

Bluetooth Low Energy in Arduino 101

1st Edition

Tony Gaitatzis

BackupBrain Publishing, 2017

ISBN: 978-1-7751280-6-9

backupbrain.co

Bluetooth Low Energy in Arduino 101

by Tony Gaitatzis

Copyright © 2015 All Rights Reserved

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review. For permission requests, write to the publisher, addressed "Bluetooth Arduino Book Reprint Request," at the address below.

backupbrain@gmail.com

This book contains code samples available under the MIT License, printed below:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

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](#)).

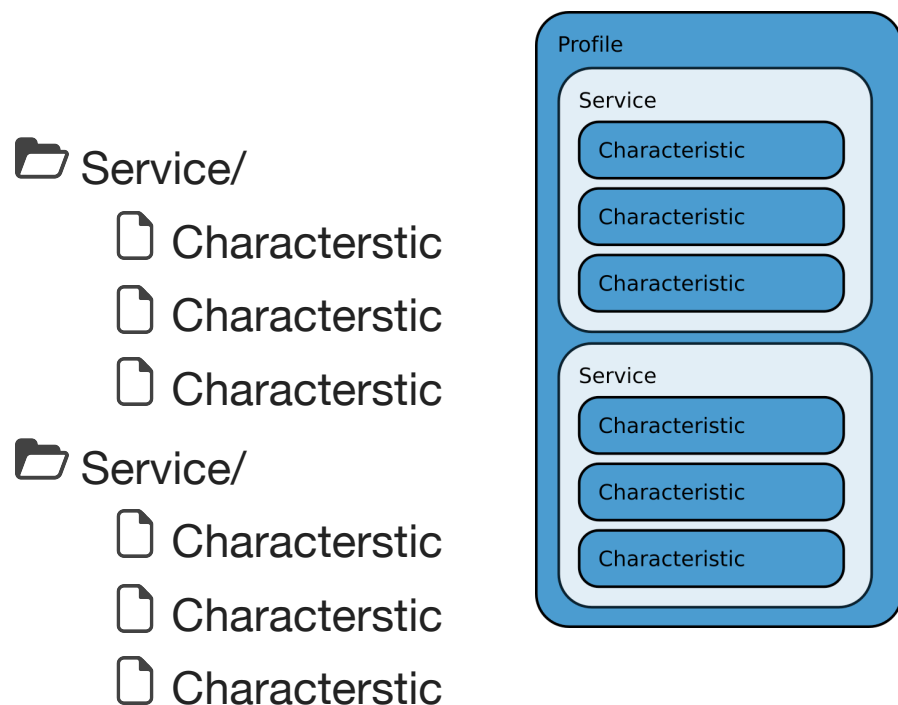


Figure 6-1. GATT Profile filesystem metaphor

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
BleRead	Read	Central can read data altered by the Peripheral
BleWrite	Write	Central can send data, Peripheral reads
BleNotify	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.

Note: 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 Arduino 101, 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:

```
BLECharacteristic readCharacteristic(  
    "1801", BLERead | BLENotify, 2  
);
```

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

```
BLECharacteristic writeCharacteristic("2A56", BLEWrite, 8);
```

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

```
int properties = BLERead | BLEWrite | BLENotify;  
BLEBooleanCharacteristic \  
    booleanCharacteristic(UUID, properties, maxLen);  
BLEIntegerCharacteristic \  
    integerDataCharacteristicName(UUID, properties, maxLen);  
BLEUnsignedIntegerCharacteristic \  
    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()`.

```
...
```

```

BLEPeripheral blePeripheral;
...
blePeripheral.setAdvertisedServiceUuid(service.uuid()); // add service UUID
blePeripheral.addAttribute(service); // Add the BLE Heart Rate service
blePeripheral.addAttribute(readCharacteristic); // add read characteristic
blePeripheral.addAttribute(writeCharacteristic); // add read characteristic
blePeripheral.begin();
...

```

Putting It All Together

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

Example 6-1. `sketches/ble_characteristics/ble_characteristics.ino`

```

#include "CurieBle.h"
static const char* bluetoothDeviceName = "MyDevice";
// Unregulated Service
static const char* serviceUuid = "180C";
// Unregulated Characteristic
static const char* characteristicUuid = "2A56";
// 20 byte transmission
static const int characteristicTransmissionLength = 20;
// create a service
BLEService service(serviceUuid);
// create a characteristic with Read and write attributes
BLECharacteristic characteristic(
    characteristicUuid,
    BLERead | BLEWrite,
    characteristicTransmissionLength
);
BLEPeripheral blePeripheral;
void setup() {
    blePeripheral.setLocalName(bluetoothDeviceName);
    Serial.println(bluetoothDeviceName);
}

```



```
blePeripheral.setAdvertisedServiceUuid(service.uuid()); // attach service
blePeripheral.addAttribute(service);
blePeripheral.addAttribute(characteristic); // attach characteristic
blePeripheral.begin();
}
void loop() {}
```

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).



 Service: 000180c-000-1000-8000-00805f9-b34fb
 Charateristic: 0002a56-000-1000-8000-00805f9-b34fb

Figure 6-2. GATT Profile hosted on the Arduino 101

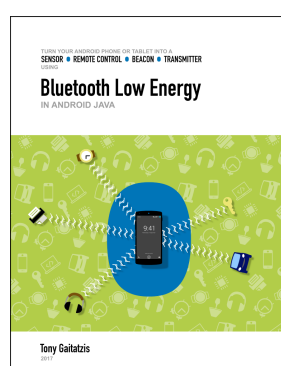
Example code

The code for this chapter is available online

at: <https://github.com/BluetoothLowEnergyInArduino101/Chapter06>

Other Books in this Series

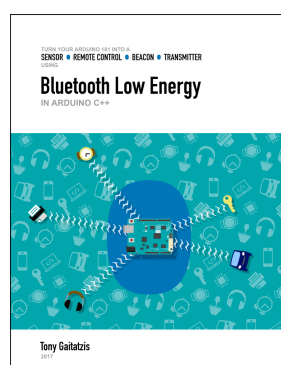
If you are interested in programming Bluetooth Low Energy Devices, please check out the other books in this series or visit bluetoothlowenergy.co:



Bluetooth Low Energy in Android Java

Tony Gaitatzis, 2017

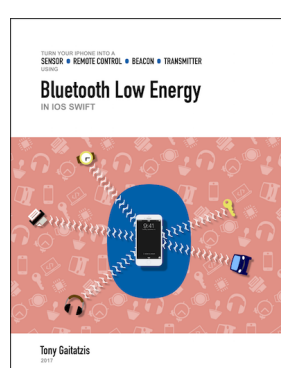
ISBN: 978-1-7751280-4-5



Bluetooth Low Energy in Arduino 101

Tony Gaitatzis, 2017

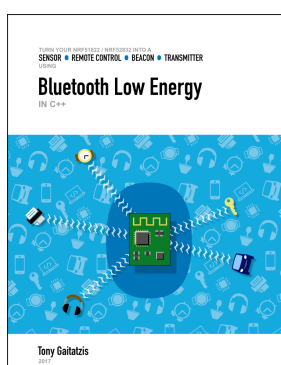
ISBN: 978-1-7751280-6-9



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