adafruit learning system

Introduction to Bluetooth Low Energy

Created by Kevin Townsend

Last updated on 2020-06-15 04:22:25 PM EDT

Introduction

Bluetooth Low Energy (BLE), sometimes referred to as "Bluetooth Smart", is a light-weight subset of classic Bluetooth and was introduced as part of the Bluetooth 4.0 core specification. While there is some overlap with classic Bluetooth, BLE actually has a completely different lineage and was started by Nokia as an in-house project called 'Wibree' before being adopted by the Bluetooth SIG.

There are plenty of wireless protocols out there for engineers and product designers, but what makes BLE so interesting is that it's almost certainly the easiest way to design something that can talk to any modern mobile platform out there (iOS, Android, Windows phones, etc.), and particularly in the case of Apple devices it's the only HW design option that doesn't require you to jump through endless hoops to be able to legally market your product for iOS devices.

This guide will give you a quick overview of BLE, specifically how data is organized in Bluetooth Low Energy, and how devices advertise their presence so that you can connect to them and start passing data back and forth.

BLE Platform Support

Support for Bluetooth 4.0 and Bluetooth Low Energy (which is a subset of BT 4.0) is available on most major platforms as of the versions listed below:

- iOS5+ (iOS7+ preferred)
- Android 4.3+ (numerous bug fixes in 4.4+)
- Apple OS X 10.6+
- Windows 8 (**XP, Vista and 7 only support Bluetooth 2.1**)
- GNU/Linux Vanilla BlueZ 4.93+

GAP

GAP is an acronym for the **Generic Access Profile**, and it controls connections and advertising in Bluetooth. GAP is what makes your device visible to the outside world, and determines how two devices can (or can't) interact with each other.

Device Roles

GAP defines various roles for devices, but the two key concepts to keep in mind are **Central** devices and **Peripheral** devices.

- **Peripheral** devices are small, low power, resource contrained devices that can connect to a much more powerful central device. Peripheral devices are things like a heart rate monitor, a BLE enabled proximity tag, etc.
- **Central** devices are usually the mobile phone or tablet that you connect to with far more processing power and memory.

Advertising and Scan Response Data

There are two ways to send advertising out with GAP. The *Advertising Data* payload and the *Scan Response* payload.

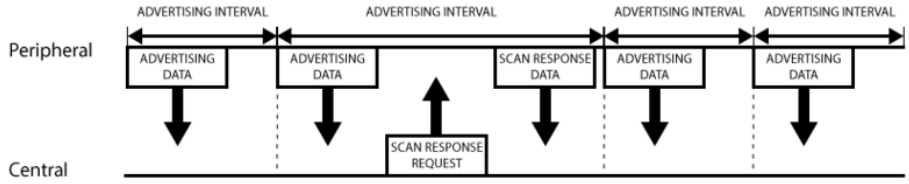
Both payloads are identical and can contain up to 31 bytes of data, but only the advertising data payload is mandatory, since this is the payload that will be constantly transmitted out from the device to let central devices in range know that it exists. The scan response payload is an optional secondary payload that central devices can request, and allows device designers to fit a bit more information in the advertising payload such a strings for a device name, etc.

Advertising Process

The following illustration should explain the advertising process and how the advertising payloads and scan response payloads work.

A peripheral will set a specific advertising interval, and every time this interval passes, it will retransmit it's main advertising packet. A longer delays saves power but feels less responsive if the device only advertises itself once every 2 seconds instead of every 20ms.

If a listening device is interested in the scan response payload (and it is available on the peripheral) it can optionally request the scan response payload, and the peripheral will respond with the additional data.



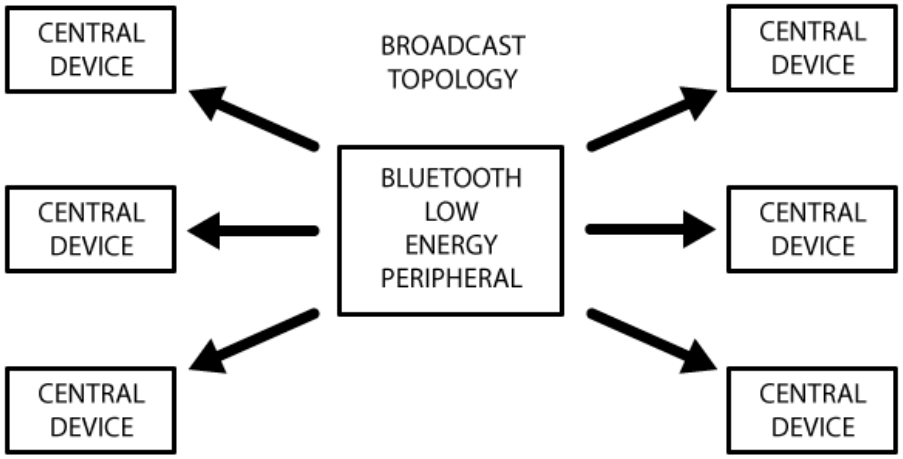
Broadcast Network Topology

While most peripherals advertise themselves so that a connection can be established and GATT services and characteristics can be used (which allows for much more data to be exchanged and in both directions), there are situations where you only want to advertise data.

The main use case here is where you want a peripheral to send data to more than one device at a time. This is only possible using the advertising packet since data sent and received in connected mode can only be seen by those two connected devices.

By including a small amount of custom data in the 31 byte advertising or scan response payloads, you can use a low cost Bluetooth Low Energy peripheral to sent data one-way to any devices in listening range, as shown in the illustration below. This is known as **Broadcasting** in Bluetooth Low Energy.

This is the approach use by Apple's iBeacon, for example, which inserts a custom payload in the main advertising packet, using the **Manufacturer Specific Data** field.



Once you establish a connection between your peripheral and a central device, the advertising process will generally stop and you will typically no longer be able to send advertising packets out anymore, and you will use GATT services and characteristics to communicate in both directions.

GATT

GATT is an acronym for the Generic Attribute Profile, and it defines the way that two Bluetooth Low Energy devices transfer data back and forth using concepts called **Services** and **Characteristics**. It makes use of a generic data protocol called the **Attribute Protocol (ATT)**, which is used to store Services, Characteristics and related data in a simple lookup table using 16-bit IDs for each entry in the table.

GATT comes into play once a dedicated connection is established between two devices, meaning that you have already gone through the advertising process governed by GAP.

The most important thing to keep in mind with GATT and connections is that *connections are exclusive*. What is meant by that is that **a BLE peripheral can only be connected to one central device (a mobile phone, etc.) at a time!** As soon as a peripheral connects to a central device, it will stop advertising itself and other devices will no longer be able to see it or connect to it until the existing connection is broken.

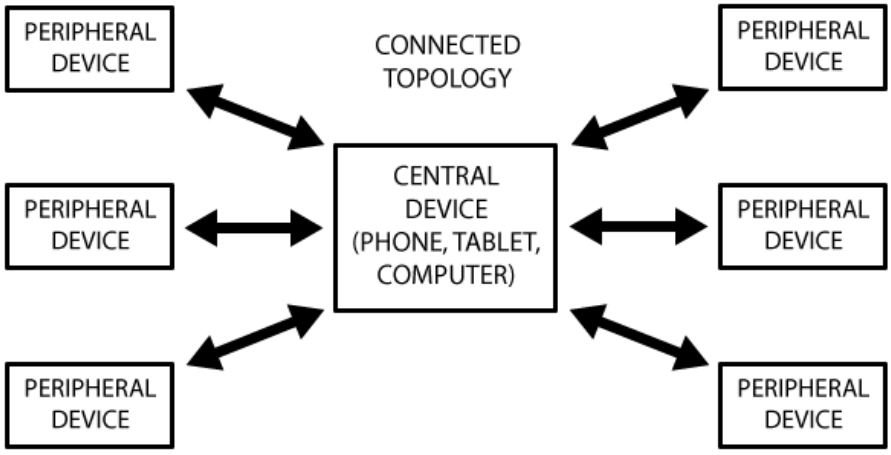
Establishing a connection is also the only way to allow two way communication, where the central device can send meaningful data to the peripheral and vice versa.

Connected Network Topology

The following diagram should explain the way that Bluetooth Low Energy devices work in a connected environment. A peripheral can only be connected to one central device (such as a mobile phone) at a time, but the central device can be connected to multiple peripherals.

If data needs to be exchanged between two peripherals, a custom mailbox system will need to be implemented where all messages pass through the central device.

Once a connection is established between a peripherals and central device, however, communication can take place in both directions, which is different than the one-way broadcasting approach using only advertising data and GAP.



GATT Transactions

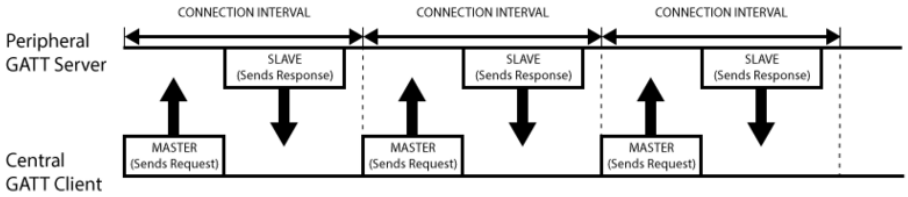
An important concept to understand with GATT is the server/client relationship.

The peripheral is known as the **GATT Server**, which holds the ATT lookup data and service and characteristic definitions, and the **GATT Client** (the phone/tablet), which sends requests to this server.

All transactions are started by the main device, the GATT Client, which receives response from the secondary device, the GATT Server.

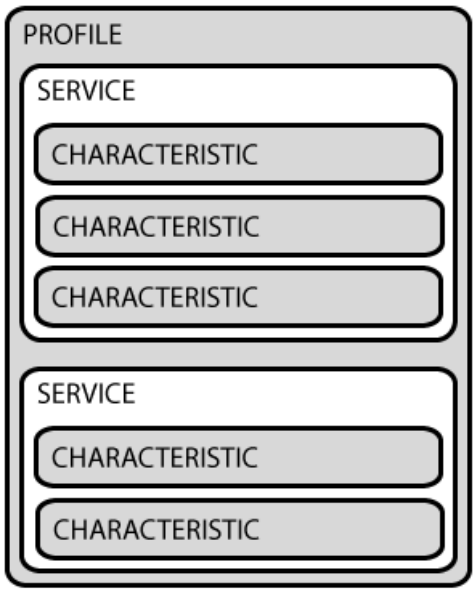
When establishing a connection, the peripheral will suggest a 'Connection Interval' to the central device, and the central device will try to reconnect every connection interval to see if any new data is available, etc. It's important to keep in mind that this connection interval is really just a suggestion, though! Your central device may not be able to honour the request because it's busy talking to another peripheral or the required system resources just aren't available.

The following diagram should illustrate to data exchange process between a peripheral (the GATT Server) and a central device (the GATT Client), with the main device initiating every transaction:



Services and Characteristics

GATT transactions in BLE are based on high-level, nested objects called **Profiles**, **Services** and **Characteristics**, which can be seen in the illustration below:



Profiles

A Profile doesn't actually exist on the BLE peripheral itself, it's simple a pre-defined collection of Services that has been compiled by either the Bluetooth SIG or by the peripheral designers. The Heart Rate Profile, for example, combines the Heart Rate Service and the Device Information Service. The complete list of officially adopted GATT-based profiles can be seen here: Profiles Overview (https://adafru.it/E8u).

Services

Services are used to break data up into logic entities, and contain specific chunks of data called characteristics. A service can have one or more characteristics, and each service distinguishes itself from other services by means of a unique numeric ID called a UUID, which can be either 16-bit (for officially adopted BLE Services) or 128-bit (for custom services).

A full list of officially adopted BLE services can be seen on the Services (https://adafru.it/E8v) page of the Bluetooth Developer Portal. If you look at the Heart Rate Service (https://adafru.it/vaO), for example, we can see that this officially adopted service has a 16-bit UUID of 0x180D, and contains up to 3 characteristic, though only the first one is mandatory: *Heart Rate Measurement*, *Body Sensor Location* and *Heart Rate Control Point*.

Characteristics

The lowest level concept in GATT transactions is the Characteristic, which encapsulates a single data point (though it may contain an array of related data, such as X/Y/Z values from a 3-axis accelerometer, etc.).

Similarly to Services, each Characteristic distinguishes itself via a pre-defined 16-bit or 128-bit UUID, and you're free to use the standard characteristics defined by the Bluetooth SIG (https://adafru.it/E8w) (which ensures interoperability across and BLE-enabled HW/SW) or define your own custom characteristics which only your peripheral and SW understands.

As an example, the Heart Rate Measurement characteristic (https://adafru.it/E8x) is mandatory for the Heart Rate Service, and uses a UUID of 0x2A37. It starts with a single 8-bit value describing the HRM data format (whether the data is UINT8 or UINT16, etc.), and the goes on to include the heart rate measurement data that matches this config byte.

Characteristics are the main point that you will interact with your BLE peripheral, so it's important to understand the concept. They are also used to send data back to the BLE peripheral, since you are also able to write to characteristic. You could implement a simple UART-type interface with a custom 'UART Service' and two characteristics, one for the TX channel and one for the RX channel, where one characteristic might be configured as read only and the other would have write privileges.

Further Information

Bluetooth SIG Resources

- Bluetooth Core Specification (https://adafru.it/E8y)
- Bluetooth Developer Portal (https://adafru.it/E8z)
- Officially Adopted BLE Profiles and Services (https://adafru.it/E8u)
- Officially Adopted BLE Characteristics (https://adafru.it/E8w)