Tutorial 4 - Bluetooth Low Energy Part 1

Aim

The Arduino BLE Sense Rev 2 is powered by the nRF52840 from the Nordic Semiconductor. The nRF52840 is a versatile SoC which supports not only Bluetooth Low Energy (BLE), but also Zigbee and Open Thread discussed in the lecture. However, MicroPython supports BLE only. Furthermore, this tutorial focuses on the BLE peripheral role only, which will usually be acting as a Generic ATTribute Profile (GATT) server. Therefore, this tutorial will demonstrate how to create a BLE peripheral device via MicroPython step by step.

Required Software

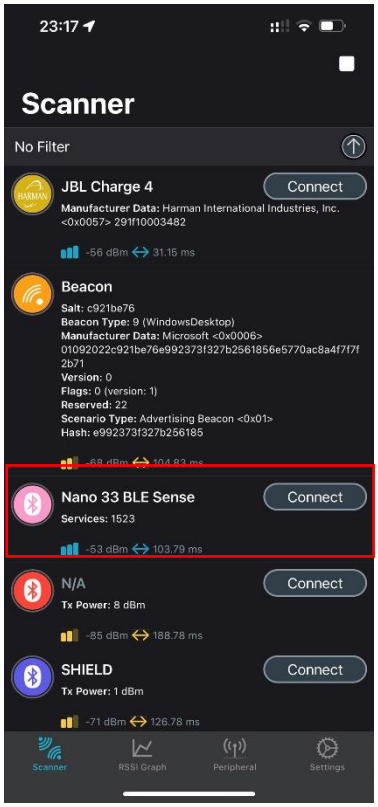
nRF Connect – available on IOS or Google App store.

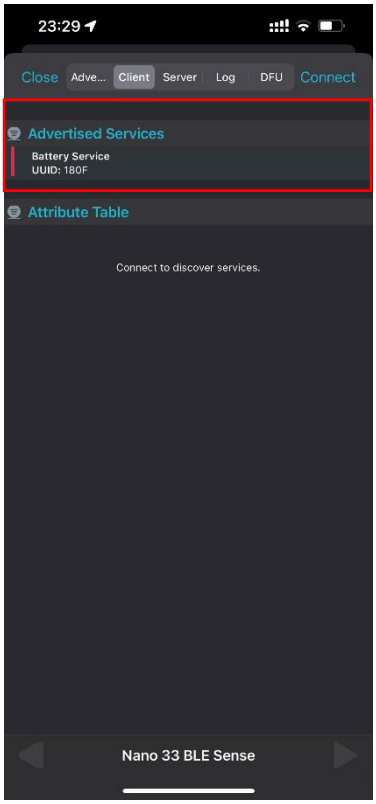
Advertising

The first step is to make the device discoverable. You can modify the device name to your student ID, (e.g., replacing 33 in the screenshot below with your ID), so that you can identify it from those of your fellow students.

```
import time
from ubluepy import Service, Characteristic, UUID, Peripheral,
constants

periph = Peripheral()
periph.advertise(device_name="Nano 33 BLE Sense", services=[])

while (True):
    time.sleep_ms(500)
```

Once you run the above code, open the nRF Connect App on your phone. Note: The Bluetooth need to be turned on your phone. We use the IOS version as an example and the Android version is the same. Swipe down to scan nearby devices.





The device will pop up, as shown in the red bounding box. If you have already known which services will be in your GATT server, you can also advertise these services. For example, the battery service is being advertised.

```
import time
from ubluepy import Service, Characteristic, UUID, Peripheral,
constants

battery_svc_uuid = UUID(0x180F)
battery_svc = Service(battery_svc_uuid)

periph = Peripheral()
periph.advertise(device_name="Nano 33 BLE Sense",
services=[battery_svc])

while (True):
    time.sleep_ms(500)
```

Universally Unique IDs (UUIDs)

The above code has shown how to define a service in MicroPython. As discussed in the lecture, UUIDs are unique numbers used to identify services, characteristics and descriptors, also known as attributes. There are two types of UUIDs, whose lengths are 16-bit (short) and 128-bit (long) respectively.

The first type is a short 16-bit UUID defined by Bluetooth SIG. It is more energy and memory efficient due to its small range of possible unique IDs. You can check more pre-defined UUIDs on Bluetooth SIG's website [1].

What if we want to have own-defined services/characteristics? Hence there is a need for a second type of UUID so you can transmit your own custom UUIDs as well. It looks something like this: 4A98xxxx-1CC4-E7C1-C757-F1267DD021E8 and is called the "base UUID". You can have any base UUIDs that you like, and the four x's represent a field where you will insert your own 16-bit IDs for your custom services and characteristics and use them just like a predefined UUID. You may use an online UUID generator to generate a 128-bit UUID randomly. The possibility of getting two identical 128-bit UUIDs is extremely small if you select UUIDs randomly.

Please be noted that each 16-bit UUID also has a unique 128-bit representation following such format: 0x0000xxxx-0000-1000-8000-00805f9b34fb.

Handle Connection Event

In MicroPython, you can control the connection/disconnection event by defining a handler callback function.

Note: The device needs to restart the advertisement after being disconnected.

```
import time
from ubluepy import Service, Characteristic, UUID, Peripheral,
constants

def event_handler(id, handle, data):
    global periph
    global services
    if id == constants.EVT_GAP_CONNECTED:
        pass
    elif id == constants.EVT_GAP_DISCONNECTED:
        # restart advertisement
        periph.advertise(device_name="Nano 33 BLE Sense")


periph = Peripheral()
periph.setConnectionHandler(event_handler)
periph.advertise(device_name="Nano 33 BLE Sense")


while (True):
    time.sleep_ms(500)
```

Send Data to the Peripheral

The first step is to create a custom service and characteristic. The following lines define a custom service with “0x4A981234-1CC4-E7C1-C757-F1267DD021E8” UUID and a custom characteristic with “0x4A981235-1CC4-E7C1-C757-F1267DD021E8” UUID. You need to set the “Characteristic.PROP_WRITE” flag when initialise the custom characteristic. Then add the custom characteristic to the custom service.

```
custom_svc_uuid = UUID("4A981234-1CC4-E7C1-C757-F1267DD021E8")
custom_wrt_char_uuid = UUID("4A981235-1CC4-E7C1-C757-F1267DD021E8")

custom_svc = Service(custom_svc_uuid)   # handle = handle + 1 (service)
custom_wrt_char =  Characteristic(custom_wrt_char_uuid,
props=Characteristic.PROP_WRITE)         # char = 2
custom_svc.addCharacteristic(custom_wrt_char) # handle = handle + char
```

The next step is to handle the corresponding write event. We may handle the write event in the event handler callback function as follows:

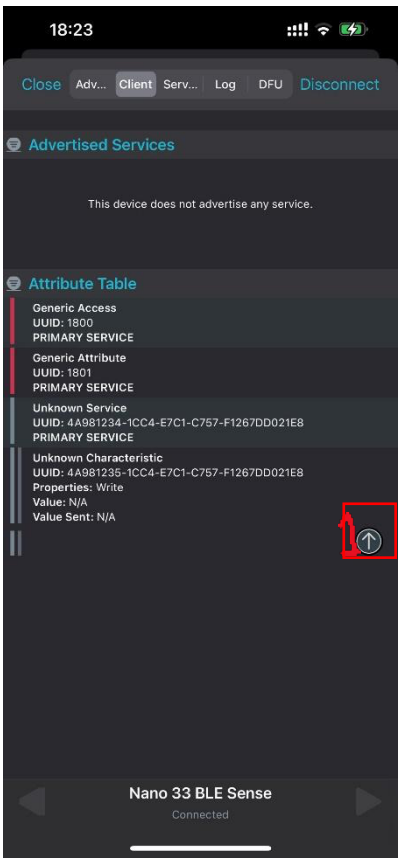
```
def event_handler(id, handle, data):
    global periph
    global services
    if id == constants.EVT_GAP_CONNECTED:
        pass
    elif id == constants.EVT_GAP_DISCONNECTED:
        # restart advertisement
        periph.advertise(device_name="Nano 33 BLE Sense")
    elif id == constants.EVT_GATTS_WRITE:
        if handle == 16:
            print(data)
```

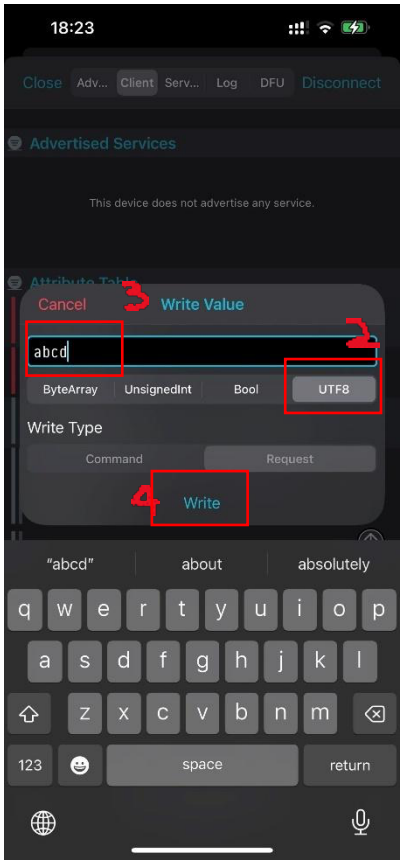
The handle value corresponding to the custom characteristic is 16. This can be calculated by the equations in the comments when defining the custom service and characteristic. The initial handle value is 13. Therefore, 16 = 13 + 1 + 2 in our example above). Please see blow for the complete example codes.

```
import time
from ubluepy import Service, Characteristic, UUID, Peripheral,
constants

def event_handler(id, handle, data):
    global periph
    global services
    if id == constants.EVT_GAP_CONNECTED:
        pass
    elif id == constants.EVT_GAP_DISCONNECTED:
        # restart advertisement
        periph.advertise(device_name="Nano 33 BLE Sense")
    elif id == constants.EVT_GATTS_WRITE:
        if handle == 16:
```

```
            print(data)

custom_svc_uuid = UUID("4A981234-1CC4-E7C1-C757-F1267DD021E8")
custom_wrt_char_uuid = UUID("4A981235-1CC4-E7C1-C757-F1267DD021E8")

custom_svc =
Service(custom_svc_uuid)
custom_wrt_char =  Characteristic(custom_wrt_char_uuid,
props=Characteristic.PROP_WRITE)
custom_svc.addCharacteristic(custom_wrt_char)


periph = Peripheral()
periph.addService(custom_svc)
periph.setConnectionHandler(event_handler)
periph.advertise(device_name="Nano 33 BLE Sense")

while (True):
    time.sleep_ms(500)
```

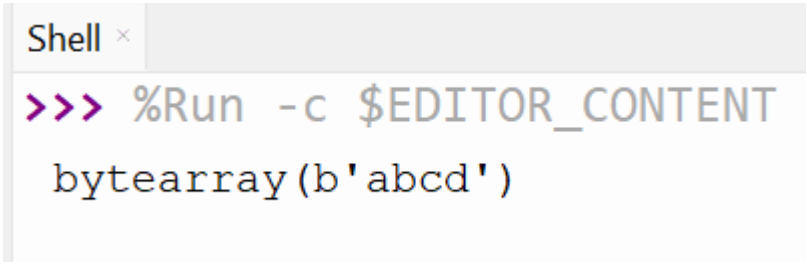
Now run the completed code and open the nRF Connect App on your phone. Scan and connect the device.





You should be able to see interface like the left photo. Click the up-arrow button and it will pop up a window to let you input any data. Here we select UTF8 to send text strings “abcd” to the device. Press “Write” to send.

Now check your device's MicroPython prompt. You should be able to see the message that you typed on your phone.



Read Data from the Peripheral

The peripheral may also "send" message to the host. There are two ways to achieve this: either the host reads the value in the characteristic, or the host subscribe the notification from the peripheral.

Similarly, the first step is to create a custom characteristic to allow the host to read. Note: there is no need to add a new service since we can add the new characteristic to the existing service.

```
custom_svc_uuid = UUID("4A981234-1CC4-E7C1-C757-F1267DD021E8")
custom_wrt_char_uuid = UUID("4A981235-1CC4-E7C1-C757-F1267DD021E8")
custom_read_char_uuid = UUID("4A981236-1CC4-E7C1-C757-F1267DD021E8")

custom_svc = Service(custom_svc_uuid)
custom_wrt_char =  Characteristic(custom_wrt_char_uuid,
props=Characteristic.PROP_WRITE)
custom_read_char =  Characteristic(custom_read_char_uuid,
props=Characteristic.PROP_READ | Characteristic.PROP_NOTIFY,
attrs=Characteristic.ATTR_CCCD)
custom_svc.addCharacteristic(custom_wrt_char)
custom_svc.addCharacteristic(custom_read_char)
```

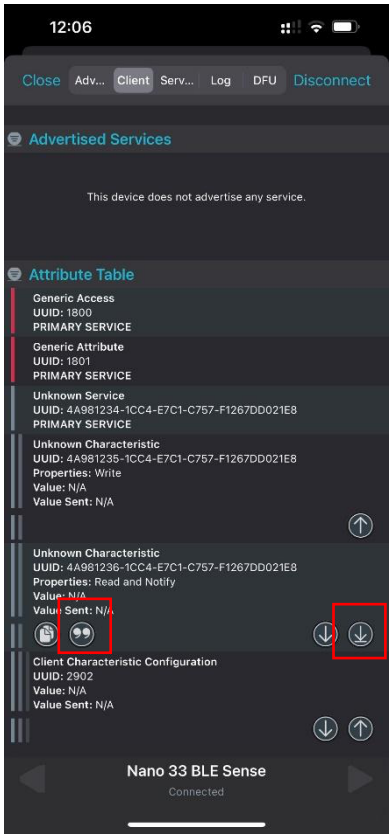
To send data to the host, we only need to call "custom_read_char.write()" method. The following code below shows a loopback experiment that a BLE device receives a message from the host before sending it back to the host:

```
import time
from ubluepy import Service, Characteristic, UUID, Peripheral,
constants

def event_handler(id, handle, data):
    global periph
    global services
    global custom_read_char
    global notif_enabled
    if id == constants.EVT_GAP_CONNECTED:
        pass
    elif id == constants.EVT_GAP_DISCONNECTED:
        # restart advertisement
```

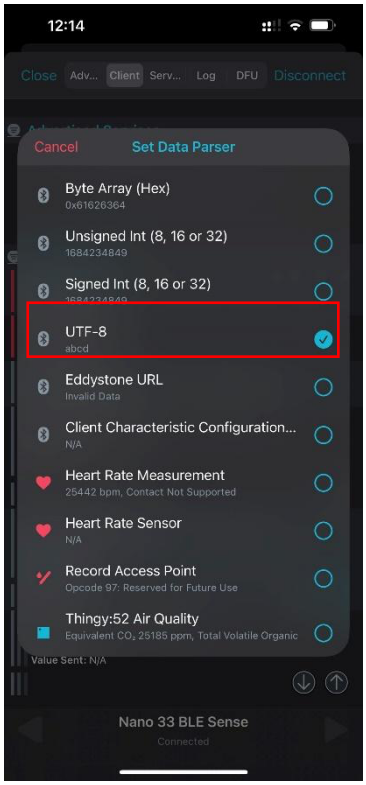
```
        periph.advertise(device_name="Nano 33 BLE Sense")
    elif id == constants.EVT_GATTS_WRITE:
        if handle == 16:                    # custom_wrt_char
            if notif_enabled:
                custom_read_char.write(data)
        elif handle == 19:                  # CCCD of custom_read_char
            if int(data[0]) == 1:
                notif_enabled = True
            else:
                notif_enabled = False

notif_enabled = False

custom_svc_uuid = UUID("4A981234-1CC4-E7C1-C757-F1267DD021E8")
custom_wrt_char_uuid = UUID("4A981235-1CC4-E7C1-C757-F1267DD021E8")
custom_read_char_uuid = UUID("4A981236-1CC4-E7C1-C757-F1267DD021E8")

custom_svc = Service(custom_svc_uuid)
custom_wrt_char =  Characteristic(custom_wrt_char_uuid,
props=Characteristic.PROP_WRITE)
custom_read_char =  Characteristic(custom_read_char_uuid,
props=Characteristic.PROP_READ | Characteristic.PROP_NOTIFY,
attrs=Characteristic.ATTR_CCCD)
custom_svc.addCharacteristic(custom_wrt_char)
custom_svc.addCharacteristic(custom_read_char)

periph = Peripheral()
periph.addService(custom_svc)
periph.setConnectionHandler(event_handler)
periph.advertise(device_name="Nano 33 BLE Sense")

while (True):
    time.sleep_ms(500)
```

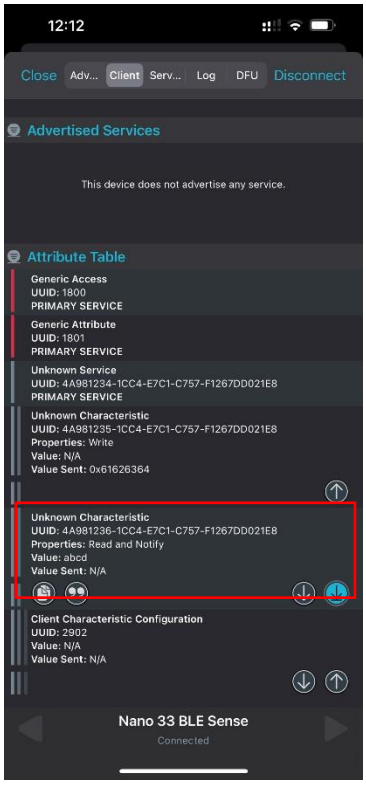
Note: Please make sure the device enables the notification before you call the “custom_read_char.write()” method or use a flag to avoid that (as shown in example).

Run the completed code and open the nRF Connect App on your phone. After connecting the device, you should be able to see:



Then press the down-arrow button that highlighted in the red bounding box to subscribe the notification. Press the “quotation mark” to select UTF-8 as the data parser. Then follow the procedure in “sending data to the peripheral” to send a text message to the device.





Now you can see the “custom_read_char” characteristic receives the message that you typed on your phone.

Reference

[1] https://btprodspecificationrefs.blob.core.windows.net/assigned-numbers/Assigned%20Number%20Types/Assigned_Numbers.pdf