The Raspberry Pi as BLE peripheral

You will now use the Raspberry Pi to start advertising itself to BLE central devices and expose some data.

The example you will use is similar to some commercial BLE "Smart Light" products on the market at the moment. Typically they are controlled using a smart phone app whereby their colour and state(on/off) can be changed. They can also include other novel features. For example, PlayBulb provide a BLE light that has a built in speaker that pulses and changes colour to match the mood of the music (a party bulb!).

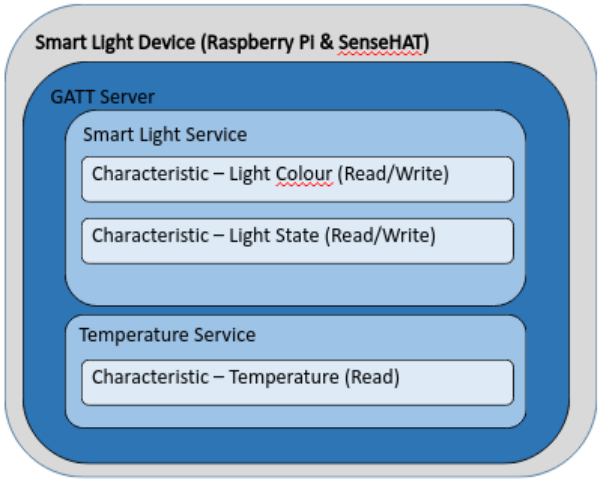
You will now use the 8x8 LED matrix on the RPi to create a Smart light proto type that can be controlled from a Mobile device. It will provide the following functionality:

- Control light colour
- temperature

By adverising temperature, the lightbulb could posiibly be used as part of a smart home climate control system or as an ambient/assisted living device where the colour indicates temperature (blue -> too cold, red -> too hot).

Service Design

Your RPi can have multiple BLE services. Each BLE service, in turn, can have multiple characteristics. Characteristics can be read-only, read/write, or notify. Our Smart Light will contain 2 services: a SmartLight service and a Temperature service.
The BLE service design you will implement is shown in the following diagram:



service design

Smart Light Service

The Bluetooth Special interest group (SIG) has agreed specifications for BLE services and characteristics (for example Heart Rate Monitors). To date, there is no specific agreed standard for the light services and colour so we can define our own custom spec. Each service must have its own Universally Unique Identifier (UUID). You can use an online tool GUID Generator to do this. For the purposes of this lab, we will use the following:

```
UUID_SMART_LIGHT_SRVC = 'FF00'
UUID_LIGHT_COLOUR_CHRC = 'FF01'
UUID_LIGHT_SWITCH = 'FF02'
```

Temperature Service

There is an agreed standard for Temperature characteristics. Based on this, the UUID for temperature will use the short 16 bit UUID defined by the BLuetooth SIG:

```
UUID_TEMP_SRVC = '2A60'
UUID_TEMP_CHRC = '2A6E'
```

Turn on BLE advertising

Before changing anything, lets check if the generic service example runs.

First, change the bluetooth device name permanently to include your name so that it can be identified easily. On the Raspberry Pi, create a file called `/etc/machine-info` (perhaps use `nano /etc/machine-info`) and add the following content:

```
PRETTY_HOSTNAME=YOUR_NAME
```

Next, configure your Bluetooth controller on the RPi to allow BLE advertising. At the command line on the RPi, run the following command to allow both reading and writing to ble services.

```
sudo hciconfig hci0 leadv 0
```

("leadv" stands for LE Advertising...)

Clone Example Service

Open a terminal window and make sure you are in your home directory by entering `cd ~`.

Use `Git` to clone the starter code for your GATT service. In your home directory, run the following command

```
~ $ git clone https://github.com/fxwalsh/Bluetooth-Low-Energy-LED-Matrix.git
```

Change directory to the cloned repository and examine the files in the src directory.

```
~ $ cd Bl*/src
~/Bluetooth-Low-Energy-LED-Matrix/src $ ls
bluez_components.py  gatt-server
```

`bluez_components.py` contains "boiler-plate" functions that control the BLE adapter. The `gatt-server` implements a generic BLE service that we can modify to implement our Smart Light Service. At the command prompt in the src directory, type `sudo ./gatt-server` to start the service. Hopefully there are no errors and you see something similar to the following:

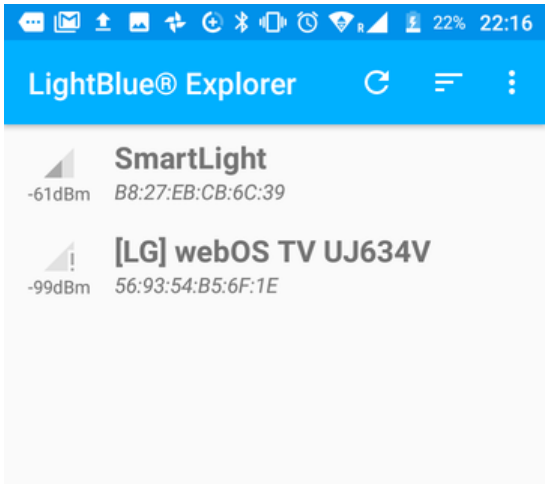
```
pi@sensePi:~/Bluetooth-Low-Energy-LED-Matrix/src $ sudo ./gatt-server
Registering GATT application...
GetManagedObjects
GATT application registered
```

Explore your device

Your RPi should now be advertising and waiting for a central device to connect to it. You can use a Smart phone app to do this. LightBlue Explorer is a BLE test app that allows you to scan for Bluetooth Low Energy devices and communicate with them. It also is available for both Android and Apple iOS Devices. Go to the App Store on your device and search for it using 'LightBlue' as the search word and install it on your device. The following desciption is using Android version of LightBlue.

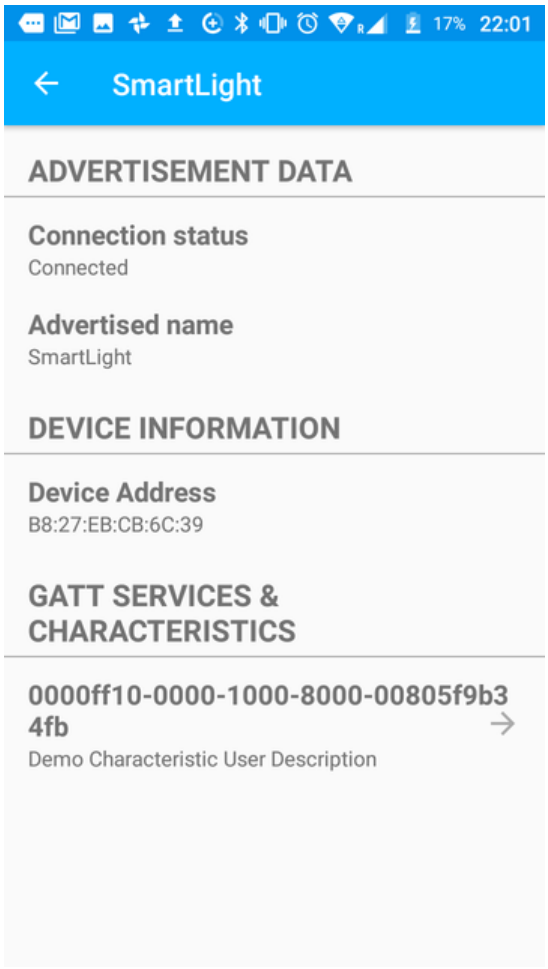
Scan for BLE Devices

To test our RPi Smart Light peripheral, we'll be using LightBlue in Central Mode which allows you to scan and connect to BLE peripherals. Start LightBlue on your device. After some onboarding messages it should start scanning and you should see nearby devices appear on the screen. You can choose a peripheral from a list of nearby devices and explore information about that connected peripheral, its services, and characteristics.



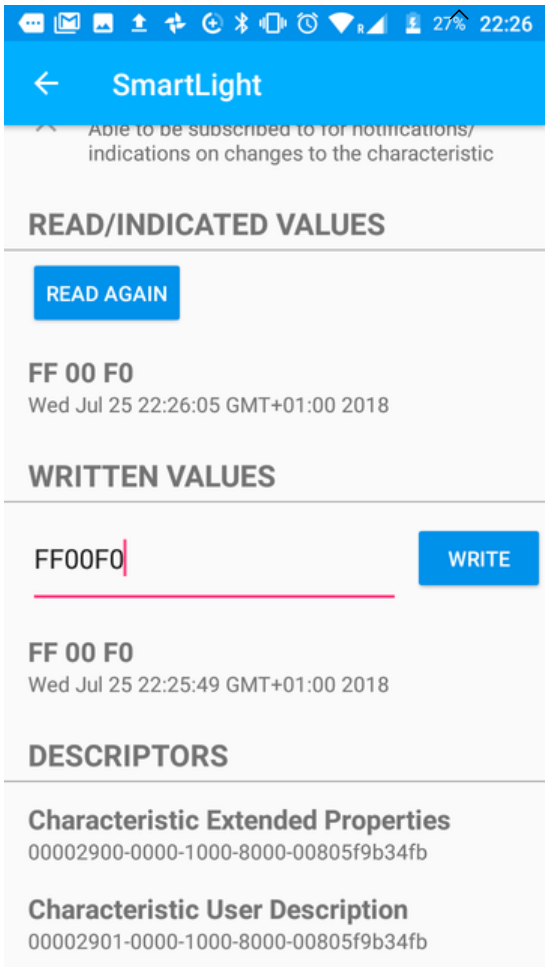
lightblue peripheral list

Find the Smart Light peripheral in the list. Tap on the service to connect and you should see the advertisement data for the Demo Service showing just one characteristic.



lightblue service

Tap on the characteristic and, in "Written Values", enter "FF00F0" and hit Write; the screen should show the new value as 0xFF0F. Hit the Read button and it should return the same value.



lightblue characteristic

If you now check the console output in the RPi where the service is running, you should see output confirming the write and read.

```
GetManagedObjects
GATT application registered
DemoCharacteristic Read: [0]
Demo Characteristic WriteValue called
DemoCharacteristic Read: dbus.Array([dbus.Byte(255), dbus.Byte(0), dbus.Byte(240)]
```

rpi console - gatt service

Congratulations, you've just implemented and tested a simple BLE peripheral on a Raspberry Pi. Now we will modify this service to control the LED matrix and read temperature sensor on the SenseHAT.

Stopping the Service

To stop the GATT server, enter `ctrl+c` in the RPi console. The console should then return to the command prompt.