Bluetooth for Linux Developers Study Guide

Bluetooth LE Primer

Release	:	1.0.0
Document Version:		1.0.0
Last updated	:	16th November 2021

Contents

1. Revision History

Version	**Date**	**Author**	**Changes**
1.0.0	16th November 2021	Martin Woolley Bluetooth SIG	**Release:** Initial release. **Document**: This document is new in this release.

2. Introduction

In this module you'll become acquainted with some of the most fundamental concepts and terminology associated with Bluetooth Low Energy (LE). You should read the text in full unless you are confident that you already know about and understand the subjects indicated by the section headings.

3. About Bluetooth® profiles

Bluetooth Profiles

You've probably encountered the term *profile* in the context of Bluetooth technology but what does it mean?

A profile is a Bluetooth technical specification which describes device roles, processing algorithms, concurrency limitations, security requirements and state data definitions amongst other things. In particular, and this is the aspect of profiles we'll be focusing on in this study guide, a Bluetooth LE profile defines a remote interface to a device's state data and associated capabilities. Not clear? Read on….

An example will hopefully help; imagine we're creating a Bluetooth device which has a couple of buttons and an LCD display which supports ASCII text characters. As profile designers it's our job to think about how we might make those fundamental device capabilities accessible to a remote device like a smartphone, connected to our device over Bluetooth. We'd probably want the smartphone to be able to send short text strings to our device and have the text scroll across our LCD display. And we'd probably want it to be possible for the smartphone to be informed whenever a user presses either of the buttons on our device. Simple ideas which happily, are also simple to implement using Bluetooth concepts drawn from a part of the Bluetooth core specification called the Generic Attribute Profile or *GATT* for short.

GATT

GATT allows us to define a table of data which represents aspects of our device and their state at any given time and operations which can be carried out against that data. So, in our example, we'd have data that represented the state of the two buttons and the state of the LCD display. We'd support operations like reading the value which represents the state of a button (e.g., *pressed* | *not pressed* | *long press*) or writing some text to the LCD so that it can be displayed.

The table that contains this state data is called the Attribute Table.

GATT allows us to apply a clean, hierarchical structure to our state data in the form of constructs called Services, Characteristics and Descriptors.

Services are top level containers and typically correspond to some kind of primary feature of a device.

Characteristics are individual items of state data, have a defined type, an associated value and support one or more operations. For example, it may be defined that a connected peer device can read the value of one characteristic but cannot write to it. Characteristics belong to a service. Services are containers which contain one or more characteristics and provide a context for the characteristics they contain. The same characteristic type can be a member of more than one service and based on the different contexts these services provide, the rules for using the characteristic might vary.

Descriptors belong to specific characteristics and contain metadata like a textual description for the characteristic or they might provide some means of controlling the behaviour of a characteristic. Descriptors are optional. Characteristics have zero or more descriptors attached to them. For example, GATT defines an operation called *notification* which involves a device sending a message containing a characteristic value to the connected peer asynchronously and without requiring a response from the other device. If a characteristic supports notifications, typically notifications will be transmitted either when the characteristic value changes or periodically, controlled by a timer. But notifications will only be sent if the connected peer device has requested them and this is done by setting a flag in a particular type of descriptor which the characteristic must have if it supports notifications.

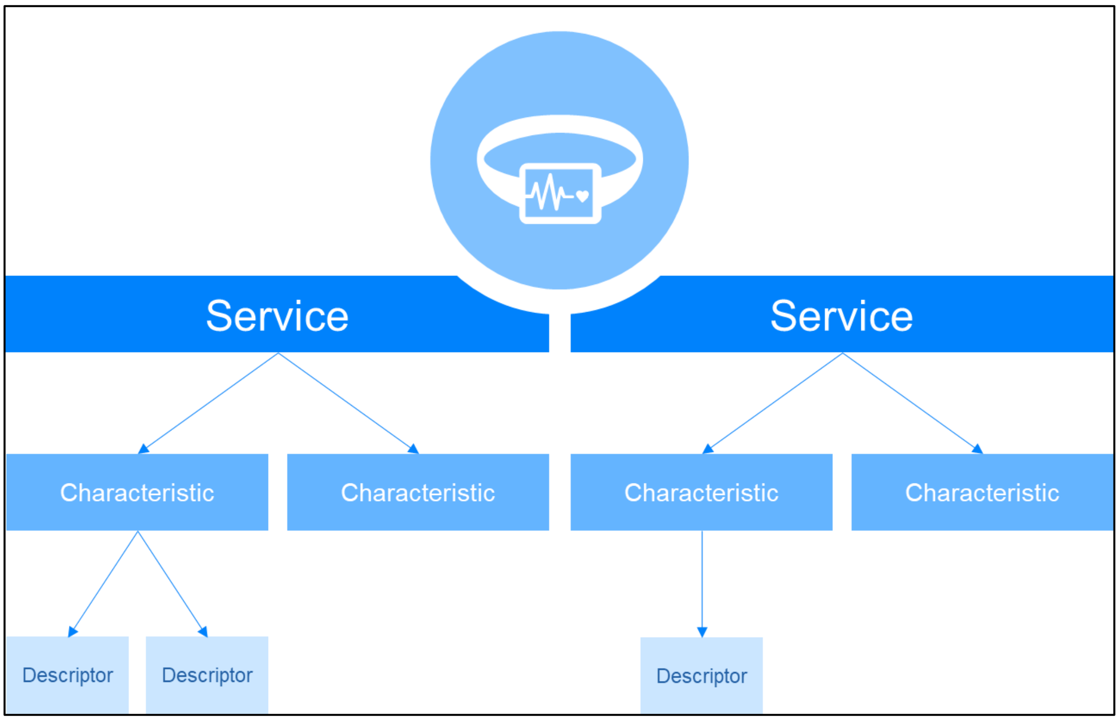


Figure 2 - Services, Characteristics and Descriptors

Indications are similar to notifications but require the client to respond to each indication it receives.

Some services, characteristics and descriptors are defined by the Bluetooth SIG. You can find a list of each type linked off this page: https://www.bluetooth.com/specifications/assigned-numbers

You can define your own custom services, characteristics and descriptors though. This is one of the things that makes Bluetooth LE so flexible.

A device which hosts a set of services, characteristics and descriptors is called a **GATT server**. A device which accesses services, characteristics and descriptors in another device over a Bluetooth LE connection is called a **GATT client**.

GAP

GAP is the Generic Access Profile, another part of a Bluetooth stack. Its primary responsibilities concern how devices discover each other and how connections are then created or taken down.

According to GAP, Bluetooth LE devices play one of 4 possible roles, as defined by that masterpiece, the Bluetooth Core Specification. The four roles are called

- Peripheral
- Central
- Broadcaster
- Observer

A Peripheral advertises, inviting and accepting connections from Central devices. *Advertising* means transmitting small amounts of data quite frequently, which other Bluetooth devices can receive and act upon if they think the advertising device is of interest.

A Central device scans, looking for advertising packets and based on their content, may decide to connect to a device it thinks is suitable, often directed by a user.

A Broadcaster is like a peripheral in that it advertises but it does not accept connections. Its sole purpose is to advertise.

An Observer scans and processes advertising packets but never tries to connect to another device.

Examples:

A heart rate monitor is a Peripheral. A wheel speed sensor on a bike is a Peripheral.

A smartphone application is typically a Central but some devices also support applications which act as Peripherals.

A Bluetooth beacon (iBeacon, AltBeacon, EddyStone and so on) is a Broadcaster.

A beacon application on a smartphone which alerts you to special offers broadcast by beacons for example, is an Observer.

A Peripheral cannot create connections, only accept them. Therefore, a Peripheral cannot connect to another Peripheral. A Peripheral cannot scan for advertising packets from other devices, only transmit them. Therefore, two Peripherals cannot communicate using advertising data only.

A device can have the ability to be both a Peripheral and a Central if it's equipped with all required parts of the Bluetooth stack to support this.

Advertising packets can contain no more than 31 bytes of data in the payload field.

Note: It's common that a device which is a GAP Peripheral will take on the role of GATT server once its been connected to. It's also common for a device in the GAP Central role to become the GATT client after connecting to a remote GAP Peripheral. But GAP and GATT are quite independent and there are no restrictions regarding permutations of GAP and GATT roles. A GAP Central device could just as easily become the GATT server after connecting, for example.

ATT

The Attribute Protocol (ATT) is the layer of the Bluetooth LE stack which allows a connected GATT client to communicate with a GATT server and vice versa. For example, ATT allows clients to discover the GATT services the remote server has in its attribute table, request the current value of a characteristic, change a characteristic value, turn on or off notifications and indications and much more.

GATT provides a structural and contextual representation of a device in terms of services, characteristics and descriptors and ATT allows that data structure to be queried and updated over a Bluetooth LE communications link.

Applications don't usually work directly with ATT. Instead, they use APIs which take care of correctly preparing and transmitting or receiving and decoding ATT protocol data units (PDUs).

Profile Example

So, let's apply this set of ideas to our simple, imaginary device with its display and two buttons. Here's how the profile design might look:

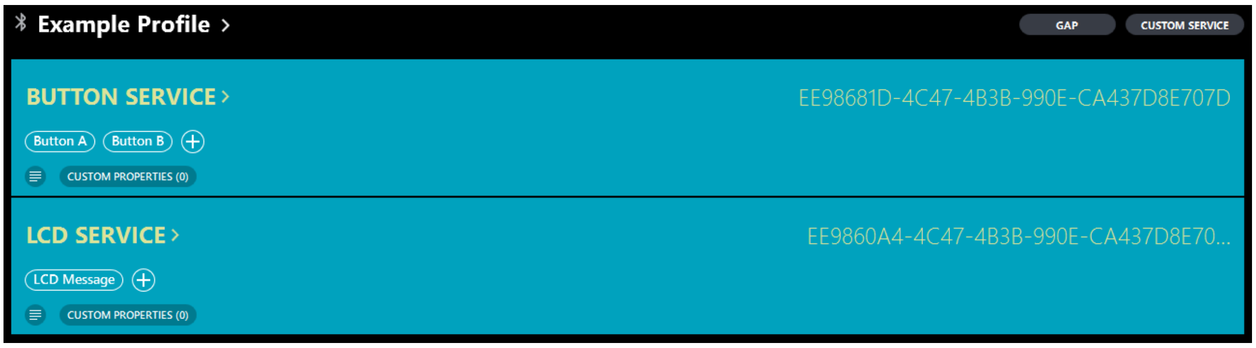
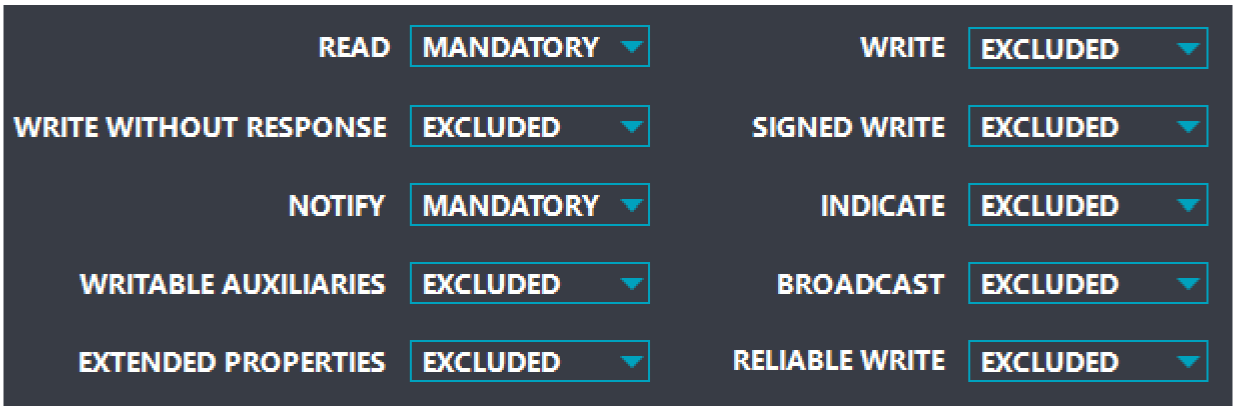


Figure 3 - Example profile design

In Figure 3, the blue bars represent services and as you can see we've chosen to define two, one for the buttons and one for the LCD display. The oval blobs inside each service are characteristics. The Button Service has one for each of the two physical buttons so we can access the state of each button independently. The LCD Service has a single characteristic which will contain the text currently displayed.

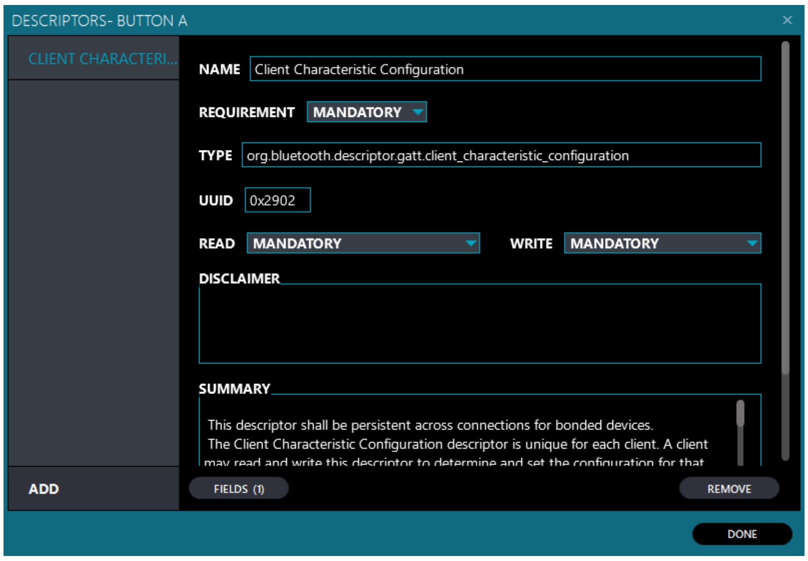
The definition for each of the characteristics includes the Bluetooth operations which are to be supported. Figure 4 shows that READ and NOTIFY are to be supported by the Button A characteristic (and also as you'd expect, Button B).



READ	MANDATORY	WRITE	EXCLUDED
WRITE WITHOUT RESPONSE	EXCLUDED	SIGNED WRITE	EXCLUDED
NOTIFY	MANDATORY	INDICATE	EXCLUDED
WRITABLE AUXILIARIES	EXCLUDED	BROADCAST	EXCLUDED
EXTENDED PROPERTIES	EXCLUDED	RELIABLE WRITE	EXCLUDED

Figure 4 - GATT operations supported by the Button A characteristic

Since we’re supporting notifications, each of the button characteristics has the Client Characteristic Configuration Descriptor attached so that clients can set its value to switch notifications on or off.



DESCRIPTORS- BUTTON A

CLIENT CHARACTERI...

NAME: Client Characteristic Configuration

REQUIREMENT: MANDATORY

TYPE: org.bluetooth.descriptor.gatt.client_characteristic_configuration

UUID: 0x2902

READ: MANDATORY

WRITE: MANDATORY

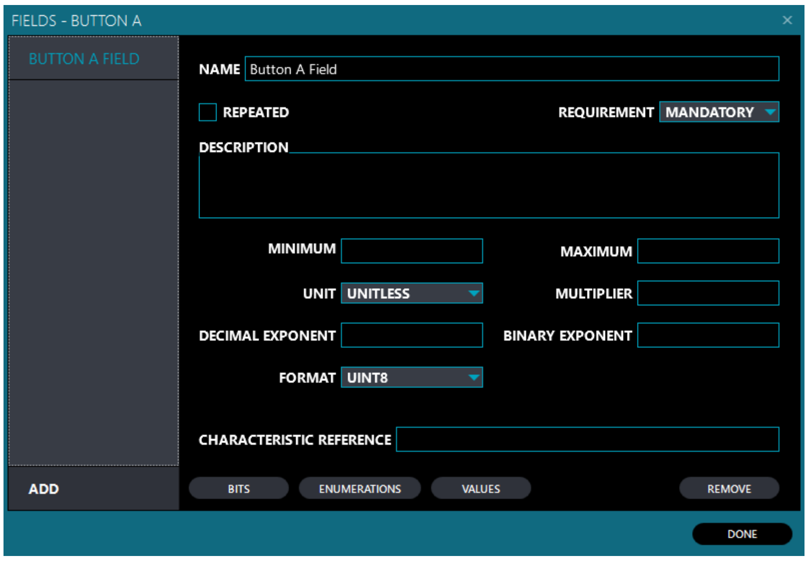
DISCLAIMER

SUMMARY

This descriptor shall be persistent across connections for bonded devices.
The Client Characteristic Configuration descriptor is unique for each client. A client may read and write this descriptor to determine and set the configuration for that

Figure 5 - Descriptor attached to the Button A characteristic

Characteristics have a value and this gets broken down into one or more “fields” which are essentially individual data items. In the case of our example profile, things are nice and simple and each of the characteristics has a value which consists of a single field only. You can see in Figure 6 that the Button A characteristic has a single field of type uint8 (unsigned 8 bits).



FIELDS - BUTTON A

BUTTON A FIELD

NAME: Button A Field

REPEATED: ☐

REQUIREMENT: MANDATORY

DESCRIPTION

MINIMUM

MAXIMUM

UNIT: UNITLESS

MULTIPLIER

DECIMAL EXPONENT

BINARY EXPONENT

FORMAT: UINT8

CHARACTERISTIC REFERENCE

Figure 6 - Button A field definition

You now know enough theory to proceed onto the next exercise.