Connecting to BLE router

1. Power up Thingy. Board shall be visible via *hcitool lescan* command as 00:AA:13:A1:5C:A5 Nordic_AWS_Demo

```
eugene@eugene-Latitude-5580:~/nordic-aws-iot-demo/ble_router$ sudo hcitool lescan
LE Scan ...
D0:4A:8C:1D:C5:1E BLETR
D0:4A:8C:1D:C5:1E (unknown)
D8:AB:08:0E:33:C3 BLETR
D8:AB:08:0E:33:C3 (unknown)
13:61:1A:C7:63:46 (unknown)
FC:8F:90:40:CB:1A (unknown)
00:AA:13:A1:5C:A5 Nordic_AWS_Demo
00:AA:13:A1:5C:A5 (unknown)
```

2. Execute *ble6_conn.sh*. This will create network interface *bt0*, and will start to ping the board. Wait for 3-4 seconds and execute script again, ping should start and response should present.

```
eugene@eugene-Latitude-5580:~/nordic-aws-iot-demo/ble_router$ sudo ./ble6_conn.sh
[sudo] password for eugene:
ping: unknown iface bt0
eugene@eugene-Latitude-5580:~/nordic-aws-iot-demo/ble_router$ sudo ./ble6_conn.sh
PING FE80:0000:0000:0000:02AA:13FF:FEA1:5CA5(fe80::2aa:13ff:fea1:5ca5) from fe80::
 bytes
64 bytes from fe80::2aa:13ff:fea1:5ca5: icmp_seq=1 ttl=255 time=501 ms
64 bytes from fe80::2aa:13ff:fea1:5ca5: icmp_seq=2 ttl=255 time=341 ms
64 bytes from fe80::2aa:13ff:fea1:5ca5: icmp_seq=3 ttl=255 time=179 ms
64 bytes from fe80::2aa:13ff:fea1:5ca5: icmp_seq=4 ttl=255 time=368 ms
64 bytes from fe80::2aa:13ff:fea1:5ca5: icmp_seq=5 ttl=255 time=417 ms
64 bytes from fe80::2aa:13ff:fea1:5ca5: icmp_seq=6 ttl=255 time=186 ms
64 bytes from fe80::2aa:13ff:fea1:5ca5: icmp_seq=7 ttl=255 time=304 ms
```

3. Set up local Mosquitto broker by running *mos_bro.sh*. Kill mosquitto process first if needed. Edit script to use *mosquitto.conf* file from repository, where PSK file is also located.
4. Set up local Mosquitto subscriber by running *mos_sub.sh*.
5. Press button on Thingy and connection to broker will be established. Then data from sensors will be sent periodically and displayed by subscriber.

```
eugene@eugene-Latitude-5580:~/nordic-aws-iot-demo/ble_router$ ./mos_sub.sh
/v2/feeds/637693378.json {"temperature": 24, "humidity": 39, "pressure": 1013, "marker": true}
/v2/feeds/637693378.json {"temperature": 24, "humidity": 40, "pressure": 1013, "marker": true}
/v2/feeds/637693378.json {"temperature": 24, "humidity": 41, "pressure": 1013, "marker": true}
/v2/feeds/637693378.json {"temperature": 24, "humidity": 39, "pressure": 1012, "marker": true}
/v2/feeds/637693378.json {"temperature": 25, "humidity": 39, "pressure": 1013, "marker": true}
/v2/feeds/637693378.json {"temperature": 24, "humidity": 39, "pressure": 1013, "marker": true}
/v2/feeds/637693378.json {"temperature": 24, "humidity": 38, "pressure": 1013, "marker": true}
/v2/feeds/637693378.json {"temperature": 25, "humidity": 39, "pressure": 1012, "marker": true}
/v2/feeds/637693378.json {"temperature": 24, "humidity": 40, "pressure": 1013, "marker": true}
```

Code highlights

Located in *main.c*

1. Local broker address shall be set here. Run *ifconfig bt0* to find this address

```
190 // Address of Xively cloud.
191 static const uint8_t                           m_broker_addr[IPV6_ADDR_SIZE] =
192 {
193 //        0x20, 0x01, 0x07, 0x78,
194 //        0x00, 0x00, 0xff, 0xff,
195 //        0x00, 0x64, 0x00, 0x00,
196 //        0xd8, 0x34, 0xe9, 0x7a
197 //      0x20, 0x01, 0x41, 0xd0,
198 //      0x00, 0x0a, 0x3a, 0x10,
199 //      0x00, 0x00, 0x00, 0x00,
200 //      0x00, 0x00, 0x00, 0x01
201           0xFE, 0x80, 0x00, 0x00,
202           0x00, 0x00, 0x00, 0x00,
203           0xfa, 0x59, 0x71, 0xff,
204           0xfe, 0x9f, 0x80, 0x2a
205 };
```

2. Data representation can be changed here, in function *app_xively_publish_callback* which is executed periodically

```
834     // Prepare data in JSON format.
835 //    sprintf(m_data_body, APP_MQTT_XIVELY_DATA_FORMAT, m_temperature);
836     sprintf(m_data_body, "{\"temperature\": %d, \"humidity\": %d, \"pressure\": %d, \"marker\": true}",
837             // \"accelerometer\": [%f, %f, %f], \"gyroscope\": [%f, %f, %f], \"magnetometer\": [%f, %f,
838             (uint16_t)tmp, hmd, (uint16_t)prs
839             /*, Accelerometer.x, Accelerometer.y, Accelerometer.z, Gyroscope.x, Gyroscope.y, Gyroscope.z
840             );
```

3. Debug output can be seen via Segger RTT (real time terminal)

```
854         APPL_LOG("[TEMPE]: %d\n\r", m_temperature);
855         APPL_LOG("[HUMID]: %d\r\n", hmd);
856         APPL_LOG("[PRESS]: %d\n\r", (uint16_t)prs);
```

Debugging setup

1. Download and install https://www.segger.com/downloads/jlink/
#J-LinkSoftwareAndDocumentationPack

2. Setup Eclipse IDE for GNU ARM & RISC-V C/C++ Developers. It contains Jlink support. Just create new debug configuration and run it. Board will be flashed and will work even when debug is disconnected.

3. When debug is launched, run *JlinkRTTClient* to see debug output from *APPL_LOG* functions