RSL10 Firmware Over-The-Air User's Guide

M-20860-002
July 2019

© SCILLC, 2019
Previous Edition © 2018
"All Rights Reserved"

ON Semiconductor®

Table of Contents

Page

1. Introduction . . . 3
   - 1.1 Purpose . . . 3
   - 1.2 Intended Audience . . . 3
   - 1.3 Conventions . . . 3
   - 1.4 Further Reading . . . 3
2. Overview . . . 4
   - 2.1 Prerequisites . . . 4
   - 2.2 Overview . . . 4
3. The FOTA Firmware . . . 6
   - 3.1 FOTA Partitioning . . . 6
   - 3.2 Firmware Startup . . . 7
   - 3.3 Application Only Update . . . 7
   - 3.4 Application + FOTA Bluetooth Low Energy Stack Update . . . 7
4. Performing Your First FOTA Update . . . 9
   - 4.1 Overview . . . 9
   - 4.2 Generating the FOTA Firmware Image . . . 9
   - 4.3 Setting Up the RSL10 Bootloader and Loading a Firmware Image Using UART . . . 11
   - 4.4 Performing a FOTA Update Using the FOTA.Console PC Tool . . . 12
5. FOTA Image . . . 14
   - 5.1 Overview . . . 14
   - 5.2 mkfotaimg.py . . . 14
   - 5.3 Sub-Image Format . . . 15
   - 5.4 Signing the FOTA Image . . . 17
   - 5.5 FOTA Signature Validation . . . 17
     - 5.5.1 Performing Signature Validation . . . 17
     - 5.5.2 FOTA Image Checking with the DFU Component . . . 18
     - 5.5.3 More About Digital Signature Validation . . . 19
6. The DFU . . . 20
   - 6.1 DFU Component . . . 20
   - 6.2 Update Sequence . . . 20
   - 6.3 FOTA Stack Source Code . . . 20
   - 6.4 DFU Bluetooth Low Energy Service . . . 23
   - 6.5 DFU Service Characteristics . . . 23
   - 6.6 DFU Protocol . . . 24
7. The Fota.Console Command Line Tool . . . 26
   - 7.1 Fota.Console . . . 26
8. Integrating FOTA Into Your Application . . . 27
   - 8.1 Modifying the Application . . . 27
   - 8.2 Performing a FOTA Update . . . 30
9. RSL10 FOTA Mobile Application . . . 36
   - 9.1 RSL10 FOTA Application . . . 36

CHAPTER 1

Introduction

1.1 Purpose

This manual describes Firmware Over-The-Air (FOTA) with RSL10. It provides the prerequisites and instructions necessary to develop FOTA-ready firmware applications and to perform FOTA updates in the field.

1.2 Intended Audience

This manual is for firmware developers who are designing and implementing RSL10 applications with FOTA capability.

1.3 Conventions

The following conventions are used in this manual to signify particular types of information:

`monospace font`
: Macros, functions, defines and addresses.

italics
: File and path names, or any portion of them.

`<angle brackets>`
: Optional parameters and placeholders for specific information. To use an optional parameter or replace a placeholder, specify the information within the brackets; do not include the brackets themselves.

1.4 Further Reading

For more information about RSL10, refer to the following documents:

- *RSL10 Hardware Reference*
- *RSL10 Firmware Reference*
- *Getting Started with RSL10*
- RSL10 *Sample Code User's Guide*
- RSL10 *Evaluation and Development Board Manual*
- *RSL10 Datasheet*

CHAPTER 2

Overview

2.1 Prerequisites

- RSL10 SDK CMSIS-Pack version 2.4.0 or later (available at www.onsemi.com)
- RSL10 USB Dongle
- RSL10 Evaluation and Development Board (EVB)
- RSL10 Bluetooth Low Energy Explorer (available at www.onsemi.com)
- Python v2.7 or later:
  - Install package *ecdsa* version 0.13 or later.
  - Install package *pyserial* version 3.2 or later.
  - Make sure Python is added to the system path.
  - You can install the above packages using PyPI (for example, `python -m pip install ecdsa`).

- *Fota.Console* tool (available in the utility *apps zip* package), which has the following dependencies:
  - Microsoft™ .Net Framework version 4.6 or later
  - SiliconLabs™ VCP driver version 6.7.3 or later (available in the RSL10 Bluetooth Low Energy Explorer installation, *ON Semiconductor/Bluetooth Low Energy Explorer/Driver/CP210x_Windows_Drivers.zip*)

- Or the mobile application, as an alternative to the *Fota.Console* tool (see Chapter 9, "RSL10 FOTA Mobile Application" on page 36).

2.2 Overview

The RSL10 software ecosystem includes a set of tools that allows Firmware Over-The-Air (FOTA) updates from a PC with the RSL10 USB dongle to a remote RSL10 device over a Bluetooth Low Energy wireless link. On the PC side, a Python utility (*mkfotaimg.py*) generates FOTA-compatible firmware images and a PC command line tool (*FOTA.Console.exe*) transfers the images to the remote device. The PC command line tool uses the RSL10 USB Dongle as a central device to scan, connect and transmit the firmware image. The remote RSL10 device firmware side consists of a *bootloader* program, sample code, and a FOTA Bluetooth Low Energy stack that contains the Device Firmware Update (DFU) Bluetooth Low Energy component. Figure 1, below, illustrates a typical FOTA update setup from the PC point of view:

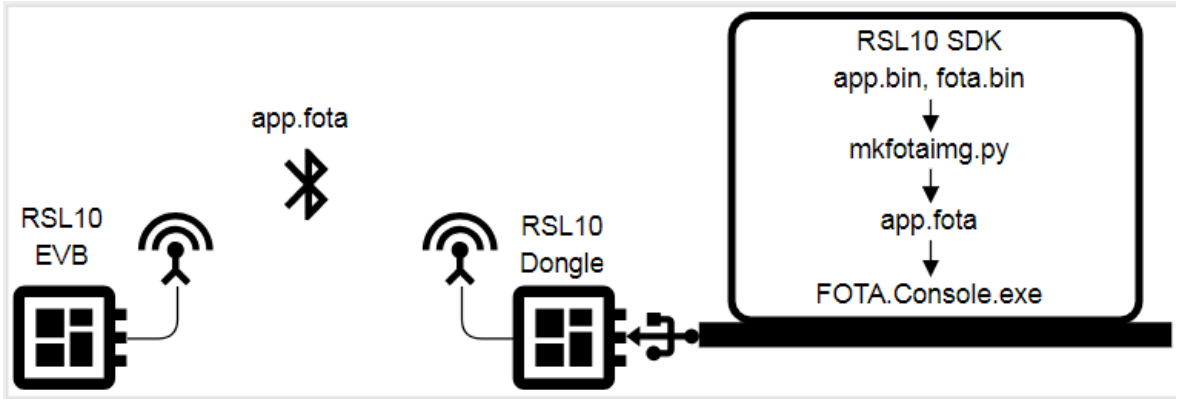


Figure 1. FOTA Update Setup

The following chapters provide details about the tools, protocols and firmware required to perform FOTA updates using RSL10. We present details about the FOTA firmware images and tools, walk you step-by-step through your first firmware update using a pre-configured sample application (*ble_peripheral_server_hrp_fota*), and show how to modify an existing application to support FOTA updates.

CHAPTER 3

The FOTA Firmware

3.1 FOTA Partitioning

A device capable of receiving FOTA updates contains firmware composed of three parts, as illustrated in Figure 2:

1. Bootloader
2. FOTA Bluetooth Low Energy stack including DFU component (*fota.bin* sub-image)
3. User application (*app.bin* sub-image)

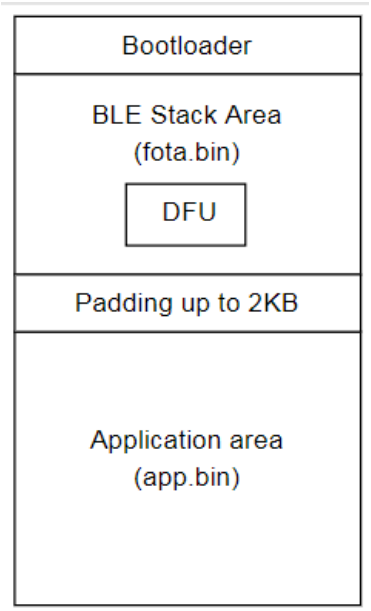


Figure 2. Memory Map

Each part depends on the previous one, except the *bootloader*, which is standalone. The FOTA Bluetooth Low Energy stack depends on the *bootloader*. The user application depends on the FOTA Bluetooth Low Energy stack, and therefore, also depends on the *bootloader*. In terms of size, the bootloader reserves 8KB of memory. The Bluetooth Low Energy stack, which contains the Device Firmware Update (DFU) component and the padding between the sub-images, allocates about 154KB. The remainder of the RSL10 main flash memory is available for the application (380KB - 154KB - 8KB approximately equal to 218KB). The boundary between the Bluetooth Low Energy stack and application areas can be dynamic, so it is possible to increase the Bluetooth Low Energy stack size at the expense of the application size, and vice versa.

When building a FOTA-compatible RSL10 sample application, you may notice that the *libfota.a* library is linked. This library functions as a stub object, and implements the Bluetooth Low Energy library functions as pointers to the real implementation, located in the *fota.bin* area. This makes the link between the application and the FOTA Bluetooth Low Energy stack particularly strong, because the application code calls functions from the Bluetooth Low Energy stack directly. Therefore, an application can only run together with the specific FOTA Bluetooth Low Energy stack revision that is used when building that application. For this reason, it is not possible to update the Bluetooth Low Energy stack without updating the application. On the other hand, it is possible to perform a FOTA update of the user application only.

Two types of FOTA updates are possible:

- Application only update
- Application + FOTA Bluetooth Low Energy stack update

The DFU component, an application embedded in the Bluetooth Low Energy stack area, implements a DFU Bluetooth Low Energy custom service for FOTA updates. It contains characteristics that allow a client device to gather information from the installed firmware (such as version numbers and IDs) and download the firmware image. As the DFU is embedded into the FOTA Bluetooth Low Energy stack area, FOTA updates are possible even when no valid user application is available in the device. More details about this component are provided in Chapter 6, "The DFU" on page 20.

3.2 Firmware Startup

Upon boot-up, the *bootloader* checks whether there is a valid user application or Bluetooth Low Energy stack programmed. The sequence of operations is as follows:

1. If there is a valid user application, start it.
2. If no valid application is found, start the FOTA Bluetooth Low Energy stack DFU component (so the device can receive FOTA updates).
3. If no valid FOTA Bluetooth Low Energy stack is found, start the *bootloader* updater (in this case, the device can only receive firmware updates over UART/USB).

The FOTA Bluetooth Low Energy stack DFU component can be activated from the user application at any time, through a call to `Sys_Fota_StartDfu()`. More details about this are provided in later chapters.

3.3 Application Only Update

To update the application, the currently installed user application needs to start the DFU component from the Bluetooth Low Energy stack. The DFU component then starts the FOTA process and receives the new application image. The application image embeds the Build ID, calculated by the GNU linker over all symbols of *fota.bin* (see Table 4 on page 23 for more about the FOTA stack Build ID). If this information does not match the installed Bluetooth Low Energy stack revision, the FOTA update is aborted. At this point the currently installed application is not destroyed.

If the Bluetooth Low Energy stack revision is compatible, the currently installed application is erased and the new image is programmed as the data comes in. When the whole image is programmed successfully, the DFU component marks the new application as valid and performs a restart. If the application update is aborted for any reason (power loss, for example), this causes the application area to contain an invalid image. The update process can be restarted from the beginning in this case, as the device still contains a valid Bluetooth Low Energy stack with the DFU component.

3.4 Application + FOTA Bluetooth Low Energy Stack Update

This type of update is performed in multiple steps:

1. Start the DFU component.
2. When the FOTA process receives a Bluetooth Low Energy stack image instead of an application image, no revision check is performed, and the downloaded image is saved in the application area. The FOTA process executes code from the Bluetooth Low Energy stack area, so it is not possible to directly replace the Bluetooth Low Energy stack.
3. After completely programming the new Bluetooth Low Energy stack image into the download/application area, the DFU component performs a reset and thereby activates the *bootloader*.

4. The *bootloader* detects a valid Bluetooth Low Energy stack image in the download area and copies it to the Bluetooth Low Energy stack area. This is now possible because the previous Bluetooth Low Energy stack is no longer needed. After successfully copying the new Bluetooth Low Energy stack, the bootloader invalidates the Bluetooth Low Energy stack in the download/application area and proceeds with the firmware startup.
5. As with a usual *bootloader* startup, the DFU component of the new Bluetooth Low Energy stack detects no valid application, and therefore starts the FOTA DFU again to receive the new application image.
6. From this point on, the process is the same as it is with an application only upgrade.

Because the application area is used to temporarily hold the Bluetooth Low Energy stack image, the size of the Bluetooth Low Energy stack cannot exceed half of the size of the area between the end of the *bootloader* area and the end of the main flash area. This limits the Bluetooth Low Energy stack size to a maximum of (380-8)/2 = 186 KB.

If the *bootloader* copy operation of the Bluetooth Low Energy stack image from the application area to the Bluetooth Low Energy stack area is aborted, at next startup the *bootloader* detects the still-valid Bluetooth Low Energy stack image in the application area and repeats the copy. This makes it possible to recover from power loss during the update process.

CHAPTER 4

Performing Your First FOTA Update

4.1 Overview

Before we dive into all details regarding the FOTA tools, firmware images, and protocol specifications, this chapter walks you step-by-step through the process of performing your first FOTA update. The goal is to provide you a basic hands-on understanding of the RSL10 FOTA update process, and to ensure that your hardware and software are correctly setup. This chapter shows how to:

1. Generate a FOTA firmware image using the preconfigured *ble_peripheral_server_hrp_fota* sample application. This application is similar to the Heart Rate Bluetooth Low Energy application (*ble_peripheral_server_hrp*) with added features to support FOTA updates.
2. Set up the RSL10 *bootloader* and load a firmware image using UART.
3. Perform a FOTA update using the *FOTA.Console* PC tool. Alternatively, a FOTA update can be performed using the mobile application. See Chapter 9, “RSL10 FOTA Mobile Application” on page 36.

This tutorial assumes that you have installed the prerequisites and have the required version of the RSL10 CMSIS-Pack installed (see *Getting Started with RSL10* for instructions on how to import a CMSIS-Pack).

4.2 Generating the FOTA Firmware Image

1. Open the Examples tab in the Pack Manager perspective to see example projects, included in the RSL10 CMSIS-Pack.
2. Find the *ble_peripheral_sever_hrp_fota* example project and click the **Copy** button to import it into your workspace. (See Figure 3, below.)

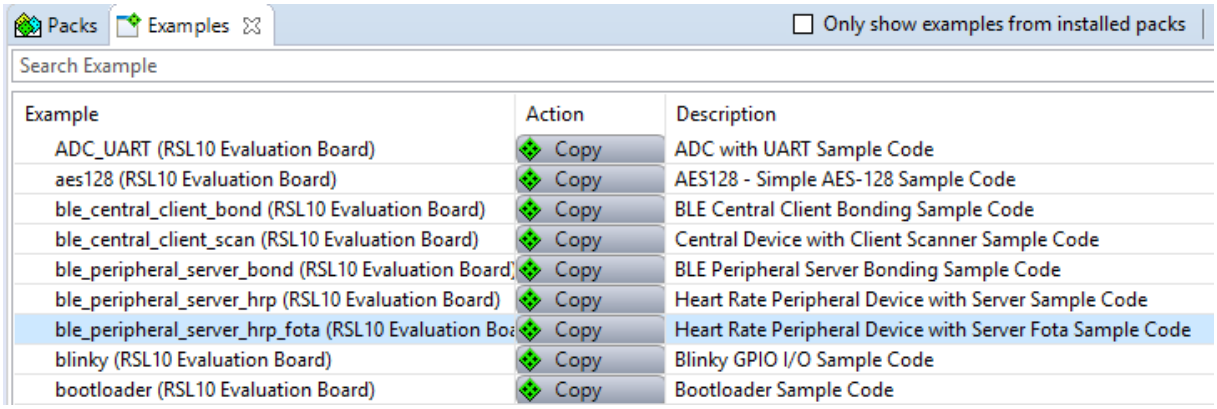


Figure 3. Importing the FOTA Sample Project

3. The **C/C++** perspective opens and displays your newly copied project. In the **Project Explorer** panel, you can expand your project folder and explore the files inside your project, as seen in Figure 4 on page 10. On the right side, the *ble_peripheral_server_hrp_fota.rteconfig* file displays the selected software components, including the new component named **Fota**. If you expand **RTE** > **Device** > **RSL10**, you can find the FOTA library (*libfota.a*), the FOTA Bluetooth Low Energy Stack binary file (*fota.bin*), and the Python tool *mkfotaimg.py*. These files are automatically added to your sample project once the **Fota** component is selected.

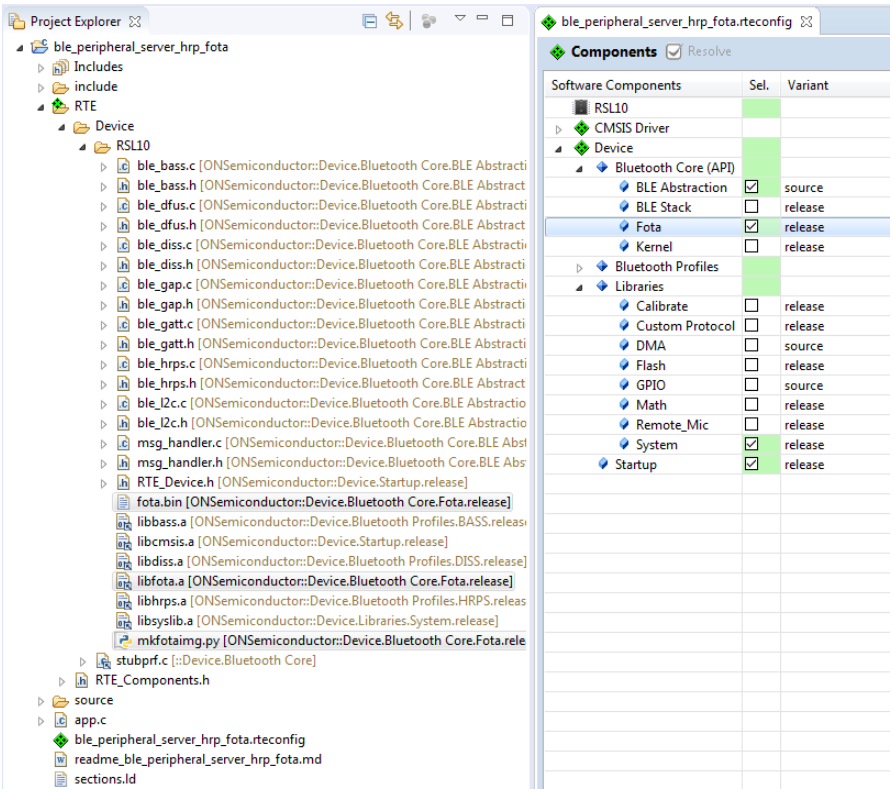


Figure 4. Files in FOTA Sample Project

4. Build the *ble_peripheral_server_hrp_fota* project. After a successful build, the *ble_peripheral_server_hrp_fota.fota* image is generated under the *Debug* folder, as seen below in Figure 5:

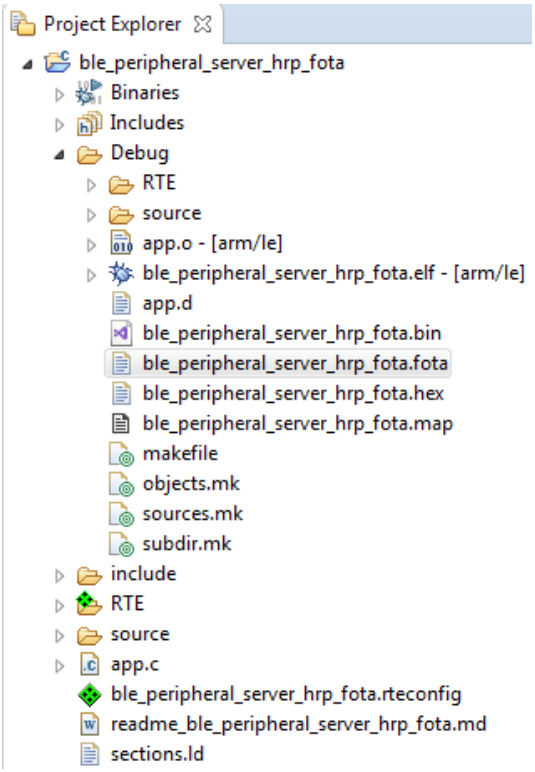


Figure 5. .fota Image in Debug Folder

The generated *.fota* file contains both the Fota Bluetooth Low Energy stack sub-image (*fota.bin*) and the application sub-image. This file can be used by the bootloader UART PC updater tool to perform a UART firmware update, or by the *FOTA.Console.exe* PC tool to perform a FOTA update.

4.3 Setting Up the RSL10 Bootloader and Loading a Firmware Image Using UART

1. If you are running a FOTA update for the first time, your RSL10 EVB does not have the RSL10 *bootloader* flashed into it.
2. Import and build the *bootloader* sample application, available in the **Examples** tab.
3. Start a debug session to flashload the *bootloader* application into the RSL10 Evaluation and Development Board (or alternatively use the external flashloader to load the *bootloader.hex* file).
4. Reset your board. After reset, you can see that the LED of the RSL10 Evaluation and Development Board is continuously on, indicating that *bootloader* has not found a valid user application or a valid FOTA Bluetooth Low Energy stack, and therefore has activated its updater mode. In this mode, the *bootloader* is waiting for commands over UART to download a new user application firmware image.
5. Use the Windows Device Manager to find out the COM port number assigned to your RSL10 EVB, identified by the JLink CDC UART Port (COM4, as seen in Figure 6. below):



Figure 6. RSL10 EVB COM Port Number

6. Using a command prompt, navigate to the *bootloader/scripts* folder. You can see the *updater.py* tool together with additional *.dll* dependencies.
7. Invoke the *updater.py* tool to load the *ble_peripheral_server_hrp_fota.fota* image over UART with the following command:

```
> python updater.py COM4 ble_peripheral_server_hrp_fota.fota
Image      : PSHRP  ver=2.4.0 / FOTA   ver=2.4.0
Bootloader : BOOTL* ver=2.0.1
********************************************************************************
```

You can expect similar output. For each flash sector transferred and written to the flash memory, an asterisk (*****) symbol is printed on the screen.

If you find errors executing this step, make sure you have the required versions of Python and the *pyserial* package.

8. You can use the RSL10 Bluetooth Low Energy Explorer or another application to confirm that your new firmware is running and advertising under the name `Peripheral_HRP_FOTA`, as shown in Figure 7 on page 12:

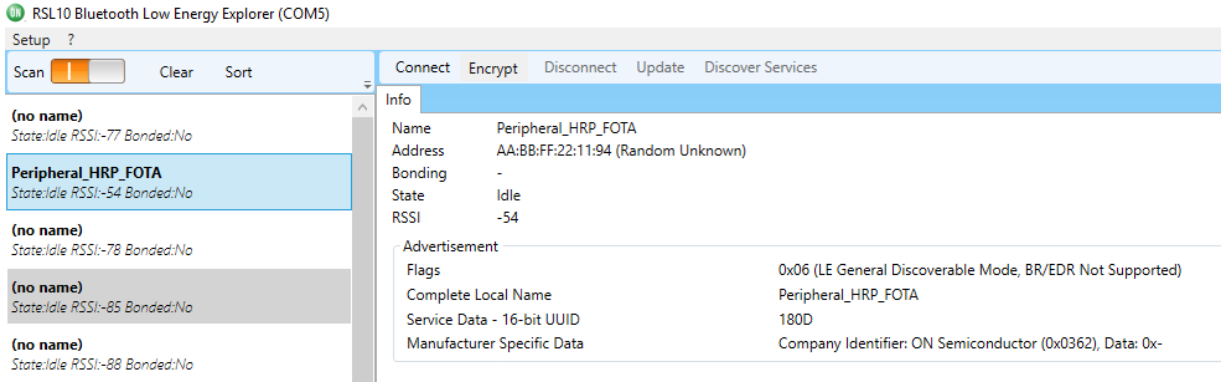


Figure 7. FIrmware Running

Your device is now ready to perform a FOTA update, as it contains all the required components: the *bootloader* program and the Fota Bluetooth Low Energy stack.

4.4 Performing a FOTA Update Using the FOTA.Console PC Tool

1. Make sure no other process is using the RSL10 Dongle (e.g.: close RSL10 Bluetooth Low Energy Explorer).
2. Use the Windows Device Manager to find the COM port number assigned to your RSL10 Dongle, identified by the Silicon Labs USB to UART Bridge (COM5, in the example below in Figure 8):

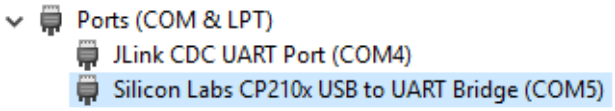


Figure 8. RSL10 Dongle COM Port Number

3. The *Fota.Console* tool is available in the RSL10 Utility Apps zip package. Make sure to extract this package and use a command prompt tool to navigate to the *Fota.Console* folder.
4. Run the *Fota.Console.exe* tool to start a FOTA update using the following command:

```
Fota.Console.exe /COM=COM5 /IN=ble_peripheral_server_hrp_fota.fota /
   NAME=Peripheral_HRP_FOTA
```

Make sure to copy the *.fota* image file into the same folder as the *Fota.Console* tool, or specify the full path to this file in the command above.

5. The *Fota.Console* tool searches, connects and transmits the firmware image to the RSL10 Evaluation and Development Board. If everything goes as expected, you have successfully performed your first RSL10 FOTA update. You can expect output similar to this:

```
Fota.Console.exe /COM=COM5 /IN=ble_peripheral_server_hrp_fota.fota /
   NAME=Peripheral_HRP_FOTA
Fota.Console v1.0.5.0
--------------------------------------------------------------------------------

Searching for peripheral ...
Found peripheral (Address:AA:BB:FF:22:11:94 RSSI:-57dBm Name:Peripheral_HRP_FOTA)
Connecting
------------------------ Peripheral_HRP_FOTA ------------------------
FOTA stack version:  FOTA   2.4.0
Application version: PSHRP  2.4.0
Device ID:           None
------------------------ ble_peripheral_server_hrp_fota.fota ------------------------
FOTA stack version:  FOTA   2.4.0
Application version: PSHRP  2.4.0
Device ID:     None
Service UUID:  b2152466-d600-11e8-9f8b-f2801f1b9fd1
FOTA build ID: 04 00 00 00 14 00 00 00 03 00 00 00 47 4E 55 00 5F 20 56 EC 3E E6 11 BD 98
   4E F2 5E 21 48 E7 97

Establish

RebootToBootloader

Establish

UpdateAppImage
 <-- 100.0% 14.8kB/s UpdateAppImage
Finished
Completed with status Success
```

If you have trouble executing this step, make sure you have installed the required versions of the drivers and *.Net* framework.

CHAPTER 5

FOTA Image

5.1 Overview

The FOTA Image (*.fota* file) consists of two sub-images, with padding to a multiple of 2048 bytes in-between so that the start of the second image lies on an RSL10 flash sector boundary. The first sub-image is the FOTA Bluetooth Low Energy Stack (*fota.bin*) and the second one is the user application (*app.bin*). The Python utility *mkfotaimg.py* generates the FOTA image, as illustrated below in Figure 9:

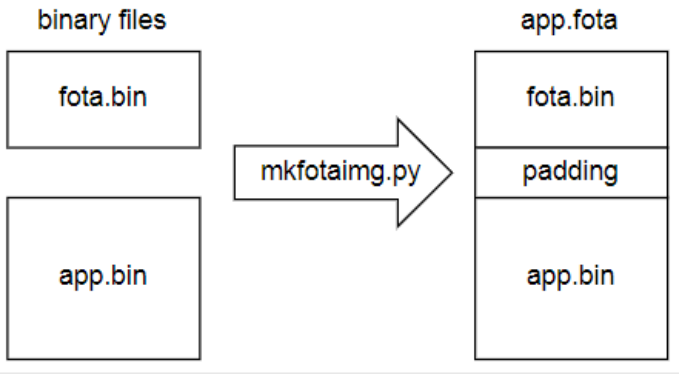


Figure 9. Image Format

The *.fota* file can then be used as input to the *FOTA.Console.exe* tool for a FOTA update, or as input to the *updater.py* tool to perform a UART firmware update via the *bootloader*. For more information about the *bootloader*, see the *RSL10 Bootloader User's Guide*.

5.2 MKFOTAIMG.PY

The *mkfotaimg.py* tool takes two positional arguments as inputs: the FOTA Bluetooth Low Energy stack sub-image and the user application sub-image. Its usage and optional arguments are shown below.

Usage:

```
> mkfotaimg.py [-h] [--version] [-d UUID] [-s KEY-PEM] [-i UUID][-n NAME] [-o OUT-IMG]
    FOTA-IMG APP-IMG
```

Arguments:

Table 1. Positional Arguments

Argument	Meaning
FOTA-IMG	FOTA stack sub-image containing the Bluetooth Low Energy stack and the DFU component (.bin file)
APP-IMG	application sub-image (.bin file)

Table 2. Optional Arguments

Argument	Meaning
-h, --help	show help message and exit
--version	show program's version number and exit
-d UUID, --devid UUID	device UUID to embed in the image (default: no ID)
-s KEY-PEM, --sign KEY-PEM	name of signing key PEM file (default: no signing)
-i UUID, --srvid UUID	advertised UUID to embed in the image (default: DFU service UUID)
-n NAME, --name NAME	advertised name to embed in the image (default: `ON FOTA RSL10`)
-o OUT-IMG	name of output image file (default: **`<APP-IMG>`***.fota*)

This Python utility can be integrated into the post-build steps of Eclipse to generate the *.fota* format file every time a user application project is built. This configuration is available in **Project** > **Properties** > **C/C++ Build** > **Settings** > **Build Steps** > **Post-Build Steps**. Two steps are required to generate the *.fota* image. First, we need to generate the *app.bin*, which is the application sub-image. This is done using the *objcopy* tool, as follows:

```
arm-none-eabi-objcopy -O binary <app_name>.elf <app_name>.bin
```

Then, we can call the *mkfotaimg.py* utility to generate the final FOTA image (*app.fota file*) from the stack sub-image (*fota.bin*) and the application sub-image (*app.bin*) generated above the *.fota* image, as follows:

```
mkfotaimg.py fota.bin <app_name>.bin
```

In order to invoke both commands, the two can be combined with the "&&" operator. Chapter 8, "Integrating FOTA Into Your Application" on page 27, walks you step-by-step through generically configuring this post-build step for any application, including the path for the Python utility, FOTA stack binary file, and application binary file.

5.3 Sub-Image Format

A sub-image contains the 1-to-1 flash content for RSL10. We use positions 7 and 8 of the vector table to store pointers for the version info and the image descriptor:

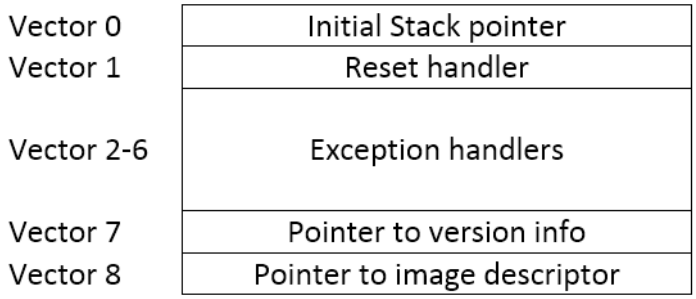


Figure 10. Vector Table Positions

Every vector is a 32-bit value. With the Reset handler vector, the image start address can be calculated as:

<image start address> = <Reset handler vector> & ~0x7FF

To find the version info and image descriptor, we calculate the corresponding image offsets by subtracting the image start address from the vector value:

<offset version info> = <Pointer to version info> – <image start address>

<offset image descriptor> = <Pointer to image descriptor> – <image start address>

All multi-byte values in the description are in little-endian format.

The version info has the following format:

```
typedef struct
{
    char     id[6];       // ID string
    uint16_t num;         //<major[15:12]>.<minor[11:8]>.<revision[7:0]>
} version;

typedef struct
{
    version  img_ver     // image version
    uint8_t  dev_id[16]; // device UUID set by mkfotaimg (default all 0s)
} version_info;
```

In the FOTA stack sub-image, the version info is directly followed by a configuration structure:

```
typedef struct
{
    uint32_t length;       // length of this structure in bytes
    uint8_t pub_key[64];   // public signing key set by mkfotaimg
                           // (default all 0s)
    uint8_t srv_id[16];    // service UUID used when advertising
                           // set by mkfotaimg (defaults to the
                           // DFU service ID)
    uint16_t dev_name_len; // device name length set by mkfotaimg
                           // (defaults to 13)
    uint8_t dev_name[29];  // device name used when advertising
                           // set by mkfotaimg (defaults to
                           // "ON FOTA RSL10")
 } config_info;
```

The image descriptor has the following format:

```
typedef struct
{
    uint32_t image_size;  // image size in bytes excluding the signature
    uint32_t build_id[8]; // FOTA stack build ID
} image_descriptor;
```

The FOTA stack build IDs from the two sub-images must match; otherwise it means that the application image has been linked against another version of the FOTA stack image as included in the FOTA image, in which case an `IMAGE_DNL_BAD_BUILDID` error is generated. The public key derived from the signing key is stored in the configuration structure of the FOTA stack sub-image (see above `struct config_info` field `pub_key`). The image size found in the image descriptor excludes the signature, so you must add 64 to the size to get the total sub-image size.

5.4 Signing the FOTA Image

The *mkfotaimg.py* tool provides an optional feature for signing the generated FOTA image with the `-s` or `--sign` optional argument. It requires a PEM file that is used as the signing key. Both sub-images (Bluetooth Low Energy stack and application) are appended by their 64-byte signatures before being concatenated into a single *.fota* file. These signatures are SHA256 sub-image hashes, signed with the elliptic curve NIST256p derived from the provided signing key. When the `-s` or `--sign` parameter is not used, the images are not signed, and a 64-byte dummy signature is appended to the sub-images.

To generate a signing key, you can take advantage of the Python *ecdsa* package and save the generated key in a PEM file, as shown below:

```
>>> import ecdsa
>>> sign_key = ecdsa.keys.SigningKey.generate(curve=ecdsa.curves.NIST256p)
>>> with open('key.pem', 'wt') as file:
...     pem = sign_key.to_pem()
...     file.write(pem.decode())
```

You can also use other tools to generate your signing key, such as *openssl*:

```
> openssl ecparam -name prime256v1 -genkey -noout -out key.pem
```

IMPORTANT: The signing key is a secret. Keep it safe!

5.5 FOTA Signature Validation

This section describes the validation of the FOTA image's signature.

5.5.1 Performing Signature Validation

To perform FOTA signature validation, proceed as follows:

1. Create a signing key, as described above in Section 5.4, "Signing the FOTA Image" on page 17. This signing key must be used for all FOTA image creations that apply to the same device.
2. Build your application, and call (normally implicit by post-build step) `mkfotaimg.py` with the parameter's **`<signingkey>`** (see Section 5.2, "mkfotaimg.py" on page 14) to generate the initial FOTA image.
3. Program the bootloader to the RSL10 as shown in Section 4.3, "Setting Up the RSL10 Bootloader and Loading a Firmware Image Using UART" on page 11.
4. Program the initial FOTA image into the device, via the bootloader, as shown in Section 4.3, "Setting Up the RSL10 Bootloader and Loading a Firmware Image Using UART" on page 11.
5. Build the new release of the application.
6. Call `mkfotaimg.py` with the same signing key to generate the new FOTA image.
7. Download the new FOTA image to the device with *Fota.Console*.

This process guarantees that only FOTA images which are signed with the same key are accepted by the device. The device can only check the image signature after the whole image has been transferred—and therefore, has already

overwritten the previous application. If the signature check fails, the device is without a valid application. You can still download a valid FOTA image if one is available; without a valid FOTA image to download, the device is not operable.

Therefore, `mkfotaimg.py` provides the parameter `-d` (see Section 5.2, "mkfotaimg.py" on page 14). If you provide this parameter with a device specific UUID (which you can generate yourself), this UUID becomes embedded in the FOTA image, causing the device to accept only new FOTA images with the same UUID. The device can check this UUID before the currently installed application is overwritten; and if there is a mismatch, the device still works.

The bootloader does no signature check; therefore, it is possible to install any FOTA image via the bootloader. An update via FOTA is only possible if the signature of the new FOTA image corresponds with the signature key already installed. If no signature key is installed on the device, any sub-image (signed or not) is accepted. If the transferred sub-image is a signed FOTA stack sub-image, then only correctly signed sub-images will be accepted.

5.5.2 FOTA Image Checking with the DFU Component

When generating a FOTA image, `mkfotaimg.py` first embeds the information provided to it through parameters (device UUID, advertising UUID, advertising name and public part of signing key) into the FOTA stack sub-image (device UUID instruct version, advertising UUID, advertising name and public part of signing key `instruct config_info`) and into the application sub-image (only device UUID instruct version). (See Section 5.3, "Sub-Image Format" on page 15.) Then `mkfotaimg.py` signs both the FOTA and application sub-images with the private part of the signing key, appends the signatures to the corresponding sub-images, pads the FOTA stack sub-image to RSL10 flash sector (2KiB) alignment, concatenates the FOTA stack sub-image followed by the application sub-image, and finally writes the result as a FOTA image to disk (see Section 5.1, "Overview" on page 14).

The DFU client (*Fota.Console*) splits the FOTA image into the two sub-images again, and transfers first the FOTA stack sub-image (only if needed) and then the application sub-image. The FOTA image can be split because both sub-images contain data on their own lengths (see Section 5.3, "Sub-Image Format" on page 15). If the build ID of the new FOTA image (see Section 5.3, "Sub-Image Format", referring to struct `image_descriptor`) is different from the build ID of the installed FOTA image (which can be read out via the FOTA Stack Build ID characteristic, shown in Section 6.5, "DFU Service Characteristics" on page 23), the DFU client has to send the FOTA stack sub-image first.

The DFU component of the FOTA stack checks the embedded information in the already installed FOTA stack sub-image version against a received sub-image (see code file *fota/dfu/app_dfu.c* function CheckImage), in this way:

1. Checks the start address of the sub-image; if the address is incorrect, the process aborts with error code `IMAGE_DNL_BAD_START`.
2. Checks the image length against available space in flash memory; if the length is incorrect, the process aborts with error code `IMAGE_DNL_BAD_SIZE`.
3. Checks the device UUID; if there is a UUID mismatch, the process aborts with error code `IMAGE_DNL_BAD_DEVID`.
4. For application sub-images only, checks build ID; if there is a build ID mismatch, the process aborts with error code `IMAGE_DNL_BAD_BUILDID`.
5. The verification steps 1 through 4 are performed as soon as the image header is received. If the verification is successful, the DFU component continues to download the rest of the sub-image and starts overwriting the application area in the flash memory.
6. Checks the received signature against the calculated image hash and the public part of the signing key; if the key is incorrect, the process aborts with error code `IMAGE_DNL_BAD_SIG` and the downloaded image is not marked as valid.
7. Restarts the device. (If the bootloader then detects a valid FOTA stack sub-image in the application area, it will copy it to the FOTA stack area first before starting the DFU component from the newly installed FOTA stack sub-image.)

5.5.3 More About Digital Signature Validation

The public part of the signing key is stored in the already installed FOTA stack sub-image in the struct `config_info` field `pub_key` (see Section 5.3, “Sub-Image Format” on page 15).

For the general workings of digital signatures, refer to:

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

For the specific algorithms used by FOTA, refer to:

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

CHAPTER 6

The DFU

6.1 DFU Component

The Device Firmware Update (DFU) component, which is embedded in the FOTA stack, acts as the server for the DFU service. This component performs the actual update via the DFU custom service. The DFU component is activated automatically at device startup if no valid application sub-image is found on the device. The application can also start the DFU component by calling the function `Sys_Fota_StartDfu()`.

6.2 Update Sequence

The DFU client scans for a device advertising the configured service ID (normally the DFU service ID). When the device is found, the DFU client connects to the device and reads its characteristics (Device ID, Versions, Build ID). The client must only accept FOTA images with matching Device IDs, unless the Device ID characteristic is all 0s, which means the device is compatible with all FOTA images.

If the Build ID of the application sub-image differs from the Build ID characteristic, the client needs to download the FOTA stack sub-image first. After this downloads successfully, the client needs to disconnect from the device, which restarts the device with the new FOTA stack. Now the client needs to reconnect to the same device.

If the Build ID of the application sub-image matches the Build ID characteristic, the client needs to download the Application sub-image. After successfully downloading, the client needs to disconnect from the device, which restarts the device with the new user application.

6.3 FOTA Stack Source Code

The RSL10 CMSIS-Pack includes a prebuilt version of the FOTA Stack available in the form of a software component, as shown in Chapter 4, “Performing Your First FOTA Update” on page 9. The *fota.bin* and *libfota.a* files are located under **<cmsis_pack_root>***/ONSemiconductor/RSL10/***<version>***/lib/Release*. The source code to generate these files is also available in the CMSIS-Pack, under **<cmsis_pack_root>***/ONSemiconductor/RSL10/***<version>***/source/firmware/fota*.

If you would like to customize and rebuild these files, follow these steps:

1. Import the source code project. Navigate to **File** > **Import** > **General** > **Existing Projects into Workspace**, set the **Select root directory:** option with the path to the source code, mark the checkbox **Copy projects into workspace**, and click **Finish**. The project appears in the left side of **Project Explorer**, as shown in Figure 11 on page 21.

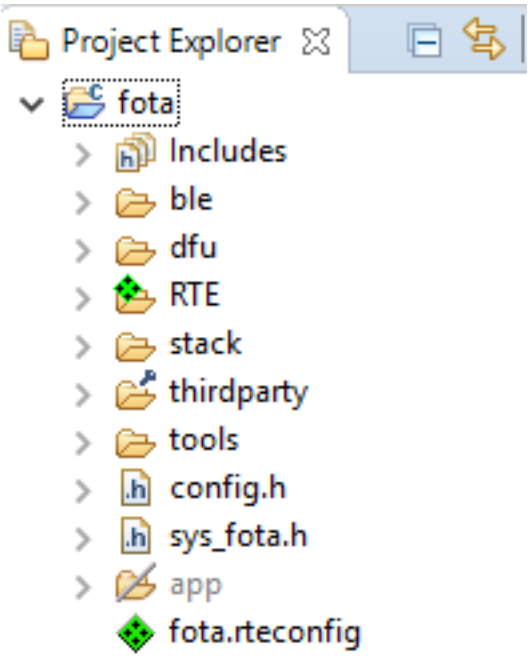


Figure 11. Copying the Project

2. Modify the source code. For example, change the FOTA stack version number in *config.h*.

```
#define VER_MAJOR                       2
#define VER_MINOR                       5 //4
#define VER_REVISION                    0
```

3. Build the project. After building successfully, *libfota.a* and *fota.bin* are generated under the *Debug* or *Release* folder

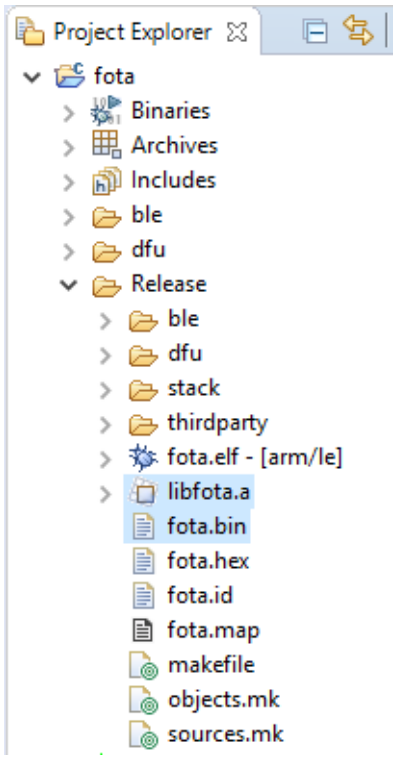


Figure 12. Building the Project

4. To use your customized files, replace the *libfota.a* and the *fota.bin* in your in your CMSIS-Pack installation (**<cmsis_pack_root>***/ONSemiconductor/RSL10/***<version>***/lib/Release*).
5. After replacing these files, it is necessary to refresh the *RTE* folder of existing projects in your workspace, so that the IDE uses the newly generated files. The simplest way to do this is to delete the *RTE* folder and refresh your sample project (for example, you can do this in *ble_peripheral_server_hrp_fota*).
6. Rebuild the *ble_peripheral_server_hrp_fota* project so that a new *.fota* image is generated based on your modified *libfota.a* and *fota.bin* files.
7. Run *Fota.Console* to perform a FOTA update. The tool detects the updated version number of the stack **FOTA stack version: FOTA <higher_version_number>**, in comparison to the one installed on the device. (In this example, we are using **FOTA stack version: FOTA 2.5.0**, for when version 2.4.0 is installed.) This means a full update is required (FOTA Stack + application). This is performed in two steps: first, the FOTA stack is updated; next, the device resets and the user application is updated. Notice the `UpdateFotaImage` and `UpdateAppImage` logs below.

```
> Fota.Console.exe /COM=COM5 /IN=ble_peripheral_server_hrp_fota.fota /
   NAME=Peripheral_HRP_FOTA
Fota.Console v1.0.5.0
--------------------------------------------------------------------------------
Searching for peripheral ...
Found peripheral (Address:AA:BB:FF:22:11:94 RSSI:-43dBm Name:Peripheral_HRP_FOTA)
Connecting
------------------------ Peripheral_HRP_FOTA ------------------------
FOTA stack version:  FOTA   2.4.0
```

```
Application version: PSHRP  2.4.0
Device ID:           None
------------------------ ble_peripheral_server_hrp_fota.fota -----------------------
FOTA stack version:  FOTA   2.5.0
Application version: PSHRP  2.4.0
Device ID:     None
Service UUID:  b2152466-d600-11e8-9f8b-f2801f1b9fd1
FOTA build ID: 04 00 00 00 10 00 00 00 03 00 00 00 47 4E 55 00 23 F3 2C 65 FD E3 13 B0 EC
    97 F5 8F 1F E0 E1 47

Establish
RebootToBootloader
Establish

UpdateFotaImage
 <-- 100.0% 15.0kB/s UpdateFotaImage
Establish

UpdateAppImage
 <-- 100.0% 6.3kB/s UpdateAppImage
Finished
Completed with status Success
```

6.4 DFU Bluetooth Low Energy Service

On the RSL10 device the DFU service is used as the server; the DFU service client runs on the PC (the *FOTA.Console* tool).

Table 3. DFU Service UUID

Requirement	UUID
Mandatory for DFU component, optional for device application	b2152466-d600-11e8-9f8b-f2801f1b9fd1

If the device application does not implement this service, another method must be used to activate the DFU mode in the device (e.g. push button).

6.5 DFU Service Characteristics

Table 4. DFU Service Characteristics and Their Properties

Characteristic	UUID	Properties	Length	Description	Requirements
Transport	b2152466-d601-11e8-9f8b-f2801f1b9fd1	Notify, Write without response	variable (max. 512)	Internally used characteristic to transport data with the DFU protocol.	Mandatory for DFU component, prohibited for device application.
Device ID	b2152466-d602-11e8-9f8b-f2801f1b9fd1	Read	16	Device ID as found in the version info of the FOTA stack sub-image. Only FOTA images with the same device ID are compatible, unless this characteristic is all 0s, in which case all FOTA images are compatible.	Mandatory for DFU component, optional for device application.

Table 4. DFU Service Characteristics and Their Properties (Continued)

Characteristic	UUID	Properties	Length	Description	Requirements
BootLoader Version	b2152466-d603-11e8-9f8b-f2801f1b9fd1	Read	8 (Format struct version)	Version of the installed BootLoader.	Mandatory for DFU component, optional for device application.
FOTA Stack Version	b2152466-d604-11e8-9f8b-f2801f1b9fd1	Read	8 (Format struct version)	Version of the installed FOTA stack sub-image (of type struct version).	Mandatory for DFU component, optional for device application
Application Version	b2152466-d605-11e8-9f8b-f2801f1b9fd1	Read	8 (Format struct version)	Version of the installed application sub-image. If no valid application sub-image is currently installed, then all 0s is returned.	Mandatory for DFU component, optional for device application.
FOTA Stack Build ID	b2152466-d606-11e8-9f8b-f2801f1b9fd1	Read	32	Build ID as found in the descriptor of the FOTA stack sub-image. If the Build ID of the FOTA image to download is different from the one in this characteristic, then both sub-images must be updated; otherwise, updating only the application sub-image is sufficient.	Mandatory for DFU component, optional for device application
Enter DFU	b2152466-d607-11e8-9f8b-f2801f1b9fd1	Write	1	A write with the value 1 switches from the Application mode to the DFU mode.	Prohibited for DFU component, mandatory for device application.

6.6 DFU Protocol

The application layer uses a Command/Response scheme. Every Command/Response consists of a standard header and an optional body. The header has the following format:

```
typedef struct
{
    uint8_t  code;     // unique Command/Response code
    uint8_t  param[3]; // Command/Response specific parameters
    uint32_t body_len; // length of the following body (0 = no body)
} header;
```

Currently only a single Command/Response is specified: IMAGE_DOWNLOAD (code = 1). The command is used to transfer a sub-image to the RSL10 device. The three parameters are not used by the command and must be set to 0. The body contains one of the two sub-images including the signature. The response signals back the success or failure of the operation; in the case of failure, the device can send the response before the whole command body is transferred. The response itself has no body. `Param[0]` is used for the status; the other parameters are not used, and must be set to 0.

Table 5, below, shows the specific status codes:

Table 5. Status Codes

Code	Meaning
0	The sub-image has been downloaded successfully
1	Download rejected due to incompatible Device ID
2	Download rejected due to incompatible Build ID (only for application sub-images)
3	Download rejected due to image size too large or small
4	Download failed due to flash storage error
5	Download failed due to invalid signature
6	Download rejected due to invalid start address

For the transport layer, an HDLC-like protocol with windowing is used to transport the upper layer SDUs. For the data-link layer, the transport characteristic of the DFU service is used.

CHAPTER 7

The Fota.Console Command Line Tool

7.1 FOTA.CONSOLE

Fota.Console is a simple PC command line tool for downloading firmware images using Bluetooth Low Energy technology. An RSL10 USB dongle on the PC side is required to establish a Bluetooth Low Energy connection.

Usage:

```
> Fota.Console.exe /COM=COMx /IN=<APP-IMG>.fota [/NAME=<device name>][/
    ADDR=<aa:bb:cc:dd:ee:ff>]
```

Parameters:

Table 6. Parameters

Parameter	Meaning
/COM	COM port connected to RSL10 Dongle (e.g. `COM1`)
/IN	Path to a *.fota* file that is sent over a Bluetooth Low Energy connection.
/NAME	Optional advertising name of the peripheral device
/ADDR	Optional Bluetooth Low Energy address of the peripheral to connect. e.g. 60:C0:BF:00:14:62

CHAPTER 8

Integrating FOTA Into Your Application

This chapter shows how to modify an existing sample application to make it capable of receiving FOTA updates. We start with the *ble_peripheral_server_bond* application and walk you, step-by-step, through all the required project configurations and firmware changes. Then, we show how to perform a FOTA update to confirm that the integration has been executed successfully.

8.1 Modifying the Application

1. Copy the *ble_peripheral_server_bond* sample application into your workspace. The **C/C++** perspective opens and displays your newly copied project, as shown below in Figure 13.

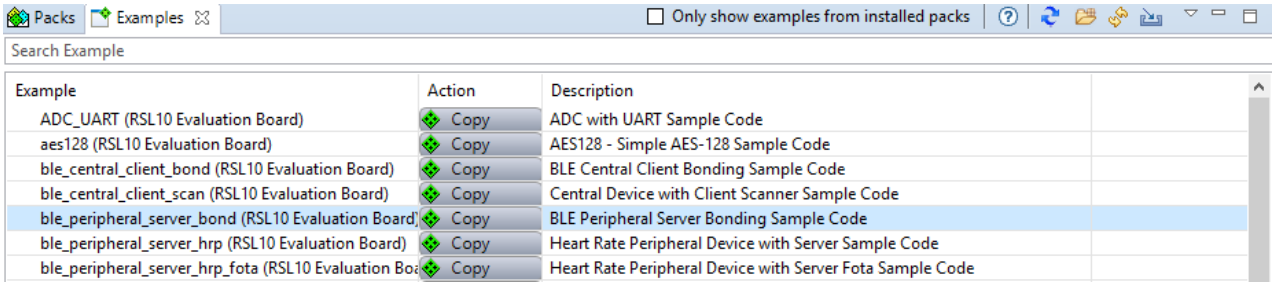


Figure 13. Bringing the Project Into the Workspace

2. Add the FOTA software component to your project by using the RTE Configuration Wizard (the *ble_peripheral_server_bond.rteconfig* file). When you select the **Fota** component and click save, *libfota.a*, *fota.bin* and the tool *mkfotaimg.py* are copied into your project, under **RTE** > **Device** > **RSL10**, as shown below in Figure 14:

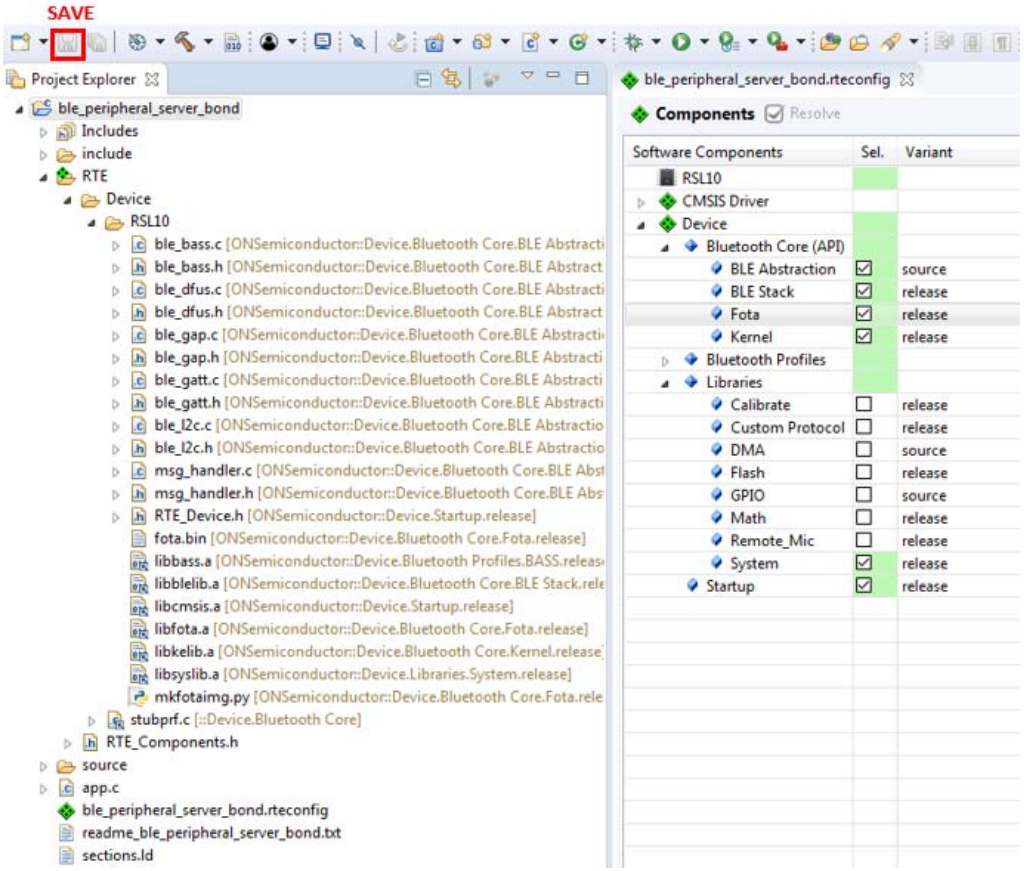


Figure 14. Adding the FOTA Software Component

3. Modify the source code to set the FOTA version number and ID:
   a. In *app.h*, define your version numbers and ID:

```
/* -----------------------------------------------------------------------
* Application Version
* ----------------------------------------------------------------------- */
#define VER_ID                  "PSBOND"
#define VER_MAJOR               2
#define VER_MINOR               4
#define VER_REVISION            0
```

   b. In *app.c*, include *sys_fota.h* and use the `SYS_FOTA_VERSION` macro to set the version:

```
#include "sys_fota.h"

/* -----------------------------------------------------------------------
* Application Version
* ----------------------------------------------------------------------- */
SYS_FOTA_VERSION(VER_ID, VER_MAJOR, VER_MINOR, VER_REVISION);
```

4. Modify the sample application to activate the DFU when the button (DIO5) on the RSL10 EVB is pressed.

a. In *app.h*, define DIO5 as `BUTTON_DIO`, this way:

```
#define BUTTON_DIO                      5
```

b. In *app_config.c*, modify the `Device_Initialize()` function to configure DIO5 as a GPIO input:

```
void Device_Initialize(void)
{
...
    /* Configure the push button of the RSL10 EVB as GPIO input */
Sys_DIO_Config(BUTTON_DIO, DIO_MODE_GPIO_IN_0 | DIO_WEAK_PULL_UP | DIO_LPF_DISABLE);
...
}
```

c. In *app.c*, add the code below in the `while(1)` loop of the `main()` function, to start the DFU when the button is pressed:

```
while (1)
{
    ...

    /* Start Update when button is pressed */
    if (DIO_DATA->ALIAS[BUTTON_DIO] == 0)
    {
        Sys_Fota_StartDfu(1);
    }

    ...
}
```

The DFU component is activated by calling the `Sys_Fota_StartDfu(mode)` function defined in *sys_fota.h*. The mode value 0 is for an application without the Bluetooth Low Energy stack (for example, *blinky*), and 1 is for an application that uses the Bluetooth Low Energy stack (such as *ble_peripheral_server_bond*).

5. Change the device name for testing:

```
/* Advertising data is composed by device name and company id */
//#define APP_DEVICE_NAME                 "ble_periph_server_bond"
#define APP_DEVICE_NAME                 "Peripheral_BOND_FOTA"
```

6. Replace your *sections.ld* file with the one that contains the FOTA placement. The new *sections.ld* file can be copied from *ble_peripheral_server_hrp_fota*, or directly from **`<CMSIS Pack root folder>`** > **ONSemiconductor** > **RSL10** > **`<version>`** > **source** > **firmware** > **fota** > **app**:

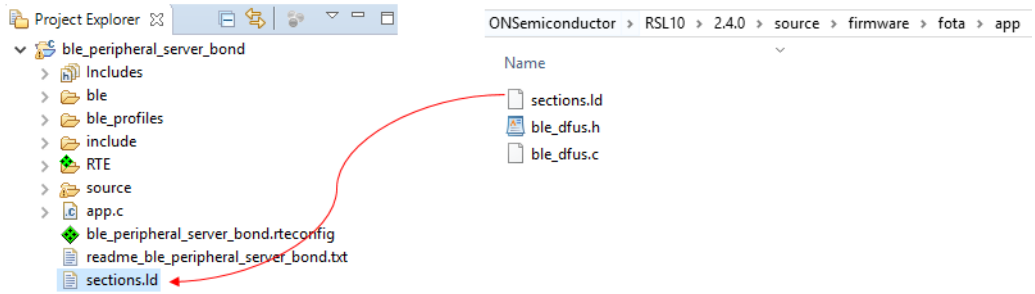


Figure 15. Replacing the sections.ld File

7. Modify the project post-build steps to generate the FOTA image in both Debug and Release build configurations, by going to **Project** > **Properties** > **C/C++ Build** > **Settings** > **Build Steps** > **Post-build steps**. Two steps are required to generate the *.fota* image. First, we need to generate the application sub-image *.bin* file using *objcopy*. Then, we use the *mkfotaimg.py* tool to generate the FOTA image (*.fota* file). Both commands can be concatenated with "&&" and added to the post-build steps as in the example that follows. To use this command, copy the code from the Post-Build steps of the existing sample application *ble_peripheral_server_hrp_fota*., as the code in the text below is not directly copyable.

```
${cross_prefix}objcopy -O binary"${BuildArtifactFileName}""${BuildArtifactFileBaseName}.bin"&&"${workspace_loc:/
    ${ProjName}}/RTE/Device/RSL10/mkfotaimg.py" -o"${BuildArtifactFileBaseName}.fota""${workspace_loc:/
    ${ProjName}}/RTE/Device/RSL10/fota.bin""${BuildArtifactFileBaseName}.bin"
```

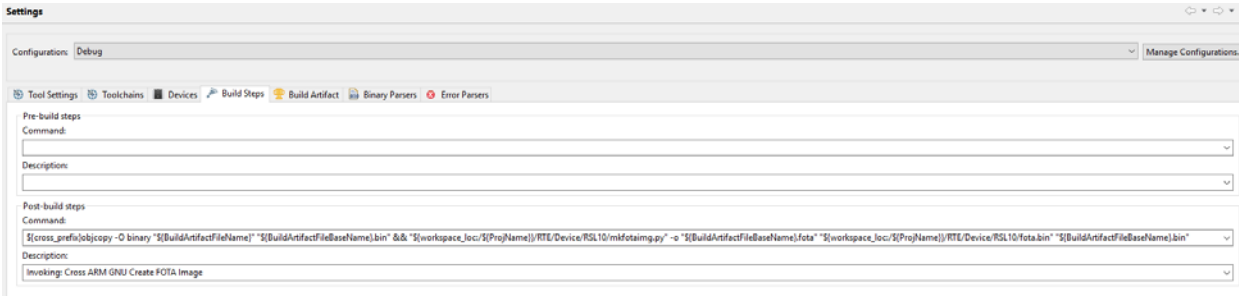


Figure 16. Post-Build Steps

8. Build the *ble_peripheral_server_bond* application. If no error occurs, the FOTA image (*ble_peripheral_server_bond.fota file*) can be found in the *Debug* or *Release* folder.

8.2 Performing a FOTA Update

1. Make sure you have *bootloader* running on your RSL10 EVB. If you have executed the steps in Chapter 4, "Performing Your First FOTA Update" on page 9, you already have *bootloader* flashed on your board and you can skip this step. Otherwise, refer to Section 4.3, "Setting Up the RSL10 Bootloader and Loading a Firmware Image Using UART" on page 11 for instructions.
2. Activate the *bootloader* updater mode: press both the reset and the user (DIO5) push buttons on the RSL10 EVB simultaneously, and then release the reset button first. Upon releasing both push buttons, the LED (DIO6) on the RSL10 EVB is set constant high, indicating that the *bootloader* updater is active, waiting for a firmware image over UART.

NOTE: If there is a valid user application in flash (for example, the *ble_peripheral_server_hrp_fota*) and no command is received over UART after a few seconds, the *bootloader* updater times out and reboots into the user application.

3. Load the *ble_peripheral_server_bond.fota* image using the *bootloader* with the following command:

```
> python updater.py COM4 ble_peripheral_server_bond.fota
Image      : PSBOND ver=2.4.0 / FOTA   ver=2.4.0
Bootloader : BOOTLD ver=2.0.1
********************************************************************************
```

After loading the image, the *bootloader* resets the device and boots up the user application.

4. Verify that your application is running by looking for the advertisement name **Peripheral_BOND_FOTA** on the RSL10 Bluetooth Low Energy Explorer, as in Figure 17 on page 31:

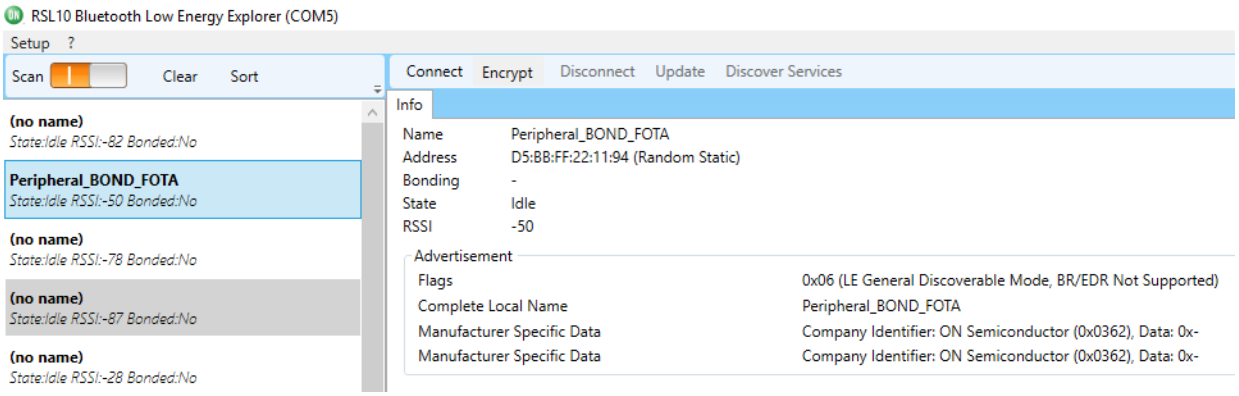


Figure 17. Application Running

5. Push the DIO5 button on your RSL10 EVB so that the device starts the FOTA DFU mode. You can see the name **ON FOTA RSL10** on the RSL10 Bluetooth Low Energy Explorer, as in Figure 18, below:

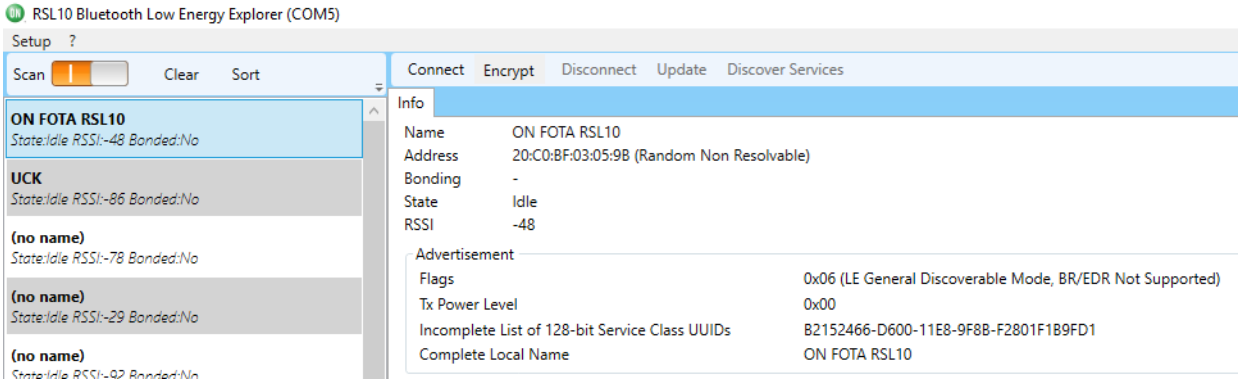


Figure 18. FOTA DFU Mode Running

Note: after 30 seconds, the FOTA DFU Mode times out and reboots into the user application.

6. Within 30 seconds of activating the FOTA DFU Mode, use the *FOTA.Console* PC application to update the firmware as follows:

```
Fota.Console.exe /COM=COM5 /IN=ble_peripheral_server_bond.fota
Fota.Console v1.0.5.0
--------------------------------------------------------------------------------

Searching for peripheral ...
Found peripheral (Address:20:C0:BF:03:05:9B RSSI:-43dBm Name:ON FOTA RSL10)
Connecting
------------------------ ON FOTA RSL10 ------------------------
Bootloader version:  BOOTLD 2.0.1
FOTA stack version:  FOTA   2.4.0
Application version: PSBOND 2.4.0
Device ID:           None
------------------------ ble_peripheral_server_bond.fota ------------------------
FOTA stack version:  FOTA   2.4.0
Application version: PSBOND 2.4.0
Device ID:     None
Service UUID:  b2152466-d600-11e8-9f8b-f2801f1b9fd1
FOTA build ID: 04 00 00 00 14 00 00 00 03 00 00 00 47 4E 55 00 5F 20 56 EC 3E E6 11 BD 98
   4E F2 5E 21 48 E7 97

Establish

UpdateAppImage
 <-- 100.0% 4.4kB/s UpdateAppImage
Finished
Completed with status Success
```

7. You can repeat these steps to update the firmware with the *ble_peripheral_server_hrp.fota* image as well. The steps described here can be applied to any RSL10 sample application.
8. You might have noticed that in this part of the tutorial we need to press the DIO5 push button to activate the FOTA DFU mode. On the other hand, in Chapter 4, "Performing Your First FOTA Update" on page 9, we use the *Fota.Console* tool to activate the DFU mode. The push button is required here, because we have not added the DFU Bluetooth Low Energy custom service yet, and in particular, the characteristic `Enter DFU` (Section 6.5, "DFU Service Characteristics" on page 23 for details).

   To get you up and running, we have included one example of this DFU custom component implementation. Copy the *ble_dfus.c* and *ble_dfus.h* files from **`<CMSIS Pack root folder>`** > **ONSemiconductor** > **RSL10** > **`<version>`** > **source** > **firmware** > **fota** > **app** and paste them in the *ble_profiles/include* and *ble_profiles/source* folders, as shown below in Figure 19:

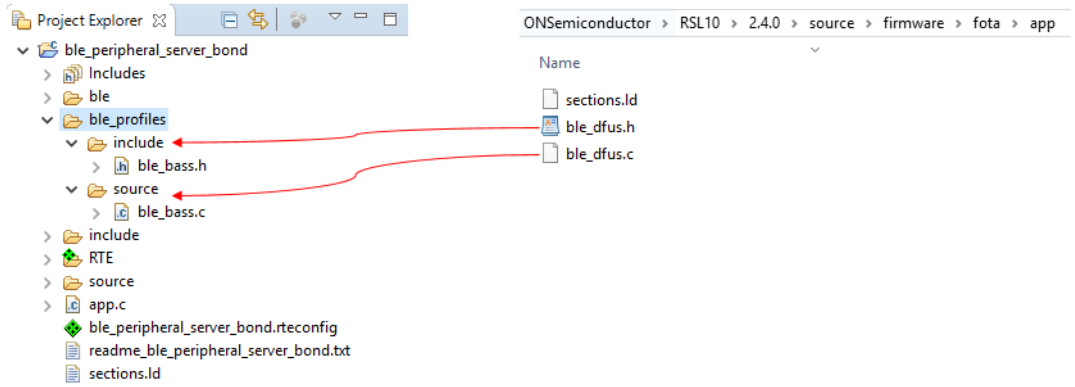


Figure 19. Sample DFU Custom Service Implementation

9. Modify *app.c* to include *ble_dfus.h* and call `DFUS_Initialize()` in the `main()` function:

```
#include <ble_dfus.h>
...
int main(void)
{
...
    DFUS_Initialize(); /* Initialize DFU Service Server */
...
}
```

10. Comment the call to `GATTM_AddAttributeDatabase()` in the *APP_GAPM_GATTM_Handler()* function in *app_msg_handler.c*, so that the component from the application is not added. The DFU component in FOTA already calls `GATTM_AddAttributeDatabase()` to add the characteristics for FOTA, so two separate calls to set the attribute database would make the application confused. If you wish to include both components, you need to merge their description in a single array.

```
void APP_GAPM_GATTM_Handler(ke_msg_id_t const msg_id, void const *param,
                            ke_task_id_t const dest_id,
                            ke_task_id_t const src_id)
{
    switch(msg_id)
    {
        case GAPM_CMP_EVT:
        {
            ...

            else if(p->operation == GAPM_SET_DEV_CONFIG && p->status == GAP_ERR_NO_ERROR)
            {
                //GATTM_AddAttributeDatabase(att_db, CS_NB); /* Add all custom services/
                                                              attributes */

            }
            ...
        }
```

11. Build the application and load it into your RSL10 EVB, using FOTA or the *bootloader* as shown in the previous steps.
12. Connect and discover services using the RSL10 Bluetooth Low Energy Explorer, and you can see that the DFU service is added in the attribute database, as shown in Figure 20 on page 34:

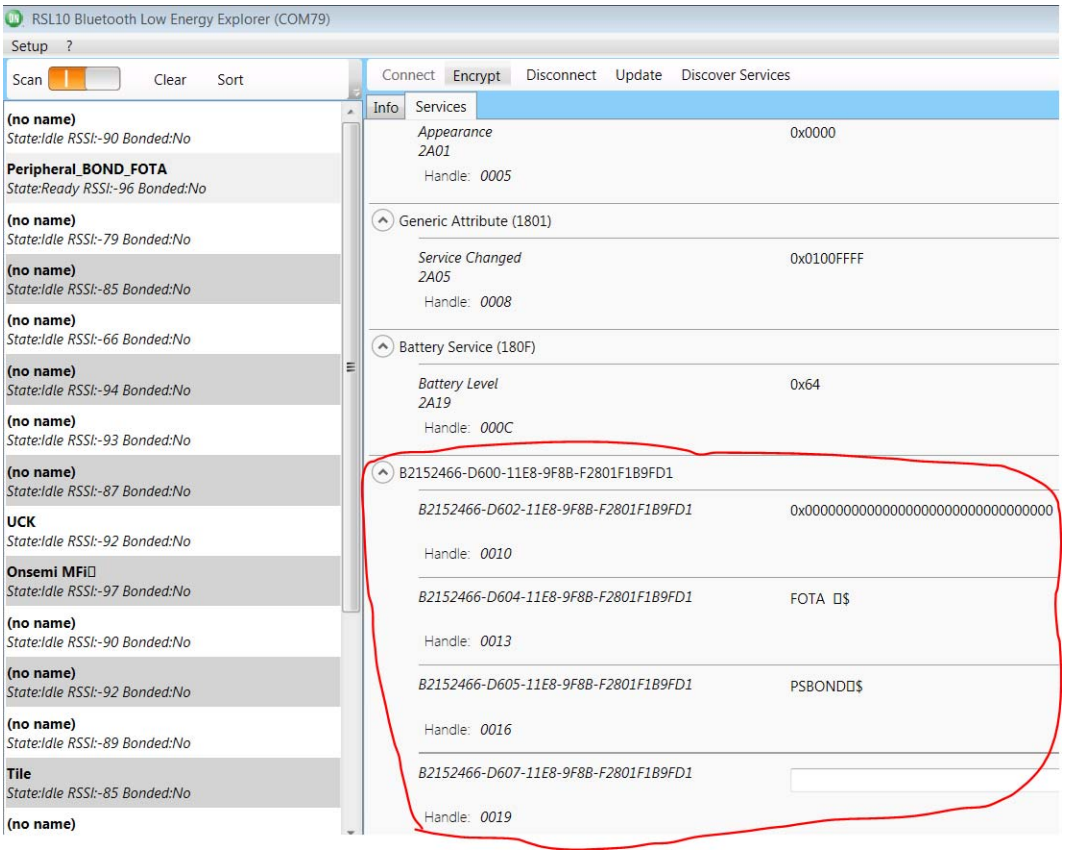


Figure 20. DFU Service in Attribute Database

13. The last characteristic shown (the one with the input text box) is the **Enter DFU** characteristic. If you write **01** into it, it enters the DFU mode and starts advertising under the name **ON FOTA RSL10**. This is what the *Fota.Console* application does prior to transmitting the image file.
14. Now that you have confirmed that your DFU service is up and running, you can use the *Fota.Console* tool with the attribute `/NAME=Peripheral_BOND_FOTA` to perform a FOTA update without the need to manually enter the DFU mode through the push button.

```
Fota.Console.exe /COM=COM5 /IN=ble_peripheral_server_bond.fota /
   NAME=Peripheral_BOND_FOTA
Fota.Console v1.0.5.0
------------------------------------------------------------------------------

Searching for peripheral ...
Found peripheral (Address:D5:BB:FF:22:11:94 RSSI:-48dBm Name:Peripheral_BOND_FOTA)
Connecting
------------------------ Peripheral_BOND_FOTA ------------------------
FOTA stack version:  FOTA   2.4.0
```

```
Application version: PSBOND 2.4.0
Device ID:           None
------------------------ ble_peripheral_server_bond.fota ------------------------
FOTA stack version:  FOTA   2.4.0
Application version: PSBOND 2.4.0
Device ID:     None
Service UUID:  b2152466-d600-11e8-9f8b-f2801f1b9fd1
FOTA build ID: 04 00 00 00 14 00 00 00 03 00 00 00 47 4E 55 00 88 5D D8 D8 CB 65 F0 94 C8
    BE D6 07 99 2F D8 72

Establish

RebootToBootloader

Establish

UpdateAppImage
 <-- 100.0% 7.7kB/s UpdateAppImage
Finished
Completed with status Success
```

CHAPTER 9

RSL10 FOTA Mobile Application

9.1 RSL10 FOTA Application

RSL10 FOTA is a simple Android mobile application to demonstrate Firmware-Over-The-Air (FOTA) for ON Semiconductor RSL10 Bluetooth Low Energy devices. The RSL10 FOTA application acts as a central device to scan, connect and transmit the firmware image to a remote RSL10 device. The remote RSL10 device firmware must have FOTA-enabled firmware to receive the FOTA firmware image.

Follow these steps to use the RSL10 FOTA Mobile Application:

1. Download and install RSL10 FOTA application from the Google Play Store website.
2. Launch the application and select the FOTA enabled firmware device from the list of devices.

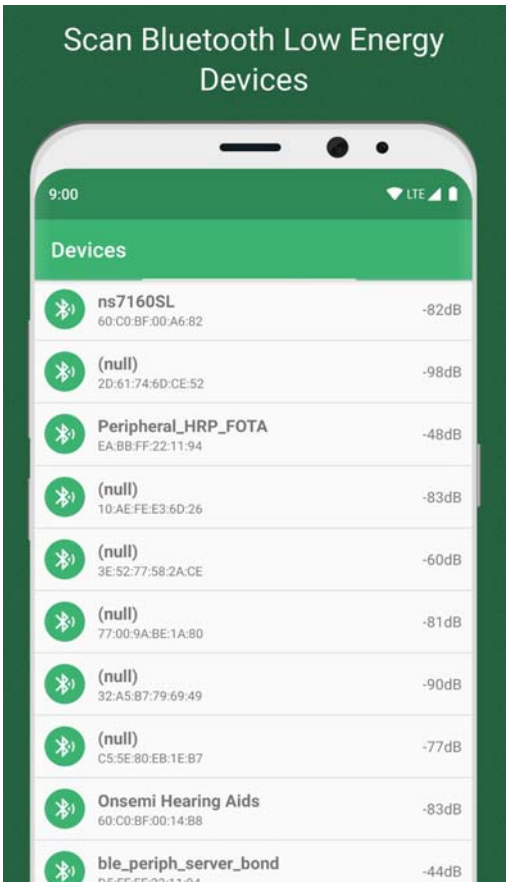


Figure 21. Mobile Application Screen Showing Bluetooth Low Energy Devices

3. When the appropriate device is selected, select the firmware image from the mobile by clicking the **Select File** button.
4. Once the FOTA firmware image file is selected, click the **Update** button.

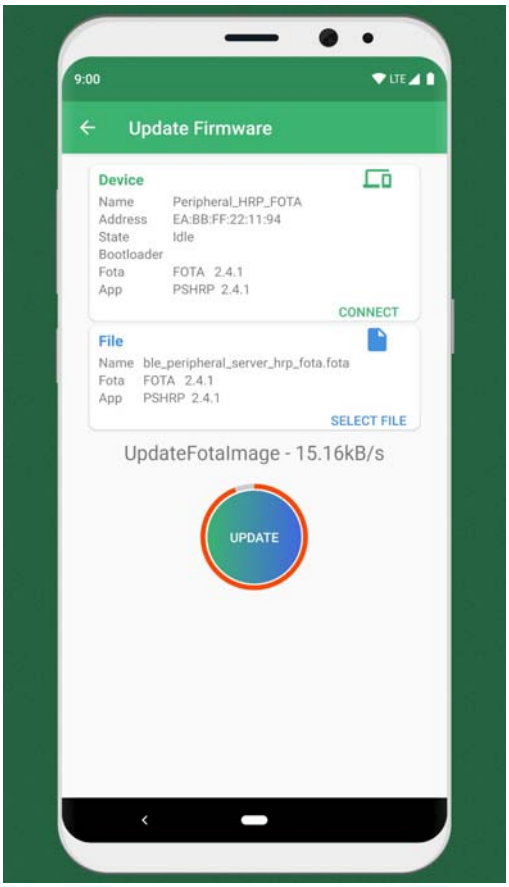


Figure 22. Mobile Application Screen: Updating Firmware

5. Upon firmware update completion, **Update finished with code: 0 (Success)** is displayed.

Note the following important details when using the RSL10 FOTA Mobile Application:

- Device location permission is needed to scan for Bluetooth Low Energy devices. If the permission is not granted at app startup, no Bluetooth Low Energy devices can be found.
- Device storage permission is needed to select a FOTA file.
- The FOTA file can be selected from the *Downloads* folder. The file can be transferred to the device, either over USB or by email.
- Requires Android version 5.0 or higher.

Ezairo is a registered trademarks of SCILLC. Bluetooth is a registered trademark of Bluetooth SIG, Inc. Windows is a registered trademark of Microsoft Corporation. All other brand names and product names appearing in this document are trademarks of their respective holders.

ON Semiconductor and  are trademarks of Semiconductor Components Industries, LLC dba ON Semiconductor or its subsidiaries in the United States and/or other countries. ON Semiconductor owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of ON Semiconductor's product/patent coverage may be accessed at www.onsemi.com/site/pdf/Patent-Marking.pdf. ON Semiconductor reserves the right to make changes without further notice to any products herein. ON Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does ON Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. Buyer is responsible for its products and applications using ON Semiconductor products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by ON Semiconductor. "Typical" parameters which may be provided in ON Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. ON Semiconductor does not convey any license under its patent rights nor the rights of others. ON Semiconductor products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use ON Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold ON Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that ON Semiconductor was negligent regarding the design or manufacture of the part. ON Semiconductor is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303-675-2175 or 800-344-3860 Toll Free USA/Canada
Fax: 303-675-2176 or 800-344-3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800-282-9855 Toll Free USA/Canada
Europe, Middle East and Africa Technical Support: Phone: 421 33 790 2910

ON Semiconductor Website: www.onsemi.com

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative