Tutorial 5 – Bluetooth Low Energy Part 2

Aim

In tutorial 4, we demonstrate how to create a Bluetooth Low Energy (BLE) peripheral device via MicroPython step by step and use nRF Connect software to connect to the BLE device. This tutorial will demonstrate how to use Python on your laptop to communicate with the device via BLE.

Required Software

Bleak - https://github.com/hbldh/bleak

- Supports Windows 10, version 16299 (Fall Creators Update) or greater
- Supports Linux distributions with BlueZ >= 5.43
- OS X/macOS support via Core Bluetooth API, from at least OS X version 10.11

You may create a python virtual environment and execute “pip install bleak”.

Scan Devices

To establish the connection, we need to know the device’s Bluetooth address. We can use the Bleak to scan the nearby BLE devices and find the Arduino board based on the advertising name. Here is the code to discover nearby devices on your laptop:

```
import argparse
import asyncio
from bleak import BleakScanner
async def main():
    print("scanning for 5 seconds, please wait...")
    devices = await BleakScanner.discover(
        return_adv=True
    )
    for d, a in devices.values():
        print()
        print(d)
        #print(d.address, d.rssi, d.name)
        print("-" * len(str(d)))
        print(a)
if __name__ == "__main__":
    asyncio.run(main())
```

Before running this Python script on your laptop, run the completed MicroPython code in tutorial 4 on the Arduino board.

The output of the Python script will look like:

```
scanning for 5 seconds, please wait...

04:21:44:B6:2B:B6: JBL Charge 4
--------------------------------
AdvertisementData(local_name='JBL Charge 4', manufacturer_data={87: b')\x1f\x10\x004\x82'}, rssi=-46)

47:6B:9C:C8:44:B6: None
----------------------
AdvertisementData(manufacturer_data={76: b'\x10\x07a\x1f\x94\x07\xee\xf78'}, tx_power=6, rssi=-53)

56:34:D3:AC:84:58: None
----------------------
AdvertisementData(manufacturer_data={76: b'\x10\x05/\x18\xbf\xe7\x92'}, tx_power=8, rssi=-76)

6B:9C:46:FB:B0:F3: SHIELD
-------------------------
AdvertisementData(local_name='SHIELD', manufacturer_data={19522: b'\xbf\xfb\xfb\xbf\n1907398144'}, tx_power=1, rssi=-73)

4D:6E:AC:44:78:9C: None
----------------------
AdvertisementData(manufacturer_data={76: b'\x10\x077\x1f\x9d\xb0\xc7\x80x'}, tx_power=6, rssi=-83)

F1:56:8B:BD:1E:D5: Nano 33 BLE Sense
------------------------------------
AdvertisementData(local_name='Nano 33 BLE Sense', rssi=-82)

5C:7C:08:D4:07:48: None
----------------------
AdvertisementData(manufacturer_data={76: b'\x10\x06@\x1d\xb4\x0c\xf7H'}, tx_power=7, rssi=-81)

79:F4:2B:55:CC:7D: None
----------------------
AdvertisementData(manufacturer_data={76: b'\x10\x02\x13\x00'}, tx_power=12, rssi=-50)
```

The red bounding box shows the advertisement information from the Arduino board, including the BLE Media Access Control (MAC) address, advertising name and BLE packet Received Signal Strength Indicator (RSSI). Check the comment in the code to access these attributes individually.

Note: macOS doesn't show the Bluetooth MAC address, instead, it will show the 128-bit UUID to represent the address. If you are using macOS, use this UUID in the labs.

Connect and Disconnect

To connect the device, create the BleakClient object via the asynchronous context manager. You need to define the Bluetooth address based on the above example code.

```python
import argparse
import asyncio
import platform
import sys
from bleak import BleakClient
from bleak import BleakScanner
ADDRESS = (
    "F1:56:8B:BD:1E:D5"                                    # Change to
your device's address if on Linux/Windows
    if platform.system() != "Darwin"
    else "B9EA5233-37EF-4DD6-87A8-2A875E821C46"            # Change to
your device's address if on macOS
)
async def main():
    async with BleakClient(ADDRESS) as client:
        print(f"Connected: {client.is_connected}")
        await asyncio.sleep(5.0)
    print("disconnected")
if __name__ == "__main__":
    asyncio.run(main())
```

Run the above code and it will print out the connection status. After 5 seconds, it will disconnect with the Arduino board automatically.

Read and Write

Now we can define the custom service and characteristics in tutorial 4 and enable the notification after establishing the connection. A callback function is needed to handle the notification event, and in the example below, the received data is printed out in the callback function. In the example, we use a for loop to wait for user's input and send the input message to the "custom_wrt_characteristic". You will receive the message back in the "handle_rx" callback function.

```
import argparse
import asyncio
import platform
import sys
from bleak import BleakClient
from bleak import BleakScanner
from bleak.backends.characteristic import BleakGATTCharacteristic
ADDRESS = (
    "F1:56:8B:BD:1E:D5"                                 # Change to
your device's address if on Linux/Windows
    if platform.system() != "Darwin"
    else "B9EA5233-37EF-4DD6-87A8-2A875E821C46"         # Change to
your device's address if on macOS
)
custom_svc_uuid = "4A981234-1CC4-E7C1-C757-F1267DD021E8"
custom_wrt_char_uuid = "4A981235-1CC4-E7C1-C757-F1267DD021E8"
custom_read_char_uuid = "4A981236-1CC4-E7C1-C757-F1267DD021E8"
async def main():
    def handle_rx(_: BleakGATTCharacteristic, data: bytearray):
        print("received:", data)
    async with BleakClient(ADDRESS) as client:
        print(f"Connected: {client.is_connected}")
        await client.start_notify(custom_read_char_uuid, handle_rx)
        print("Connected, start typing and press ENTER...")
        loop = asyncio.get_running_loop()
        custom_svc = client.services.get_service(custom_svc_uuid)
        wrt_char = custom_svc.get_characteristic(custom_wrt_char_uuid)
        while True:
            # This waits until you type a line and press ENTER.
            # A real terminal program might put stdin in raw mode so
that things
            # like CTRL+C get passed to the remote device.
            data = await loop.run_in_executor(None,
sys.stdin.buffer.readline)
            # data will be empty on EOF (e.g. CTRL+D on *nix)
            if not data:
                break
            await client.write_gatt_char(wrt_char, data)
            print("sent:", data)
```

```
if __name__ == "__main__":
    asyncio.run(main())
```

You should get some outputs like:

```
Connected: True
Connected, start typing and press ENTER...
abcd
sent: b'abcd\r\n'
```

Conclusion

This tutorial introduces the Bleak Python library which can handle the BLE communication with the Arduino board.