Chapter 4: More WICED BLE

Time 2 Hours

After completing chapter 4 you will have all the required knowledge to create the most basic WICED Bluetooth Low Energy Peripheral.

3.1 Notify & Indicate 2

3.2 Other Characteristic Descriptors 2

3.1 Security – Pairing – Bonding 2

3.2 Authentication & Authorization 2

3.3 Wiced\_bt\_cfg.c 2

3.4 Buffer Pools 2

3.5 Advertising packet fields 2

3.6 GATT Service Discovery 2

3.7 Mesh 2

3.8 Central 2

3.9 Non-GATT DB Based Attribute Protocols 2

3.10 GATT MTU 3

3.11 More Bluetooth Management Events 3

3.12 More GATT Events 3

3.13 OTA Bootloading 3

3.14 Low Power 3

3.15 WICED Chips 3

3.16 Terms 3

# 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 two other interesting Characteristic Descriptors, the Characteristic User Description and the Characteristic Presentation Format.

The Characteristic User Description is just a text string that describes in human format the Characteristic. 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

Another interesting Characteristic Descriptor is the

# 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.

## Authentication & Authorization

# Wiced\_bt\_cfg.c

# Buffer Pools

# Advertising packet

## Advertising packet fields

## Using the advertising packet to get connected

## iBeacon

## Eddystone

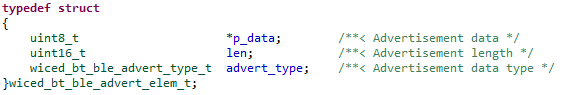
Lots of different information can be sent in an advertisement packet (e.g. device name, device appearance, primary service UUID etc.), but the total payload size is limited to a maximum of 31 bytes. The first element sent is the flags which is required. Each element (including the flags) uses the first byte to specify its payload length, the second byte to specify the type of data that follows, and the remaining bytes for the actual data.

For example, if you want to send flags, a 128-bit service UUID, and an 8-character device name, you would have:

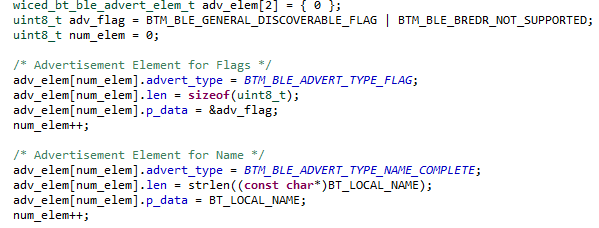
* Flags: 1-byte size + 1-byte Flag type + 1-byte Flag data = 3 bytes
* UUID: 1-byte size + 1-byte UUID type + 16-bytes for the 128-bit UUID data = 18 bytes
* Name: 1-byte size + 1-byte Name type + 8-byte Name data = 10 bytes
* Total = 3 + 18 + 10 = 31 bytes

You can send a shortened part of the name if necessary (i.e. *BTM\_BLE\_ADVERT\_TYPE\_NAME\_SHORT* instead of *BTM\_BLE\_ADVERT\_TYPE\_NAME\_COMPLETE*).

The advertisement data is set by creating an array of type *wiced\_ble\_advert\_elem\_t*. Each entry in the array is a structure that contains the type of the entry, the size of the entry, and a pointer to the data:



For example, a two-entry advertisement packet with Flags (for a BLE only device that is generally discoverable) and the name of the device would look like this:



Once the array is populated, the function *wiced\_bt\_ble\_set\_raw\_advertisement\_data* is called with the number of advertisement packet entries and a pointer to the array like this:



Finally, advertisements are started by calling *wiced\_bt\_start\_advertisements*:



# GATT Service Discovery

# More Bluetooth Management Events

# More GATT Events

# WiFi / BT Combo (Chapter 6)

# Low Power

# L2CAP

# WICED Chips Architecture of hci

# WICED Bluetooth 201

## 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