = 0x0007 uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle = 0x0008, end grp handle = 0x0008 uuid: 0000fff0-0000-1000-8000-00805f9b34fb
attr handle = 0x0009, end grp handle = 0x000b uuid: 0000ffe5-0000-1000-8000-00805f9b34fb
attr handle = 0x000c, end grp handle = 0xffff uuid: 0000ffe0-0000-1000-8000-00805f9b34fb

$ sudo gatttool -i hci0 -b F8:1D:78:63:0D:96 --characteristics
handle = 0x0002, char properties = 0x02, char value handle = 0x0003, uuid = 00002a00-0000-1000-80
00-00805f9b34fb
handle = 0x0004, char properties = 0x02, char value handle = 0x0005, uuid = 00002a01-0000-1000-80
00-00805f9b34fb
handle = 0x0006, char properties = 0x02, char value handle = 0x0007, uuid = 00002a04-0000-1000-80
00-00805f9b34fb
handle = 0x000a, char properties = 0x0c, char value handle = 0x000b, uuid = 0000ffe9-0000-1000-80
00-00805f9b34fb
handle = 0x000d, char properties = 0x10, char value handle = 0x000e, uuid = 0000ffe4-0000-1000-80
00-00805f9b34fb

$ sudo gatttool -i hci0 -b F8:1D:78:63:0D:96 --char-write-req --value=56ff000000f0aa --handle=0x0
00b --listen
```

Capture : wireshark-gatttool.pcapng

Wireshark :

```
281 13.583789   localhost ()   remote ()   ATT       19 Sent Write Command, Handle: 0x000b (Unknown)
282 13.591114   controller     host        HCI_EVT    8 Rcvd Number of Completed Packets

▸ Frame 281: 19 bytes on wire (152 bits), 19 bytes captured (152 bits)
▸ Bluetooth
▸ Bluetooth HCI H4
▸ Bluetooth HCI ACL Packet
▾ Bluetooth L2CAP Protocol
    Length: 10
    CID: Attribute Protocol (0x0004)
▾ Bluetooth Attribute Protocol
  ▸ Opcode: Write Command (0x52)
    Handle: 0x000b (Unknown)
    Value: 568f333000f0aa

0000  02 07 00 0e 00 0a 00 04  00 52 0b 00 56 8f 33 30   ........ .R..V.30
0010  00 f0 aa                                            ...
```

Capture : btsnoop_hci_led.pcap

Application Qt pour Android

L'application Qt est scindée en deux parties :

- une interface graphique en QML et,
- une partie pour gérer le Bluetooth BLE et la communication sous la forme de deux classes C++.

On utilisera le kit de développement Android for armeabi-v7a (GCC 4.9, Qt 5.10.1 for Android armv7).

Le fichier `.pro` intègre les modules `qml`, `quick` et `bluetooth` :

```
TEMPLATE = app

QT += qml quick bluetooth
CONFIG += c++11

SOURCES += main.cpp \
```

```
    ClientBLE.cpp \
    appareilble.cpp

RESOURCES += qml.qrc

HEADERS += \
    ClientBLE.h \
    appareilble.h
```

Pour la partie C++, on aura deux classes :

- `ClientBLE` : pour rechercher les périphériques Magic Blue Led et communiquer avec
- `AppareilBLE` : pour fournir le nom et l'adresse (sous la forme de propriétés Qt) d'un périphérique Magic Blue Led

Le fichier `main.cpp` instanciera un objet `ClientBLE`, l'associera au document QML puis chargera le tout :

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>

#include "ClientBLE.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;

    ClientBLE *clientBLE = new ClientBLE();
    engine.rootContext()->setContextProperty("ClientBLE", clientBLE);

    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));

    return app.exec();
}
```

Dans la classe `ClientBLE`, il faut commencer par découvrir les périphériques Magic Blue Led à proximité. Pour cela il faut créer une instance de [QBluetoothDeviceDiscoveryAgent](), fixer un *timeout* pour la recherche, connecter les signaux/slots et appeler `start()` :

```
m_discoveryAgent = new QBluetoothDeviceDiscoveryAgent();

m_discoveryAgent->setLowEnergyDiscoveryTimeout(5000);

// Slot pour la recherche d'appareils BLE
connect(m_discoveryAgent, SIGNAL(deviceDiscovered(QBluetoothDeviceInfo)), this, SLOT(ajouterAppar
eil(QBluetoothDeviceInfo)));
connect(m_discoveryAgent, SIGNAL(error(QBluetoothDeviceDiscoveryAgent::Error)), this, SLOT(recher
cheErreur(QBluetoothDeviceDiscoveryAgent::Error)));
connect(m_discoveryAgent, SIGNAL(finished()), this, SLOT(rechercheTerminee()));

m_discoveryAgent->start(QBluetoothDeviceDiscoveryAgent::LowEnergyMethod);
```

Le slot `ajouterAppareil()` aura pour rôle de filtrer les périphériques Magic Blue Led puis de les stocker dans une liste d'objets `AppareilBLE`.

```
void ClientBLE::ajouterAppareil(const QBluetoothDeviceInfo &info)
{
    // Bluetooth Low Energy ?
    if (info.coreConfigurations() & QBluetoothDeviceInfo::LowEnergyCoreConfiguration)
    {
        // Magic Blue Led ?
        if(info.name().startsWith("LEDBLE"))
        {
            AppareilBLE *a = new AppareilBLE(info.name(), info.address().toString(), this);
            m_devices.append(a);
            m_appareilDetecte = true;
        }
    }
}
```

Le slot `rechercheTerminee()` est déclenchée à la fin de la recherche et émet les signaux vers l'interface QML. Le slot `rechercheErreur()` sera lui utilisé en cas d'erreur : la plus probable étant la désactivation du Bluetooth sur le terminal mobile Android.

```
void ClientBLE::rechercheTerminee()
{
    m_etatRecherche = false;
    emit recherche();
    emit detecte();
    emit appareilsUpdated();
}

void ClientBLE::rechercheErreur(QBluetoothDeviceDiscoveryAgent::Error erreur)
{
    m_etatRecherche = false;
    emit recherche();
    emit detecte();
    emit appareilsUpdated();
}
```

Pour interfacer la recherche assurée par la classe et la partie QML, on aura besoin :

- de trois propriétés : `appareilDetecte` , `etatRecherche` et `listeAppareils`
    - associées à trois attributs : `m_appareilDetecte` , `m_etatRecherche` et `m_devices`
    - et notifiées par trois signaux : `detecte()` , `recherche()` et `appareilsUpdated()`
- et de deux méthodes appelables par la GUI QML : `rechercher()` et `arreter()`

La déclaration partielle de la classe `ClientBLE` pour la partie recherche de périphériques Magic Blue Led :

```
class ClientBLE : public QObject
{
    Q_OBJECT
    Q_PROPERTY(bool appareilDetecte MEMBER m_appareilDetecte NOTIFY detecte)
    Q_PROPERTY(bool etatRecherche MEMBER m_etatRecherche NOTIFY recherche)
    Q_PROPERTY(QVariant listeAppareils READ getAppareils NOTIFY appareilsUpdated)

public:
    ClientBLE();
    ~ClientBLE();
```

```
    Q_INVOKABLE void rechercher();
    Q_INVOKABLE void arreter();
    QVariant getAppareils();

protected slots:
    void ajouterAppareil(const QBluetoothDeviceInfo&);
    void rechercheTerminee();
    void rechercheErreur(QBluetoothDeviceDiscoveryAgent::Error);

private:
    QList<QObject*> m_devices; // liste de périphériques Magic Blue
    QBluetoothDeviceDiscoveryAgent  *m_discoveryAgent; // pour la recherche des périphériques
    bool m_etatRecherche; // état de la recherche
    bool m_appareilDetecte; // indique si au moins un périphérique a été détecté

signals:
    void recherche();
    void detecte();
    void appareilsUpdated();
};
```

La GUI QML est construite autour d'un élément `Window` et une disposition avec des `ColumnLayout` et `RowLayout`. La recherche est gérée par un `ToggleButton` (on fera de même pour la connexion, qui n'est possible que si au moins un périphérique Magic Blue Led est détecté). Comme on a fixé un *timeout* de 5 s pour la détection des périphériques, on ajoutera un `BusyIndicator` (qu'on utilisera aussi pour la connexion).

La définition partielle de la partie QML dédiée à la recherche de périphériques Magic Blue Led :

```
import QtQuick 2.9
import QtQuick.Window 2.2
import QtQuick.Controls 2.2
import QtQuick.Extras 1.4
import QtQuick.Layouts 1.3

Window {
    id: window
    title: qsTr("Magic Blue LED BLE")
    visible: true
    width: Screen.desktopAvailableWidth
    height: Screen.desktopAvailableHeight
    color: "#272822"

    property bool recherche: ClientBLE.etatRecherche
    onRechercheChanged: {
        if (ClientBLE.etatRecherche)
        {
            boutonRecherche.text = "Arrêter";
            boutonRecherche.checked = true;
            indicateur.running = true;
            message.text = qsTr("Recherche en cours")
        }
        else
        {
            boutonRecherche.text = "Rechercher";
            boutonRecherche.checked = false;
            indicateur.running = false;
            message.text = qsTr("Recherche finie")
        }
```

```
    }

    property bool detecte: ClientBLE.appareilDetecte
    onDetecteChanged: {
        if (!ClientBLE.appareilDetecte)
        {
            boutonConnexion.enabled = false;
            message.text = qsTr("Aucun Magic Blue Led trouvé !")
        }
    }

    ...

    ToggleButton {
            id: boutonRecherche;
            width: Screen.desktopAvailableWidth/6
            height: Screen.desktopAvailableHeight/6
            text: ClientBLE.etatRecherche ? qsTr("Arrêter") : qsTr("Rechercher");
            checked: ClientBLE.etatRecherche;
            onClicked: {
                indicateur.running = true;
                ClientBLE.etatRecherche ? ClientBLE.arreter() : ClientBLE.rechercher();
            }
     }

     ...
}
```

On utilisera un `ListView`, associée à la `QList`, pour afficher le nom et l'adresse des Magic Blue détectés :

```
ListView {
        id: listeAppareils
        width: parent.width
        anchors { fill: parent; margins: 2 }
        spacing: 5
        model: ClientBLE.listeAppareils

         // l'affichage des éléments de la liste
        delegate: Rectangle {
            id: appareil
            anchors.horizontalCenter: parent.horizontalCenter
            height: 80
            width: parent.width/4
            color: "lightsteelblue"
            border.width: 2
            border.color: "#cecece"
            radius: 5

            MouseArea {
                anchors.fill: parent
                onClicked: {
                    listeAppareils.currentIndex = index;
                    message.text = model.modelData.nom;
                    boutonConnexion.enabled = true;
                }
            }
```

```
            // le nom et l'adresse des périphériques Bluetooth
            Text {
                id: deviceName
                font.pointSize: 20
                anchors.horizontalCenter: parent.horizontalCenter
                color: "#A6A6A6"
                horizontalAlignment: Text.AlignHCenter
                elide: Text.ElideMiddle
                width: parent.width
                wrapMode: Text.Wrap
                text: model.modelData.nom
                anchors.top: parent.top
                anchors.topMargin: 5
            }
            Text {
                id: deviceAddress
                font.pointSize: 20*0.7
                anchors.horizontalCenter: parent.horizontalCenter
                color: "#A6A6A6"
                horizontalAlignment: Text.AlignHCenter
                elide: Text.ElideMiddle
                width: parent.width
                wrapMode: Text.Wrap
                text: model.modelData.adresseMAC
                anchors.bottom: appareil.bottom
                anchors.bottomMargin: 5
            }
        }
}
```

Si l'utilisateur clique sur un des périphériques Magic Blue détecté, son nom sera affiché dans un `Text` et on pourra s'y connecter afin de le piloter.

Le principe de communication avec un périphérique Bluetooth BLE utilisé ensuite dans l'application a déjà été décrite dans cet exemple : [Mise en oeuvre du Bluetooth BLE sous Qt](#).

L'API Qt permet de créer des connexions avec des périphériques, de découvrir leurs services, ainsi que de lire et d'écrire des données stockées sur le périphérique à partir d'une instance de la classe [QLowEnergyController](#).

La méthode `connecterAppareil(const QString &adresseServeur)` est appelée au moment de la demande connexion et on lui passe en argument l'adresse MAC du périphérique BLE à joindre. On connecte les *slots* dont on a besoin : connexion/déconnexion et les services à découvrir. On démarre avec `connectToDevice()`.

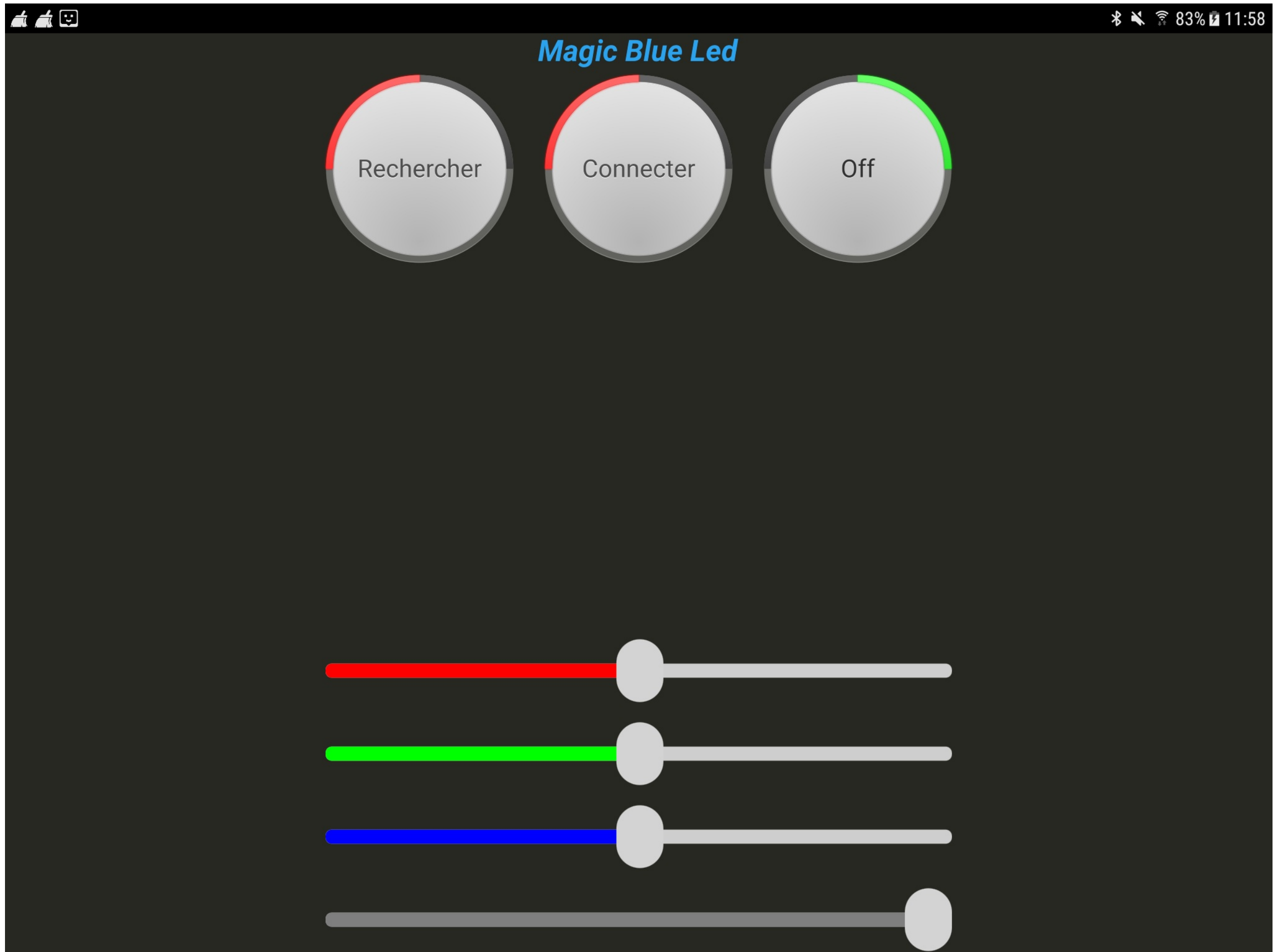
Ensuite, la méthode `ajouterService()` va être déclenchée pour chaque service découvert. On appelle alors `createServiceObject()` en lui passant l'UUID du service pour créer une instance de ce service. On appelle notre méthode `connecterService()` pour découvrir les caractéristiques du service. On connecte les *slots* dont on a besoin et on lance la découverte avec `discoverDetails()`. Pour chaque caractéristique découverte, le *slot* `serviceDetailsDiscovered()` sera déclenché.

Pour piloter la Magic Blue Led, on fournit des méthodes `write()` :

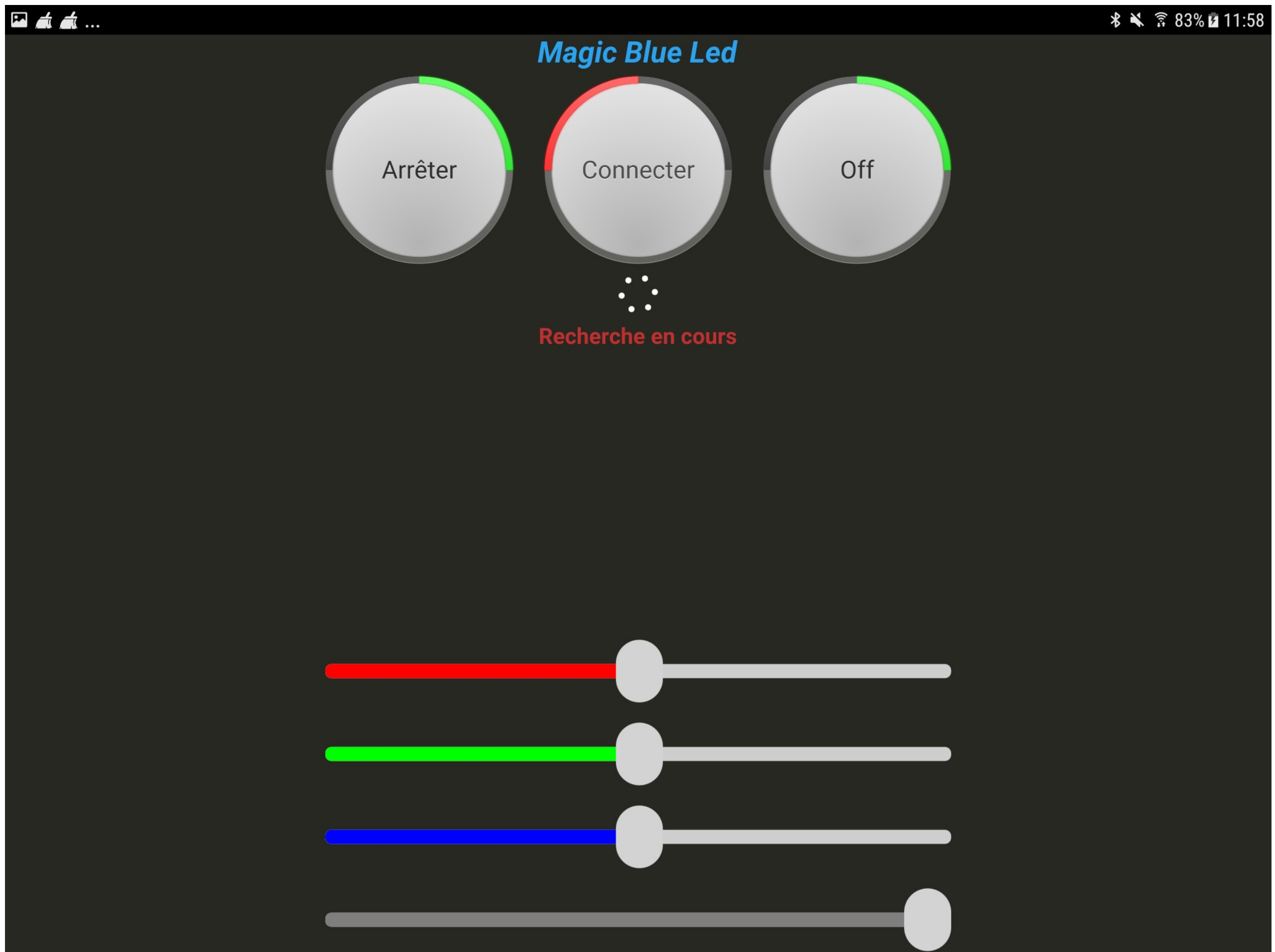
```
void ClientBLE::write(const QByteArray &data)
{
    if(m_service && m_characteristic.isValid())
    {
        if (m_characteristic.properties() & QLowEnergyCharacteristic::Write)
```

```
        {
            if(data.length() <= MAX_SIZE)
            {
                m_service->writeCharacteristic(m_characteristic, data, QLowEnergyService::WriteWi
thoutResponse);
            }
        }
    }
}

// RGB
void ClientBLE::write(int rouge, int vert, int bleu, int white/*=0*/)
{
    Q_UNUSED(white)
    QByteArray datas(7, 0);

    datas[0] = 0x56;
    datas[1] = static_cast<char>(rouge); //RR
    datas[2] = static_cast<char>(vert); // GG
    datas[3] = static_cast<char>(bleu); // BB
    datas[4] = 0x00; // WW
    datas[5] = 0xf0;
    datas[6] = 0xaa;

    write(datas);
}

// On/Off
void ClientBLE::write(bool etat)
{
    QByteArray datas(3, 0);

    datas[0] = 0xcc;
    if(etat)
        datas[1] = 0x23;
    else
        datas[1] = 0x24;
    datas[2] = 0x33;

    write(datas);
}
```

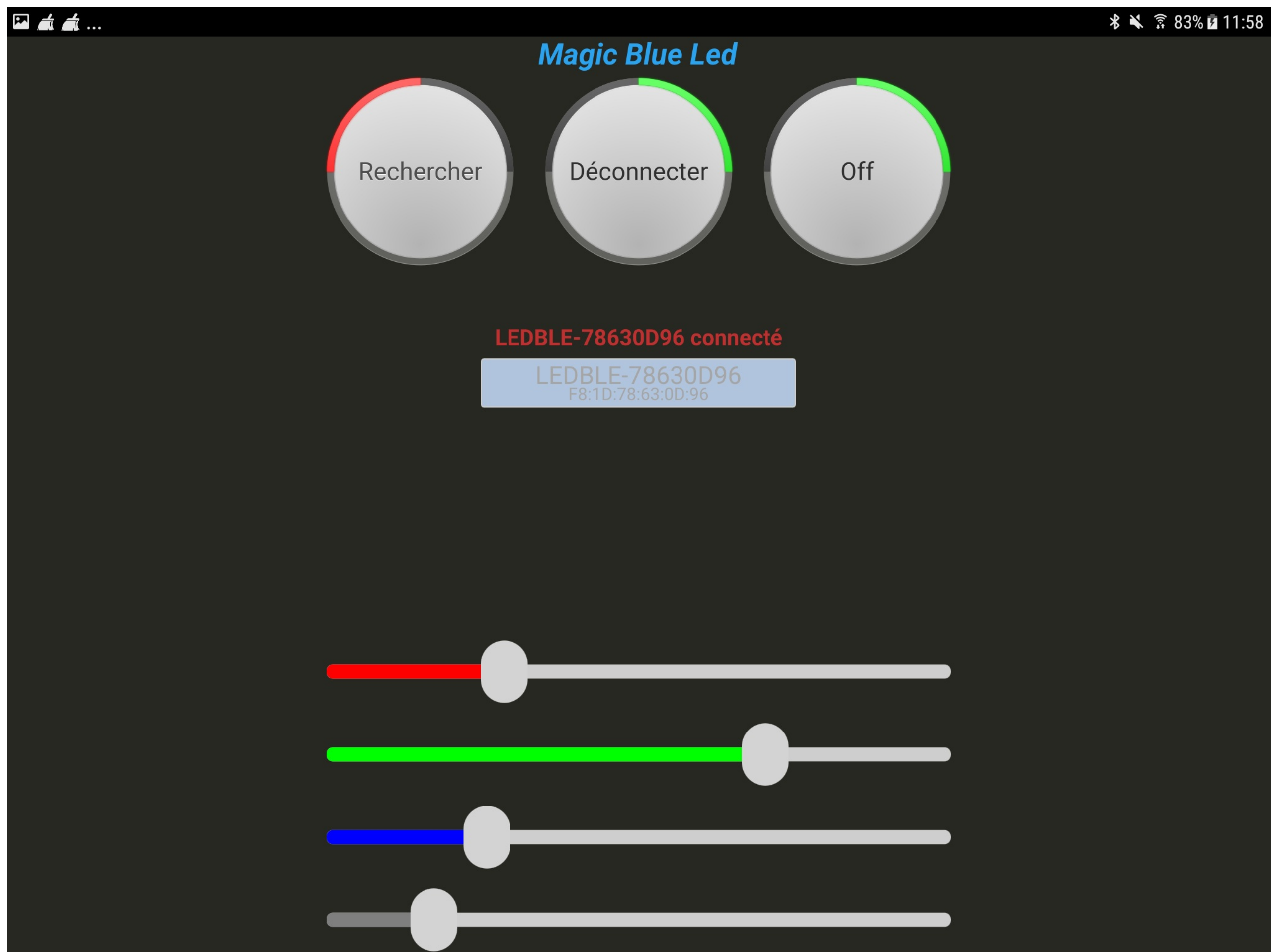
On obtient :



On lance la recherche :



On séléctionne la Magic Blue Led, on se connecte et on la pilote :



Code source : ClientLedBLE.zip

Thierry Vaira <tvaira@free.fr>
2018-2019 (rev. 1.1)
tvaira.free.fr