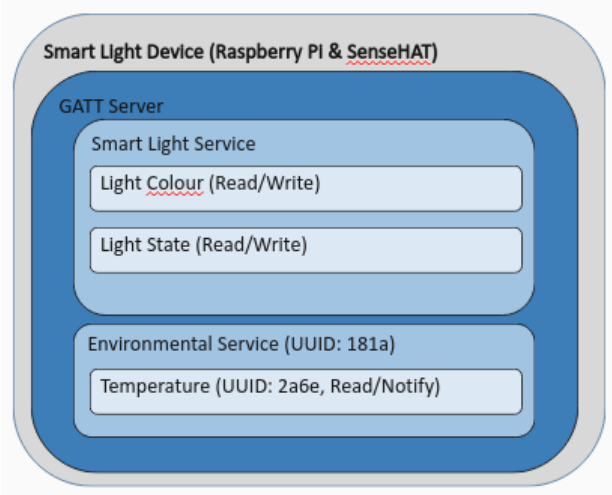
Temperature Service

Now you will add another service to the RPi GATT server. The Temperature service will read from the temperature sensor on the SenseHAT and provide both read and notify functionality for the temperature characteristic.

The environment service design is as follows:



environment service

The design applies the UUIDs for environment sensing service and temperature characteristic specified by the Bluetooth SIG.

Environment Service Class

NOTE: ALL THE FOLLOWING CODE SHOULD BE ADDED ABOVE THE `main()` FUNCTION

Add the following class to the `smartlight-gatt-server` script:

```
class EnvironmentService(Service):

    ENV_UUID = '181a'

    def __init__(self, bus, index):
        Service.__init__(self, bus, index, self.ENV_UUID, True)
        self.add_characteristic(TemperatureCharacteristic(bus, 0, self))
```

As you can see the service uses the reserved UUID for Environment Sensing services and adds just on characteristic, `TemperatureCharacteristic`

Temperature Characteristic class

Add the folowing class to the `smartlight-gatt-server` script.

```
class TemperatureCharacteristic(Characteristic):
    TEMP_UUID = '2a6e'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.TEMP_UUID,
                ['read', 'notify'],
                service)
        self.notifying = False
        self.temp = temp_sint16(sense.get_temperature())
        GObject.timeout_add(1000, self.get_temp)

    def notify_temp(self):
        if not self.notifying:
            return
        self.PropertiesChanged(
                GATT_CHRC_IFACE,
                        { 'Value': self.temp }, [])
        print('temp notify: ' + repr(self.temp))
        return self.notifying

    def get_temp(self):
        self.temp = temp_sint16(sense.get_temperature())
        if not self.notifying:
            return True
        self.notify_temp()
        return True

    def ReadValue(self, options):
        self.get_temp()
        print('temp read: ' + repr(self.temp))
        return self.temp

    def StartNotify(self):
        if self.notifying:
            print('Already notifying, nothing to do')
            return
        self.notifying = True

    def StopNotify(self):
        if not self.notifying:
            print('Not notifying, nothing to do')
            return
        self.notifying = False

## Utility function to convert temp value to 16 bit int value, preserving 2 decimal places
def temp_sint16(value):
    answer = []
    value_int16=int(value * 100).to_bytes(2, byteorder='little', signed=True)
    for bytes in value_int16:
        answer.append(dbus.Byte(bytes))
    return answer
```

Finally, add the Environment Service to the application by adding the following line to the `main()` function:

```
app.add_service(EnvironmentService(bus, 1))
```

Notice that this characteristic permits notifications and includes `StartNotify` and `StopNotify` functions to control notifications sent to a connected device. The service requests the temperature from the SenseHAT every second (1Hz) by using the `GObject.timeout_add()` function.

Run it!

Now run the service again. You should now see a new service included for your smartlight device. To read the temperature from the device, do