Chapter 4D: BLE Centrals

Time 2 ½ Hours

This chapter introduces you to the Central side of the BLE connection. By the end of this chapter you should be able to create a BLE Central that finds the right BLE Peripheral, connects to it, Reads and Writes and Accepts notifications.

4B.1 Central Roles 2

4B.2 WICED Scanning 2

4B.3 Attribute Protocol 2

4B.3.1 Read 2

4B.3.2 Write 2

4B.3.3 Notify 2

4B.3.4 Indicate 2

4B.3.5 Find Information 2

4B.3.6 Find by Type Value 2

4B.3.7 Read by Group Type 2

4B.4 Service Discovery 2

4B.5 Running a GATT Server 2

4B.6 Exercises 2

Exercise 4B.1 Make an Observer 2

Exercise 4B.2 Add a filter to show only your device 2

Exercise 4B.3 Update (2) to connect to your device & turn on/off the LED 2

Exercise 4B.4 Add keys to turn on CCCD 2

Exercise 4B.5 Make your project do service discovery 2

Exercise 4B.6 Make a central that is a “TV” running a GATT DB 2

Exercise 4B.7 Make a remote control peripheral to talk to the TV 2

Exercise 4B.8 Make your TV Handle multiple remote controls 2

Exercise 4B.9 Update your peripheral remote control to do service discovery 2

Exercise 4B.10 Run the advertising scanner 2

# Central Roles

# WICED Scanning

Analyzing the advertising packet

# Attribute Protocol

## More on Read, Write, Notify and Indicate

## Find Information

Input – starting handle and ending handle

Output – handles, types

## Find by Type Value

Input – starting handle, ending handle, Attribute type & Attribute value

Output – handles, values

* Discover all of the characteristics of a service

## Read by Group Type

Input – starting, ending handle, group type (Service, Char, etc)

Output – handle, end group handle, value

# Service Discovery

# Running a GATT Server

# Exercises

* 1. Make an Observer

This project will listen to all the BLE devices that are broadcasting. It should print out the BD Address of each device. To build this project follow these steps:

Run BT designer and make a project with no gatt database

Delete the advertising functions and function calls

Make a function to process the scanned advertising packets which just prints out the BD Address of the remote. This function should have the protype of:

typedef void (wiced\_bt\_ble\_scan\_result\_cback\_t) (wiced\_bt\_ble\_scan\_results\_t \*p\_scan\_result, uint8\_t \*p\_adv\_data);

Add a call to wiced\_bt\_ble\_scan to the BTM enabled event with a function pointer to your advertiser processor function.

Build and Program

What is the cause of “Unhandled Bluetooth Management Event: 0x16 (22)” and how do you fix it?

* 1. Add a filter to show only your device

Copy Exercise 01. Change your advertising packet call back to only print out devices that match

1. Your Device Name
2. Your Service UUID
3. Your Manufactures Code

You can use the function wiced\_bt\_ble\_check\_advertising\_data to look at the advertising packet and find fields of type wiced\_bt\_ble\_advert\_type\_t. If there is a field that matches it will return a pointer to those bytes and a length.

You can use strncmp (to compare strings) and memcmp (to compare UUIDs) to see if the fields match what you are looking for.

* 1. Update to connect to your peripheral device & turn on/off the LED

To do this exercise will require two development kits. One to be your peripheral, which should be programmed with the firmware from chapter 04c exercise xx.

Your project will scan and find the peripheral. Then connect to it. Then let you send 0’s and 1’s to led characteristic. To simplify the project, you will hardcode the handle, but in the upcoming exercises you will add service discovery.

The steps are as follows:

Copy the previous project

Build and program it to make sure it still works

Add a keyboard interface that has a ‘?’ To print out help. Remember from chapter 2, you will need to initaizlie the puart and setu a call back in the \_app\_init function.

Create a RxCallback function to handle key presses. Make a switch with one case per key (used by this program).

Add keys to turn on/off scanning ‘s’ ON and ‘S’ off (call the wiced\_bt\_ble\_scan with the correct arguments)

Remove the start scan from the BTM\_Enabled Event

Program your projects and make sure that the keyboard interface works and that the s/S turns the scanning on and off.

Update your adv packet function to connect to a the Peripheral when it finds one that it recognizes by calling wiced\_bt\_gatt\_le\_connect.

After starting the connection, turn off scanning

Create a new function to serve as the gatt callback function. The easiest way to do this is to copy the callback from one of the previous projects and then remove all the cases except the default case. Regardless, the function must match this prototype:

typedef wiced\_bt\_gatt\_status\_t wiced\_bt\_gatt\_cback\_t(wiced\_bt\_gatt\_evt\_t event, wiced\_bt\_gatt\_event\_data\_t \*p\_event\_data);

Register your gatt callback function in the BTM enabled event with function wiced\_bt\_gatt\_register

Add a global variable uint16\_t to save the connection id.

In the gatt callback when you get a GATT\_CONNECTION\_STATUS\_EVT figure out if it is a connect or a disconnect. Then save the connection id to your global variable

Add a ‘d’ to call disconnect

Program your project. Make sure that it finds and connects. Then disconnects. Then start the scanning and make sure that it reconnects

Notice the Unhandled BTM event 0x16 message. Add a case to print out the “right” message

Create a new global variable called ledHandle. HARDCODE its initial value to the handle of your LED Characteristic. You won’t change the variable in this exercise, but in a future one you will find the handle via a service discovery.

Create a new function to write the led with a uint8\_t argument that represents either 1 or 0. If there is no connection or the ledHandle is 0 then you should return.

Add cases for 1 and 0 in your command that call your write led function with 0 or 1

Program and test

* 1. Add keys to turn on CCCD

In this project, we will add setting up the CCCD to turn on notifications, and printing out messages when the notifications happens.

Copy the previous project

Build and program it to make sure that it still works.

Add a new global variable called cccdHandle to hold the handle of the CCCD. Setup its initial value to 0x?? (this is a hardcode which we will fix in the next exercise)

Add a new function called writeCCCD. This function is almost the same as writeLED except that its argument should be uint16\_t instead of uint\_8 (because CCCD is a 16-bit number). All the function needs to be updated to reflect the length of 2 instead of 1.

Add cases ‘n’ and ‘N’ to set and unset the CCCD and add those keys to the help printout.

Add a case for GATT\_OPERATION\_CPLT\_EVT into the gatt callback. This case should printout the connection id, operation, status, handle, length and the raw bytes of data

Build Program and Test

* 1. Make your project do service discovery
  2. Make a central that is a “TV” running a GATT DB
  3. Make a remote control peripheral to talk to the TV
  4. Make your TV Handle multiple remote controls
  5. Update your peripheral remote control to do service discovery
  6. Run the advertising scanner