Chapter 9: Over the air firmware update

Updating firmware in the field is critical to many consumer devices. For devices that support Bluetooth®, it is natural for the user to expect updates to happen using the Bluetooth® link rather than requiring the device to be connected in some other way such as a USB connection.

This chapter discusses how firmware can be updated using the Bluetooth® connection. This process is called Over the air (OTA) firmware update.

Table of contents

Document conventions

Convention	Usage	Example

`Courier New`	Displays code and text commands	`CY_ISR_PROTO(MyISR);`
		`make build`

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 GUI commands, menu paths and selections, and icon names in procedures	Click the **Debugger** icon, and then click **Next**.

9.1 Introduction

The OTA firmware upgrade feature allows an external device to use the Bluetooth® link to transfer and install a newer firmware version to devices that support OTA. This chapter describes the functionality of the OTA firmware upgrade library used in various 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 while 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. This ensures that unknown firmware is not uploaded to your device.

9.2 Design and architecture

The OTA update process depends on how much flash the system has and how much is on-board. There are three possibilities:

1. If there is enough on-board flash to hold 2 complete application images, the flash memory is organized into two partitions (DS1 and DS2) for failsafe upgrade capability. During startup the boot code of the chip checks DS1 and if a valid image is found, it starts executing application code from there. If DS1 does not contain a valid image, the boot code checks DS2 and starts application code execution from there if a valid image is found. If neither partition is valid, the boot code enters download mode and waits for the code to be downloaded over the HCI UART.

   The addresses of the DS1 and DS2 partitions are programmed in a file with extension *.btp* in the platforms directory of the SDK. For example:

   > *mtb_shared/wiced_btsdk/dev-kit/baselib/20835B1/<version>/COMPONENT_20835B1/platforms/20835_SFLASH.btp*:

   ```
   ConfigDSLocation  = 0x4000
   ConfigDS2Location = 0x42000
   ```

   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.

   During an OTA upgrade the device performing the procedure (Downloader) pushes chunks of the new image to the device being upgraded. 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 as follows. For the non-secure download, the library calculates the CRC 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.

2. If there is not enough on-board flash to hold 2 complete application images, then a single on-board flash region (DS1) always contains the executable firmware. In this case, the update image is downloaded to a storage area on external flash, is validated, and is copied to the DS1 location. A small program in a small corner of the external flash (DS2) is used to copy this data while the active DS1 area is not in use.

3. If the device has no on-chip flash at all, then external flash is used with two regions (DS1 and DS2) just like the first case.

The default storage location for the second application image for a few different BTSDK devices is shown here:

Case	Devices	On-Chip Flash Size	Flash Organization	Default OTA Storage
1	20719 and 20720	1 MB	DS1 and DS2	on_chip_flash
2	20819 and 20820	256 kB	DS1	external_sflash on_chip_flash (for CYBT-213043-EVAL kit)
3	20706 and 20735	0	DS1 and DS2	external_sflash

9.3 OTA firmware

In the firmware, OTA requires the following:

9.3.1 OTA library

The first step is to include the *Firmware Upgrade (btsdk-ota)* library in your application using the library manager.

The application must have the following setting in the *makefile* to set up the proper build settings:

```
OTA_FW_UPGRADE?=1
```

This setting is what causes the OTA .bin file to be generated by the build. That's the file that you will send over the air to update the firmware.

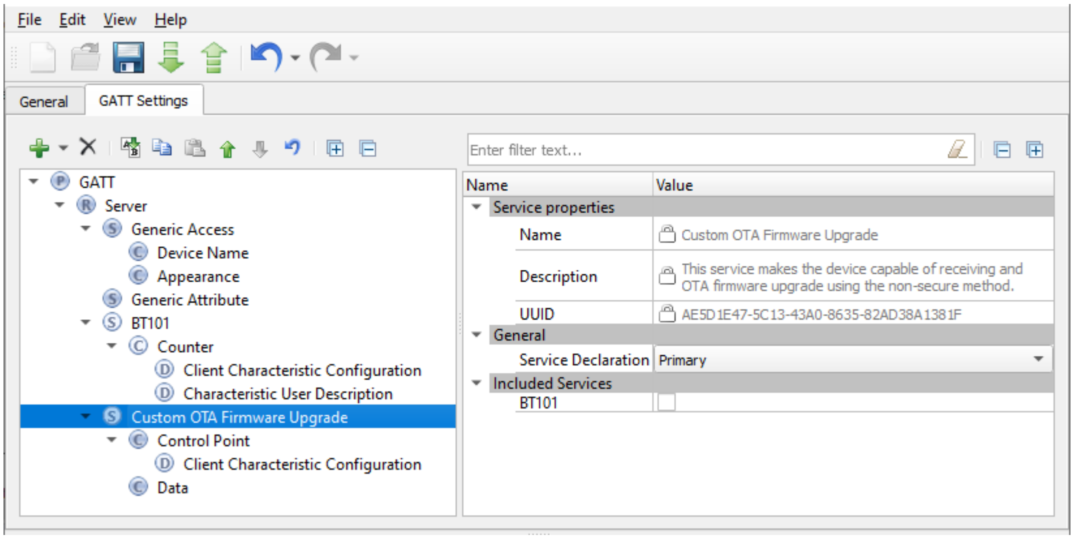
You must also add the following to the *makefile* below the existing `COMPONENTS+= bsp_design_modus` line:

```
#
# Components (middleware libraries)
#
ifeq ($(OTA_FW_UPGRADE),1)
COMPONENTS+=fw_upgrade_lib
endif
```

9.3.2 Bluetooth® LE OTA service (non-secure)

The GATT database must have a Primary Service for OTA. This is a custom Service that is defined by Infineon with two Characteristics, one of which has a CCCD. Luckily, Infineon has included this service in the list of choices, so everything will automatically be setup correctly.

From the Server, right click, select **Add Service**, and choose **Custom OTA Firmware Upgrade**. All of the Characteristics, Properties, and UUIDs should be left with the default values.



- GATT
  - Server
    - Generic Access
      - Device Name
      - Appearance
    - Generic Attribute
    - BT101
      - Counter
        - Client Characteristic Configuration
        - Characteristic User Description
    - Custom OTA Firmware Upgrade
      - Control Point
        - Client Characteristic Configuration
      - Data

Name	Value
Service properties	
Name	Custom OTA Firmware Upgrade
Description	This service makes the device capable of receiving and OTA firmware upgrade using the non-secure method.
UUID	AE5D1E47-5C13-43A0-8635-82AD38A1381F
General	
Service Declaration	Primary
Included Services	
BT101	□

9.3.3 Initialization

Include the following in your source file to get access to the OTA API:

```
#include "wiced_bt_ota_firmware_upgrade.h"
```

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, NULL);
```

9.3.4 GATT connect event

In the GATT connection status event, it is necessary to pass the connection status information to the OTA library by calling the following (`p_conn` is a pointer of type `wiced_bt_gatt_connection_status_t` to the event connection status).

```
wiced_ota_fw_upgrade_connection_status_event(p_conn);
```

This should be called on both a connection and disconnection.

9.3.5 GATT attribute request event

Several of the `GATT_ATTRIBUTE_REQUEST_EVT` events – namely `GATTS_REQ_TYPE_READ`, `GATTS_REQ_TYPE_WRITE`, `GATTS_REQ_TYPE_PREP_WRITE` and `GATTS_REQ_TYPE_CONF` – must call the appropriate OTA function when the GATT request is made to one of the OTA characteristics or descriptors. Note that OTA has its own library functions to handle these events, so they must be called for OTA events instead of the normal application code that is called for normal application functionality.

Note: *Although OTA is a GATT service, the handling of the data is not the same as a typical GATT service. Specifically, the data is not put into the GATT database. Rather, it is placed into flash so that it can be eventually used as the application firmware. That's the reason that the functions below must be called instead of the normal GATT handling that writes to and reads from the database.*

Namely the following functions must be called:

For `GATTS_REQ_TYPE_READ`:

```
    result = wiced_ota_fw_upgrade_read_handler(p_attr->conn_id, &(p_attr->data.read_req));
```

For `GATTS_REQ_TYPE_WRITE` or `GATTS_REQ_TYPE_PREP_WRITE`:

```
    result = wiced_ota_fw_upgrade_write_handler(p_attr->conn_id, &(p_attr->data.write_req));
```

For `GATTS_REQ_TYPE_CONF`:

```
    result = wiced_ota_fw_upgrade_indication_cfm_handler(p_attr->conn_id, p_attr->data.handle);
```

The OTA characteristics and descriptors in the switch statements that must trigger these calls are:

```
    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:
```

This functionality has already been added for you in the template. Review the `GATT_ATTRIBUTE_REQUEST_EVT` case to understand what was added.

In addition, code has been added to the read condition to handle reads of the device name and appearance characteristics.

9.3.6 Buffer pool sizes

The large buffer pool must be at least the max MTU size plus 12. Both are defined in *app_bt_cfg.c*. Verify that the buffer pools are large enough in your application.

9.3.7 Disabling of PUART

The PUART sometimes interferes with the OTA process when using the Windows peer app and may causes the update to fail. If this is the case, you can disable the debug UART in the GATT write handler before calling the OTA library write handler function. The code in the template contains that line of code but you can comment it out if you want to see additional debug messages during OTA.

9.4 Secure OTA

If you have a paired connection, all packets are encrypted, so why do we need Secure OTA? The answer is that you may want to verify that the firmware being sent to your device came from a reliable source and not from some hacker trying to change your device's behavior unintentionally. Secure OTA allows you to verify that the new firmware:

1. Is from that reliable source (i.e. the same source that initially programmed your device).
2. Is intended for that device (i.e. someone isn't trying to put firmware meant for a different application onto your device).
3. Is the same or newer major version than what is already on the device (i.e. someone isn't trying to put an older version of the firmware on your device).

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.

9.4.1 Bluetooth LE OTA service (secure)

In the Bluetooth Configurator, remove the "Custom OTA Firmware Upgrade" Service and replace it with the "Custom Secure OTA Firmware Upgrade" Service.

9.4.2 Key generation

Tools are provided in the BTSDK to create, sign, and verify random keys for Windows, Linux, and OSX. These tools are in the the *BTSDK Utilities (btsdk-utils)* library, which is typically included as a dependency of the BSP. If it is not, you can include it using the library manager. Once the library is included, executables can be found in:

mtb_shared/wiced_btsdk/tools/btsdk-utils/<version>/ecdsa256/bin/<Windows|Linux64|OSX>

The steps are (shown for Windows but others are similar):

1. Double-click on *ecdsa_genkey.exe* from Windows explorer to run the program. This will generate random keys. The files created are:
   ecdsa256_key.pri.bin
   ecdsa256_key.pub.bin
   ecdsa256_key_plus.pub.bin
   ecdsa256_pub.c

Note: If you re-run the program, it will overwrite any existing key files.

2. Copy the file *ecdsa256_pub.c* to the application directory. This gives the application access to the public key to include in the firmware by the build.
3. The OTA file will be signed using the private key once it is generated. That way, if the application is able to un-encrypt it using the public key, it knows that it must have been sent by a trusted source since no one else has the private key.

9.4.3 Application ID and Version Checking

You can add an application ID and firmware version to the *makefile* using the following three variables:

```
APP_VERSION_APP_ID
APP_VERSION_MAJOR
APP_VERSION_MINOR
```

When you update the firmware, you can increment the major version numbers. When the secure OTA process occurs, it will reject the firmware if its ID doesn't match or if the major version is lower than the version already on the device. That way, a nefarious actor cannot push an older (and potentially more vulnerable) version of the firmware onto your device or firmware that isn't intended for your device at all.

Note: *The variables all default to 0 so if you don't define them, the ID will be 0 and the version will be 0.0.*

Note: *The minor version is not checked for OTA purposes so a lower minor version will not prevent OTA from succeeding.*

9.4.4 Header files and global variables

Add the following header files to the main application C file:

```
#include "bt_types.h"
#include "p_256_multprecision.h"
#include "p_256_ecc_pp.h"
```

Add an external global variable declaration of type "Point" for the public key that is defined in *ecdsa256_pub.c*. For example:

```
extern Point    ecdsa256_public_key;
```

9.4.5 Initialization

In the firmware initialization section, change the first argument to the OTA init function from NULL to a pointer to a public key that was generated earlier. For example:

```
/* Initialize OTA (secure) */
wiced_ota_fw_upgrade_init(&ecdsa256_public_key, NULL, NULL);
```

9.4.6 Build the firmware and sign the OTA image

You can do a build/program step as usual. However, when you want to perform OTA upgrade, you must follow these steps to convert the output (**.bin*) file to a signed output file. Otherwise, the firmware on the kit will not perform the update.

1. Build the firmware as usual.
2. Open a Windows Explorer window and go to the *mtb_shared_wiced_btsdk/tools/btsdk-utils/<version>/ecdsa256/bin/Windows* directory that contains the keys you generated earlier.
3. Once the firmware is built, copy the bin file from the build directory for the application to the directory with the keys.
4. Drag and drop the *.bin file onto `ecdsa_sign.exe` to sign the image.

Note: Alternately, you can open a command terminal or power shell window (shift-right-click in the directory from Windows explorer) and enter the command such as this:

```
.\ecdsa_sign.exe .\app_CYW920835M2EVB-01.ota.bin
```

This will produce a file called *app_ CYW920835M2EVB-01.ota.bin.signed*. This operation uses the private key. Only someone with the private key can generate this signed image. Therefore, if you don't give out your private key, you are the only one who can supply new firmware to be loaded onto the device.

5. Load the signed file into the device using the preferred OTA tool (i.e. Windows, Android, or iOS).

9.5 Applications for loading new firmware

The ModusToolbox installation contains peer applications that can be used to transmit new firmware over Bluetooth LE: one for Windows, one for iOS and one for Android. The source code is available for all three and pre-compiled executables are provided for Android (*.apk*) and Windows (*.exe*) The Windows executable is provided for 32-bit (x86) and 64-bit (x64) architectures but it will only work on Windows 10 or later since Bluetooth LE is not natively supported in earlier versions.

The pre-compiled peer applications can be found in the *wiced_btsdk* after adding the *BTSDK Peer Apps for OTA library (btsdk-peer-apps-ota)* using the library manager. Once the library is included, the executables can be found at:

> *mtb_shared/wiced_btsdk/tools/btsdk-peer-apps-ota/<version>/Windows/WsOtaUpgrade/Release/x64/WsOtaUpgrade.exe*
>
> *mtb_shared/wiced_btsdk/tools/btsdk-peer-apps-ota/<version>/Android/LeOTAApp/app/build/outputs/apk/app-debug.apk*

9.5.1 Windows

In addition to running from the directory above, you can also access the Windows application from the command line interface if you add `WsOtaUprade` to the `CY_BT_APP_TOOLS` variable in the application's *makefile*. For example:

```
CY_BT_APP_TOOLS=BTSpy ClientControl WsOtaUpgrade
```

Once you do that, the command to open the tool from the command line interface is:

```
make open CY_OPEN_TYPE=WsOtaUpgrade
```

When you run *WsOtaUpgrade* using the make open command, it will automatically load the *.bin file from the application's build directory.

Note: *When you are doing secure OTA, you must replace the *.bin file in the build/<BSP>/<Build type> directory with the *.bin.signed file if you want to use the* `make open` *method.*

If you want to run the tool manually, you must specify the **.bin* (or **.bin.signed)* file for your application. It is located in the application's *build/<BSP>/<Build type>* directory. You can either drag/drop the file onto *WsOtaUpgrade* using Windows explorer, or you can specify it on the command line.

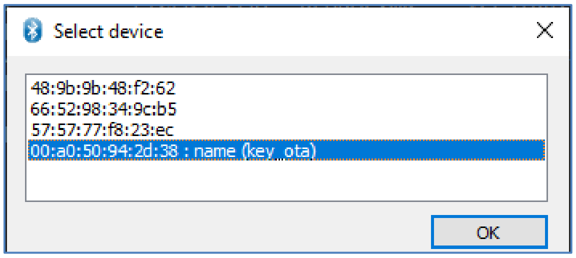
To summarize, here are three different ways to run *WsOtaUpgrade* on Windows:

1. Open a modus-shell command window and navigate to the directory containing the application. Enter the command `make open CY_OPEN_TYPE=WsOtaUpgrade`.
   a. The `CY_BT_APP_TOOLS` variable must include `WsOtaUpgrade` to use this method.
   b. For secure OTA, you must replace the **.bin* file with the **.bin.signed* file in the build directory before running the command.
2. Open a Windows explorer window and navigate to the directory containing the application. From an second Windows explorer window, navigate to the directory containing the **.bin* file (or **.bin.signed* for secure OTA). Drag and drop the **.bin* file for your application onto *WsOtaUpgrade.exe* in the explorer window.

3. Open a modus-shell command window and navigate to the directory containing the application. Then, run the application with the path to the **.bin* file (or **.bin.signed* for secure OTA) provided as an argument. For example:

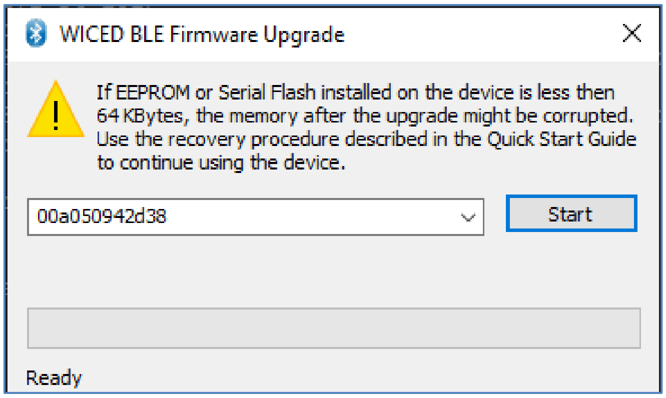
```
./WsOtaUpgrade.exe <path_to_app>/build/CYW920835M2EVB-01/Debug/app_CYW920835M2EVB-01.bin
```

No matter how you start the program, you will get a window that looks like the following. Select the device you want to update and click **OK**.



```
48:9b:9b:48:f2:62
66:52:98:34:9c:b5
57:57:77:f8:23:ec
00:a0:50:94:2d:38 : name (key_ota)
```

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

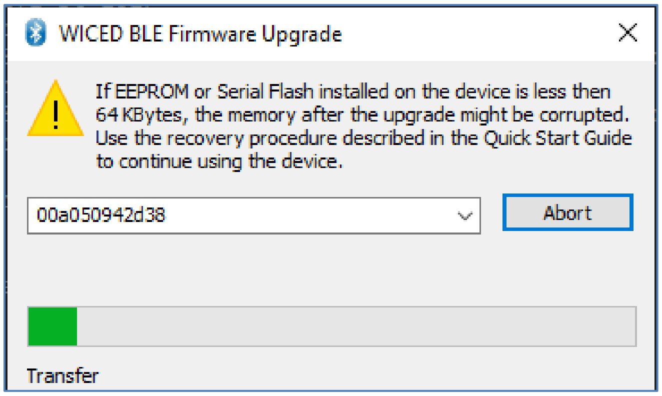


If EEPROM or Serial Flash installed on the device is less then 64 KBytes, the memory after the upgrade might be corrupted. Use the recovery procedure described in the Quick Start Guide to continue using the device.

00a050942d38

Ready

Once the update starts, you will see a progress bar. It may take up to a minute for the new firmware to be downloaded to the secondary flash location.

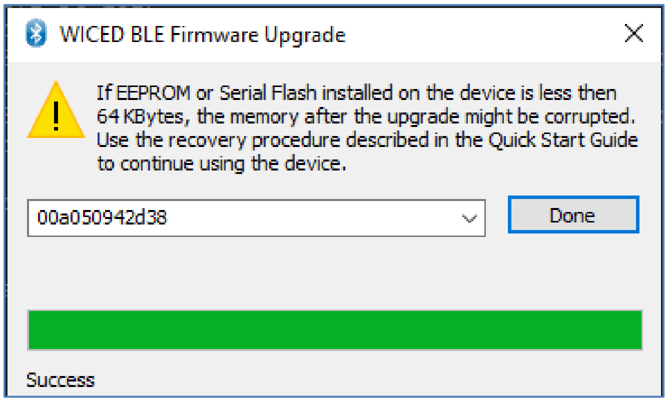


If EEPROM or Serial Flash installed on the device is less then 64 KBytes, the memory after the upgrade might be corrupted. Use the recovery procedure described in the Quick Start Guide to continue using the device.

00a050942d38

Transfer

Once it finishes, the window will show "Success" at the bottom if the update worked. Click **Done** to close the window.



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.

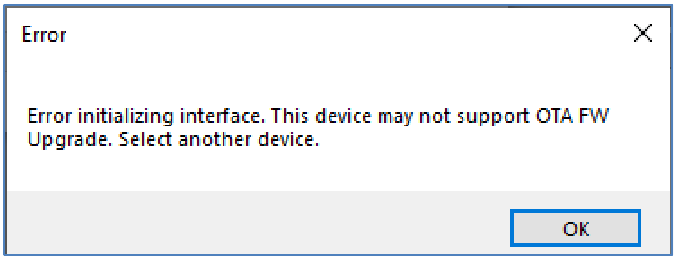
00a050942d38

Success

After OTA upgrade finishes, reset the kit and the updated firmware will be running.

9.5.1.1 Troubleshooting

You may see the following error message from the Windows OTA update application when you click the **Start** button:



Error initializing interface. This device may not support OTA FW Upgrade. Select another device.

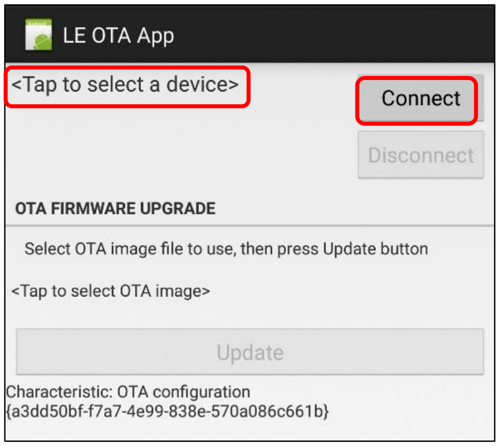
This message can be caused by several different issues. The following items should resolve the issue:

1. Verify that the OTA service name, characteristic names, fields properties, permissions and UUIDs all match what was shown earlier. Remember that the Windows application uses hard-coded UUID values to identify the service and characteristics so them must match exactly.
2. Verify that the code for supporting OTA is correct.
3. Quit the application, restart the application, reset the kit, and try again.
4. Quit the application, turn off the PC's Bluetooth® radio, re-enable the Bluetooth® radio, reset the kit, and try again.
5. Reboot the PC, reset the kit, and try again.

9.5.2 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 Debug 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:



LE OTA App

<Tap to select a device>

Connect

Disconnect

OTA FIRMWARE UPGRADE

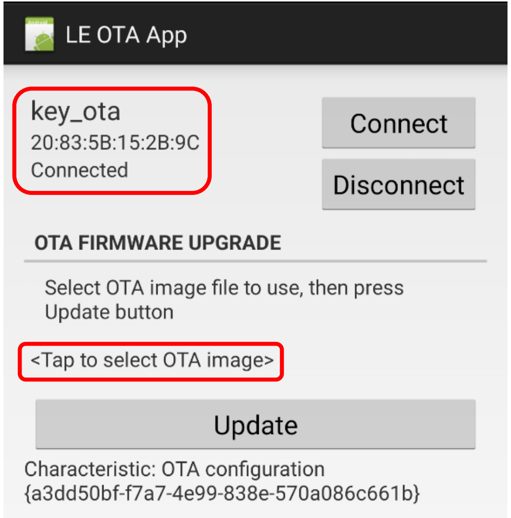
Select OTA image file to use, then press Update button

<Tap to select OTA image>

Update

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:



LE OTA App

key_ota
20:83:5B:15:2B:9C
Connected

Connect

Disconnect

OTA FIRMWARE UPGRADE

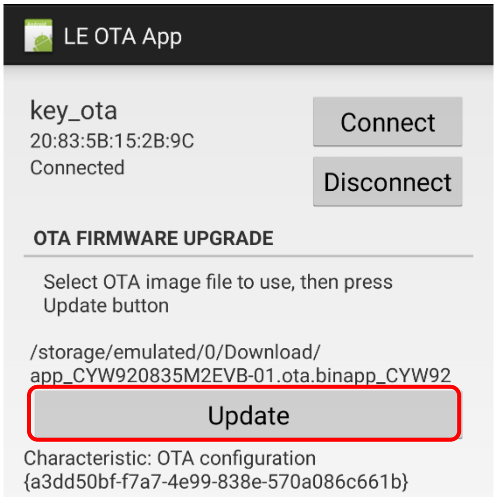
Select OTA image file to use, then press Update button

<Tap to select OTA image>

Update

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:



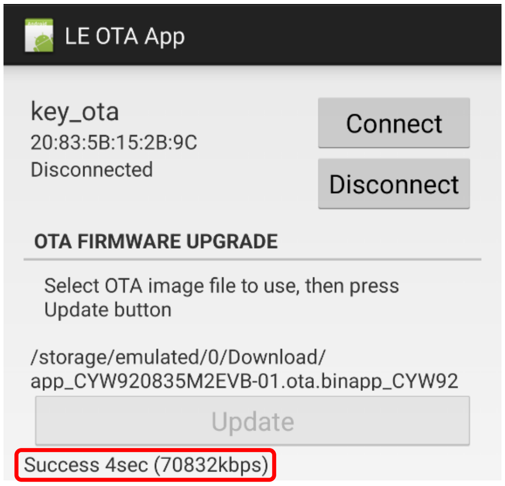
key_ota
20:83:5B:15:2B:9C
Connected

Select OTA image file to use, then press Update button

/storage/emulated/0/Download/
app_CYW920835M2EVB-01.ota.binapp_CYW92

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 (if it isn't already) and close the app.



key_ota
20:83:5B:15:2B:9C
Disconnected

Select OTA image file to use, then press Update button

/storage/emulated/0/Download/
app_CYW920835M2EVB-01.ota.binapp_CYW92

Success 4sec (70832kbps)

9.6 Exercises

Exercise 1: Bluetooth® LE OTA firmware upgrade (non-secure)

In this exercise, you will modify an application that counts button presses to add OTA firmware upgrade capability. Once OTA support is added, you will modify the application to decrement the count instead of incrementing and you will upload the new firmware using OTA.

Application Creation

□ 1. Create a new ModusToolbox™ application for the CYW920835M2EVB-01 BSP.

On the application template page, use the **Browse** button to start from the template in *Templates/ch09_ex01_ota_app*. Keep the application name the same.

□ 2. Use the library manager to add the ***Firmware Upgrade*** *(btsdk-ota)* and the ***BTSDK Peer Apps for OTA*** *(btsdk-peer-apps-ota)* libraries.

3. Open the makefile and:

□ a. Set `OTA_FW_UPGRADE?=1` in the app features.

□ b. Add the following below the existing `COMPONENTS+=` line:

```
ifeq ($(OTA_FW_UPGRADE),1)
COMPONENTS+=fw_upgrade_lib
endif
```

□ c. Add `WsOtaUpgrade` to the `CY_BT_APP_TOOLS` line:

```
CY_BT_APP_TOOLS=BTSpy ClientControl WsOtaUpgrade
```

4. Use the Bluetooth® Configurator to:

□ a. Change the device name to **<inits>_ota.**

□ b. Add the **Custom OTA Firmware Upgrade Service** to the Server.

□ c. Save changes and close the configurator.

5. Edit *app.c*:

□ a. Add `#include "wiced_bt_ota_firmware_upgrade.h"`

□ b. Add `wiced_ota_fw_upgrade_init(NULL, NULL, NULL);` after the GATT database is initialized.

□ c. Add `wiced_ota_fw_upgrade_connection_status_event(p_conn);` in the GATT connection status event (for both connect and disconnect events).

□ d. The GATT Attribute Request Event handler code is provided in the template. Review it to see how OTA GATT attribute request events are passed to the library.

Note: Search for `HANDLE_OTA` in app.c.

Note: Whenever the handle matches one of the OTA cases, the normal GATT database processing is NOT done.

Testing

□ 1. Build the application and program it to your kit.

2. Look at the PUART window during initialization.

Note: Write down your kit's Bluetooth Device Address. You may need this to identify the correct device when you perform OTA upgrade.

□ 3. Use AIROC™ Connect to make sure the application functions as expected.

Press the button and notice that the counter characteristic increases each time the button is pressed.

Note: You will see two Services. The Service with a Read/Write Characteristic and a Read/Notify Characteristic is the Counter Service. The Service with a Write/Indicate Characteristic and a Write Characteristics is the OTA Service.

□ 4. Disconnect from the kit in AIROC™ Connect.

□ 5. Remove the device from the paired devices on your phone.

□ 6. This is necessary because the kit will lose bonding information once the firmware is reloaded.

□ 7. Unplug the kit from your computer.

This will ensure that OTA is used instead of regular programming to update the firmware.

□ 8. Update the application so that each button press decrements the value instead of incrementing it.

Build the application without programming.

Note: In the Quick Panel, use the link "Build ch9_ex01_ota Application".

□ 9. Connect your kit directly to a power outlet using a USB charger.

□ 10. Use OTA to update your kit using either the Windows or Android application.

Note: The Windows OTA application only works on Windows 10 or later since earlier versions of Windows do not support Bluetooth LE.

Note: If the OTA process fails on Windows, try resetting the kit and trying again. If that still fails, try using the Android version.

□ 11. Once OTA upgrade is done, reset the kit and then use AIROC™ Connect to verify that the updated firmware functionality is working. Namely, pressing the button should decrement the Counter value instead of incrementing it.

Exercise 2: Bluetooth® LE OTA firmware upgrade (secure)

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

□ 1. Create a new application using the completed *ch09_ex01_ota* exercise as the starter application. Name the new application ***ch09_ex02_ota_sec***.

□ 2. In the m*akefile*, add variables for the application ID and version:

```
APP_VERSION_APP_ID =  1
APP_VERSION_MAJOR  =  5
APP_VERSION_MINOR  =  2
```

3. Use the Bluetooth® Configurator to:

□ a. Change the name to **<inits>_*ota_sec***.

□ b. Delete the **Custom OTA Firmware Upgrade Service** and add **the Custom OTA Secure Firmware Upgrade Service**.

□ c. Save changes and close the configurator.

□ 4. Generate public/private keys to use in encrypting and unencrypting the firmware image.

Note: See section 9.4.2 for a reminder on how to generate keys.

□ 5. Once generated, copy *ecdsa256_pub.c* to the top-level application directory (the same directory as *app.c*).

□ 6. Add the additional required includes to *app.c*:

```
#include "bt_types.h"
#include "p_256_multprecision.h"
#include "p_256_ecc_pp.h"
```

□ 7. Add an external global variable declaration to *app.c* of type "Point" for the public key that is defined in *ecdsa256_pub.c*:

```
extern Point    ecdsa256_public_key;
```

□ 8. In the firmware initialization section, change the first argument to the OTA `init` function from NULL to a pointer to a public key that was generated earlier. For example:

```
/* Initialize OTA (secure) */
wiced_ota_fw_upgrade_init(&ecdsa256_public_key, NULL, NULL);
```

Testing

□ 1. Build the application and program it to your kit.

□ 2. Use AIROC™ Connect to make sure the application function as expected.

Note: Remember that the counter should be counting down since that's how we left it in the previous exercise.

□ 3. Disconnect from the kit in AIROC™ Connect.

□ 4. Remove the device from the paired devices on your phone.

This is necessary because the kit will lose bonding information once the firmware is reloaded.

□ 5. Unplug the kit from your computer.

This will ensure that OTA is used instead of regular programming to update the firmware.

□ 6. Undo the change from the previous exercise to count up again instead of down on each button press.

□ 7. Edit the *makefile* to increment the major version from 5 to 6.

□ 8. Build the application.

□ 9. Sign the image.

Note: See section 9.4.6 for a reminder on how to sign the image.

□ 10. Connect your kit directly to a power outlet using a USB charger.

□ 11. Use OTA to update your kit. You can use either the Windows or the Android app.

*Note: Remember that instead of using the *.bin file from the build directory, you must use the *.bin.signed file that you signed with the application's private key.*

*Note: If you are using the `make open` command from the CLI, you must replace the *.bin file in the build directory with the *.bin.signed file. This is necessary since the make command automatically loads the file with the extension *.bin.*

Note: Don't forget that every time you re-build the application you must sign the bin file and copy it into the build directory.

Note: 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. A failure may also mean that you didn't properly sign the image.

□ 12. Once OTA upgrade is done, reset the kit and connect to it using AIROC™ Connect.

Verify that the new firmware functionality is working. The counter should again be counting up.

□ 13. Try doing OTA with the unsigned image.

Notice that it will fail after loading the image. The prior firmware will be retained in this case.

□ 14. Edit the *makefile* to decrement the major version from 6 back to 5.

□ 15. Clean and then build the application, sign the image, and attempt to do OTA.

Notice that it will fail after loading the image because the firmware on the device is version 5.3 and you are attempting to load version 5.2. The prior firmware will be retained in this case.

Note: The clean operation is necessary because we only modified the makefile, not any source files. Without a clean first, the build would not re-generate the hex or bin files.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Published by
Infineon Technologies AG
81726 Munich, Germany

IMPORTANT NOTICE
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”) .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.