Bluetooth Low Energy in C++ for nRFx Microcontrollers

1st Edition

Tony Gaitatzis

BackupBrain Publishing, 2017

ISBN: 978-1-7751280-7-6

backupbrain.co

Bluetooth Low Energy in C++ for nRFx Microcontrollers

by Tony Gaitatzis

Services and Characteristics

Before data can be transmitted back and forth between a Central and Peripheral, the Peripheral must host a GATT Profile. That is, the Peripheral must have Services and Characteristics.

Identifying Services and Characteristics

Each Service and Characteristic is identified by a Universally Unique Identifier (UUID). The UUID follows the pattern 0000XXXX-0000-1000-8000-00805f9b34fb, so that a 32-bit UUID 00002a56-0000-1000-8000-00805f9b34fb can be represented as 0x2a56.

Some UUIDs are reserved for specific use. For instance any Characteristic with the 16-bit UUID 0x2a35 (or the 32-bit UUID 00002a35-0000-1000-8000-00805f9b34fb) is implied to be a blood pressure reading.

For a list of reserved Service UUIDs, see ***Appendix IV: Reserved GATT Services***.

For a list of reserved Characteristic UUIDs, see ***Appendix V: Reserved GATT Characteristics***.

Generic Attribute Profile

Services and Characteristics describe a tree of data access points on the peripheral. The tree of Services and Characteristics is known as the Generic Attribute (GATT) Profile. It may be useful to think of the GATT as being similar to a folder and file tree (Figure 6-1).

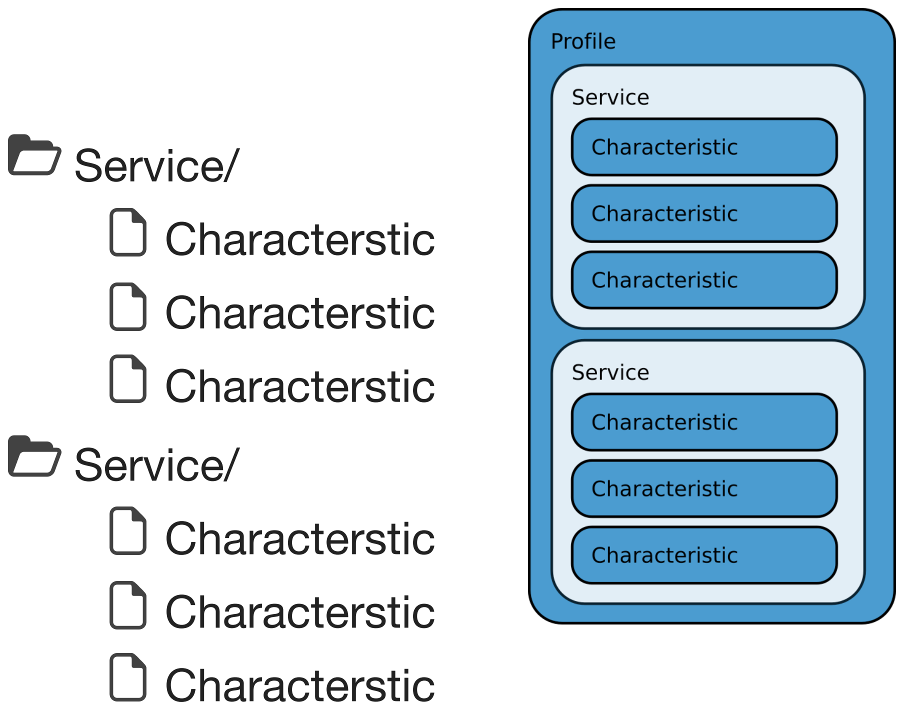


Figure 6-1. GATT Profile filesystem metaphor

Characteristics act as channels that can be communicated on, and Services act as containers for Characteristics. A top level Service is called a Primary service, and a Service that is within another Service is called a Secondary Service.

Permissions

Characteristics can be configured with the following attributes, which define what the Characteristic is capable of doing (Table 6-1):

Table 6-1. Characteristic Permissions

Descriptor	Description
Read	Central can read this Characteristic, Peripheral can set the value.
Write	Central can write to this Characteristic, Peripheral will be notified when the Characteristic value changes and Central will be notified when the write operation has occurred.
Notify	Central will be notified when Peripheral changes the value.

Because the GATT Profile is hosted on the Peripheral, the terms used to describe a Characteristic's permissions are relative to how the Peripheral accesses that Characteristic. Therefore, when a Central uploads data to the Peripheral, the Peripheral can "read" from the Characteristic. The Peripheral "writes" new data to the Characteristic, and can "notify" the Central that the data is altered.

Data Length and Speed

It is worth noting that Bluetooth Low Energy has a maximum data packet size of 20 bytes, with a 1 Mbit/s speed.

Programming the Peripheral

The Generic Attribute Profile is defined by setting the UUID and permissions of the Peripheral's Services and Characteristics.

Characteristics can be configured with the following permissions (Table 6-2):

Table 6-2. BLECharacteristic Permissions

Value	Permission	Description
BLE_GATT_CHAR_PROPERTIES_READ	**Read**	Central can read data altered by the Peripheral
BLE_GATT_CHAR_PROPERTIES_WRITE	**Write**	Central can send data, Peripheral reads
BLE_GATT_CHAR_PROPERTIES_WRITE_WITHOUT_RESPONSE	**Write**	Central can write to this Characteristic, Peripheral is not notified of success
BLE_GATT_CHAR_PROPERTIES_NOTIFY	**Notify**	Central is notified as a result of a change

Characteristics have a maximum length of 20 bytes. Since 16 bit and 8-bit data types are easy to pass around in C++, we will be using uint16_t (unsigned 16-bit integer) and uint8_t (unsigned 8-bit integer) values in the examples. Any data type including custom byte buffers can be transmitted and assembled over BLE.

Define a Service with UUID 180c (an unregistered generic UUID):

```
BLEService service("180C");
```

or

```
BLEService service("0000180C-000-1000-8000-00805f9-b34fb");
```

The first method lets the Peripheral automatically generate most of the UUID, and the second method forces the Peripheral to use a particular UUID. The first method is simpler but less precise. The second method is precise and useful for projects where there is a need to share the UUID with outside people or APIs.

Certain UUIDs are unavailable for use. If a bad UUID is chosen, the Peripheral may crash without warning.

There are several types of Characteristic available in nRF51822, depending on the type of data you need to transmit. Arrays, Integers, Floats, Booleans, and other data types have their own Characteristic constructors.

For instance, this 2-byte long Characteristic with UUID 1801 can be read by a Central and can notify the Central of changes:

```
static char readValue[2] = {0};
ReadOnlyArrayGattCharacteristic
    <uint8_t, sizeof(readValue)> readCharacteristic(
    "1801",
    (uint8_t *)readValue,
    GattCharacteristic::BLE_GATT_CHAR_PROPERTIES_READ | \
    GattCharacteristic::BLE_GATT_CHAR_PROPERTIES_NOTIFY
);
```

This 8-byte long Characteristic with UUID 2A56 (Digital Characteristic) can be written to by the Central:

```
static char writeValue[8] = {0};
WriteOnlyArrayGattCharacteristic
    <uint8_t, sizeof(writeValue)>writeCharacteristic(
    writeCharacteristicUuid,
    (uint8_t *)writeValue,
    GattCharacteristic::BLE_GATT_CHAR_PROPERTIES_WRITE);
```

Here are some examples of various data type specific Characteristics that can be created:

```
int properties = BLERead | BLEWrite | BLENotify;
ReadWriteBooleanCharacteristic booleanCharacteristic(
    UUID,
    properties,
    maxLen
);
ReadWriteIntegerCharacteristic integerDataCharacteristicName(
    UUID,
    properties,
    maxLen
);
BLEUnsignedIntCharacteristic yourCharacteristicName(
    UUID,
    properties,
    maxLen
);
BLELongCharacteristic yourCharacteristicName(
    UUID,
    properties,
    maxLen
);
BLEUnsignedLongCharacteristic yourCharacteristicName(
    UUID,
    properties,
    maxLen
);
BLEFloatCharacteristic yourCharacteristicName(UUID, properties, maxLen);
```

The Services and Characteristics are added to the GATT Profile via the BLEPeripheral. By adding the two Characteristics after the Service, they are assumed to be part of the same Service. This must happen before blePeripheral.begin().

```
...
BLE &ble = BLE::Instance(BLE::DEFAULT_INSTANCE);
...
// Set up custom service
static const uint16_t customServiceUuid  = 0x180C;
GattCharacteristic *characteristics[] = {
    &readCharacteristic, &writeCharacteristic
};
GattService customService(
    customServiceUuid,
    characteristics,
    sizeof(characteristics) / sizeof(GattCharacteristic *)
);
ble.addService(customService);
...
ble.gap().startAdvertising();
...
```

Putting It All Together

Create a new sketch named ble_characteristics and copy the following code.

Example 6-1. sketches/ble_characteristics/ble_characteristics.c

```
#include "mbed.h"
#include "ble/BLE.h"

/** User interface I/O **/

// instantiate USB Serial
Serial serial(USBTX, USBRX);
// Status LED
DigitalOut statusLed(LED1, 0);
// Timer for blinking the statusLed
Ticker ticker;

/** Bluetooth Peripheral Properties **/

// Broadcast name
```

```
const static char BROADCAST_NAME[] = "MyDevice";
// Device Information UUID
static const uint16_t deviceInformationServiceUuid  = 0x180a;
// Battery Level UUID
static const uint16_t batteryLevelServiceUuid  = 0x180f;
// array of all Service UUIDs
static const uint16_t uuid16_list[] = { customServiceUuid };

// Number of bytes in Characteristic
static const uint8_t characteristicLength = 20;
// Device Name Characteristic UUID
static const uint16_t deviceNameCharacteristicUuid = 0x2a00;
// Modul Number Characteristic UUID
static const uint16_t modelNumberCharacteristicUuid = 0x2a24;
// Serial Number Characteristic UUID
static const uint16_t serialNumberCharacteristicUuid = 0x2a04;
// Battery Level Characteristic UUID
static const uint16_t batteryLevelCharacteristicUuid = 0x2a19;
// model and serial numbers
static const char* modelNumber = "1AB2";
static const char* serialNumber = "1234";
int batteryLevel = 100;

/** Functions **/

/**
 * visually signal that program has not crashed
 */
void blinkHeartbeat(void);

/**
 * Callback triggered when the ble initialization process has finished
 *
 * @param[in] params Information about the initialized Peripheral
 */
void onBluetoothInitialized(
    BLE::InitializationCompleteCallbackContext *params
);

/**
 * Callback handler when a Central has disconnected
 *
 * @param[i] params Information about the connection
 */
void onCentralDisconnected(
    const Gap::DisconnectionCallbackParams_t *params
);

/** Build Service and Characteristic Relationships **/

// Create a read/write/notify Characteristic
static uint8_t deviceNameCharacteristicValue[characteristicLength] = \
    BROADCAST_NAME;
ReadOnlyArrayGattCharacteristic<uint8_t, sizeof(characteristicValue)> \
    deviceNameCharacteristic(
    deviceNameCharacteristicUuid,
    characteristicValue);

static uint8_t modelNumberCharacteristicValue[characteristicLength] = \
    modelNumber;
ReadOnlyArrayGattCharacteristic<uint8_t, sizeof(characteristicValue)> \
    modelNumberCharacteristic(
    modelNumberCharacteristicUuid,
    characteristicValue);

static uint8_t serialNumberCharacteristicValue[characteristicLength] = \
    serialNumber;
ReadOnlyArrayGattCharacteristic<uint8_t, sizeof(characteristicValue)> \
    serialNumberCharacteristic(
    serialNumberCharacteristicUuid,
    characteristicValue);
```

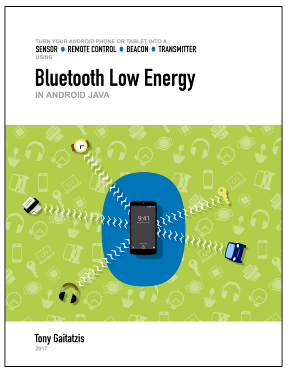
```
// Bind Characteristics to Services
GattCharacteristic *deviceInformationCharacteristics[] = {
    &deviceNameCharacteristic,
    &modelNumberCharacteristic,
    serialNumberCharacteristic
};
GattService deviceInformationService(
    deviceInformationServiceUuid,
    deviceInformationCharacteristics,
    sizeof(deviceInformationCharacteristics) / \
    sizeof(GattCharacteristic *)
);



static uint8_t batteryLevelCharacteristicValue = batteryLevel;
ReadOnlyGattCharacteristic<uint8_t, sizeof(characteristicValue)> \
    batteryLevelCharacteristic(
    serialNumberCharacteristicUuid,
    characteristicValue,
    GattCharacteristic::BLE_GATT_CHAR_PROPERTIES_READ | \
    GattCharacteristic::BLE_GATT_CHAR_PROPERTIES_NOTIFY
);

GattCharacteristic *batteryLevelCharateristics[] = {
    &batteryLevelCharacteristic
}
GattService batteryLevelService(
    batteryLevelServiceUuid,
    batteryLevelCharateristics,
    sizeof(batteryLevelCharateristics) / sizeof(GattCharacteristic *)
);

/**
 * Main program and loop
 */
int main(void) {
    serial.baud(9600);
    serial.printf("Starting Peripheral\r\n");
    ticker.attach(blinkHeartbeat, 1); // Blink status led every 1 second
    // initialized Bluetooth Radio
    BLE &ble = BLE::Instance(BLE::DEFAULT_INSTANCE);
    ble.init(onBluetoothInitialized);

    // wait for Bluetooth Radio to be initialized
    while (ble.hasInitialized()  == false);
    while (1) {
        // save power when possible
        ble.waitForEvent();
    }
}

void blinkHeartbeat(void) {
    /* Do blinky on LED1 to indicate system aliveness. */
    statusLed = !statusLed;
}

void onBluetoothInitialized(
    BLE::InitializationCompleteCallbackContext *params)
{
    BLE&        ble   = params->ble;
    ble_error_t error = params->error;

    // quit if there's a problem
    if (error != BLE_ERROR_NONE) {
        return;
    }
    // Ensure that it is the default instance of BLE
    if(ble.getInstanceID() != BLE::DEFAULT_INSTANCE) {
        return;
    }
    serial.printf("Describing Peripheral...");
    // attach Services
    ble.addService(customService);

    // process disconnections with a callback
    ble.gap().onDisconnection(onCentralDisconnected);
    // advertising parametirs
    ble.gap().accumulateAdvertisingPayload(
        // Device is Peripheral only
        GapAdvertisingData::BREDR_NOT_SUPPORTED |
        // always discoverable
        GapAdvertisingData::LE_GENERAL_DISCOVERABLE);
    // broadcast name
    ble.gap().accumulateAdvertisingPayload(
        GapAdvertisingData::COMPLETE_LOCAL_NAME,
        (uint8_t *)BROADCAST_NAME, sizeof(BROADCAST_NAME)
    );
    //  advertise services
    ble.gap().accumulateAdvertisingPayload(
        GapAdvertisingData::COMPLETE_LIST_16BIT_SERVICE_IDS,
        (uint8_t *)uuid16_list, sizeof(uuid16_list)
    );
    // allow connections
    ble.gap().setAdvertisingType(
        GapAdvertisingParams::ADV_CONNECTABLE_UNDIRECTED);
    // advertise every 1000ms
    ble.gap().setAdvertisingInterval(1000); // 1000ms

    // set the GATT values
    ble.gattServer().write(
        deviceNameCharacteristic.getValueHandle(),
        BROADCAST_NAME,
        characteristicLength
    );
    ble.gattServer().write(
        modelNumberCharacteristic.getValueHandle(),
        modelNumber,
        characteristicLength
    );
    ble.gattServer().write(
        serialNumber.getValueHandle(),
        serialNumber,
        characteristicLength
    );
    ble.gattServer().write(
        batteryLevelCharateristics.getValueHandle(),
        batteryLevel,
        characteristicLength
    );
    // begin advertising
    ble.gap().startAdvertising();
    serial.printf(" done\r\n");
}

void onCentralDisconnected(const Gap::DisconnectionCallbackParams_t *params)
{
    BLE::Instance().gap().startAdvertising();
    serial.printf("Central disconnected\r\n");
}
```

When run, this sketch will create a Peripheral that advertises as "MyDevice" and will have a GATT profile featuring a single Characteristic with read and write permissions (Figure 6-2).

- Device Information Service: 000180a-000-1000-8000-00805f9-b34fb
    - Device Name Charateristic: 0002a00-000-1000-8000-00805f9-b34fb
    - Model Number Charateristic: 0002a24-000-1000-8000-00805f9-b34fb
    - Serial Number Charateristic: 0002a04-000-1000-8000-00805f9-b34fb
- Battery Level Service: 000180f-000-1000-8000-00805f9-b34fb
    - Battery Level Charateristic: 0002a19-000-1000-8000-00805f9-b34fb

Other Books in this Series

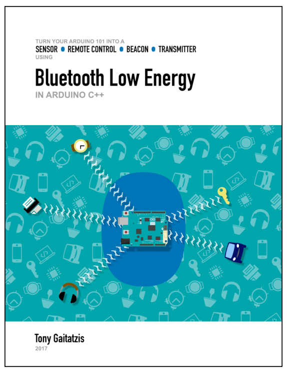
If you are interested in programming Bluetooth Low Energy Devices, please check out the other books in this series or visit bluetoothlowenergy.co:



Bluetooth Low Energy in Android Java

Tony Gaitatzis, 2017

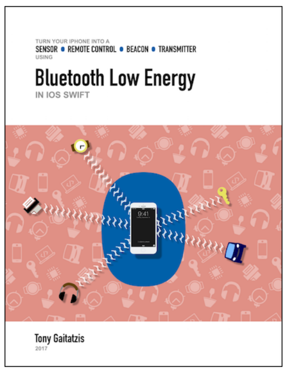
ISBN: 978-1-7751280-4-5



Bluetooth Low Energy in Arduino 101

Tony Gaitatzis, 2017

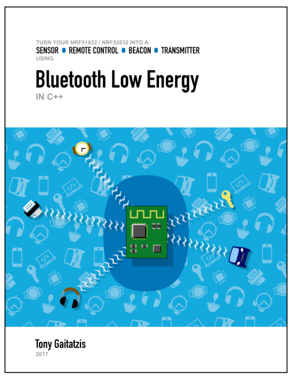
ISBN: 978-1-7751280-6-9



Bluetooth Low Energy in iOS Swift

Tony Gaitatzis, 2017

ISBN: 978-1-7751280-5-2



Bluetooth Low Energy in C++ for nRF Microcontrollers

Tony Gaitatzis, 2017

ISBN: 978-1-7751280-7-6