Chapter 6B: More Classic Bluetooth

Time: 3 Hours

At the end of this chapter you will know about additional Classic Bluetooth Profiles such as A2DP, AVRCP, HSP, HFP, and HID as well as other more advanced topics.

6B.1 Profiles

Classic Bluetooth devices communicate with one another by using one or more of a standard set of profiles (often called services) which is maintained by the Bluetooth Special Interest Group (SIG). By using standard profiles, devices only need to determine the profile to use to start communicating rather than having to transmit the communication parameters themselves. A list of Bluetooth Profiles can be found at:

https://en.wikipedia.org/wiki/List_of_Bluetooth_profiles

Some of the more commonly used profiles are:

6B.1.1 Advanced Audio Distribution Profile (A2DP)

The A2DP profile is used for streaming multi-media audio. It is used, for example, when streaming audio from a mobile phone to a wireless headset or a car sound system. This profile is often used in conjunction with AVRCP, HSP, or HFP as described below.

The A2DP profile is designed for a unidirectional audio stream of up to 2-channel stereo. There may be more than one A2DP profile on a single device.

6B.1.2 Audio/Video Remote Control Profile (AVRCP)

The AVRCP profile is designed to provide a standard remote-control interface for devices such as televisions, stereo equipment, in-car navigation systems, etc.

There are several versions available depending on the functionality required, each of which is a superset of the previous version.

Version	Functionality
1.0	Basic remote (play, pause, stop, etc.).
1.3	1.0 plus metadata (such as artist, track name, etc.) and player state (such as playing, stopped, etc.)
1.4	1.3 plus multiple media player browsing including a "Now Playing" list and search capabilities. Also has support for absolute volume.
1.5	1.4 plus corrections/clarifications to absolute volume control
1.6	1.5 plus browsing and track information. Support for sending cover art through BIP/OBEX (Basic Imaging Profile and Object Exchange Profile)

6B.1.3 Headset Profile (HSP)

The HSP provides support for headsets including two-way 64 kbit/sec audio and minimal controls for ringing, answer a call, hang up and adjust the volume.

In a typical headset, A2DP will be used when listening to music since it provides the best quality stereo connection, but HSP will be used when making a phone call since it allows two-way communication.

6B.1.4 Hands-Free Profile (HFP)

The HFP is commonly used to allow car hands-free kits to communicate with mobile phones. It provides relatively low-quality monaural audio to allow the user to control some features of their phone such as making calls, playing music, etc. It is often used with other profiles such as A2DP to provide high quality audio streaming.

6B.1.5 Human Interface Device Profile (HID)

The HID is used for devices such as mice, keyboards, and joysticks. It provides a low-latency link with minimal power requirements.

Keyboards and keypads must be secure, but for other devices using the HID profile security is optional.

6B.1.6 Object Exchange (OBEX)

The OBEX (short for OBject EXchange, also called IrOBEX) is used to transmit binary objects between devices (such as business cards, data, or even applications). The transfer is similar to HTTP, as it provides a way to connect to a server (another Bluetooth device) and request or provide objects.

6B.1.7 Personal Area Networking Profile (PAN)

This profile is intended to allow the use of Bluetooth Network Encapsulation Protocol on Layer 3 protocols for transport over a Bluetooth link.

6B.1.8 File Transfer Profile (FTP)

Provides the capability to browse, manipulate and transfer objects (files and folders) in an object store (file system) of another system. Uses GOEP (Generic Object Exchange Profile) as a basis.

6B.1.9 Intercom Profile (ICP)

Commonly referred to as “walkie-talkie profile”, this profile is used to allow voice calls between two Bluetooth-capable handsets over Bluetooth, but the standard was withdrawn in 2010.

6B.1.10 Device ID Profile (DIP)

This profile is used to enable identification of the manufacturer, project ID, product version, and the version of the Device ID specification being met. This assists in identifying the correct drivers when a Bluetooth device attempts to connect to a PC.

6B.1.11 Health Device Profile (HDP)

This profile is used for the transmission and reception of Medical Device data.

The Health Thermometer Profile (HTP) and Heart Rate Profile (HRP) fall under this category as well.

6B.2 Master

TODO: Write this

6B.3 OTA (Over the Air) Upgrade

GJL: This is BLE so it should move to chapter 4C (or maybe chapter 7 – advanced topics?). The existing 4C should become 4D. In the examples and exercises, I'm assuming that OTA will be exercise 4C.1

The firmware upgrade feature provided in WICED Studio allows an external device to use the Bluetooth link to transfer and install a newer firmware version on devices equipped with CYW20719 (as well as CYW20706 and CYW20735) chips. This section describes the functionality of the WICED Firmware Upgrade library used in various WICED Studio sample applications.

The library is split into two parts. The over the air (OTA) firmware upgrade module of the library provides a simple implementation of the GATT procedures to interact with the device performing the upgrade. The firmware upgrade HAL module of the library provides support for storing data in the non-volatile memory and switching the device to use the new firmware when the upgrade is completed. The embedded application may use the OTA module functions (which in turn use the HAL module functions), or the application may choose to use the HAL module functions directly.

The library contains functionality to support secure and non-secure versions of the upgrade. In the non-secure version, a simple CRC32 verification is performed to validate that all bytes that have been sent from the device performing the upgrade are correctly saved in the serial flash of the device. The secure version of the upgrade validates that the image is correctly signed and has correct production information in the header.

6B.3.1 Design and Architecture

External or on-chip flash memory of the Cypress WICED chips is organized into two partitions for the failsafe upgrade capability. During the startup operation the boot code of the chip checks the first partition and if a valid image is found, assumes that the first partition is active and then starts executing the code in the first partition. If the first partition does not contain a valid image, the boot code checks the second partition and then starts execution of the code in the second partition if a valid image is found there. If neither partition is valid, the boot code enters download mode and waits for the code to be downloaded over HCI UART. The addresses of the partitions are programmed in a file with extension "btp" located in the platform directory of the SDK.

The firmware upgrade process stores received data in the inactive partition. When the download procedure is completed and the received image is verified and activated, the currently active partition is invalidated, and then the chip is restarted. After the chip reboots, the previously inactive partition becomes active. If for some reason the download or the verification step is interrupted, the valid partition remains valid and the chip is not restarted. This guarantees the failsafe procedure.

The following table shows the recommended memory configuration for an application upgrading the firmware on a device with external 4Mbit serial flash:

Section Name	Offset	Length	Description
Static Section (SS)	0x0000	0x2000	Static section used internally by the chip firmware
Volatile Section (VS1)	0x2000	0x1000	First volatile section used for the application and the stack to store data in the external or on-chip flash memory. One serial flash sector.
Volatile Section (VS2)	0x3000	0x1000	Used internally by the firmware when VS1 needs to be defragmented.
Data Section (DS1)	0x4000	0x3E000	First partition.
Data Section (DS2)	0x42000	0x3E000	Second partition.

During an OTA upgrade the device performing the procedure (Downloader) pushes chunks of the new image to the device being upgraded. The embedded application receives the image and stores it in the external or on-chip flash. When all the data has been transferred, the Downloader sends a command to verify the image and passes a 32-bit CRC checksum. The embedded app reads the image from the flash and verifies the image. For the non-secure download, the library calculates the checksum and verifies that it matches received CRC. For the secure download case, the library performs ECDSA verification and verifies that the Product Information stored in the new image is consistent with the Product Information of the firmware currently being executed on the device. If verification succeeds, the embedded application invalidates the active partition and restarts the chip. The simple CRC check can be easily replaced with crypto signature verification if desired, without changing the download algorithm described in this document.

Applications for Loading New Firmware

WICED Studio contains two peer applications that can be used to transmit new firmware – one for Android and one for Windows over BLE. Both applications contain the source file as well as pre-compiled executables (.apk for Android and .exe for Windows). The Windows executable is provided for 32-bit (x86) and 64-bit (x64) architectures.

These peer applications can be found in the peer_apps folder inside the ota_firmware_upgrade application folder, or on the file system in:

<Install Dir>\WICED-Studio-n.n\common\peer_apps\ota_firmware_upgrade\Windows\WsOtaUpgrade\Release

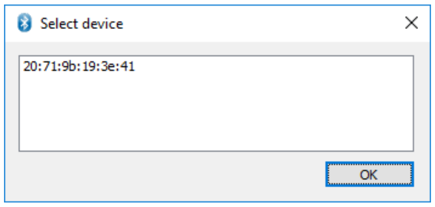
<Install Dir>\WICED-Studio-n.n\common\peer_apps\ota_firmware_upgrade\Andoird\LeOTAApp\app\build\outputs\apk

Windows

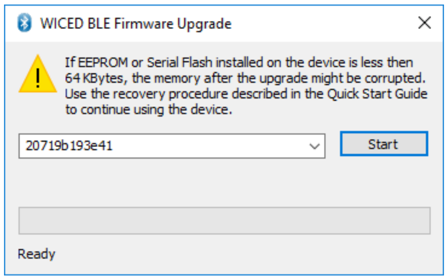
To use the Windows peer application, you must first copy the *.bin file from the build directory of the WICED application into the same folder as the Windows peer application. Then run the application with the *.bin file provided as an argument. For example, from a command or PowerShell window:

```
.\WsOtaUpgrade.exe ex01_ota-WW101_2_CYW900719Q40EVB_01-rom-ram-Wiced-release.ota.bin
```

You will get a window that looks like the following. Select the device you want to update and click "OK".

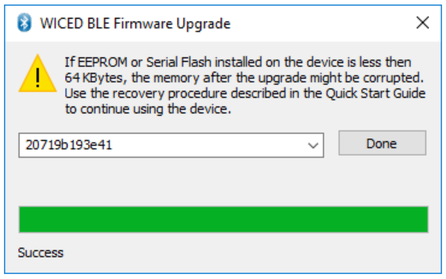


On the next window, verify that the device type is correct and click "Start".


WICED BLE Firmware Upgrade

If EEPROM or Serial Flash installed on the device is less then 64 KBytes, the memory after the upgrade might be corrupted. Use the recovery procedure described in the Quick Start Guide to continue using the device.

If the update worked, the window will show "Success" at the bottom. Click "Done" to close the window.

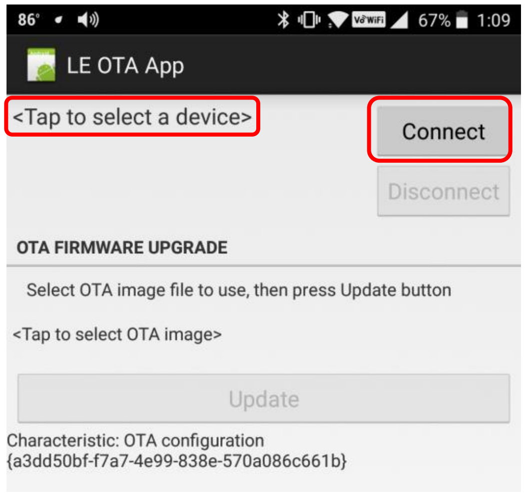

WICED BLE Firmware Upgrade

If EEPROM or Serial Flash installed on the device is less then 64 KBytes, the memory after the upgrade might be corrupted. Use the recovery procedure described in the Quick Start Guide to continue using the device.

Android

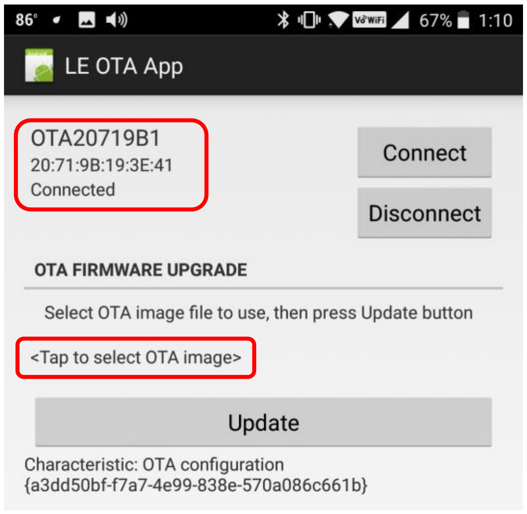
To use the Android app:

1. Install the app-debug.apk file on your Android device if you have not already done so.
2. Copy the *.bin file from the build directory onto the device in a location where you can find it.
3. Run the app called *LE OTA App*. The startup screen will look like this:



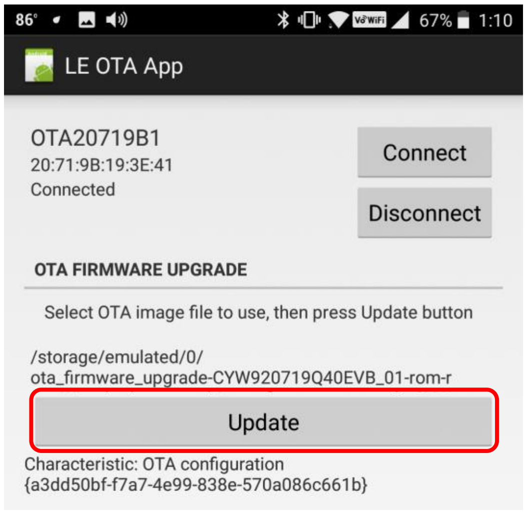
Characteristic: OTA configuration
{a3dd50bf-f7a7-4e99-838e-570a086c661b}

4. Tap where it says <Tap to select a device> and choose your device from the list.
5. Tap on the "Connect" button. Once connected, the screen will look like this:



Characteristic: OTA configuration
{a3dd50bf-f7a7-4e99-838e-570a086c661b}

6. Tap where it says <Tap to select OTA Image>, navigate to where you saved the *.bin file on your device and select it. Once the file is selected, the screen will look like this:



OTA20719B1
20:71:9B:19:3E:41
Connected

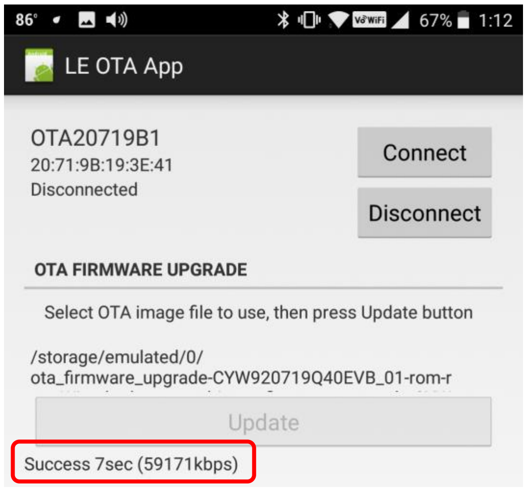
OTA FIRMWARE UPGRADE

Select OTA image file to use, then press Update button

/storage/emulated/0/
ota_firmware_upgrade-CYW920719Q40EVB_01-rom-r

Characteristic: OTA configuration
{a3dd50bf-f7a7-4e99-838e-570a086c661b}

7. Tap the Update button. Once the update is done, you should see "Success" at the bottom of the screen. Disconnect from the device and close the app.



OTA20719B1
20:71:9B:19:3E:41
Disconnected

OTA FIRMWARE UPGRADE

Select OTA image file to use, then press Update button

/storage/emulated/0/
ota_firmware_upgrade-CYW920719Q40EVB_01-rom-r

Success 7sec (59171kbps)

OTA Firmware

In the firmware, OTA requires the following:

Header Files

Include the following header files at the top of your main C file:

```
        #include "wiced_bt_firmware_upgrade.h"
        #include "wiced_bt_fw_upgrade.h"
```

Library

Include the OTA library in the makefile.mk:

```
        $(NAME)_COMPONENTS := fw_upgrade_lib.a
```

BLE OTA Service (Non-Secure)

The GATT database must have a Primary Service defined for the OTA service. The Service and its two Characteristics are defined as follows:

```
// OTA Firmware Upgrade Service
PRIMARY_SERVICE_UUID128(HANDLE_OTA_FW_UPGRADE_SERVICE, UUID_OTA_FW_UPGRADE_SERVICE),

        CHARACTERISTIC_UUID128_WRITABLE(HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_CONTROL_POINT,
        HANDLE_OTA_FW_UPGRADE_CONTROL_POINT, UUID_OTA_FW_UPGRADE_CHARACTERISTIC_CONTROL_POINT,
        LEGATTDB_CHAR_PROP_WRITE | LEGATTDB_CHAR_PROP_NOTIFY | LEGATTDB_CHAR_PROP_INDICATE,
        LEGATTDB_PERM_VARIABLE_LENGTH | LEGATTDB_PERM_WRITE_REQ /*| LEGATTDB_PERM_AUTH_WRITABLE*/
        ),

           CHAR_DESCRIPTOR_UUID16_WRITABLE(HANDLE_OTA_FW_UPGRADE_CLIENT_CONFIGURATION_DESCRIPTOR,
           UUID_DESCRIPTOR_CLIENT_CHARACTERISTIC_CONFIGURATION, LEGATTDB_PERM_READABLE |
           LEGATTDB_PERM_WRITE_REQ /*| LEGATTDB_PERM_AUTH_WRITABLE */),

        CHARACTERISTIC_UUID128_WRITABLE(HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_DATA,
        HANDLE_OTA_FW_UPGRADE_DATA, UUID_OTA_FW_UPGRADE_CHARACTERISTIC_DATA,
        LEGATTDB_CHAR_PROP_WRITE, LEGATTDB_PERM_VARIABLE_LENGTH | LEGATTDB_PERM_WRITE_REQ /*|
        LEGATTDB_PERM_AUTH_WRITABLE */),
```

Note that the LEGATTBD_PERM_AUTH_WRITABLE are commented out in all three of the above so writes can be done without an authenticated link. Uncomment them if you want an authenticated link before allowing OTA updates.

Initialization

During the application initialization (typically just after initializing the GATT database with wiced_bt_gatt_db_init), the following function call must be made:

```
        /* Initialize OTA (non-secure) */
        wiced_ota_fw_upgrade_init(NULL, NULL);
```

GATT Event Handler Functions

The handler functions for the GATT events *GATTS_REQ_TYPE_READ*, *GATTS_REQ_TYPE_WRITE*, and *GATTS_REQ_TYPE_CONF* must call the appropriate OTA functions. Note that these are in addition to any other functionality required for the normal application functionality.

If your starting project does not have an indication confirmation handler function, it must be created, and that case must be added to the GATT event callback function:

```
case GATTS_REQ_TYPE_CONF:
    status = ex01_ota_indication_cfm_handler(p_data->conn_id, p_data->data.handle);
    break;
```

Handler for *GATTS_REQ_TYPE_READ:*

```
wiced_bt_gatt_status_t ex01_ota_read_handler( wiced_bt_gatt_read_t *p_read_req, uint16_t
conn_id )
{
    wiced_bt_gatt_status_t status = WICED_BT_GATT_INVALID_HANDLE;

    switch(p_read_req->handle)
    {
    // If the indication is for an OTA service handle, pass it to the library to process
    case HANDLE_OTA_FW_UPGRADE_SERVICE:
    case HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_CONTROL_POINT:
    case HANDLE_OTA_FW_UPGRADE_CONTROL_POINT:
    case HANDLE_OTA_FW_UPGRADE_CLIENT_CONFIGURATION_DESCRIPTOR:
    case HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_DATA:
    case HANDLE_OTA_FW_UPGRADE_DATA:
    case HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_APP_INFO:
    case HANDLE_OTA_FW_UPGRADE_APP_INFO:
        status = wiced_ota_fw_upgrade_read_handler(conn_id, p_read_req);
        break;
    default:
        // Handle normal (non-OTA) read requests here
    }

    return status;
        }
```

Handler for *GATTS_REQ_TYPE_WRITE*:

```
wiced_bt_gatt_status_t ex01_ota_write_handler( wiced_bt_gatt_write_t *p_write_req,
uint16_t conn_id )
{
    wiced_bt_gatt_status_t status = WICED_BT_GATT_INVALID_HANDLE;

    switch(p_write_req->handle)
    {
    // If the indication is for an OTA service handle, pass it to the library to process
    case HANDLE_OTA_FW_UPGRADE_SERVICE:
    case HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_CONTROL_POINT:
    case HANDLE_OTA_FW_UPGRADE_CONTROL_POINT:
    case HANDLE_OTA_FW_UPGRADE_CLIENT_CONFIGURATION_DESCRIPTOR:
    case HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_DATA:
    case HANDLE_OTA_FW_UPGRADE_DATA:
    case HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_APP_INFO:
    case HANDLE_OTA_FW_UPGRADE_APP_INFO:
        wiced_set_debug_uart( WICED_ROUTE_DEBUG_NONE); // This is needed due to a timing
issue with the Windows peer client
        status = wiced_ota_fw_upgrade_write_handler(conn_id, p_write_req);
        break;
    default:
        // Handle normal (non-OTA) read requests here
    }

    return status;
}
```

Handler for GATTS_REQ_TYPE_CONF:

```
wiced_bt_gatt_status_t ex01_ota_indication_cfm_handler(uint16_t handle, uint16_t conn_id)
{
    wiced_bt_gatt_status_t status = WICED_BT_GATT_INVALID_HANDLE;

    switch(handle)
    {
    // If the indication is for an OTA service handle, pass it to the library to process
    case HANDLE_OTA_FW_UPGRADE_SERVICE:
    case HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_CONTROL_POINT:
    case HANDLE_OTA_FW_UPGRADE_CONTROL_POINT:
    case HANDLE_OTA_FW_UPGRADE_CLIENT_CONFIGURATION_DESCRIPTOR:
    case HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_DATA:
    case HANDLE_OTA_FW_UPGRADE_DATA:
    case HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_APP_INFO:
    case HANDLE_OTA_FW_UPGRADE_APP_INFO:
        status = wiced_ota_fw_upgrade_indication_cfm_handler(conn_id, handle);
        break;
    default:
       // Handle normal (non-OTA) indication confirmation requests here
    }

    return status;
}
```

Disabling of PUART

Note in the WRITE handler above, the debug UART is disabled before calling the OTA library write handler function. This is only required if the PUART is being used. For some reason, the PUART interferes with the OTA process when using the Windows peer app and causes the update to fail frequently.

Secure OTA

To use secure OTA firmware upgrade, we must create a key pair (public/private) and make a few changes in the firmware. The changes are shown in detail below.

Compile Macro

We will use a #define to conditionally compile the required changes. For example, we can use a flag called OTA_SECURE_FIRMWARE_UPGRADE to specify if we want to use Secure boot.

Key Generation

The changes to the firmware for secure OTA are relatively simple:

Include Keys

Header Files and Global Variables

BLE OTA Service (Secure)

Initialization

6B.4 Exercises

Exercise - 6B.1 WICED OTA Bootloader (Non-Secure)

Introduction

In this exercise, you will modify chapter 4A exercise 1 to add OTA firmware upgrade capability. Once OTA support is added, you will modify the project to control 2 LEDs instead of just one and you will upload the new firmware using OTA.

Project Creation

1. Copy Chapter 4A, Exercise 1 to a new folder (i.e. ch04c). Rename the project folder and project files to *ex01_ota*.
    a. Hint: Don't forget to update header file names in the two C files and don't forget to update the source file names in the makefile.
    b. Hint: Change the device name from *<inits>_LED* to *<inits>_ota* in the wiced_bt_cfg.c file and the <inits>_ota.c file.
    c. Hint: Many function names and variable names start with *key_led*. You can do a global search/replace to change these to *ex01_ota* if you want them to be consistent with the project name. Make sure you do this in the ex01_ota_db.h file too.
    d. Hint: Delete the .wic file since it is no longer a valid starting point for this project.
2. Create a Make Target for the project and verify that it still builds before proceeding.
3. Review the OTA Firmware section in this chapter and update the files as necessary:
    a. Add header file includes to the C files
    b. Add library to makefile.mk
    c. Add the OTA service to ex01_ota_db.c
    d. Update the GATT event handler functions
    e. Disable the PUART before OTA write

Testing

1. Build the project and program it to your kit.
2. Use CySmart to make sure the project functions as expected. Write values of 00, 01, 02, and 03 to the LED characteristic. The LED should only turn on for a value of 01.
    a. Hint: The Service with a single Characteristic is the LED Service and the Service with two Characteristics is the OTA Service.
3. Disconnect from the kit in CySmart.
4. Unplug the kit from your computer. This will ensure that OTA is used instead of regular programming to update the firmware.

5. Update the project so that the values written control the LEDs like this:

Characteristic Value	LED2	LED1
00	OFF	OFF
01	OFF	ON
02	ON	OFF
03	ON	ON

6. Copy the Make Target for the project and change "download" to "build". This will allow you to build the project without it trying to download to the kit.
7. Build the project.
8. Connect your kit directly to a power outlet using a USB charger.
9. Use OTA to update your kit. You can use either the Windows or the Android app.
    a. Hint: Don't forget to copy over the *.bin file from the build folder every time you re-build the project so that you are updating the latest firmware.
    b. Hint: If you need to find the Bluetooth Address of your device, use CySmart (either PC or Phone) to scan for it.
    c. Hint: If the OTA process fails on Windows, try resetting the kit and trying again. If that still fails, try using the Android version since it is more robust.
10. Once OTA upgrade is done, connect to the kit using CySmart and verify that the new firmware functionality is working.

Exercise - 6B.2 WICED OTA Bootloader (Secure)

Introduction

In this exercise, you will update the previous exercise to use Secure OTA firmware upgrade.

Project Creation

TODO: Write this

Testing

TODO: Write this