INTRO TO BLUETOOTH LOW ENERGY

by Mohammad Afaneh

edited by Jessica Cast

Table of Contents

Preface

About the author

Mohammad Afaneh has been developing embedded software and firmware since 2006. He has worked at, and consulted for multiple large companies including: Allegion (Schlage locks), Motorola, Technicolor, Audiovox, and Denon & Marantz Group. Throughout his career, he has worked on multiple IoT (Internet of Things) products including: wireless electronic door locks, satellite receivers, wireless doorbells, and various other side projects.

In July 2015, he decided to leave his full-time job to start his own company Novel Bits, LLC where he shares his knowledge and experience through educational resources on his website, via on-site training, and e-books, all focused on Bluetooth Low Energy development.

You can reach Mohammad at his email mohammad@novelbits.io, or by connecting with him on LinkedIn.

Why I wrote this book

When I first started learning BLE, I spent hours, days, and weeks reading every resource I could get my hands on. However, all the ones I came across were either:

> Too detailed, leaving me overwhelmed with all the jargon and details I didn't really care about as a beginner.
>
> Too short, leaving me with many unanswered questions.

This is aside from the fact that the last *read-worthy* book written on Bluetooth Low Energy was published or updated back in 2015! That's 3 years ago, which is an eternity for a rapid changing wireless technology standard such as BLE!

I've even spent days and weeks going through the official **2,800+ page** Bluetooth specification document in an attempt to find answers to the many questions I had. With this book in your hands, my goal is to save you from going down this painful path of learning BLE. Instead, you'll spend a few hours reading through this guide that will teach you the core concepts of BLE and Bluetooth 5 — only what you truly need to get started with this exciting wireless technology.

I hope you really enjoy reading the book and find the content valuable in guiding you on your BLE learning journey!

Who is this book for?

This book is for anyone looking to learn and get started with Bluetooth Low Energy (BLE): whether you're an embedded developer or a mobile developer working on a companion BLE app, you'll need to understand the basics of the Bluetooth Low Energy protocol.

Without understanding the core concepts, you'll be scratching your head, wondering what each API really does each time you call it. Yes, you may be able to cruise through building a mobile app with no issues, but when the time comes for real-life testing and you're hit with something like a connection bug, understanding the BLE protocol and how data is communicated between devices can be crucial, saving you countless hours of debugging and research.

How to read this book

This book is best read in sequence, from beginning to end. It can, however, also be used as a reference if you already have enough knowledge about Bluetooth Low Energy, or you're interested in learning about a specific topic within BLE.

Acknowledgments

I dedicate this work to my mother Ameena, my better half Dana, and my two sons Bassam and Yaseen. Thank you so much for your endless love and support!

— *Mohammad*

1. Basics of Bluetooth Low Energy

1.1. What is Bluetooth Low Energy?

Bluetooth started as a short-distance cable-replacement technology to replace wires in devices such as a mouse, a keyboard, or a PC. If you own a modern car or a smartphone, chances are you’ve used Bluetooth at least once in your life. It's everywhere: in speakers, wireless headphones, cars, wearables, medical devices, and even flip-flops!

The first official version of Bluetooth was released by Ericsson in 1994. It was named after King Harald “Bluetooth” Gormsson of Denmark who helped unify warring factions in the 10th century CE.

There are two types of Bluetooth devices: one is referred to as **Bluetooth Classic (BR/EDR)**, used in wireless speakers, car infotainment systems, and headsets, and the other is **Bluetooth Low Energy (BLE)**. BLE, introduced in Bluetooth version 4.0, is more prominent in applications where power consumption is crucial (such as battery-powered devices) and where small amounts of data are transferred infrequently (such as in sensor applications).

These two types of Bluetooth devices are incompatible with each other even though they share the same brand and even specification document. A Bluetooth Classic device cannot communicate (directly) with a BLE device. This is why some devices such as smartphones choose to implement both types (also called **Dual Mode Bluetooth devices**), allowing them to communicate with both types of devices.

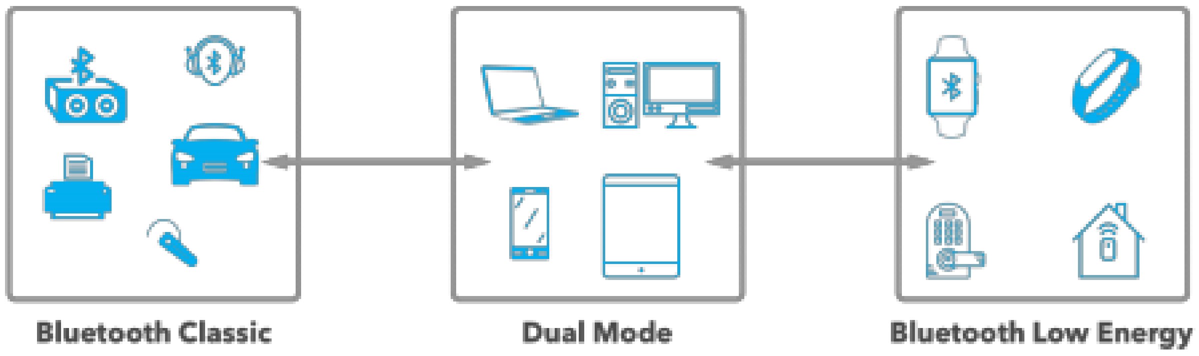


Figure 1*: Types of Bluetooth devices*

Here are a few important notes about BLE:

The official Bluetooth specification document combines both types of Bluetooth (Bluetooth Classic and BLE), sometimes making it difficult to locate BLE-specific specifications.

BLE was introduced in the 4.0 version of the Bluetooth specification, released in 2010. BLE is sometimes referred to as Bluetooth Smart or BTLE, and sometimes mistaken as Bluetooth 4.0 (since this version really included both types of Bluetooth). Both Bluetooth Classic and BLE operate in the same frequency spectrum (the 2.4 GHz Industrial, Scientific, and Medical (ISM) band).

Since many **Internet of Things (IoT)** systems involve small devices and sensors, BLE has become the more common protocol of the two (versus Bluetooth Classic) in IoT. In December 2016, the Bluetooth Special Interest Group (SIG), the governing body behind the Bluetooth standard, released Bluetooth version 5.0 (for marketing simplicity, the point number is removed and the official name is Bluetooth 5). A majority of the enhancements and features introduced in this version focused on BLE, not Bluetooth Classic.

You may have also heard of another term related to Bluetooth: **Bluetooth mesh**. Bluetooth mesh was released in July 2017. It builds on top of BLE and it requires a complete BLE **stack** (a software that acts as an interface for another piece of software or hardware) to work, but it’s not part of the core Bluetooth specification. We’ll talk more about Bluetooth mesh in the chapter “Introduction to Bluetooth Mesh”.

To summarize, here's a figure showing the progression of BLE over the years:

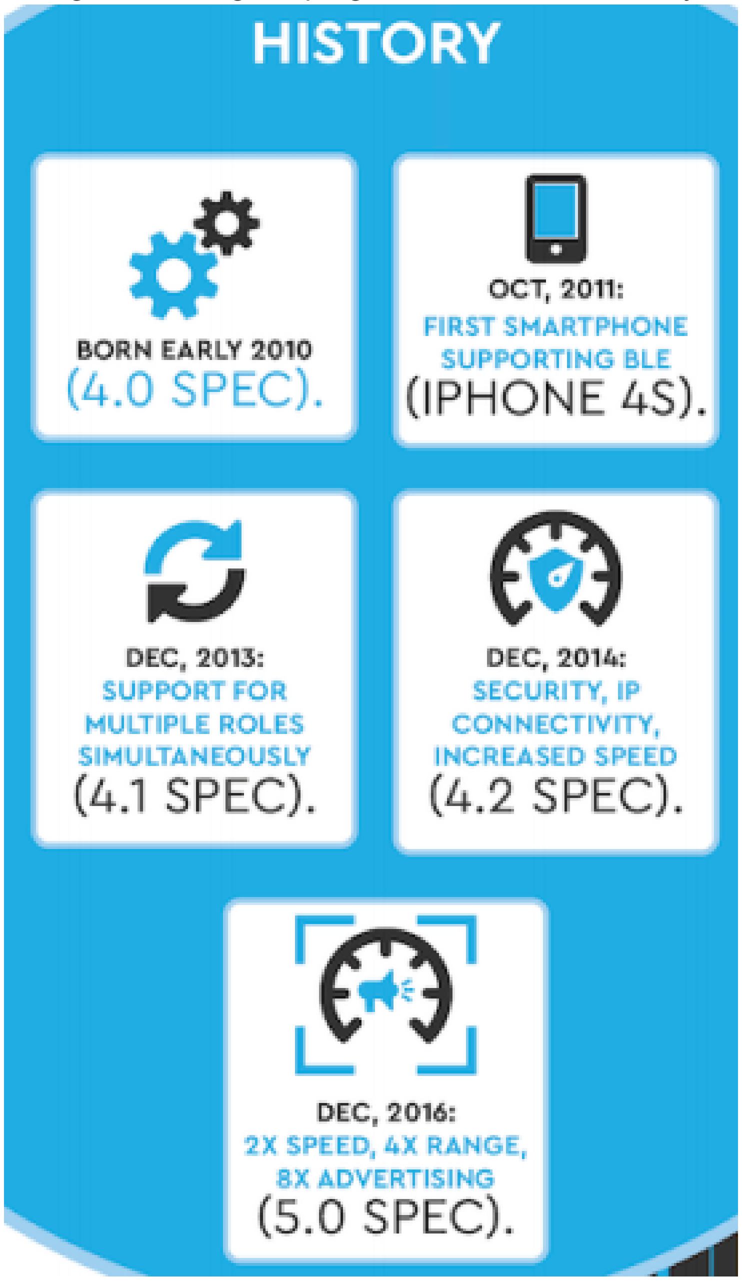

HISTORY
BORN EARLY 2010
(4.0 SPEC).
OCT, 2011:
FIRST SMARTPHONE SUPPORTING BLE
(IPHONE 4S).
DEC, 2013:
SUPPORT FOR MULTIPLE ROLES SIMULTANEOUSLY
(4.1 SPEC).
DEC, 2014:
SECURITY, IP CONNECTIVITY, INCREASED SPEED
(4.2 SPEC).
DEC, 2016:
2X SPEED, 4X RANGE, 8X ADVERTISING
(5.0 SPEC).

***Figure 2**: History of BLE*

1.2 Technical Facts About BLE

Some of the most important technical facts about BLE include:

The frequency spectrum occupied is **2.400 - 2.4835 GHz**.

The frequency spectrum is segmented into **40 “2 MHz”-wide channels**. The maximum data rate supported by the radio (introduced in Bluetooth version 5) is **2 Mbps**.

The range varies significantly depending on the environment surrounding the communicating BLE devices as well as the mode used (for example, in long-range mode, the range will be significantly longer than in the 2M/high-speed mode). A typical range is **10-30 meters (30-100 feet)**.

Power consumption also varies widely. It depends on the implementation of the application, the different BLE parameters, and the chipset used. The peak current consumption of a BLE chipset during radio transmission is typically **under 15 mA**.
Security is **optional** in BLE communication, and it is up to the device and applications developers to implement it. That said, though, there are also varying levels of security that can be implemented.

For all encryption operations, BLE uses **AES CCM with a 128-bit key**. BLE is designed for **low-bandwidth data transfer** applications. Implementing BLE for high-bandwidth applications will significantly compromise the low power consumption promise. So, minimizing radio usage as much as possible achieves the optimal power consumption.

Bluetooth versions (when it relates to BLE) are backwards compatible with each other. However, the communication may be limited to the features of the older version of the two communicating devices.
For example, a Bluetooth 5 BLE device can communicate with a Bluetooth 4.1 BLE device, but 5-specific features won’t be supported. On the other hand, Connections, Primary Advertisements, discovering Services, discovering Characteristics, and reading/writing to these Characteristics are all possible between two BLE devices regardless of their supported Bluetooth version (since they were supported by the initial version of BLE).

1.3. Bluetooth Classic vs. BLE

It’s important to note that there’s a big difference between Bluetooth Classic and Bluetooth Low

Energy in terms of technical specification, implementation, and the types of applications to which they're each suited. This is in addition to the fact that they are **incompatible** with each other.

Some of the notable differences are summarized in the following table:

Bluetooth Classic	**BLE**
Used for streaming applications such as audio streaming, file transfers, and headsets	Used for sensor data, control of devices, and low-bandwidth applications
Not optimized for low power, but has a higher data rate (3Mbps maximum compared to 2Mbps for BLE)	Meant for low power, low duty data cycles
Operates over 79 RF (radio frequency) channels Discovery occurs on 32 channels	Operates over 40 RF channels. Discovery occurs on 3 channels, leading to quicker discovery and connections than Bluetooth Classic

Table 1: Bluetooth Classic vs. BLE

BLE has gone through some major revisions and changes in the short time since its official release in 2010, with the most recent major update being Bluetooth 5 released in December 2016. Bluetooth 5 introduced many important upgrades to the Bluetooth specification, most of which were focused on BLE. Some of the most important enhancements include twice the speed, four times the range, and eight times the advertising data capacity.

1.4. Advantages and Limitations of BLE

Every technology has its limitations, and BLE is no exception. As we mentioned earlier, BLE is most suitable for applications with relatively short range and infrequent low-bandwidth data transfers.

1.4.1. Limitations of BLE

1.4.1.1. Data Throughput

The data throughput of BLE is limited by the physical radio data rate, which is the rate at which the radio transmits data. This rate depends on the Bluetooth version used. For Bluetooth 4.2 and earlier, the rate is fixed at 1 Mbps. For Bluetooth 5 and later, however, the rate varies depending on the mode and **PHY** (discussed later in the Physical Layer section) being used. The rate can be at 1 Mbps like earlier versions, or 2 Mbps when utilizing the high-speed feature. When utilizing the long-range feature, the rate drops to either 500 or 125 Kbps. We'll discuss each of these in more detail in the section on Bluetooth 5.

At the application layer and for the end-user, the data rate is much lower than the radio data rate due to the following factors:

- **Gaps in between packets**: The Bluetooth specification defines a gap of 150 microseconds between packets being transmitted as a requirement for adhering to the specification. This gap is time lost with no data being exchanged between two devices.
- **Packet overhead**: All packets include header information and data handled at levels lower than the application level, which count towards the data being transmitted but are not part of the data utilized by your application.
- **Slave data packets requirement**: The requirement to send back data packets from the slave, even when no data needs to be sent back and empty packets are sent.
- **Retransmission of data packets**: In the case of packet loss or interference from devices in the surrounding environment, the lost or corrupted data packets get resent by the sender.

1.4.1.2. Range

BLE was designed for short range applications and hence its range of operation is limited. There are a few factors that limit the range of BLE including:

- It operates in the 2.4 GHz ISM spectrum which is greatly affected by obstacles that exist all around us such as metal objects, walls, and water (especially human bodies).
- Performance and design of the antenna of the BLE device.
- Physical enclosure of the device which affects the antenna performance, especially if it's

an internal antenna.

Device orientation, which effectively relates to the positioning of the antenna (e.g. in smartphones).

1.4.1.3. Gateway Requirement for Internet Connectivity

In order to transfer data from a BLE-only device to the Internet, another BLE device that has an IP connection is needed to receive this data and then, in turn, relay it to another IP device (or to the internet).

1.4.2. Advantages of BLE

Even with the previously mentioned limitations of BLE, it has presented some significant advantages and benefits over other similar technologies in the IoT space.

Some of these advantages include:

Lower power consumption
Even when compared to other low-power technologies, BLE achieves a lower power consumption than its competitors. It's optimized, and less power consumed, by turning the radio off as much as possible, in addition to sending small amounts of data at low transfer speeds.

No cost to access the official specification documents
With most other wireless protocols and technologies, you would have to become a member of the official group or consortium for that standard in order to access the specification. Becoming a member of those groups can cost a significant amount (up to thousands of dollars per year). With BLE, the major version (4.0, 4.1, 4.2, 5) specification documents are available to download from the Bluetooth website for free. **Lower cost of modules and chipsets** when compared to other similar technologies.

Last but not least, **its existence in most smartphones in the market**. This is probably the biggest advantage BLE has over its competitors such as ZigBee, Z-Wave, and Thread.

1.4.3. Applications Most Suitable for BLE

Based on the limitations and benefits we mentioned earlier, there are a number of use cases where BLE makes the most sense:

Low-bandwidth data
For cases where a device transfers small amounts of data representing sensor data or for controlling actuators, BLE has proven to be a suitable wireless protocol to utilize.
Device Configuration
Even in cases where BLE doesn't satisfy the main requirements of a system, it can still be used as a secondary interface to configure a device before the main wireless connection is established.
For example, some WiFi-enabled devices are adding BLE as a means to configure and establish the WiFi connection of the device instead of using a technology such as WiFi direct (a technology that allows two WiFi devices to connect directly without going through a WiFi router. You can Learn more about it at its Wikipedia page here). **Using a smartphone as an interface**
Small, low-power devices usually don't have large screens and are only capable of displaying limited amounts of data to the end user. Due to the proliferation of smartphones nowadays, BLE can be utilized to offer an alternate, much richer user interface to these small devices (even if just for this sole purpose). Another by-product benefit of using a smartphone is that the data can be relayed up to the cloud. **Personal and wearable devices**
For use cases where a device is portable and can be located in areas where no other persistent wireless connections exist (such as WiFi), BLE can be used (since it's a direct peer-to-peer connection).
Broadcast-only devices
You've probably heard of, and maybe seen, **Beacon** devices before. These devices have one simple task: to broadcast data so other devices may discover them and read their data. There are other technologies that have been used for this kind of application. However, BLE is becoming more and more popular because most people carry smartphones which already support BLE out-of-the-box.

These are all great use cases that could benefit from utilizing BLE. On the other hand, use cases that are not (generally) suitable for BLE include:
Video streaming.
High-quality audio streaming.

Large data transfers for prolonged periods of time (if battery consumption is a concern).

1.5. Architecture of BLE

The following figure shows the different layers within the architecture of BLE. The three main blocks in the architecture of a BLE device are: the **application**, the **host**, and the **controller**.

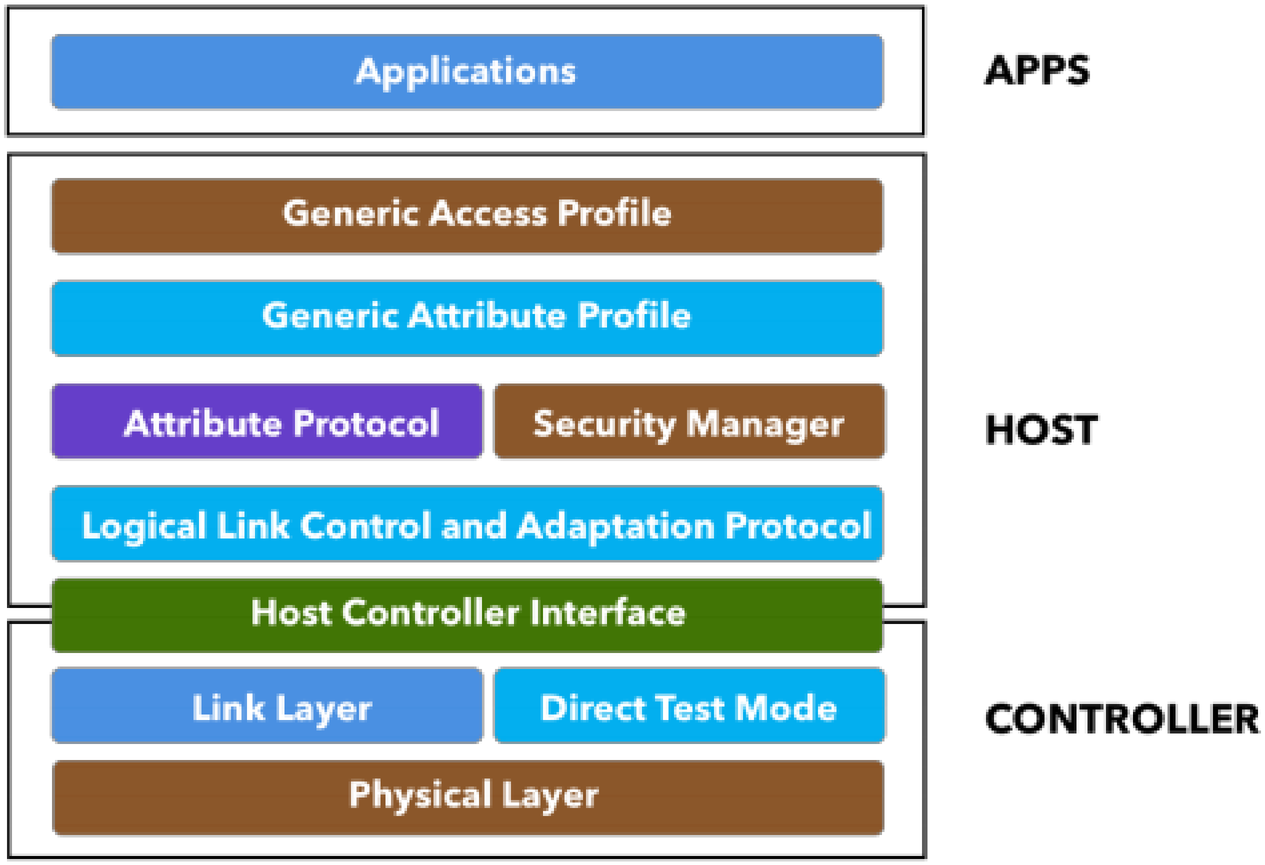


Figure 3*: Architecture of BLE*

In this book, we'll focus on the upper level layers of the architecture, while briefly covering the lower levels of the architecture. We'll go over each of the lower-level layers in this chapter and then look at each of the upper layers (the **Generic Access Profile**, the **Generic Attribute Profile**, the **Attribute Protocol**, and the **Security Manager**) each in their own chapter.

1.5.1. Application

The **application** layer is use-case dependent and refers to the implementation on top of the Generic Access Profile and Generic Attribute Profile — it's how your application handles data received from and sent to other devices and the logic behind it.
This portion is the code that you would write for your specific BLE application and is generally not part of the BLE stack for the platform which you develop. This part will not be covered in the book, since it depends on the specifics of your application and use case.

1.5.2. Host

The **host** contains the following layers:

Generic Access Profile (GAP)

Generic Attribute Profile (GATT)

Attribute Protocol (ATT)

Security Manager (SM)

Logical Link Control and Adaptation Protocol (L2CAP)

Host Controller Interface (HCI) — Host side

1.5.3. Controller

The **controller** contains the following layers:

Physical Layer (PHY)

Link Layer

Direct Test Mode

Host Controller Interface (HCI) — Controller side

1.5.4. Layers of the BLE Architecture

1.5.4.1. Physical Layer (PHY)

The **physical layer (PHY)** refers to the radio hardware used for communication and for modulating/de-modulating the data. BLE operates in the ISM band (2.4 GHz spectrum), which is segmented into 40 RF channels, each separated by 2 MHz (center-to-center), as shown in the following figure:

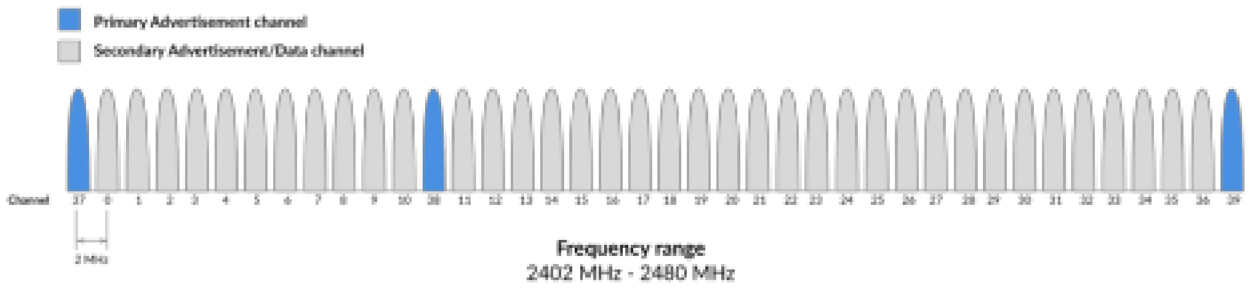


***Figure 4**: Frequency spectrum and RF channels in BLE*

Three of these channels are called the **Primary Advertising Channels**, while the remaining 37 channels are used for **Secondary Advertisements** and for data transfer during a connection. We'll cover these concepts in detail in the chapter titled "**Advertising and Scanning**", but let's briefly cover the concepts here.

Advertising always starts with **advertisement packets** being sent on the three **Primary Advertising Channels** (or a subset of these channels). This allows the devices scanning for **advertisers** to find them and read their **advertisement data**. The **scanner** can then initiate a **connection** if the **advertiser** allows it. It can also request what's called a **scan request**, and if the advertiser supports this scan request functionality, it will respond with a **scan response**. Scan requests and scan responses allow the advertiser to send additional advertisement data to devices that are interested in receiving this data.

Here are some other important technical details pertaining to the Physical Radio:

It uses Frequency Hopping Spread Spectrum (FHSS), which allows the two communicating devices to switch to randomly (agreed-on) selected frequencies for exchanging data. This greatly improves reliability and allows the devices to avoid frequency channels that may be congested and used by other devices in the surrounding environment.

The transmit power levels are:

Maximum: 100mW (+20 dBm) for version >= 5, 10mW (+10 dBm) for version <= 4.2

Minimum: 0.01 mW (-20 dBm)

In older versions of Bluetooth (4.0, 4.1, and 4.2), the data rate was fixed at 1 Mbps. The physical layer radio (PHY) in this case is referred to as the **1M PHY** and is mandatory in all versions including Bluetooth 5. With Bluetooth 5, however, two new optional PHYs were introduced:

- **2Mbps PHY**, used to achieve twice the speed of earlier versions of Bluetooth.
- **Coded PHY**, used for longer range communication.

Note: We'll be covering these two new PHYs as well as the concept of **coding** in more detail in the chapter on Bluetooth 5.

1.5.4.2. Link Layer

The **link layer** is the layer that interfaces with the **physical layer (radio)** and provides the higher-level layers an abstraction and a way to interact with the radio (through an intermediary level called the **HCI layer** which we'll discuss shortly). It is responsible for managing the state of the radio as well as the timing requirements necessary for satisfying the BLE specification. It is also responsible for managing hardware accelerated operations such as: CRC, random number generation, and encryption.

The three main states in which a BLE device operates in are:

- Advertising state
- Scanning state
- Connected state

When a device advertises, it allows other devices that are scanning to find the device and possibly **connect** to it. If the advertising device allows **connections** and a scanning device finds it and decides to connect to it, they each enter into the **connected** state.

The link layer manages the different states of the radio, shown in the following figure:

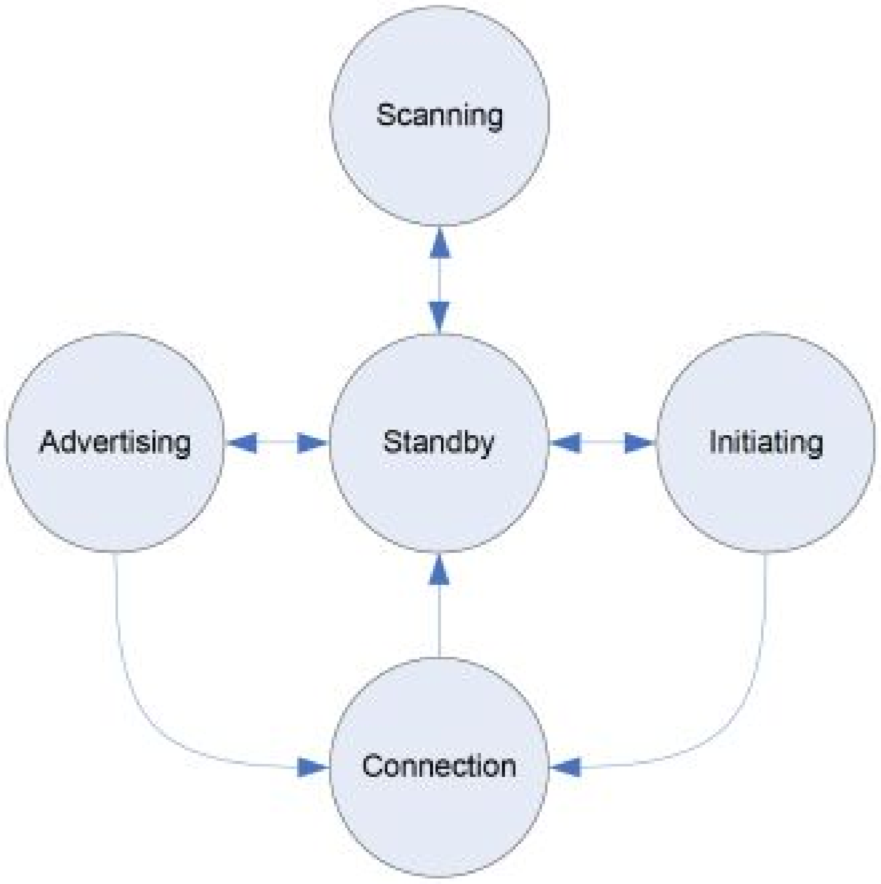


Figure 5*: Link layer states*

Standby: the default state in which the radio does not transmit or receive any data.

Advertising: the state in which the device sends out advertising packets for other devices to discover and read.

Scanning: the state in which the device scans for devices that are Advertising **Initiating**: the state in which a scanning device decides to establish a connection with a device that is advertising.

Connected: the state in which a device has an established link with another device and regularly exchanges data with this other device. This applies to both a device that was in the advertising state or one that was scanning for advertisements and then decided to **initiate** a connection with the advertising device. In this connected state, the device that initiates the connection is called the **master**, and the device that was advertising is now called the **slave**.

We'll be covering advertising, scanning, and connected states in more detail in the later chapters.

Bluetooth Address

Bluetooth devices are identified by a 48-bit address, similar to a MAC address. There are two main types of addresses: **Public Addresses** and **Random Addresses**.

Public Address

This is a fixed address that does not change and is factory-programmed. It must be registered with the IEEE (similar to a WiFi or Ethernet device MAC address).

Random Address

Since manufacturers have a choice on what type of **address** to use (Random vs. Public), Random addresses are more popular since they do not require registration with the IEEE. A random address is programmed on the device or generated at runtime. It can be one of two sub-types:

- Static Address
  - Used as a replacement for Public addresses.
  - Can be generated at boot up OR stay the same during lifetime.
  - Cannot change until a power cycle.
- Private address
  This one is also split up into two additional sub-types:
  - Non-resolvable Private Address:
    - Random, temporary for a certain time.
    - Not commonly used.
  - Resolvable Private Address:
    - Used for privacy.
    - Generated using **Identity Resolving Key (IRK)** and a random number.
    - Changes periodically (even during the lifetime of the connection).
    - Used to avoid being tracked by unknown scanners
    - Trusted devices (or **Bonded**, which is described later in the chapter on **Security**) can resolve it using the previously stored IRK.

1.5.4.3. Direct Test Mode

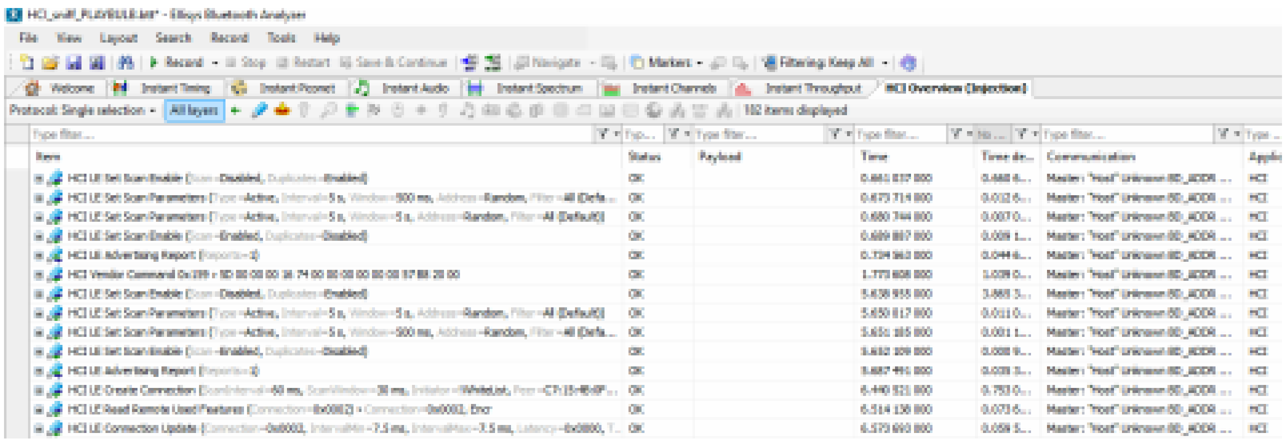
Direct Test Mode (DTM) is only needed for performing RF tests and used during manufacturing and for certification tests. This layer is beyond the scope of this book, so we won’t get into it in any detail.

1.5.4.4. Host Controller Interface (HCI) Layer

The **HCI layer** is a standard protocol defined by the Bluetooth specification that allows the **host** layer to communicate with the **controller** layer. These layers could exist in separate chipsets, or they could exist in the same chipset. In this sense, it also allows interoperability between chipsets, so a device developer can choose two Bluetooth certified devices, a controller and a host, and be 100% confident that they are compatible with each other in terms of communication between the host and controller layers.

In the case where the host and controller are in separate chipsets, the HCI layer will be implemented over a physical communication interface. The three officially supported hardware interfaces by the spec are: UART, USB, and SDIO (Secure Digital Input Output). In the case where the two layers (host and controller) live on the same chipset, the HCI layer will be a logical interface instead.

The job of the HCI layer is to relay commands from the host down to the controller and send events back up from the controller to the host. Following is an example of a capture of HCI commands, HCI events, and ATT commands being exchanged between the host and controller layers:


HCI_sniff_PLAYBULB.btt* - Ellisys Bluetooth Analyzer
File View Layout Search Record Tools Help
Record Stop Restart Save & Continue Navigate Markers Filtering: Keep All
Welcome Instant Timing Instant Piconet Instant Audio Instant Spectrum Instant Channels Instant Throughput HCI Overview (Injection)
Protocol: Single selection All layers 182 items displayed

Item	Status	Payload	Time	Time de...	Communication	Applic
HCI LE Set Scan Enable (Scan=Disabled, Duplicates=Enabled)	OK		0.661 837 000	0.660 6...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Set Scan Parameters (Type=Active, Interval=5 s, Window=500 ms, Address=Random, Filter=All (Defa...	OK		0.673 714 000	0.012 6...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Set Scan Parameters (Type=Active, Interval=5 s, Window=5 s, Address=Random, Filter=All (Default))	OK		0.680 744 000	0.007 0...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Set Scan Enable (Scan=Enabled, Duplicates=Disabled)	OK		0.689 887 000	0.009 1...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Advertising Report (Reports=1)	OK		0.734 563 000	0.044 6...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI Vendor Command 0x199 - 5D 00 00 00 16 74 00 00 00 00 00 00 57 88 20 00	OK		1.773 608 000	1.039 0...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Set Scan Enable (Scan=Disabled, Duplicates=Enabled)	OK		5.638 955 000	3.865 3...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Set Scan Parameters (Type=Active, Interval=5 s, Window=5 s, Address=Random, Filter=All (Default))	OK		5.650 817 000	0.011 0...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Set Scan Parameters (Type=Active, Interval=5 s, Window=500 ms, Address=Random, Filter=All (Defa...	OK		5.651 185 000	0.001 1...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Set Scan Enable (Scan=Enabled, Duplicates=Disabled)	OK		5.652 109 000	0.000 9...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Advertising Report (Reports=1)	OK		5.687 491 000	0.035 3...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Create Connection (ScanInterval=60 ms, ScanWindow=30 ms, Initiator=WhiteList, Peer=C7:15:4B:0F...	OK		6.440 521 000	0.753 0...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Read Remote Used Features (Connection=0x0002) - Connection=0x0002, Error	OK		6.514 136 000	0.073 6...	Master: "Host" Unknown BD_ADDR ...	HCI
HCI LE Connection Update (Connection=0x0002, IntervalMin=7.5 ms, IntervalMax=7.5 ms, Latency=0x0000, T...	OK		6.573 693 000	0.059 5...	Master: "Host" Unknown BD_ADDR ...	HCI

***Figure 6**: Capture of HCI packets*

Examples of the messages include: command packets, configuring the controller, requesting actions, controlling the connection and connection parameters, event packets, command completion and status events.

1.5.4.5. Logical Link Control and Adaptation Protocol (L2CAP) Layer

The **Logical Link Control and Adaptation Protocol (L2CAP)** layer acts as a protocol multiplexing layer. It is borrowed from the Bluetooth Classic standard, and performs the following tasks in the case of BLE:

- Takes multiple protocols from the upper layers and places them in standard BLE packets that are passed down to the lower layers beneath it.
- Handles fragmentation and recombination. It takes the larger packets from the upper layers and splits them into chunks that fit into the maximum BLE payload size supported for transmission. On the receiver side, it takes multiple packets and combines them into one packet that can be handled by the upper layers.

For BLE, the L2CAP layer handles two main protocols: the Attribute Protocol (ATT) (covered in the chapter on GATT), and the Security Manager Protocol (SMP) (covered briefly in the chapter on Security).

1.5.4.6. Upper-level Layers

The Attribute Protocol (ATT), Generic Attribute Profile (GATT), Security Manager (SM) and Generic Access Profile (GAP) will all be covered in detail in the following chapters.

2. BLE Peripherals and Centrals

There are a few important terms that you’ll come across while learning about BLE. Two of the most important are: **BLE central** and **BLE peripheral**. These two terms relate to the role of a BLE device, but they can be confusing sometimes.

Let’s go over each of these terms in a bit more detail.

2.1. Peripherals

A **peripheral** device is a device that announces its presence by sending out **advertising packets** and accepts a connection from another BLE device (*the BLE central — which will be explained shortly*).

Another related term is a **BLE broadcaster**. A broadcaster is a device that sends out advertising packets as well, but with one difference from a peripheral: the broadcaster does not allow a connection from a central device. On the other hand, an **observer** device only discovers advertising devices, but does not have the capability to initiate a connection with the advertiser.

A typical example of an application that involves a broadcaster is in **Beacon** technologies. Beacons are devices that have the sole purpose of advertising and broadcasting their existence, while not accepting connections from other devices. They are becoming popular in two main use cases: retail marketing and indoor location services.

For example, some department stores utilize a smartphone app that can detect Beacons in certain locations within the store. If a customer who has the store's app installed on their smartphone (and has enabled location services) approaches a Beacon, the app displays a special offer to the customer on their phone.

The way a Broadcaster is differentiated from a peripheral device is by the advertising packets that get transmitted by the device. There are different types of advertising packets: some indicate the capability to accept a connection and others are simply for broadcasting presence. When the BLE central discovers the advertising packets of another BLE device (whether broadcaster or peripheral), it knows whether it can initiate a connection or not based on the type of advertising packets.
Once a peripheral gets connected to a BLE central, it also becomes known as the **slave** in that connection. The central device, in this case, gets called the **master**. These are roles defined within the link layer, whereas the peripheral and central roles are defined within the GAP layer.

2.2. Centrals

We've briefly mentioned the BLE Central, but to formally define it:

A **Central** is a device that discovers and listens to other BLE devices that are advertising. It is also capable of establishing a connection to BLE peripherals (usually multiple at the same time).

An **Observer**, on the other hand, is a similar type of BLE device, but one that is not capable of initiating a connection with a peripheral device.

2.3. Observers and Broadcasters vs. Centrals and Peripherals

Let’s go over some advantages and disadvantages of the four different types of device: Observers, Broadcasters, Centrals, and Peripherals.

Broadcaster	**Peripheral**	**Observer**	**Central**
No need for a radio receiver	Needs both a receiver and transmitter	No need for a transmitter	Needs both a receiver and transmitter
No bi-directional data transfer	Supports bi directional data transfer	No bi-directional data transfer	Supports bi directional data transfer
Reduced hardware, reduced BLE software stack	Requires the full BLE software stack	Reduced hardware, reduced BLE software stack	Requires the full BLE software stack

***Table 2**: Comparison between Observers, Broadcasters, Peripherals, and Centrals*

2.4. Power Consumption and Processing Power Considerations

BLE is asymmetrical by design. Much of the heavy lifting regarding connection management, time management, and processing responsibilities lies on the central side. This helps reduce power consumption and processing power requirements on the peripheral side, thus, making it possible to integrate BLE into smaller and more resource-constrained devices (e.g., battery-powered devices).

A BLE central device can still be battery powered, but will usually have a relatively large battery that's rechargeable. Most commonly, in a BLE system, the central device is a smartphone, tablet, or a computer.

A central device also supports connecting to multiple Peripherals at the same time. A typical example of this is a smartphone that maintains a connection to a smartwatch, a smart-home thermostat, and a fitness tracker, all at the same time.

2.5. Multi-Role BLE Devices

In some use cases, a BLE device would benefit from acting in multiple roles simultaneously. For example, a device may want to monitor multiple sensors (peripheral devices), and at the same time be able to advertise its presence to a smartphone to allow access to sensor data from a mobile app interface.

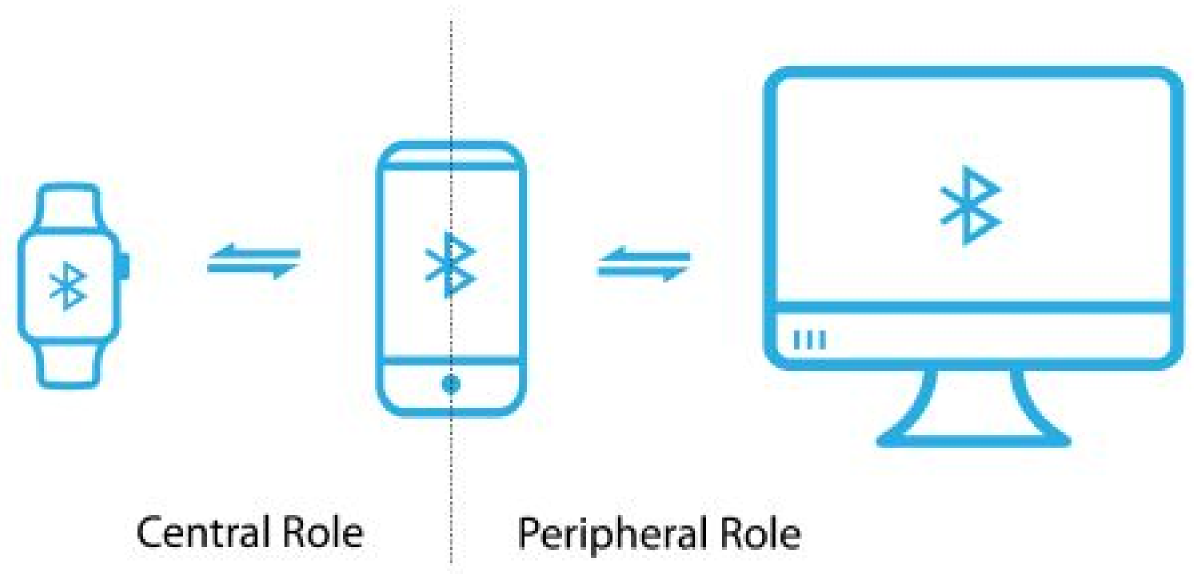


Figure 7*: Multiple roles in BLE*

2.6. The Role of Smartphones in BLE

One of the biggest advantages of BLE over other competing low-power wireless technologies (such as ZigBee, Z-Wave, Thread, etc.) is its existence in the majority of smartphones in the market. Most (if not all) smartphones already included Bluetooth Classic since the very early days, and most Bluetooth chipset vendors are now integrating BLE support along with Bluetooth Classic in their chipsets. The result is that the vast majority of smartphones nowadays support BLE.

Having the capability for a smartphone to interact and connect to BLE devices provides a couple of significant advantages:

Smartphones provide a familiar user interface for consumers, offering a rich user experience when using a mobile app to interface with a BLE device (compared to interfacing with the BLE device directly).

Smartphones are usually connected to the Internet. This means that the data transmitted from the BLE device can be sent up to the cloud and stored somewhere else for later access or analysis.

2.6.1. Challenges with BLE Development on Smartphones

There are two major mobile operating systems: Android and iOS. Android introduced native support for BLE APIs in Android 4.3 (released July 2012), while iOS provided native BLE support a bit earlier in iOS 5 (released October 2011).

One important thing to note is that this also depends on the hardware running the operating system. For iOS, this included all iOS devices starting with the iPhone 4s. For Android, it’s a completely different story: Android runs on devices manufactured by many different vendors, so there’s no easy way to determine which devices first started supporting BLE. This Android fragmentation problem introduces a big challenge with developing Android BLE applications that behave consistently across the dozens of existing Android phones.

3. Advertising and Scanning

3.1. Generic Access Profile (GAP)

The **Generic Access Profile (GAP)** provides the framework that defines how BLE devices interact with each other. This includes the following aspects:

Modes & Roles of BLE devices.

Advertisements (advertising, scanning, advertising parameters, advertising data, scanning parameters).

Connection establishment (initiating, accepting, connection parameters)

Security.

The implementation of this framework is mandatory per the official specification, and it is what allows two or more BLE devices to interoperate, communicate, and be able to exchange data with each other.

We talked briefly about the advertising and scanning states of a BLE device, and we mentioned that a BLE device always starts in the advertising state. This is the case even when it wants to operate in the connected state most of the time. In order for two BLE devices to discover each other, one of them has to advertise while the other scans the three Primary Advertising channels (RF channels 37, 38, and 39) looking for advertisement packets sent by the advertising device.

If the advertising device supports a connection and a central device discovers it, it may choose to establish a connection. In this chapter, we will focus on these initial states: **advertising** and **scanning**.

3.2. Advertising State

In the advertising state, a device sends out packets containing useful data for others to receive and process. The packets are sent at a fixed interval defined as the **advertising interval**.

There are **40 RF channels** in BLE, each separated by 2 MHz (center-to-center), as shown in the following figure. Three of these channels are called the **Primary Advertising Channels**, while the remaining 37 channels are used for **Secondary Advertisements** and for data packet transfer during a connection.

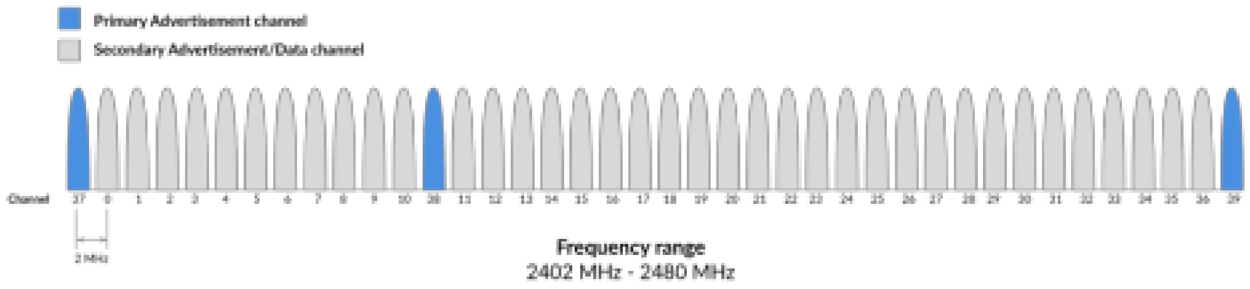


***Figure 8**: RF channels in BLE*

Note: Since these are the three channels that a device starts by advertising on, and usually switches between them, they are spread apart in the frequency spectrum to avoid radio interference between a device that's advertising on one channel and another that's advertising on a different channel. Also, the locations of these primary channels (RF channels 37, 38, and 39) were chosen within the spectrum to avoid interference with the most commonly used WiFi channels.

Advertisements always start with advertisement Packets sent on the three Primary Advertising Channels (or a subset of these channels). This allows centrals to find the advertising device (peripheral) and parse its advertisement packets. The central can then initiate a connection if the advertiser allows it.

The central can also request what's called a **scan request**, and if the Advertiser supports it, it will respond with a **scan response**. Scan requests and responses allow the advertiser to send additional advertising data that would not fit in the initial advertisement packet.

Note: Primary advertisement data is limited to 31 bytes. Secondary advertisement data, on the other hand, supports up to 254 bytes of data.

As we've mentioned before, some devices (broadcasters) stay in the advertising state and do not accept connections (connectionless), while others (peripherals) allow the transition to the connected state if a central initiates a connection (connection-oriented). For example, most Beacons stay in the advertising state during the lifetime of the device.

The main advantage of staying in the advertising state is that multiple centrals can discover the advertising data without the need for a connection. However, the downsides are the lack of security and the inability for the advertiser to receive data from a central (data transfer is unidirectional).

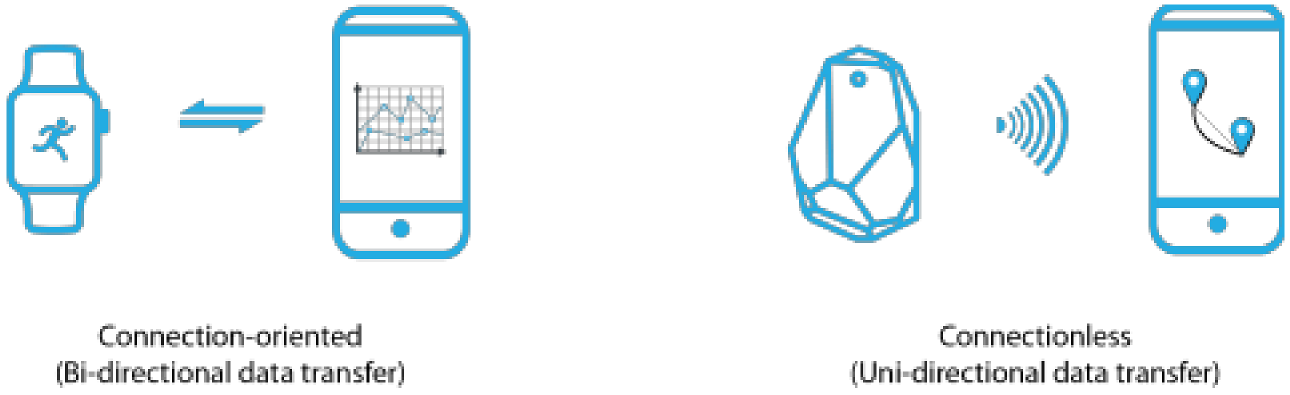

Connection-oriented
(Bi-directional data transfer)
Connectionless
(Uni-directional data transfer)

***Figure 9**: Connection-oriented vs. connectionless*

3.3. Scanning State

Centrals tune to the three Primary Advertising Channels one at a time. So, in order for a central to discover a peripheral, the central has to be tuned to the same channel on which the peripheral is advertising at that given point. To increase the possibility of this happening, and in order to make it happen quickly, a few advertising and scanning parameters can be adjusted.

A device that listens for advertisements, and then sends scan Requests from the advertisers is defined to be in the **active scanning** mode, while a device that passively listens to advertising packets and does not send scan requests is said to be in the **passive scanning** mode.

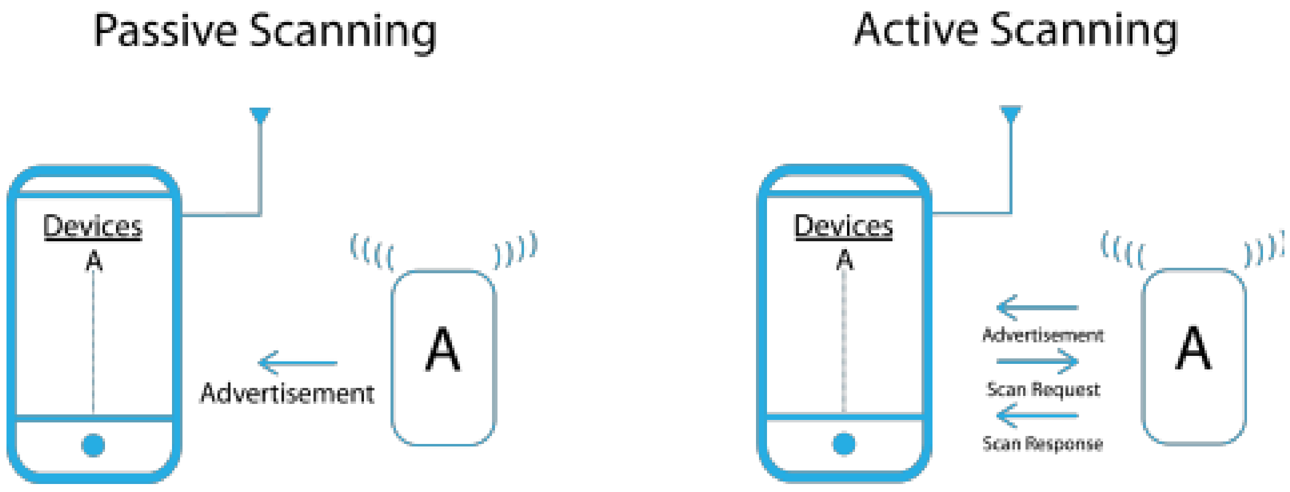


***Figure 10**: Passive vs. Active Scanning*

3.4. Advertising Events

An **advertising event** is made up of multiple advertising packets being sent on all, or a subset of, the three Primary Advertising Channels (37, 38, and 39). There are **seven** types of advertising events (think of these as the different types of advertising packets):

Connectable and Scannable Undirected Event
This type allows other devices to receive the advertisement packets, send a scan request to the advertiser, and establish a connection with it.

Connectable Undirected Event
This type allows other devices to receive the advertisement packets and establish a connection with the advertiser.

Connectable Directed Event
This type allows a **specific** device to receive the advertisement packets and establish a connection with the advertiser.

Non-Connectable and Non-Scannable Undirected Event
This type allows other devices to receive its advertisement packets. However, it does **not** allow scan requests or the establishment of a connection with the advertiser.

Non-Connectable and Non-Scannable Directed Event
This type allows a **specific** device to receive the advertisements without the ability to establish a connection with the advertiser or to send scan requests.

Scannable Undirected Event
This type allows other devices to send a scan request to the advertiser to receive additional advertisement data.

Scannable Directed Event
This type allows a **specific** device to send a scan request to the advertiser to receive additional advertisement data.

3.5. Advertising Parameters

The different **advertising parameters** are:

Advertising Interval
The most important parameter related to advertisements is the **advertising interval**. The advertising interval value ranges all the way from **20 milliseconds** up to **10.24**

seconds in small increments of **625 microseconds**. The advertising interval greatly impacts battery life and should be chosen carefully. It's recommended to choose the longest advertising interval that provides a balance between fast connectivity and reduced power consumption.

Advertising/Scan Response Data

Let's take a look at what fields are usually included in an advertisement packet, and what the packet format looks like. Note that scan responses share the same format.

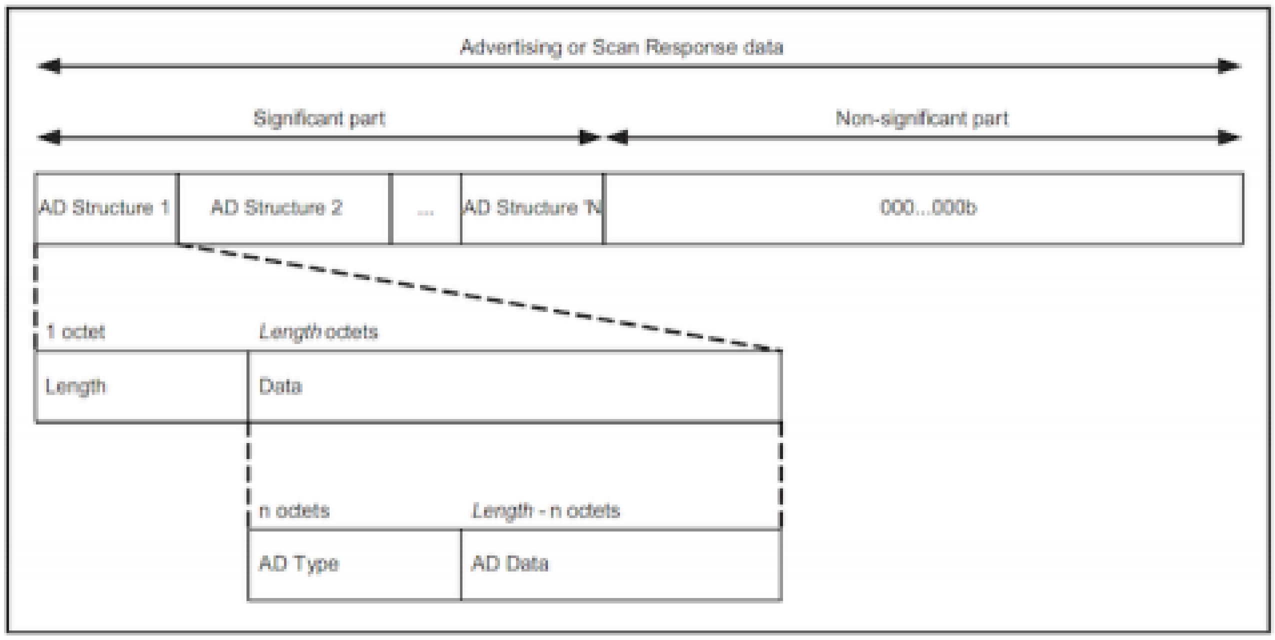


***Figure 11**: Advertising data packet format*
(Source: Bluetooth 5 specification document)

The **advertising data** follows a format for organizing data similar to TLV (Type-Length Value) used in data communications, except that the length comes before the type. The advertising data goes into the PDU portion of the BLE packet and contains the following:

Length: The length of the data that follows the length value itself (includes the AD Type as well as the AD Data).

Advertising Data Type (AD Type): The type of advertisement data included in this specific TLV.

Advertising Data: The actual value of the advertisement data.

Advertising Data (AD) types are defined in the Bluetooth Core Specification Supplement document (not the Core Specification document).

Some of the most commonly used AD Types:

Local Name: contains the device name that is read by scanners that discover the advertising device.

Tx Power Level: transmit power level, defined in units of dBm.

Flags: multiple one-bit boolean (a binary variable, having two possible values: **TRUE** [**1**] or **FALSE** [**0**]) flags, including:

- Limited Discoverable Mode
- General Discoverable Mode
- BR/EDR Not Supported
- Simultaneous LE and BR/EDR to Same Device Capable (controller)
- Simultaneous LE and BR/EDR to Same Device Capable (host)

Note: BR/EDR refers to Bluetooth Basic Rate/Enhanced Data Rate (i.e. Bluetooth Classic).

Service Solicitation: a list of one or more UUIDs indicating what **services** are supported and exposed by the device's GATT server. This helps central devices learn the available services (explained in a later chapter) exposed by a device before establishing a connection.

Appearance: this defines the external appearance of the device according to the Bluetooth Assigned Numbers. These include appearances such as phone, heart rate sensor, key ring and many more.
If you cannot find an appearance that fits the nature of your device, you can use the **UNKNOWN APPEARANCE** value.

3.6. Scanning Parameters

The three main **scanning parameters** are:

Scan Type: Passive vs. Active Scanning.

Scan Window: indicates how long to be scanning for advertisements.

Scan Interval: indicates how often to scan for advertisements.

The scanner will listen for the complete **scan window** at every **scan interval**, and in each scan window it will listen on a different Primary Advertising Channel. Scan window and scan interval are configurable aspects of a scanner's behavior.

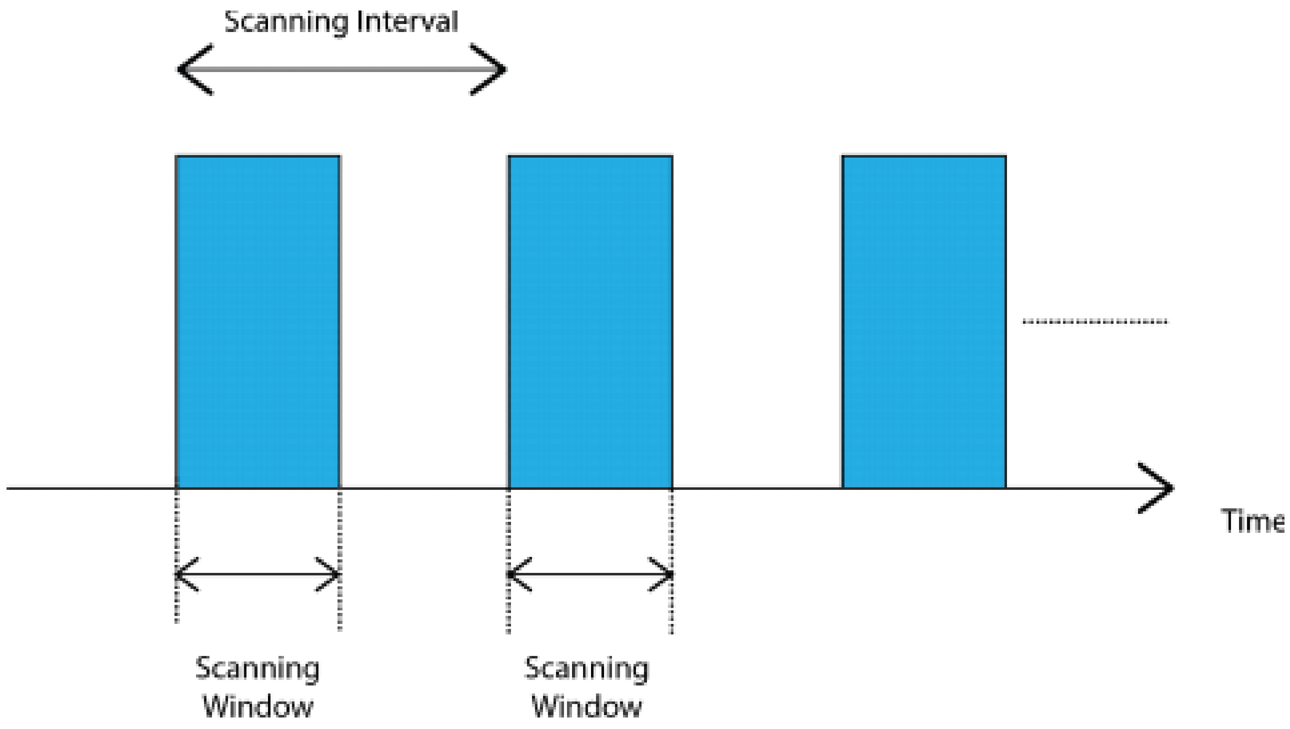


Figure 12*: Scanning parameters*

4. Connections

In order for two BLE devices to connect to each other, the following steps need to occur:

The peripheral needs to start advertising and send out connectable advertisement packets.

The central device needs to be scanning for advertisements while the peripheral is advertising.

If the central happens to be listening on an advertising channel that the peripheral is advertising on, then the central device discovers the peripheral. It is then able to read the advertisement packet and all the necessary information in order to establish a connection.

The central then sends a **CONNECT_IND** packet (also known as a **connection request** packet).

The peripheral always listens for a short interval on the same advertising channel after it sends out the advertising packet. This allows it to receive the connection request packet from the central device — which triggers the forming of the connection between the two devices.

After this occurs, the connection is considered **created**, but not yet **established**. A connection is considered **established** once the device receives a packet from its peer device. After a connection becomes established, the central becomes known as the **master**, and the peripheral becomes known as the **slave**. The master is responsible for managing the connection, controlling the connection parameters, and the timing of the different events within a connection.

4.1. Connection Events

During what's called a **connection event**, the master and slave alternate sending data packets to each other until neither side has more data left to send. Here are a few aspects of connections that are **very important** to know:

A **connection event** occurs periodically and continuously until the connection is closed or lost.

A connection event contains at least one packet sent by the master.

The slave always sends a packet back if it receives a packet from the master If the master does not receive a packet back from the slave, the master will close the connection Event — *it resumes sending packets at the next connection Event*. The connection Event can be closed by either side.

The starting points of consecutive Connection Events are spaced by a period of time called the **connection interval**.

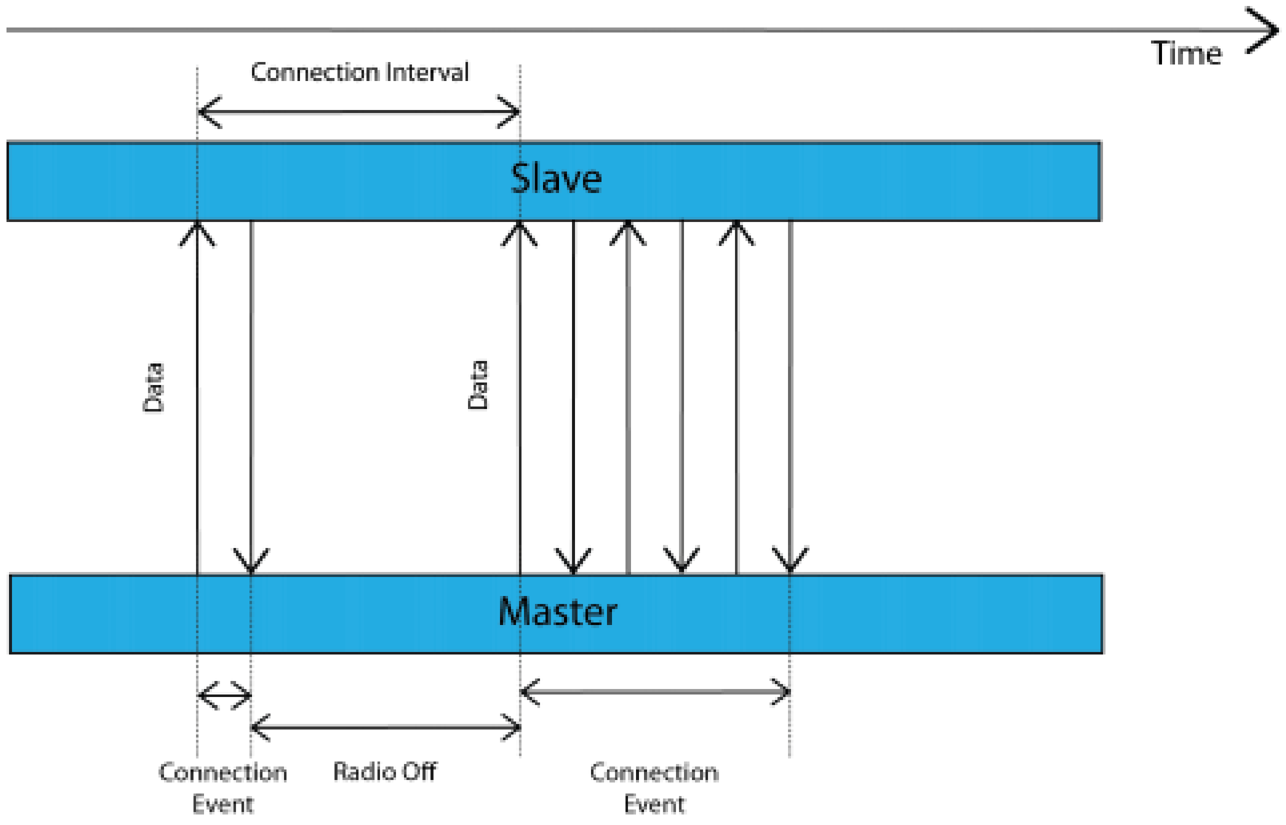


***Figure 13**: The connection interval and connection events*

4.2. Connection Parameters

The parameters that define connections are:

Connection interval

The Connection Interval value ranges between **7.5 milliseconds - 4.0 seconds** in increments of **1.25 milliseconds**. It is set by the central device in the connection request packet. The central may take into account the **Peripheral Preferred Connection Parameters (PPCP)**, which is a way for the peripheral to inform the central of a set of parameters that it prefers. In the end, though, it is up to the central whether to respect these values or ignore them.

Slave Latency

The **slave latency** parameter allows the peripheral to skip a number of consecutive connection Events and not listen to the central at these connection events without compromising the connection. This allows the peripheral to sleep for longer periods of time, potentially reducing power consumption. The slave latency value defines the **number** of connection events it can safely skip.

For example, if the slave latency is set to **three**, then the peripheral may skip **three** consecutive connection events, but it then needs to wake up the radio and listen to the central a to listen — and respond — at every connection event.

Supervision Timeout
The **supervision timeout** is used to detect a loss in connection. It is defined as the **maximum time between two received data packets before the Connection is considered lost**. Its value ranges between **100 milliseconds - 32 seconds** in increments of **10 milliseconds**. Another condition for this timeout value is:

������������������� > (1 + �����������������) *
������������� * 2

The one exception — where the supervision timeout does not apply — is after a connection is **created**, but not yet established. In this case, the master will consider the connection to be lost if it does not receive the first packet from the slave within:

6 * �������������

Data Length Extension (DLE)
This is a setting that can be enabled or disabled. It allows the packet size to hold a larger amount of payload (up to 251 bytes vs. 27 when disabled). This feature was introduced in version 4.2 of the Bluetooth specification.

Maximum Transmission Unit (MTU)
MTU stands for **Maximum Transmission Unit** and is used in computer networking to define the maximum size of a Protocol Data Unit (PDU) that can be sent by a specific protocol. The **Attribute MTU** (**ATT_MTU** as defined by the specification) is the largest size of an ATT payload that can be sent between a client and a server.

The effective ATT_MTU gets determined by the minimum value of the maximum ATT_MTU values that the master and slave support. For example, if a master supports an ATT_MTU of **100 bytes** and the slave responds that it supports an ATT_MTU of **150 bytes**, then the master will decide that the ATT_MTU to be used for the connection from thereon is **100 bytes**.

Note: To achieve maximum throughput, make sure you enable DLE (that is, if you are running Bluetooth 4.2 or greater). This reduces the packet overhead and any unnecessary header data that gets transmitted with smaller packets.

4.3. Channel Hopping

As we discussed at the beginning of this chapter, there are 37 RF channels utilized for transmitting data packets during a connection. However, not all 37 channels are necessarily used during a connection. The **used** channels are defined by the **channel map**, which is included in the connection request packet sent by the central to the peripheral to initiate a connection. For each connection event, the data packets will be sent on a different channel within the channel map.

The sequence of channels used for each of the connection events is determined by the channel map as well as another value called the **hop increment**. The hop increment — like the channel map — is also included in the connection request packet. The combination of the channel map and hop increment determines which channel gets used at each connection interval.

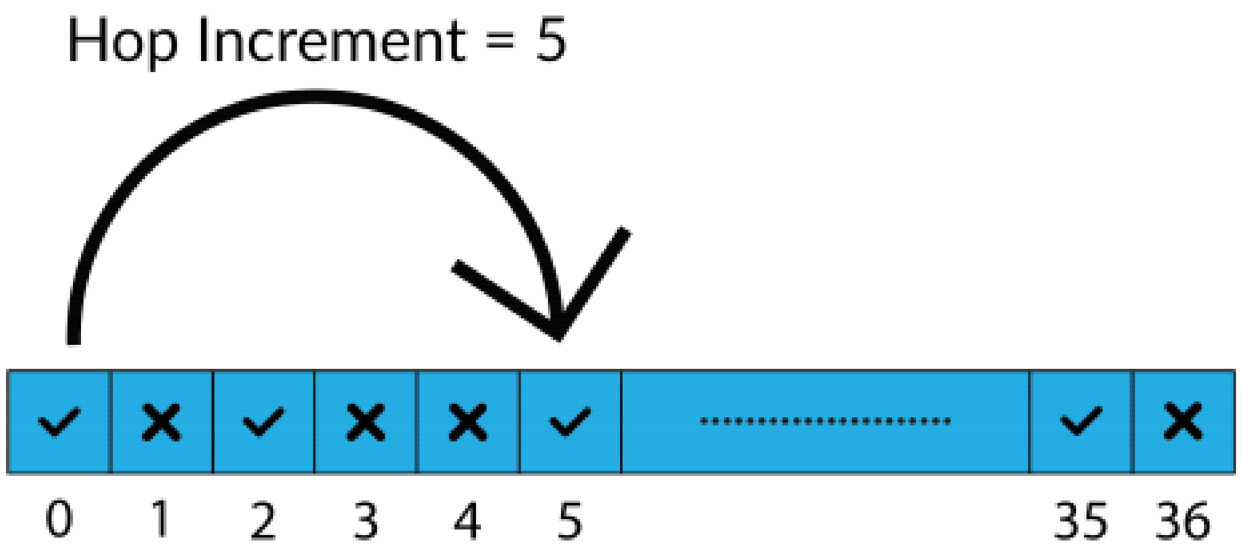


***Figure 14**: Channel map and hop increment*

There are two **channel selection algorithms** used within BLE: **channel selection algorithm #1**, and **channel selection algorithm #2**. Covering the details of these algorithms is outside the scope of this book. To learn more about these algorithms and how they work, refer to the Bluetooth specification document (version 5.0 | Vol 6, Part B, Section 4.5.8.2).

4.4. White List & Device Filtering

BLE supports device filtering for procedures related to: the advertising state, the scanning state, and the initiating state (for establishing connections).

A **white list** is a list of addresses and address types of specific devices. It is used for determining which peer devices a particular device is interested in. An entry for an **anonymous** device address type allows matching all advertisements sent with no address.

Device Filtering gets processed at the link layer in the controller (the lower layer of the Bluetooth stack), which saves time and overhead from being performed at the host (the upper layer of the stack). However, the host is responsible for configuring the white list.

Here's a list of the different white list filter policies that apply to each of these states:

Advertising State Filter Policy (peripheral side)

This filter policy defines how the advertiser processes both scan and connection requests. The different configurations include:

- Process scan and connection requests only from devices in the white list.
- Process scan and connection requests from all devices (white list not used).
- Process scan requests only from devices in the white list, while processing connection requests from all devices.
- Process connection requests only from devices in the white list, while processing scan requests from all devices.

Scanning State Filter Policy (central side)
This filter policy defines how the scanner processes advertising packets. The different configurations include:

- Process advertising packets from all devices (white list not used).
- Process advertising packets only from devices in the white list.

Initiating State Filter policy (central side)
This filter policy defines how a connection initiator processes advertising packets. The different configurations include:

- Process and initiate a connection to all devices listed in the white list.
- Process and initiate a connection only to a device specified by the host.

Notice that it's not an option to process and connect to a connectable advertising device that's not in the white list.

5. Services and Characteristics

Before explaining what **services** and **characteristics** are, we first need to cover two very important concepts: the **Generic Attribute Profile (GATT)** and the **Attribute Protocol (ATT)**.

GATT stands for Generic Attribute Profile. To understand what GATT is, we first need to understand the underlying framework for GATT: the Attribute Protocol (ATT). The GATT only comes into play after a connection has been established between two BLE devices.

Author's note: *If you find GAP and GATT and ATT a confusing set of similar acronyms… don't blame me… I'm just the messenger! That said, it's important to keep them straight!*

5.1. Attribute Protocol (ATT)

ATT defines how a server exposes its data to a client and how this data is structured. There are two roles within the ATT:

Server:
This is the device that exposes the data it controls or contains, and possibly some other aspects of server behavior that other devices may be able to control. It is the device that accepts incoming commands from a peer device, and sends **responses**, **notifications** and **indications**.

For example, a thermometer device will behave as a server when it exposes the temperature of its surrounding environment, the unit of measurement, its battery level, and possibly the time intervals at which the thermometer reads and records the temperature. It can also **notify** the client (defined later) when a temperature reading has changed rather than have the client poll for the data waiting for a change to occur. **Client**:
This is the device that interfaces with the server with the purpose of reading the server's exposed data and/or controlling the server's behavior. It is the device that sends commands and requests and accepts incoming notifications and indications. In the previous example, a mobile device that connects to the thermometer and reads its temperature value is acting in the Client role.

The data that the server exposes is structured as **attributes**. An attribute is the generic term for any type of data exposed by the server and defines the structure of this data. For example, services and characteristics (both described later) are types of attributes. Attributes are made up of the following:

Attribute type (Universally Unique Identifier or UUID)
This is a 16-bit number (in the case of Bluetooth SIG-Adopted Attributes), or 128-bit number (in the case of custom **attribute types** defined by the developer, also sometimes referred to as **vendor-specific UUID**s).

For example, the UUID for a SIG-adopted temperature measurement value is **0x2A1C** SIG-adopted attribute types (UUIDs) share all but 16 bits of a special 128-bit base UUID:

0000**0000-**0000-1000-8000-00805F9B34FB

The published 16-bit UUID value replaces the 2 bytes in **bold** in the base UUID.

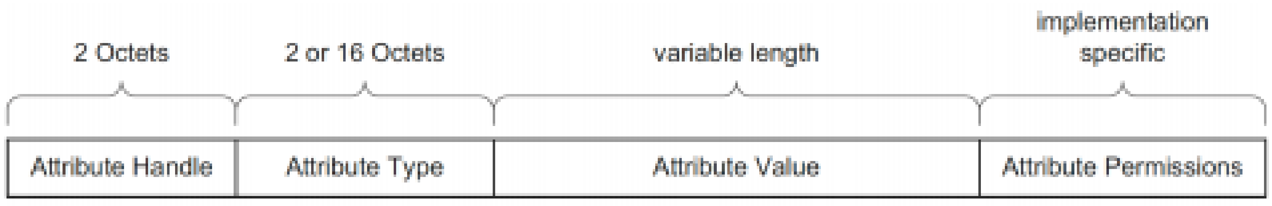
A custom UUID, on the other hand, can be any 128-bit number that does not use the SIG-adopted base UUID. For example, a developer can define their own attribute type (UUID) for a temperature reading as:

F5A1287E-227D-4C9E-AD2C-11D0FD6ED640

One benefit of using a SIG-adopted UUID is the reduced packet size since it can be transmitted as the 16-bit representation instead of the full 128-bit value.

Attribute Handle
This is a 16-bit value that the server assigns to each of its attributes — *think of it as an address*. This value is used by the client to reference a specific attribute, and is guaranteed by the server to uniquely identify the attribute during the life of the connection between two devices. The range of handles is **0x0001-0xFFFF**, where the value of **0x0000** is reserved.

Attribute Permissions
Permissions determine whether an attribute can be **read** or **written** to, whether it can be **notified** or **indicated**, and what **security levels** are required for each of these operations. These permissions are not defined or discovered via the Attribute Protocol (ATT), but rather defined at a higher layer (GATT layer or Application layer).
The following figure shows a logical representation of an Attribute:



(Octets are equivalent to bytes)

Figure 15*: Logical representation of an attribute*
(Source: Bluetooth 5 specification document)

5.2. Generic Attribute Profile (GATT)

Now that we’ve covered the concept of attributes, we’ll go over three important concepts in BLE that you will come across very often:

- **Services**
- **Characteristics**
- **Profiles**

These concepts are used specifically to allow hierarchy in the structuring of the data exposed by the Server. Services and characteristics are types of attributes that serve a specific purpose. Characteristics are the lowest level attribute within a database of attributes. Profiles are a bit different and are not discovered on a server — we will explain them later in this chapter.

The GATT defines the format of services and their characteristics, and the procedures that are used to interface with these attributes such as service discovery, characteristic reads, characteristic writes, notifications, and indications.

GATT takes on the same **roles** as the Attribute Protocol (ATT). The roles are not set per device — rather they are determined per transaction (such as request ↔ response, indication ↔ confirmation, notification). So, in this sense, a device can act as a server serving up data for clients, and at the same time act as a client reading data served up by other servers (all during the same connection).

5.3. Services & Characteristics

5.3.1. Services

A **service** is a grouping of one or more attributes, some of which are characteristics. It’s meant to group together related attributes that satisfy a specific functionality on the server. For example, the SIG-adopted **battery service** contains one characteristic called the **battery level**.

A service also contains other attributes (non-characteristics) that help structure the data within a service (such as **service declarations**, **characteristic declarations**, and others).

Here’s what a service looks like:

***Figure 16**: Profiles, Services, and Characteristics*
(Source: Bluetooth 5 specification document)

From the figure, we can see the different attributes that a service is made up of:

- One or more **include services**
- One or more characteristics
  - Characteristic properties
  - A characteristic value
  - Zero or more characteristic descriptors

An include service allows a service to refer to other services for purposes such as extending the included service. There are two types of services:

- **Primary Service:** represents primary functionality of a device.
- **Secondary Service:** provides auxiliary functionality of a device and is referenced (included) by at least one other primary service on the device (it is rarely used and won’t be referenced in this book).

5.3.2. Characteristics

A **characteristic** is always part of a service and it represents a piece of information/data that a server wants to expose to a client. For example, the battery level characteristic represents the remaining power level of a battery in a device which can be read by a client. The characteristic contains other attributes that help define the value it holds:

- **Properties:** represented by a number of bits and which defines how a **characteristic value** can be used. Some examples include: **read**, **write**, **write without response**, **notify**, **indicate**.
- **Descriptors:** used to contain related information about the characteristic Value. Some examples include: **extended properties**, **user description**, fields used for subscribing to notifications and indications, and a field that defines the presentation of the value such as the format and the unit of the value.

Understanding these concepts is important, however, as an application developer you’ll probably interface with APIs provided by the chipset or mobile operating system SDK that abstract out many of these concepts.

For example, you may have an API for enabling notifications on a certain characteristic that you can simply call (you don’t necessarily need to know that the stack ends up writing a value of **0x0001** to the characteristic's **Client Characteristic Configuration Descriptor (CCCD)** on a server to enable notifications).

It's important to keep in mind that while there are no restrictions or limitations on the characteristics contained within a service, services are meant to group together related characteristics that define a specific functionality within a device.

For example, even though it's technically possible — it does not make sense to create a service called the **humidity service** that includes both a **humidity** characteristic and a **temperature** characteristic. Instead, it would make more sense to have two separate services specific to each of these two distinct functionalities (temperature reading, and humidity reading).

It's worth mentioning that the Bluetooth SIG has adopted quite a few services and characteristics that satisfy a good number of common use cases. For these adopted services, specification documents exist to help developers implement them along with ensuring conformance and interoperability with this service.

If a device claims conformance to a service, it must be implemented according to the service specification published by the Bluetooth SIG. This is essential if you want to develop a device that is guaranteed to be connectable with third-party devices from other vendors. The Bluetooth SIG-adopted services make the connection specification "pre-negotiated" between different vendors.

You can find the list of **adopted services** here, and their respective **specifications** here. **Adopted characteristics** can be found here.

5.4. Profiles

Profiles are much broader in definition than services. They are concerned with defining the behavior of both the client and server when it comes to services, characteristics and even connections and security requirements. Services and their specifications, on the other hand, deal with the implementation of these services and characteristics on the server side only.

Just like in the case of services, there are also **SIG-adopted** profiles that have published specifications. In a profile specification, you will generally find the following:

Definition of roles and the relationship between the GATT server and client.

Required Services.

Service requirements.

How the required services and characteristics are used.

Details of connection establishment requirements including advertising and connection parameters.

Security considerations.

Following is an example of a diagram taken from the Blood Pressure Profile specification document. It shows the relationship between the roles (server, client), services, and characteristics within the profile.

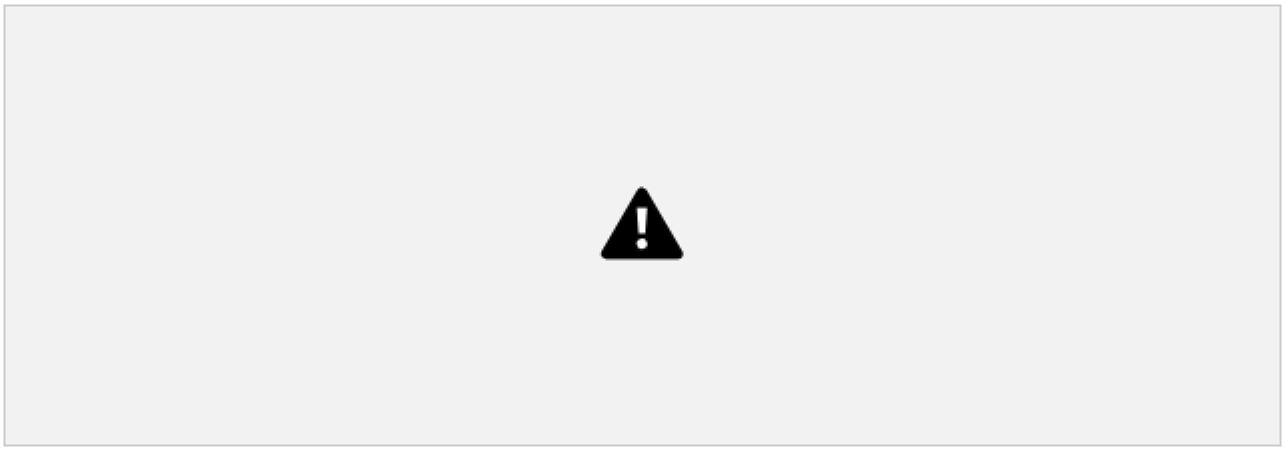


Figure 17*: Blood pressure profile*
(Source: Blood Pressure Profile specification document)

The roles are represented by the yellow boxes, whereas the services are represented by the orange boxes. You can find the list of **SIG-adopted profiles** here.

5.5. Example GATT

Let's look at an example of a GATT implementation. For this example, we'll look at an example **GATT.xml** file that's used by the Silicon Labs Bluetooth Low Energy development framework (BGLib).

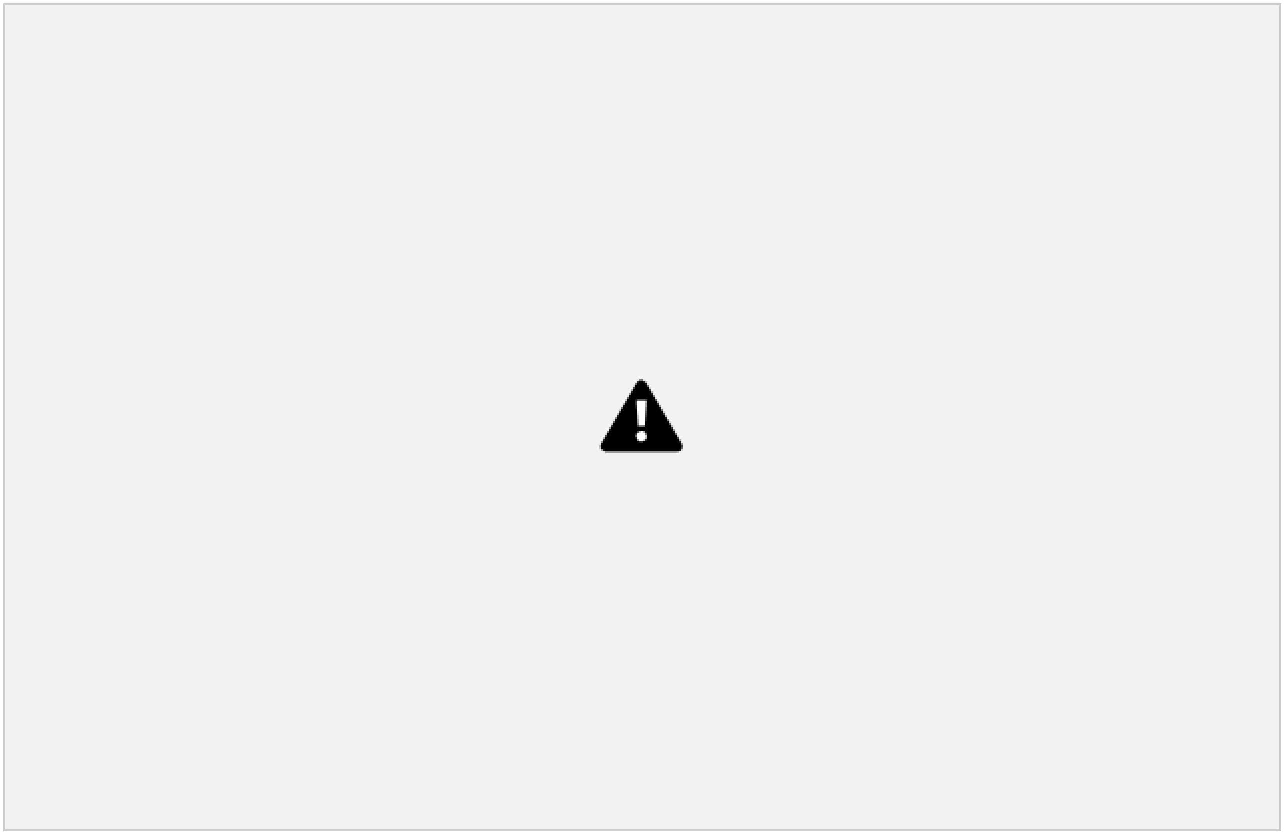


Figure 18*: GATT.xml example from Silicon Labs sample application*

In this XML, you'll notice the following:

There are two services defined:

Generic Access Profile (GAP) service with UUID: **0x1800** (SIG-adopted service).

Cable Replacement service with UUID: **0bd51666-e7cb-469b-8e4d 2742f1ba77cc** (a custom or vendor-specific service).

The Generic Access Profile service is mandatory per the spec, and it includes the following mandatory characteristics:

Name with UUID **0x2a00** and value: **Bluegiga CR Demo**.

Appearance with UUID **0x2a01** and value **0x4142**.

Appearance value definitions can be found here.

Note: the creation and inclusion of this Service is usually handled by the chipset's SDK, and usually APIs are provided to simply set the Name and Appearance values. The Cable Replacement service has one characteristic named **data**

The **data** characteristic has a UUID: **e7add780-b042-4876-aae1-112855353cc1**

It has both **writes** and **indications** enabled.

5.6. Attribute Operations

There are six different types of attribute operations. They are:

Commands: sent by the client to the server and do not require a **response** (*defined below*).

Requests: sent by the client to the server and require a response. There are two types of requests:

Find Information Request

Read Request

Responses: sent by the server in response to a request.

Notifications: sent by the server to the client to let the client know that a specific characteristic value has changed. In order for this to be triggered and sent by the server, the client has to enable notifications for the characteristic of interest. Note that a notification does not require a response from the client to acknowledge its receipt.

Indications: sent by the server to the client. They are very similar to notifications, but require an acknowledgment to be sent back from the client to let the server know that the indication was successfully received.

Note: Notifications and indications are exposed via the **Client Characteristic Configuration Descriptor (CCCD)** attribute. Writing a “1” to this attribute value enables notifications, whereas writing a “2” enables indications. Writing a “0” disables both notifications and indications.

Confirmations: sent by the client to the server. These are the acknowledgment packets sent back to the server to let it know that the client successfully received an indication.

5.6.1. Flow Control and Sequence of Attribute Operations

Requests are sequential in nature and require a response from the server before a new request can be sent. Indications have the same requirement: a new indication cannot be sent before a confirmation for the previous indication is received by the server.

Requests and indications, however, are mutually exclusive in terms of the sequence requirement. So, an indication can be sent by the server before it responds to a request that it had received earlier.

Commands and notifications are different, and do not require any flow control — they can be sent at any time. Because of this — and because a server or client may not be able to handle these packets (due to buffer or processing limitations) — they are considered unreliable. When reliability is a concern, requests and indications should be used instead.

5.6.2. Reading Attributes

Reads are **requests** by nature since they require a response. There are different types of reads. Here we list the two most important ones:

> **Read Request:** a simple request referencing the attribute to be read by its **handle**. **Read Blob Request:** similar to the read request but adds an offset to indicate where the read should start, returning a portion of the value. This type of read is used for reading only part of a characteristic's value.

5.6.3. Writing To Attributes

Writes can be either commands or requests. Here are the most common types of writes:

> **Write Request:** as the name suggests, this requires a response from the server to acknowledge that the attribute has been successfully written to.
>
> **Write Command:** this has no response from the server.
>
> **Queued Writes (atomic operation behavior):** these are classified as requests and require a Response from the server. They are used whenever a large value needs to be written and does not fit within a single message. Instead of writing parts of the value and risking someone else reading the incorrect (partial) value, two types of write requests are used to make sure the operation completes safely:
>
> > **One or more Prepare Write Requests:** each includes an offset at which the sent value should be written within the attribute value. The sent values are also referred to

as **prepared values**, and they get stored in a buffer on the server side — not written to the attribute yet. This operation requires a response from the server. **One Execute Write Request:** used to request from the server to either execute or cancel the write operation of the prepared values. It requires a response from the server and once a response has been received from the server, the client can now be sure that the attribute holds the complete value it sent to the server.

5.6.4. Exchange MTU Request

The server and the client agree on a common value that is used for both data transfer directions. The client is the side that sends this exchange MTU request packet, and can only send it once per connection (per the Bluetooth specification, version 5.0, Vol 3, Part F, Section 3.4.2.1).

The server then responds with an exchange MTU response packet indicating the ATT_MTU it can support. The agreed-on value then becomes the minimum of the ATT_MTU values exchanged between the client and server.

It’s important to know that different BLE stacks have different maximum values of ATT_MTU that they can support.

5.7. Designing your GATT

5.7.1. General Guidelines

While GATT is a pretty flexible framework, there are a few general guidelines to follow when designing it and creating the services and characteristics within it. Following are some recommendations:

Make sure to implement the following mandatory service and its characteristics:

- **Generic Access Profile** (GAP) service.
- **Name** and **Appearance** characteristics within the GAP service.

One thing to keep in mind is that vendor SDKs usually do not require you to explicitly implement this service, but rather they provide APIs for setting the **name** and **appearance**. The SDK then handles creating the GAP service and setting the

characteristics according to the user provided values.
Utilize the Bluetooth SIG-adopted profiles, services, and characteristics in your design whenever possible. This has the following benefits:

You get the benefit of reducing the size of data packets involving UUIDs for services and characteristics (including advertisement packets, discovery procedures, and others) — since 16-bit UUID values are used instead of 128-bit values. Bluetooth chipset and module vendors usually provide implementations of these profiles, services, and characteristics in their SDKs — reducing development time considerably.

Interoperability with other third-party devices and applications, allowing more devices to interface with your device and provide a richer user experience. Group characteristics that serve related functionality within a single service. Avoid having services with too many characteristics. A good separation of services makes it faster to discover certain characteristics and leads to a better GATT design that's modular and user-friendly.

In the next chapter, we'll go over a practical example showing how to design the GATT for a BLE home automation system.

6. GATT Design Exercise

Designing the GATT for your BLE device can be a challenge. To make this task easier, let's go through a complete exercise of designing the GATT for a simple BLE home automation system.

The home automation system is a hypothetical one, but one that will help you better understand the steps taken in designing the GATT of a real-life application, rather than some other generic, abstract system. Here's a diagram showing the different elements of the home automation system and how they interact with each other.

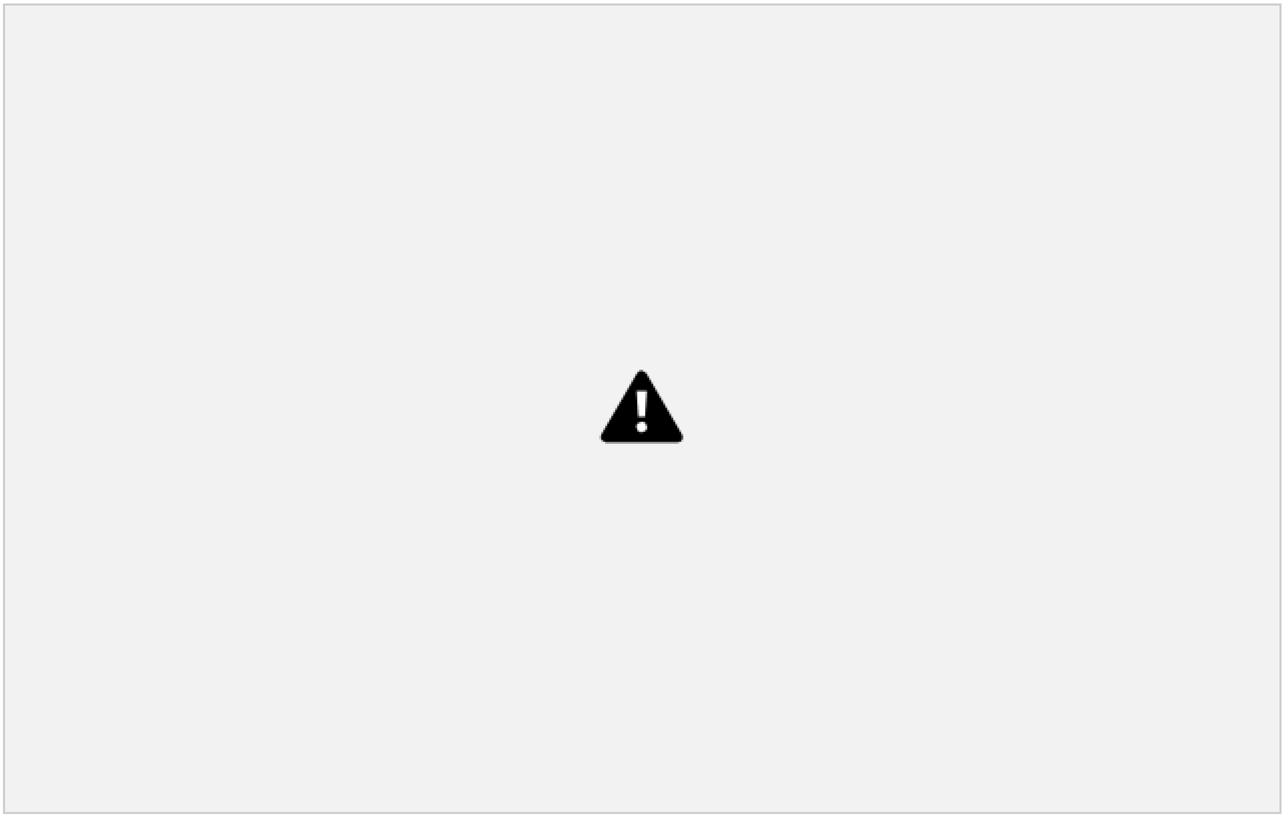


Figure 19*: BLE home automation project example*

The system consists of multiple elements (devices). Some of these are off-the-shelf components that we don't have control over, while others are devices whose firmware we do have control over and to which we will design their GATT structure.

6.1. General System Description

Let’s go ahead and describe the main user scenarios of the system:

v. The homeowner can use the **remote control** to turn on/off the **Bluetooth lightbulb**. w. The homeowner can monitor changes in the temperature and humidity of the **environment sensor**.

x. The homeowner is notified of the battery levels of the **remote control**, **Bluetooth lightbulb**, and **environment sensor**.

6.2. System Elements

Now, let's go over the different elements within the system.

v. **Gateway**

The **gateway** will act as a BLE central when communicating with all the other devices except the smartphone, where it will act as a peripheral. We have control over this device and we will be designing its GATT.
The commands for controlling the Bluetooth lightbulb will be routed from the remote control through the gateway and to the Bluetooth lightbulb.

w. **Remote Control**

The **remote control** is a device that will act as a peripheral only, and one that we will be designing the GATT for.

x. **Environment Sensor**

This is an off-the-shelf device over whose GATT design we have no control, so we will simply be interested in reading the data from it (temperature and humidity readings). y.

Bluetooth Lightbulb

This is another off-the-shelf device over whose GATT we have no control. z.

Smartphone

This is also another existing device over whose GATT we have no control.

6.3. GATT Design

Let’s go through the GATT design process, step-by-step:

6.3.1. Step 1: Documenting the Different User Scenarios and Data Points

Even though the GATT is usually more focused on the peripheral role (since a peripheral is usually the server exposing the data), the central can still act as the server in some cases for specific data points it needs to expose. Also, since we are designing both sides of the system (peripheral and central on the gateway), it helps to think in terms of what needs to happen from each side since this could affect some aspects of the system and GATT design.

6.3.1.1. Gateway

The gateway device acts in both the central and peripheral role. Each of these roles are used to enable communication with different devices within the system. The main purpose of the gateway is to act as a central device to read data from multiple peripherals. It then exposes this information via the peripheral role to another central device (the smartphone) that is able to relay this data to a **cloud server**.

Let's go through the user scenarios from the gateway's perspective for each of the central and peripheral roles.

Peripheral Role

The remote control notifies the gateway when specific buttons are pressed to turn on/off the Bluetooth lightbulb.

The following data points within the system need to be reported up to a cloud server via the gateway. These data points get exposed as a GATT Server in the peripheral role to the central device (the smartphone) that has an Internet connection (which allows it to relay this data up to the cloud server).

Environment sensor temperature reading

Environment sensor humidity reading

Bluetooth lightbulb status (on/off)

Individual battery levels for the environment sensor, Bluetooth lightbulb, and the remote control.

Central Role

The gateway device needs to read some of the data exposed by devices within the system and get notified of other data points exposed by these devices.

6.3.1.2. Remote Control

The remote control provides one main function: turning the Bluetooth lightbulb on or off. It acts strictly in the peripheral role and needs to expose the following data points:

On button press: the gateway needs to be notified of this event when it occurs. **Off button press**: the gateway needs to be notified of this event when it occurs. **Battery level**: the gateway needs to be able to read this data and be notified when it changes.

6.3.2. Step 2: Define the Services, Characteristics, and Access Permissions

The next step is to group the **characteristics** into meaningful groups (**services**) based on their functionalities and define the access permissions for each of these characteristics.

6.3.2.1. Gateway

We have one GATT Server for the gateway, and it exists for the peripheral role. By looking at the data points (characteristics) we listed previously, we can group them into the following services:

Environment Sensor Service:

Environment sensor temperature reading characteristic: "Temperature".
Access permissions: Read, notify.

Environment sensor humidity reading characteristic: "Humidity".
Access permissions: Read, notify.

Battery level characteristic: "Battery Level".
Access permissions: Read, notify.

Playbulb Service:

Light status characteristic: "Light Status".
Access permissions: Read, notify.

Battery level characteristic: "Battery Level".
Access permissions: Read, notify.

Remote Control Service:

Battery level characteristic: "Battery Level".
Access permissions: Read, notify.

In addition to these services, it is mandatory (per the Bluetooth specification) to implement the following service:

GAP service:

Name characteristic: the device name.
Access: Read.

Appearance characteristic: a description of the device.

Access: Read.

6.3.2.2. Remote Control

We have one GATT server for the remote control. We can define the following services and characteristics:

GAP service (mandatory):

Device name characteristic: "Device Name".

Access permissions: Read.

Appearance characteristic.

Access permissions: Read.

Battery service:

Battery level characteristic : "Battery Level".

Access: Read, notify.

Button service:

On button characteristic: "On Button Press".

Access permissions: Notify.

Off button characteristic: "Off Button Press".

Access permissions: Notify.

6.3.3. Step 3: Re-use Bluetooth SIG-Adopted Services & Characteristics

6.3.3.1. Gateway

The environment sensor, Bluetooth lightbulb, and remote control services are all custom services, since there are no SIG-adopted ones that can be utilized for them. We have three devices for which we need to expose the battery level, so we can reuse the SIG-adopted battery level characteristic. We will reuse it for each device within each device's service in the gateway GATT. We'll also re-use the mandatory GAP service.

6.3.3.2. Remote Control

For the **remote control**, we can reuse both the battery service and the mandatory GAP service.

6.3.4. Step 4: Assign UUIDs to Custom Services and Characteristics

For any **custom services** and **characteristics** within the GATT, we can use an online tool to generate UUIDs such as the Online GUID Generator.

A common practice is to choose a base UUID for the custom service and then increment the **3rd** and **4th Most Significant Bytes (MSB)** within the UUID of each included characteristic.

For example, we could choose the UUID:

0000**0001**-1000-2000-3000-111122223333

for a specific service and then

0000000**[N]**-1000-2000-3000-111122223333, (where **N > 1**) for each of its characteristics.

The only restriction for choosing UUIDs for custom services and characteristics is that they must not collide with the Bluetooth SIG base UUID:

XXXXXXXX-0000-1000-8000-00805F9B34FB

However, following the previously mentioned common practice makes it a bit easier to relate services and their characteristics to one another.

The following tables list the services and characteristics along with their UUIDs for each of the gateway and remote control devices:

__Table 3__: Gateway GATT design

__Table 4__: Remote control GATT design

6.3.5. Step 5: Implement the Services and Characteristics Using the Vendor SDK APIs

Each platform, whether it's an embedded or mobile one, has its own APIs for implementing services and characteristics. This is left to the reader to implement for the specific platform they choose for their BLE device or application.

7. Bluetooth 5

Bluetooth 5 focuses on broadening the range of Internet of Things (IoT) applications that can utilize BLE. It brings us twice the speed, four times the range, and eight times the advertising capacity.

Let’s go over the list of new features introduced by Bluetooth 5 (from the Bluetooth specification document):

New features added in version 5:

CSA 5 features (Higher Output Power)
Slot Availability Mask (SAM)
2 Msym/s PHY for LE
LE Long Range
High Duty Cycle Non-Connectable Advertising
LE Advertising Extensions
LE Channel Selection Algorithm #2

In this chapter, we’ll focus on the most important of these changes:

2 Msym/s PHY for LE (2x the speed)
LE Long Range (4x the range)
LE Advertising Extensions (8x the Advertising capacity)

To learn more about the other new features introduced in Bluetooth 5 and not covered in this chapter, refer to the Bluetooth 5 specification document.

Note: Msym/s (Megasymbols per second) is used here instead of Mbps because it refers to the actual radio transmission capability. In some cases (as we will see for the Coded PHY), multiple symbols will be used to represent a single bit, therefore reducing the Mbps rate. In the remainder of this chapter, we will be using "M" for short in place of "Msym/s".

7.1. Twice the Speed, Four Times the Range

Recall that a PHY refers to the physical radio. The Bluetooth specifications before Bluetooth 5 allowed a single PHY, operating at 1 MSym/sec.

As we’ve mentioned previously, Bluetooth 5 introduced two new (optional) PHYs:

v. **2M PHY**:

Used to achieve twice the speed of earlier versions of Bluetooth. It offers a couple of extra benefits as well:

Reduced power consumption, since the same amount of data is transmitted in less time, thus reducing radio-on time.

Improvement of wireless coexistence because of the decreased radio-on time.

One downside to using the 2M PHY is that it has the potential of reducing the range, as the higher speed results in a decrease in radio sensitivity on the receiving end. Another is that the use of the 2M PHY is restricted to the secondary advertisement and data channels. It's important to note that this new PHY represents a hardware change, so older chipsets and modules may not support it.

w. **Coded PHY**:

Used to achieve four times the range of earlier versions of Bluetooth. The obvious benefit of using the **coded PHY** is increased range, with two trade-offs: **Higher power consumption**: due to the fact that we're transmitting multiple symbols to represent one bit of data, resulting in longer radio-on time to transmit the same amount of data.

Reduced speeds: due to the fact that more bits are needed to transmit the same amount of data (125 kbps or 500 kbps, depending on the **coding scheme** used — *explained below*).

Ranges as far as 800 meters line-of-sight have been recorded while testing with the coded PHY. This makes it possible to use BLE in applications such as ones that require communication with a device hundreds of meters away.

The data rates we discussed above define the rate at which the radio transmits raw data. When it comes to the application data rate — in terms of how much bandwidth your application can utilize — these numbers are reduced.

This is due to mandatory (time) gaps in between packets (150 microseconds, per the Bluetooth specification), packet overhead, as well as some other requirements defined by the specification (for specific use cases such as responses and confirmation packets).

As an example, in the case of the 2M PHY, one can achieve a maximum application data rate of around 1.4 Mbps.

7.1.1. 2M PHY

At the application level, you do not need to know much about the low-level details of this PHY, other than setting it when you want to achieve higher speeds. But keep in mind that using this PHY potentially reduces the range.

Another restriction that was mentioned above is that the **2M PHY** is not allowed in primary advertisements. There are two ways to utilize this mode:

- Secondary advertisements (**extended advertising mode**) are used and sent on the 2M PHY, which allow a connection on that PHY from the central device.
- Advertising on the primary or secondary channels using the 1M or the coded PHY. A connection is then established, and either side can request a PHY update to use the 2M PHY during the connection.

One important thing to note is that the link between a peripheral and a central can be asymmetric, meaning that the packets from the peripheral can be sent using the 1M PHY, while packets from the central can be sent using the 2M PHY.

7.1.2. Coded PHY

As mentioned earlier, Bluetooth 5 achieves the longer range compared to earlier versions of Bluetooth by introducing the new coded PHY. So, what does **coding** mean? And how does it help achieve a longer range of communication?

It achieves this by utilizing a telecommunications technique called **Forward Error Correction (FEC)**. FEC allows the receiver to recover the data from errors that occur due to noise and interference. It accomplishes this by introducing redundancy in the data being transmitted, using a specific algorithm. So, instead of requiring retransmission of data when an error occurs, the receiver can recover the originally transmitted data by utilizing the redundancy in the data.

There are two coding schemes used by the coded PHY:

- **S = 2**, where 2 symbols represent 1 bit therefore supporting a bit rate of **500 kbps**. **S**

= 8, where 8 symbols represent 1 bit therefore supporting a bit rate of **125 kbps**.

7.2. Eight Times the Advertising Capacity

7.2.1. Extended Advertisements

Bluetooth 5 introduced the concept of **secondary advertising channels** which allow the device to offload data to advertise more data than what's allowed on the **primary advertisement channels**. Advertisements that are transmitted only on the primary advertisement channels are called **legacy advertisements**, whereas advertisements that start by transmission on the primary channels and then continue on the secondary channels are called **extended advertisements**.

In the case of extended advertisements, the advertisement packets sent on the primary advertisement channels provide the information necessary to discover the offloaded advertisements that are sent on the **secondary advertisement channels**. These are utilized for sending significantly more data (8x) than legacy advertisements allow (up to 255 bytes vs. 31 bytes). They are also useful in reducing congestion on the three primary advertising channels.

Advertisement packets sent on the secondary advertisement channels can use any of the three PHYs (1M PHY, 2M PHY, or coded PHY), whereas the primary advertisement channels can only use the coded PHY or the original 1M PHY. This means that a central must use the 1M PHY or coded PHY when initially searching for peripherals that are sending out advertising packets.

7.2.2. Periodic Advertisements

Think of the following use case: we have multiple temperature sensors distributed in a building. The temperature readings from these sensors change over time, and they need to be distributed along with other data (location, time of reading, etc.) to multiple devices that pass this data up to the cloud.

We could potentially use extended advertisements since we may have more data than would fit into the legacy advertisement packet (31 bytes), but that means the centrals will have to be looking for advertisements all the time, potentially consuming a lot of power (especially if the advertisement data does not change often).

Instead, we could utilize a new feature that was introduced in Bluetooth 5: **periodic advertisements**. Periodic advertisements are a special case of **extended advertisements** and allow a central to "synchronize" to a peripheral that is sending these extended advertisements at a fixed interval. This helps reduce power consumption when the advertisements are sent periodically at longer intervals, while allowing multiple centrals to be synchronized to the same peripheral.

The way periodic advertisements work is by transmitting advertising packets on the primary advertisement channels, which hold information (e.g., time offset, PHY, etc.) to help locate the extended advertisement packet. That packet, in turn, contains fields that define the data needed to synchronize to the periodic advertisement packets — similar to how connections are synchronized using a channel map, hop increment, the selected PHY, etc.

7.3. More on Extended Advertisements

Extended advertisements utilize the **Secondary Advertisement Channels**, which are the same channels used by data packets transmitted during a connection between two devices. Extended advertisements are not considered part of the advertisement events we talked about previously (also called **legacy advertisements**), which occur on the primary advertising channels (37, 38, and 39).

Extended advertisements are used to "offload" data that would otherwise exist on the Primary Advertising Channels — also called **auxiliary packets**. Offloading is accomplished by first advertising on the primary channel data values that point to an auxiliary packet on the secondary channel. The advertisement packets sent on the primary channels contain the PHY channel and the offset to the start time of the extended advertisement packet.

Another important aspect is that extended advertisements can use any of the three PHYs (1M PHY, 2M PHY, or the coded PHY), whereas primary advertisement packets can only be sent using the 1M PHY or the coded PHY.

Since non-**Bluetooth 5** devices are not able to discover extended advertisements, it is recommended that peripherals also use an advertising set (additional advertisements) with legacy advertising PDUs for older central devices to be able to discover these peripherals.

Here's a diagram showing an example of extended advertising:

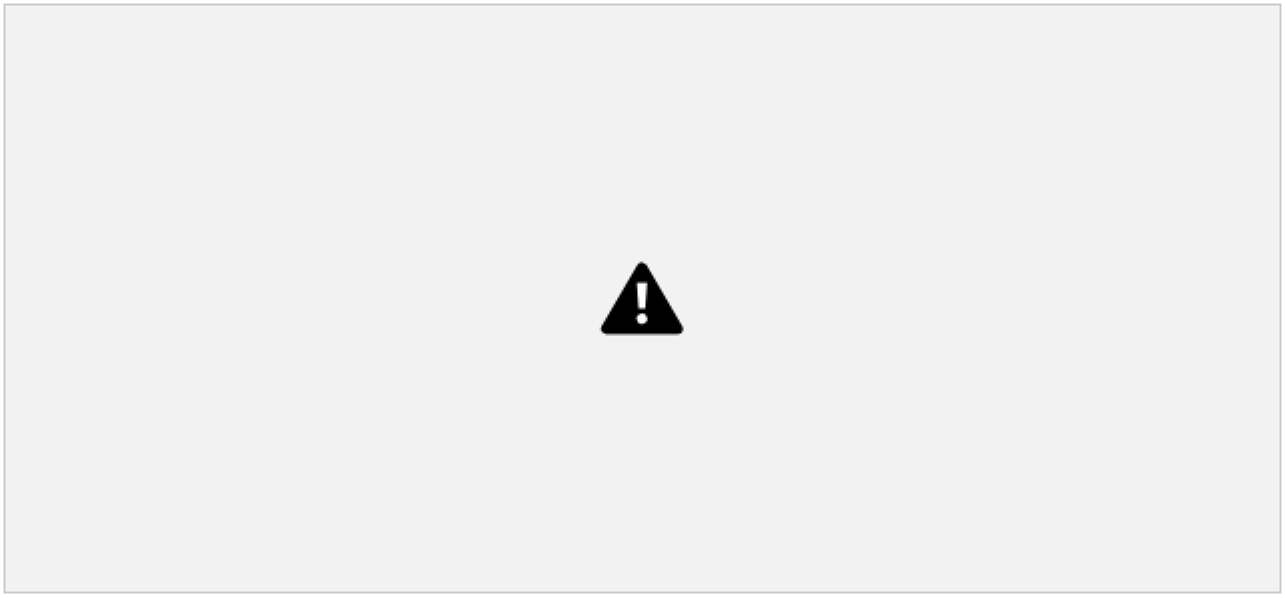


Figure 20*: Extended advertising*
(Source: Bluetooth 5 specification document)

The periodic advertising mode allows two or more devices to communicate in a connection less manner. The peripheral device sends out synchronization information along with the other extended advertisement data allowing another device to become **synchronized** with this peripheral. This synchronization allows devices to receive the peripheral device's extended advertisements at regular, deterministic intervals.

Here's a diagram showing an example of periodic advertising:

Figure 21*: Periodic advertising*
(Source: Bluetooth 5 specification document)

8. Security

Security has become one of the most voiced concerns about IoT systems. With all the headline news that mention hacks and vulnerabilities discovered in many IoT products, it has become one of the major concerns for manufacturers and developers of IoT devices.

In this chapter, we will:

- Go over the different security concerns.
- Take a look at the security measures that BLE provides.
- Cover Privacy concerns.

8.1. Security Concerns

Some of the most common security concerns with any system include:

- **Authentication**: Authentication is proof that the other side is who they claim they are. So if you’re connecting to a BLE device, you want to be sure that you are actually connecting to the device of interest — and not some other malicious device that’s pretending to be that device.
- **Integrity**: Integrity ensures us that the data received is free from corruption and tampering by unauthorized devices.
- **Confidentiality**: Confidentiality is concerned with making sure the data is not readable by unauthorized users or devices.
- **Privacy**: Privacy is concerned with how private the communication is, and whether a third party is able to track our device — especially by its Bluetooth address.

These are some general concerns related to security that apply to any system. The importance of each one of these concerns depends on the application and use case of the product.

8.1.1. Types of Attacks

Based on the above mentioned concerns, there are different types of attacks that a malicious device or person may implement. Some of these include:

- **Passive Eavesdropping**: This describes when a malicious device listens in on the communication between two devices, and is able to understand the data — usually by gaining access to the encryption key in the case the data is encrypted. **Active Eavesdropping**: This is also known as a **Man-In-The-Middle (MITM)** attack. In this attack, the malicious device impersonates both devices (the peripheral and the central). It could then intercept the communication between them, route it so they do not realize that the attack is happening, and possibly even injecting data into the packets.

Privacy and Identity Tracking: In this attack, devices and users are tracked by the Bluetooth address — possibly revealing their location and correlating it with their behavior.

8.2. Security in BLE

Security in BLE is handled by the **security manager (SM)** layer of the architecture. It is shown in the following diagram:

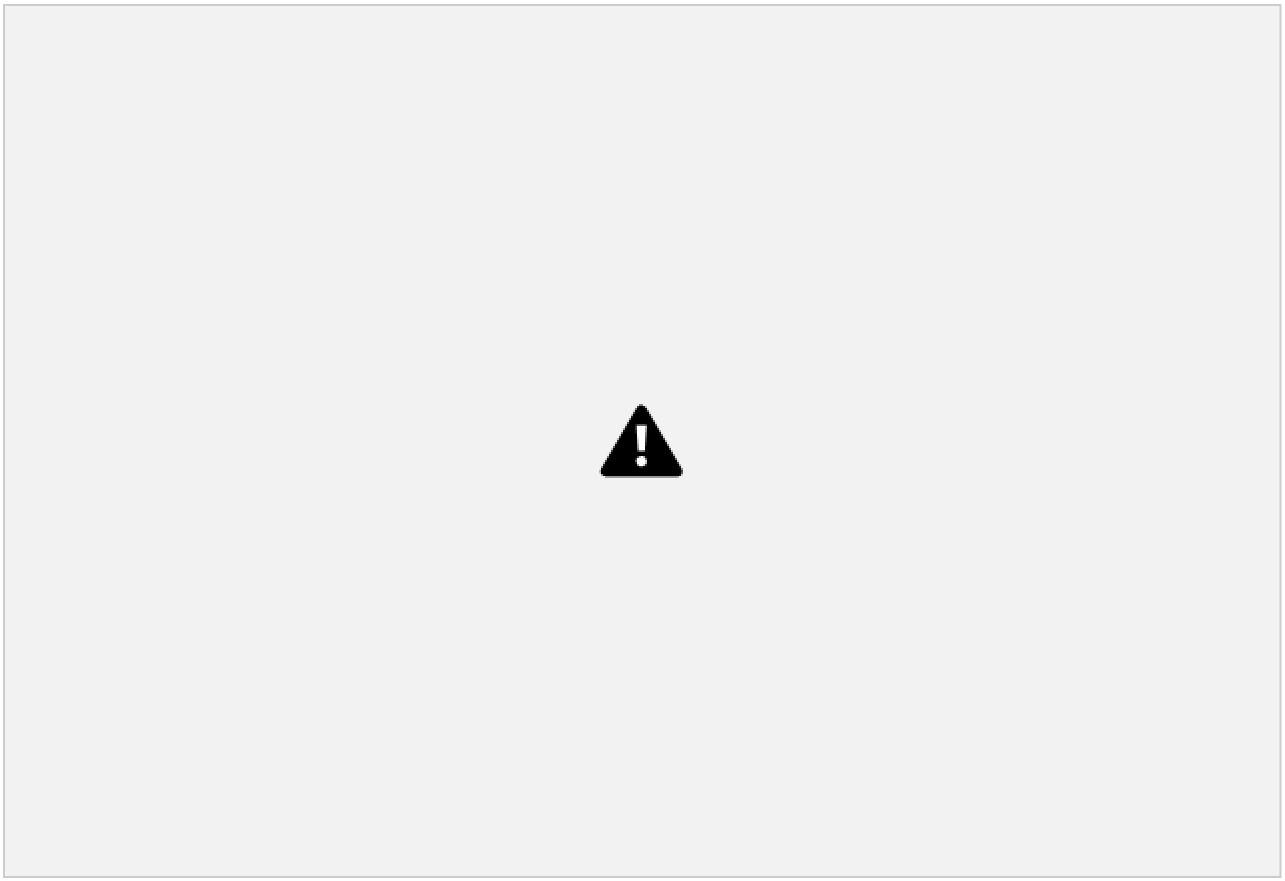


Figure 22*: Security manager (SM) in the architecture of BLE*

The security manager defines the protocols and algorithms for generating and exchanging keys between two devices. It involves five security features:

- **Pairing**: the process of creating shared **secret keys** between two devices. **Bonding**: the process of creating and storing shared **secret keys** on each side (central and peripheral) for use in subsequent connections between the devices. **Authentication**: the process of verifying that the two devices share the same secret keys.
- **Encryption**: the process of encrypting the data exchanged between the devices. Encryption in BLE uses the 128-bit AES Encryption standard, which is a **symmetric-key** algorithm (meaning that the same key is used to encrypt and decrypt the data on both sides).
- **Message Integrity**: the process of **signing** the data, and **verifying** the signature at the other end. This goes beyond the simple integrity check of a calculated CRC.

The Bluetooth specification has evolved over time to provide stronger security measures. This is especially true for BLE, which introduced the concept of **LE Secure Connections (LESC)** in version 4.2. LESC utilizes the Elliptic-curve Diffie-Hellman (ECDH) protocol during the pairing process (covered later in this chapter), which makes the communication much more secure compared to the methods used in earlier versions of Bluetooth.

Bluetooth 4.2 also introduced the term **legacy connections**, which collectively refers to the pairing methods defined by the earlier specification versions. It's important to note, though, that legacy connections are still supported in Bluetooth 4.2 and later. We'll cover the differences between these methods in the upcoming sections.

The Security Manager addresses the different security concerns as follows:

- **Confidentiality** via encryption.
- **Authentication** via pairing & bonding.
- **Privacy** via resolvable private addresses.
- **Integrity** via digital signatures.

In BLE, the master device is the **initiator** of security procedures. The slave (**responder**) may request the start of a security procedure by sending a security request message to the master, but it is up to the master to then send the packet that officially starts the security process.

To better understand how security works in BLE, we need to understand two important concepts: **pairing** and **bonding**. But first, let’s review a sequence diagram showing the security process:

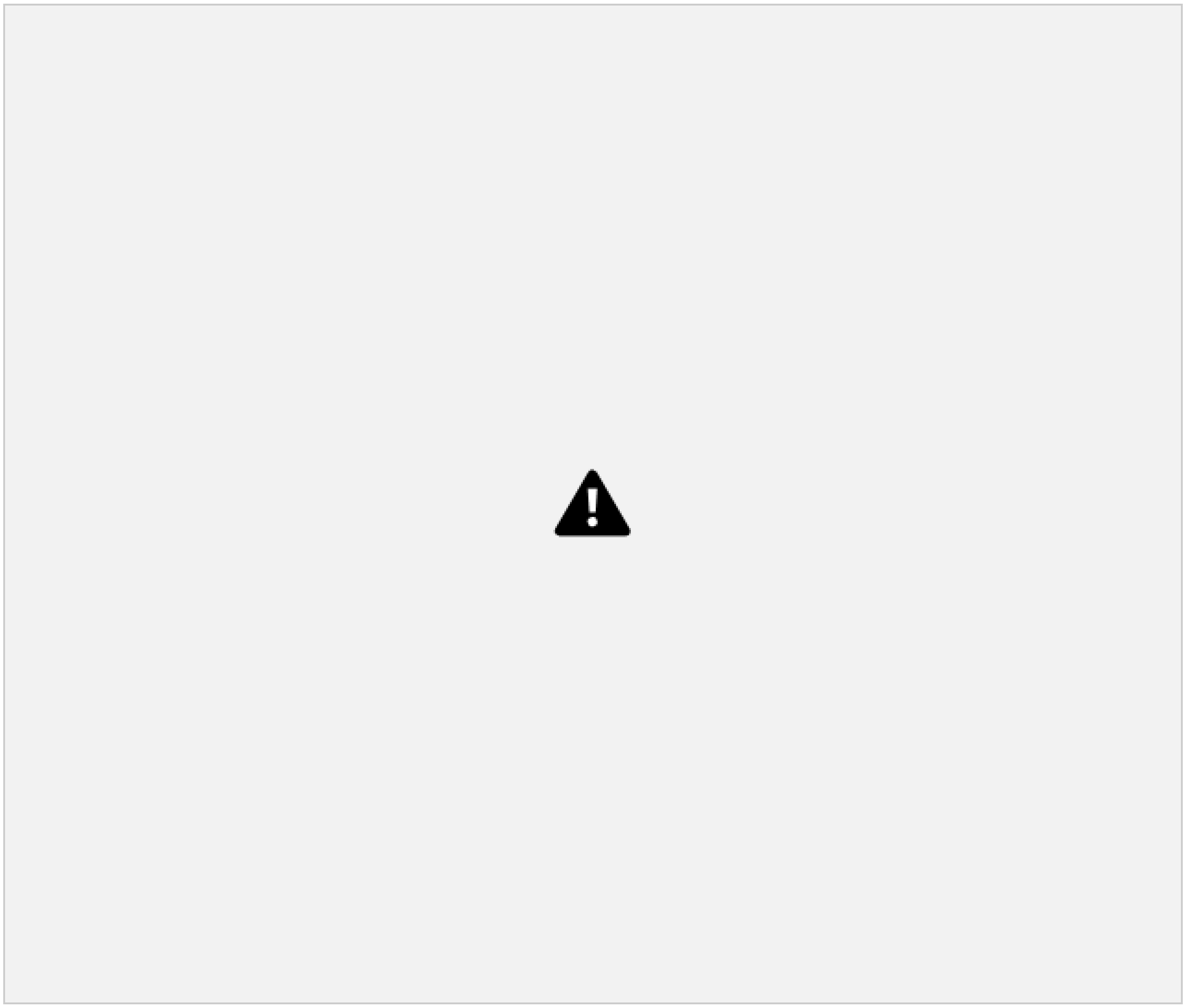


Figure 23*: The different phases of the security process in BLE*

Pairing is the combination of Phases 1 and 2. **Bonding** is represented by Phase 3 of the process. One important thing to note is that Phase 2 is the only phase that differs between LE Legacy Connections and LE Secure Connections.

8.2.1. Pairing and Bonding

Pairing is a temporary security measure that does not persist across connections. It has to be initiated and completed each time the two devices reconnect and would like to encrypt the connection between them. In order to extend the encryption across subsequent connections, bonding must occur between the two devices.
Let’s go over the different phases in more detail:

8.2.1.1. Phase One

In this phase, the slave may request the start of the pairing process. The master initiates the pairing process by sending a **pairing request** message to the slave, which then responds with a **pairing response** message.

The pairing request and pairing response messages represent an exchange of the features supported by each device, as well as the security requirements for each device. Each of these messages include the following:

- **Input Output (IO) capabilities**: display support, keyboard support, yes/no input support.
- **Out-Of-Band (OOB)** method support.
- **Authentication requirements**: includes MITM protection requirement, bonding requirement, secure connections support.
- **Maximum encryption key size** that the device supports.
- The different **security keys** each device is requesting to use.

The information exchanged between the two devices in this phase determines the pairing method used. Here’s a table showing the different combinations of the exchanged IO capabilities (on the two pairing devices) and the resulting pairing method chosen:

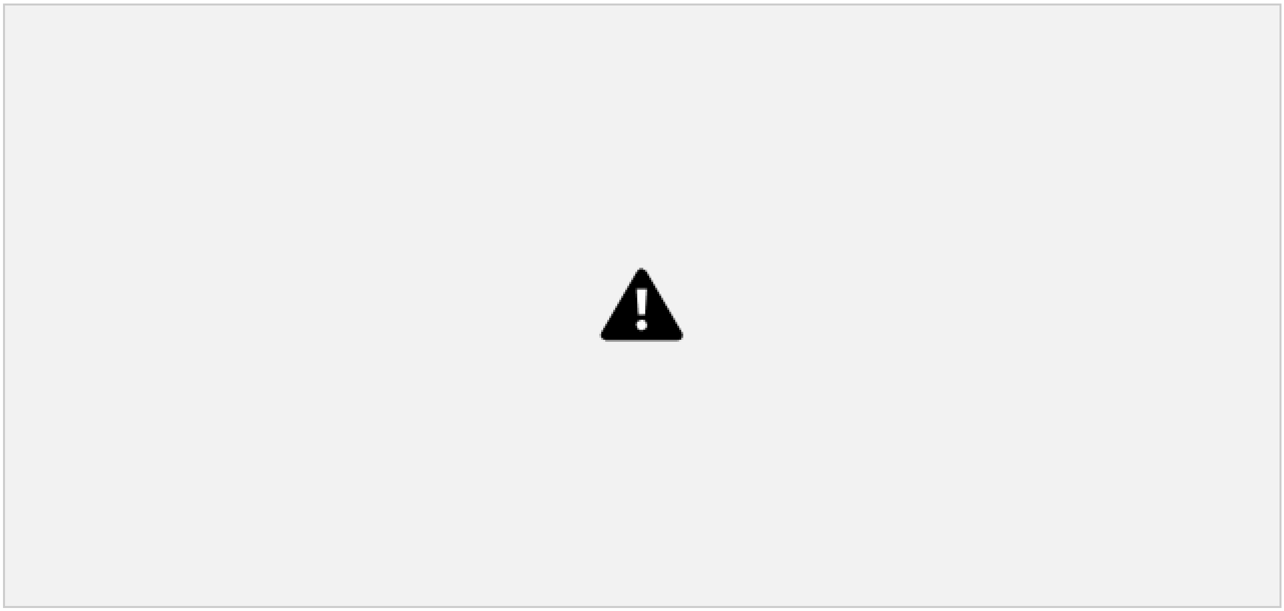


Figure 24: IO Capabilities Lookup Table
(Source: Bluetooth 5 specification document)

8.2.1.2. Phase Two

As mentioned previously, **phase two** differs based on which method is used: **LE secure connections** or **LE legacy connections**.

Let's explain how this phase differs between the two methods:

> **Legacy Connections**:
> In **legacy connections**, there are two keys used: the **temporary key (TK)** and the **short term key (STK)**. The TK is used along with other values exchanged between the two devices to generate the STK.
>
> **Secure Connections**:
> In **secure connections**, the pairing method does not involve exchanging keys over the air between the two devices. Rather, the devices utilize the ECDH protocol to each generate a **public/private key** pair. The devices then exchange the public keys only, and from that generate a shared secret key called the **long term key (LTK)**.
>
> The advantage of using ECDH is that it prevents eavesdroppers from figuring out the shared secret key — even if they capture both public keys. To learn more about ECDH

and how it works, refer to its Wikipedia page here. I've also found that this video explains it very well.

8.2.1.3. Phase Three

Phase three represents the **bonding** process. This is an optional phase that's utilized to avoid the need to re-pair on every connection to enable a secure communication channel.

The result of bonding is that each device stores a set of keys that can be used in each subsequent connection and allows the devices to skip the pairing phase. These keys are exchanged between the two devices over a link that's encrypted using the keys resulting from phase two.

8.2.2. Pairing methods

Legacy Connections and Secure Connections each have different Pairing Methods. Some of the Methods share the same name, but the process and the data exchanged differs among them. The Pairing Method that gets used is determined based on the features exchanged between the two devices in Phase One.

8.2.2.1. LE Legacy Connections (All Bluetooth versions)

As we mentioned earlier, in **legacy connections** a short term key (STK) is generated from the temporary key (TK) and two randomly generated values.

Just Works:
In this method, TK is set to **0**. For obvious reasons, this method is the least secure of all methods (amongst all Bluetooth versions).

Out of Band (OOB):
In this method, the TK is exchanged between the two devices over a technology other than BLE — **near field communication (NFC)** being the main one. This method can make the pairing process much more secure, especially if the non-BLE technology used provides stronger security. This is the most secure method of the legacy pairing methods.

Passkey:
In this method, the TK is a six-digit number that is transferred between the devices by the end-user. For example, it may be entered manually into one of the devices. The