

RX23W Group

Firmware update over the air sample program

Introduction

This application note describes a sample program that implemented OTA (Over The Air) firmware update using the Bluetooth® Low Energy wireless communication that operates on the RX23W Group.

Target Device

RX23W Group

Related Documents

- RX23W Group Target Board for RX23W Quick Start Guide(R20QS0014)
- RX23W Group User's Manual: Hardware (R01UH0823)
- RX23W Group Bluetooth Low Energy Protocol Stack Basic Package User's Manual(R01UW0205)
- RX23W Group BLE Module Firmware Integration Technology Application Note(R01AN4860)
- RX23W Group Bluetooth Low Energy Profile Developer's Guide Application Note(R01AN4553)
- RX23W Group Bluetooth Low Energy Application Developer's Guide(R01AN5504)
- RX23W Group Sample Program for Highspeed Communication (R01AN5437)

The *Bluetooth*® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by Renesas Electronics Corporation is under license. Other trademarks and registered trademarks are the property of their respective owners.

Contents

1. Introduction.....	5
1.1 Package structure.....	5
2. Run the sample program	7
2.1 Operation environment.....	7
2.2 OTA Server setup.....	9
2.2.1 Write Firmware to Target Board for RX23W	9
2.3 Update by FWU_Client for Android	13
2.3.1 Installation of application	13
2.3.2 Firmware file transfer.....	13
2.3.3 Firmware update procedure	14
2.3.3.1 Operation procedure of smartphone application	14
2.3.3.2 How to check the update.....	19
2.3.4 Confirmed device.....	20
2.4 Update by FWU_Client for iOS	21
2.4.1 Installation of application	21
2.4.2 Firmware file transfer.....	21
2.4.2.1 Transfer from Finder or iTunes	22
2.4.2.2 Transfer from iCloud.....	23
2.4.3 Firmware update procedure	24
2.4.4 Confirmed device.....	24
2.5 Update by FWU_Client for Python	25
2.5.1 OTA Client RX23W setup.....	25
2.5.2 Python Application setup.....	25
2.5.3 Firmware update procedure	26
2.5.3.1 How to confirm the completion of the update.....	27
3. OTA firmware update feature.....	29
3.1 System structure.....	32
3.2 Section layout.....	33
3.3 Firmware update operation	34
3.3.1 Application update mode.....	34
3.3.2 BLE protocol stack update mode	35
3.4 Profile specifications.....	35
3.4.1 Renesas OTA Reset Service	36
3.4.2 Renesas OTA Service	36
3.4.3 Data format.....	37
3.5 OTA communication sequence	39
3.6 Switching startup programs.....	43
3.7 Bootloader	44

3.8	Relocator	45
3.8.1	Program Operations	45
3.8.2	Operation when power is cut off	46
3.9	Downloader	47
3.9.1	Bluetooth LE operation	47
3.9.2	Bonding information management	47
3.9.3	Writing to Code Flash	47
3.9.4	Operation when power is cut off	47
3.10	User application	48
3.10.1	Switching to downloader	48
3.10.2	Sample program user application	48
4.	Create an OTA update project	49
4.1	Add source code (r_ble_ota)	49
4.2	Section division settings	50
4.3	Standard library settings	54
4.4	Update firmware output settings	56
4.4.1	Firmware information JSON file properties	58
4.5	Flash FIT module settings	59
4.6	Addition of Renesas OTA Reset Service	60
4.7	Assignment of source code to each section	62
4.8	Editing dbstc.c	65
4.9	r_ble_ota configuration settings	66
4.10	User application modification	68
4.10.1	Main function settings	68
4.10.2	User application main function registration and initialization section settings	69
4.11	Addition of Renesas OTA Reset Service	70
4.11.1	Switching process to Renesas OTA Reset Service and downloader	70
4.11.2	Implementation of bonding	72
4.11.3	Indication processing of Service Changed Characteristic of GATT Service	72
5.	Update firmware settings	74
6.	Android application FWU_Client project	75
6.1	How to build	75
6.2	How to debug	75
6.3	Correspondence to API specification change of Android 10.0	76
7.	FWU_Client for iOS	77
7.1	Operation Confirmation Environment	77
7.2	Debug	77
7.3	iOS GATT Database Cache Function	77

8.	FWU_Client for Python	79
8.1	System Configuration	79
8.2	Control Command	79
8.3	Operation Sequence of Each Command	81
8.3.1	scan command	81
8.3.2	conn command	81
8.3.3	disconn command	83
8.3.4	update command	84
8.3.5	exit command	85
8.4	OTA Client	86
8.4.1	Block Diagram	86
8.4.2	Application Packet	87
8.4.3	Communication with command line interface	92
8.4.4	Correction Point for app_lib	93
8.4.5	Using High Speed Communication Sample Program	93
8.4.6	Notes	93
8.5	Python Application	94
8.5.1	Block Diagram	94
8.5.2	Log Outputs	94
9.	Modification from Version 1.00	96
9.1	Implement Service Changed Characteristic for user applications	96
9.2	Update FIT version	96
9.3	Modification Source Code in OTA Server	96
10.	Limitations and Notes	97
11.	Check items when updating fails	98
11.1	When return Error Response from OTA Server	98
11.2	When it stops immediately after the update starts	98
11.3	When using the code generator function of Smart Configurator	99
12.	Appendix	103
12.1	OTA firmware update time (reference value)	103
	Revision History	104

1. Introduction

The OTA firmware update sample program can update the firmware of RX23W with one of the following configurations using the Bluetooth Low Energy wireless communication. Refer to “2 Run the sample program” for more details.

- Transfer firmware from Android smartphone to RX23W to be updated
- Transfer firmware from iOS smartphone to RX23W to be updated
- Transfer firmware from RX23W with Windows PC to RX23W to be updated

The RX23W firmware can be updated in any of the following programs. Refer to “3 OTA firmware update feature” for more information on the OTA firmware update feature.

- User application only
- Both user application and Bluetooth Low Energy protocol stack

Refer to “4 Create an OTA update project” for how to create firmware for the RX23W that supports OTA firmware updates.

1.1 Package structure

Table 1.1 shows the package configuration of the OTA firmware sample program.

Table 1.1 Package structure of OTA firmware sample program

<pre> r01an5910xx0111-rx23w-otafwup ble_sample_tbrx23w_ota_client.zip ble_sample_tbrx23w_ota_server.zip r01an5910ej01110-rx23w-otafwup.pdf r01an5910jj01110-rx23w-otafwup.pdf ---Android FWU_Client.apk FWU_Client.zip ---iOS FWU_Client.zip ---ProjectSetting section.esi service json Renesas_OTA_Reset_Service_Client.json Renesas_OTA_Reset_Service_Server.json Renesas_OTA_Service_Client.json Renesas_OTA_Service_Server.json service api r_ble_otac.c r_ble_otac.h r_ble_ota_resetc.c r_ble_ota_resetc.h r_ble_ota_resets.c r_ble_ota_resets.h ---PythonApplication </pre>	<p>OTA Client sample project OTA Server sample project Application note (English) Application note (Japanese)</p> <p>FWU_Client application package file FWU_Client project file</p> <p>Project file for iOS version FWU_Client</p> <p>OTA Server section information export file</p> <p>QE for BLE Renesas OTA Reset Service file for client QE for BLE Renesas OTA Reset Service file for server QE for BLE Renesas OTA Service file for client QE for BLE Renesas OTA Service file for server</p> <p>QE for BLE Renesas OTA Service client service API QE for BLE Renesas OTA Reset Service client service API QE for BLE Renesas OTA Reset Service server service API</p>
---	--

<pre> BinaryFileReader.py CommandExecuter.py ClientDeviceCommunication.py FirmwareInfoReader.py main.py port.json SerialCommunication.py UserInterface.py SampleFirmware Difference ota_sample_Version1.11_src_diff r_ble_ota r_ble_ota_config.h smc_gen Config_BLE_PROFILE app_main.c gatt_db.c gatt_db.h r_ble_bas.c r_ble_bas.h ota_sample_Version1.12_src_diff r_ble_ota r_ble_ota_config.h downloader r_ble_ota_dl_cmt_driver.c FWU Client ota_sample_Version1.10 firmware.bin firmware_information.json ota_sample_Version1.11 firmware.bin firmware_information.json ota_sample_Version1.12 firmware.bin firmware_information.json mot ble_sample_tbrx23w_ota_client.mot ble_sample_tbrx23w_ota_server.mot Tool CompareFirmware.py </pre>	<pre> Binary read script Script to process commands OTA Client packet analysis script Firmware information read script Python Application main script Comport configuration file Serial communication script Input/Output script Source code difference of sample firmware v1.11 for update Source code difference of sample firmware v1.12 for update Firmware file for OTA update ROM file of OTA client sample program ROM file of OTA server sample program Firmware comparison script </pre>
--	--

2. Run the sample program

This chapter describes the procedure for executing the OTA firmware update sample program. In this sample, the program for RX23W (OTA Server) that is updated by the firmware update by OTA and the Client program (FWU_Client) that updates it are provided. FWU_Client is available in three versions: Android version, iOS version, and Python version that uses a PC and another Target Board. Firmware update can be executed with one of the following configurations.

- Transfer firmware from Android smartphone to RX23W to be updated
- Transfer firmware from iOS smartphone to RX23W to be updated
- Transfer firmware from RX23W with Windows PC to RX23W to be updated

In this sample, a firmware file (firmware.bin) and a JSON-formatted firmware update information file (firmware_information.json) are used as the update firmware. These files are stored in the [Project Name] _Version [Application Version] folder. The configuration of the update firmware folder (project name: ota_sample, application version: 1.10) is shown below.

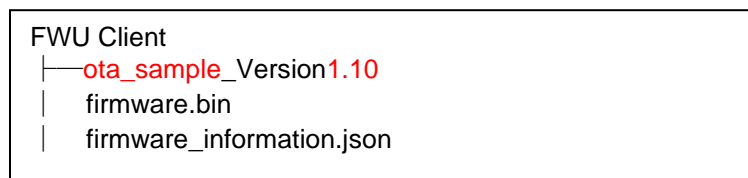


Figure 2.1 Structure of update firmware folder

The following three firmware are included as sample OTA firmware updates.

Table 2.1 OTA firmware sample

Firmware	Description
ota_sample_Version1.10	This is the firmware to be updated. The application has an LED Switch Service (LSS) *.
ota_sample_Version1.11	Firmware for updating user applications. Add the Battery Information Service to the GATT database.
ota_sample_Version1.12	Firmware that updates the Bluetooth LE communication part. Adds the feature to blink the LED on the target board while downloading the firmware.

*LED Switch Service is a demo service included in the BLE FIT Module.

In the execution of this sample program, the update procedure for the user application only is shown. It is possible to update application and Bluetooth LE communication part by the same procedure.

By executing this sample program, version 1.10 will be updated to Version 1.11. Battery Information Service (BAS) is added to the GATT Database.

2.1 Operation environment

The environment required for the operation of the OTA firmware update sample program is shown.

Table 2.2 Operating environment of sample program

Environment	Conditions
RX23W evaluation board	Target Board for RX23W 2 units (*1)
Compiler	C/C++ Compiler for RX Family V2.08.01
Integrated development environment	Renesas e ² studio 2021-07
Host PC	Windows 10
Script interpreter	Python 3.7.3
Smartphone	Android 8.0.0 or later iOS14.0 以上
Smartphone development environment	Android Studio 4.1.1 Xcode 12.5.1

*1: 2 units when transferring firmware from the RX23W OTA Client, 1 unit when transferring firmware from a smartphone

2.2 OTA Server setup

This section shows how to set up OTA Server to run this sample. Write the Version 1.10 firmware to the Target Board for RX23W using the Renesas Flash Programmer.

2.2.1 Write Firmware to Target Board for RX23W

Write the firmware file "ble_sample_tbrx23w_ota_server.mot" containing LSS to Target Board for RX23W. Use the Renesas Flash Programmer (RFP) to write the firmware to the RX23W.

When writing the firmware, switch ESW1-2 on the Target Board for RX23W to ON and connect the ECN of the micro-B connector to the PC as shown below.

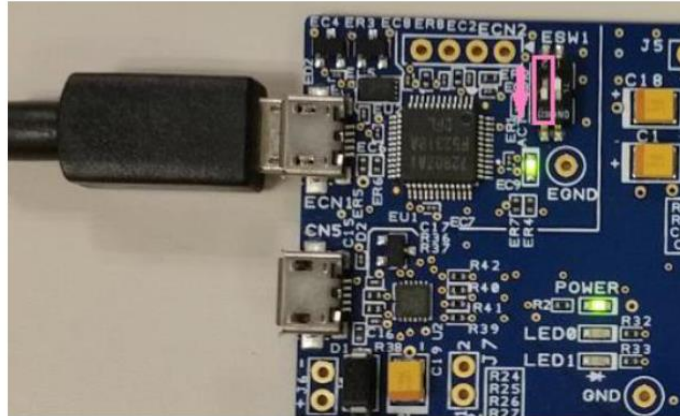


Figure 2.2 Target Board in Debugging Mode

Launch the RFP and select "File" → "Create New Project ..." from the menu.

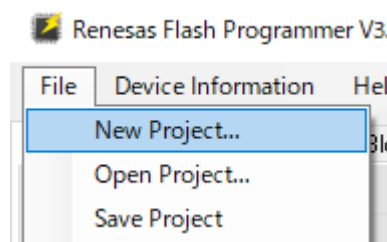


Figure 2.3 Create RFP new project

In the Create New Project dialog, set the following and click the "Connect" button.

- Microcontroller: RX200
- Project name Any name
- Location: Any location
- Communication tool: E2 emulator Lite
- Communication interface: FINE
- Power: Do not supply

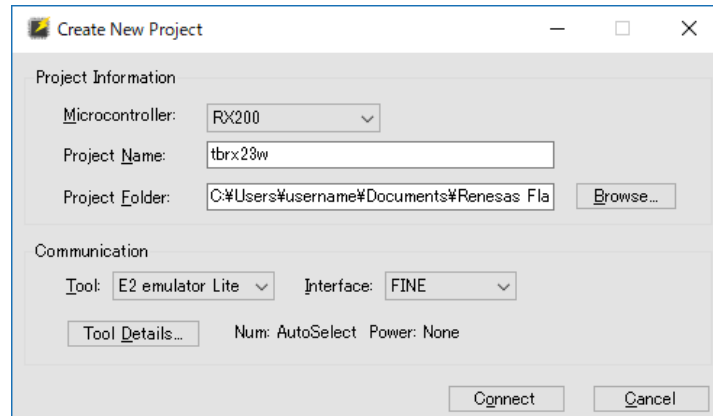


Figure 2.4 Create new project

Since the server device sample (ble_sample_tbrx23w_ota_server) uses a data flash, it is necessary to delete the information written to the data flash to prevent malfunction. The following screen capture shows to delete all data flash and code flash in RFP.

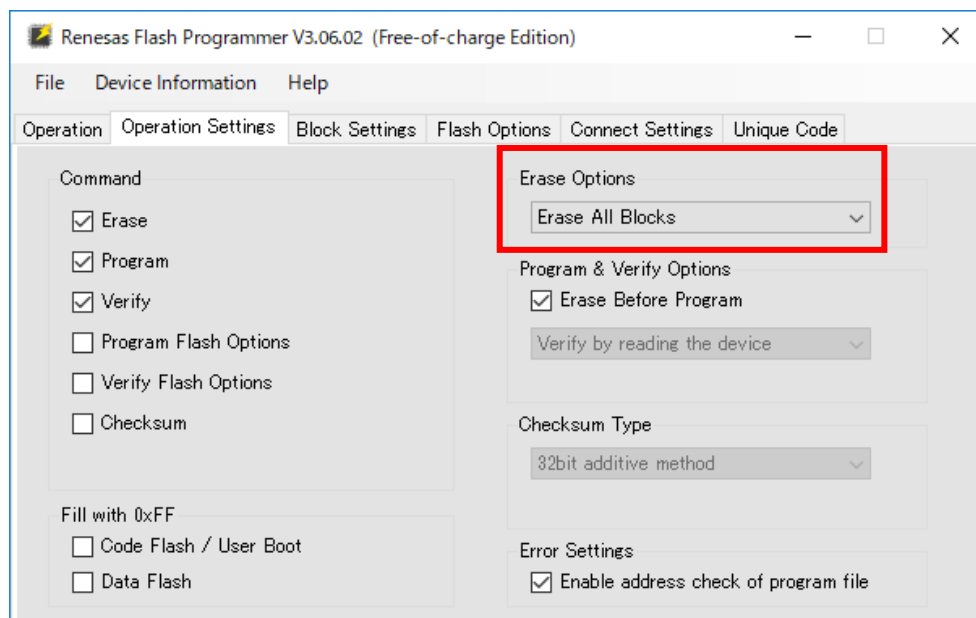


Figure 2.5 Settings for delete all data flash and code flash(Operation settings tab)

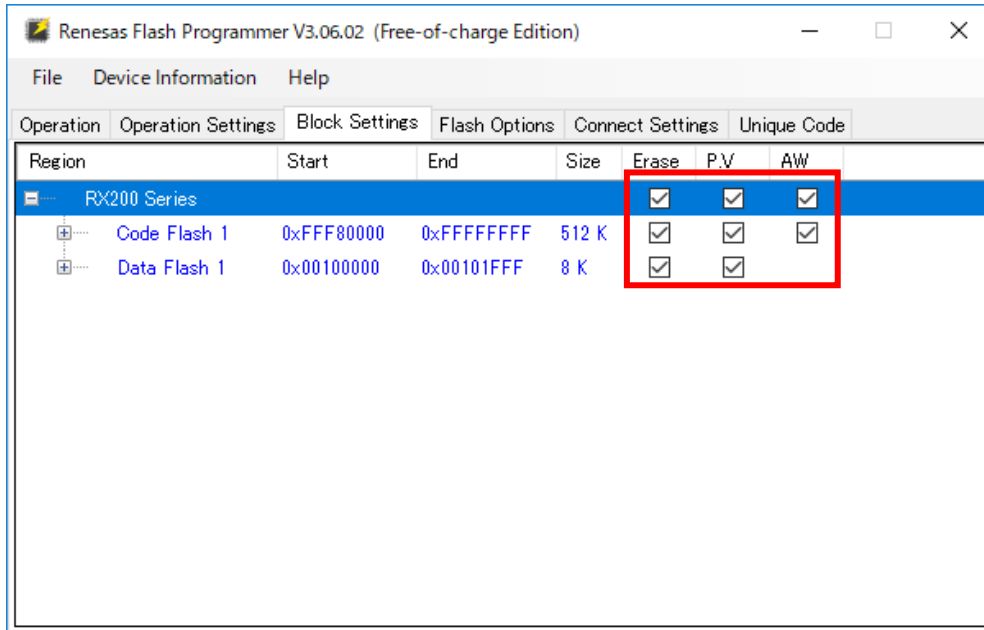


Figure 2.6 Settings for delete all data flash and code flash(Block Settings tab)

Go to the Operation tab, select the mot file and press the “Start” button to write the firmware.

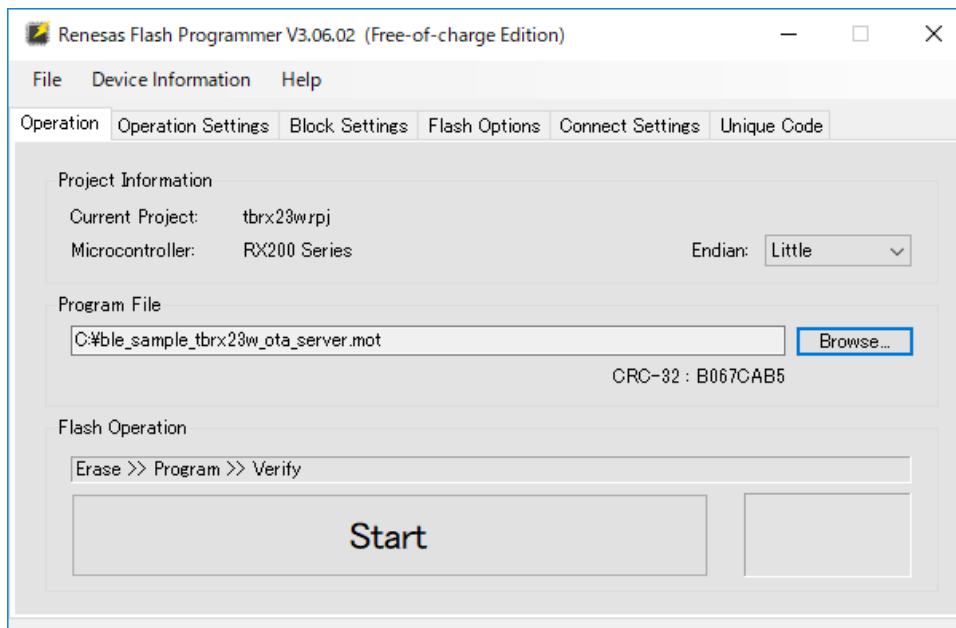


Figure 2.7 Write firmware screen

Finally, connect your PC to CN5 connector with a micro B type USB cable and change emulator switch (2 of ESW1) to OFF as shown in Figure 2.8.

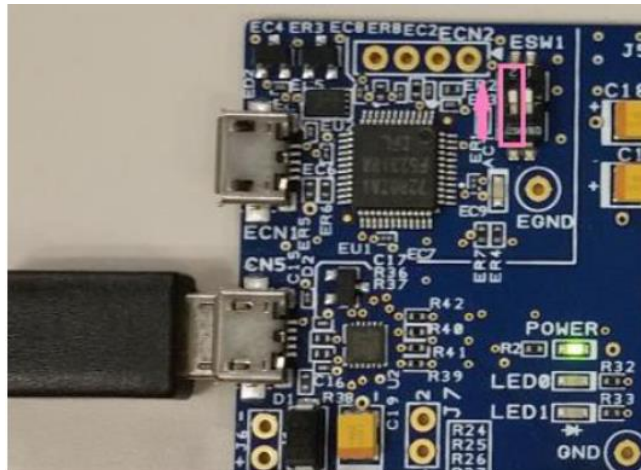


Figure 2.8 Target Board in Running Mode

If you connect to the OTA Server program using the GATTBROWSER of the smartphone application, you can confirm that the LED Switch Service is running as shown below.

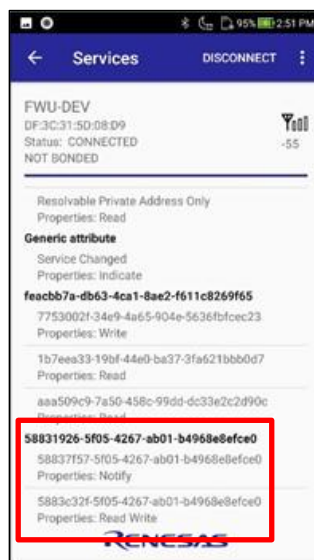


Figure 2.9 Connection with GATTBROWSER

The smartphone application GATTBROWSER is available on Google Play.

<https://play.google.com/store/apps/details?id=com.renesas.ble.gattbrowser>

For details on GATTBROWSER, refer to the application note below.

<https://www.renesas.com/document/apn/gattbrowser-android-smartphone-application-instruction-manual>

2.3 Update by FWU_Client for Android

2.3.1 Installation of application

Install FWU Client on your smartphone. For Android devices, transfer the application package file "FWU_Client.apk" to your smartphone and install it.

2.3.2 Firmware file transfer

Transfer the ota_sample_Version1.11 folder to your smartphone as shown in Figure 2.10. Create a "FWU Client" folder in the internal storage of your smartphone, and copy ota_sample_Version1.11 in the "FWU Client" folder of this sample package.

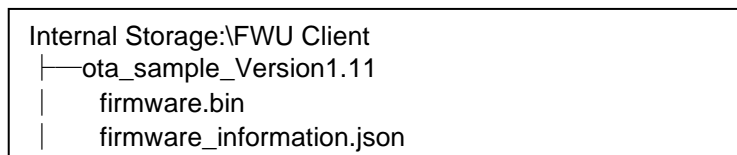


Figure 2.10 File structure of smartphone application

FWU_Client gets a folder that matches the project name (set by BLE_OTA_PROJECT_NAME) of the connected OTA Server. FWU_Client selects the latest version information and executes the update. If you transfer ota_sample_Version1.12, FWU_Client update both application parts and Bluetooth LE communication parts.

2.3.3 Firmware update procedure

2.3.3.1 Operation procedure of smartphone application

When you start FWU_Client, the device search screen will be displayed.



Figure 2.11 Device search screen

Tap OTA Server (device name: FWU-DEV) to connect*. After establishing the connection, the firmware information of the connected OTA Server is displayed on the device information screen. Updatable sections and versions are displayed at the bottom of the screen.

* If the key information held by the smartphone and the OTA Server key are different, you may not be able to connect. If the connection fails, delete the OTA Server key information held on the smartphone device and try again.



Figure 2.12 Device information screen

Swipe the screen to move to the Application update screen. On the update screen, the firmware information of the device to be updated and the firmware information that can be updated are displayed. If the section is updatable, the update button will turn blue, but if there is no firmware to update, the update button will be disabled. When updating application parts and the Bluetooth LE communication part, open the PROTOCOL STACK tab.

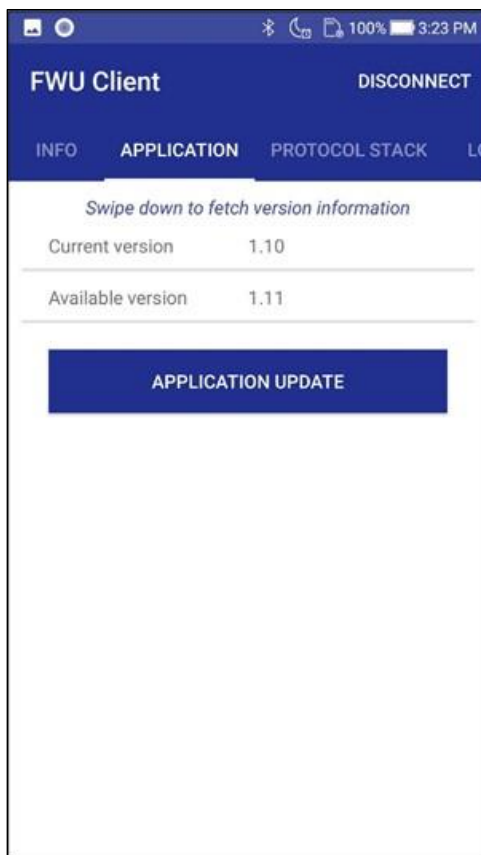


Figure 2.13 Application update screen

Tap the APPLICATION UPDATE button to start the update. The progress is displayed on the progress bar. Details being updated are displayed on the LOGS tab.

Depending on the communication environment, it takes about 2 to 5 minutes to update the application section.

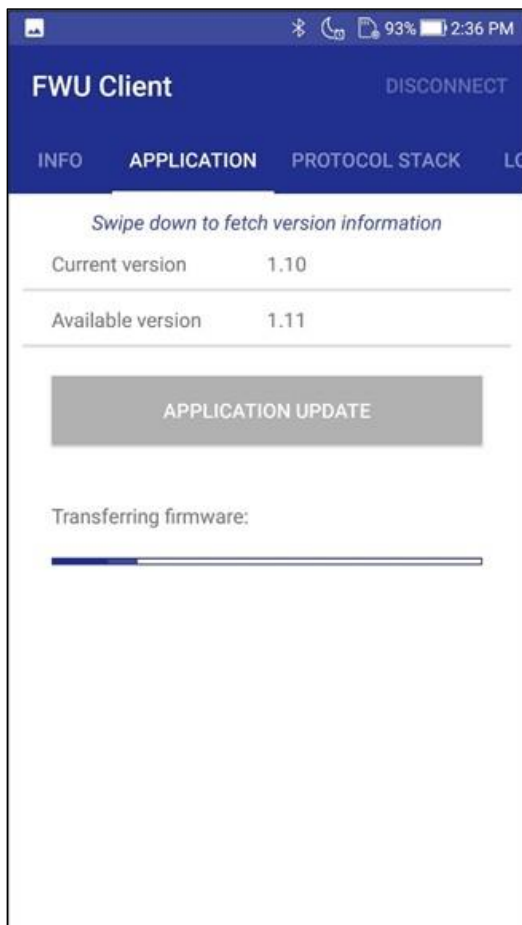


Figure 2.14 Update screen being updated

If the update is successful, OTA Server to be updated will be restarted. The FWU Client returns to the device search screen. If a problem occurs during the update, the cause will pop up and the device search screen will be displayed.

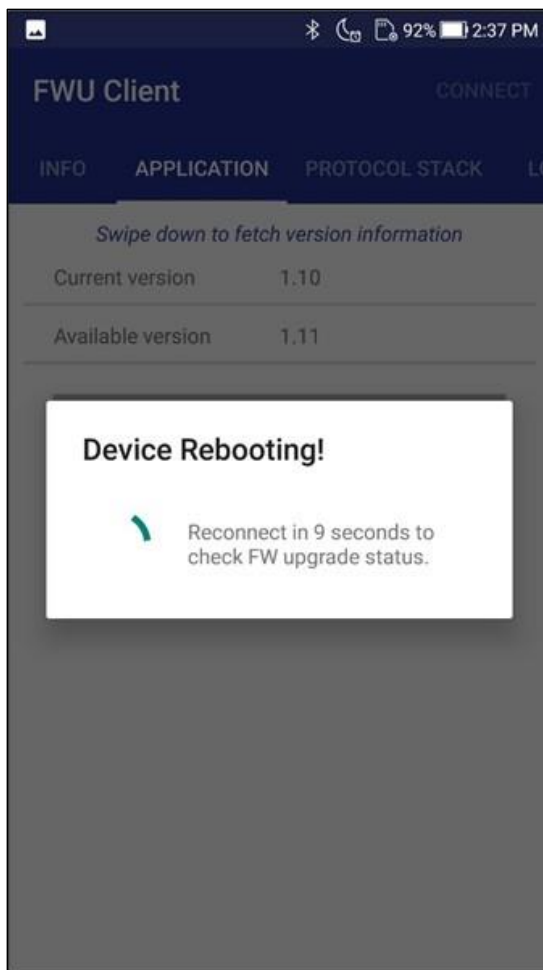


Figure 2.15 Screen when update is completed

2.3.3.2 How to check the update

Connect to OTA Server on the device search screen and check the OTA Server firmware version information on the device information screen.

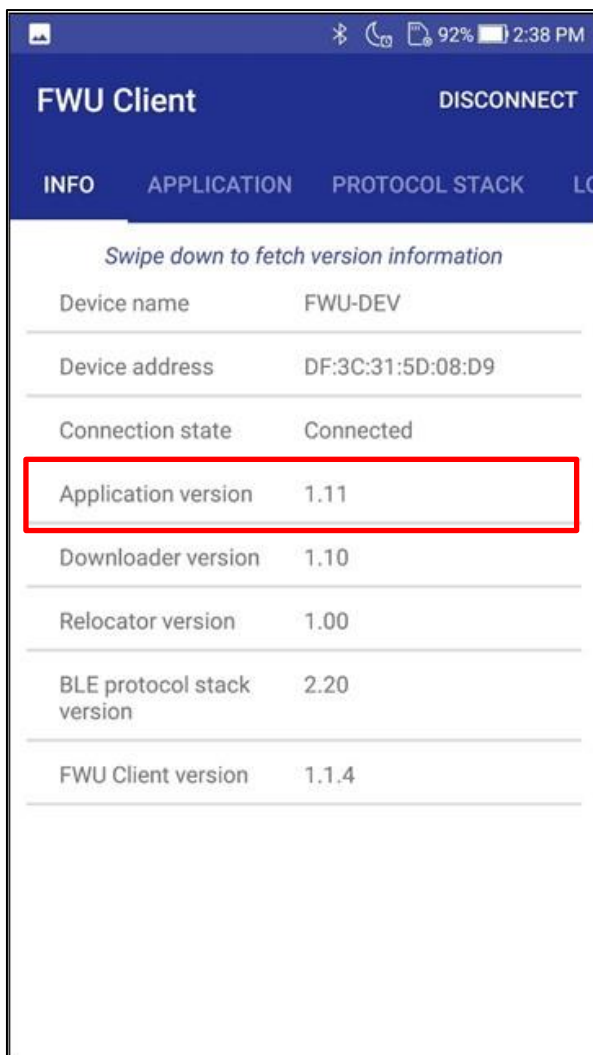


Figure 2.16 Confirmation of update completion

Connect OTA Server from the GATTBrowser and confirm that the Battery Information service has been added.

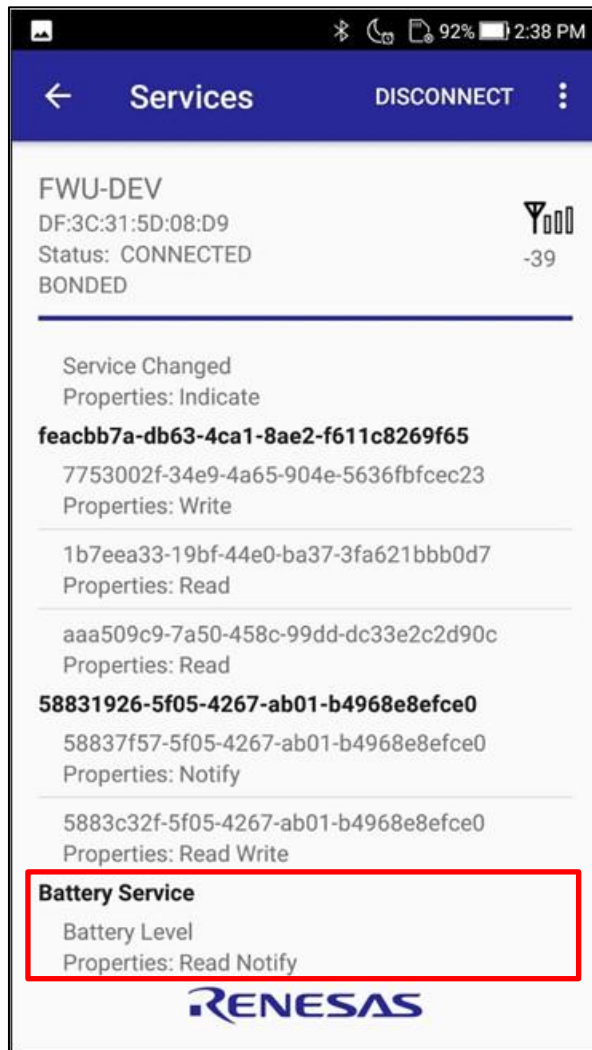


Figure 2.17 Confirmation of adding Battery Information Service

2.3.4 Confirmed device

The Android devices that have been confirmed to work are as follows.

Table 2.3 Confirmed device

Product manufacturer	Device name	Android Version
ASUS	ZenFone5	8.0.0
ASUS	ZenFone4	8.0.0
SHARP	SH-M05	8.0.0

2.4 Update by FWU_Client for iOS

To install FWU_Client, you need to enroll in the Apple Developer Program. Go to Apple's official website (<https://developer.apple.com>).

2.4.1 Installation of application

Open the FWU_Client project (FWU_Client.xcodeproj) in Xcode.

Set the Team and Bundle identifiers for your environment from the Signing & capabilities tab of your project. Connect your iOS device and select iOS Device from the project scheme. Press the Build button to launch the application.

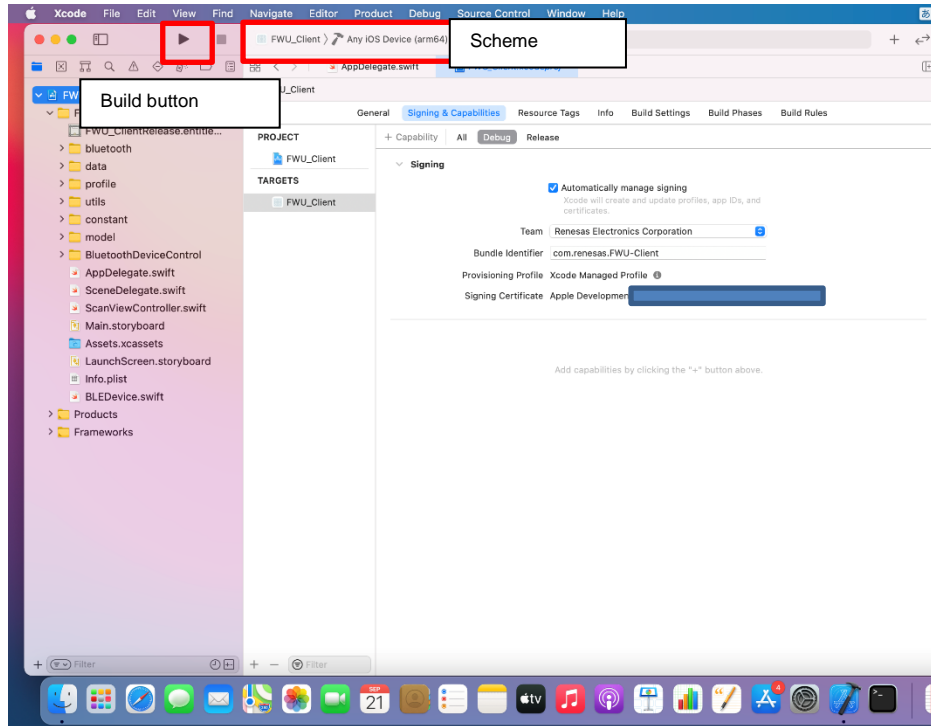


Figure 2.18 Project Setting Screen in Xcode

2.4.2 Firmware file transfer

There are two ways to transfer the update firmware to your iOS device:

- Transfer from Finder or iTunes.
- Transfer from iCloud

Firmware stored in FWU_CLIENT can be checked from the iOS "Files" app.

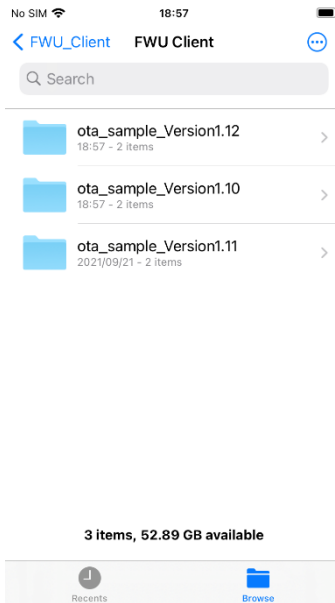


Figure 2.19 Confirmation of update file by "Files" app

2.4.2.1 Transfer from Finder or iTunes

Connect the PC and iOS device wired. Drag and drop and forward the “FWU Client” folder to the FWU_Client app area of the Finder or iTunes Files tab. The “FWU Client” folder contains the folder that contains firmware.bin and firmware_information.json, which is the update firmware

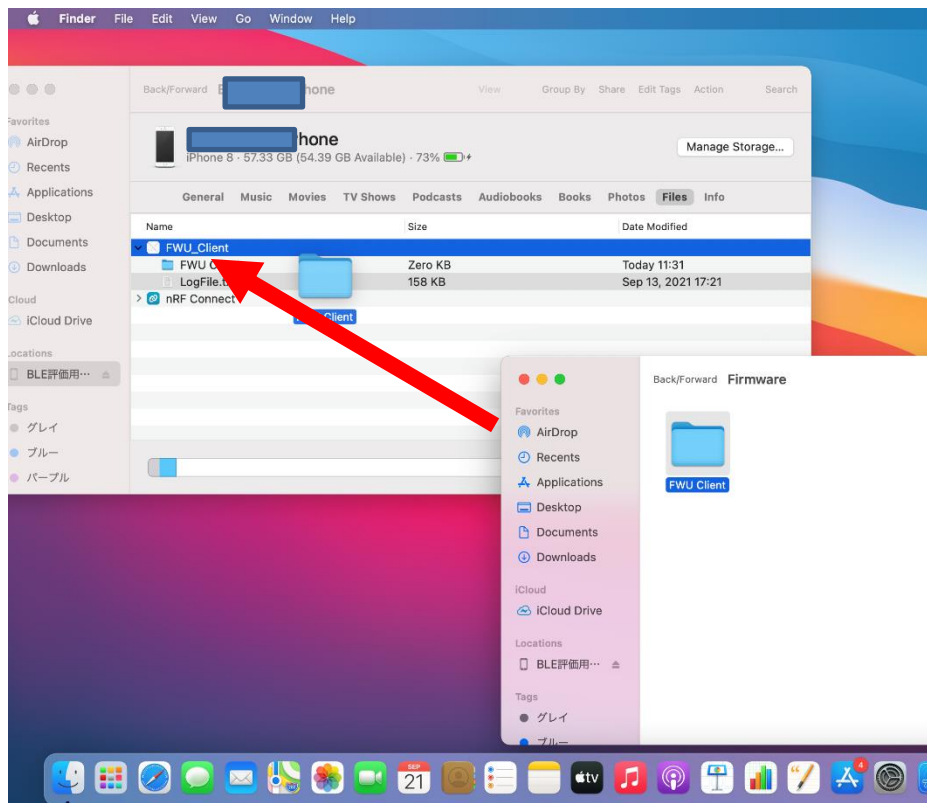


Figure 2.20 Transfer of update firmware using Finder.

2.4.2.2 Transfer from iCloud

Access the iCloud of Apple ID registered on the iOS device. Upload the FWU Client folder to any directory on iCloud.

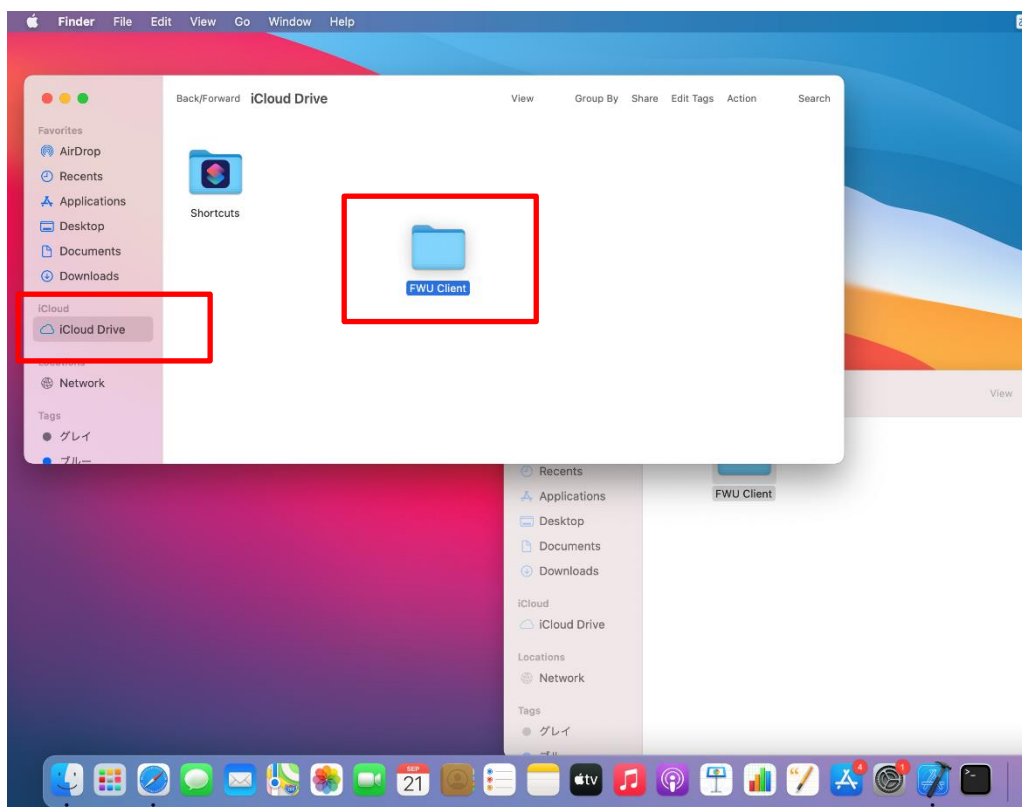


Figure 2.21 Transfer file to iCloud

From the "Files" app of the iOS device, access iCloud and check the file for update. Tap the download button and store it on the device.

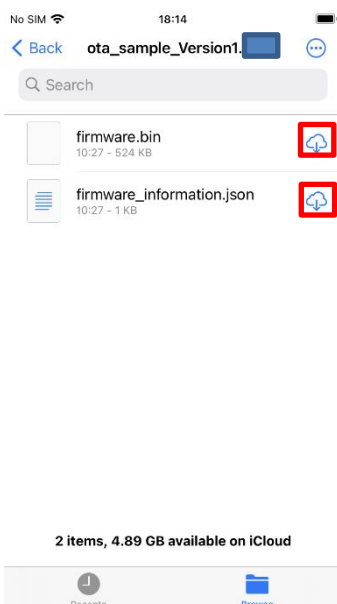


Figure 2.22 Download File from iCloud to Device

Launch FWU_Client. Tap iCloud Update from the + button and select the transferred firmware folder for transferred.

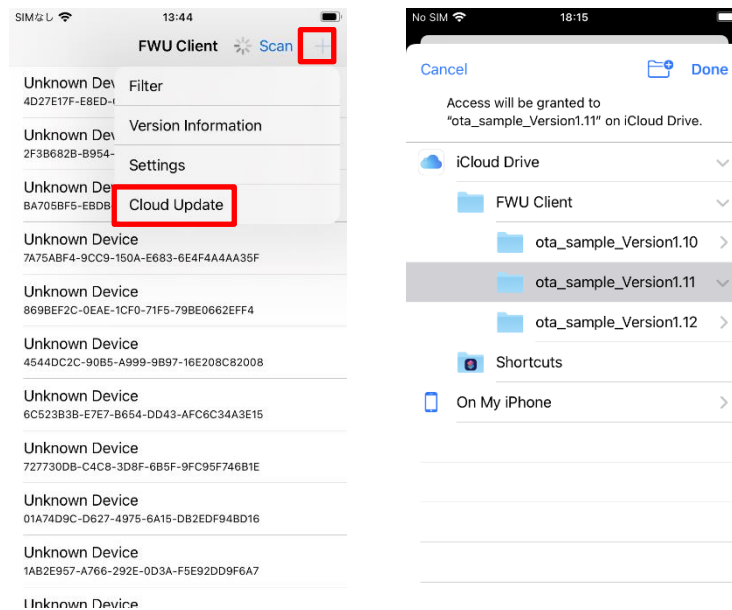


Figure 2.23 Copy from iCloud to FWU_Client area

The selected firmware folder is copied to the FWU_Client app area. This operation can be performed after the connection device information screen. Please see Section 2.3.

2.4.3 Firmware update procedure

The way to perform the update is the same as the Android version. The update time of the application part by the iOS version FWU_Client is about 30 seconds. This version of OTA Server performs Indication of Service Changed Characteristic after connection to delete the IOS GATT Database cache. Because the iOS version GATTBrowser should disconnect when receiving the service changed characteristic indication, use the Android version GATTBrowser to check the update.

2.4.4 Confirmed device

Table 2.4 Operation confirmation device

Device name	iOS Version
iPhone 8	14.7.1

2.5 Update by FWU_Client for Python

Python version FWU_Client uses the PC and Target Board for RX23W to provide OTA Server firmware updates. This section provides setup and update instructions for the following programs included in this sample package:

- Python Application Program (Python Application)
- OTA Client program (ble_sample_tbrx23w_ota_client)

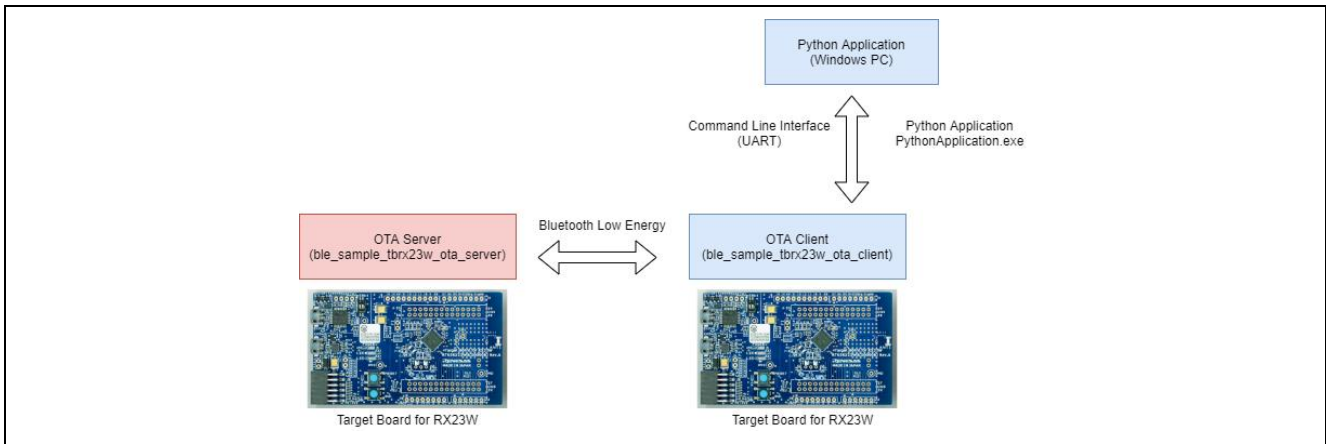


Figure 2.24 Communication Architecture using a PC and another Target board for RX23W

2.5.1 OTA Client RX23W setup

For the setup procedure of the peer RX23W, refer to the procedure described in “2.2 OTA Server setup” and write the OTA Client firmware file "ble_sample_tbrx23w_ota_client.mot". OTA Client receives update firmware from Python Application running on a PC.

2.5.2 Python Application setup

Perform Python Application requires a pySerial module. Install using pip etc.

Edit "port.json" in the Python Application folder and set the port number of the OTA Client device.

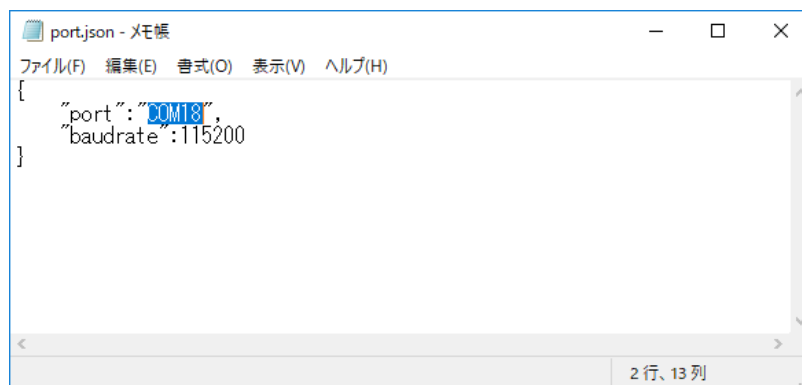


Figure 2.25 Port number setting in port.json

2.5.3 Firmware update procedure

Launch "main.py". Execute the commands shown in the red frame in order. Enter the BD address of the OTA Server to be used in the blue frame "df:3c:31:5d:08:d9 1" of the conn command. It will be displayed on the "BD ADDRESS" and "BD Address Type" after scanning.

```
C:\¥r01an5910xx0110-rx23w-otafwup¥PythonApplication>python main.py
Enter Command
scan 1 1 9 FWU-LEV
BD ADDRESS : df:3c:31:5d:08:d9
BD Address Type : 1
Advertise Raw Data : 02,01,06,08,09,46,57,55,2d,44,45,56

BD ADDRESS : df:3c:31:5d:08:d9
BD Address Type : 1
Advertise Raw Data : 08,09,46,57,55,2d,44,45,56

SCAN END
Enter Command
conn 5 df:3c:31:5d:08:d9 1
ConnectionSuccess
BD ADDRESS : df:3c:31:5d:08:d9
ProjectName : ota_sample
ApplicationVersion : 01.10
DownloaderVersion : 01.10
RelocatorVersion : 01.00
BLE Protocol Stack version : 02.20
Server Activate Mode : User Application Activate

Enter Command
update app C:\¥r01an5910xx0110-rx23w-otafwup¥SampleFirmware¥FWU Client¥ota_sample_Version1.11
address: fff90800 progress : 100.0% 017/017
UPDATE SUCCESS
Update Time: 17.73 sec
Enter Command
```

Figure 2.26 Python Application execution screen

2.5.3.1 How to confirm the completion of the update

Run the conn command again and connect to the OTA Server. Make sure that Application Version has been updated.

```
Update Time: 17.73 sec
Enter Command
conn 5 df:3c:31:5d:08:d9 1
ConnectionSuccess
BD ADDRESS : df:3c:31:5d:08:d9
ProjectName : ota_sample
ApplicationVersion : 01.11
DownloaderVersion : 01.10
RelocatorVersion : 01.00
BLE Protocol Stack version : 02.20
Server Activate Mode : User Application Activate
Enter Command
disconn
Disconnected
Enter Command
exit
Serial Port Closing...
```

Figure 2.27 Confirmation of Update Completed

Connect from GATTBrowser and check the addition of Battery Service

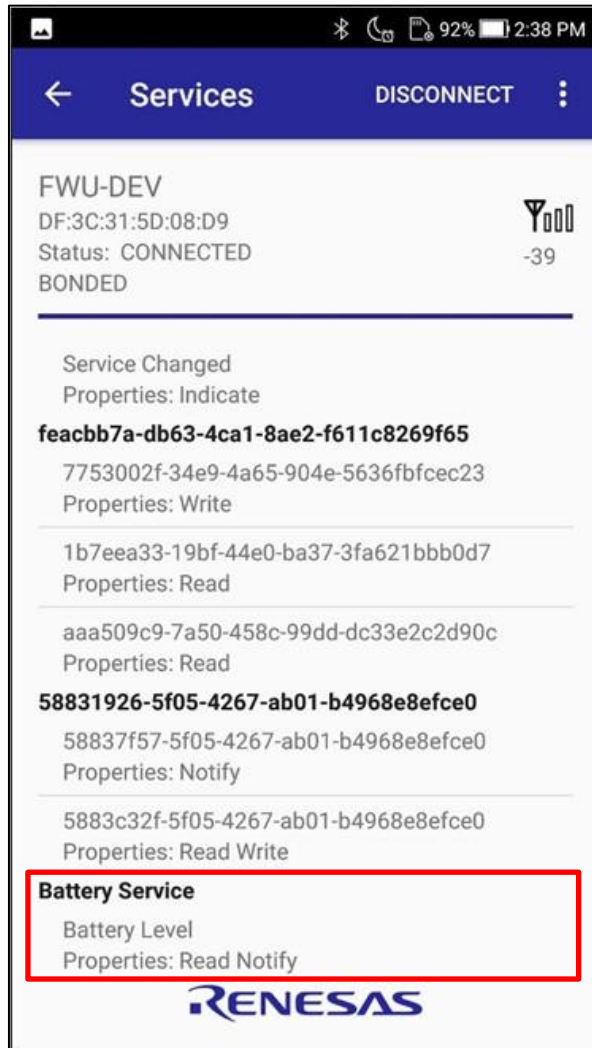


Figure 2.28 Confirmation of adding Battery Information Service

3. OTA firmware update feature

This chapter details the OTA firmware update function of OTA Server provided by the sample program. This sample realizes firmware update by OTA using three independent programs and a boot loader. The BLE Protocol Stack is shared by the user application and the firmware receiver (downloader).

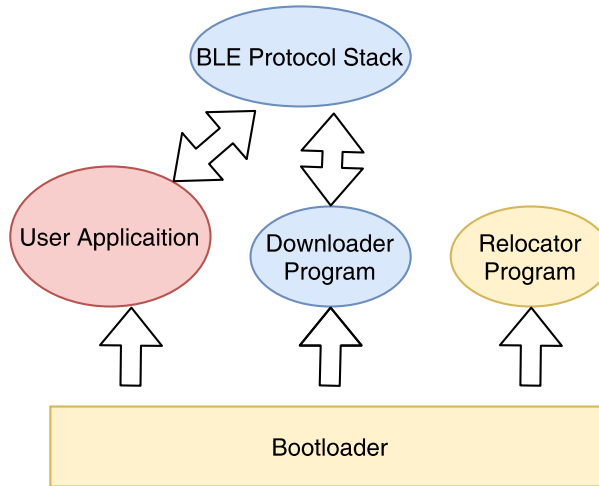


Figure 3.1 Software overview of OTA firmware update feature

- **User application**
The user application is a Bluetooth LE application during normal operation.
- **Downloader**
The downloader uses the BLE protocol stack to receive the update firmware from the OTA Client. The downloader holds the received firmware in the code flash.
- **Relocator**
The relocator writes to the correct area for executing the firmware received by the downloader.
- **Bootloader**
The bootloader reads the flag information of the boot program written to the data flash and executes the program.

The OTA firmware update feature of this sample has two update modes: application-only update and Bluetooth LE communication section and application update. The following is a simplified diagram of the firmware update by OTA.

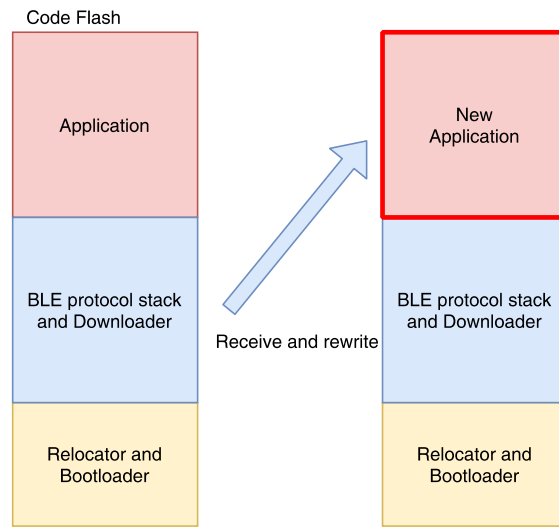


Figure 3.2 Update only the application section

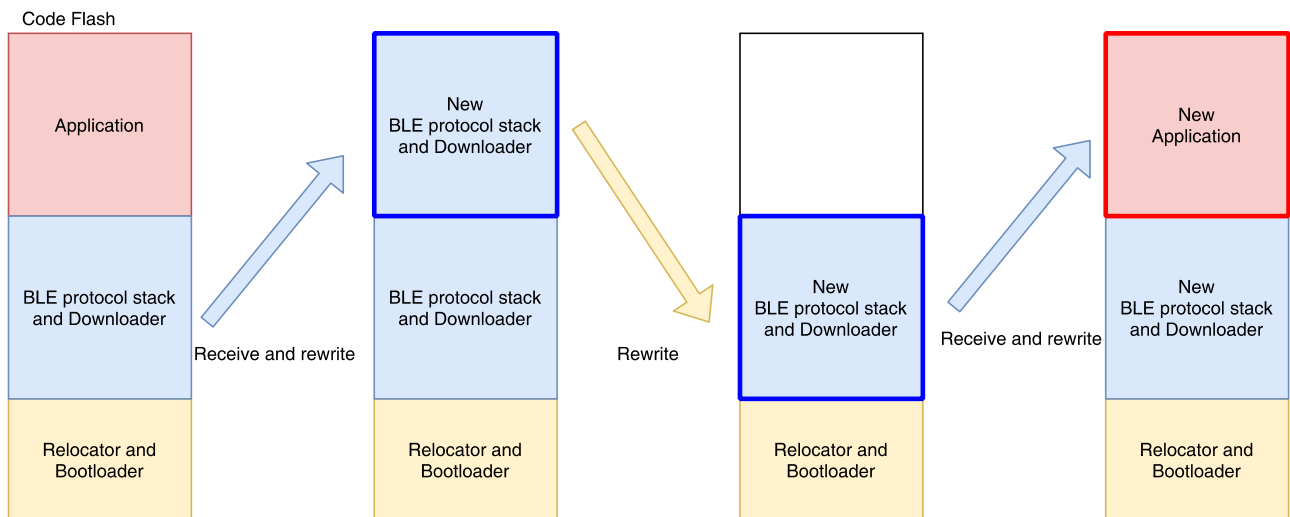


Figure 3.3 BLE protocol stack section and application section update

The BLE protocol stack section can be updated by sharing the application section as a temporary section when updating the BLE protocol stack.

The following are three programs transition diagrams for updating the firmware by OTA sever. The user application starts updating from the OTA client using the Renesas OTA Reset Service on GATT. The downloader uses the Renesas OTA Service to receive the firmware for updates from the OTA client. The downloader launches the relocator and relocates the firmware to the correct section on the code flash. After that, in the case of updating only the application, start the user application. In the case of updating the BLE protocol stack, launch the downloader again and receive the application section.

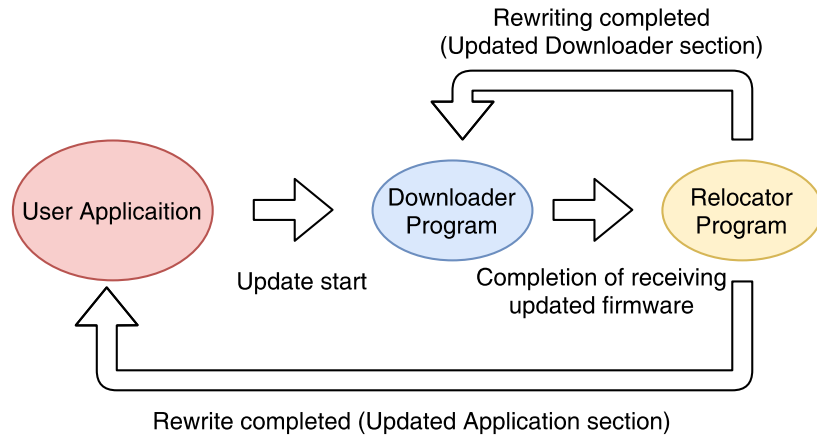


Figure 3.4 Activate programs transition diagram during firmware update

3.1 System structure

Shows the system structure of the OTA firmware update sample program.

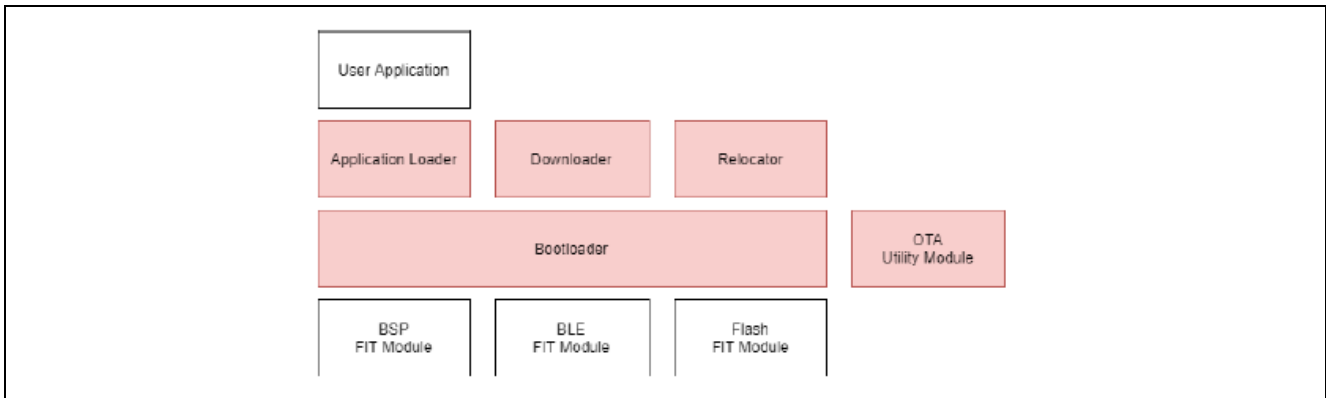


Figure 3.5 System structure of OTA firmware update sample program

See below for details on the FIT module used by the OTA firmware update feature.

- RX Family Board Support Package (BSP) FIT Module
<https://www.renesas.com/document/apn/rx-family-board-support-package-module-using-firmware-integration-technology>
- RX23W Group BLE FIT Module
<https://www.renesas.com/document/apn/rx23w-group-ble-module-firmware-integration-technology-application-note>
- RX23W Group Flash FIT Module
<https://www.renesas.com/document/apn/rx-family-flash-module-using-firmware-integration-technology>

Table 3.1 Package structure of OTA firmware sample program

Peripheral functions	Detail	Purpose of use
Bluetooth Low Energy	-	Perform OTA firmware update sequence
Data Flash	Block 1	Retention program flag information for bootloader
CMT	Channel 0	Timeout monitoring during OTA firmware update sequence. Used in downloader. Does not limit its use in user applications.

3.2 Section layout

Figure 3.6 shows the ROM/RAM section configuration of the OTA firmware update sample program. Each program is placed in separate sections on the code flash to support self-programming capabilities. RAM uses section overlay functions to share the area of 0x00001404 to 0x0000BA00 with each program.

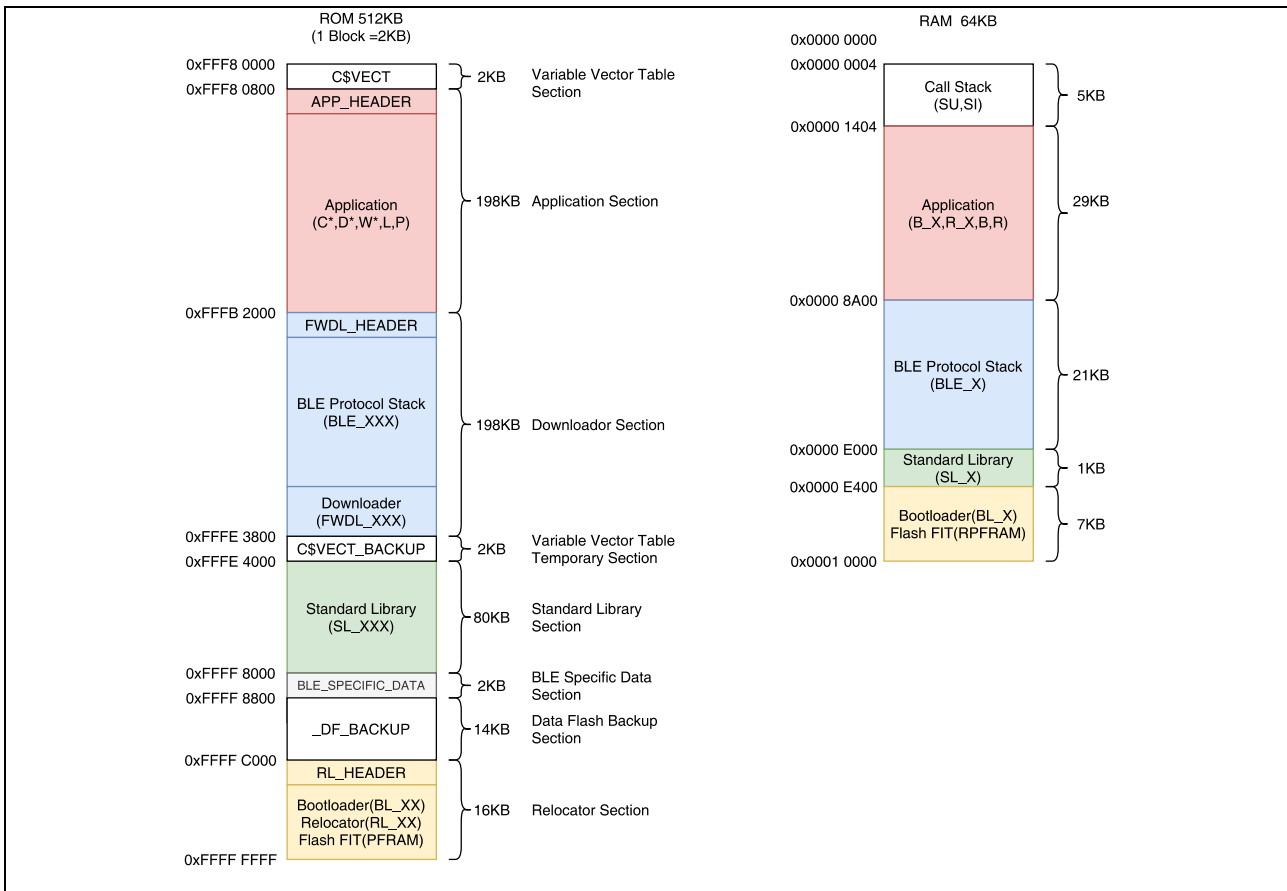


Figure 3.6 ROM/RAM section layout of OTA firmware update sample program

3.3 Firmware update operation

The OTA firmware update feature can be updated in any of the following operation modes.

- Application update mode: User application only
- BLE protocol stack update mode: Both user application and Bluetooth Low Energy communication section

3.3.1 Application update mode

The operation sequence of application update mode is shown. In this mode, the application section and variable vector table section are updated. The downloader writes the application section portion of the updated firmware to the application section and the variable vector table to the C\$VECT_BACKUP section. The relocater is relocated from the C\$VECT_BACKUP section to the C\$VECT section. The relocater then launches the application program.

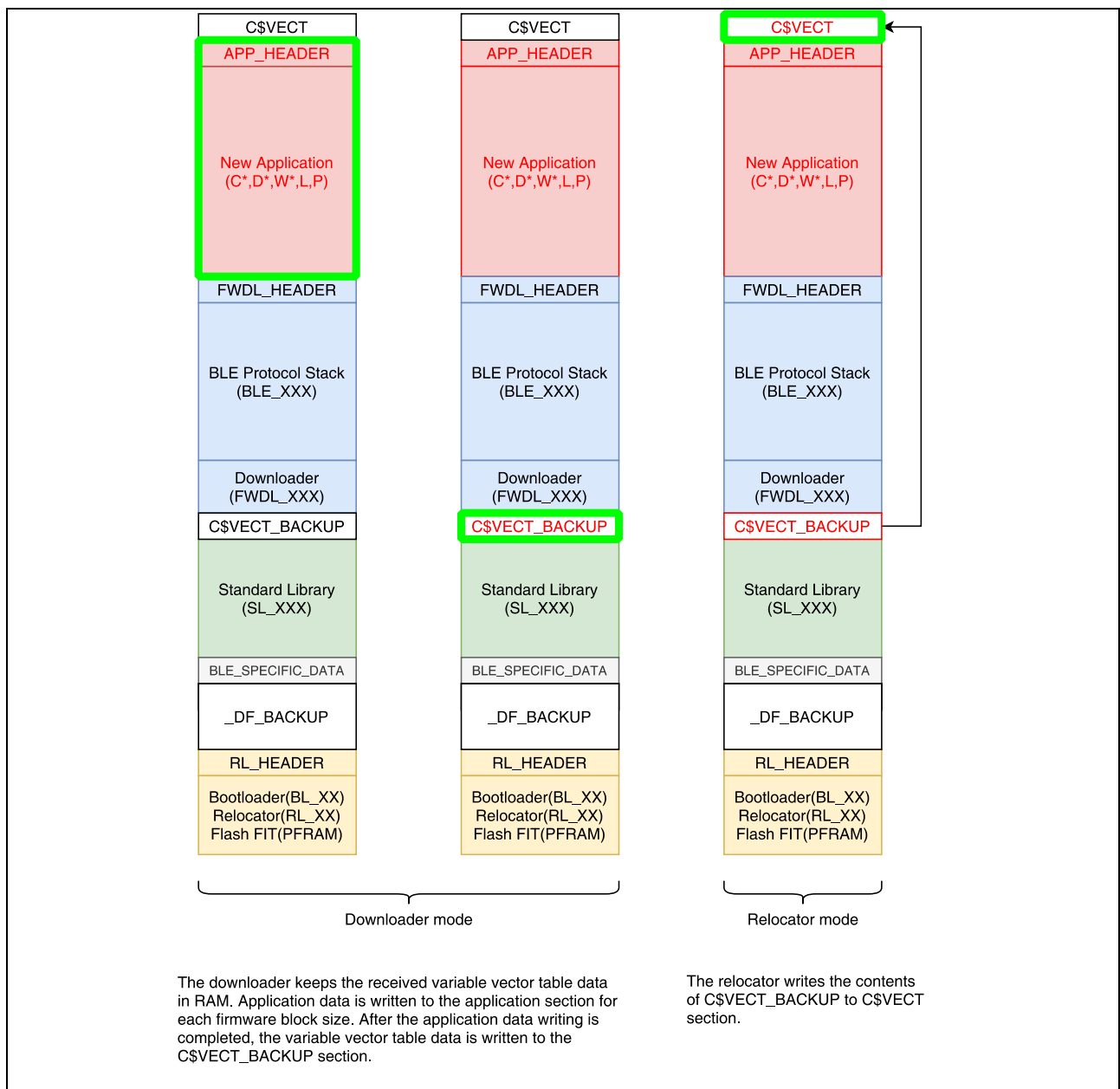


Figure 3.7 OTA firmware update (update only user application)

3.3.2 BLE protocol stack update mode

The operation sequence of BLE protocol stack update mode is shown. The downloader writes the BLE protocol stack and downloader to the Application section, writes the variable vector table to the C\$VECT_BACKUP section, and the relocators relocate them to the Downloader and C\$VECT sections, respectively. Then launch the updated downloader and enter application update mode.

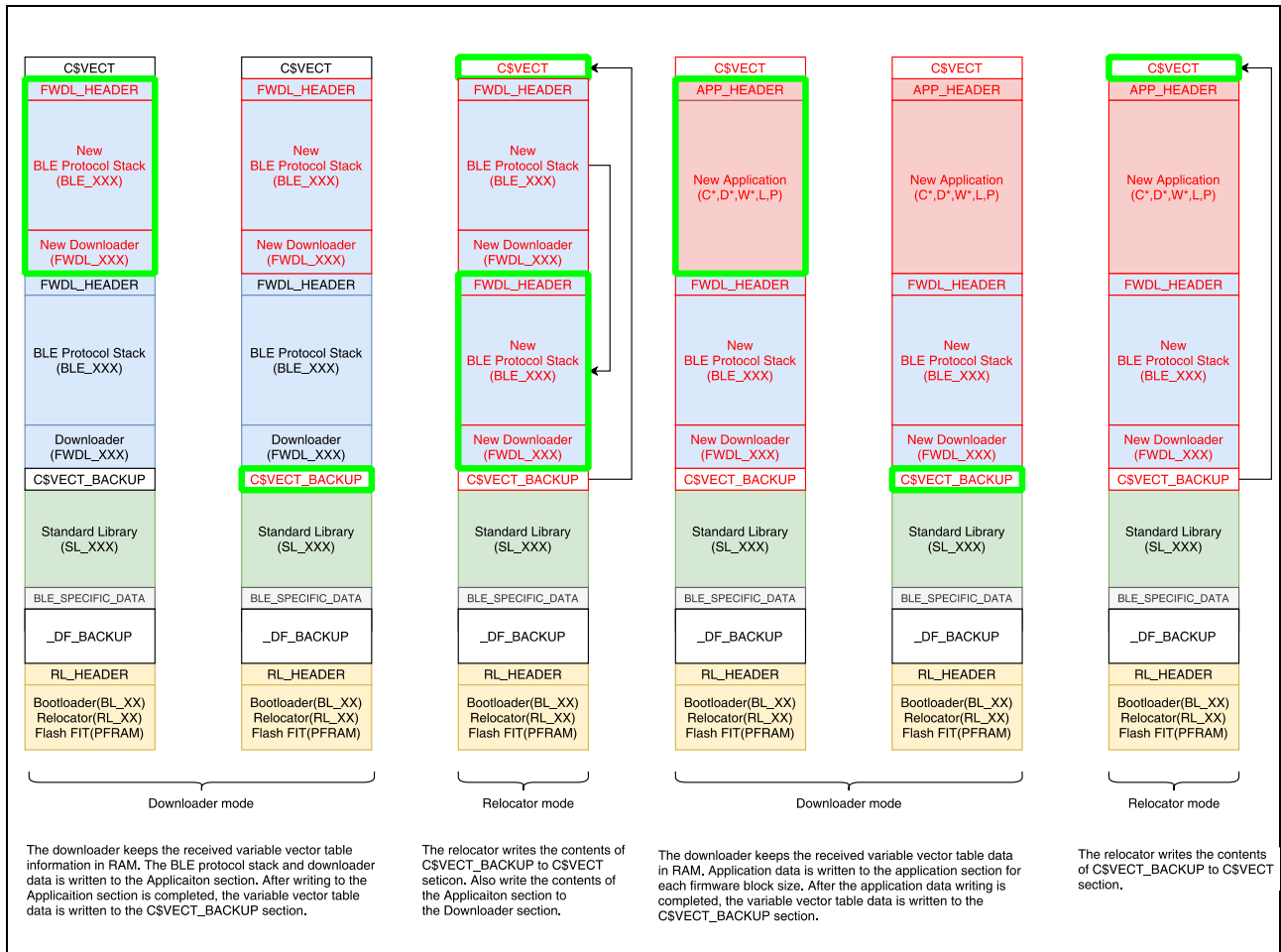


Figure 3.8 OTA firmware update (BLE protocol stack update mode)

3.4 Profile specifications

In OTA firmware update, communication is performed using the following two GATT services. The MTU size is assumed to be 247 bytes to reduce the update time.

- Renesas OTA Reset Service
- Renesas OTA Service

The Renesas OTA Reset Service is used by embedding it in the user application. This service provides firmware version information and project information to the OTA Client. It also provides a function to switch to downloader by OTA.

Renesas OTA Service is used in downloader. This service defines the extraction of firmware information and the transfer destination of the update firmware.

3.4.1 Renesas OTA Reset Service

Table 3.2 Service overview

Service Name	Type	UUID
Renesas OTA Reset Service	Primary Service	feacbb7a-db63-4ca1-8ae2-f611c8269f65

Characteristic Name	Property	Permission	Description	UUID
Virtual Reset Button Characteristic	Write Only	Encryption	18byte, Reset Code (18 bytes strings)	7753002f-34e9-4a65-904e-5636fbfcec23
Project Information Characteristic	Read Only	Encryption	variable length Project Name (20 bytes strings)	1b7eea33-19bf-44e0-ba37-3fa621bbb0d7
Version Information Characteristic	Read Only	Encryption	Version information (8 byte)	aaa509c9-7a50-458c-99dd-dc33e2c2d90c

3.4.2 Renesas OTA Service

Table 3.3 Service overview

Service Name	Type	UUID
Renesas OTA Service	Primary Service	9d5998f8-105b-4691-92be-4b1b4d3ee8bb

Characteristic Name	Property	Permission	Description	UUID
Data Control Characteristic	Write, Indication	Encryption	Characteristic for exchanging information with the server when the client sends the firmware. The first byte is the command. The following is the data corresponding to the command.	629c8ef7-aa42-4f1e-8330-fe832961b926
Data Transfer Characteristic	Write Without Response	Encryption	Used when receiving Firmware. Send the firmware corresponding to the address shared by the Data Control Characteristic command.	13561280-ecb3-4691-9ab0-33649c7e03db

3.4.3 Data format

Table 3.4 shows the error codes used by Renesas OTA Reset Service and Renesas OTA Service.

shows the OTA commands executed by Data Control Characteristic. All integer types are specified by Little Endian.

Table 3.4 Error codes used by Renesas OTA Service and Renesas OTA Reset Service

Error Code	Description
OTA ERROR INVALID ADDRESS (0x81)	This error code occurs in the following cases. <ul style="list-style-type: none"> When executing the Firmware Download Start command, the start addresses of the sections sent and received by OTA Server and Client do not match.
OTA ERROR INVALID LENGTH (0x82)	This error code occurs in the following cases. <ul style="list-style-type: none"> The size specified when executing the section update command exceeds the section Different from the specified size
OTA ERROR CHECK SUM (0x83)	This error code occurs in the following cases. <ul style="list-style-type: none"> The checksum sent by the Client and the checksum calculated by the OTA Server are different when the Firmware Download End and each Update End command are executed.
OTA ERROR RESET CODE (0x84)	This error code occurs in the following cases. <ul style="list-style-type: none"> The Reset Code written to the Virtual Reset Button Characteristic does not match that of the OTA Server.
OTA ERROR PROJECT INFORMATION (0x85)	This error code occurs in the following cases. <ul style="list-style-type: none"> The project name sent in the Project Information Request does not match when the Variable Vector Update Start command is executed.
OTA ERROR INVALID SECTION (0x86)	This error code occurs in the following cases. <ul style="list-style-type: none"> In each Update Start or Update End command, the section start address does not match that of the OTA Server.
OTA ERROR INVALID VERSION (0x87)	This error code occurs in the following cases. <ul style="list-style-type: none"> When the version information specified in the Version Information Request cannot be updated when the Application Update Start and BLE Downloader Start commands are executed.
OTA ERROR INVALID CMD (0x90)	This error code occurs in the following cases. <ul style="list-style-type: none"> When the OTA Server is in a state where it does not accept the executed command.

Table 3.5 Data Control characteristic commands

Command name	Data structure	Description
VariableVectorTableUpdateStart (0x01)	0-3byte: Variable vector table start address 4-7byte: Variable vector table size	Starts sending firmware data for the variable vector table.
VariableVectorTableUpdateEnd (0x02)	0-3byte: Variable vector table start address 4-7byte: Variable vector table size 8-11byte: Checksum	Notifies that the transmission of the firmware data of the variable vector table is completed. A checksum of the sent firmware data is sent to check the validity of the firmware.
ApplicationUpdateStart (0x03)	0-3byte: Application section start address 4-7byte: Application section size	Start checking the section information of the application section and sending the firmware data. Firmware data is sent after the Firmware Download Start command.
ApplicationUpdateEnd (0x04)	0-3byte: Application section start address 4-7byte: Application section size 8-11byte: Checksum	Notifies that the transmission of firmware data in the application section has been completed. A checksum of the sent firmware data is sent to check the validity of the firmware.
BLEDownloaderUpdateStart (0x05)	0-3byte: Downloader section start address 4-7byte: Downloader section size	Start checking the section information of the downloader section and sending the firmware data. Firmware data is sent after the Firmware Download Start command.
BLEDownloaderUpdateEnd (0x06)	0-3byte: Downloader section start address 4-7byte: Downloader section size 8-11byte: Checksum	Notifies that the downloader section firmware data has been sent. A checksum of the sent firmware data is sent to check the validity of the firmware.
FirmwareDownloadStart (0x07)	0-3byte: Download target section start address 4-7byte: Download target section size	The firmware data of the download target section specified by each Update Start command is sent by dividing it into the size (firmware block) specified by this command.
FirmwareDownloadEnd (0x08)	0-3byte: Download target section start address 4-7byte: Download target section size 8-11byte: Checksum	Notifies that the firmware data specified by the Firmware Download Start command has been sent.
ProjectInformationRequest (0x09)	0-18byte: Project name to be updated	Sends the project name of the firmware that the Client updates. The OTA Server sends the Project Information Response command after receiving this command.
ProjectInformationResponse (0x0A)	0-18byte: OTA Server project name	Responds to the firmware project name written to the OTA Server.
VersionInformationRequest (0x0B)	0-1byte: Application section version 2-3byte: Downloader section version 4-5byte: Relocator section version 6-7byte: BLE Protocol Stack version	Sends firmware version information that the Client updates. The OTA Server sends the Project Information Response command after receiving this command.
VersionInformationResponse (0x0C)	0-1byte: Application section version 2-3byte: Downloader section version 4-5byte: Relocator section version 6-7byte: BLE Protocol Stack version	Responds to the firmware version information written to the OTA Server.

3.5 OTA communication sequence

The OTA Server receives and updates the firmware according to the following sequence. The OTA Client and smartphone application (FWU_Client) execute this sequence.

- User application

In the user application, check the version information and project name, and switch to the downloader program. The communication sequence is shown below.

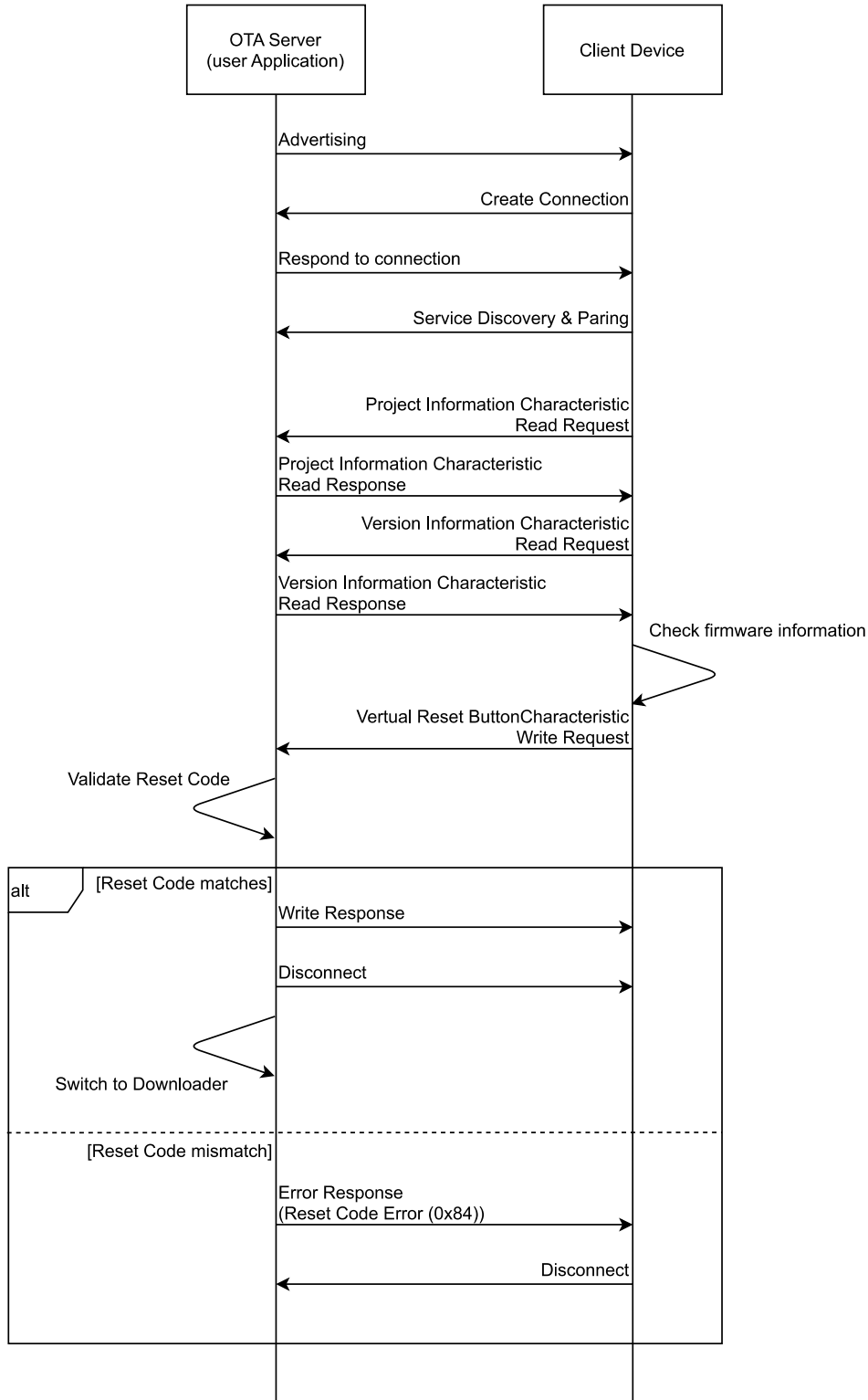


Figure 3.9 Program switching sequence using OTA Reset Service in user application

- Downloader
 Downloader receives firmware for application section updates or downloader section updates. For both updates, the variable vector table is downloaded first, followed by the reception of the specified section. The communication sequence is shown below.

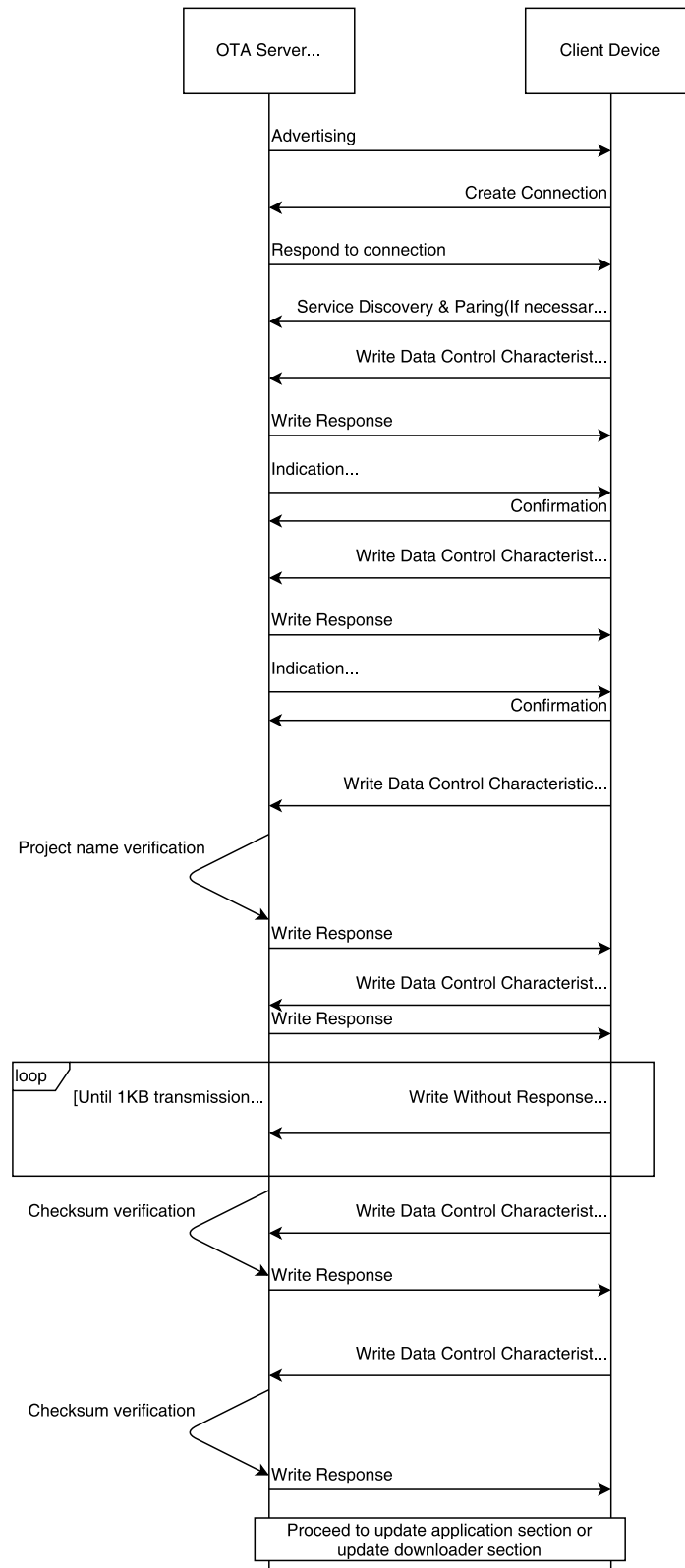


Figure 3.10 Receiving variable vector table

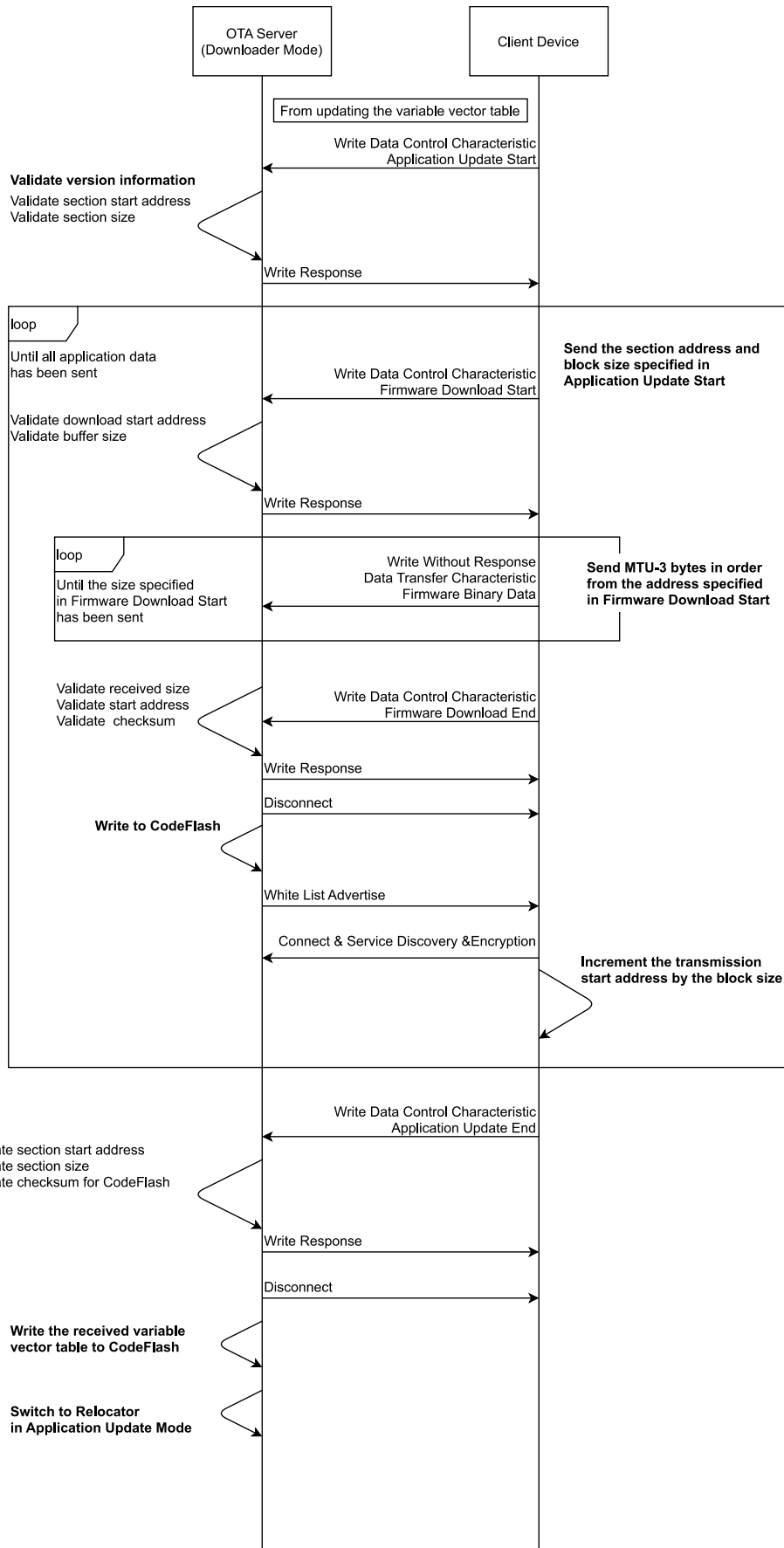


Figure 3.11 Receive application section

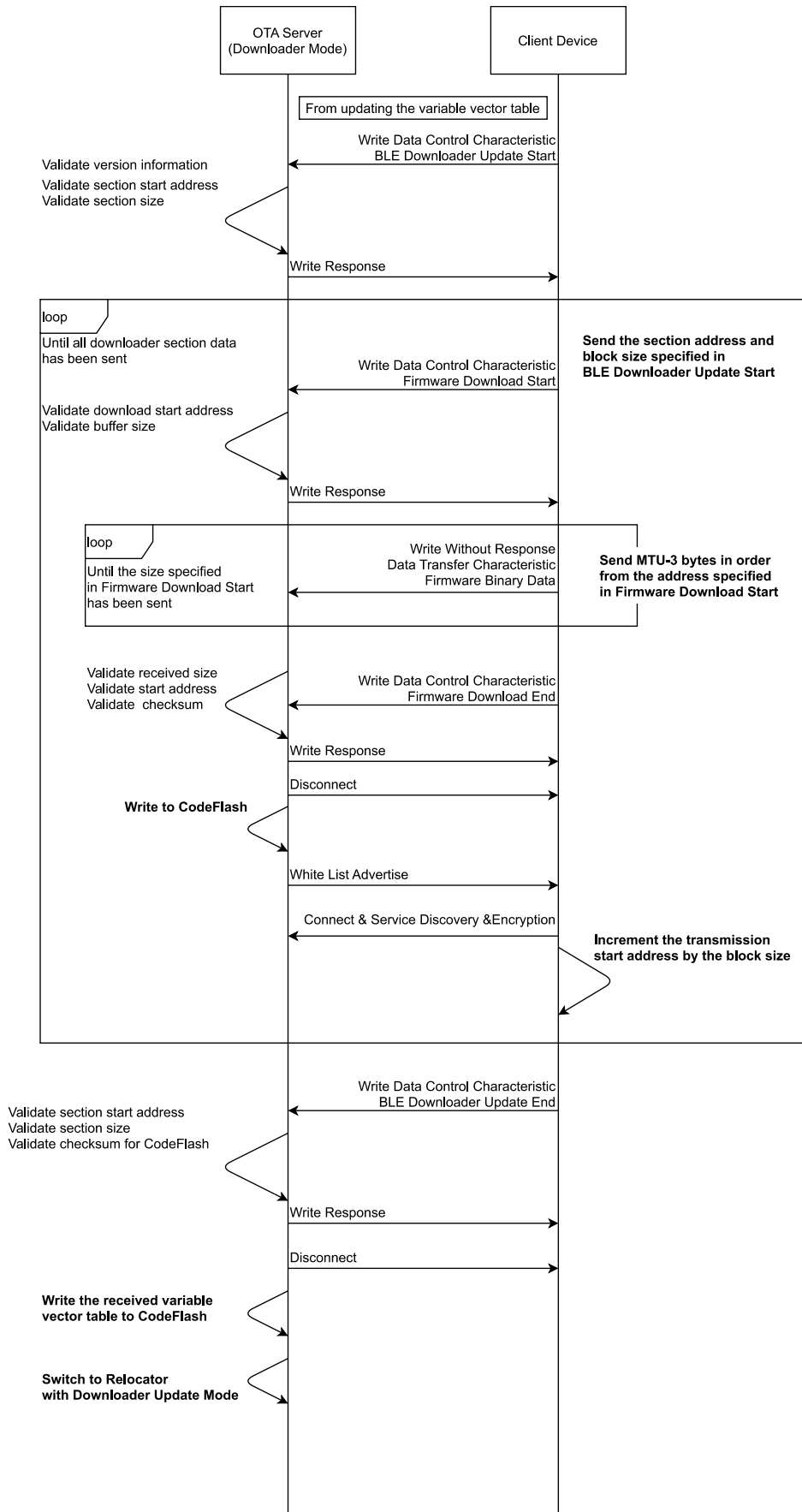


Figure 3.12 Receive downloader section

If an inappropriate value is written to the Data Control Characteristic, OTA Server returns an Error Response. If the Client Device receives a check sum error for the Firmware Download End command, Client Device can start over with the Firmware Download Start command. If any other Error Response is received, the Client Device disconnects from the OTA Server and ends the update.

3.6 Switching startup programs

To switch the startup program, write the flag information of the execution program in the two areas of Data Flash and Code Flash, and reset the MCU. By writing the flag information in two independent areas, it protects from the operation stop due to the power cutoff. The switching sequence of the startup program is shown below.

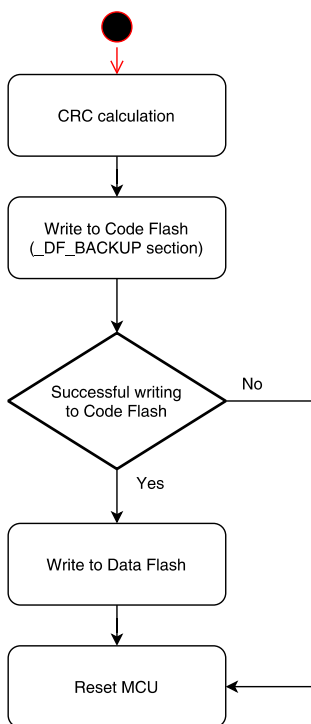


Figure 3.13 Startup program switching process

The flag information of the startup program is shown below.

Table 3.6 Startup program flag information

Parameters	Type	Description
crc	uint16_t	This is the result of CRC calculation for the area from activate_mode to the end of unique_code. Used to validate flag information.
activate_mode	e_ble_ota_activate_mode_t	Specify the startup program.
status_info source code	st_ble_ota_dataflash_status_t - e_ble_ota_activate_mode_t - e_ble_ota_status_t	Information parameters to pass to the startup program. - source: Program information that was being executed - code: Information to pass to the startup program
unique_code[12]	uint8_t	It is defined to provide redundancy in the data when calculating the CRC.

3.7 Bootloader

The bootloader selects the OTA Server boot program based on the flag information of the executable program held in Data Flash. The bootloader realizes the following two functions.

- Reading the flag information of the startup program and checking the validity
- Program execution based on flag information

The flow chart of the boot loader is shown below.

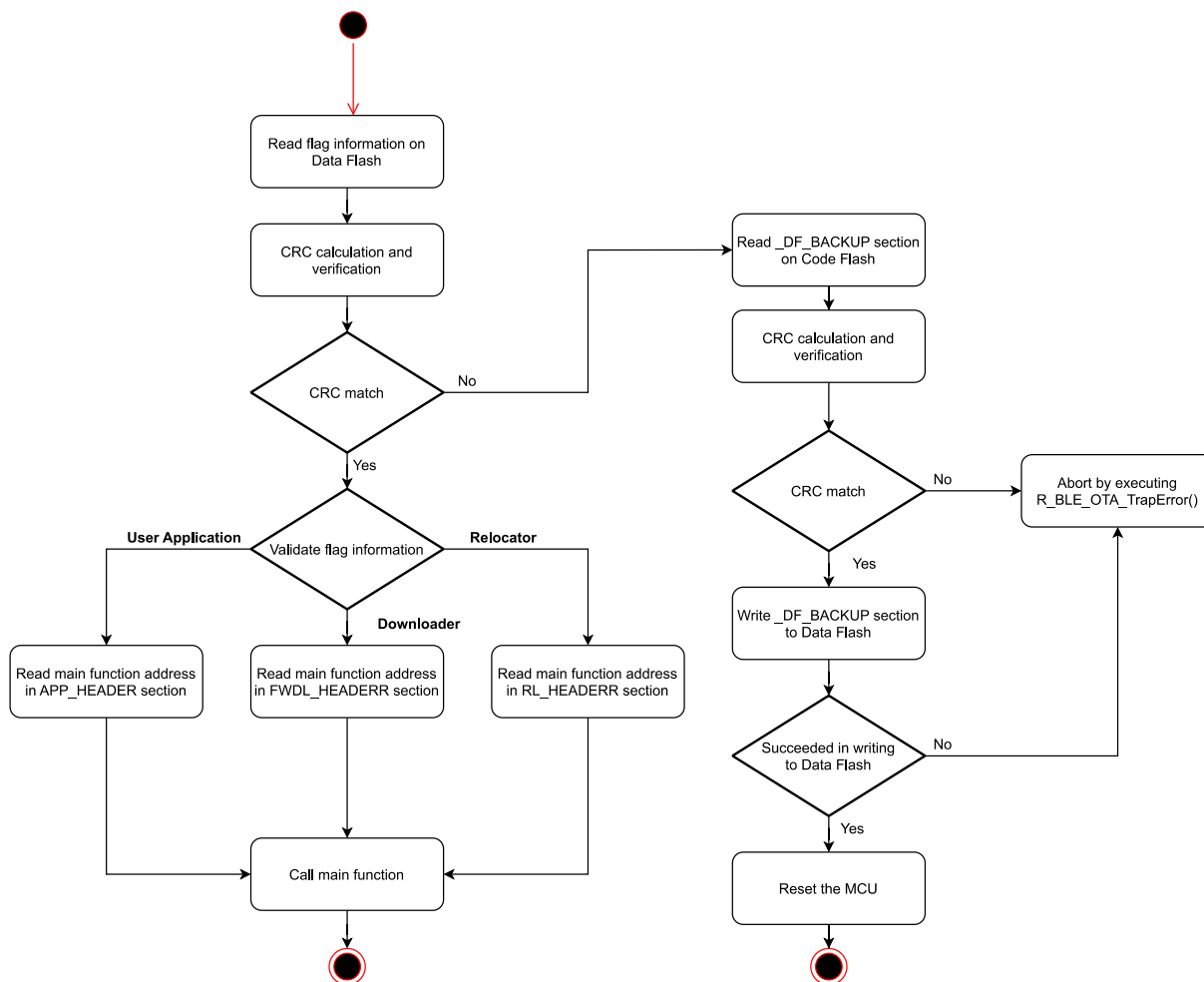


Figure 3.14 Bootloader flowchart

The flag information of the startup program is written in two places in the `_DF_BACKUP` section of Data Flash and Code Flash so that it will not be erased by power interruption during writing.

Read the flag information of Data Flash, calculate the CRC, and check if it matches the flag information. If the CRCs do not match, read the `_DF_BACKUP` section and validate with the CRC. If the CRC in the `_DF_BACKUP` section matches, copy the flag information in the `_DF_BACKUP` section to Data Flash and Reset. If the CRC in the `_DF_BACKUP` section does not match, it becomes infeasible and calls the `R_BLE_OTA_TrapError` function to stop the program.

After checking the validity of the Data Flash flag information, read the startup information and execute the main function of each program described in the HEADER section of each section. The user application main function is called via the `application_loader` function in the `r_ble_ota_app_loader.c` file.

3.8 Relocator

3.8.1 Program Operations

The relocator relocates the firmware received by the downloader from the firmware temporary retention section to the correct section. The type of firmware that the downloader wrote to the firmware temporary hold section is included in the flag information written to Data Flash.

The operation flowchart of the relocator is shown below.

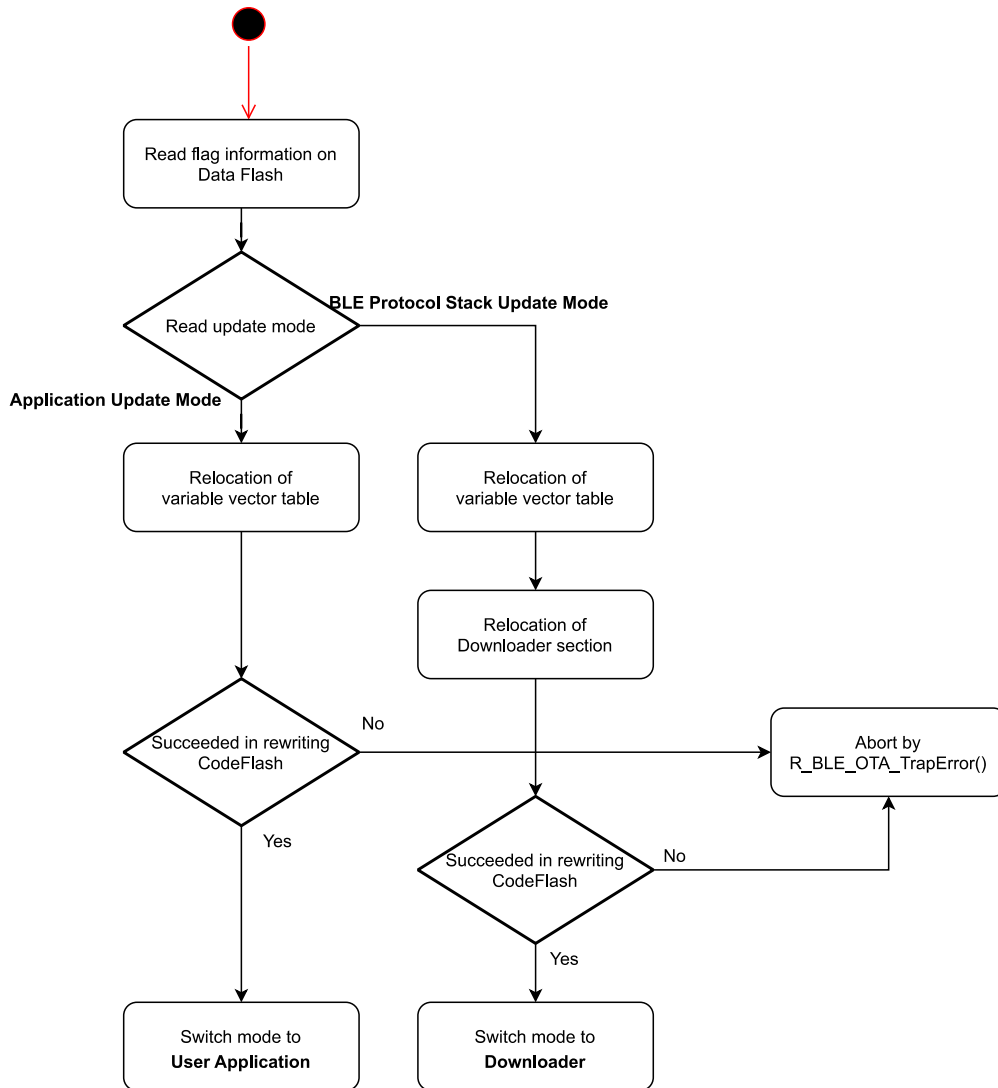


Figure 3.15 Relocator flowchart

When updating the application section, the firmware temporary retention section and the application section match, so only the variable vector table is relocated. When updating the downloader section, relocate the variable vector table and the firmware temporary retention section to the downloader section.

After relocating the firmware, in the case of an application update, set the program to be executed next to the user application, and in the case of updating the downloader section, set the program to be executed next to the downloader.

3.8.2 Operation when power is cut off

The relocater relocates the variable vector table and the program in the downloader section. The relocater does not change the section before rewriting. Therefore, even if a power loss occurs during the relocation, the sequence shown in Figure 3.15 will be executed from the beginning.

3.9 Downloader

In the downloader, the OTA Server receives the update firmware from the Client. For the downloader, refer to "3.3 Firmware update operation" and "3.5 OTA communication sequence".

When the OTA Server switches from the user application to downloader, the user application will not run until the update is complete.

3.9.1 Bluetooth LE operation

The downloader acts as a GAP Peripheral and advertises, including the UUID of the Renesas OTA Service, at 30msec intervals. The downloader Advertise settings are implemented in `r_ble_ota_ble_interface.c`.

During the downloader operation, the MCU is reset at the following timing.

- When 10 seconds have passed since the start of Advertising
- When there is no writing to Data Control Characteristic for 10 seconds after connecting from Central
- When disconnected from Client

3.9.2 Bonding information management

The downloader does not keep the pairing information in the non-volatile area. Uses the bonding information in the non-volatile area written by the security library of `app_lib`.

The downloader registers in the Resolving List based on the pairing information in the non-volatile area, but at the time of pairing, the IRK and address of the opposite device are not registered in the Resolving List. If you use an RPA device, be sure to bond it while the user application is running.

3.9.3 Writing to Code Flash

The downloader writes the received firmware to Code Flash for each firmware block size. The variable vector table is written to the variable vector table backup section, and the firmware of the application section and downloader section is written to the firmware temporary retention section.

Since Code Flash cannot be read while writing Code Flash, the Bluetooth LE connection is disconnected, and writing is performed.

3.9.4 Operation when power is cut off

If a power outage occurs while the downloader is running, OTA Server will run the downloader again.

Resume the update sequence from the beginning of Figure 3.10 Receiving variable vector table, even if the firmware is in the process of being downloaded.

3.10 User application

The user application is the application part by OTA Server. It will be started after the update by OTA.

3.10.1 Switching to downloader

To initiate an OTA firmware update from a user application, switch the startup program running to downloader.

To switch to downloader by OTA, use Renesas OTA Reset Service. The switching sequence from the user application to the downloader is shown below.

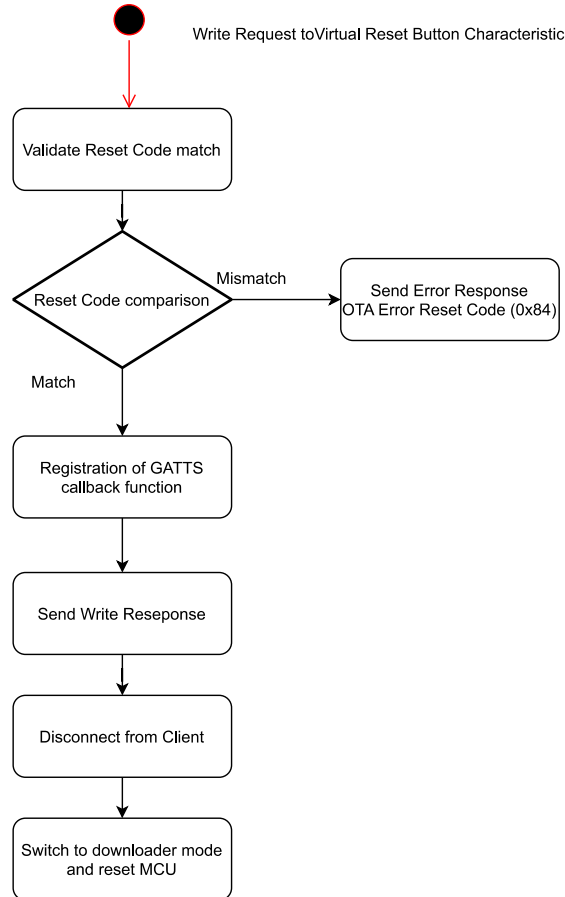


Figure 3.16 Relocator flowchart

3.10.2 Sample program user application

The OTA Server included in this package implements the Peripheral sample included in the Application Developer’s Guide(R01AN5504) in the user application part.

4. Create an OTA update project

This chapter shows how to add OTA update functionality to an existing user project. In this sample, only "Balance Library" of BLE Protocol Stack is supported. It is assumed that the BLE FIT module has been added to the existing project.

4.1 Add source code (r_ble_ota)

Copy the "r_ble_ota" folder from the sample OTA Server project "ble_sample_tbrx23w_ota_server" to an existing project.

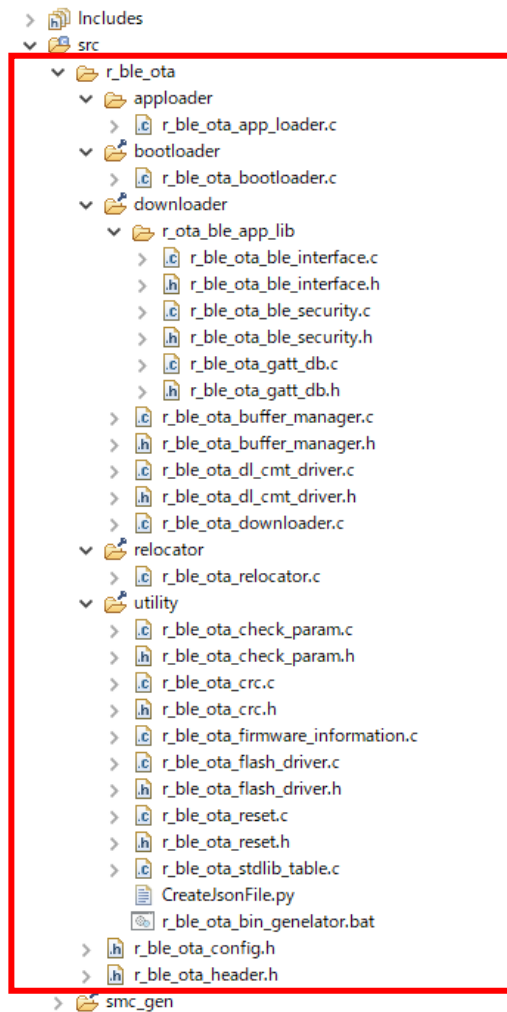


Figure 4.1 r_ble_ota folder

Register the path of the copied folder in the existing project. Right-click on the project and add the `r_ble_ota` and `r_ble_ota / downloader` folders to the "C/C++ Settings"->"Tool Settings"->"Compiler"->"Source"->"Find Include Files Folder (-include)" please.

```
"${workspace_loc}/${ProjName}/src/r_ble_ota)"
```

```
"${workspace_loc}/${ProjName}/src/r_ble_ota/downloader}"
```

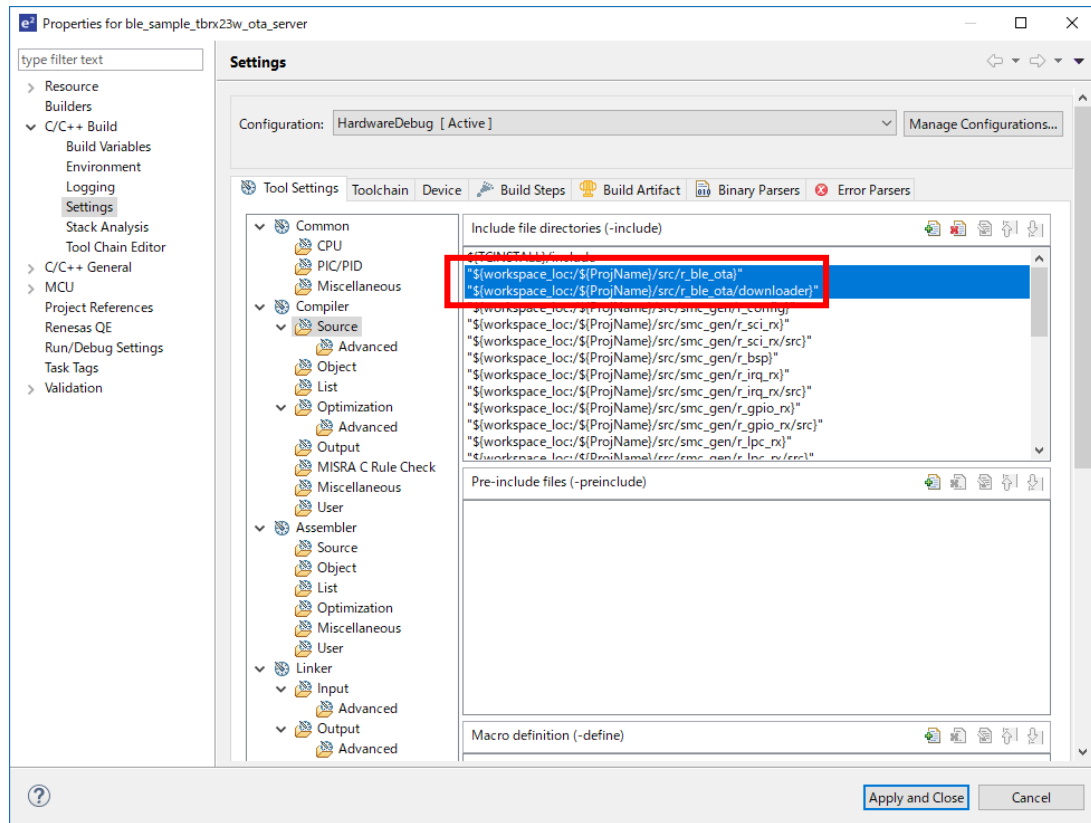


Figure 4.2 Add include path

4.2 Section division settings

Divide the program into sections that support the OTA firmware update feature. Set the following two items in e² studio.

1. Create section (-start)

Change the settings of "C/C++ Settings"->"Tool Settings"->"Linker"->"Section"->"Section (-start)".

Divide the section area as shown in Figure 4.3. By importing the "ProjectSetting\section.esi" file included with this package, you can generate a section layout similar to the OTA Server sample program.

If you want to extend the application section, change the start address of `FWDL_HEADER` and change the following macro in `r_ble_ota_heade.h` to an appropriate value.

```
BLE_OTA_APPLICATION_SECTION_SIZE
```

```
BLE_OTA_TEMPORALY_SECTION_SIZE
```

```
BLE_OTA_DOWNLOADER_SECTION_SIZE
```

The RAM used by the downloader and relocater programs is section-overlaid on the RAM area used by the user application.

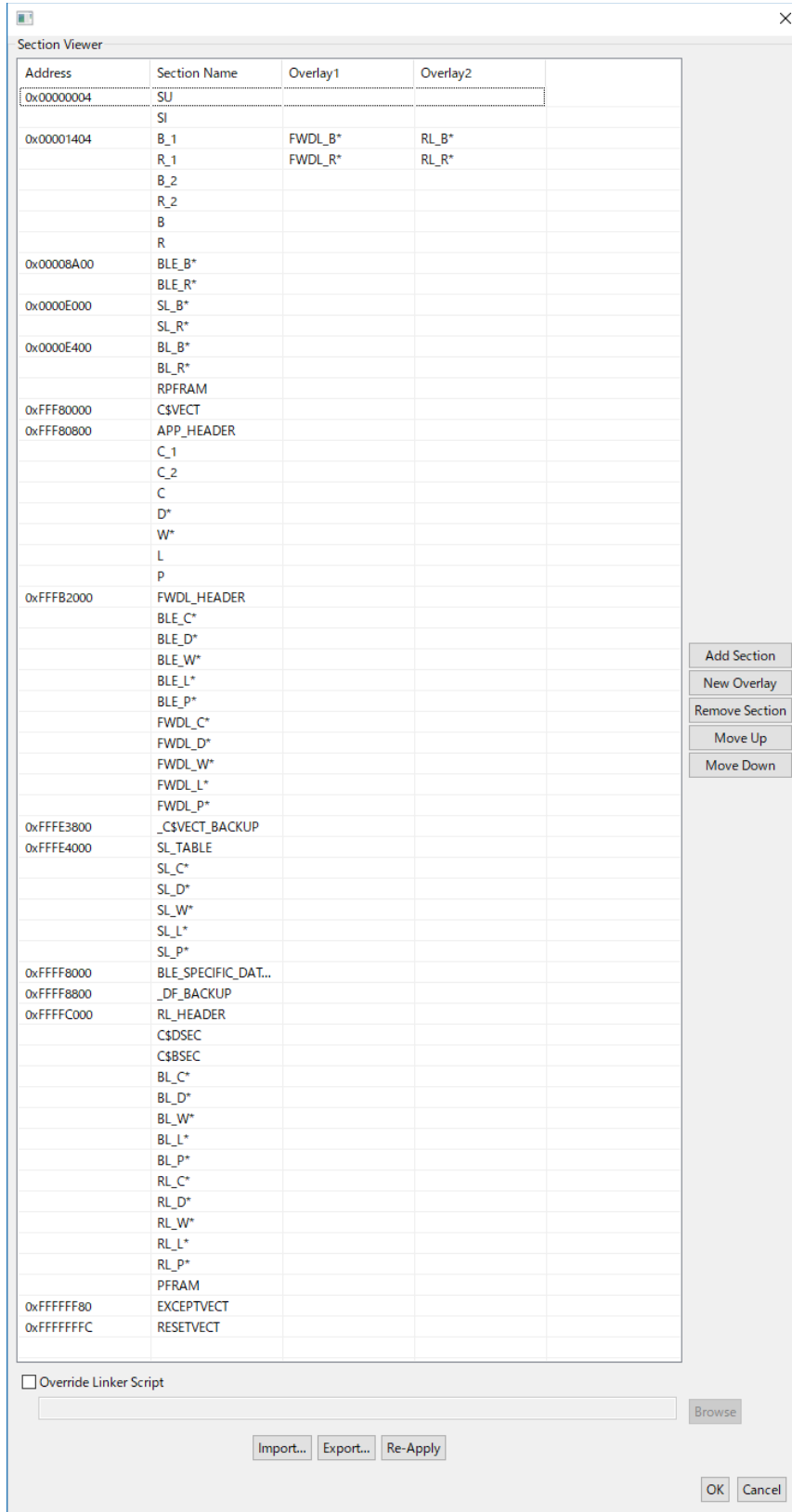


Figure 4.3 Section settings

```

24
25     /* OTA Configuration */
26
27     /* Application Main Function */
28     #define MAIN_FUNCTION                app_main
29     /* Project Information */
30
31     /* BLE OTA_PROJECT_NAME length < 18*/
32     #define BLE_OTA_PROJECT_NAME        "ota_sample"
33
34     /* BLE OTA_PROJECT_NAME length = 18*/
35     #define BLE_OTA_RESET_CODE          "inge9ubled9hy4tljn"
36
37     /* Downloader Mode Device Name */
38     #define BLE_OTA_DOWNLOADER_DEVICE_NAME    "FWU-DEV"
39
40     /* Version Information */
41     #define BLE_OTA_APPLICATION_MAJOR_VERSION    (0x01)
42     #define BLE_OTA_APPLICATION_MINOR_VERSION    (0x0A)
43     #define BLE_OTA_DOWNLOADER_MAJOR_VERSION    (0x01)
44     #define BLE_OTA_DOWNLOADER_MINOR_VERSION    (0x0A)
45     #define BLE_OTA_RELOCATER_MAJOR_VERSION    (0x01)
46     #define BLE_OTA_RELOCATER_MINOR_VERSION    (0x00)
47
48     /* Section Information */
49     #define BLE_OTA_APPLICATION_SECTION_SIZE    (0x00031800)
50     #define BLE_OTA_TEMPORALY_SECTION_SIZE    (0x00031800)
51     #define BLE_OTA_DOWNLOADER_SECTION_SIZE    (0x00031800)
52     #define BLE_OTA_RELOCATER_SECTION_SIZE    (0x00004000)
53     #define BLE_OTA_STDLIB_SECTION_SIZE        (0x00014000)
54
55     /* Downloader Firmware Block Size */
56     #define BLE_OTA_DL_FW_BUFFER_SIZE          (0x4000)
57     #define BLE_OTA_DL_CVECT_BUFFER_SIZE      (1024)
58
59     /* Memory Configuration */
60     /* Data Flash*/
61     #define BLE_OTA_DATAFLASH_BLOCK_LENGTH    (1024)
62     #define BLE_OTA_USE_DATAFLASH_START_ADDRESS    (0x00100400)
63

```

Figure 4.4 r_ble_ota_config.h settings

2. Setting the section that maps from ROM to RAM

Change the settings of "C/C++ Settings"->"Tool Settings"->"Linker"->"Section"->"Section (-start)". Add a ROM-to-RAM mapping for each program is shown in Figure 4.5.

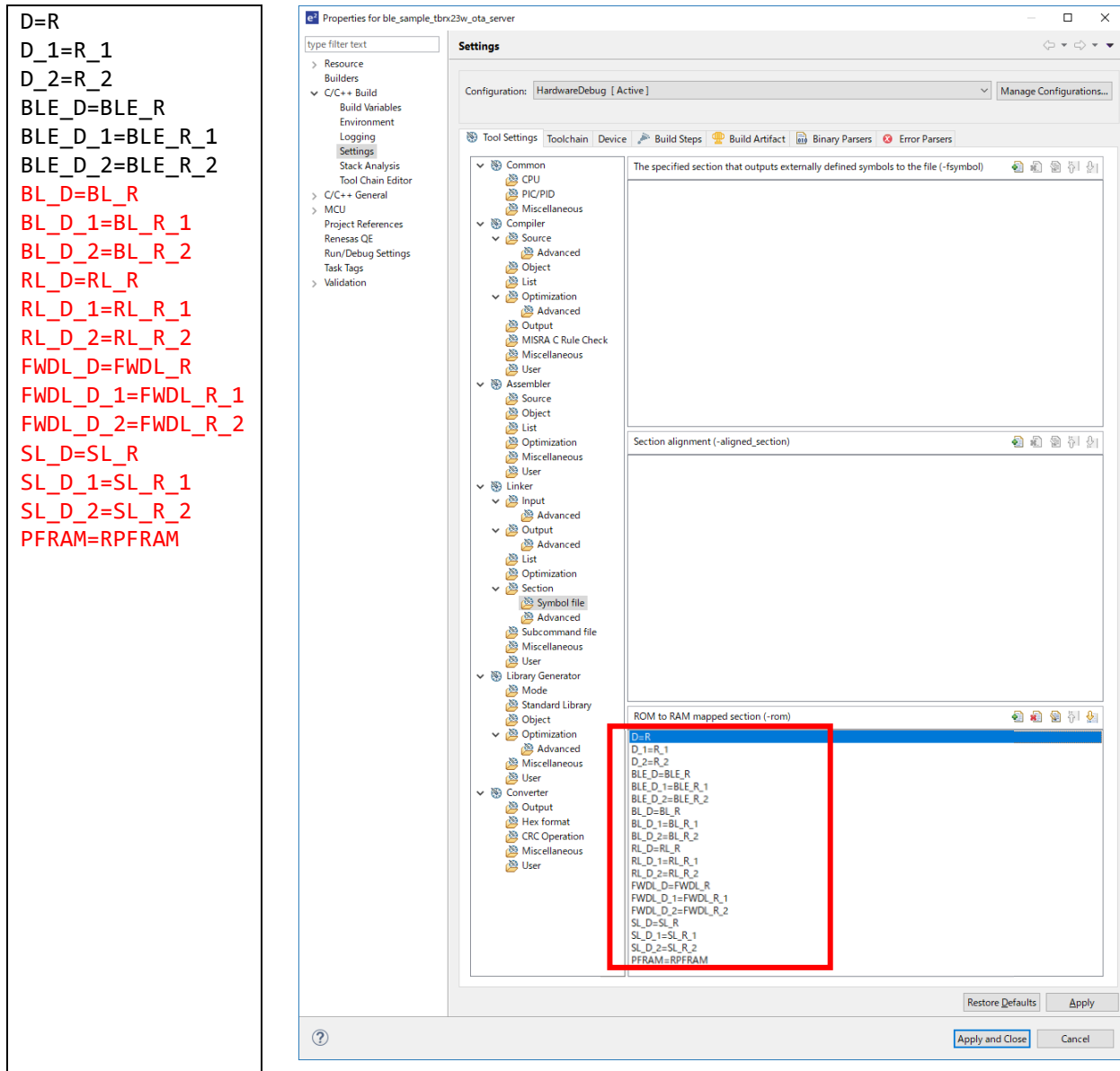


Figure 4.5 ROM to RAM mapped section settings

4.3 Standard library settings

The standard library will not be updated by OTA. The library to be used is written to the firmware in advance. Select the library to use in the Library Generator settings of “Tool Settings”.

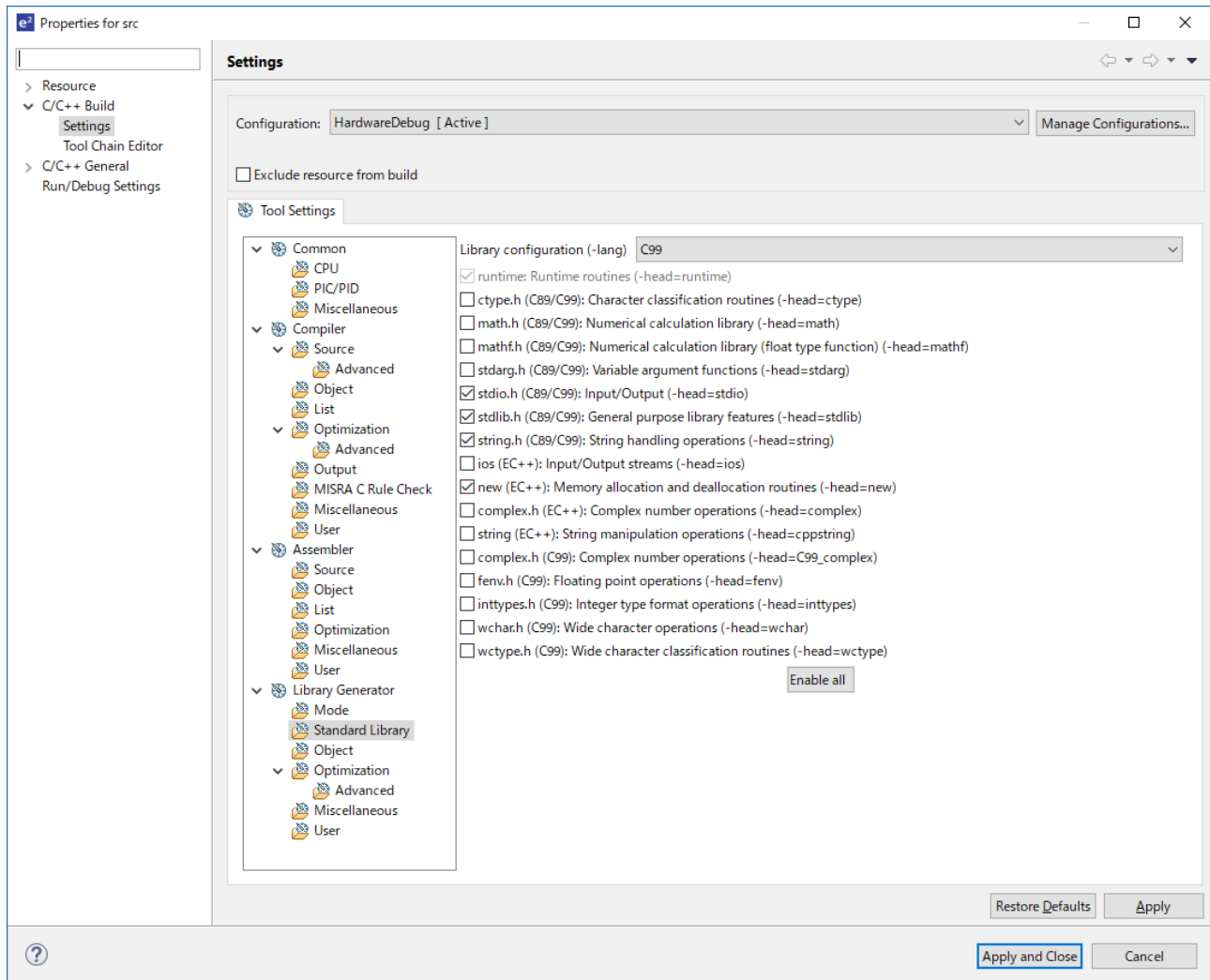


Figure 4.6 Standard library selection

The OTA Server program provides a dedicated section for deploying the standard library. In the Library Generator object settings, set the section as follows:

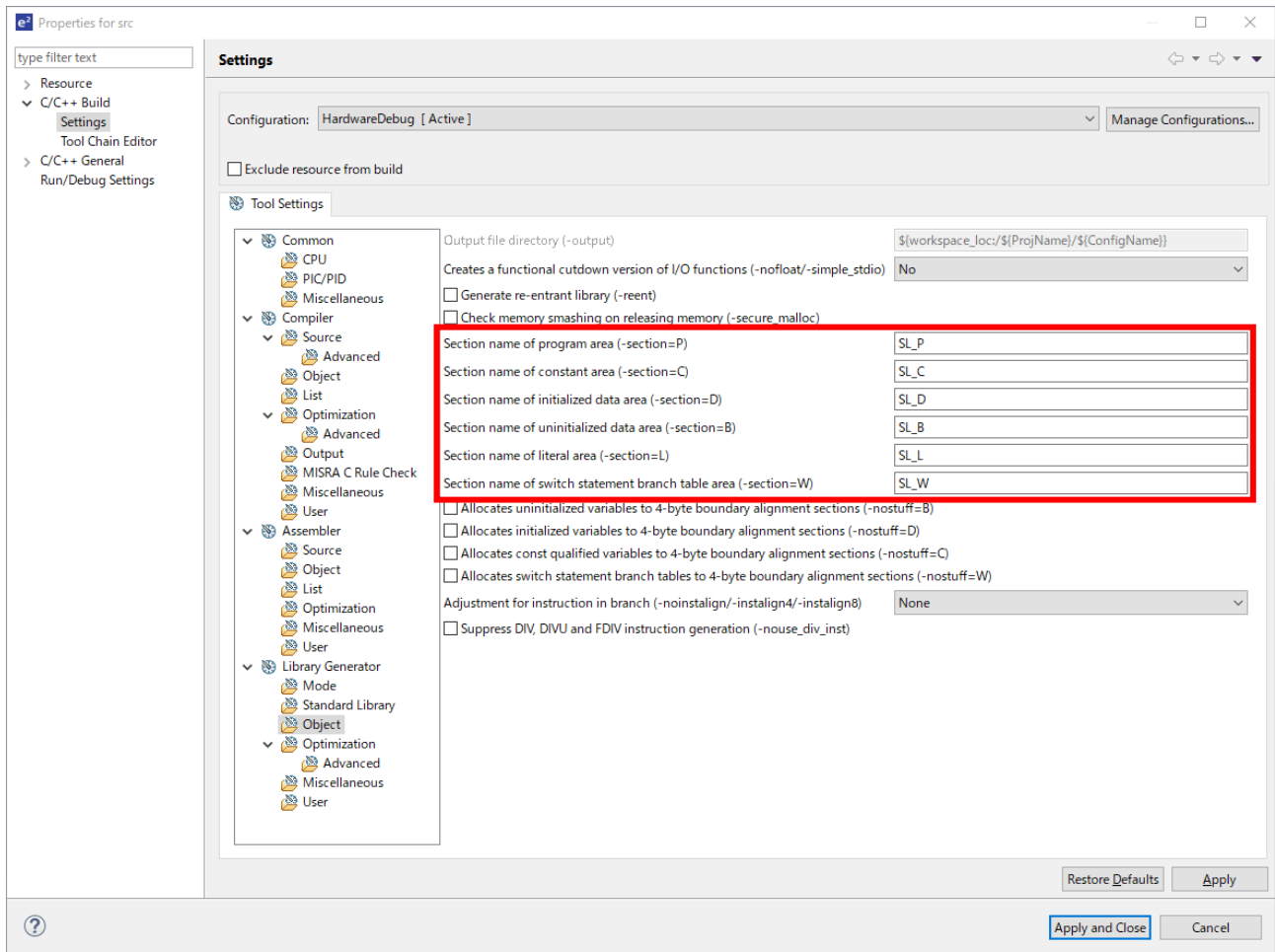


Figure 4.7 Standard library section name settings

The OTA Server program stores all the functions defined in the standard library in an array so that changes in the program do not change the placement of the standard library in code flash.

In "r_ble_ota/utility/r_ble_ota_stdlib_table.c", enable the macro definition corresponding to the standard library to be used.

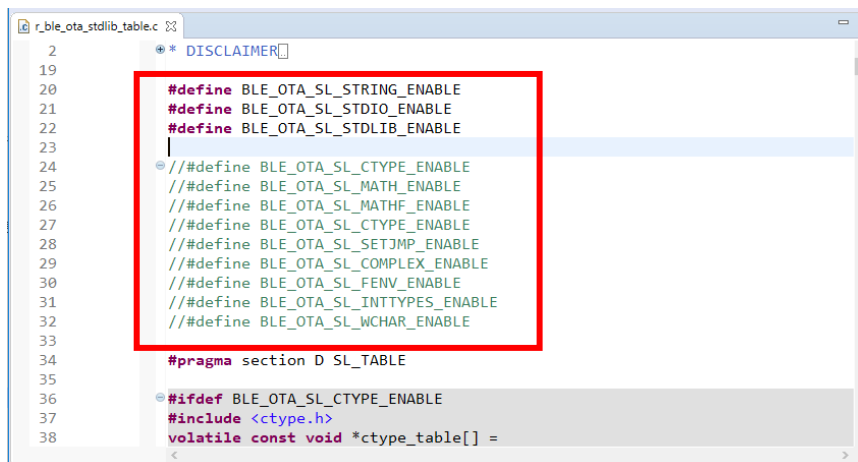


Figure 4.8 Enable the standard library to use

4.4 Update firmware output settings

The update firmware used in this sample program consists of a json file that describes firmware information such as section information and a binary firmware file (.bin).

"r_ble_ota_bin_genelator.bat" will generate a hex file (binary format) (firmware.bin) from the load module file (.abs) file generated by the CCRX compiler.

In addition, "r_ble_ota_bin_genelator.bat" extracts the firmware information required for OTA updating from the generated execution binary file (.bin) and linkage list file (.map), and creates a "firmware_information.json" file. Since python script is executed from the batch file, please pass the execution path of python.

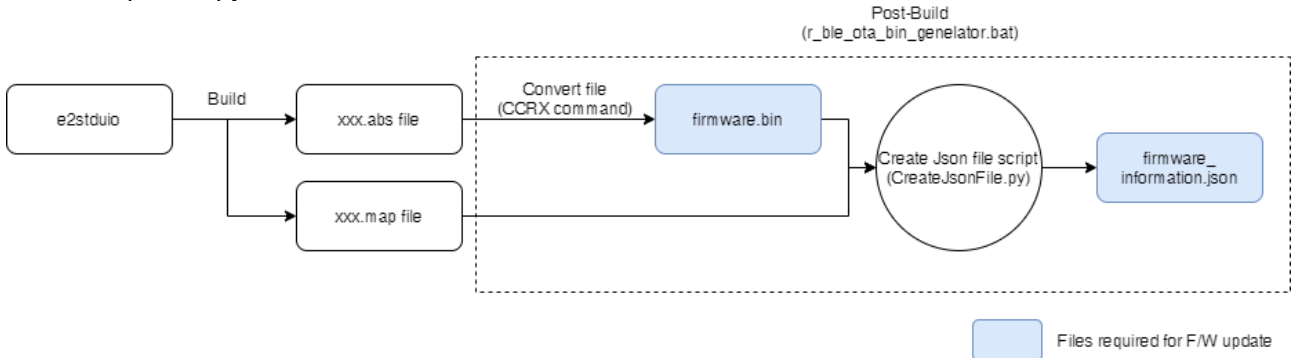


Figure 4.9 Output of update firmware file

Change the Post Build Steps setting to run the batch file after the build.

```
..\src\r_ble_ota\utility\r_ble_ota_bin_genelator.bat
```

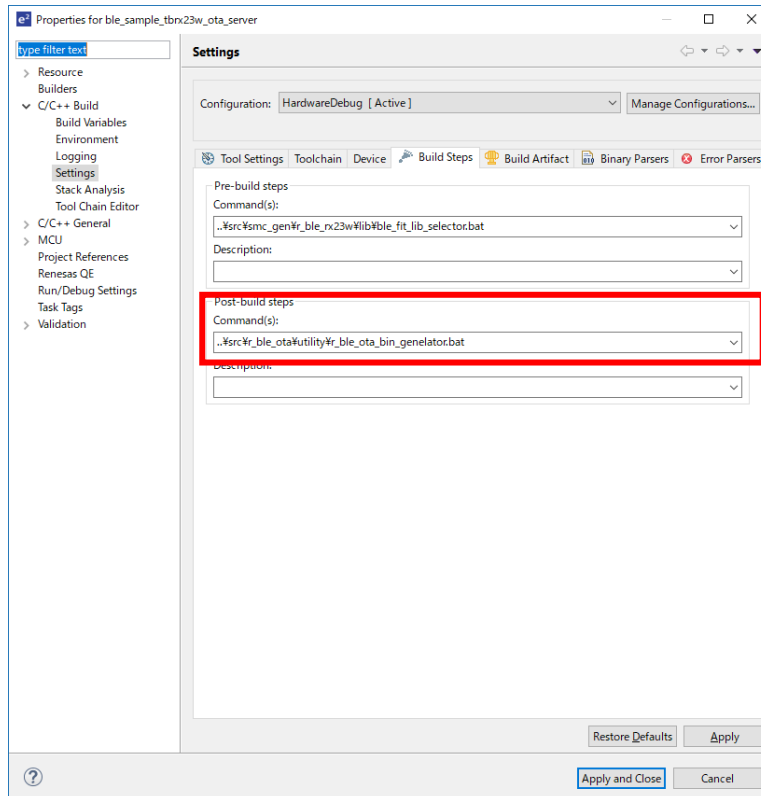


Figure 4.10 Registering a batch file for post build

Also, change the linker settings to write the map information of the object to the linkage list file (.map).

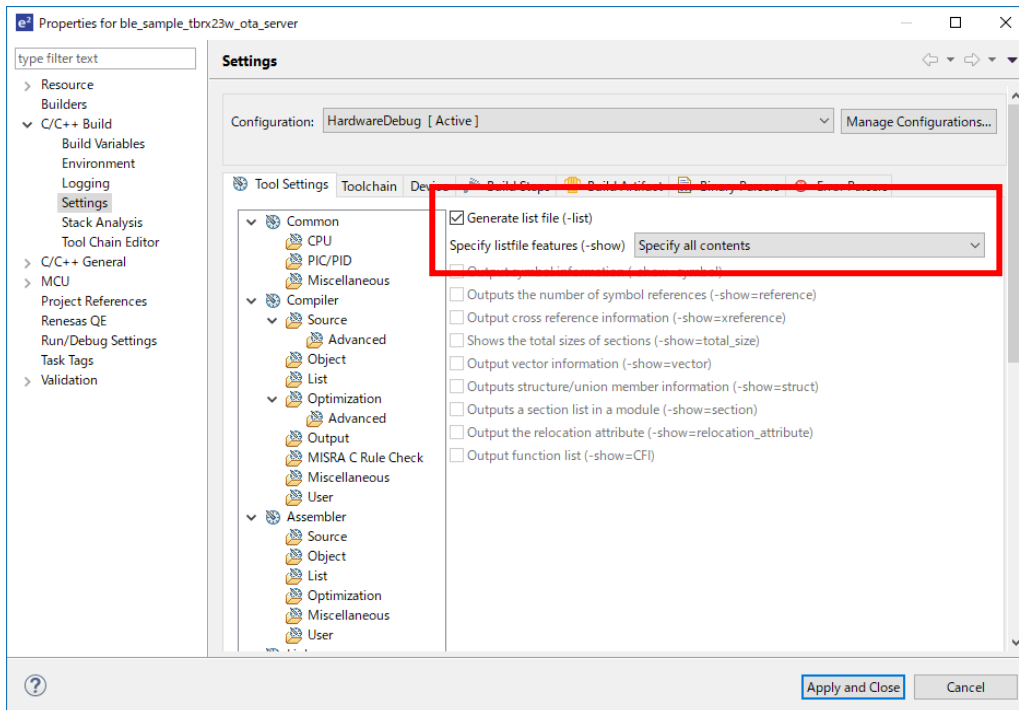


Figure 4.11 Linkage list file (.map) generation settings

For the two created files, the "FWU Client" folder and the [Projectname] _Version [Application Version] folder will be created in the folder with the build configuration name (e.g. HardwareDebug) after the build is completed.

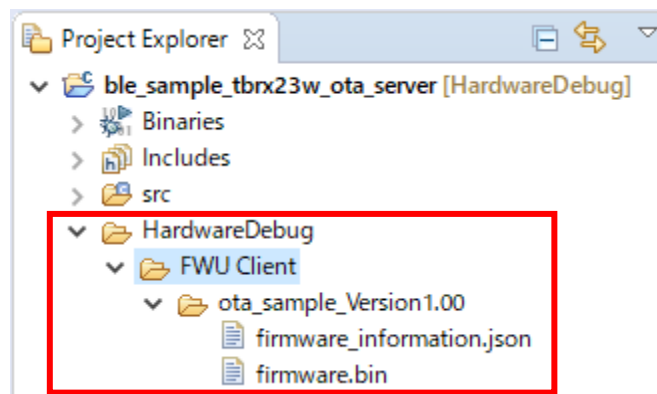


Figure 4.12 Output of OTA update compatible firmware file

4.4.1 Firmware information JSON file properties

The OTA Firmware Update Sample Program uses a JSON-formatted firmware update information file that contains the firmware project name, version information, and section information, in addition to the firmware binary file generated for the RX23W.

The information contained in the JSON file is shown in Table 4.1.

OTA Server program version and section information can be found in "r_ble_ota_firmware_information.c". Make sure that the description in the JSON file matches the contents of the source code.

Table 4.1 Contents of the firmware update information JSON file

Property	Default Value	Description
ProjectName	ota_sample	The project name. The OTA Client identifies the OTA Server based on this property.
ResetCode	inge9ubled9hy4tljn	The value to write to the Virtual Reset Button of the OTA Reset Service.
MajorApplicationVersion	0x01	The major version of the application.
MinorApplicationVersion	0x00	The minor version of the application.
MajorDownloaderVersion	0x01	The major version of the downloader.
MinorDownloaderVersion	0x00,0x01	The minor version of the downloader.
MajorRelocatorVersion	0x01	The major version of the relocater.
MinorRelocatorVersion	0x00	The minor version of the relocater.
MajorBLEProtocolStackVersion	0x02	The major version of the BLE protocol stack.
MinorBLEProtocolStackVersion	0x00	The minor version of the BLE protocol stack.
SectionInformation	---	Describes the section allocation of memory for each program. The top property is the starting address of each section. The size property is the section size actually used by the program The max property is set to the size specified by the section size macro.
VariableVectorTableSection	top-0xFFFF8000 size-0x00000400 max-0x00000400	Variable vector table section layout information.
ApplicationSection	top-0xFFFF80800 size-0x00020000 max-0x00031800	Application section layout information.
DownloaderAndProtocolStackSection	top-0xFFFFB2000 size-0x00026800 max-0x00031800	Downloader section layout information.
VariableVectorTableBackupSection	top-0xFFFE3800 size-0x00000800 max-0x00000800	Placement information for the temporary write section of the variable vector table.
StandardLibrarySection	top-0xFFFE4000 size- 0x00007800 max- 0x00014000	Standard library section layout information.
SwapTemporarySection	top-0xFFFFF8000 size-0x00004000 max-0x00004000	Reserve section layout information.
SwapSection	top-0xFFFFFC000 size-0x00004000 max-0x00004000	Relocater section layout information.

4.5 Flash FIT module settings

While rewriting the code flash Since access to the code flash is prohibited, writing to the code flash is executed by placing the instruction code in RAM.

The OTA Server program uses the Flash FIT module to rewrite the code flash. Change the following settings of the Flash FIT module to enable RAM execution of the Flash FIT module.

- Encode code flash programming: Includes code to program ROM area.

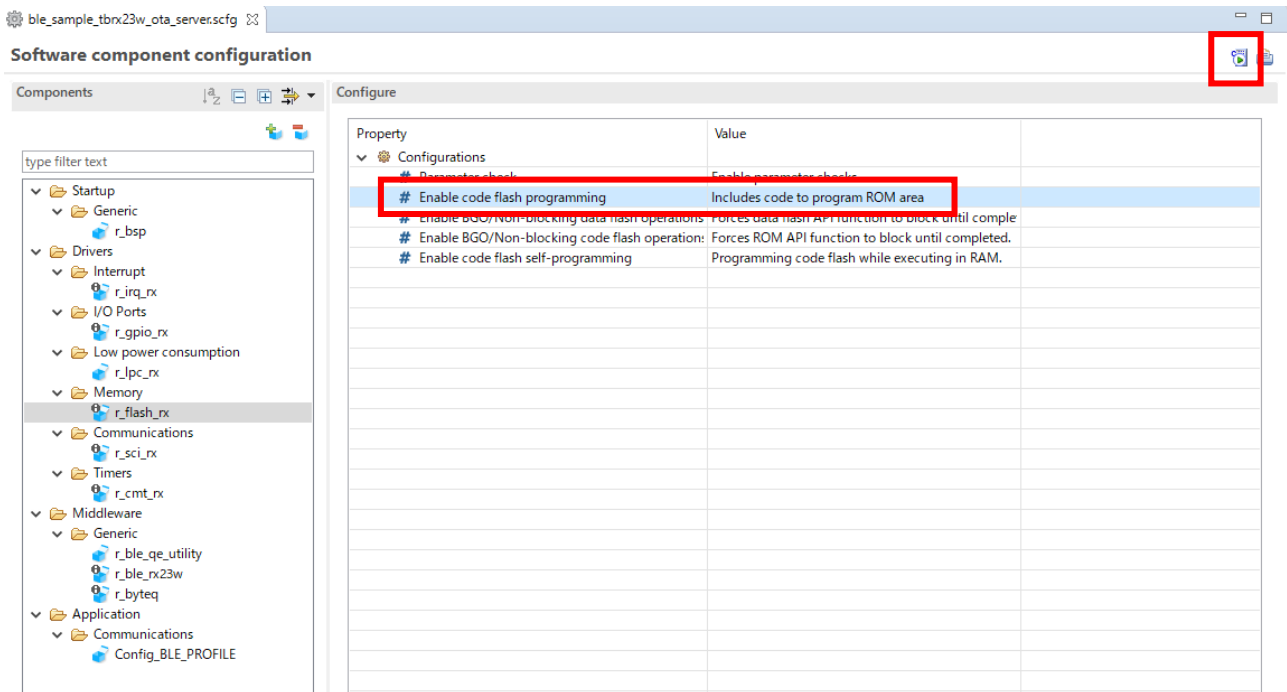


Figure 4.13 Flash FIT module settings

Press the code generation button on the upper right to reflect the settings.

4.6 Addition of Renesas OTA Reset Service

Add the Renesas OTA Reset service to the GATT database to check the firmware version information and switch to the downloader while the user application is running.

Register the service from QE for BLE to the GATT database. Click the Import button and specify the included JSON file (Renesas_OTA_Reset_Service_Server.json) to add it.

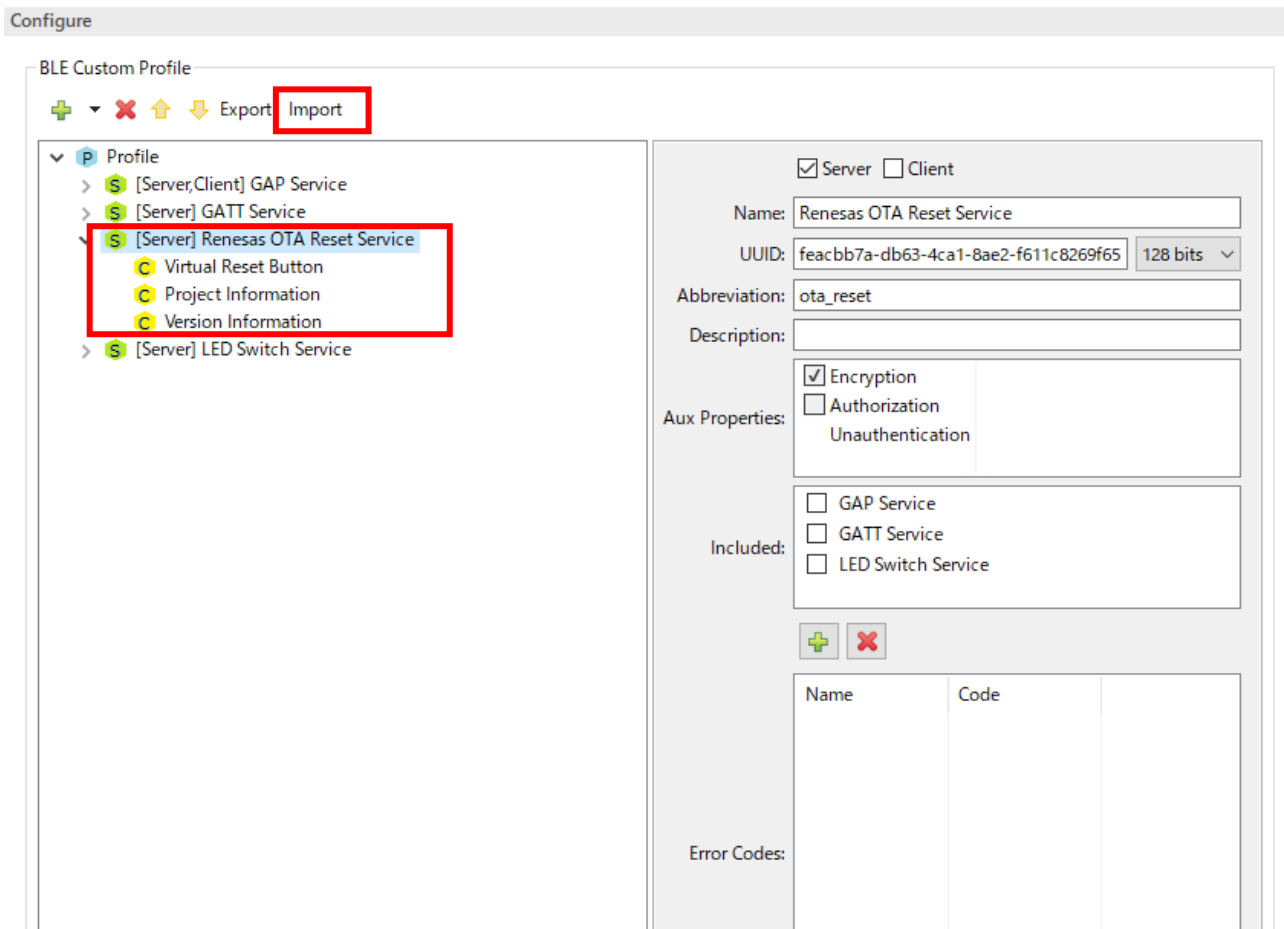


Figure 4.14 Import OTA Reset Service

After adding the service, execute code generation.

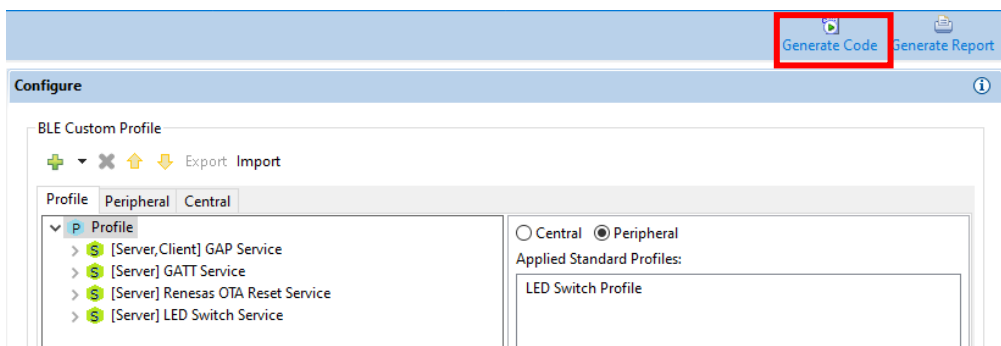


Figure 4.15 Execute Generation Code

The files in the smc_gen\Config_BLE_PROFILE folder will be overwritten. The file before overwriting is stored in the trash folder. If you need the original source code, copy and restore files other than gatt_db.c / gatt_db.h.

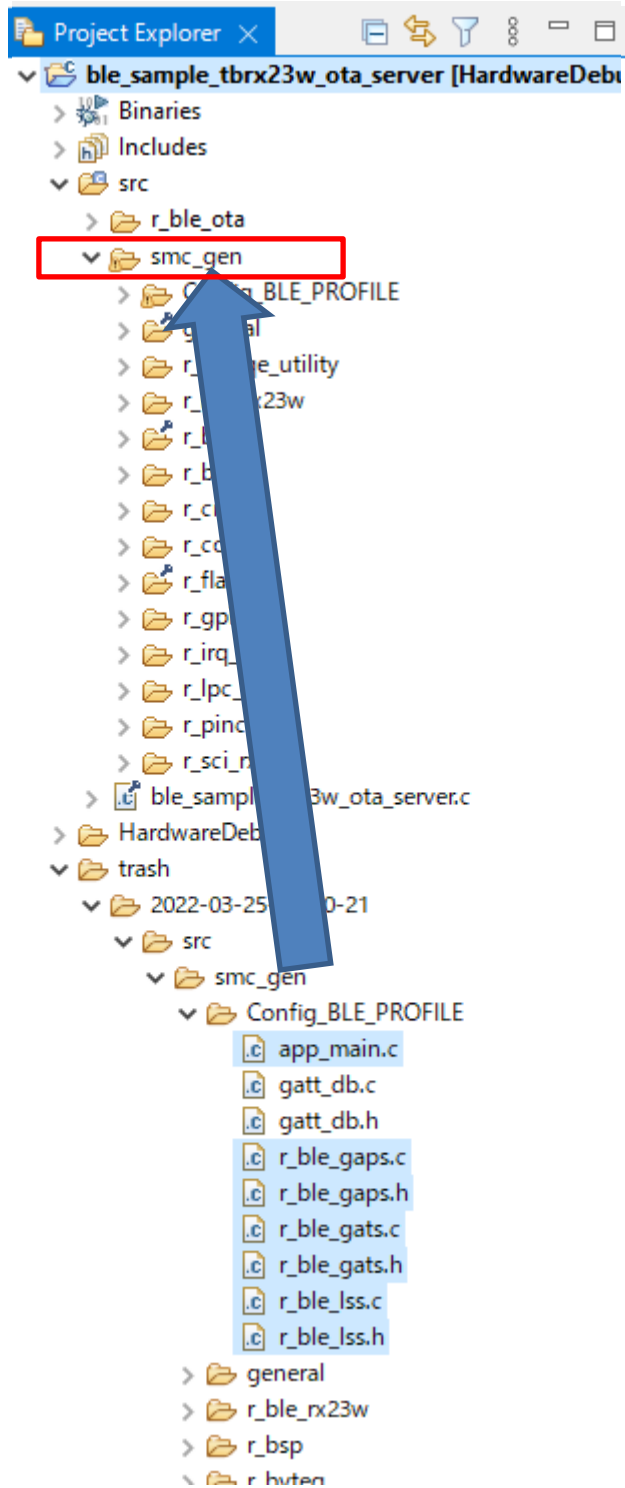


Figure 4.16 Copy user programs in trash folder

4.7 Assignment of source code to each section

Assign each program to the section set in “4.2 Section division settings”. In e² studio, you can specify the section where the program is placed for each source code folder. Right-click the folder from the project explorer, select Properties to open the menu screen, and specify the section of each program from the items of "C/C++ Build"->"Settings"->"Compiler"->"Object".

Place the following folders and files in the bootloader section (BL_*).

r_ble_ota/bootloader

r_ble_ota/utility

smc_gen/general

smc_gen/r_flash

smc_gen/r_bsp

<ProjectName>.c (Place the main function called from the BSP module in BL_* sections)

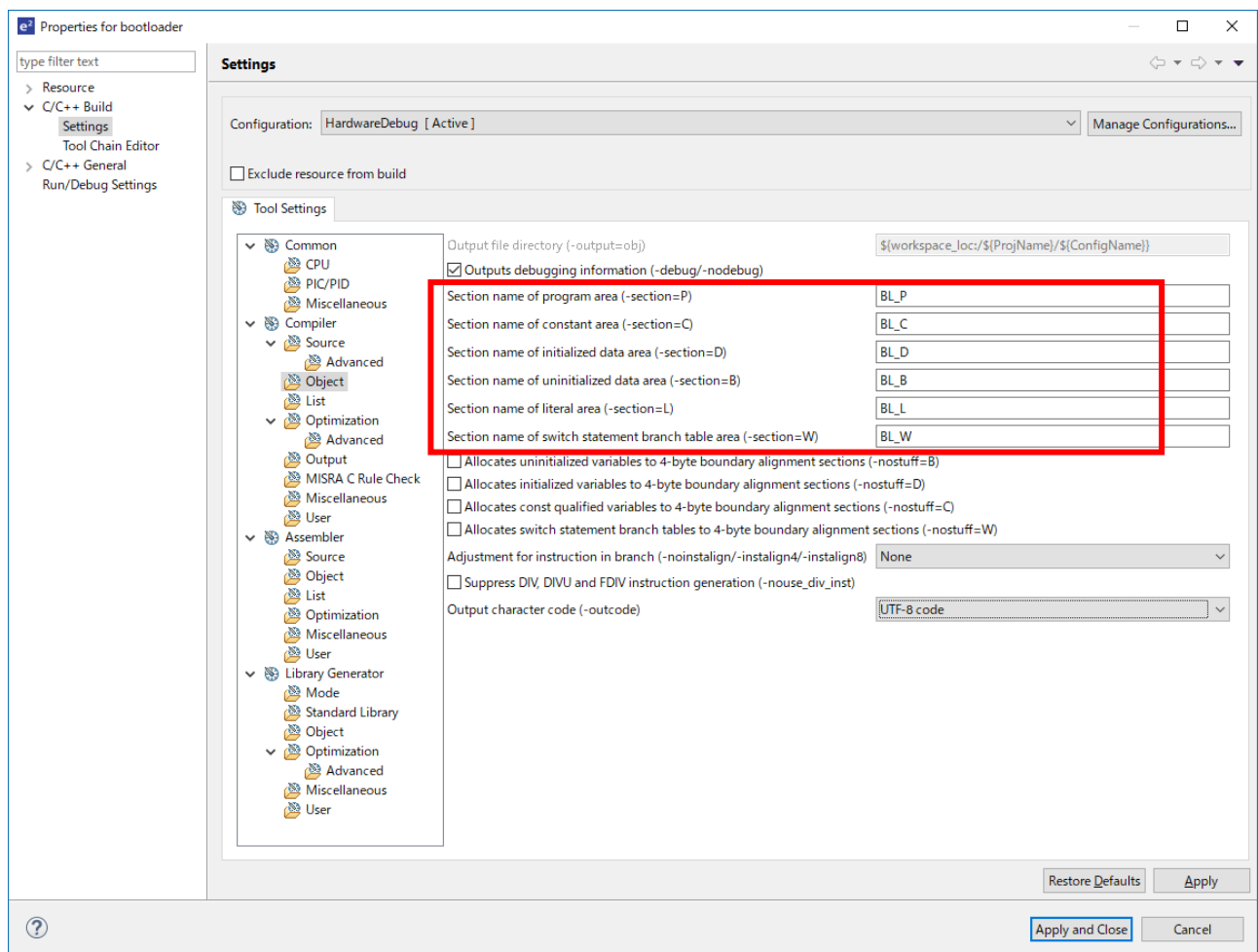


Figure 4.17 Specifying the bootloader section

Place the following folders in the Downloader section (FWDL_ *).

r_ble_ota/downloader

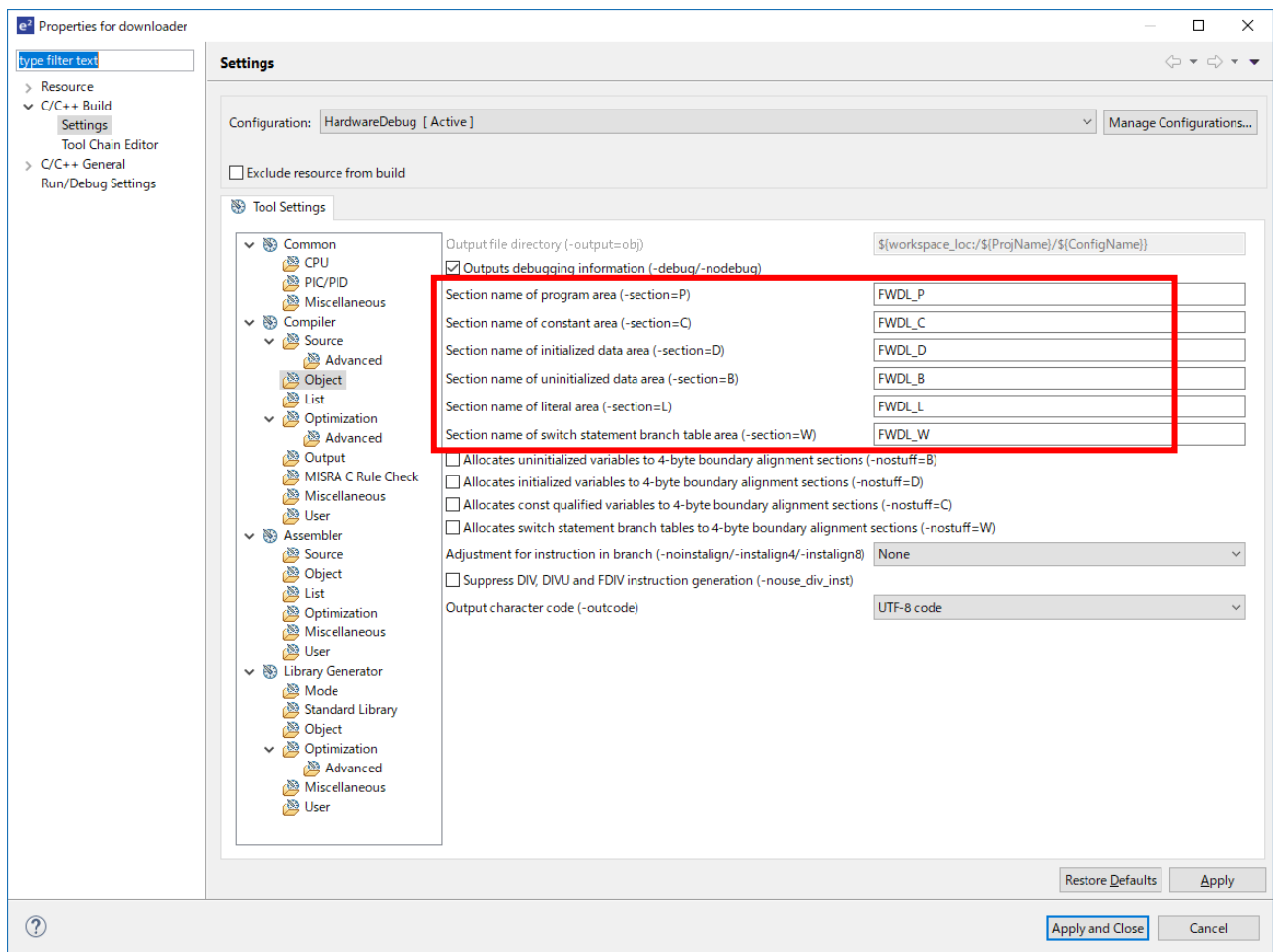


Figure 4.18 Specifying the downloader section

Place the following folders in the Relocator section (FWDL_ *).

r_ble_ota/relocator

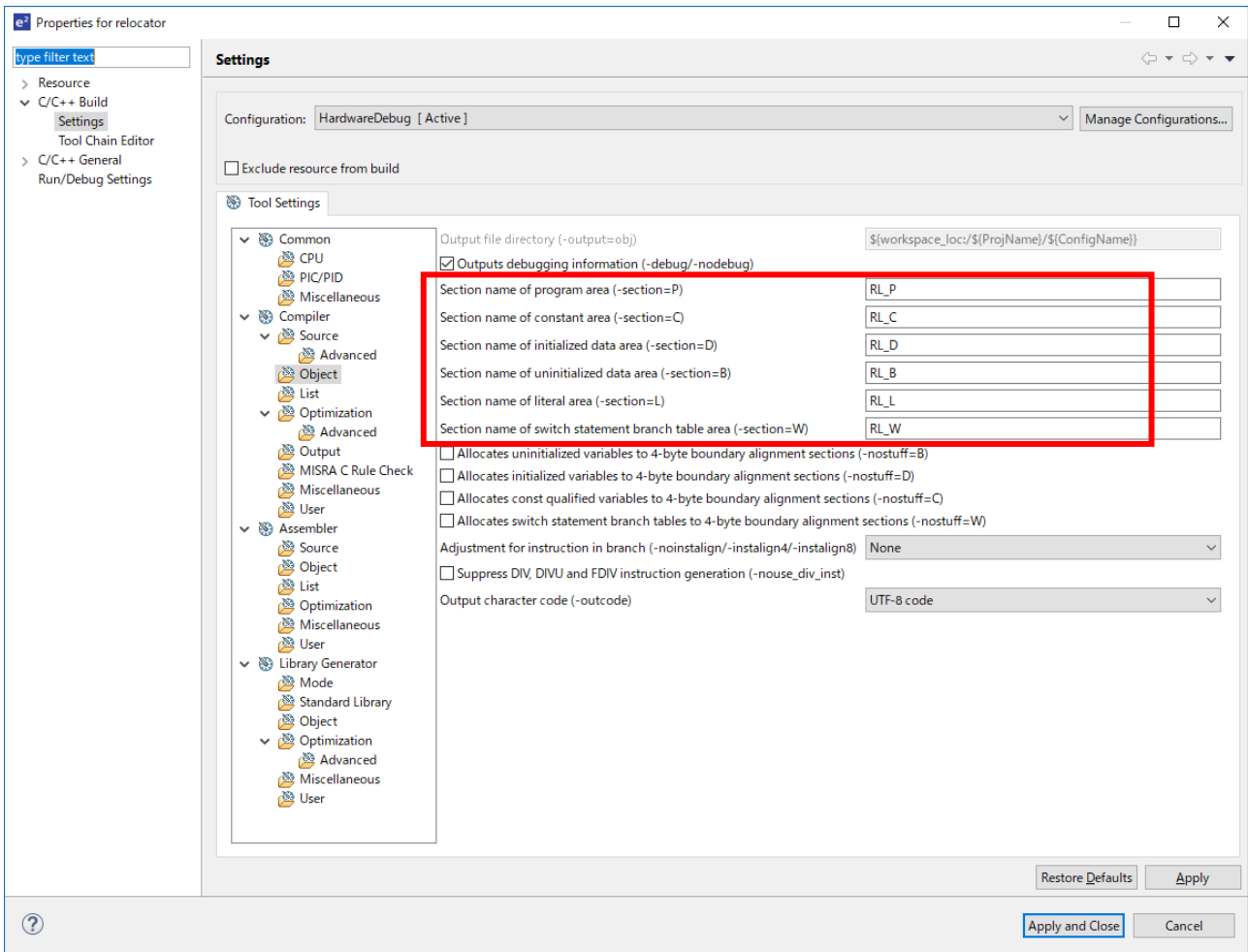


Figure 4.19 Specifying the relocator section

4.8 Editing dbstc.c

In the RX family, including the RX23W, RAM is initialized by the `__INITSCT` function when the microcontroller starts. The `__INITSCT` function refers to the `_DTBL` table and `BTBL` to initialize RAM. Modify this table to initialize the bootloader and standard library RAM at boot time.

These variables are defined in the following files of the BSP FIT module.

src\smc_gen\r_bsp\mcu\all\dbstc.c

```

70
71
72 /* Section start */
73 #pragma section C C$DSEC
74
75 extern st_dtbl_t const _DTBL[] = {
76 #ifndef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
77     { __sectop("SL_D_8"), __sectend("SL_D_8"), __sectop("SL_R_8") },
78     { __sectop("BL_D_8"), __sectend("BL_D_8"), __sectop("BL_R_8") },
79 #endif
80     { __sectop("BL_D"), __sectend("BL_D"), __sectop("BL_R") },
81     { __sectop("BL_D_2"), __sectend("BL_D_2"), __sectop("BL_R_2") },
82     { __sectop("BL_D_1"), __sectend("BL_D_1"), __sectop("BL_R_1") },
83     { __sectop("SL_D"), __sectend("SL_D"), __sectop("SL_R") },
84     { __sectop("SL_D_2"), __sectend("SL_D_2"), __sectop("SL_R_2") },
85     { __sectop("SL_D_1"), __sectend("SL_D_1"), __sectop("SL_R_1") },
86 #if (BSP_CFG_RTOS_USED == 4) && (BSP_CFG_RENESAS_RTOS_USED == RENESAS_RI600PX)
87     { __sectop("DRI_ROM"), __sectend("DRI_ROM"), __sectop("BRI_RAM") }
88 #endif /* Renesas RI600PX */
89 };
90
91 /* Section start */
92 #pragma section C C$BSEC
93
94 extern st_btbl_t const _BTBL[] = {
95 #ifndef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
96     { __sectop("BL_B_8"), __sectend("BL_B_8") },
97     { __sectop("SL_B_8"), __sectend("SL_B_8") },
98 #endif
99     { __sectop("BL_B"), __sectend("BL_B") },
100    { __sectop("BL_B_2"), __sectend("BL_B_2") },
101    { __sectop("BL_B_1"), __sectend("BL_B_1") },
102    { __sectop("SL_B"), __sectend("SL_B") },
103    { __sectop("SL_B_2"), __sectend("SL_B_2") },
104    { __sectop("SL_B_1"), __sectend("SL_B_1") },
105 #if (BSP_CFG_RTOS_USED == 4) && (BSP_CFG_RENESAS_RTOS_USED == RENESAS_RI600V4)
106    { __sectop("BRI_RAM"), __sectend("BRI_RAM") }
107 #endif /* Renesas RI600V4 */
108 };
109
110 /* Section start */
111 #pragma section
112
113 #if (BSP_CFG_RTOS_USED == 4) && (BSP_CFG_RENESAS_RTOS_USED == RENESAS_RI600PX)
114 #pragma section C C$
115 #endif /* Renesas RI600PX */
116
117 #if 0
118 /* CTBL prevents excessive output of L1100 messages when linking.
119  Even if CTBL is deleted, the operation of the program does not change. */
120 uint8_t * const _CTBL[] = {
121     __sectop("C_1"), __sectop("C_2"), __sectop("C"),
122     #ifndef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
123     __sectop("C_8"),
124     #endif
125     __sectop("W_1"), __sectop("W_2"), __sectop("W")
126 };
127 #endif
128 /* Preprocessor directive */
129 #pragma packoption

```

Figure 4.20 Edit dbstc.c

Then disable the CBTL table and remove the diff to the relocator section. Deleting this table has no effect on the executable file.

```

112 #if (BSP_CFG_RTOS_USED == 4) && (BSP_CFG_RENESAS_RTOS_USED == RENESAS_RI600
113 #pragma section C C$
114 #endif /* Renesas RI600PX */
115
116 #if 0
117 /* CTBL prevents excessive output of L1100 messages when linking.
118  Even if CTBL is deleted, the operation of the program does not change. */
119 uint8_t * const _CTBL[] = {
120     __sectop("C_1"), __sectop("C_2"), __sectop("C"),
121     #ifndef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
122     __sectop("C_8"),
123     #endif
124     __sectop("W_1"), __sectop("W_2"), __sectop("W")
125 };
126 #endif
127 /* Preprocessor directive */
128 #pragma packoption

```

Figure 4.21 CBTL disabled

4.9 r_ble_ota configuration settings

OTA Server behavior, section information and version information are set in r_ble_ota_config.h. The items to be set are shown in Table 4.2.

Table 4.2 r_ble_ota configuration options

Macro name	Default Value	Description
BLE_OTA_DEBUG_PUTS		Specify the name of the function to output the log of update event information in downloader. If it is an empty macro, no log output will be performed.
MAIN_FUNCTION	app_main	Specifies the main function of the user application. The argument and return type will be void type.
BLE_OTA_PROJECT_NAME	"ota_sample"	Set the firmware project information. The maximum 18 bytes strings. Used to validate whether the firmware is to be updated. Note: This config is set when creating a project. Do not change when updating the firmware.
BLE_OTA_RESET_CODE	"inge9ubled9hy4tljn"	The value to write to the Virtual Reset Button of the OTA Reset Service. Specify with a fixed 18 strings. Note: This config is set when creating a project. Do not change when updating the firmware.
BLE_OTA_DOWNLOADER_DEVICE_NAME	"FWU-DEV"	Sets the device name in downloader. The values set in this macro are used as the GAP Service Device Name Characteristic and the COMPLETE NAME in advertisement scan response data. Note: recommend similar values to applications.
BLE_OTA_APPLICATION_MAJOR_VERSION	0x01	Sets the major version of the application section.
BLE_OTA_APPLICATION_MINOR_VERSION	0x0A	Sets the minor version of the application section.
BLE_OTA_DOWNLOADER_MAJOR_VERSION	0x01	Sets the major version of the downloader section. Change when updating the BLE protocol stack or downloader in the downloader section.
BLE_OTA_DOWNLOADER_MINOR_VERSION	0x0A	Sets the minor version of the downloader section. Change when updating the BLE protocol stack or downloader in the downloader section.
BLE_OTA_RELOCATER_MAJOR_VERSION	0x01	Sets the major version of the relocater section. Change when updating the relocater program or bootloader program in the relocater section.
BLE_OTA_RELOCATER_MINOR_VERSION	0x00	Sets the minor version of the relocater section. Change when updating the relocater program or bootloader program in the relocater section.
BLE_OTA_APPLICATION_SECTION_SIZE	0x31800 (198KB)	Sets the size of the application section.
BLE_OTA_TEMPORALY_SECTION_SIZE	0x31800 (198KB)	Sets the ROM size that temporarily section the firmware received by the OTA Server. Set the same value as BLE_OTA_APPLICATION_SECTION_SIZE.
BLE_OTA_DOWNLOADER_SECTION_SIZE	0x31800 (198KB)	Sets the size of the downloader section.
BLE_OTA_RELOCATER_SECTION_SIZE	0x4000 (16KB)	Sets the size of the relocater section.
BLE_OTA_STDLIB_SECTION_SIZE	0x14000 (80KB)	Sets the size of the standard library section.
BLE_OTA_DL_FW_BUFFER_SIZE	0x4000	Sets the maximum firmware block size received in downloader.

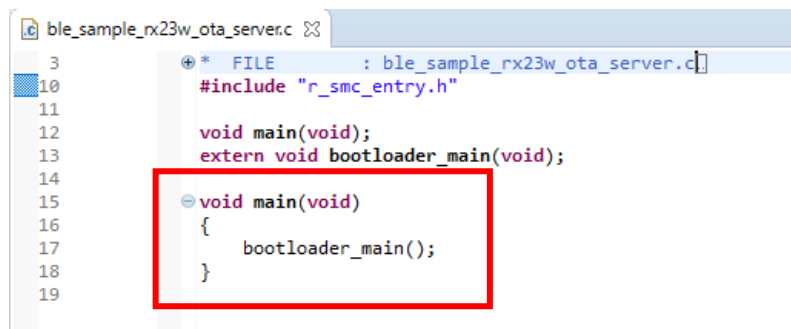
Macro name	Default Value	Description
BLE_OTA_DL_CVECT_BUFFER_SIZE	0x0400	Sets the buffer size when receiving a variable vector table. No changes are required.
BLE_OTA_DATAFLASH_BLOCK_LENGTH	0x0400	Sets the data flash block size. No changes are required.
BLE_OTA_USE_DATAFLASH_START_ADDRESS	0x00100400	Sets the start address of the data flash block used to switch programs.
BLE_OTA_CODEFLASH_BLOCK_LENGTH	0x0800	Set the minimum rewrite size of the code flash of the microcomputer to be used.
BLE_OTA_CODEFLASH_START_ADDRESS	0xFFFF80000	Set the start address of the MCU to be used.
BLE_OTA_CODEFLASH_SIZE	0x00080000 (512KB)	Set the size of the code flash of the MCU to be used.
BLE_OTA_RAM_START_ADDRESS	0x00000000	Set the RAM start address of the MCU to be used.
BLE_OTA_RAM_SIZE	0x00010000	Set the RAM size of the MCU to be used.
BLE_OTA_TRAPERROR_LED_ENABLE		Set whether to enable LED blinking when an irreversible error occurs. If this macro is disabled, the LED control for error will be disabled.

4.10 User application modification

In this chapter, the main function of the user application is set to the boot loader. Set the section of RAM to initialize before launching the user application.

4.10.1 Main function settings

The OTA Server program implements a boot loader for program switching. Call the `bootloader_main` function before executing the user's main function.

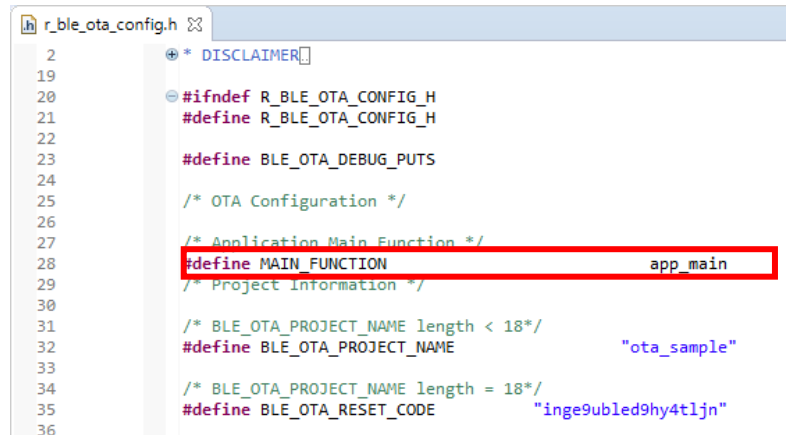


```
ble_sample_rx23w_ota_server.c
3
10 #include "r_smc_entry.h"
11
12 void main(void);
13 extern void bootloader_main(void);
14
15 void main(void)
16 {
17     bootloader_main();
18 }
19
```

Figure 4.22 setting main function

4.10.2 User application main function registration and initialization section settings

The main function of the user application is executed via the application_loader function. Set the user's main function in the MAIN_FUNCTION macro in r_ble_config.h.



```

2      + * DISCLAIMER
19
20      #ifndef R_BLE_OTA_CONFIG_H
21      #define R_BLE_OTA_CONFIG_H
22
23      #define BLE_OTA_DEBUG_PUTS
24
25      /* OTA Configuration */
26
27      /* Application Main Function */
28      #define MAIN_FUNCTION                app_main
29      /* Project Information */
30
31      /* BLE_OTA_PROJECT_NAME length < 18*/
32      #define BLE_OTA_PROJECT_NAME        "ota_sample"
33
34      /* BLE_OTA_PROJECT_NAME length = 18*/
35      #define BLE_OTA_RESET_CODE         "inge9ubled9hy4tljn"
36

```

Figure 4.23 Setting user main function to bootloader.

The user application is called from the application_loader function in the r_ble_ota_app_loader.c file.

The application_loader function performs RAM initialization. If your application wants to use a section other than the default, change the initialized_ram function.



```

1      + * DISCLAIMER
19
20      #include <string.h>
21      #include "r_ble_ota_header.h"
22      extern void MAIN_FUNCTION(void);
23
24      static void initialized_ram(void)
25      {
26          memcpy(__sectop("R_1"), __sectop("D_1"), __seclsize("R_1"));
27          memcpy(__sectop("R_2"), __sectop("D_2"), __seclsize("R_2"));
28          memcpy(__sectop("R"), __sectop("D"), __seclsize("R"));
29
30          memset(__sectop("B"), 0x00, __seclsize("B"));
31          memset(__sectop("B_1"), 0x00, __seclsize("B_1"));
32          memset(__sectop("B_2"), 0x00, __seclsize("B_2"));
33
34      #ifdef BSP_MCU_DOUBLE_PRECISION_FLOATING_POINT
35          memcpy(__sectop("R_8"), __sectop("R_8"), __seclsize("R_8"));
36          memset(__sectop("B_8"), __sectop("B_8"), __seclsize("B_8"));
37      #endif
38      }
39
40      void application_loader(void)
41      {
42          initialized_ram();
43          MAIN_FUNCTION();
44      }
45

```

Figure 4.24 r_ble_ota_app_loader.c

4.11 Addition of Renesas OTA Reset Service

Implement the following process in the user application to perform OTA firmware update by FWU_Client.

- 1 Switching process to Renesas OTA Reset Service and downloader.
- 2 Implementation of bonding.
- 3 Indication processing of Service Changed Characteristic of GATT Service (only when using iOS)

4.11.1 Switching process to Renesas OTA Reset Service and downloader

The Renesas OTA Reset Service is used by FWU_Client to read the firmware project name and version information and switch the program to the downloader.

Add the following functions to the user application.

- Setting a value for each characteristic of Renesas OTA Reset Service
- Match confirmation when writing Reset Code and switching process to downloader.

All of this is implemented in the `r_ble_ota_resets.c` file that comes with the package. You can add processing by replacing this file in the project with the one in the bundled package.

Replace the files `r_ble_ota_resets.c` and `r_ble_ota_resets.h` generated by QE for BLE with the files with the same name included in this package.

File to be replaced:

```
{ProjectName}\smc_gen\Config_BLE_PROFILE\r_ble_ota_resets.c
```

```
{ProjectName}\smc_gen\Config_BLE_PROFILE\r_ble_ota_resets.h
```

The file included in the package:

```
r01an5910xx0111-rx23w-otafwup\ProjectSetting\service\service api\r_ble_ota_resets.c
```

```
r01an5910xx0111-rx23w-otafwup\ProjectSetting\service\service api\r_ble_ota_resets.h
```

To enable this function, implement the empty `ota_resets_cb` function and execute the `R_BLE_OTA_RESETS_Init` function before executing `R_BLE_Execute` in `app_main.c`. Please refer to the `app_main.c` file of this sample (`ble_sample_tbrx23w_ota_server`) for the detailed implementation method.

```
#include "r_ble_ota_resets.h"

void ota_resets_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
}

/* Initialize Renesas OTA Reset Service server API */
R_BLE_OTA_RESETS_Init(ota_resets_cb);
```

Figure 4.25 Example of adding Renesas OTA Reset Service to `app_main.c`

`r_ble_ota_resets.c` sets the `g_ble_ota_project_info` variable and the version information of each HEADER section in the GATT database when the Read Request comes to Project Information Characteristic and Version Information Characteristic, respectively. This process is implemented in the following functions respectively.

```
static void ble_ota_resets_Project_info_read_req_cb(const void *p_attr, uint16_t conn_hdl)
```

```
static void ble_ota_resets_version_info_read_req_cb(const void *p_attr, uint16_t conn_hdl)
```

These functions are registered in each characteristic definition variable.

```

/* Project Information characteristic definition */
static const st_ble_servs_char_info_t gs_project_info_char = {
    .start_hdl    = BLE_OTA_RESETS_PROJECT_INFO_DECL_HDL,
    .end_hdl      = BLE_OTA_RESETS_PROJECT_INFO_VAL_HDL,
    .char_idx     = BLE_OTA_RESETS_PROJECT_INFO_IDX,
    .app_size     = sizeof(st_ble_seq_data_t),
    .db_size      = BLE_OTA_RESETS_PROJECT_INFO_LEN,
    .decode       = (ble_servs_attr_decode_t)decode_st_ble_seq_data_t,
    .encode       = (ble_servs_attr_encode_t)encode_st_ble_seq_data_t,
    .read_req_cb  = ble_ota_resets_Project_info_read_req_cb,
};

/* Version Information characteristic definition */
static const st_ble_servs_char_info_t gs_version_info_char = {
    .start_hdl    = BLE_OTA_RESETS_VERSION_INFO_DECL_HDL,
    .end_hdl      = BLE_OTA_RESETS_VERSION_INFO_VAL_HDL,
    .char_idx     = BLE_OTA_RESETS_VERSION_INFO_IDX,
    .app_size     = sizeof(st_ble_ota_resets_version_info_t),
    .db_size      = BLE_OTA_RESETS_VERSION_INFO_LEN,
    .decode       = (ble_servs_attr_decode_t)decode_st_ble_ota_resets_version_info_t,
    .encode       = (ble_servs_attr_encode_t)encode_st_ble_ota_resets_version_info_t,
    .read_req_cb  = ble_ota_resets_version_info_read_req_cb,
};

```

Figure 4.26 Registration of callback function of Read Request of r_ble_ota_resets.c

In r_ble_ota_resets.c, check the Reset Code match in the Write Request event to the Virtual Reset Button Characteristic. When the Reset Code matches, disconnects from the opposing device in the Write Comp event. After that, user application is reset to downloader.

By calling the R_BLE_OTA_SwitchModeReset function implemented in "r_ble_ota/utility/r_ble_ota_reset.c", you can switch to downloader with any trigger in the user application. The usage of the R_BLE_OTA_SwitchModeReset function is shown below. Read the current settings from DataFlash and specify the downloader in activate_mode. The R_BLE_OTA_SwitchModeReset function writes the startup program flag information to the non-volatile area and resets the MCU.

```

st_ble_ota_dataflash_info_t data_flash_info;

R_BLE_OTA_FLASH_DRIVER_Open();
R_BLE_OTA_FLASH_DRIVER_ReadDataFlash(&data_flash_info);
R_BLE_OTA_FLASH_DRIVER_Close();

data_flash_info.activate_mode = BLE_OTA_ACTIVATE_MODE_DOWNLOADER;
data_flash_info.status_info.code = BLE_OTA_STATUS_SUCCESS;
data_flash_info.status_info.source = BLE_OTA_ACTIVATE_MODE_APPLICATION;

R_BLE_OTA_SwitchModeReset(&data_flash_info);

```

Figure 4.27 Example of using R_BLE_OTA_SwitchModeReset

Access to the code flash is prohibited while the R_BLE_OTA_SwitchModeReset function is running. When an interrupt occurs, access to the code flash occurs via the vector table. If any interrupt is expected to occur, set interrupt disable in r_ble_ota_resets_gatts_cb in r_ble_resets.c.

```

st_ble_ota_dataflash_info_t data_flash_info;

R_BLE_OTA_FLASH_DRIVER_Open();
R_BLE_OTA_FLASH_DRIVER_ReadDataFlash(&data_flash_info);
R_BLE_OTA_FLASH_DRIVER_Close();

data_flash_info.activate_mode = BLE_OTA_ACTIVATE_MODE_DOWNLOADER;
data_flash_info.status_info.code = BLE_OTA_STATUS_SUCCESS;
data_flash_info.status_info.source = BLE_OTA_ACTIVATE_MODE_APPLICATION;

R_BSP_InterruptsDisable();

R_BLE_OTA_SwitchModeReset(&data_flash_info);

```

Figure 4.28 Example for disabled interrupts

r_ble_ota_resets.c registers one GATT Server callback function in the host stack of R_BLE_API. Set the number of registered GATTS callback functions to the number used by the user application + 1 or more. In app_main.c generated by QE for BLE, it is set to the maximum value BLE_GATTS_MAX_CB (15).

```

/* GATT server initialization parameters */
static st_ble_abs_gatts_init_param_t gs_abs_gatts_init_param =
{
    .p_cb_param = gs_abs_gatts_cb_param,
    .cb_num      = BLE_GATTS_MAX_CB,
};

```

Figure 4.29 Setting the number of registered callback functions

4.11.2 Implementation of bonding

The pairing information is read from the Data Flash specified by the security library of app_lib of the OTA Server BLE Protocol Stack running in the downloader and used, but the non-volatile area of the pairing information is not rewritten. Therefore, it is recommended to perform bonding in the user application.

Please use the security library of app_lib of BLE Protocol Stack to keep the pairing information in Data Flash while the user application is running.

For details on how to use the security library, refer to Chapter 5.2 “Security Data Management” of the “Bluetooth Low Energy Protocol Stack Basic Package User's Manual (R01UW0205)” and “Bluetooth Low Energy application developer's guide (R01AN5504) Section 9 Security”.

4.11.3 Indication processing of Service Changed Characteristic of GATT Service

The iOS device caches some of the information about Bluetooth devices to BD address. The information to be cached also includes the structure of GATT Database.

The OTA Server has another GATT Database in the downloader program execution, and you need to delete cache information to perform update operation.

To delete the cache of GATT Database, send the Indication of Service Changed Characteristic of GATT Service. When using the iOS version FWU-Client, implement it so that Service Changed Characteristic can be indicated after connection.


```
uint16_t cccd = 0;
R_BLE_GATS_GetServChangedCliCnfg(p_data->conn_hdl, &cccd);
if((cccd & BLE_GATTS_CLI_CNFG_INDICATION) == BLE_GATTS_CLI_CNFG_INDICATION)
{
    st_ble_gats_serv_changed_t serv_changed;
    serv_changed.start_hdl = 0x0000;
    serv_changed.end_hdl = 0xffff;
    R_BLE_GATS_IndicateServChanged(p_data->conn_hdl, &serv_changed);
}
```

Figure 4.30 Service Changed Characteristic Indication

In this sample, the Indication is sent immediately after the Indication is permitted by the Client Characteristic Configuration Descriptor (CCCD) of the Service Changed Characteristic.

5. Update firmware settings

If the source code is changed and the firmware is changed, the version information of the corresponding section will be updated. It is recommended that the application version be incremented each time the firmware is changed.

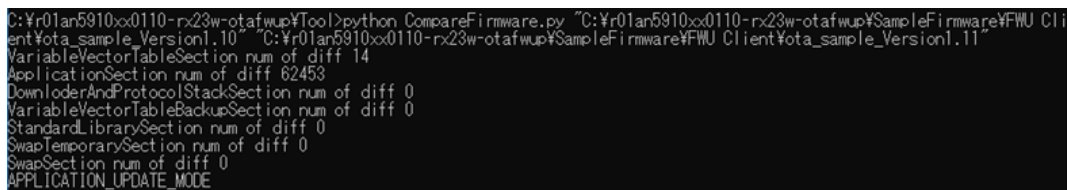
This OTA sample program uses the XXX_VERSION macro in r_ble_ota_config.h to determine whether the firmware can be updated. If the version information is inconsistent, it will not work properly after updating.

A script (CompareFirmware.py) that determines whether the update firmware can be updated is provided in the Zip package of this application note. You can check the firmware difference using this script. Specify the path of the directory of the folder that contains the firmware information in the run-time argument of the script.

Execution example:

```
$> python CompareFirmware.py "ota_sample_Version1.10" "ota_sample_Version1.11"
```

After execution, the number of differences in each section and the comparison result will be displayed.



```
C:\Yr01an5910xx0110-rx23w-otafwup\Tool>python CompareFirmware.py "C:\Yr01an5910xx0110-rx23w-otafwup\SampleFirmware\FWU Client\ota_sample_Version1.10" "C:\Yr01an5910xx0110-rx23w-otafwup\SampleFirmware\FWU Client\ota_sample_Version1.11"
VariableVectorTableSection num of diff 14
ApplicationSection num of diff 62453
DownloaderAndProtocolStackSection num of diff 0
VariableVectorTableBackupSection num of diff 0
StandardLibrarySection num of diff 0
SwapTemporarySection num of diff 0
SwapSection num of diff 0
APPLICATION_UPDATE_MODE
```

Figure 5.1 CompareFirmware.py execution screen

The meaning of the result is as follows.

Table 5.1 CompareFile.py comparison results

Result	Description
APPLICATION_UPDATE_MODE	There is a firmware diff only in the application section. This firmware can be updated in application update mode.
DOWNLOADER_UPDATE_MODE	There is a firmware diff between the application section and the downloader section. This firmware can be updated in BLE protocol stack update mode.
FIRMWARE_ERROR	There is a diff in the standard library section or the relocater section. This firmware cannot be updated by OTA.
FIRMWARE_INFORMATION_ERROR	Section information in json file does not match.
NO_REQUIRED_UPDATE	The diff for each section shown in the json file does not exist. This firmware does not need to be updated.

6. Android application FWU_Client project

This section simply describes how to build and debug the smartphone application FWU_Client on Android Studio. For more detail to Android Studio, see the Android Developer site (<https://developer.android.com/>).

6.1 How to build

FWU_Client was developed in Android Studio Version 4.1. Select Open from the Android Studio menu and select the FWU_Client folder in the package to open the project. Select "Build"- "Make Project" on the menu bar to build.

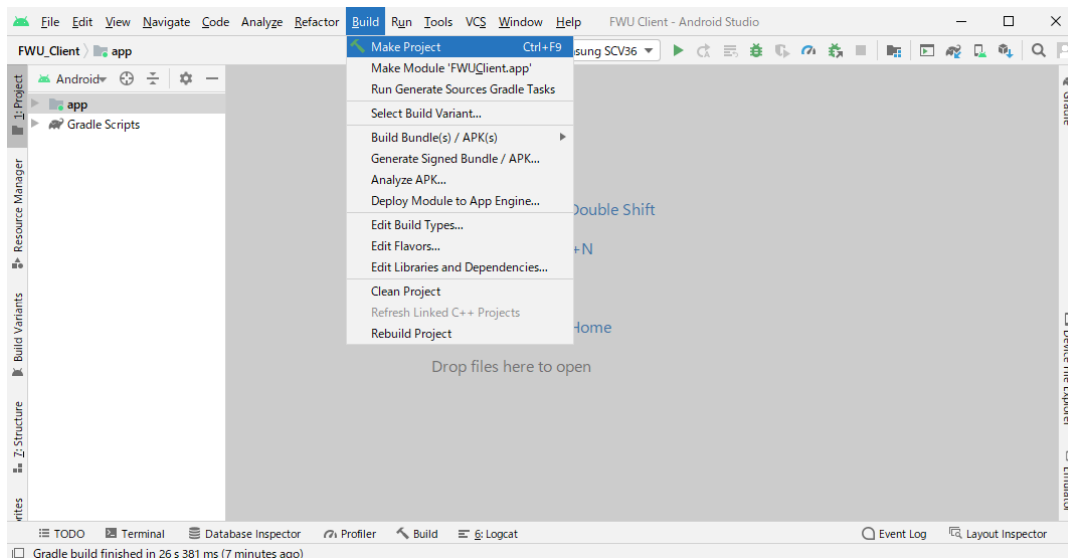


Figure 6.1 Build menu

6.2 How to debug

Enable USB debugging in the developer options for Android devices. When you connect an Android device with USB debugging enabled to your PC, the debug device list is displayed. Select "Run"- "Debug app" from the menu bar to start debugging.

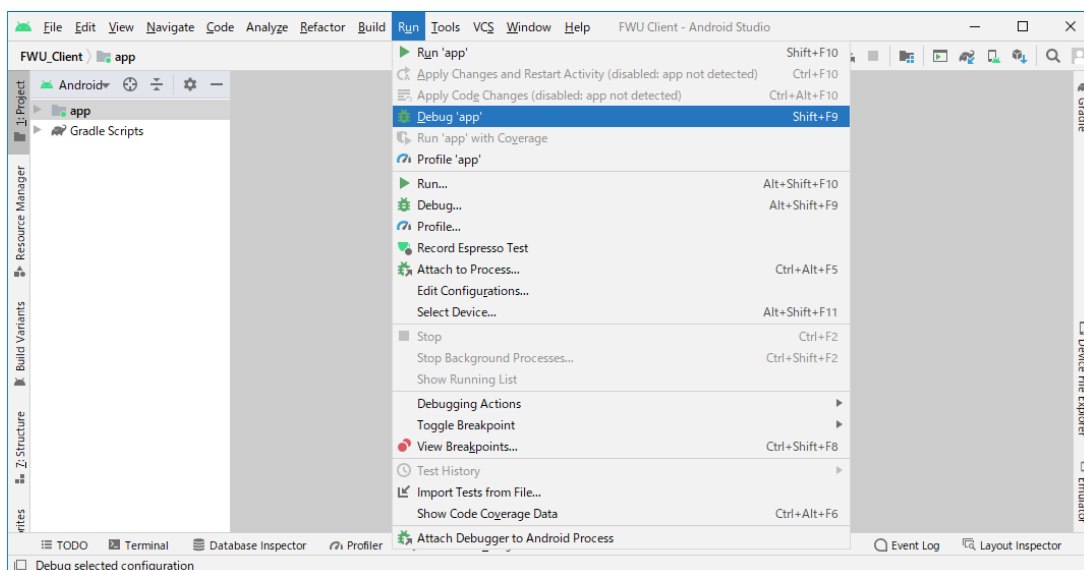


Figure 6.2 Debug menu

6.3 Correspondence to API specification change of Android 10.0

The Android version FWU_Client uses APIs that have been changed in Android 10.0 to read files. File access is now prohibited to the path of the directory acquired by the following API. Use the ACTION_OPEN_DOCUMENT_TREE intent etc. to retrieve access to the firmware folder from the user.

(utils/FileUtils.java line 39) Environment.getExternalStorageDirectory()

In addition, the necessary privileges for scan execution of Bluetooth LE are enhanced. About permission check, please fix the following.

(bluetooth_device_scan/BluetoothDeviceScanActivity.java line 480,482)

ACCESS_COARSE_LOCATION → ACCESS_FINE_LOCATION

7. FWU_Client for iOS

This section describes the development environment of smartphone app FWU_Client for iOS.

7.1 Operation Confirmation Environment

Indicates the operation check environment.

Table 7.1 IOS version FWU_Client Operation Confirmation Environment

category	Environment
IDE	Xcode 12.5.1
Operation device	iPhone 8 iOS 14.7.1
Develop device	MacBook Air (M1,2020) macOS Big Sur version 11.2.3

7.2 Debug

See section 2.4.1.

7.3 iOS GATT Database Cache Function

The iOS device caches some of the information about Bluetooth devices to BD address. The information to be cached also includes the structure of GATT Database. iOS version FWU_Client waits for Service Characteristic Indication to ensure that the OTA Server GATT Database is detected.

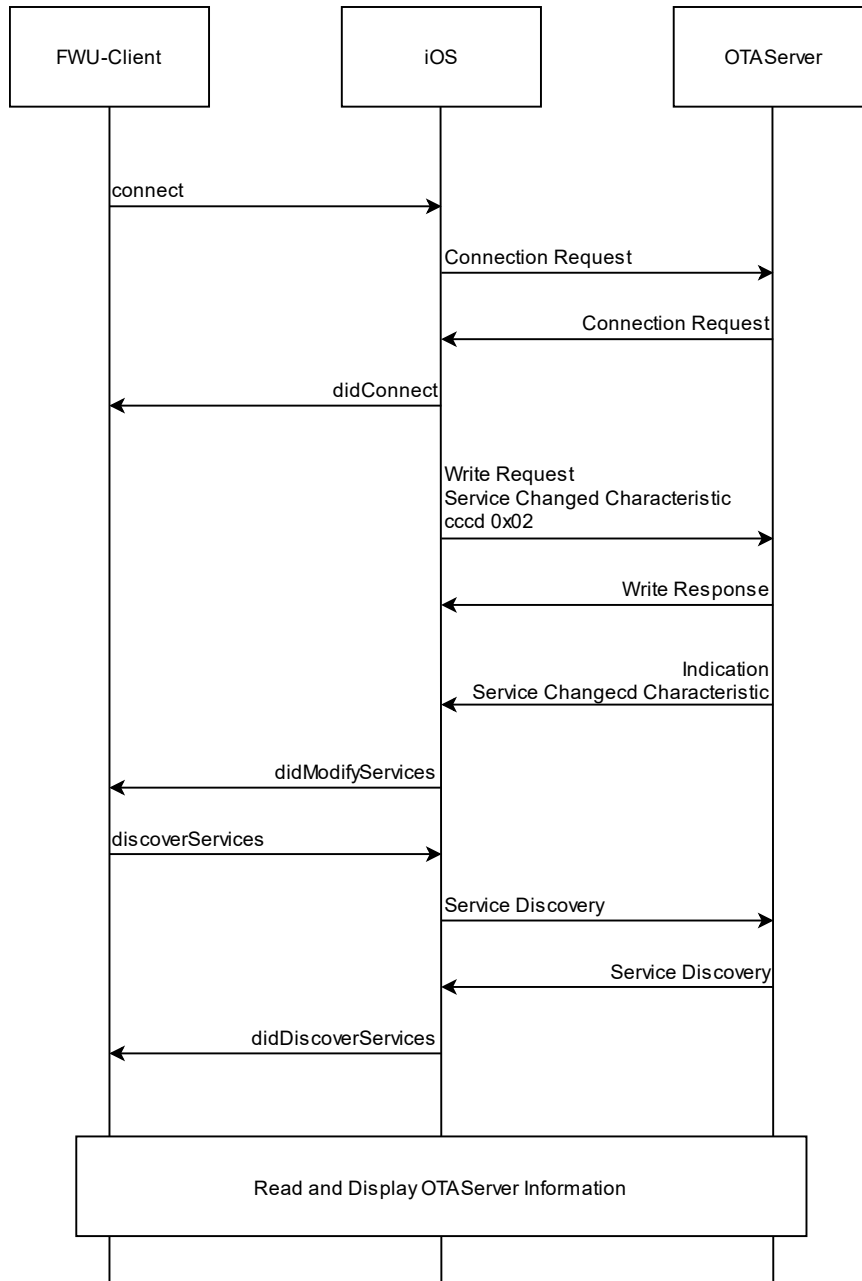


Figure 7.1 Sequence chart when connecting iOS version FWU_Client

8. FWU_Client for Python

8.1 System Configuration

Python version FWU_Client consists of a program OTA Client for RX23W and Python Application to control OTA Client. OTA Client uses Bluetooth LE to update OTA Server firmware. Python Application transfers update firmware using the application packet defined by OTA Client. The firmware transfer path communicates using the command line interface (CLI) of app_lib provided by BLE FIT Module.

Python Application receives control commands and performs firmware updates.

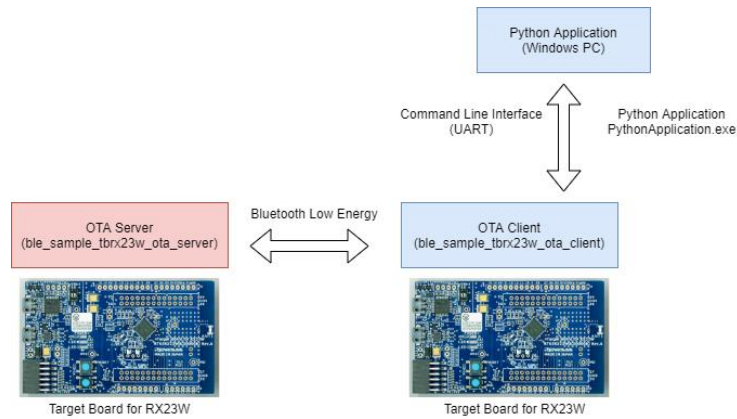


Figure 8.1 system configuration diagram

8.2 Control Command

Python Application in Python version FWU_Client receives the command in Table 8.1 and performs the OTA Server firmware update.

Table 8.1 Python Application Command List

コマンド	引数	動作
scan	timeout (s) scan_mode filter_type filter_data	OTA Client runs a scan for Timeout time. scan_mode 0: all device 1: filter_type and filter_data 2: Renesas OTA Reset Service UUID 3: Renesas OTA Service UUID example: scan 5 0 scan 5 1 9 FWU-DEV
conn	timeout (s) bd_address addr_type [public:0,random:1]	Connect OTA Server and read the firmware information. example: conn 5 74:90:50:FF:FF:FF 0 conn 5 DF:3C:31:5D:08:D9 1
update	Update_mode [app,ble] Firmware_directory_path	OT Client starts updating OTA Server devices connected by the conn command. example: update app C:\FWU Client\ota_sample_Version1.11 update ble C:\FWU Client\ota_sample_Version1.12
disconn	None	OTA Client disconnects the connection with the connected device
exit	None	Terminate Python Application

8.3 Operation Sequence of Each Command

This Chapter shows the operation sequence at each command execution of the Python version FWU_Client. See Section 8.4.2 for more information on packets between OTA Client and Python Application.

8.3.1 scan command

The scan command scans the peripheral Bluetooth LE device and displays the data in the advertisement packet. The red letters in the figure indicate the packet between the OTA Client and the Python Application.

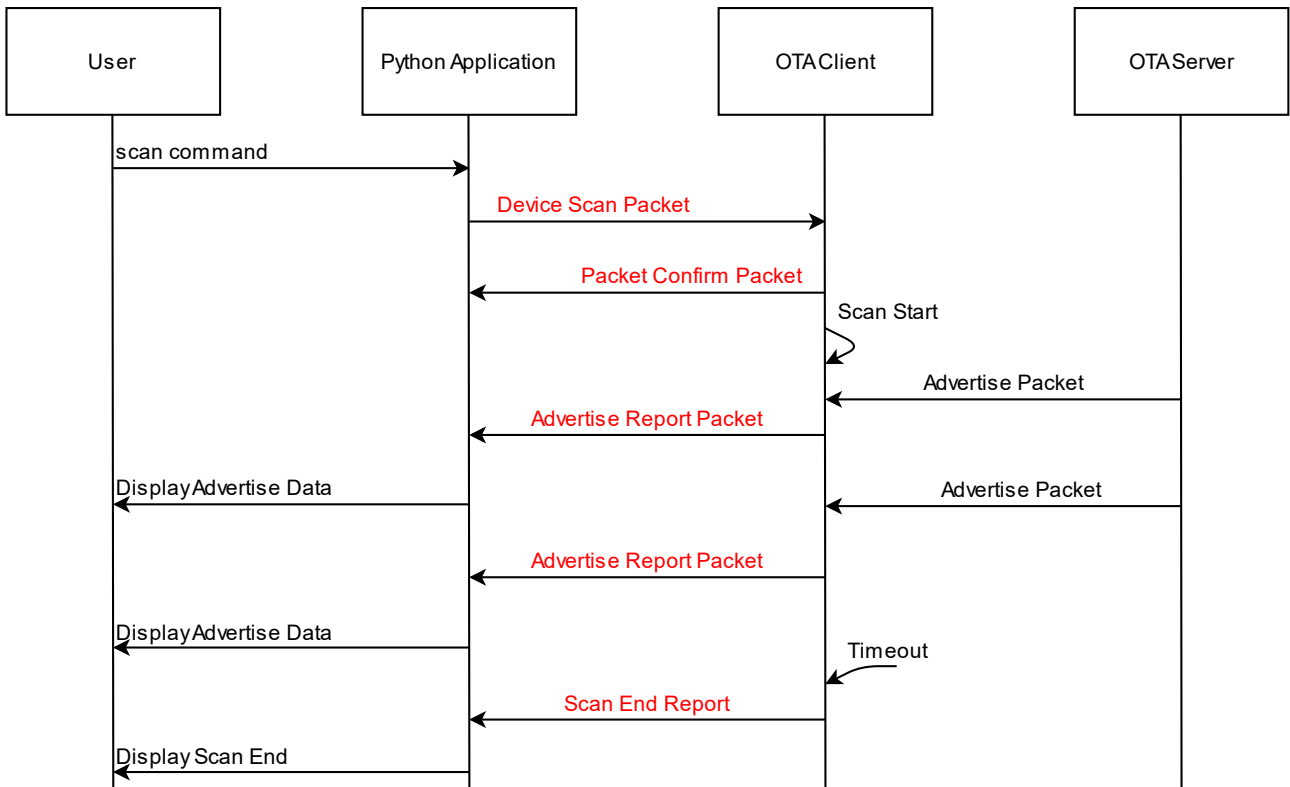


Figure 8.2 scan command execution sequence

8.3.2 conn command

The conn command connects to the OTA Server and reads and displays the firmware information. If you connect to a non-OTA Server, OTA Client will be disconnected. If the key information of OTA Client is different from that of OTA Server, delete the key information of OTA Client and disconnect. The red letters in the figure indicate the packet between the OTA Client and the Python Application.

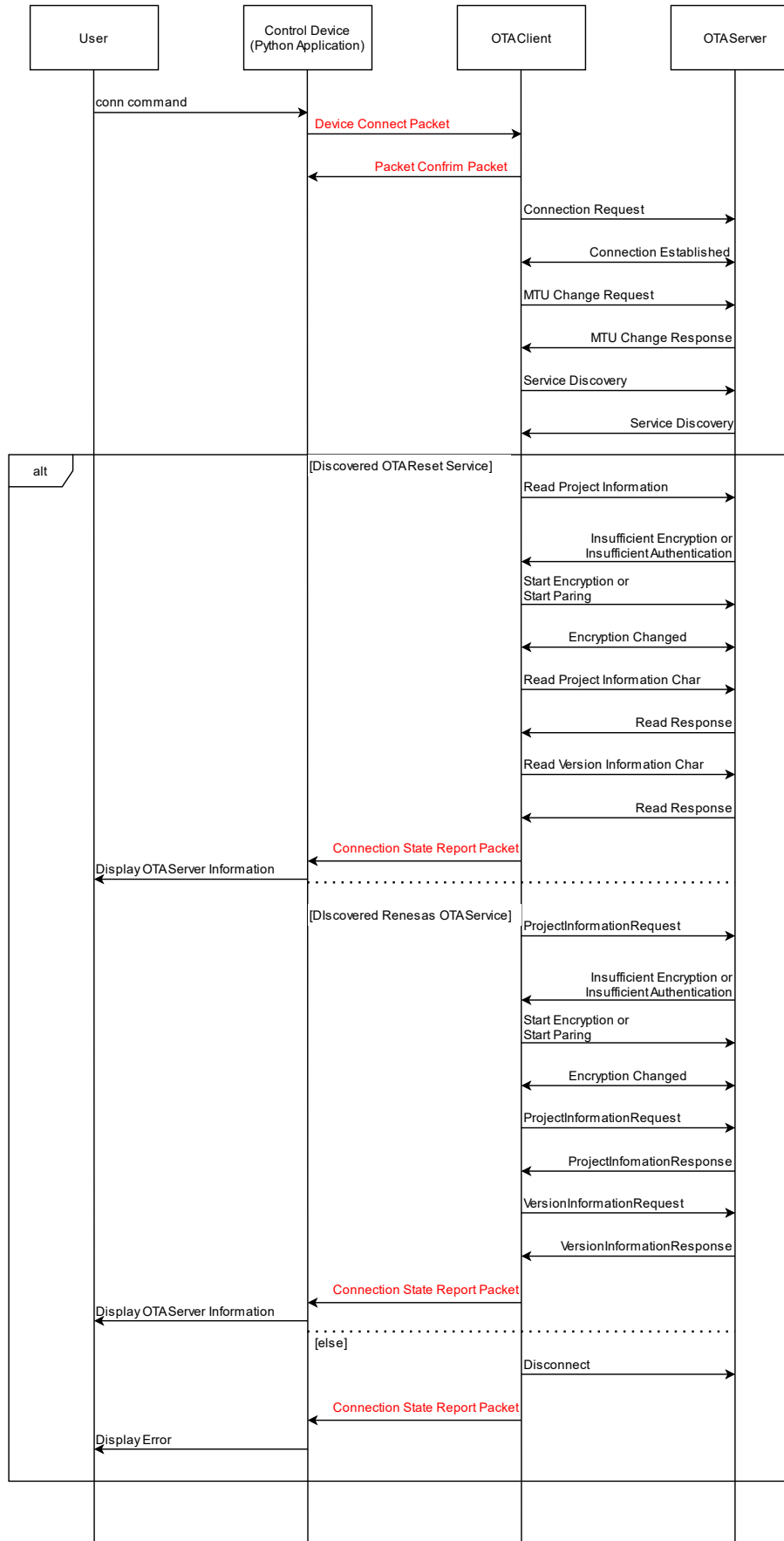


Figure 8.3 conn command execution sequence

8.3.3 disconn command

The disconn command disconnects from the OTA Server. The red letters in the figure indicate the packet between the OTA Client and the Python Application.

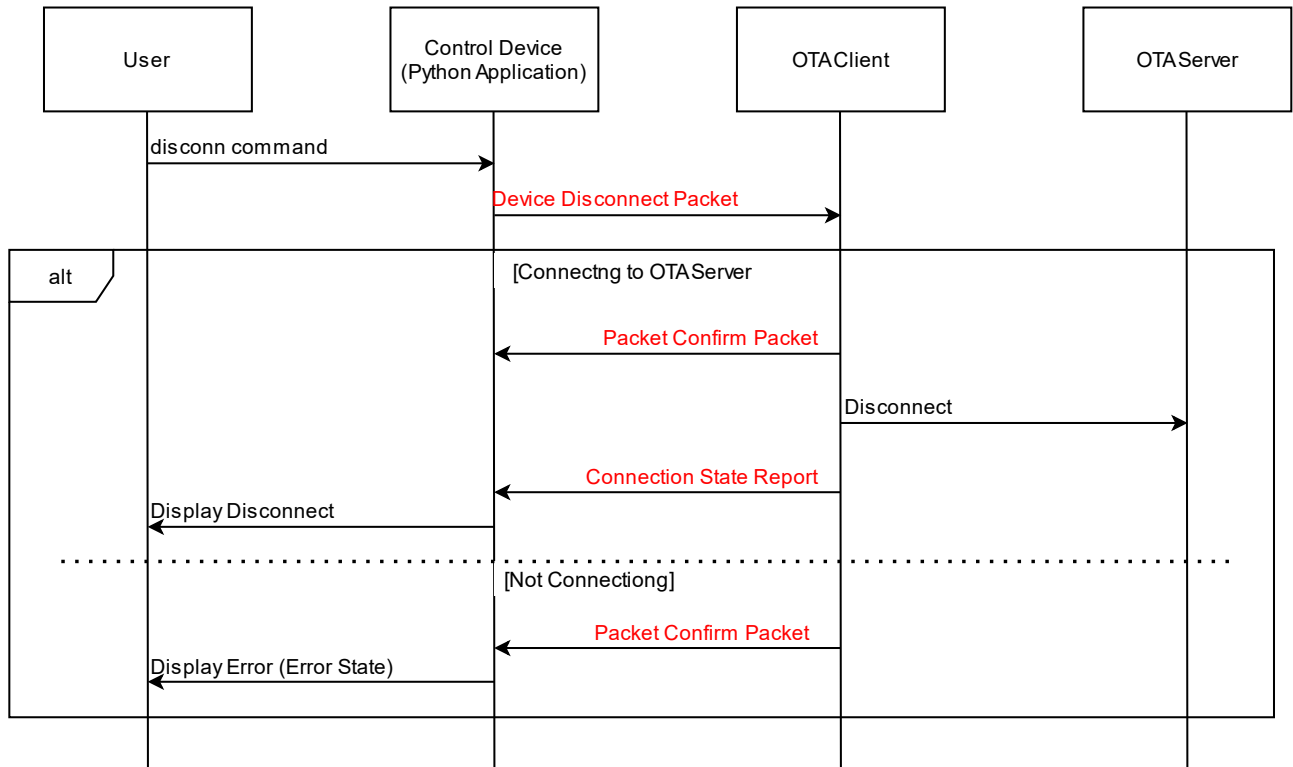


Figure 8.4 disconn command execution sequence

8.3.4 update command

The update command executes the OTA update sequence for the OTA Server shown in Section 3.5. Get the firmware from the Python Application each time OTA Client send a firmware block. The red letters in the figure indicate the packet between the OTA Client and the Python Application.

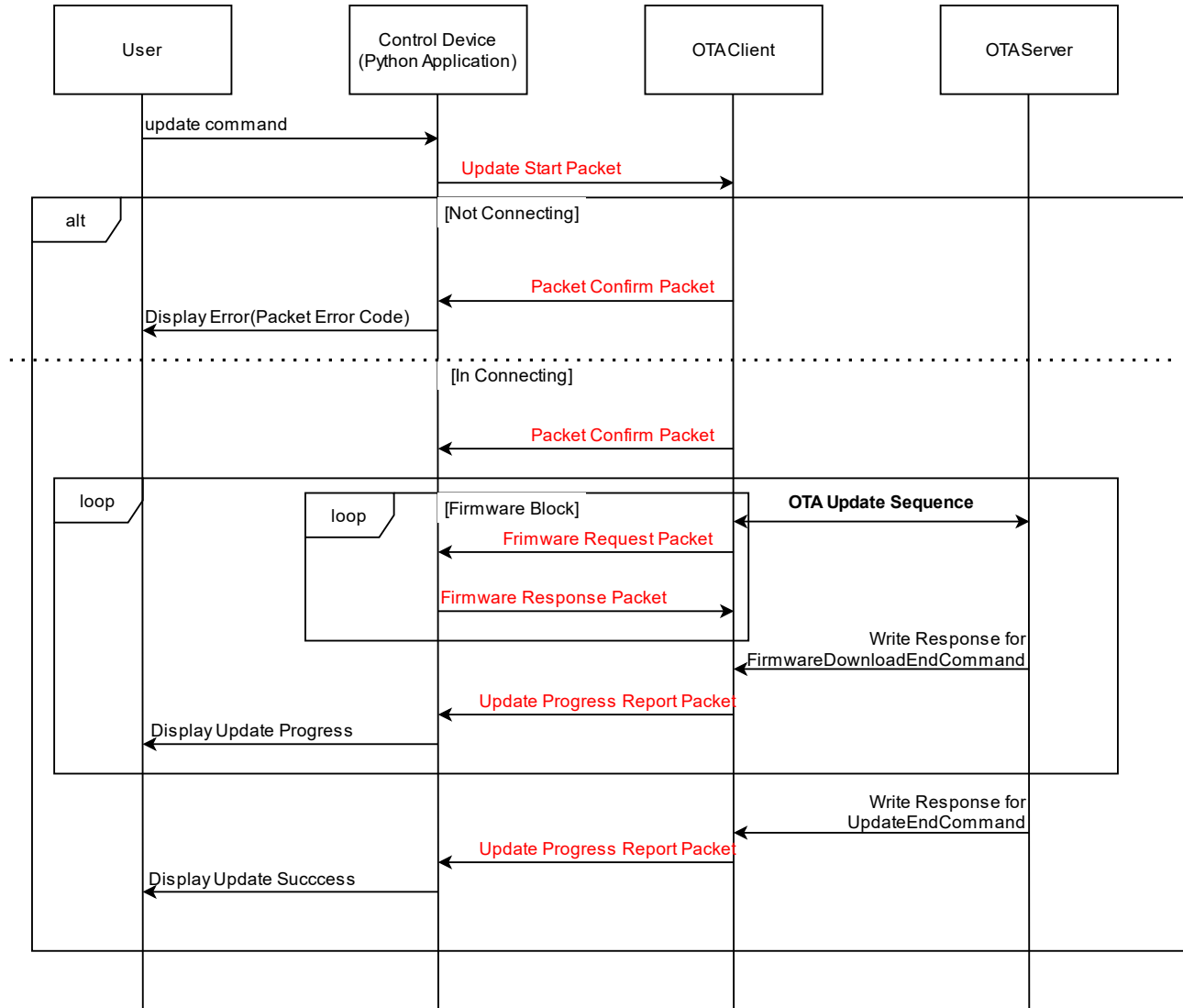


Figure 8.5 update command execution sequence

8.3.5 exit command

The exit command is a command to exit the Python Application.

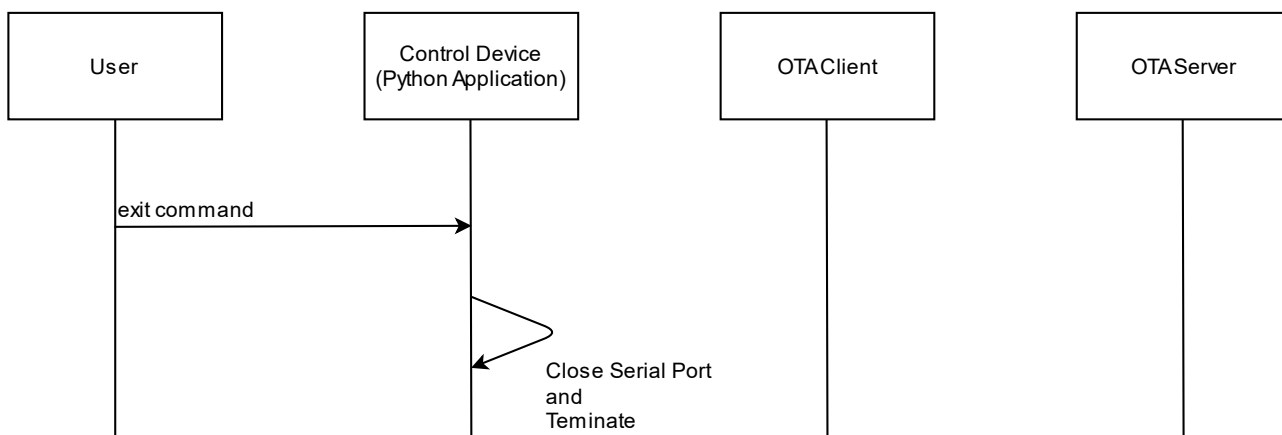


Figure 8.6 exit command execution sequence

8.4 OTA Client

OTA Client has firmware update function for the OTA Server. OTA Client provides the update functionality by application packets defined in Section 8.4.2. The Device that controls OTA Client by application packet is called as a "Control Device".

In this sample, Python Application is implemented as an OTA Client "Control Device".

8.4.1 Block Diagram

OTA Client consists of three modules and main modules that use it.

Figure 8.7 shows a block diagram.

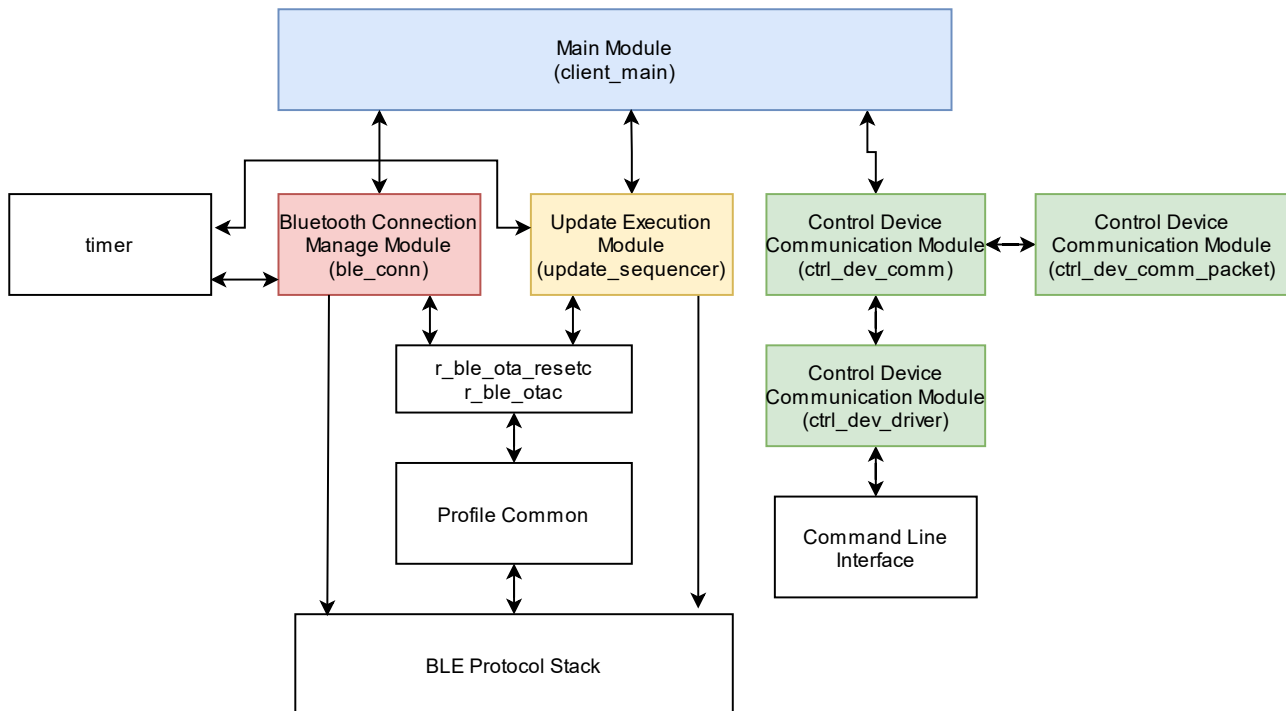


Figure 8.7 OTA Client block diagram.

Describes the functionality of each module.

1. Bluetooth connection management module (ble_conn)

Ble_conn scan, connects to the OTA Server device and read project information.

2. Update execution module (update_sequencer)

Update_sequencer updates the firmware of the connected OTA Server.

3. Control Device Communication Module (ctrl_dev_comm, ctrl_dev_comm_packet, ctrl_dev_driver)

Ctrl_dev_comm notifies the main module to receive application packets defined in Section 8.4.2.

Ctrl_dev_comm_packet implements data structures and serialization functions, and deserialization functions of application packets defined in Section 8.4.2.

Ctrl_Dev_Driver notifies Ctrl_Dev_COMM for application packet data received from any communication path. This sample receives application packets using the BLE FIT Module's app_lib's command line interface function.

4. Main Module(client_main)

client_main uses the three modules above to provide OTA Server firmware updates. Main module initialize and set communication parameters each module.

8.4.2 Application Packet

OTA Client defines application packets for searching peripheral devices, connections to update devices, and executing connection device firmware update sequences. Control devices and OTA Client perform data communication with packets defined below

OTA Client Communication Packets have the following data structures. N depends on the communication path. N is 70-255 bytes. The checksum field is a value obtained by adding packet data bytes from the packet type code by one byte.

Packet type code 1byte	Packet data N byte	checksum 1byte
---------------------------	-----------------------	-------------------

Each packet data and operation are as follows.

Packet Name	Device Scan			Packet type code	0x01
Sender	Control Device				
Client Behavior	OTA Client filters advertisement packets on the specified data. OTA Client runs a scan for a specified amount of scan time. Device scan results are notified in Advertisement Report packets.				
Packet data					
Packet type code	Scan time (sec)	Filter data type	Filter data length	Filter data	Checksum
1byte	1byte	1byte	1byte	N byte	1byte

Packet Name	Device Connect			Packet type code	0x02
Sender	Control Device				
Client Behavior	OTA Client connects to the specified Bluetooth LE device. The result of the connection is notified in the Connection state report packet.				
Packet data					
Packet type code	Timeout (sec)	BD Address 74:50:90:ff:00:ff → 0x74,0x50,0x90,0xff,0x00,0xff	Address type: random 0x01 public 0x00	Checksum	
1byte	1byte	6byte	1byte	1byte	

Packet Name	Device Disconnect	Packet type code	0x03
Sender	Control Device		
Client Behavior	The OTA Client disconnects from the connected device. The disconnection result is notified in the connection state report packet.		
Packet data			
Packet type code	Checksum		
1byte	1byte		

Packet Name	Update Start	Packet type code	0x04			
Sender	Control Device					
Client Behavior	OTA Client performs a firmware update sequence for the connected device. Select application update mode or BLE protocol stack update mode in update mode field. During the update, the OTA Client sends update progress packets and firmware request packets.					
Packet data						
Packet type code	Update mode 0x00: Application Update 0x01: BLE protocol stack update	Start Address for Variable Vector Table section	Length for Variable Vector Table section (byte)	Start Address for Application section	Length for Application Section (byte)	
1byte	1byte	4byte	4byte	4byte	4byte	
Start Address for Downloader section	Length for Downloader section (byte)	Version Information	Project Name	Reset Code	Firmware Block size (KB)	Checksum
4byte	4byte	6byte	18byte	18byte	1byte	1byte

Packet Name	Firmware Response	Packet type code	0x05	
Sender	Control Device			
Client Behavior	This packet responds to the firmware request packet notified by the OTA Client. Sends firmware of the address and size specified in the firmware request packet.			
Packet data				
Packet type code	Firmware Request number	Firmware Data length	Firmware data	Checksum
1byte	1byte	1byte	N byte	1byte

Packet Name	Error Confirm	Packet type code	0x06	
Sender	Control Device			
Client Behavior	Packets sent when the checksums of packets notified by the OTA Client do not match			
Packet data				
Packet type code	Response Packet Code	Packet Error Code	Checksum	
1byte	1byte	1byte	1byte	

Packet Name	Advertise Report			Packet type code	0x81
Sender	OTA Client				
Client Behavior	The response to the device scan packet. Notifies to the control device the device found by the scan.				
Packet data					
Packet type code	BD Address	BD Address type random:0x01 public:0x00	Advertise Data Length	Advertise Data	Checksum
1byte	6byte	1byte	1byte	N byte	1 byte

Packet Name	Scan End Report			Packet type code	0x82
Sender	OTA Client				
Client Behavior	Notifies the control device that the scan is complete.				
Packet data					
Packet type code	Checksum				
1byte	1byte				

Packet Name	Connection State Report			Packet type code	0x83
Sender	OTA Client				
Client Behavior	Notifies the information of the connected Bluetooth LE device.				
Packet data					
Packet type code	Connection state 0x00: success 0x01: failed 0x02: read failed 0x03: timeout 0x04: UUID not match 0x05: disconnected	BD Address	Project Name	Version Information	Executing program in connected OTA Server 0x00: user application 0x01: downloader
1byte	1byte	6byte	18byte	8byte	1byte
Checksum					
1byte					

Packet Name	Update Progress Report			Packet type code	0x84	
Sender	OTA Client					
Client Behavior	Notifies the control device of the progress of the OTA Server firmware update. Periodically sent to the control device during the update sequence.					
Packet data						
Packet type code	Update State 0x00: success 0x01: in updating 0x02: update failed	Update failed reason	Updating Firmware address	Number of sent blocks.	Number of total blocks.	Checksum
1byte	1byte	1byte	4byte	1byte	1byte	1byte

Packet Name	Firmware Request			Packet type code	0x85	
Sender	OTA Client					
Client Behavior	Request the control device for the firmware required for the update.					
Packet data						
Packet type code	Firmware Request Number	Start Address	Length			Checksum
1byte	1byte	4byte	1byte			1byte

Packet Name	Packet Confirm			Packet type code	0x86	
Sender	OTA Client					
Client behavior	Returns a response to device search, device connection, device disconnection, and update start packets.					
Packet data						
Packet type code	Response packet Code	Packet Error Code	Checksum			
1byte	1byte	1byte	1byte			

Table 8.2 Packet Error Code List

Packet Error Code	Error Code Name	Error Reason
0x00	Success	Success
0x01	Parameter Error	Argument error.
0x02	Status Error	OTA Client is not accepting packets.
0x03	Checksum Error	Checksum mismatch
0x04	Invalid Error	Others

8.4.3 Communication with command line interface

The OTA Client of this sample communicates with the control device using the Command Line Interface provide by BLE FIT Module. The control device enters the 1-byte data as a 2-byte string. For example, the command structure when executing connection request command is as follows:

```
ota ctrl 02745090ff00ff0054
```

Packet transmission to control devices uses the CLI R_BLE_CLI_Printf function. One byte data is displayed with "app:" first as a two-character string. For example, firmware data request packet is output as follows when OTA Client requests a firmware of 0xF0Byte from the firmware's 0xFFFF80000 address

```
app:84010000f8ff0f8B
```

Communication paths that send and receive application packets are implemented in r_ble_ota_cl_ctrl_dev_drive.h. By changing this implementation, you can exchange application packets in your own communication path. Implement the functions defined in r_ble_ota_cl_ctrl_dev_drive.h. In addition, the binary data of the packet received from the communication path is notified to the callback function passed by the R_BLE_OTA_CL_CTRL_DEV_DRIVER_Init function.

8.4.4 Correction Point for app_lib

OTA Client has Corrected some of the app_lib provided by BLE FIT Module to handle the OTA Server GATT Database change and to be able to communicate Command Line Interface. The fixes and reasons for modifications are as follows.

Table 8.3 Correction Point for app_lib

Fix points	Reason	Switching Macro
src/r_ble_rx23w/app_lib/cli/r_ble_cli.c src/r_ble_rx23w/app_lib/cli/r_ble_cli.h	Extends maximum one line size in CLI. Stop echo back when executing CLI command.	BLE_OTA_CLI_CONFIG
src/r_ble_rx23w/app_lib/profile_cmn/r_ble_servc_if.h	Add GATT Database Initialization Function.	BLE_OTA_SERVC_CONFIG

8.4.5 Using High Speed Communication Sample Program

OTA Client uses the Flow Control API of the high-speed Communication Sample Program to transfer firmware to OTA Server. For a high-speed communication sample program, please refer to the following documents.

- RX23W Group High-Speed Communication Sample Program (R01AN5437)

8.4.6 Notes

- Slow Period implemented in scan parameter in r_ble_ota_cl_client_main.c are set zero.
- OTA Clients cannot connect to multiple OTA Servers at the same time.

8.5 Python Application

Python Application controls OTA Client based on control commands. Python Application acts as an OTA Client control device. The Python Application feature provides control OTA Client by application packet and control command interface.

8.5.1 Block Diagram

Python Application consists of seven modules. Figure 8.8 shows a block diagram of each module.

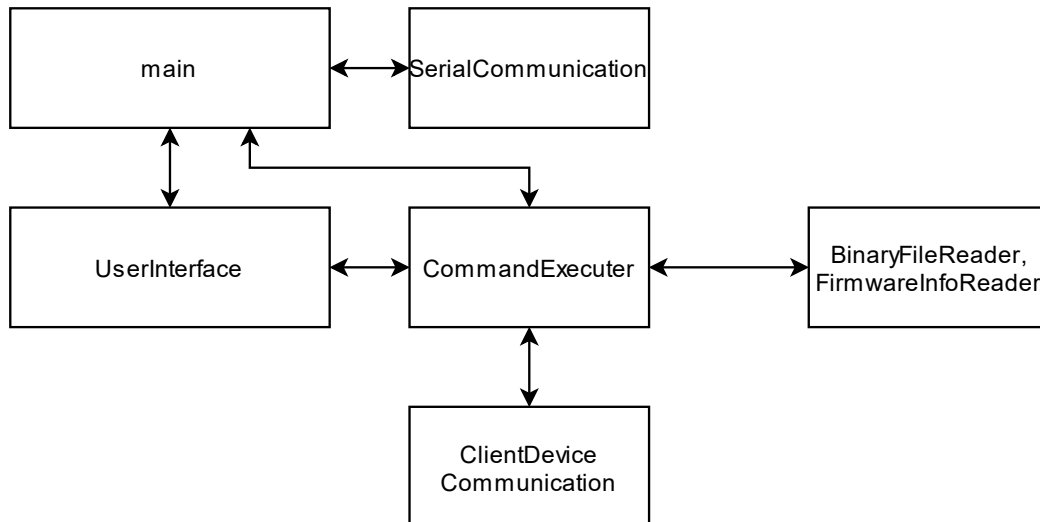


Figure 8.8 Block Diagram for Python Application

The functions of each module are as follows.

- main.py
This module parse and run the SERIAL port settings and the entered commands.
- SerialCommunication.py
This module sends and receive packet data with the OTA Client using the PySerial module.
- UserInterface.py
Standard I / O wrapper layer.
- CommandExecuter.py
This module creates an application packet from the parameters of the input control command and execute the sequence shown in Chapter 0.
- ClientDeviceCommunication.py
This module defines data structures in chapter 8.4.2 and Implemented serialization functions of application packets.
- BinaryFileReader.py, FirmwareInfoReader.py
This Module read the firmware.bin file and firmwareInfo.json file of the update firmware each.

8.5.2 Log Outputs

Python Application outputs the display content of the user interface and the communication history of the application packet to a text file.

Table 8.4 Logs output by Python Application

Filename	Contents
AllSerialLog.txt	Application packet communication history.
UserInterfaceLog.log	User Interface Command Log

9. Modification from Version 1.00

9.1 Implement Service Changed Characteristic for user applications

In this sample, the Service Changed Characteristic Indication is performed after writing Service Changed Characteristic CCCD.

9.2 Update FIT version

BLE FIT Module version has been changed from 2.11 to 2.20. In addition, the version of the FIT module of the General-Purpose Input / Output Driver (GPIO) included in the OTA Client and OTA Server projects has been changed from 3.70 to 3.90.

9.3 Modification Source Code in OTA Server

Version 1.10 updates some of the OTA Server source code.

r_ble_ota/downloader/r_ble_ota_ble_app_lib/r_ble_ota_ble_interface.c

- In R_BLE_OTA_GATTS_IndicateDataControl function, changed to call R_BLE_GATTS_Indication in the function.
- Added BLE_OTA_DOWNLOADER_DEVICE_NAME defined by r_ble_ota_config.h to Advertise Packet as << Complete Device Name >>
- Changed to set BLE_OTA_DOWNLOADER_DEVICE_NAME in the Device Name characteristic of GAP Service of GATT Database for Downloader.
- Added the process to perform Indication of Service Changed Characteristic when writing CCCD of Service Changed Characteristic of GATT Service.

r_ble_ota/downloader/r_ble_ota_downloader.c

- Changed the R_BLE_OTA_GATTS_IndicateDataControl function that was executed in the BLE_OTA_DL_CMD_PROJECT_INFORMATION_REQUEST and BLE_OTA_DL_CMD_VERSION_INFORMATION_REQUEST events in the r_ble_ota_dl_data_control_cmd_handler function to be executed in the BLE_OTA_DL_EVENT_DATA_CONTROL_WRITE_COMP event of the r_ble_ota_dl_event_handler function.

r_ble_ota/r_ble_ota_config.h

- Device Name added a macro for setting

smc_gen/app_main.c

- Added the process to perform Indication of Service Changed Characteristic when writing CCCD of Service Changed Characteristic of GATT Service.

smc_gen/r_ble_gatts.h

- Added event enumeration for CCCD of Service Changed Characteristic of GATT Service.

smc_gen/r_ble_ota_resets.c

- Added Open / Close processing of R_BLE_OTA_FLASH_DRIVER before and after calling R_BLE_OTA_FLASH_DRIVER_ReadDataFlash.

10. Limitations and Notes

The OTA firmware update sample program has the following limitations and notes.

- BSP and FLASH FIT modules are placed in sections that are not subject to update, so OTA firmware cannot be updated. Therefore, do not change the BSP and FLASH settings before and after updating the firmware.
- The maximum size of the Downloader section where the BLE protocol stack and downloader are placed is less than or equal to the size of the Application section (default 198KB).
- The OTA firmware update sample program assumes that the user application is a Peripheral.
- If you change the BLE FIT module configuration, you need to update the BLE protocol stack / downloader / application.
- If the data flash block specified by BLE_CFG_EN_SEC_DATA in r_ble_rx23w_config.h does not have the IRK information of the remote device and the remote device uses RPA, the firmware update cannot be continued if the RPA of the remote device changes. Make sure to enable remote IRK requests in the pairing parameters and perform pairing in user application.
- Once you switch to downloader, you cannot start the user application until the application update is completed.
- This version does not support MCU power saving mode in downloader.
- In this version, BLE library settings are supported only for Balance type.
- BGO (background operation / interrupts) Mode of Flash FIT module cannot be used.
- The version of the CCRX compiler cannot be changed.
- Android version FWU_Client sets the MTU size to 247.
- The Android version of FWU_Client uses the API modified in Android 10.0 to read files. When using Android 10.0 or later, change the code referring to Chapter 6.3.

11. Check items when updating fails

11.1 When return Error Response from OTA Server

1 When BLE_OTA_ERROR_CODE_RESET_CODE_ERROR (0x84)

This error occurs when the ResetCode is different between FWU_Client and OTA Server. If this error occurs, check the firmware_information.json file in the update firmware. Make sure that the value of "ResetCode" in this file matches that of OTA Server.

2 When BLE_OTA_ERROR_CODE_INVALID_ADDRESS (0x81) or BLE_OTA_ERROR_CODE_INVALID_CMD (0x90)

This error occurs when the update sequence is inconsistent between FWU_Client and OTA Server. The OTA Server will reset if disconnected during the update sequence. There is a possibility that the error will not occur due to the improvement of the communication environment.

Also, if the OTA Server application uses the watchdog timer (WDT), a WDT reset can occur while the downloader is running. You can switch from the user application to the downloader via a software reset, but on the RX23W series, the software reset does not stop the WDT. Clear the WDT count in the downloader's main loop.

11.2 When it stops immediately after the update starts

When it stops immediately after the update sequence starts, it is possible that the reconnection after switching from the user application to the downloader has failed.

If the value of "ResetCode" in the firmware_information.json file is "0000000000000000", it is highly possible that the application switching process is not implemented. Replace the service API file of OTA Reset Service according to "Section 4.11.1 Switching process to Renesas OTA Reset Service and downloader".

If an interrupt occurs when switching an executable program, an illegal read to the code flash during code flash rewriting may occur and the operation may stop. If an interrupt occurs, set interrupt disable by referring to "Section 4.11.1 Switching process to Renesas OTA Reset Service and downloader".

11.3 When using the code generator function of Smart Configurator

Additional configuration and implementation are required to use the code generator functions of Smart Configurator. Figure 11.1 shows an example of the code generator function. When code is generated using this function, a function call to the application section occurs while the boot loader is running. Therefore, it may not work properly after updating the firmware.

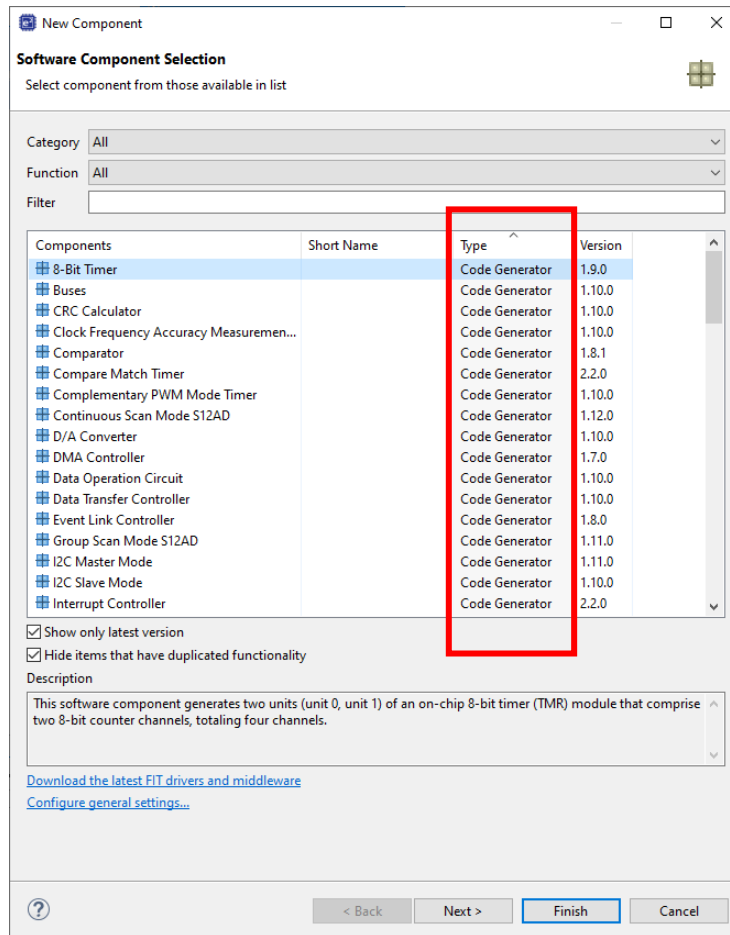


Figure 11.1 example code generator function

This section describes what to do when using this function. This guide uses an 8-bit timer as an example.

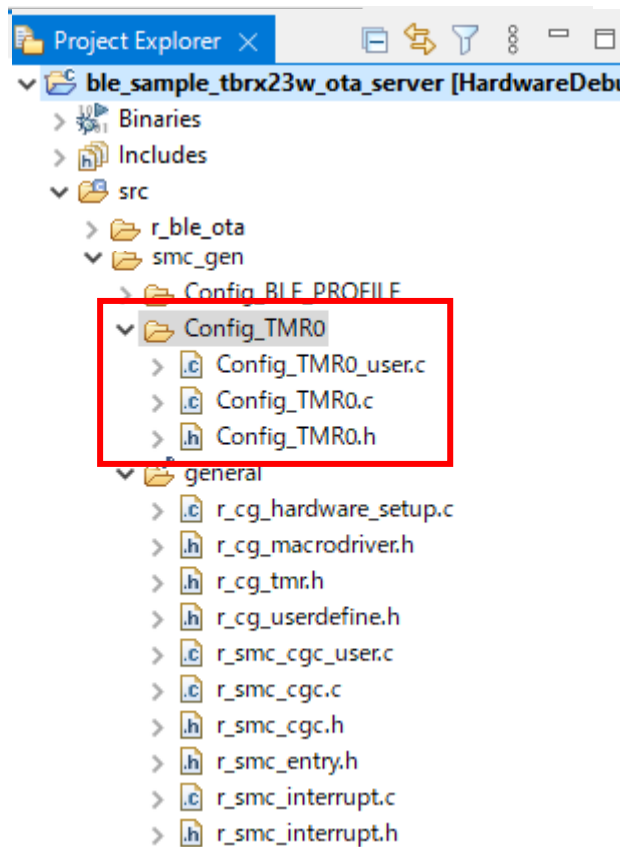


Figure 11.2 Generated folder structure by code generation function

First, disable the function call in the application section while the bootloader is running. Disables the code generation function part of the program generated in the R_Systeminit function in the general\r_cg_hardware_setup.c file. Add a yellow highlight area.

```

void R_Systeminit(void)
{
    /* Enable writing to registers related to operating modes, LPC, CGC, software
reset and LVD */
    SYSTEM.PRCR.WORD = 0xA50FU;

    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Write 0 to the target bits in the POECR2 registers */
    POE.POECR2.BYTE = 0x00U;

    /* Initialize clocks settings */
    R_CGC_Create();

    #if 0
    /* Set peripheral settings */
    R_Config_TMR0_Create();
    #endif

    /* Register undefined interrupt */
    R_BSP_InterruptWrite(BSP_INT_SRC_UNDEFINED_INTERRUPT, (bsp_int_cb_t)r_undefined_exc
eption);

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;
}

```

Figure 11.3 Disable Code Generation

Next, implement a register setting function in `app_main.c` that has the same function as the `R_Systeminit` function for the user application. Here, the function name is `R_Systeminit_App`. Here, implement it in the `app_main.c` file. There are two processes to add.

- Include configuration header files
- Implementation of `R_Systeminit_App` function (no need to call `R_CGC_Create` function and `R_BSP_InterruptWrite` function)

```

#include "Config_TMR0.h"
.....
void R_Systeminit_App(void)
{
    /* Enable writing to registers related to operating modes, LPC, CGC, software
    reset and LVD */
    SYSTEM.PRCR.WORD = 0xA50FU;

    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Write 0 to the target bits in the POECR2 registers */
    POE.POECR2.BYTE = 0x00U;

    /* Set peripheral settings */
    R_Config_TMR0_Create();

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;
}

```

Figure 11.4 Header inclusion and implementation of R_Systeminit_App function

Finally, call the R_Systeminit_App function immediately after calling the R_BLE_Open function in the app_main function. You will lose access to the application section while the bootloader is running.

```

void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();

    /* Hint: Input process that should be done before main loop such as calling
    initial function or variable definitions */
    /* Start user code for process before main loop. Do not edit comment generated
    here */
    /* Add System Init */
    R_SystemInit_App();

    /* Configure CommandLine */
    R_BLE_CLI_Init();
}

```

Figure 11.5 Example of calling R_Systeminit_App in app_main function

12. Appendix

12.1 OTA firmware update time (reference value)

Shows the update time of the included sample firmware.

Updating Version	Update Size	RX23W	iPhone8 (iOS)	ZenFone 4 (Android)
Version1.10 to Version 1.11	64KB Application Section: 64KB	20(sec)	29 (sec)	160 (sec)
Version1.10 to Version 1.12	218KB Application Section:64KB Downloader Section:154KB	70(sec)	97 (sec)	450(sec)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	June 30, 2021	—	First edition issued.
1.10	Sep 30, 2021	5	Add Update for iOS in section 1.1.
		21	Add FWU_Client for iOS in section 2.4.
		72	Add Service Changed Characteristic Indication in section 4.11.3.
		77	Add FWU_Client for iOS in section 7.
		79	Add FWU_Client for Python in section 8.
		96	Add Modification from Version1.00 in section 9.
		97	Add 10. Limitations and Notes in section 10.
		103	Add Firmware Update time for iOS device in section 12.1.
1.11	Apr 12, 2022	39	Add Operation in error response in section 3.5
		45	Add Operation when power is cut off in section 3.8
		47	Add Operation when power is cut off in section 3.9
		53	Add Fixed an omission in the standard library section (SL_*) in the Setting the section that maps from ROM to RAM
		68	Chapter 4.10 has been divided into chapters for user application settings and Chapter 4.11 for adding user application functions.
		70	Added the explanation about the implementation of the r_ble_ota_resets.c file.
		98	Added confirmation items when updating fails in section 11.
		-	Correction of minor expressions

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity.

Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state

Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).

7. Prohibition of access to reserved addresses

Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
 - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
 - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.