Chapter 4:   Bluetooth Low Energy (LE) Basics

After completing this chapter, you will understand various Bluetooth LE concepts, such as GAP, GATT, Link Layer, and so forth. You will also understand the basic use of the Bluetooth LE Stack and firmware architecture.

Table of Contents

Document conventions

Convention	Usage	Example
Courier New	Displays code	CY_ISR_PROTO(MyISR);
Italics	Displays file names and paths	*sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures	[**Enter**] or [**Ctrl**] [**C**]
Menu > Selection	Represents menu paths	**File > New Project > Clone**
Bold	Displays commands, menu paths and selections, and icon names in procedures	Click the **Debugger** icon, and then click **Next**.

4.1 Introduction

Whether you are using stand-alone Bluetooth solutions like the CYW20719 and CYW20819 or Hosted Bluetooth LE solutions such as a PSoC 6 with a CYW4343W or CYW43012, the basic Bluetooth architecture and the API are the same.

Much of the material in this chapter and the other Bluetooth related chapters in this course is taken from the WICED Bluetooth 101 course. That is a 3-day course that covers much more ground (both in terms of topics and depth of coverage) than could fit in the time available for this single chapter. If you want more information about Bluetooth, I highly recommend that you attend that class or review it on your own. The class material can be found online at:

https://github.com/cypresssemiconductorco/CypressAcademy_WBT101_Files

Some of the topics from the full course that are not included here are:

- Advanced Security Topics such as Passkey and Numeric Comparison

- Beacons

- Bluetooth LE Over-The-Air Firmware Update

- Bluetooth LE Centrals

- Classic Bluetooth

- Mesh

4.2 Overview

With the addition of Bluetooth Low Energy to the Bluetooth specification in 2010, it has become very popular in IoT devices such as smart watches, health monitors, beacons, etc. These applications typically have in common small batteries that are often not charged frequently. Therefore, low power is more critical than data transfer speed. Moreover, these types of devices don't require a constant connection. Rather, they can connect somewhat infrequently to send a burst of data.
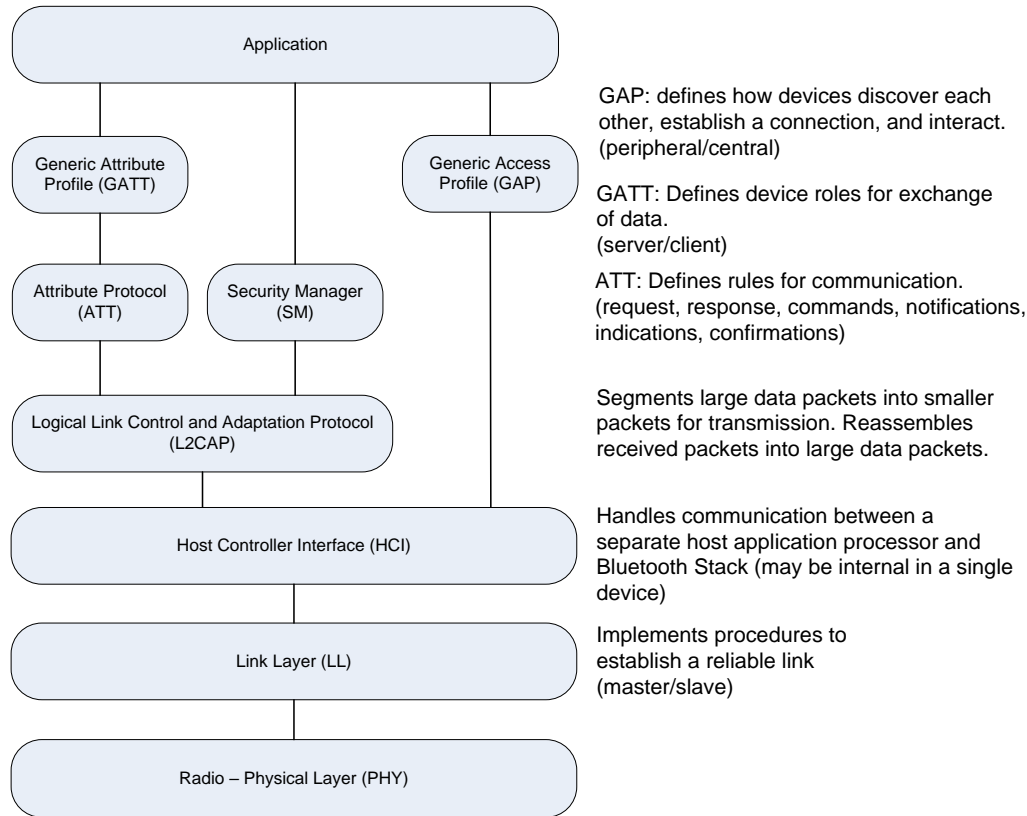
The scenario described above is ideal for Bluetooth LE. In fact, the way low power is achieved in Bluetooth LE is not by lowering the power of the radio (that is, the range), but rather by having the radio turned off most of the time. That is, Bluetooth LE connections can stay active while only turning on the radio for a small percentage of each connection interval (for example, a few hundred microseconds). The connection interval can be varied depending on the application from 7.5 milliseconds to 4 seconds to trade off power and performance.

The MCU can also be put into sleep modes a large portion of the time to further reduce power.

Bluetooth LE is also sometimes referred to as "Bluetooth Smart". The two terms are interchangeable. Devices that support both Classic Bluetooth and Bluetooth LE (for example, smartphones) are sometimes called "Smart Ready".

4.3 Bluetooth LE Stack

As with most complex systems, the Bluetooth LE Stack (from now on referred to as "the Stack") is broken into layers as shown in this diagram. The following sections describe the layers in more detail.

GAP: defines how devices discover each other, establish a connection, and interact. (peripheral/central)

GATT: Defines device roles for exchange of data. (server/client)

ATT: Defines rules for communication. (request, response, commands, notifications, indications, confirmations)

Segments large data packets into smaller packets for transmission. Reassembles received packets into large data packets.

Handles communication between a separate host application processor and Bluetooth Stack (may be internal in a single device)

Implements procedures to establish a reliable link (master/slave)

For the solutions we are using in this class, the Bluetooth device is running in hosted mode. That means that the 4343W or 43012 runs the lower part of the Stack (PHY and LL), while the PSoC 6 runs the upper part of the Stack. The two devices communicate using the Host Controller Interface (HCI), which runs over a UART interface.

4.3.1 Physical Layer (PHY)

Bluetooth LE operates in the 2.4 GHz ISM band (2.400 – 2.480 GHz) using 40 channels with 2 MHz spacing between channels. 3 channels are used for advertising (that is, establishing a connection) and 37 channels are used for data. Gaussian Frequency Shift Keying (GFSK) modulation is used.

Note that this is the same overall band as Bluetooth Classic and Wi-Fi (802.11)! In order to work in the crowded 2.4 GHz ISM band, the 3 advertising channels (37, 38, and 39) are spread across the spectrum. For example, a region with 3 Wi-Fi access points operating on 3 different channels may look like this when superimposed on the Bluetooth LE channels:
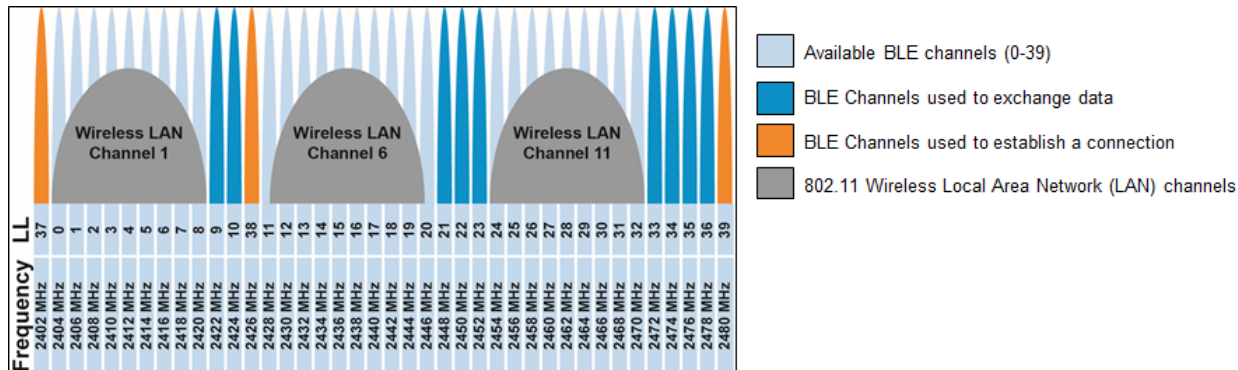
Bluetooth LE uses adaptive frequency hopping (AFH) to avoid channels with poor signal strength or high error rates. In the example above, channels 0-8, 11-20, and 24-32 will likely be identified as channels that should be excluded from frequency hopping due to the interference from the Wi-Fi signals.

The maximum raw data transfer rate in Bluetooth LE is 1 Mbps. In Bluetooth v5, the data rate can be doubled to 2 Mbps at the expense of range. Including overhead, the actual data transfer rate is ~300 Kbps in Bluetooth v4.1 and is ~800 Kbps in Bluetooth v4.2 and beyond, due to the data length extension which allows larger payloads in each packet (27 bytes vs. 251 bytes). The max payload size can be different between transmit and receive to optimize application throughput.

4.3.2 Link Layer (LL)

The link layer provides the methods for devices to find each other and connect. It also handles maintaining a reliable link once it has been established.

A device that is available will *Advertise* so that it can be discovered by nearby devices. The advertisement packet includes device information such as services supported and what type of connections, if any, the device will allow.

Devices that want to gather information or form connections will *Scan* for nearby devices that are advertising. Scan packets are sent out at irregular intervals to increase the chances of coinciding with a listening device's window. Once devices know about each other, the one that initiates the connection (that is, the one that was scanning for devices) will be the *LL Master*, while the one that accepts the connection will be the *LL Slave*.

Once a connection is established, the link layer uses AES-128 encryption and 24-bit cyclic redundancy check (CRC) to guarantee a private and reliable connection. The link layer also implements AFH as described previously.

4.3.3   Logical Link Control Adaptation Protocol (L2CAP)

The L2CAP layer is responsible for taking large packets of data from the upper layers and segmenting them into smaller packets for the link layer, and vice versa. The largest possible size for data packets being transmitted in Bluetooth LE is called the Maximum Transmission Unit (MTU). It can be set in the range of 23 to 512 bytes.

4.3.4   Generic Access Profile (GAP)

The GAP defines how devices discover each other, how they establish a connection, and how they interact with each other based on their roles. There are 4 GAP roles. The first 2 involve a connection, while the last two involve an exchange of data without a connection (that is, advertise/scan only). They are:

| GAP Role	Description
Peripheral	A device that connects to a Central. Typically, this is an IoT device like a fitness monitor.
Central	A device that connects to a Peripheral. Most often, this is a smart phone or tablet.
Broadcaster	A device that only advertises. It may transmit useful data within the advertising packets. This may be an IoT device such as a beacon or a GPS tag.
Observer	A device that scans for devices and may use data from their advertising packets.

4.3.5   Generic Attribute Profile (GATT)

Once a connection is made, the GATT determines how data is exchanged. There are 2 GATT roles:

GATT Role	Description
Server	A device that contains data to be shared. Typically, this is an IoT device like a fitness monitor.
Client	A device that requests data from the server. Most often, this is a smart phone or tablet.

Note: *Based on the above roles, the GATT server is typically a GAP peripheral while the GATT client is typically a GAP central.*

Servers use a GATT Database to store data in a format defined by the Bluetooth spec. The database responds to read/write requests from server itself (for example, when new data is available from a sensor) and from a connected client. Allowed transactions are defined when the database is set up in the server (for example, which values the client can write/read vs. read).

As an example, a client may be allowed to write/read configuration settings on the server but may only be allowed to read the values of sensors.
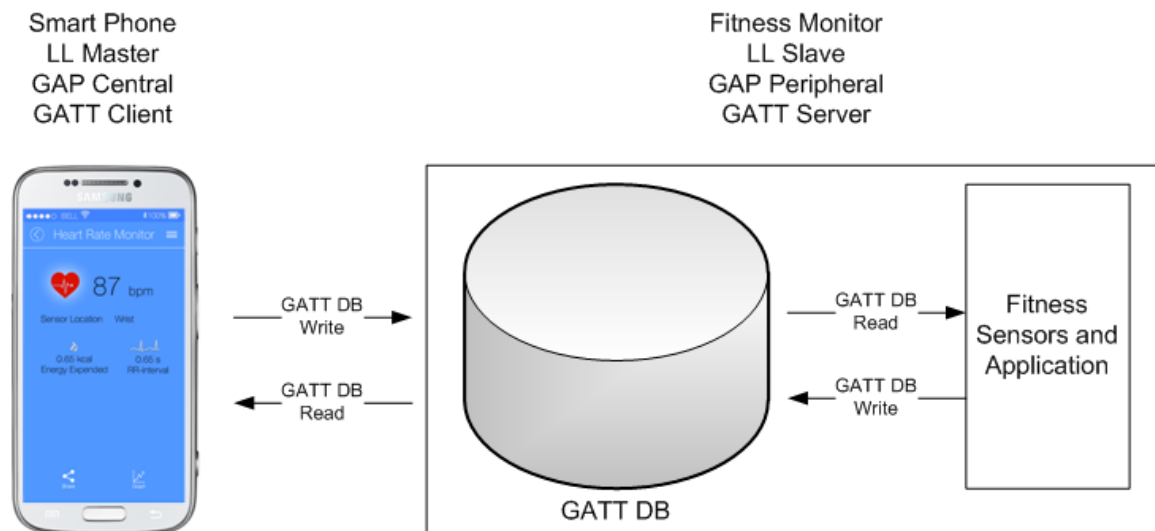
4.3.6 Attributes - Profiles, Services, and Characteristics

The GATT Database is a just a table with up to 65535 rows. Each row in the table represents one Attribute and contains a Handle, a Type, a Value and Permissions.
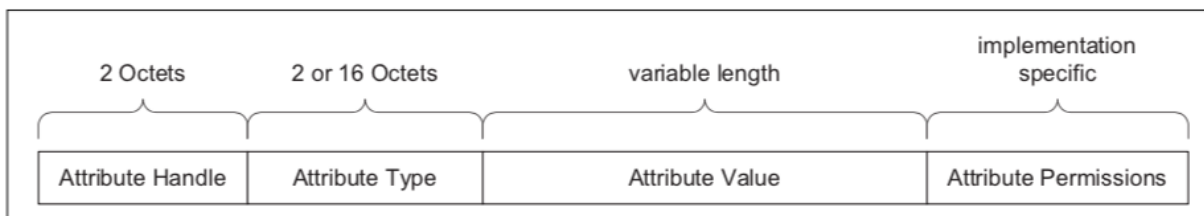
(This figure and others in this section are taken from the Bluetooth Core Specification)

The **Handle** is a 16-bit unique number to represent that row in the database. These numbers are assigned by you, the firmware developer, and have no meaning outside of your application. You can think of the Handle as the database primary key.

The **Type** of each row in the database is identified with a Universally Unique IDentifier (UUID). The UUID scheme has two interesting features:

- Attribute UUIDs are 2 octets or 16 octets long. You can purchase a 2-octet UUID from the SIG for around $5K
- Some UUIDs are defined by the Bluetooth SIG and have specific meanings and some can be defined by your application firmware to have a custom meaning

The Bluetooth spec frequently refers to UUIDs by a name surrounded by « ». To figure out the actual hex value for that name, you need to look at the assigned numbers table on the Bluetooth SIG website. Also, most

of the common UUIDs are inserted for you into the right place by the Bluetooth configurator (more on this later).

The **Permissions** for Attributes tell the Stack what it can and cannot do in response to requests from the Central/Client. The Permissions are just a bit field specifying Read, Write, Encryption, Authentication, and Authorization. The Central/Client can't read the permission directly, meaning if there is a permission problem the Peripheral/Server just responds with a rejection message. The Bluetooth configurator helps you get the permission set correctly when you make the database, and the Stack takes care of enforcing the Permissions.

4.3.6.1 Profiles – Services – Characteristics

The GATT Database is "flat" – it's just a bunch rows with one Attribute per row. This creates a problem because a totally flat organization is painful to use. So, the Bluetooth SIG created a semantic hierarchy. The hierarchy has two levels: Services and Characteristics. Note that Services and Characteristics are just different types of Attributes.

In addition to Services and Characteristics, there are also Profiles which are a previously agreed to, or Bluetooth SIG spec'd related, set of data and functions that a device can perform. If two devices implement the same Profile, they are guaranteed to interoperate. A Profile contains one or more Services.

A Service is just a group of logically related Characteristics, and a Characteristic is just a value (represented as an Attribute) with zero, one or more additional Attributes to hold meta data (e.g. units). These meta-data Attributes are typically called Characteristic Descriptors.

For instance, a Battery Service could have one Characteristic - the battery level (0-100 %) - or you might make a more complicated Service, for instance a CapSense Service with a bunch of CapSense widgets represented as Characteristics.

There are two Services that are required for every Bluetooth LE device. These are the Generic Attribute Service and the Generic Access Service. Other Services will also be included depending on what the device does.

Each of the different Attribute Types (i.e. Service, Characteristic, etc.) uses the Attribute Value field to mean different things.

4.3.6.2 Service Declaration in the GATT DB

To declare a Service, you need to put one Attribute in the GATT Database. That row just has a Handle, a Type of 0x2800 (which means this GATT Attribute is a declaration of a Service), the Attribute Value which in this case is just the UUID of the Service, and the Attribute Permission.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»	16-bit Bluetooth UUID or 128-bit UUID for Service	Read Only, No Authentication, No Authorization

GATT Row for a Service Declaration

For the Bluetooth defined Services, you are obligated to implement the required Characteristics that go with that Service. You are also allowed implement custom Services that can contain whatever Characteristics you want. The Characteristics that belong to a Service must be in the GATT database after the declaration for the Service to which they belong and before the next Service declaration.

You can also include all the Characteristics from another Service by declaring an Include Service.

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	0x2802 – UUID for «Include»	Included Service Attribute Handle	End Group Handle	Service UUID	Read Only, No Authentication, No Authorization

GATT Row for an Included Declaration

4.3.6.3   Characteristic Declaration in the GATT DB

To declare a Characteristic, you are required to create a minimum two Attributes: Characteristic Declaration (0x2803) and Characteristic Value.

The **Characteristic Declaration** creates the property in the GATT database, sets up the UUID and configures the Properties for the Characteristic (which controls permissions for the characteristic as you will see in a minute). This Attribute does not contain the actual value of the characteristic, just the handle of the Attribute (called the Characteristic Value Attribute) that holds the value.

Attribute Handle	Attribute Types	Attribute Value			Attribute Permissions
0xNNNN	0x2803–UUID for «Characteristic»	Charac-teristic Properties	Character-istic Value Attribute Handle	Character-istic UUID	Read Only, No Authentication, No Authorization

GATT Row for a Characteristic Declaration

Each Characteristic has a set of Properties that define what the Central/Client can do with the Characteristic. These Characteristic Properties are used by the Stack to enforce access to Characteristic by the Client (for example, Read/Write) and they can be read by the Client to know what they can do. The Properties include:

- Broadcast – The Characteristic may be in an Advertising broadcast
- Read – The Client/Central can read the Characteristic
- Write Without Response – The Client/Central can write to the Characteristic (and that transaction does not require a response by the Server/Peripheral)
- Write – The Client/Central can write to the Characteristic and it requires a response from the Peripheral/Server
- Notify – The Client can request Notifications from the Server of Characteristic values changes with no response required by the Client/Central. The Stack sends notifications from the GATT server when a database characteristic changes.
- Indicate – The Client can ask for Indications from the Server of Characteristic value changes and requires a response by the Client/Central. The Stack sends indications from the GATT server when a database characteristic changes and waits for the client to send the response.
- Authenticated Signed Writes – The client can perform digitally signed writes
- Extended Properties – Indicates the existence of more Properties (mostly unused)

When you configure the Characteristic Properties, you must ensure that they are consistent with the Attribute Permissions of the characteristic value.

The **Characteristic Value** Attribute holds the value of the Characteristic in addition to the UUID. It is typically the next row in the database after the Characteristic Declaration Attribute.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0xuuuu – 16-bit Bluetooth UUID or 128-bit UUID for Characteristic UUID	Characteristic Value	Higher layer profile or implementation specific

GATT Row for a Characteristic Value Declaration

4.3.7 Security Manager (SM), Pairing and Bonding

Bluetooth LE has two security modes, and several levels in each mode. They are:

Security	Level 1	Level 2	Level 3
Mode 1	No security	Unauthenticated Encrypted	Authenticated Encrypted
Mode 2	Unauthenticated Data Signed	Authenticated Data Signed	N/A

Authentication is the process of identifying a device and deciding whether a connection will be allowed. It can be done in one of several ways depending on the capabilities of the devices. The possible capabilities are:

- No Input, No Output
- Display Only
- Display
- Display: Yes/No
- Keyboard Only

Once two Bluetooth LE devices have established a connection (including authentication and key exchange if necessary), they are considered Paired. If the authentication information and keys are stored in memory by both devices, then the devices are Bonded. Devices that are bonded can connect in the future without going through the pairing process again.

The whole process looks like this:

BLE Pairing And Bonding Procedure
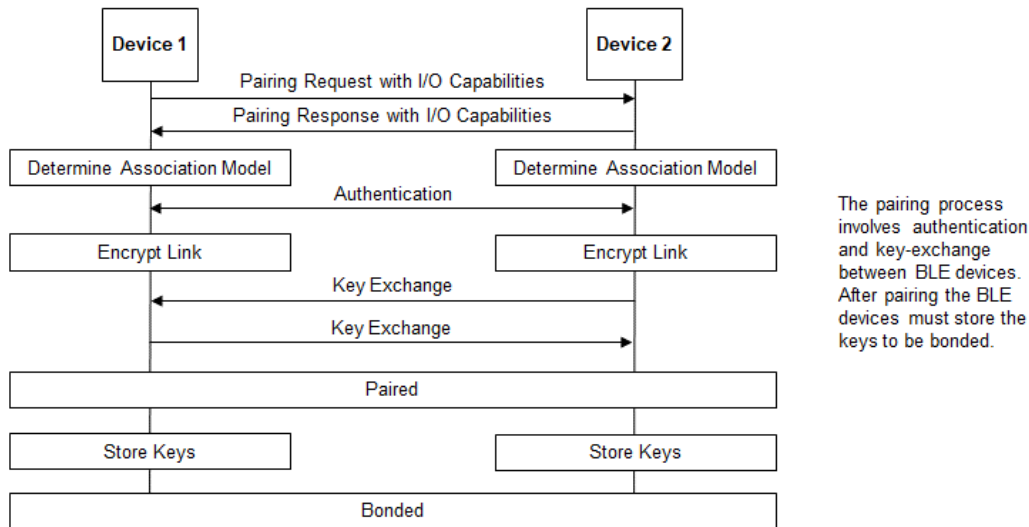
In Bluetooth v4.2, privacy 1.2 was introduced. This involves using a 48-bit resolvable private address (RPA) that can be changed frequently (every 1 second) to prevent tracking. Only peer devices that have the 128-bit identity resolving key (IRK) of a Bluetooth LE device can connect to it.
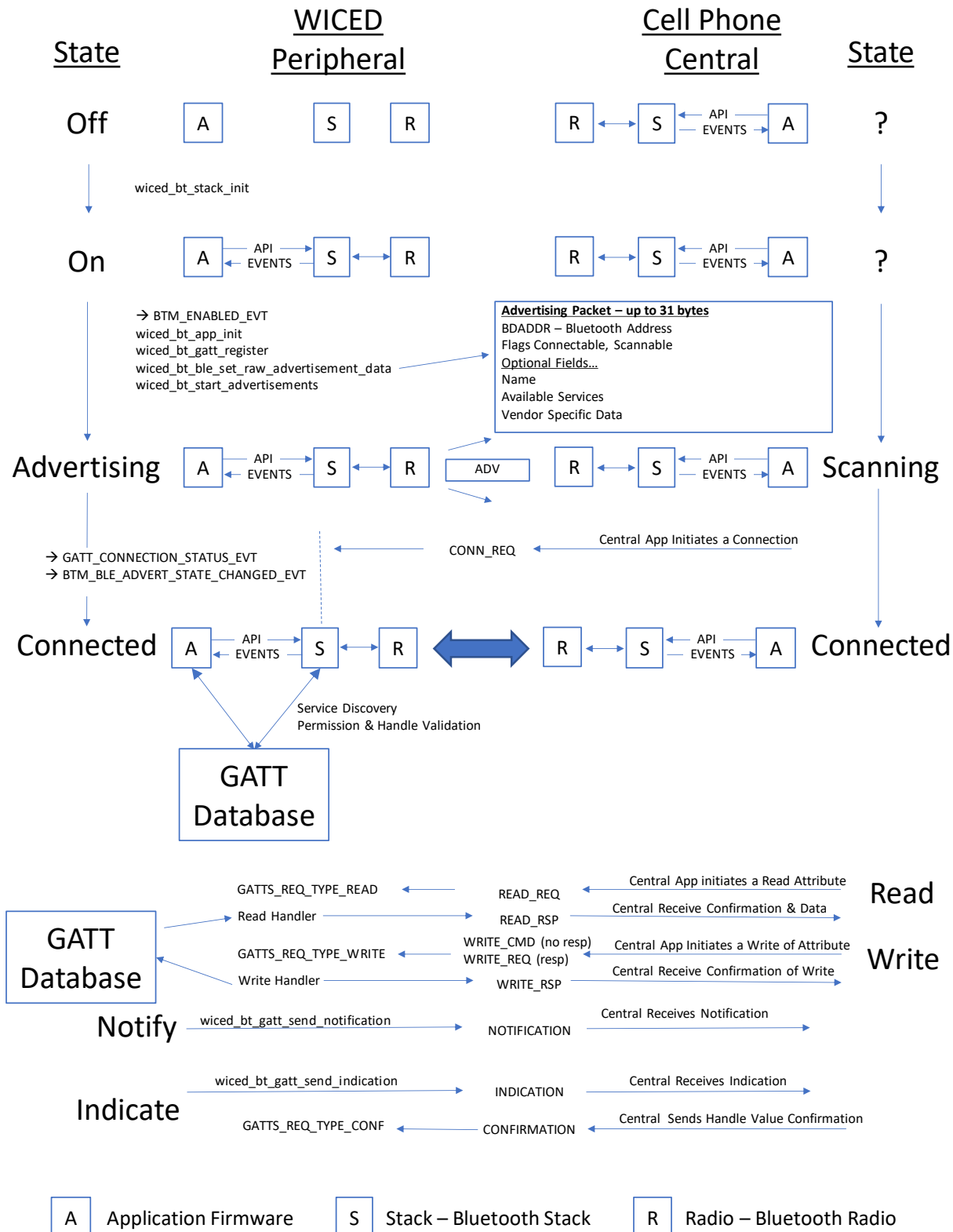
4.4   Bluetooth LE System Lifecycle

All Bluetooth wireless systems work the same basic way. You write Application (A) Firmware which calls Bluetooth APIs in the Stack (S). The Stack then talks to the Radio (R) hardware which in turn, sends and receives data. When something happens in the Radio, the Stack will also initiate actions in your Application firmware by creating Events (for example, when it receives a message from the other side). Your Application is responsible for processing these events and doing the right thing. This basic architecture is also true of Apps running on a cell phone (in iOS or Android), but we will not explore that in more detail in this course other than to run existing Apps on those devices.

There are 4 steps your application firmware needs to handle:

1. Turn on the Stack
2. Start Advertising as connectable
3. Process connection events from the Stack
4. Process read/write events from the Stack

Here is the overall picture which I will describe in pieces as we go:

| | WICED Peripheral | | | Cell Phone Central | | |

State | **WICED Peripheral** | **Cell Phone Central** | **State**

Off — A   S   R | R ↔ S ← API/EVENTS ← A   —   ?

wiced_bt_stack_init

On — A ← API/EVENTS → S ↔ R | R ↔ S ← API/EVENTS ← A   —   ?

→ BTM_ENABLED_EVT
wiced_bt_app_init
wiced_bt_gatt_register
wiced_bt_ble_set_raw_advertisement_data
wiced_bt_start_advertisements

Advertising Packet – up to 31 bytes
BDADDR – Bluetooth Address
Flags Connectable, Scannable
Optional Fields…
Name
Available Services
Vendor Specific Data

Advertising — A ← API/EVENTS → S ↔ R   ADV   R ↔ S ← API/EVENTS ← A — Scanning

→ GATT_CONNECTION_STATUS_EVT
→ BTM_BLE_ADVERT_STATE_CHANGED_EVT

CONN_REQ ← Central App Initiates a Connection

Connected — A ← API/EVENTS → S ↔ R ⟷ R ↔ S ← API/EVENTS ← A — Connected

Service Discovery
Permission & Handle Validation

GATT Database

GATT Database

GATTS_REQ_TYPE_READ ← READ_REQ ← Central App initiates a Read Attribute — Read
Read Handler → READ_RSP ← Central Receive Confirmation & Data

GATTS_REQ_TYPE_WRITE ← WRITE_CMD (no resp) / WRITE_REQ (resp) ← Central App Initiates a Write of Attribute — Write
Write Handler → WRITE_RSP ← Central Receive Confirmation of Write

Notify — wiced_bt_gatt_send_notification → NOTIFICATION → Central Receives Notification

Indicate — wiced_bt_gatt_send_indication → INDICATION → Central Receives Indication
GATTS_REQ_TYPE_CONF ← CONFIRMATION ← Central Sends Handle Value Confirmation

A Application Firmware   S Stack – Bluetooth Stack   R Radio – Bluetooth Radio

4.4.1　　　Turning on the Stack

In the beginning, you have a Bluetooth SoC device and a cell phone, and they are not connected. The Stack state is Off, so that's where we will start.

Like all great partnerships, every Bluetooth LE connection has two sides, one side called the **Peripheral** and one side called the **Central**. In the following picture, you can see that the Peripheral starts Off, there is no connection from the Peripheral to the Central (which is in an unknown state). In fact, at this point the Central doesn't know anything about the Peripheral and vice versa.
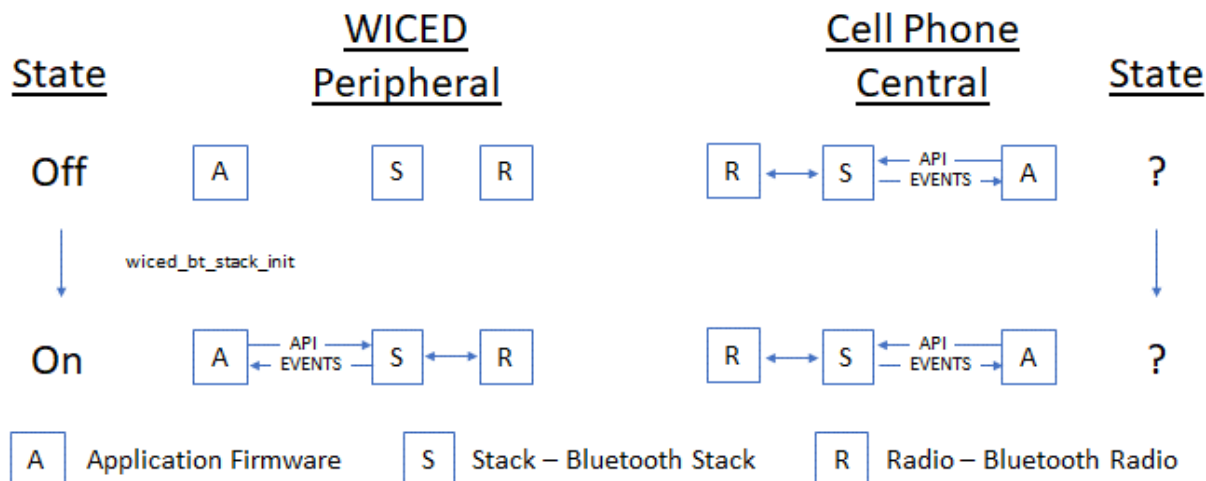
From a practical standpoint, the Peripheral should be the device that requires the lowest power – often it will be a small battery powered device like a beacon, a watch, etc. The reason is that the Central needs to Scan for devices (which is power consuming) while the Peripheral only needs to Advertise for short periods of time. Note that the GATT database is often associated with the Peripheral, but that is not required and sometimes it is the other way around.

The first thing you do in your firmware is to turn on Bluetooth LE. That means that you initialize the Stack and provide it with a function that will be called when the Stack has events for you to process (this is often called the "callback" function for obvious reasons).

4.4.2 Start Advertising

For a Central to know of your existence, you need to send out Advertising packets. The Advertising Packet will contain your Bluetooth Device Address (BDA), some flags that include information about your connection availability status, and one or more optional fields for other information, like your device name or what Services you provide (for example, Heart Rate, Temperature, etc.).

There are 4 primary types of Bluetooth Advertising Packets:

- `BTM_BLE_EVT_CONNECTABLE_ADVERTISEMENT`
- `BTM_BLE_EVT_CONNECTABLE_DIRECTED_ADVERTISEMENT`
- `BTM_BLE_EVT_SCANNABLE_ADVERTISEMENT`
- `BTM_BLE_EVT_NON_CONNECTABLE_ADVERTISEMENT`

When a Scannable Advertising Packet is scanned, the peripheral sends a Scan Response Packet (`BTM_BLE_EVT_SCAN_RSP`), which contains another 31 bytes of information.

The Stack is responsible for broadcasting your advertising packets at a configurable interval into the open air. That means that all Bluetooth LE Centrals that are scanning and in range may hear your advertising packet and process it. Obviously, this is not a secure way of exchanging information, so be careful what you put in the advertising packet. I will discuss ways of improving security later.
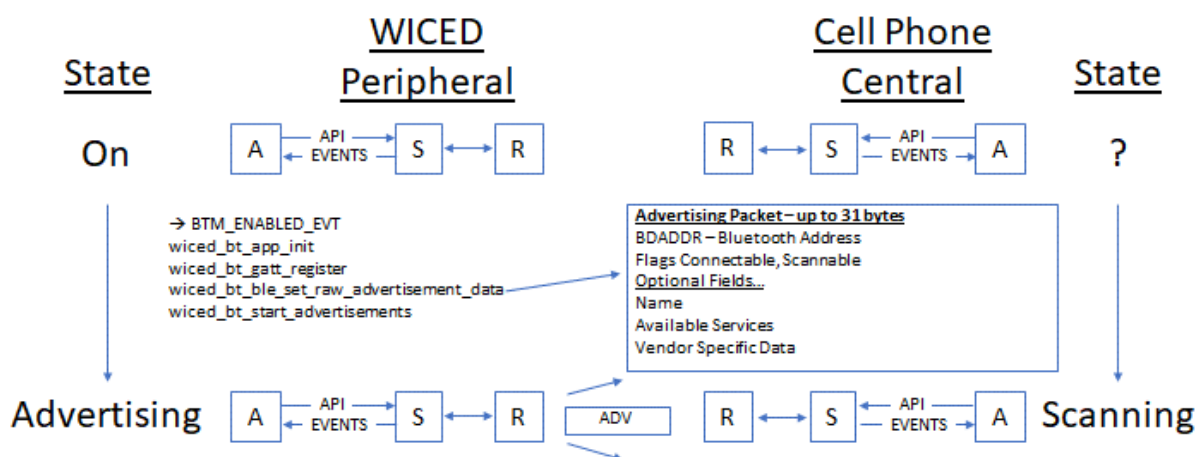
The first item in the advertising packet is called Flags. It tells the remote device how to make a connection by identifying the type of Bluetooth supported (Bluetooth LE, Classic, BR/EDR) and the way connections are allowed. The packet can also carry extra information, such as the device name, address, role and so on, but it has a maximum size of 31 bytes.

The format of the packet is quite simple. Each item you wish to advertise starts with a length byte, followed by the type (e.g. Flags or Name) and then the data, the size of which is determined by that length byte. The items are simple concatenated together, up to 31 bytes.

4.4.3 Make a Connection

Once a Central device processes your advertising packet, it can choose what to do next such as initiating a connection. When the Central App initiates a connection, it will call an API that will trigger its Stack to generate a Bluetooth Packet called a "conn_req". This will then go out the Central's radio and through the air to your radio.

The radio will feed the packet to the Stack and it will automatically stop advertising. You do not have to write code to respond to the connection request, but the Stack will generate two callbacks to your firmware (more on that later).

You are now connected and can start exchanging messages with the central.
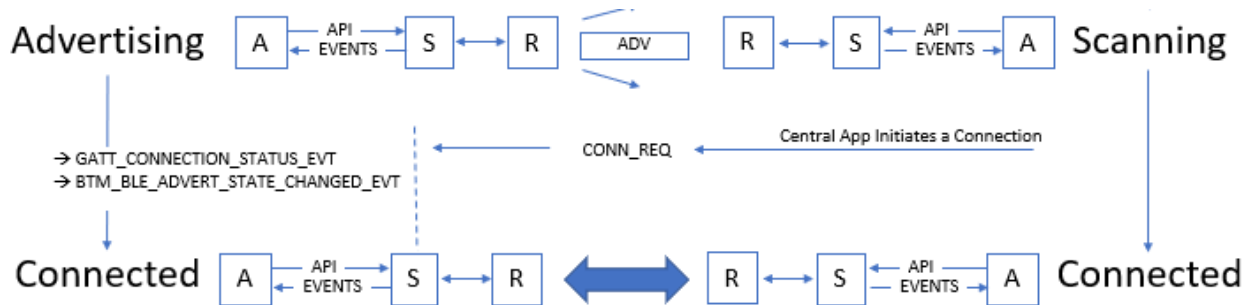
4.4.4   Exchange Data

Now that you are connected, you need to be able to exchange data. In the world of Bluetooth LE this happens via the Attribute Protocol (ATT). The basic ATT protocol has 4 types of transactions:

- Read & Write which are initiated by the Client
- Notify & Indicate which are initiated by the Server.

ATT Protocol transactions are all keyed to a very simple database called the GATT database which typically (but not always) resides on the Peripheral. The side that maintains the GATT Database is commonly known as the GATT Server or just Server. Likewise, the side that makes requests of the database is commonly known as the GATT Client or just Client. The client is typically (but not always) the Central. This leads to the obvious confusion that the Peripheral is the Server and the Central is the Client, so be careful.

You can think of the GATT Database as a simple table. The columns in the table are:

- Handle - 16-bit numeric primary key for the row
- Type - A Bluetooth SIG specified number (called a UUID) that describes the Data
- Data - An array of 1-x bytes
- Permission Flags

See more detail about the GATT database in section 4.3.5 Generic Attribute Profile (GATT). With all of that, here is the final section of the big picture.
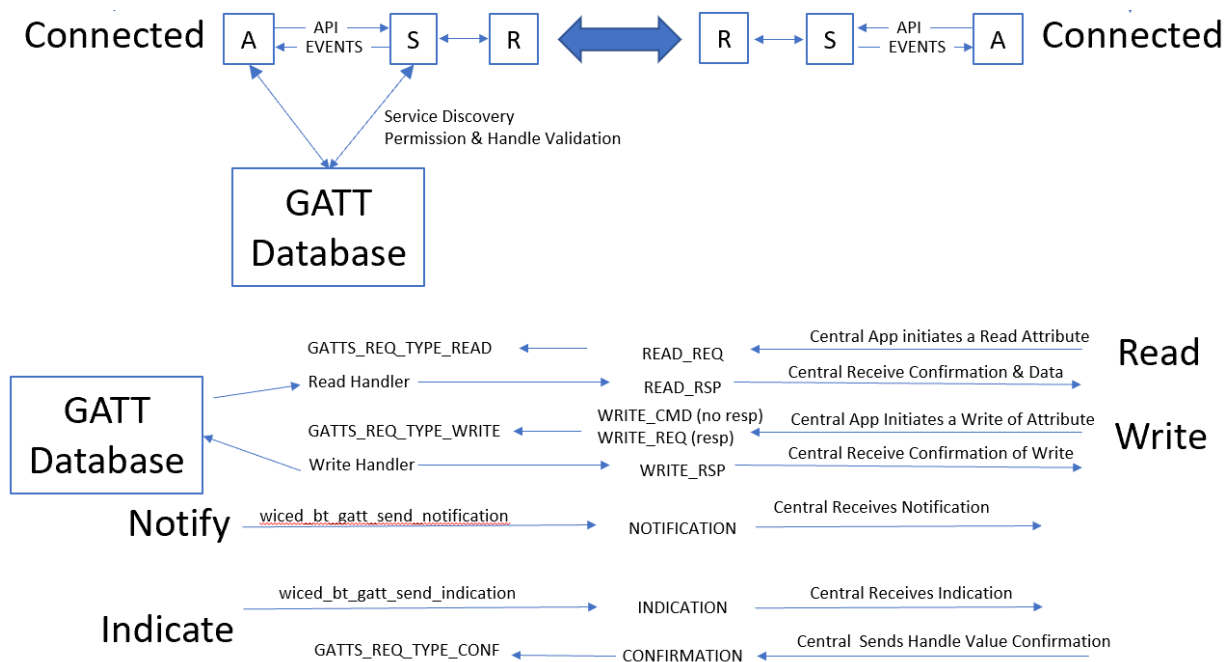
4.5 CySmart

We provide a PC and mobile application (Android and iOS) called CySmart that can act as a Central device to scan, connect, and interact with services, characteristics, and attributes of Bluetooth LE Peripherals.

We will only use the CySmart mobile application for this class. There are other utilities available for iOS and Android (such as Lightblue Explorer) that will also work. Feel free to use one of those if you are more comfortable with it.

4.5.1 CySmart PC Application

To use the CySmart PC Application, a CY5677 CySmart USB Dongle is required. When CySmart is started, it will search for supported targets and will display the results. Select the dongle that you want to use and click on "Connect".
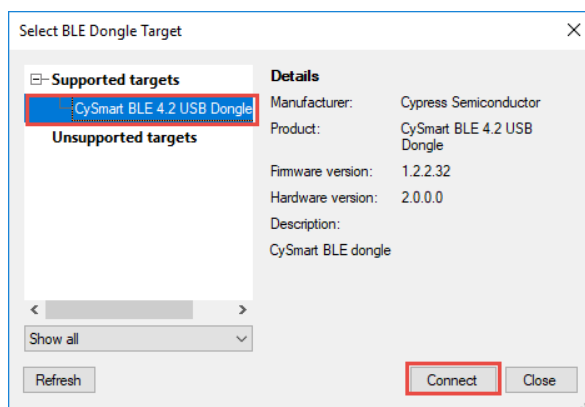
Once a dongle is selected, the main window will open as shown.
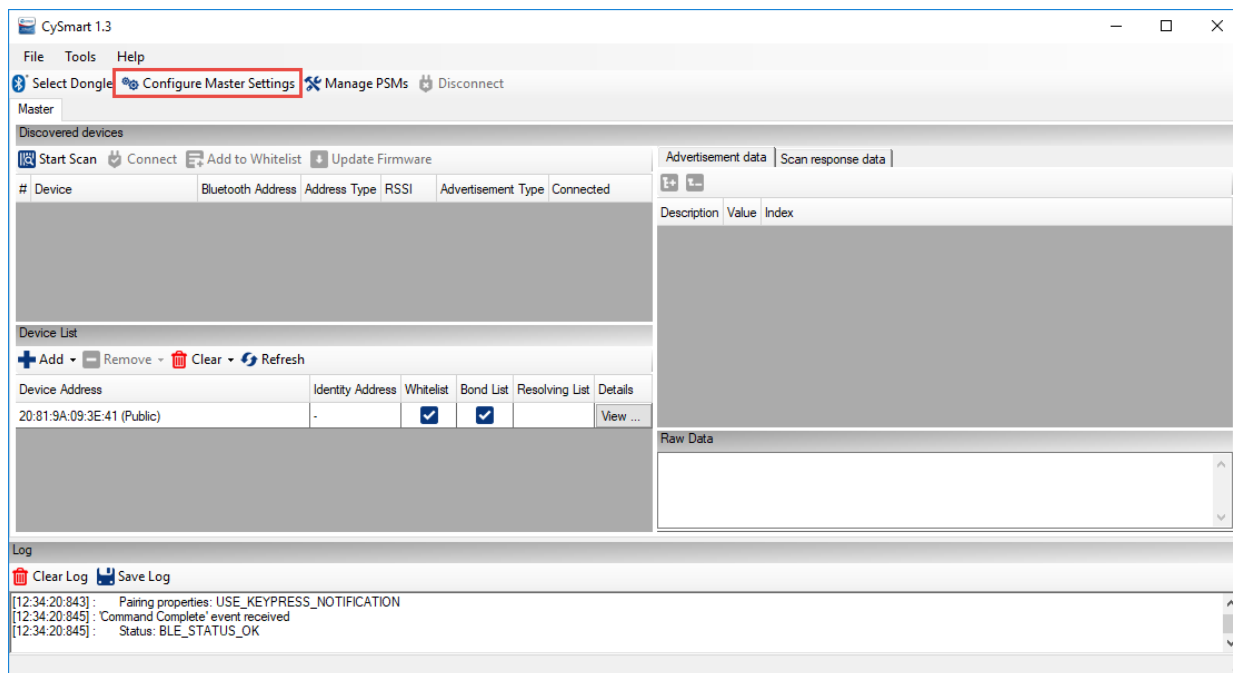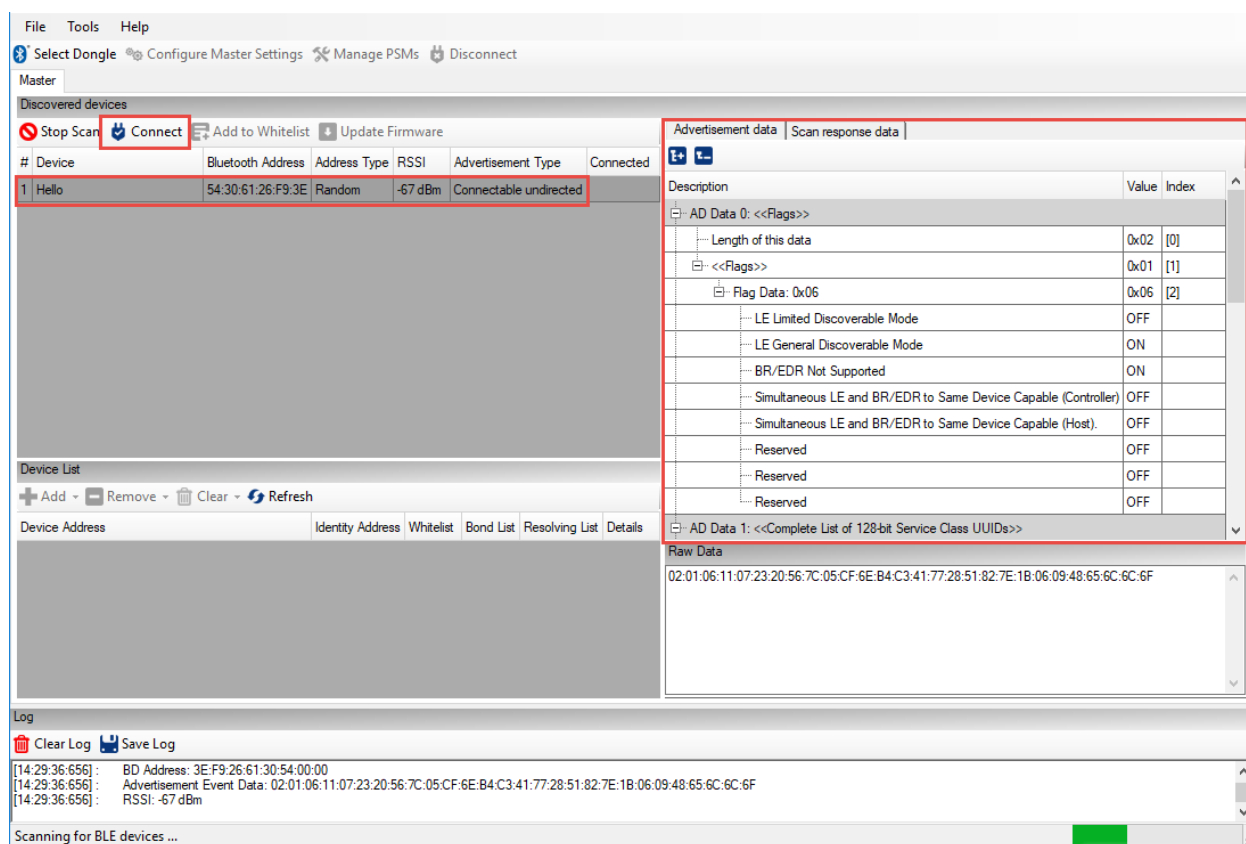
Before starting a scan, it is a good idea to configure the master settings so that scanning is done less frequently. This is especially important in a class environment where there may be many devices advertising at the same time. The tool may act strangely if it is trying to scan too fast. I recommend setting the Scan Interval to 1000 ms and the Scan Window to 100 ms. Note that these settings are NOT saved when you close CySmart, so you will need to set them each time you restart it.
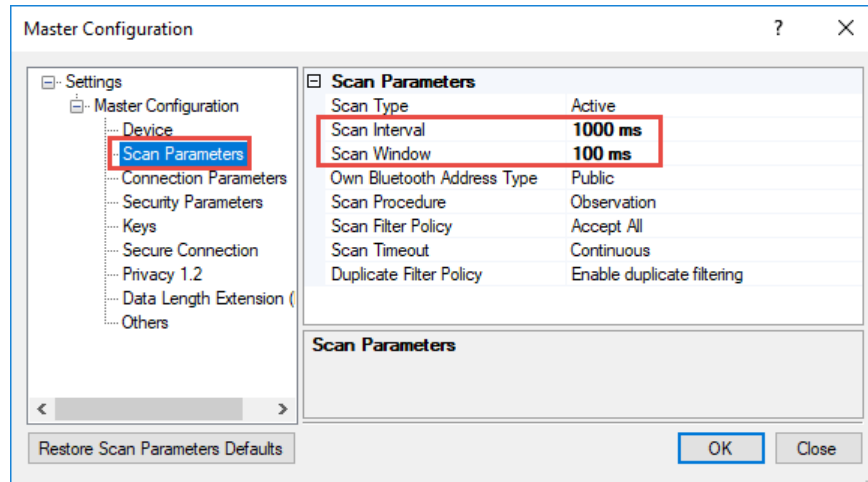
Once you click "OK" to close the Master Configuration widow, click on "Start Scan" from the main window to search for advertising Bluetooth LE devices.
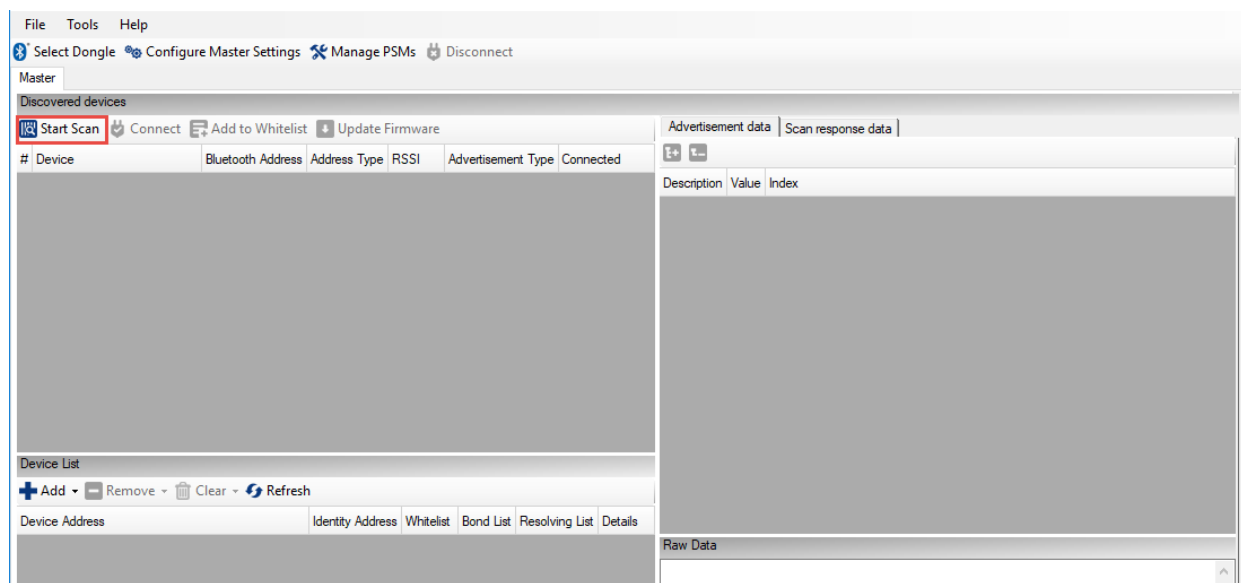
Once the device to which you want to connect appears, click on "Stop Scan". Then, click on the device in which you are interested. You can then see its Advertisement data and Scan response data on the ride side of the window. Click "Connect" to connect to the device.

When the device is connected, click on "Pair" and then "Discover All Attributes". Once that is complete, you will see a representation of all Services, Characteristics, and Attributes from the GATT database. You can read and write values by clicking on an attribute and using the buttons on the right side of the window. Click "Enable All Notifications" if you want to see real-time value updates on the left side for characteristics that have notification capability.
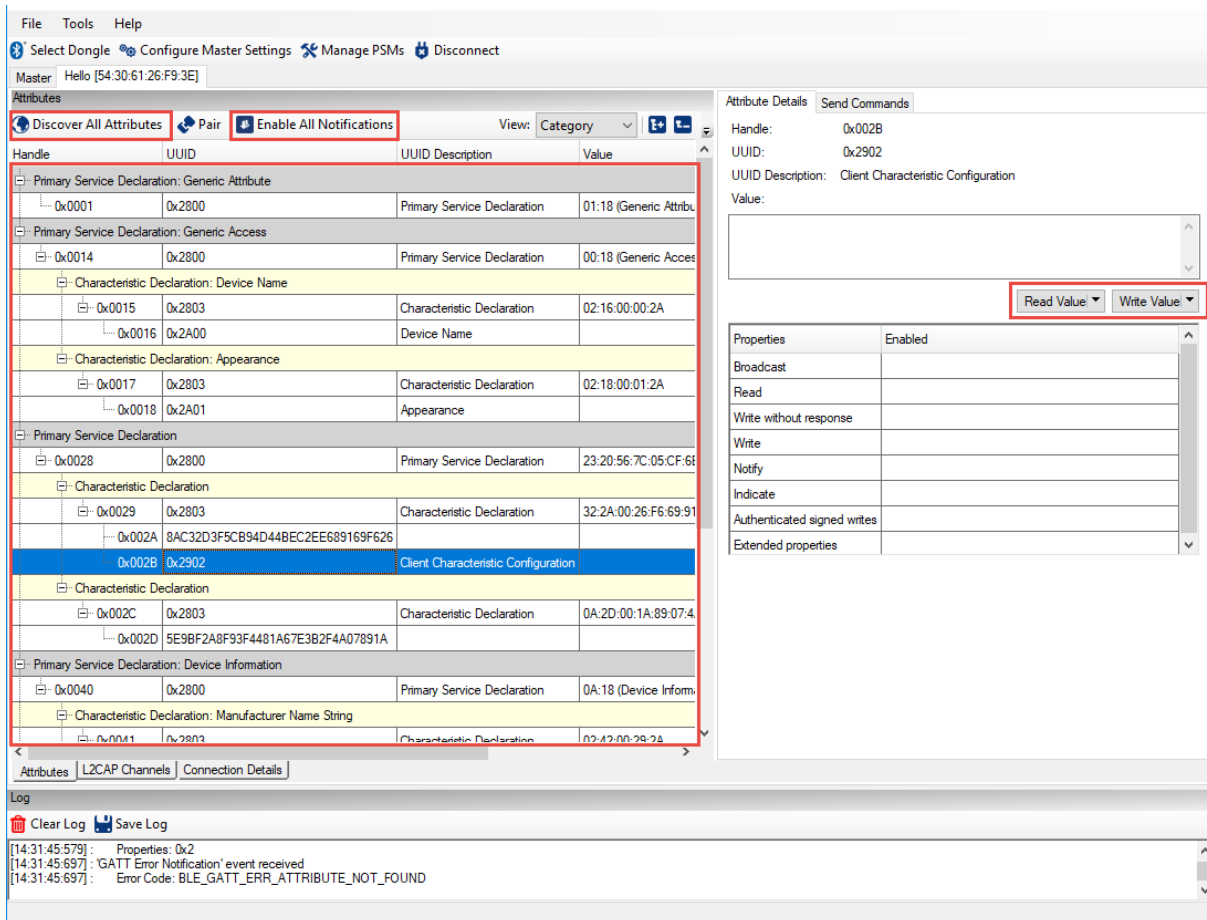

The complete User Guide for the CySmart PC application can be opened in the tool under *Help -> Help Topics*. It can also be found on the CySmart website at:

http://www.cypress.com/documentation/software-and-drivers/cysmart-bluetooth-le-test-and-debug-tool

Scroll down to the Related Files section of the page to find the User Guide.

4.5.2 CySmart Mobile Application

The CySmart mobile application is available on the Google Play store and the Apple App store. The app can connect and interact with any connectable Bluetooth LE device. It supports specialized screens for many of the Bluetooth LE adopted services and a few custom services such as CapSense and RGB LED control. In addition, there is a GATT database browser that can be used to read and write attributes for all services even if they are not supported with specialized screens.
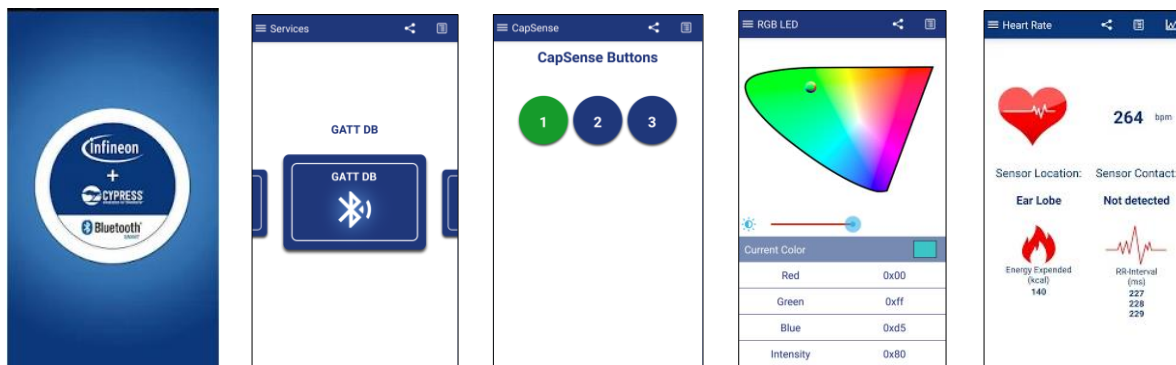
You can find complete documentation and source code on the CySmart Mobile App website at:

http://www.cypress.com/documentation/software-and-drivers/cysmart-mobile-app

You can find documentation for the custom profiles and services that are supported by the CySmart mobile app at:

http://www.cypress.com/documentation/software-and-drivers/cypresss-custom-ble-profiles-and-services