Chapter 4: More WICED BLE Peripherals

Time 2 Hours

This chapter expands up your basic knowledge of BLE Peripherals by introducing more Attribute Procedures, GATT Database Features, Security, WICED Configuration Files, More Advertising Packet Features, …

3.1 Notify & Indicate 2

3.2 Other Characteristic Descriptors 3

3.1 Security – Pairing – Bonding 4

3.1.1 Pairing 6

3.1.2 Bonding 6

3.1.3 Authentication & Authorization 6

3.2 WICED Configuration: Wiced\_bt\_cfg.c 6

3.3 WICED Configuration: Buffer Pools 6

3.4 Advertising packet 6

3.4.1 Using the Advertising Packet to Get Connected 6

3.4.2 iBeacon 7

3.4.3 Eddystone 7

3.5 GATT Service Discovery 7

3.6 WICED Bluetooth Events 8

3.6.1 More Bluetooth Management Events 8

3.6.2 More GATT Events 8

3.7 WICED Chips & the Architecture of HCI 8

3.7.1 HCI 9

3.7.2 BT Spy 10

3.8 Projects 11

3.9 WICED Bluetooth 201 12

3.9.1 Low Power 12

3.9.2 L2CAP 12

3.9.3 Other Profiles 12

3.9.4 Scan Response 12

3.9.5 Central 12

3.9.6 GATT MTU 12

3.9.7 Mesh 12

3.9.8 Non-GATT DB Based Attribute Protocols 12

3.9.9 Privacy 12

3.9.10 OTA Bootloading 12

3.9.11 Multirole devices 12

3.9.12 Direct Test Mode 12

3.10 Terms 12

# Notify & Indicate

In the previous chapter, we talked about how the GATT Client can Read and Write the GATT Database running on the GATT Server. For instance, if your Server is a CapSense Peripheral device, you might want to send the Client an update each time the CapSense values change. That leaves us with the obvious question of how does the Server initiate communication to the Client? And when is it allowed to do so?

The answer to the first question is, the Server can notify the Central that one of the values in the GATT Database has changed by sending a Notification message. That message has the Handle of the Characteristic that has changed and a new value for that Characteristic. The Notification messages are not responded to by the Central, and as such are not reliable. If you need a reliable message, you can send an Indication which the Central must respond to.

To send a Notification or Indication use the APIs

* wiced\_bt\_send\_notification(conn\_id, handle, length, value)
* wiced\_bt\_send\_indication(conn\_id, handle, length, value)

By convention, the GATT Server will not send Notification or Indication messages unless they are turned on by the Client.

How do you turn on Notifications or Indications? In the last chapter, we talked about the GATT Attribute Database, specifically, the Characteristic. If you recall, a Characteristic is composed of a minimum of two Attributes.

* Characteristic Declaration
* Characteristic Value

However, information about the Characteristic can be extended by adding more Attributes, which go by the name of Characteristic Descriptors.

For the Client to tell the Server that it wants to have Indications or Notifications, two things need to happen. First, the Server must add a new Attribute to the Characteristic Descriptors called the Client Characteristic Configuration Descriptor, often called the CCCD. This Attribute is simply a 16-bit mask field, where bit 0 represents the Notification flag, and bit 1 represents the Indication flag.

In other words, the Client can Write a 1 to bit 0 of the CCCD to tell the Server that it wants Notifications. To add the CCCD to your GATT DB use the following Macro:

* CHAR\_DESCRIPTOR\_UUID16\_WRITABLE
  + Handle
  + UID\_DESCRIPTOR\_CLIENT\_CHARACTERISTIC\_CONFIGURATION
  + LEGATTDB\_PERM\_READABLE | LEGATTDB\_PERM\_WRITE\_REQ | LEGATTDB\_PERM\_AUTH\_WRITABLE | LEGATTDB\_PERM\_AUTH\_READABLE

Then in your GATT Attribute Write Callback you will need to save the value that was written to you. If a one is written to the CCCD from then on, when a value changes in your system, you will be able to send out a new value.

# Other Characteristic Descriptors

There are a number of other interesting Characteristic Descriptors that are defined by the Bluetooth SIG including:



A common Characteristic Descriptor to use is the Characteristic User Description which is just a text string that describes in human format the Characteristic Type. Many GATT Database Browsers (e.g. Light Blue) will display this information when you are looking at the GATT Database. To add the Characteristic User Description to your Characteristic just add

* CHAR\_DESCRIPTOR\_UUID16
  + handle
  + UUID\_DESCRIPTOR\_CHARACTERISTIC\_USER\_DESCRIPTION
  + LEGATTDB\_PERM\_READABLE

WICED Bluetooth has defines for the rest of the Descriptors which you can find in wiced\_bt\_uuid.h



# Security – Pairing – Bonding

BLE 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:

1. No Input, No Output
2. Display Only
3. Display
4. Display: Yes/No
5. Keyboard Only

Need to understand these better and explain the possible options depending on the capabilities. What is display only vs. display yes/no vs display?

Need to understand/explain what data signed means. How is this different from encryption?

Need details on authentication and encryption schemes.

Once two BLE 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:



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 BLE device can connect to it.

## Pairing

## Bonding

## Authentication & Authorization

# WICED Configuration: Wiced\_bt\_cfg.c

# WICED Configuration: Buffer Pools

# Advertising packet

There are two main uses of the advertising packet:

* Identifying a Peripheral with some recognizable data so that a Central knows how to connect and talk to it
* Sending out data (beacon data)

## Using the Advertising Packet to Get Connected

If you turn on the CySmart GATT browser, you will find that there are likely a bunch of unknown devices that are advertising around you. For instance, as I sit here right now I can see that there are several Bluetooth LE devices around me that I have no idea what they are.



When a Central wants to connect to a Peripheral, how does it know what Peripheral to talk to? There are two answers to that questions.

First, it may advertise a service that the Central knows about (because it is defined by the Bluetooth SIG or is custom to your company). As we talked in the previous chapter you can customize the Advertising packet with information. In the picture above, you can see that some of the devices are advertising that they support 1 service. To do that they add a field of one of these types:



to the advertising packet along with the UUID of the Service.

The other scheme that is commonly used is to advertise “Manufacturers Specific Data”. This data has two parts

* A two-byte manufacturer code as specified by the Bluetooth SIG (e.g. Cypress = 0x0131)
* The actual data which is typically a unique product ID

The way that this works is that you would write a Central application that would have a table of known Peripheral Product IDs that it knew how to talk to. Then the Peripherals would advertise their custom Product ID and Company code in the Manufacturers Data Field. When a Central saw something that it knew how to talk to it would make the connection.

## iBeacon

iBeacon is a Advertising Packet format defined

## Eddystone

# GATT Service Discovery

We know that for a Central to read and write the GATT Database, it must know the handles of the characteristics. If the handles were not established a-priori (e.g. from the Product ID scheme described in Section x.y.z), then you will need some mechanism to figure them out. That mechanism is called GATT Service Discovery.

Previously, we talked about Attribute protocols functions, Read, Write, Notify, Indicate. The Service Discovery procedures uses another Attribute function called “Read Group By Type”. The Group is just a range of Handles, and Type is the Attribute type. When a Central wants to discover all the Primary Services on a Peripheral, it will send a Read Group by Type request with the Handle Range set to 1 🡪 0xFFFF (all the possible Handles) and the Attribute Type set to <<Primary Service>>. The Peripheral will then respond with a list of the Primary Services, the UUIDs, the Handle start and end range for each Service.

As the Central will then know all the Service UUIDs and Handles, it can then iterate through each of the sub-ranges using the same Read Group by Type and look for Characteristics, Descriptor etc.

On the Peripheral, the WICED Bluetooth Stack has a reference your GATT Database, and as such it responds to these requests automatically for you.

On the Central you will need to implement this service discovery algorithm by calling the function wiced\_bt\_gatt\_send\_discover to execute the Read Group by Type request. Then iterating through the responses to figure out the Handles, UUID etc.

# WICED Bluetooth Events

## More Bluetooth Management Events

## More GATT Events

# WICED Chips & the Architecture of HCI

In many complicated systems, hierarchy, is used to manage the complexity. WICED Bluetooth is no different. The WICED Bluetooth Stack is called a Stack because it is a set of blocks that have well defined interfaces. Here is a simple picture of the software system that we have been using. You have been writing code in the block called “Application”. You have made API calls and gotten events from the “Attribute Protocol” and you implemented the “Generic Attribute Profile” by building the GATT Database. Moreover, you advertised using GAP and you Paired and Bonded by using the Security Manager.



## HCI

The next block to talk about is the “Host Controller Interface”.

For technical and cost reasons, when Bluetooth was originally built, the Radio was a separate chip from the chip that was running your Application. The Radio chip took the name of Controller, because it was the Radio and Radio Controller. And the chip running the Application was called the Host, because it was hosting the Application.

The interface between the Host and the Controller was typically a UART, SPI or SDIO. The data flying over that serial connection was formatted in a Bluetooth SIG specific packets called “HCI Packets”. The WICED Software team extended this packet format and is now called “WICED HCI”.

By standardizing the HCI interface it allowed big application processors (like those exiting in PCs and cellphones) to interface with Bluetooth. As time went by the Host and Controller have frequently merged into one chip e.g. 20719, however the WICED HCI interface persists.



The WICED Bluetooth Stack can be split into a “Host” and a “Controller” part for example the PSoC 6 and 4343W Combo Radio looks like this:



## BT Spy

The BTSPY is a debugging tool provided by Cypress that can sniff the WICED data packets that are crossing the HCI interface. In a monolithic setup, like the 20719, WICED created a Virtual HCI interface that can be reflected to a Serial UART. In a split setup, like the 4343W, the HCI interface can be “mirrored” to a one of the Serial Ports.

This tool will be talked about in detail in Chapter 6 - Debugging

# Projects

Add notify to the CapSense project

Add Paring to the CapSense project

Add Bonding to the CapSense project

Make an Eddystone project and use android to do something

Make an iBeacon project

# WICED Bluetooth 201

## Low Power

## L2CAP

## Other Profiles

### AMS – Apple Media Service

### ANS – Alert Notification Service

### BAS – Battery Service

#### Example Server Project BAS

#### Example Client Project is BAC

### HRS – Heart Rate Service

#### Example Server HRS

#### Example Client HRC

### ANCS – Apple Notification Center Service

### HID – Human Interface Device

## Scan Response

## Central

## GATT MTU

## Mesh

## Non-GATT DB Based Attribute Protocols

## Privacy

## OTA Bootloading

## Multirole devices

## Direct Test Mode

# Terms

Central

Peripheral

GATT

GAP

Service

Profile

Characteristic

Attribute