Bluetooth® Low Energy Protocol Stack
Sample Program

Introduction
This manual describes the installation, configuration and usage of sample program, which is included in the Bluetooth Low Energy software (the BLE software).

The BLE software refers to the set of software that includes the Bluetooth Low Energy protocol stack (the BLE protocol stack) compliant with the Bluetooth Low Energy specification (Bluetooth specification v4.2). The BLE protocol stack is designed to run on the Bluetooth Low Energy microcontroller RL78/G1D.

Target Device
RL78/G1D

Contents

1. Overview ........................................................................................................................... 4

2. Applicability ...................................................................................................................... 4

3. Installation ........................................................................................................................ 4

3.1. Contents ....................................................................................................................................... 4

3.2. Installation Procedure ................................................................................................................ 5

4. Sample Program ............................................................................................................... 5

4.1. Operating Environment and Development Environment ........................................................ 5

4.2. Structure ...................................................................................................................................... 6

5. Usage of Console-based Sample Program .................................................................... 8

5.1. How to Change Parameters ....................................................................................................... 8

5.2. Start the Sample Program in Modem Configuration ............................................................... 9

5.3. Start the Sample Program in Embedded Configuration ......................................................... 9

5.4. Usage of Console-based Sample Program ............................................................................ 10

5.5. Generic Access Profile (GAP) .................................................................................................. 12

5.6. Security Manager (SM) ............................................................................................................. 13

5.7. Generic Attribute Profile (GATT) ............................................................................................. 17

5.8. Find Me Profile (FMP) ............................................................................................................... 19

5.9. Proximity Profile (PXP) ............................................................................................................. 22

5.10. Health Thermometer Profile (HTP) ........................................................................................ 25

5.11. Blood Pressure Profile (BLP) ................................................................................................ 28

5.12. HID over GATT Profile (HOGP) ............................................................................................ 31

5.13. Scan Parameters Profile (ScPP) ............................................................................................ 34
5.14. Heart Rate Profile (HRP) ................................................................. 37
5.15. Cycling Speed and Cadence Profile (CSCP) ................................. 41
5.16. Cycling Power Profile (CPP) ............................................................ 45
5.17. Alert Notification Profile (ANP) .................................................... 49
5.18. Location and Navigation Profile (LNP) ............................................ 52
5.19. Vendor Specific (VS) ....................................................................... 56

6. Usage of Simple Sample Program ....................................................... 59
   6.1. Configuration ................................................................................. 59
   6.2. HEX File Preparation ................................................................. 59
   6.3. Behavior ..................................................................................... 59
   6.4. Check with Android Device ....................................................... 60
   6.5. Check with iOS Device ............................................................... 62

7. Appendix .......................................................................................... 64
   7.1. Transmit and Receive Operations in the Sample Program for the Computer 64
   7.2. Requirements and Flow Chart of Serial Communication Driver on APP MCU 67
      7.2.1. Transmit Procedure Example using the UART 2-wire Connection Method 70
      7.2.2. Receive Procedure Example using the UART Two-wire Connection Method 70
      7.2.3. Transmit Procedure Example using the UART 3-wire Connection Method 71
      7.2.4. Transmit Procedure Example using the UART 2-wire with Branch Connection Method 72
      7.2.5. Receive Procedure Example using the UART 3-wire and 2-wire with Branch Connection Methods 73
      7.2.6. Transmit Procedure Example using the CSI 4-wire Connection Method 74
      7.2.7. Transmit Procedure Example using the CSI 5-wire Connection Method 75
      7.2.8. Receive Procedure Example using the CSI 4-wire and 5-wire Connection Method 76
      7.2.9. Transmit Procedure Example using the IIC 3-wire Connection Method 77
      7.2.10. Receive Procedure Example using the IIC 3-wire Connection Method 77
   7.3. Porting of the Sample Program ..................................................... 79
   7.4. How to use the Direct Test Mode ................................................ 80
      7.4.1. Direct Test Mode (Receiver) .................................................. 81
      7.4.2. Direct Test Mode (Transmitter) ............................................. 82
      7.4.3. Direct Test Mode (Parameter Set) .......................................... 83
   7.5. Sample Custom Profile ............................................................... 85
      7.5.1. Sample Custom Profile Specification ................................... 85
      7.5.2. File Structure Corresponding to Sample Custom Profile ........ 86
      7.5.3. API Functions defined for Sample Custom Profile .................. 87
      7.5.4. Events defined for Sample Custom Profile ............................. 91
      7.5.5. Usage of the Sample Program for Sample Custom Profile ...... 93
   7.6. Simple Sample Profile ................................................................. 97
      7.6.1. Characteristic Specification ................................................. 97
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6.2.</td>
<td>File Structure</td>
<td>97</td>
</tr>
<tr>
<td>7.6.3.</td>
<td>Details of Simple Sample Profile</td>
<td>97</td>
</tr>
<tr>
<td>7.7.</td>
<td>Sample Program for the Direct Test Mode with RF Tester</td>
<td>98</td>
</tr>
<tr>
<td>7.8.</td>
<td>Printf program in the Embedded configuration</td>
<td>101</td>
</tr>
<tr>
<td>7.9.</td>
<td>FW Update Sample Program</td>
<td>102</td>
</tr>
<tr>
<td>7.9.1.</td>
<td>FW Update Profile Specification</td>
<td>102</td>
</tr>
<tr>
<td>7.9.2.</td>
<td>File Structure Corresponding to FW Update Profile</td>
<td>103</td>
</tr>
<tr>
<td>7.9.3.</td>
<td>API Functions defined for FW Update Profile</td>
<td>105</td>
</tr>
<tr>
<td>7.9.4.</td>
<td>Events defined for FW Update Profile</td>
<td>109</td>
</tr>
<tr>
<td>7.9.5.</td>
<td>Usage of the Sample Program for FW Update Profile</td>
<td>110</td>
</tr>
<tr>
<td>7.10.</td>
<td>Project Setting to use FW Update Sample Program</td>
<td>113</td>
</tr>
<tr>
<td>7.10.1.</td>
<td>Receiver device</td>
<td>113</td>
</tr>
<tr>
<td>7.10.2.</td>
<td>Sender device</td>
<td>120</td>
</tr>
<tr>
<td>7.10.3.</td>
<td>Notes of making FW Update Environment</td>
<td>121</td>
</tr>
<tr>
<td>7.11.</td>
<td>References</td>
<td>122</td>
</tr>
<tr>
<td>7.12.</td>
<td>Terminology</td>
<td>123</td>
</tr>
</tbody>
</table>
1. Overview

This manual describes the installation, configuration and usage of sample program, which is included in the Bluetooth Low Energy software (the BLE software).

The BLE software refers to the set of software that includes the Bluetooth Low Energy protocol stack (the BLE protocol stack) compliant with the Bluetooth Low Energy specification (Bluetooth specification v4.2). The BLE protocol stack is designed to run on the Bluetooth Low Energy microcontroller RL78/G1D.

For details about the BLE protocol stack APIs, see Bluetooth Low Energy Protocol Stack API Reference Manual.

2. Applicability

The descriptions in this manual apply to the BLE protocol stack Version 1.20 and later.

3. Installation

The sample program of the BLE software is included in the BLE protocol stack package.

3.1. Contents

The BLE software package includes the following:

Documents
- Bluetooth Low Energy Protocol Stack Sample Program Application Note (this document)
- rBLE Command Specification

Files used for building the executable file
- Executable file
- BLE software library
- Sample source code
- Source code that configures parameters
- CS+ for CA, CX project file
- CS+ for CC project file
- IAR Embedded Workbench workspace file
- e² studio project file

Sample program for computer
- Executable file
- Source code
- Microsoft Visual Studio Express 2015 for Desktop project file

HCI packet monitor application for computer
- Executable file
- INI file
3.2. Installation Procedure

Copy the decompressed contents to any folder in your computer.

[Note] If using the e² studio, cannot be include multi-byte characters and blank in the BLE software installation folder path.

4. Sample Program

This sample program shows how to use the BLE software. The BLE software contains two sample programs.

- Console-based Sample Program  Section 5
- Simple Sample Program  Section 6

This section describes the common concept of a sample program. Regarding the details of sample programs, refer respective dedicated section.

Caution

Sample programs in this application note shall be handled as a sample, whose quality and reliability are not guaranteed. When you use the sample program in the final products or systems manufactured by you, evaluate the safety of them at your own risk.

4.1. Operating Environment and Development Environment

The BLE software supports two different system configurations, the modem configuration and the embedded configuration. This section describes the operating environment and development environment of the sample program in each configuration.

Modem Configuration

In the modem configuration, the controller stack, host stack and profiles are implemented together on the BLE MCU (RL78/G1D), while the application is implemented on the APP MCU separately.

The BLE software provides the sample program running on the computer as the APP MCU. You can easily evaluate the BLE software using the computer.

The sample program in the modem configuration runs on the following operating environment.

Hardware

- PC/AT™ compatible computer
  - Processor : 1.6GHz and greater
  - Memory : 1.0GB and more
  - Display : 1024×768 (XGA) and higher resolution
    - 65536 and more colors
  - Interface : USB 2.0 (E1 emulator and USB TTL serial cable)

Software

- Windows 7 or later
- Microsoft Visual Studio Express 2015 for Desktop
- Microsoft .NET Framework 4 + Language Pack

Embedded Configuration

In the embedded configuration, the controller stack, host stack, profiles and the application are implemented together on the BLE MCU (RL78/G1D).

The BLE software also provides the sample program running on the BLE MCU.

The sample program in the embedded configuration runs on the following operating environment.
Hardware
- RL78/G1D Test Board

Development tools and utilities
- Renesas on-chip debugging emulator E1
- Terminal Emulator for Windows

Software
- Renesas Integrated Development Environment CS+ for CA, CX or CS+ for CC or e² studio or IAR Embedded Workbench
- Renesas Flash Programmer V3
  (You can download it from https://www.renesas.com/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html)

4.2. Structure

Figure 4-1 shows the structure of the BLE software.

Figure 4-1 Structure of the BLE software
The BLE software in the modem configuration runs on two MCUs that are APP MCU and BLE MCU, and consists of ‘rBLE_Host’ block (block in the figure) running on the APP MCU and the software blocks (blocks in the figure) running on the BLE MCU.

In addition, the software blocks (blocks in the figure) that you need to prepare is ‘application’, ‘serial communication driver’ and ‘OS’ (Operating System) blocks. However, ‘rBLE_Host’ block does not use any OS specific resources, ‘OS’ block is not required if it does not run on the APP MCU.

On the other hand, the BLE software in the embedded configuration runs on the BLE MCU (RL78/G1D) only. The software block that you need to prepare is ‘application’ block running on the BLE MCU.
5. Usage of Console-based Sample Program

5.1. How to Change Parameters

The console-based sample program has the ability to change the parameters for rBLE API, and you will be able to execute it by selecting the parameters prepared in advance.

Parameters selection is performed as follows.

```
menu-number [blank] parameter-number
```

In the function which is called at the time of execution of the menu, treats the given arguments separated by a space and calls rBLE function.
5.2. Start the Sample Program in Modem Configuration

The console-based sample program in the modem configuration is started by executing the EXE file ‘rBLE_Sample.exe’ that is stored in the folder ‘\Renesas\BLE_Software_Ver_X_XX\BLE_Sample\project\windows\Exe’.

The sample program ‘rBLE_Sample.exe’ requires arguments at its start time, please edit the contents of the batch file “run.bat” stored in the same folder as the EXE file and execute it. The arguments required at the start time are explained below.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM Port Number</td>
<td>Specify the COM port number in the computer (e.g., COM1, COM2, …)</td>
</tr>
<tr>
<td>Baud rate</td>
<td>Specify between 4,800 and 250,000 to match the settings of the BLE software</td>
</tr>
<tr>
<td>BD Address of remote device (public address)</td>
<td>Set the BD address (Bluetooth device address) of the remote device to be connected to. With this address, it is not required to obtain the BD address of remote device using device search, and connection procedure can be started immediately. Use public address as BD address.</td>
</tr>
<tr>
<td>UART 2-wire Branch Connection</td>
<td>UART 2-wire with Branch Connection : -div2wire UART 2 wire : none</td>
</tr>
</tbody>
</table>

Write the program into the BLE-MCU using the HEX file ‘RL78_G1D_CM(*).hex’ or ‘RL78_G1D_IM(*).hex’ or ‘RL78_G1D_CCM(*).hex’ stored in the following folder after the BLE software installation from package ‘\Renesas\BLE_Software_Ver_X_XX\RL78_G1D\ROM_File’ after installation.

These HEX files are used in 4800 bps with baud rate of serial communication.

5.3. Start the Sample Program in Embedded Configuration

Before starting the sample program in the embedded configuration, write the program into the RL78/G1D Test Board using the HEX file ‘RL78_G1D_CE(*).hex’ or ‘RL78_G1D_IE(*).hex’ or ‘RL78_G1D_CCE(*).hex’ stored in the following folder after the BLE software installation from package ‘\Renesas\BLE_Software_Ver_X_XX\RL78_G1D\ROM_File’ after installation.

To start the sample program, reset the RL78/G1D Test Board.

However, to use this sample program, the RL78/G1D Test Board and computer should be connected each other by the USB TTL serial cable, and you should enter commands to the sample program from the terminal emulator running on the computer.

Please setup the serial port of terminal emulator as shown below. In addition, the new-line code on receive for the terminal emulator is set to LF (LF only).

<table>
<thead>
<tr>
<th>Port Setting</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baud rate</td>
<td>250,000 bps</td>
</tr>
<tr>
<td>Data length</td>
<td>8 bit</td>
</tr>
</tbody>
</table>
Note that the BD Address of remote device is not required for the sample program in the embedded configuration, it does the device search automatically.

Figure 5-1 shows the screen shot of terminal setup window in the terminal emulator (Tera Term). In the following, it is described in the screenshot when the EXE file is executed.

![Figure 5-1 Terminal Setup window (Tera Term)](image)

### 5.4. Usage of Console-based Sample Program

When you start the console-based sample program at the command prompt, Table 5-2 shows the main menu.

[Note] When the number of the implementation profile is changes, the command number may change.

![Figure 5-2 Sample Program Start Screen](image)

Please confirm that the message “rBLE Mode (ACTIVE)” is displayed. If this message is not displayed, there is some problem and the sample program does not start successfully. Please check the cable connection or settings again.
The console-based sample program executes the operation which you may choose the menu item by its number. It shows the following main menu at the start time.

```
-- BLE Sample Program Menu --
1. GAP & SM & GATT Test
2. Profile Test
3. Vendor Specific Test
ESC Key: Menu exit
```

**Figure 5-3 Main menu at the start time**

At the main menu, there are three menu items. You can choose the menu item by its number.

In this screen, the menu items from 1 to 4 are displayed.
When you want to select the menu item, type its number and ENTER key.
When you want to go back to the previous menu, type ESC key.
When you want to see the current menu list again, type ENTER key.
When you want to exit the sample program, go back to the main menu by ESC key and enter ESC key again to terminate the sample program.

In addition, log output is displayed in different colors (using ANSI escape sequence).

The cyan notation means command execution (it called rBLE API), the yellow notation means event notification (its rBLE callback function is called), as shown in the following figure.

```
27. GAP Channel Map Req
28. GAP Read RSSI
29. SM Set_Sec
30. SM Start_Enc
31. SM TK Proc Res
32. SM L4F Proc Resp
33. SM Rk Proc Resp
34. SM Ckr Proc Resp
36. SM Chr_Id Addr Req Resp
37. GATT Discovery Char Request
38. GATT Discovery Service Request
39. GATT Discovery Char Descriptor Request
40. GATT Read Char Request
41. GATT Write Char Request
42. GATT Write Reliable Request
43. GATT Execute Write Char Request
44. GATT Notify Request
45. GATT Indicate Request
46. GATT Write Response
47. GATT Set Permission
48. GATT Set Data
ESC Key: Menu exit
>> 1
CMD --> GAP Reset
status(RBLE_OK)
>>
```

**Figure 5-4 Execution example of RBLE_GAP_Reset function**

In the following sections, basic usage of each layer is explained.
5.5. Generic Access Profile (GAP)

Commands and events for connecting device without security are shown in the following table as basic operations of the GAP. In addition, Figure 5-5 shows the log of the master device and Figure 5-6 shows the log of the slave device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Master (Command &amp; Event)</th>
<th>Slave (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td>GAP Reset</td>
<td>GAP Reset</td>
</tr>
<tr>
<td></td>
<td>RESET_RESULT</td>
<td>RESET_RESULT</td>
</tr>
<tr>
<td>Send Advertising</td>
<td>GAP Device_Search</td>
<td>GAP Broadcast_Enable</td>
</tr>
<tr>
<td></td>
<td>DEVICE_SEARCH_RESULT_IND</td>
<td>BROADCAST_ENABLE_COMP</td>
</tr>
<tr>
<td>Search device (optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Establish connection</td>
<td>GAP Create_Connection</td>
<td>CONNECTION_COMP</td>
</tr>
<tr>
<td></td>
<td>CONNECTION_COMP</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-5 Log of Master Device (when connecting device without security)
5.6. Security Manager (SM)

Commands and events for connecting device with security are shown in the following table as basic operations of the SM. In addition, Figure 5-7 and Figure 5-8 show the log of the master device and Figure 5-9 and Figure 5-10 show the log of the slave device when you do the following operations in the table. The device search operation is omitted in the each log.

Figure 5-6 Log of Slave Device (when connecting device without security)
<table>
<thead>
<tr>
<th>Operation</th>
<th>Master (Command &amp; Event)</th>
<th>Slave (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td><strong>GAP Reset</strong></td>
<td><strong>GAP Reset</strong></td>
</tr>
<tr>
<td></td>
<td><strong>RESET_RESULT</strong></td>
<td><strong>RESET_RESULT</strong></td>
</tr>
<tr>
<td>Set security</td>
<td><strong>GAP Set_Security_Request</strong></td>
<td><strong>GAP Set_Security_Request</strong></td>
</tr>
<tr>
<td></td>
<td><strong>SET_SECURITY_REQUEST_COMP</strong></td>
<td><strong>SET_SECURITY_REQUEST_COMP</strong></td>
</tr>
<tr>
<td></td>
<td><strong>GAP_Set_Bonding_Mode</strong></td>
<td><strong>GAP_Set_Bonding_Mode</strong></td>
</tr>
<tr>
<td></td>
<td><strong>SET_BONDING_MODE_COMP</strong></td>
<td><strong>SET_BONDING_MODE_COMP</strong></td>
</tr>
<tr>
<td>Send Advertising</td>
<td><strong>GAP Device_Search</strong></td>
<td><strong>BROADCAST_ENABLE_COMP</strong></td>
</tr>
<tr>
<td></td>
<td><strong>DEVICE_SEARCH_RESULT_IND</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>DEVICE_SEARCH_COMP</strong></td>
<td></td>
</tr>
<tr>
<td>Search device (optional)</td>
<td><strong>GAP Create_Connection</strong></td>
<td><strong>CONNECTION_COMP</strong></td>
</tr>
<tr>
<td></td>
<td><strong>CONNECTION_COMP</strong></td>
<td></td>
</tr>
<tr>
<td>Establish connection</td>
<td><strong>BD_ADDR_REQ_IND</strong></td>
<td><strong>BD_ADDR_REQ_IND</strong></td>
</tr>
<tr>
<td></td>
<td><strong>SM Chk_Bd_Addr_Req_Resp</strong></td>
<td><strong>SM Chk_Bd_Addr_Resp</strong></td>
</tr>
<tr>
<td>Confirm device</td>
<td><strong>GAP Start_Bonding</strong></td>
<td><strong>BONDING_REQ_IND</strong></td>
</tr>
<tr>
<td></td>
<td><strong>BONDING_REQ_IND</strong></td>
<td><strong>GAP Bonding_Response</strong></td>
</tr>
<tr>
<td>Start bonding</td>
<td><strong>TK_REQ_IND</strong></td>
<td><strong>TK_REQ_IND</strong></td>
</tr>
<tr>
<td></td>
<td><strong>SM Tk_Req_Resp</strong></td>
<td><strong>SM Tk_Req_Resp</strong></td>
</tr>
<tr>
<td>Bonding request and response</td>
<td><strong>LTK delivery</strong></td>
<td><strong>LTK_REQ_IND</strong></td>
</tr>
<tr>
<td></td>
<td><strong>KEY_IND</strong></td>
<td><strong>KEY_IND</strong></td>
</tr>
<tr>
<td>Bonding completion</td>
<td><strong>BONDING_COMP</strong></td>
<td><strong>BONDING_COMP</strong></td>
</tr>
</tbody>
</table>
Figure 5-7 Log of Master Device (when connecting device with security)
Figure 5-8 Log of Master device (when connecting device with security) (continued).

Figure 5-9 Log of Slave device (when connecting device with security)
5.7. Generic Attribute Profile (GATT)

Commands and events for obtaining the characteristic handle grouped in service of remote device are shown in the following table as basic operations of the GATT. In addition, Figure 5-11 show the log of the Master device and Figure 5-12 shows the log of the Slave device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Master (Command &amp; Event)</th>
<th>Slave (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecting to the remote device</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td>GATT Enable</td>
</tr>
<tr>
<td>Enable GATT</td>
<td>GATT Discovery_Char_Request</td>
<td>GATT Discovery_Char_Request</td>
</tr>
<tr>
<td></td>
<td>DISC_CHAR_BY_UUID_CMP</td>
<td>DISC_CHAR_BY_UUID_CMP</td>
</tr>
<tr>
<td></td>
<td>COMPLETE</td>
<td>COMPLETE</td>
</tr>
</tbody>
</table>
Figure 5-11 Log of Master (Read Characteristic using GATT)

Figure 5-12 Log of Slave (Read Characteristic using GATT)
5.8. Find Me Profile (FMP)

Commands and events for writing alert level are shown in the following table as basic operations of the FMP. In addition, Figure 5-13 show the log of the Locator device and Figure 5-14 shows the log of the Target device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operations</th>
<th>Locator (Command &amp; Event)</th>
<th>Target (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecting to the</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td>FMP Target_Enable</td>
</tr>
<tr>
<td>remote device</td>
<td></td>
<td>TARGET_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable target</td>
<td>FMP Locator_Enable</td>
<td>FMP Locator_Enable</td>
</tr>
<tr>
<td>Enable locator</td>
<td>LOCATOR_ENABLE_COMP</td>
<td>TARGET_ENABLE_COMP</td>
</tr>
<tr>
<td>Set alert</td>
<td>FMP Locator_Set_Alert</td>
<td>TARGET_ALERT_IND</td>
</tr>
</tbody>
</table>

[Note]

All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection.
About commands and events for profiles are described after connecting to the remote device.
To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.
Figure 5-13 Log of FMP Locator
Figure 5-14 Log of FMP Target
5.9. Proximity Profile (PXP)

Commands and events for reading and writing alert level are shown in the following table as basic operations of the PXP. In addition, Figure 5-15 and Figure 5-16 show the log of the Monitor device and Figure 5-17 shows the log of the Reporter device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Monitor (Command &amp; Event)</th>
<th>Reporter (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecting to</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td>PXP Reporter_Enable</td>
</tr>
<tr>
<td>the remote device</td>
<td></td>
<td>REPORTER_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable reporter</td>
<td>PXP Monitor_Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MONITOR_ENABLE_COMP</td>
<td></td>
</tr>
<tr>
<td>Enable monitor</td>
<td>PXP Monitor_Get_Alert_Level</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MONITOR_READ_CHAR_RESPONSE</td>
<td></td>
</tr>
<tr>
<td>Read alert level</td>
<td>PXP Monitor_Set_Alert_Level</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MONITOR_WRITE_CHAR_RESPONSE</td>
<td></td>
</tr>
</tbody>
</table>

[Note]

All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection. About commands and events for profiles are described after connecting to the remote device. To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.
Figure 5-15 Log of PXP Monitor

```
C:\WINDOWS\system32\cmd.exe

- BLE Sample Program Menu Version 1.00.000 --
1. SW & SW & UART Tool
2. Profile Test
3. Vendor Specific Test
4. PTS Test Case Select
ESC Key: Menu exit

-- BLE Sample Program Profile Test Menu --
1. Find Me Profile
2. Health Thermometer Profile
3. Proximity Profile
4. Blood Pressure Profile
5. Health over GATT Profile
6. Scan Parameters Profile
7. Sample Custom Profile
ESC Key: Menu exit

-- BLE Sample Program Proximity Profile Test Menu --
1. PXP Reporter_Enable
2. PXP Reporter_Disable
3. PXP Monitor_Enable
4. PXP Monitor_Disable
5. PXP Monitor_Set_Alert_Level
6. PXP Monitor_Set_Alert_Level
7. PXP Monitor_Set_Ic_Power
ESC Key: Menu exit

CMD: PXP Monitor_Enable
Status(ABLE_OK)

#BLE PXP EVENT [MONITOR_ENABLE_COMP] Status(ABLE_OK)
Connection Handle = 0

* Link Loss Service
  Start Handle = 0x000F
  End Handle = 0x0011
  Alert level char handle = 0x0010
  alert Level Value_handle = 0x0011
  Alert level properties = 0x0A
  Alert value = 0x20

* Immediate Alert Service
  Start Handle = 0x0015
  End Handle = 0x0017
  Alert level char handle = 0x0016
  alert Level Value_handle = 0x0017
  Alert level properties = 0x0A
  Alert value = 0x20

* Tx Power Service
  Start Handle = 0x0012
  End Handle = 0x0014
```


Figure 5-16 Log of PXP Monitor (continued)

Figure 5-17 Log of PXP Reporter
5.10. Health Thermometer Profile (HTP)

Commands and events for sending thermometer data are shown in the following table as basic operations of the HTP. In addition, Figure 5-18 and Figure 5-19 show the log of the Collector device and Figure 5-20 and Figure 5-21 shows the log of the Thermometer device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Collector (Command &amp; Event)</th>
<th>Thermometer (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecting to the remote device</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td></td>
</tr>
<tr>
<td>Enable Thermometer</td>
<td>HTP Collector_Enable</td>
<td>THERMOMETER_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable Collector</td>
<td>COLLECTOR_ENABLE_COMP</td>
<td></td>
</tr>
<tr>
<td>Enable Indication</td>
<td>HTP Collector_Write_Char</td>
<td>THERMOMETER_CFG_INDNTF_IND</td>
</tr>
<tr>
<td>Transmit and receive thermometer data</td>
<td>COLLECTOR_TEMP_IND</td>
<td>THERMOMETER_SEND_TEMP_COMP</td>
</tr>
</tbody>
</table>

[Note]

All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection.

About commands and events for profiles are described after connecting to the remote device.

To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.
Figure 5-18 Log of HTP Collector

```plaintext
>> 0
CMD -> HTP_Collector_Write_Char
Select Parameter No 0
char_code = 1, cfg_val = 2
Status(RBLE_OK)

>> BLE HTP EVENT (COLLECTION_WRITE_CHAR_RESPONSE) Status(RBLE_OK)
Connection Handle = 0

>> BLE HTP EVENT (COLLECTION_TEMP_IND)
Connection Handle = 0
Flags_stable_meas = 1
Flags = 0
Temperature : 1.0 (°C)
Time Stamp : 4003/03/120 15:4:18:222
Temperature Type : 2 - 2 Body (general)
Value : 0x00000000542266780ABCDE02

>> BLE GAP EVENT (DISCONNECT_COMP) Status(RBLE_OK)
reason = CON_TIMEOUT

>> BLE HTP EVENT (COLLECTION_DISABLE_COMP) Status(RBLE_OK)
Connection Handle = 0
```

Figure 5-19 Log of HTP Collector (continued)
Figure 5-20 Log of HTP Thermometer

![Image of Log of HTP Thermometer]

Figure 5-21 Log of HTP Thermometer (continued)

![Image of Log of HTP Thermometer (continued)]
5.11. Blood Pressure Profile (BLP)

Commands and events for sending measurement data are shown in the following table as basic operations of the BLP. In addition, Figure 5-22 and Figure 5-23 show the log of the Collector device and Figure 5-24 shows the log of the Sensor device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Collector (Command &amp; Event)</th>
<th>Sensor (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecting to the remote device</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td></td>
</tr>
<tr>
<td>Enable Sensor</td>
<td>BLP Sensor_Enable</td>
<td>SENSOR_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable Collector</td>
<td>BLP Collector_Enable</td>
<td>COLLECTOR_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable Indication</td>
<td>BLP Collector_Write_Char</td>
<td>SENSOR_CFG_INDNTF_IND</td>
</tr>
<tr>
<td>Transmit and receive measurement data</td>
<td>COLLECTOR_MEASUREMENTS_IND</td>
<td>BLP Sensor_Send_Measurements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SENSOR_SEND_MEASUREMENTS_COMP</td>
</tr>
</tbody>
</table>

[Note]

All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection.

About commands and events for profiles are described after connecting to the remote device.

To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.
Figure 5-22 Log of BLP Collector

Figure 5-23 Log of BLP Collector (continued)
Figure 5-24 Log of BLP Sensor
### 5.12. HID over GATT Profile (HOGP)

Commands and events for transmitting the input report data are shown in the following table as basic operations of the HOGP. In addition, Figure 5-25 and Figure 5-26 shows the log of the Report Host device and Figure 5-27 and Figure 5-29 shows the log of the HID Device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Report Host (Command &amp; Event)</th>
<th>HID Device (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecting to the remote device</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td>HGP_HDevice_Enable HDEVICE_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable HID device</td>
<td>HGP_RHost_Enable RHOST_ENABLE_COMP</td>
<td></td>
</tr>
<tr>
<td>Enable report host</td>
<td>HGP_RHost_Set_Report RHOST_WRITE_CHAR_RESPONSE</td>
<td>HDEVICE_REPORT_IND</td>
</tr>
<tr>
<td>Transmit and receive input report data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**[Note]**

All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection.

About commands and events for profiles are described after connecting to the remote device.

To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.
Figure 5-25 Log of Report Host

Figure 5-26 Log of Report Host (continued)
Figure 5-27 Log of HID Device

Figure 5-28 Log of HID Device (continued)
5.13. Scan Parameters Profile (ScPP)

Commands and events for transmitting the scan interval window data are shown in the following table as basic operations of the ScPP. In addition, Figure 5-29 shows the log of the Scan Client device and Figure 5-30 shows the log of the Scan Server device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Scan Client (Command &amp; Event)</th>
<th>Scan Server (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect to the remote device</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td>SPP_Server_Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SPPS_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable server</td>
<td>SPP_Client_Enable</td>
<td>SPPS_INTERVAL_WINDOW_CHANGE_EVT</td>
</tr>
<tr>
<td>Enable client</td>
<td>SPPC_ENABLE_COMP</td>
<td></td>
</tr>
<tr>
<td>Transmit and receive scan interval window data</td>
<td>SPP_Client_Write_Interval</td>
<td></td>
</tr>
</tbody>
</table>

[Note]

All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection.
About commands and events for profiles are described after connecting to the remote device.
To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.
Figure 5-29 Log of Scan Client
Figure 5-30 Log of Scan Server

C:\WINDOWS\system32\cmd.exe

--- BLE Sample Program Menu Version 1.00.000 --
1. SAP & SN & GATT Tool
2. Profile Test
3. Vendor Specific Test
4. PTS Test Case Select
ESC Key: Menu exit

--- BLE Sample Program Profile Test Menu ---
1. Find No Profile
2. Health Thermometer Profile
3. Proximity Profile
4. Blood Pressure Profile
5. HID over GATT Profile
6. Scan Parameters Profile
7. Sample Custom Profile
ESC Key: Menu exit

--- BLE Sample Program Scan Parameters Profile Test Menu ---
1. SPP Server Enable
2. SPP Server Disable
3. SPP Server Send Refresh
4. SPP Client Enable
5. SPP Client Disable
6. SPP Client Write Char
7. SPP Client Write Interval
ESC Key: Menu exit

```bash
>> 1
CMD >> SPP Server_Enable
Status[RBLE_OK]
```

```bash
#BLE SPP EVENT (SERVER_ENABLE_COMP) Status[RBLE_OK]
Connection Handle = 0
```

```bash
#BLE SPP EVENT (SERVER_INTERVAL_CHG_EVT)
Connection Handle = 0
  interval value = 0000
  window value  = 0000
```
5.14. Heart Rate Profile (HRP)

Commands and events for sending measurement data are shown in the following table as basic operations of the HRP.
In addition, Figure 5-31 and Figure 5-32 show the log of the Collector device and Figure 5-33 shows the log of the Sensor device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Heart Rate Collector (Command &amp; Event)</th>
<th>Heart Rate Sensor (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect to the remote device</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td>HRP Sensor_Enable</td>
</tr>
<tr>
<td>Enable Sensor</td>
<td></td>
<td>SENSOR_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable Collector</td>
<td>HRP Collector_Enable</td>
<td>SENSOR_CFG_NTF_IND</td>
</tr>
<tr>
<td>Enable Indication</td>
<td>COLLECTOR_ENABLE_COMP</td>
<td>HRP Sensor_Send_Measurements</td>
</tr>
<tr>
<td>Transmit and receive measurement data</td>
<td>COLLECTOR_MEASUREMENTS_NTF</td>
<td>SENSOR_SEND_MEASUREMENTS_COMP</td>
</tr>
</tbody>
</table>

[Note]
All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection.
About commands and events for profiles are described after connecting to the remote device.
To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.
Figure 5-31 Log of HRP Collector
Bluetooth® Low Energy Protocol Stack Sample Program

Figure 5-32 Log of HRP Collector (continued)

control_point_char_hdl = 0x003C
control_point_val_hdl = 0x003D
control_point_prop = 0x08

# Device Information Service
Start Handle = 0x0025
End Handle = 0x0035

sys_id_char_hdl = 0x0026
sys_id_val_hdl = 0x0027
sys_id_prop = 0x02

model_nb_char_hdl = 0x0028
model_nb_val_hdl = 0x0029
model_nb_prop = 0x02

serial_nb_char_hdl = 0x002A
serial_nb_val_hdl = 0x002B
serial_nb_prop = 0x02

fw_rev_char_hdl = 0x002C
fw_rev_val_hdl = 0x002D
fw_rev_prop = 0x02

hw_rev_char_hdl = 0x002E
hw_rev_val_hdl = 0x003F
hw_rev_prop = 0x02

sw_rev_char_hdl = 0x0030
sw_rev_val_hdl = 0x0031
sw_rev_prop = 0x02

manuf_name_char_hdl = 0x0032
manuf_name_val_hdl = 0x0033
manuf_name_prop = 0x02

ieee_certif_char_hdl = 0x0034
ieee_certif_val_hdl = 0x0035
ieee_certif_prop = 0x02

>>> 8 1
CMD -> HRP Collector_Write_Char
Start Ntf(1)
Status(RBLE_OK)

>>> BLE HRP EVENT (COLLECTOR_WRITE_CHAR_RESPONSE) Status(RBLE_OK)
Connection Handle = 0

>>> BLE HRP EVENT (COLLECTOR_MEASUREMENTS_NTF)
Measure: ff(255)
Energy: 0010(16)
RR Interval: 00010(16)
--- BLE Sample Program Menu Version 1.00.000 ---
1. GAP & SM & GATT Test
2. Profile Test
3. Vendor Specific Test
4. PTS Test Case Select
ESC Key: Menu exit

>> 2

--- BLE Sample Program Profile Test Menu ---
1. Find Me Profile
2. Health ThermoMeter Profile
3. Proximity Profile
4. Blood Pressure Profile
5. HID over GATT Profile
6. Scan Parameters Profile
7. Heart Rate Profile
8. Cycling Speed Profile
9. Cycling Power Profile
10. Sample Custom Profile
11. Alert Notification Profile
12. Location and Navigation Profile
ESC Key: Menu exit

>> 7

--- BLE Sample Program Heart Rate Profile Test Menu ---
1. HRP SensorEnable
2. HRP SensorDisable
3. HRP Sensor_Send_Measurements
4. HRP Collector_Enable
5. HRP CollectorDisable
6. HRP Collector_Read_Char
7. HRP Collector_Write_Control_Point
8. HRP Collector_Write_Char
ESC Key: Menu exit

>> 1

CMD -> HRP Sensor_Enable
Status(BLE_OK)

>>

BLE BLE EVENT (SENSOR_ENABLE_COMP) Status(BLE_OK)
Connection Handle = 0

>>

BLE BLE EVENT (SENSOR_CFG_LOLTF_IND)
Cfg Value = START_NTFD

>> 3 0

CMD -> HRP Sensor_Send_Measurements
Status(BLE_OK)

>>

BLE BLE EVENT (SENSOR_SEND_MEASUREMENTS_COMP) Status(BLE_OK)
Connection Handle = 0

>>

Figure 5-33 Log of HRP Sensor
5.15. Cycling Speed and Cadence Profile (CSCP)

Commands and events for sending CSC measurement data are shown in the following table as basic operations of the CSCP. In addition, Figure 5-34, Figure 5-35 and Figure 5-36 show the log of the Collector device and Figure 5-37 shows the log of the Sensor device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycling Speed and Cadence Collector (Command &amp; Event)</th>
<th>Cycling Speed and Cadence Sensor (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect to the remote device</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td>CSCP Sensor_Enable SENSOR_ENABLECOMP</td>
</tr>
<tr>
<td>Enable Sensor</td>
<td></td>
<td>CSCP Collector_Enable COLLECTOR_ENABLECOMP</td>
</tr>
<tr>
<td>Enable Collector</td>
<td></td>
<td>CSCP Collector_Write_Char COLLECTOR_WRITE_CHAR_RESPONS</td>
</tr>
<tr>
<td>Enable Indication</td>
<td></td>
<td>SENSOR_CFG_INDNTF_IND</td>
</tr>
<tr>
<td>Transmit and receive CSC measurement data</td>
<td></td>
<td>CSCP Sensor_Send_Measurements SENSOR_SEND_MEASUREMENTS_COMP</td>
</tr>
</tbody>
</table>

[Note] All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection.

About commands and events for profiles are described after connecting to the remote device.

To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.

Figure 5-34 Log of CSCP Collector
Figure 5-35 Log of CSCP Collector (continued -1)
serial_nb_char_hdl = 0x0014
serial_nb_val_hdl = 0x0015
serial_nb_prop = 0x02
/fw_rev_char_hdl = 0x0016
/fw_rev_val_hdl = 0x0017
/fw_rev_prop = 0x02

/hw_rev_char_hdl = 0x0018
/hw_rev_val_hdl = 0x0019
/hw_rev_prop = 0x02

/sw_rev_char_hdl = 0x001A
/sw_rev_val_hdl = 0x001B
/sw_rev_prop = 0x02

/manuf_name_char_hdl = 0x001C
/manuf_name_val_hdl = 0x001D
/manuf_name_prop = 0x02

/ieee_certif_char_hdl = 0x001E
/ieee_certif_val_hdl = 0x001F
/ieee_certif_prop = 0x02

>> 9 1 1
CMD -> CSCP Collector_Write_Char
Start Ntf
Status(RBLE_OK)
>>
BLE CSCP EVENT (COLLECTOR_WRITECHAR_RESPONSE) Status(RBLE_OK)
Connection Handle = 0
>>
BLE CSCP EVENT (COLLECTOR_MEASUREMENTS_NTF)
Flag :03
Wheel Rev :0x00ff00ff(16711935)
Wheel Ev Time:0x0010(16)
Speed:-- (first event)
Crank Rev :0x0200(512)
Crank Ev Time:0x0030(48)
Cadence:-- (first event)

Figure 5-36 Log of CSCP Collector (continued -2)
Figure 5-37 Log of CSCP Sensor
5.16. Cycling Power Profile (CPP)

Commands and events for sending Cycling Power measurement data are shown in the following table as basic operations of the CPP. In addition, Figure 5-38, Figure 5-39 and Figure 5-40 show the log of the Collector device and Figure 5-41 shows the log of the Sensor device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycling Power Collector (Command &amp; Event)</th>
<th>Cycling Power Sensor (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect to the remote device</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td>CPP Sensor_Enable</td>
</tr>
<tr>
<td>Enable Sensor</td>
<td>CPP Collector_Enable</td>
<td>SENSOR_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable Collector</td>
<td>COLLECTOR_ENABLE_COMP</td>
<td></td>
</tr>
<tr>
<td>Enable Indication</td>
<td>CPP Collector_Write_Char</td>
<td>SENSOR_CFG_INDNTFBRD_IND</td>
</tr>
<tr>
<td>Transmit and receive</td>
<td>COLLECTOR_WRITE_CHAR_RESPONSE</td>
<td>CPP Sensor_Send_Measurements</td>
</tr>
<tr>
<td>Cycling Power measurement data</td>
<td>COLLECTOR_MEASUREMENTS_NTF</td>
<td>SENSOR_SEND_MEASUREMENTS_COMP</td>
</tr>
</tbody>
</table>

[Note]

All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection.

About commands and events for profiles are described after connecting to the remote device.

To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.

Figure 5-38 Log of CPP Collector
Figure 5-39 Log of CPP Collector (continued -1)
Bluetooth® Low Energy Protocol Stack
Sample Program

Figure 5-40 Log of CPP Collector (continued -2)
Figure 5-41 Log of CPP Sensor

```plaintext
-- BLE Sample Program Menu Version 1.00.000 --
1. GAP & SM & GATT Test
2. Profile Test
3. Vendor Specific Test
4. PTS Test Case Select
ESC Key: Menu exit

>> 2

-- BLE Sample Program Profile Test Menu --
1. Find Me Profile
2. Health Thermometer Profile
3. Proximity Profile
4. Blood Pressure Profile
5. HID over GATT Profile
6. Scan Parameters Profile
7. Heart Rate Profile
8. Cycling Speed Profile
10. Cycling Power Profile
11. Sample Custom Profile
13. Alert Notification Profile
14. Location and Navigation Profile
ESC Key: Menu exit

>> 10

-- BLE Sample Program Cycling Power Profile Test Menu --
1. CPP Sensor_Enable
2. CPP Sensor_Disable
3. CPP Sensor_Send.Measurements
4. CPP Sensor_Broadcast.Measurements
5. CPP Sensor_Send.Vector
6. CPP Sensor_Send_CP_Control_Point
7. CPP Sensor_Send_Battery_Level
8. CPP Sensor_Send_Write_Response
9. CPP Collector_Enable
10. CPP Collector_Disable
11. CPP Collector_Read_Char
12. CPP Collector_Write_CP_Control_Point
13. CPP Collector_Write_Char
ESC Key: Menu exit

>> 1

CMD -> CPP Sensor_Enable
Status(RBLE_OK)

BLE CPP EVENT (SENSOR_ENABLE_COMP) Status(RBLE_OK)
Connection Handle = 0

BLE CPP EVENT (SENSOR_CFG_IND_NOTIFY_IND)
Char Code = 1
Char Value = START
3 0 0x3

CMD -> CPP Sensor_Send.Measurements
Status(RBLE_OK)

BLE CPP EVENT (SENSOR_SEND.Measurements_COMP) Status(RBLE_OK)
Connection Handle = 0
```
5.17. Alert Notification Profile (ANP)

Commands and events for sending New Alert data are shown in the following table as basic operations of the ANP. In addition, Figure 5-42 and Figure 5-43 show the log of the Client device and Figure 5-44 shows the log of the Server device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Alert Notification Client (Command &amp; Event)</th>
<th>Alert Notification Server (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect to the remote device</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td>ANP Server_Enable SERVER_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable Sensor</td>
<td>ANP Client_EnableCLIENT_ENABLE_COMP</td>
<td>SERVER_CFG_NTF_IND ANP Sensor_Send_New_Alert SERVER_SEND_NEW_ALERT_COMP</td>
</tr>
<tr>
<td>Enable Collector</td>
<td>ANP Client_Write_CharCLIENT_WRITE_CHAR_RESPONSE</td>
<td></td>
</tr>
<tr>
<td>Enable Indication</td>
<td>CLIENT_NEW_ALERT_NTF</td>
<td></td>
</tr>
</tbody>
</table>

[Note]

All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection.

About commands and events for profiles are described after connecting to the remote device.

To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.
Figure 5-43 Log of ANP Client (continued)
-- BLE Sample Program Menu Version 1.00.000 --
1. GAP & SM & GATT Test
2. Profile Test
3. Vendor Specific Test
4. PTC Test Case Select
ESC Key: Menu exit

>> 2

-- BLE Sample Program Profile Test Menu --
1. Find Me Profile
2. Health Thermometer Profile
3. Proximity Profile
4. Blood Pressure Profile
5. HID over GATT Profile
6. Scan Parameters Profile
7. Heart Rate Profile
8. Cycling Speed Profile
9. Cycling Power Profile
10. Sample Custom Profile
11. Alert Notification Profile
12. Location and Navigation Profile
ESC Key: Menu exit

>> 13

-- BLE Sample Program ANP Profile Test Menu --
1. ANP Server_Enable
2. ANP Server_Disable
3. ANP Server_Send_New_Aler
4. ANP Server_Send_Unread_Aler
5. ANP Client_Enable
6. ANP Client_Disable
7. ANP Client_Read_Char
8. ANP Client_Write_Aler_Notification_CP
9. ANP Client_Write_Char
ESC Key: Menu exit

>> 1

CMD -> ANP Server_Enable
Status(BLE_OK)

BLE ANP EVENT (SERVER_ENABLE_COMP) Status(BLE_OK)
Connection Handle = 0

BLE ANP EVENT (SERVER_CFG_NTIF_IND)
Connection Handle = 0
char:0 cs:1

>> 3 2 3

CMD -> ANP Server_Send_New_Aler
Status(BLE_OK)

BLE ANP EVENT (SERVER_SEND_NEW_ALERT_COMP) Status(BLE_OK)
Connection Handle = 0

Figure 5-44 Log of ANP Server
5.18. Location and Navigation Profile (LNP)

Commands and events for sending Location Speed data are shown in the following table as basic operations of the LNP. In addition, Figure 5-45, Figure 5-46 and Figure 5-47 show the log of the Collector device and Figure 5-48 shows the log of the Sensor device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Location and Navigation Collector (Command &amp; Event)</th>
<th>Location and Navigation Sensor (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect to the remote device</td>
<td>Refer to 5.5 Generic Access Profile (GAP) and 5.6 Security Manager (SM)</td>
<td>LNP Sensor_Enable SENSOR_ENABLE_COMP</td>
</tr>
<tr>
<td>Enable Sensor</td>
<td>LNP Collector_Enable COLLECTOR_ENABLE_COMP</td>
<td>SENSOR_CFG_INDNTF_IND SENSOR_SEND_LOCATION_SPEEDCOMPARE</td>
</tr>
<tr>
<td>Enable Collector</td>
<td>LNP Collector_Write_Char COLLECTOR_WRITE_CHARRESPONSE</td>
<td>LNP Sensor_Send_Location_Speed SENSOR_SEND_LOCATION_SPEEDCOMPARE</td>
</tr>
<tr>
<td>Enable Indication</td>
<td>SENSOR_CFG_INDNTF_IND SENSOR_SEND_LOCATION_SPEEDCOMPARE</td>
<td>LNP Sensor_Send_Location_Speed SENSOR_SEND_LOCATION_SPEEDCOMPARE</td>
</tr>
<tr>
<td>Transmit and receive Location Speed data</td>
<td>COLLECTOR_LOCATION_SPEED_NTF</td>
<td>SENSOR_SEND_LOCATION_SPEEDCOMPARE</td>
</tr>
</tbody>
</table>

[Note]

All profiles are connected to the remote device using GAP and SM commands, and use the handle that has been notified at the time of connection.

About commands and events for profiles are described after connecting to the remote device.

To connect to the remote device, refer to 5.5 Generic Access Profile and 5.6 Security Manager.

---

![Figure 5-45 Log of LNP Collector](image-url)
Figure 5-46 Log of LNP Collector (continued -1)
Figure 5-47 Log of LNP Collector (continued -2)
Figure 5-48 Log of LNP Sensor
5.19. Vendor Specific (VS)

Commands and events for using the Direct Test Mode are shown in the following table as basic operations of the VS. In addition, Figure 5-49 shows the log of the transmitter device and Figure 5-50 shows the log of the receiver device when you do the following operations in the table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Transmitter (Command &amp; Event)</th>
<th>Receiver (Command &amp; Event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable VS</td>
<td>VS Enable</td>
<td>VS Enable</td>
</tr>
<tr>
<td>Test start</td>
<td>VS Test.Tx_Start</td>
<td>VS Test.Tx_Start</td>
</tr>
<tr>
<td></td>
<td>TEST_TX_START_COMP</td>
<td>TEST_RX.START_COMP</td>
</tr>
<tr>
<td>Test end</td>
<td>VS Test_End</td>
<td>VS Test_End</td>
</tr>
<tr>
<td></td>
<td>TEST_END_COMP</td>
<td>TEST_END_COMP</td>
</tr>
</tbody>
</table>
Figure 5-49 Log of Direct Test Mode (Transmitter)
Figure 5-50 Log of Direct Test Mode (Receiver)
6. Usage of Simple Sample Program

This simple sample program shows how to use the BLE software. Contrary to the Sample Program, this simple sample program includes small functions, thus you can understand the behavior and the implementation easily.

This simple sample program is identical with “Embedded Configuration Sample Application (r01an3319)” Peripheral role sample application. Refer “Embedded Configuration Sample Application (r01an3319)” application note for the detail.

6.1. Configuration

This simple sample program works with embedded configuration only. Not works with modem configuration.

6.2. HEX File Preparation

There are two methods to prepare HEX file.

First one is to use the pre-built HEX file. The pre-built HEX files are located in /Renesas/BLE_Software_Ver_X_XX/RL78_G1D/ROM_File. You can find the pre-built HEX files compiled with each supported compiler (CC-RL, IAR, CA78K0R).

Second one is to build the HEX file from source codes. The project files are located in /Renesas/BLE_Software_Ver_X_XX/RL78_G1D/renesas/tools/simple_sample. You can find the project files for each supported development environment (e2 studio, CS+, IAR Embedded Workbench).

6.3. Behavior

After writing the HEX file prepared in Section 6.2 onto RL78/G1D Test Board, reset the board by pressing the reset button. After the reset, make sure LED1 and LED2 on the board start blinking.

The simple sample program starts advertising automatically. You can perform following functions after establishing connection between the board and a peer device.

- A peer device receives SW4 state (PUSH/RELEASE) from the board
- A peer device controls LED4 state (ON/OFF) on the board

You need a peer device for the simple sample program behavior checking. From next section describes the procedure to use Android Device or iOS Device as a peer device.
6.4. Check with Android Device

This section describes procedures to check the simple sample program behavior with Android Device. We use “BLE Scanner Version 3.6”. Check following URL for details of BLE Scanner.


1) Launch BLE Scanner on Android Device and scan nearby device. Select the device the name is “REL-BLE” from the discovered device list (Figure a). After selecting the device, a connection between the board and Android Device is established.

2) Select CUSTOM SERVICE (UUID: 5BC1B9F7-A1F1-40AF-9043-C43692C18D7A) from the service list (Figure b).

3) Procedures for “Android Device receives SW4 state (PUSH/RELEASE) from the board”
   You use CUSTOM CHARACTERISTIC (UUID: 5BC18D80-A1F1-40AF-9043-C43692C18D7A) for the LED4 control. When you tap on (N) button (Figure c, upside arrow), the board starts sending SW4 state to Android Device. Depends on the board SW4 state, you will see “HEX” value is changed (Figure c, downside arrow). When SW4 state is RELEASE you will see 0x00, when SW4 state is PUSH you will see 0x01. To stop sending SW4 state, re-tap the (N) button.

4) Procedures for “Android Device controls LED4 state (ON/OFF) on the board”.
   You use CUSTOM CHARACTERISTIC (UUID: 5BC143EE-A1F1-40AF-9043-C43692C18D7A) for the LED4 control. When you tap (W) button (Figure d), a dialog is opened. Then select “Byte Array”, input “01” and tap “OK” (Figure e), then you will see LED4 is ON. To turn OFF LED4 writes 0x01.
6.5. Check with iOS Device

This section describes the process to check the behavior of the simple sample program with iOS Device. We use “LightBlue Version 2.4.0”. Regarding LightBlue, see following URL.


1) Launch BLE Scanner on Android Device and scan nearby device. Select the device the name is “REL-BLE” from the discovered device list (Figure a). After selecting the device, a connection between the board and Android Device is established.

2) Procedures for “iOS Device receives SW4 state (PUSH/RELEASE) from the board”
Select Characteristic (UUID:5BC18D80-A1F1-40AF-9043-C43692C18D7A) (Figure b, upside arrow). Tap on “Listen for notifications” button (Figure c), and then the board start sending SW4 state to iOS Device. Depends on the board SW4 state, you will see “NOTIFIED VALUES” is changed (Figure d, downside arrow). When SW4 state is RELEASE you will see 0x00, when SW4 state is PUSH you will see 0x01. To stop the SW4 state sending, tap on “Stop Listening” (Figure d, upside arrow).

3) Procedures for “iOS Device controls LED4 state (ON/OFF) on the board”.
Select Characteristic (UUID:5BC1B9F7-A1F1-40AF-9043-C43692C18D7A) (Figure b, downside arrow). When you tap on “write new value” (Figure e), then new dialog is opened. Input “01” and tap on “Done” on the dialog (Figure f), then you will see LED4 on the board is ON. To turn off the LED4, write 0x00.
7. Appendix

7.1. Transmit and Receive Operations in the Sample Program for the Computer

The application running on the APP MCU is provided the BLE services from the BLE MCU via rBLE_Host. APP MCU and BLE MCU are physically connected by the UART or CSI or IIC and communicate each other using RSCIP (Renesas Serial Communication Protocol) under the control of rBLE_Host.

Figure 7-1 shows the internal structure of the sample program for computer. The sample program for computer works by calling the command I/O function from the main processing and rBLE software blocks as shown Figure 7-1.

![Figure 7-1 Internal structure of sample program](image_url)

The process of transmitting and receiving in rBLE software block, are handled by calling the rBLE_Run function from the main processing block.

The rBLE_Run function checks the transmit buffer to the BLE MCU and calls the transmit function in RSCIP driver if there is the transmit data. It also analyzes received data in the receive buffer from BLE-MCU, if any, and calls registered application function based on the event information.

Also, if there is an event notification, it calls RSCIP function corresponding to the event. Figure 7-2 shows the sequence of internal processing.
Figure 7-2 Internal processing of the sample program (main processing)

Figure 7-3 shows the sequence of events at the time of issuance of the transmit events from RSCIP. The RSCIP processes transmit requests both from the retransmit processing block and from the application in one place, so it issues transmit event request to the rBLE when transmit events from both side. The rBLE calls RSCIP transmit function as shown in Figure 7-2 if transmit request is generated from both sides.

Figure 7-4 shows the sequence of events at the time of issuance of the receive events from RSCIP. Considering that the data receive notification from serial communication driver is called from an interrupt, RSCIP issues the receive event request to the rBLE when a packet has been received. The rBLE calls RSCIP packet receive function as shown in Figure 7-2 if receive event request is generated.
Figure 7-4 Internal processing of sample program (issue of receive event)
7.2. Requirements and Flow Chart of Serial Communication Driver on APP MCU

The requirements on the APP MCU in the application development of modem configuration are summarized below.

H/W resource

- 1 channel of UART or CSI (Clocked Serial Interface) or IIC (Inter-Integrated Circuit) for serial communication is required for the communication with BLE MCU.

Timer

- The timeout function is required in the RSCIP driver. (Refer to the RSCIP implementation in rBLE_Host).

Serial communication driver
The serial communication driver using UART or CSI or IIC should be prepared by the user. In addition, the following functions are required in the serial communication driver as an interface between the RSCIP driver and serial communication driver.

<table>
<thead>
<tr>
<th>Function</th>
<th>Serial communication driver initialization function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This function initializes the serial communication driver. Initialize the serial communication driver in settings that are described in the Bluetooth Low Energy Protocol Stack User’s Manual.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>TRUE: Initialization is completed successfully, FALSE: Initialization is completed with some errors</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Serial communication driver transmit function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This function is a non-blocking function that transmits specified size of data via the serial communication line. The transmit data size is specified by the argument ‘size’ and transmit data is stored in the area pointed by the argument ‘bufptr’. When the transmission of data is completed, call following transmit completion notification function (RSCIP_Uart_Tx_Done) to RSCIP driver. void RSCIP_Uart_Tx_Done ( void ); If the method other than the two-wire UART connection is used, follow the transmit handshake procedure that is described in the Bluetooth Low Energy Protocol Stack User’s Manual.</td>
</tr>
<tr>
<td>Arguments</td>
<td>uint8_t *bufptr: Pointer to the transmit data buffer, uint16_t size: Data size to be transmitted</td>
</tr>
<tr>
<td>Return Value</td>
<td>TRUE: Transmission is completed successfully, FALSE: Transmission is completed with some errors</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Serial communication driver receive function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This function is a non-blocking function that receives specified size of data via the serial communication line. The receive data size is specified by the argument ‘size’. Store received data into the area pointed by the argument ‘bufptr’. When the reception of data is completed, call following receive completion notification function (RSCIP_Uart_Rx_Done) to RSCIP driver. void RSCIP_Uart_Rx_Done ( void ); If the method other than the two-wire UART connection is used, follow the receive handshake procedure that is described in the Bluetooth Low Energy Protocol Stack User’s Manual. In addition, after calling the RSCIP receive completion notification function, call the following RSCIP get receive status function and check its return value in order to determine whether or not to receive data continuously. BOOLEN RSCIP_Uart_Rx_Idle ( void ); FALSE indicates that the packet reception is not completed. TRUE indicates that the packet reception is completed, and the driver is waiting for the beginning of the next packet.</td>
</tr>
<tr>
<td>Arguments</td>
<td>uint8_t *bufptr: pointer to the receive data buffer</td>
</tr>
</tbody>
</table>
uint16_t size

Return Value
TRUE
data size to be received
Reception is completed successfully
FALSE
Reception is completed with some errors

Supplement
This function may be called from an interrupt.
In the sample program, receive processing is performed by rBLE_Run function, which is called from main loop, except the minimum required processing that should be done in this function.

Function void serial_exit ( void )
Overview Serial communication driver exit function
Description This function does the exit procedure of the serial communication driver.
Do the exit procedure of the serial communication driver.
Arguments none
Return Value none

In the Modem configuration, the following connection methods are available as the serial communication line. Refer to the Bluetooth Low Energy Protocol Stack User’s Manual for more details of the connection method.
Implement the driver that fits with your system resources, refer to the flow chart of the serial communication procedure examples,

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Connection method</th>
<th>Example of transmit procedure</th>
<th>Example of receive procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART</td>
<td>2-wire</td>
<td>Refer to 7.2.1 Transmit Procedure Example using the UART 2-wire Connection Method, below.</td>
<td>Refer to 2) Receive Procedure Example using the UART Two-wire Connection Method, below.</td>
</tr>
<tr>
<td></td>
<td>3-wire</td>
<td>Refer to 7.2.3 Transmit Procedure Example using the UART 3-wire Connection Method, below.</td>
<td>Refer to 7.2.5 Receive Procedure Example using the UART 3-wire and 2-wire with Branch Connection Methods, below.</td>
</tr>
<tr>
<td></td>
<td>2-wire with branch</td>
<td>Refer to 7.2.4 Transmit Procedure Example using the UART 2-wire with Branch Connection Method, below.</td>
<td></td>
</tr>
<tr>
<td>CSI</td>
<td>4-wire</td>
<td>Refer to 7.2.6 Transmit Procedure Example using the CSI 4-wire Connection Method below.</td>
<td>Refer to 7.2.8 Receive Procedure Example using the CSI 4-wire and 5-wire Connection Method, below.</td>
</tr>
<tr>
<td></td>
<td>5-wire</td>
<td>Refer to 7.2.7 Transmit Procedure Example using the CSI 5-wire Connection Method, below.</td>
<td></td>
</tr>
<tr>
<td>IIC</td>
<td>3-wire</td>
<td>Refer to 7.2.6 Transmit Procedure Example using the IIC 3-wire Connection Method below.</td>
<td>Refer to 7.2.8 Receive Procedure Example using the IIC 3-wire Connection Method, below.</td>
</tr>
</tbody>
</table>
7.2.1. Transmit Procedure Example using the UART 2-wire Connection Method

The following flowchart shows an example of transmit procedure using the UART 2-wire connection method. As a prerequisite, UART hardware in this example starts transmission by writing the transmit data address, transmit data size and start command into UART registers, and generates an interrupt after the transmission of specified data is completed.

7.2.2. Receive Procedure Example using the UART Two-wire Connection Method

The following flowchart shows an example of receive procedure using the UART 2-wire connection, 3-wire connection or 2-wire with branch connection methods. As a prerequisite, UART hardware in this example starts reception by writing the receive buffer address, receive data size and start command into UART registers, and generates an interrupt after the reception of specified size of data is completed.

* RSCIP_Uart_Rx_Done function calls the serial_read function to start the next reception operation.
7.2.3. Transmit Procedure Example using the UART 3-wire Connection Method

The following flowchart shows an example of transmit procedure using the UART 3-wire connection method. As a prerequisite, UART hardware in this example starts transmission by writing the transmit data address, transmit data size and start command into UART registers, and generates an interrupt after the transmission of specified data is completed. Also, the receive procedure example described in 7.2.5 below is used.

In addition, for reliable communication, it is necessary to add the timeout process, in which carry out monitoring during handshake procedure and re-execute the handshake procedure if a timeout occurs.

* Receive End Interrupt Routine is used in receive procedure example.
7.2.4. Transmit Procedure Example using the UART 2-wire with Branch Connection Method

The following flow chart shows an example of transmit procedure using the UART 2-wire with branch connection method.

As a prerequisite, UART hardware in this example starts transmission by writing the transmit data address, transmit data size and start command into UART registers, and generates an interrupt after the transmission of specified data is completed. Also, the receive procedure example described in 7.2.5 below is used.

In addition, for reliable communication, it is necessary to add the timeout process, in which carry out monitoring during handshake procedure and re-execute the handshake procedure if a timeout occurs.

* Receive End Interrupt Routine is used in receive procedure example.
7.2.5. Receive Procedure Example using the UART 3-wire and 2-wire with Branch Connection Methods

The following flowchart shows an example of receive procedure using the UART 3-wire connection and 2-wire with branch connection methods.

As a prerequisite, UART hardware in this example starts reception by writing the receive buffer address, receive data size and start command into UART registers, and generates an interrupt after the reception of specified size of data is completed.

* RSCIP_Uart_RxDone function calls the serial_read function to start the next reception operation.
7.2.6. Transmit Procedure Example using the CSI 4-wire Connection Method

The following flowchart shows an example of transmit procedure using the CSI 4-wire connection method.

As a prerequisite, CSI hardware in this example starts transmission by writing the transmit data address, transmit data size and start command into CSI registers, and generates an interrupt after the transmission of specified data is completed. It is assumed that an input port of APP MCU are connected to the SDIR signal and also an interrupt is generated by dual edge detection (both the falling edge and rising edge) of the SDIR signal.

In addition, for reliable communication, it is necessary to add the timeout process, in which carry out monitoring during handshake procedure and re-execute the handshake procedure if a timeout occurs.

* The edge detection interrupt service routine is used more than once in the transmission operation.
* The edge detection interrupt service routine is also used in the receive operation.
### 7.2.7. Transmit Procedure Example using the CSI 5-wire Connection Method

The following flowchart shows an example of transmit procedure using the CSI 4-wire connection method. As a prerequisite, CSI hardware in this example starts transmission by writing the transmit data address, transmit data size and start command into CSI registers, and generates an interrupt after the transmission of specified data is completed. It is assumed that an input port of APP MCU are connected to the SDIR signal and also an interrupt is generated by dual edge detection (both the falling edge and rising edge) of the SDIR signal.

In addition, for reliable communication, it is necessary to add the timeout process, in which carry out monitoring during handshake procedure and re-execute the handshake procedure if a timeout occurs.

*The edge detection interrupt service routine is used more than once in the transmission operation.
*The edge detection interrupt service routine is also used in the receive operation.*
7.2.8. Receive Procedure Example using the CSI 4-wire and 5-wire Connection Method

The following flowchart shows an example of receive procedure using the CSI 4-wire and 5-wire connection method.

As a prerequisite, CSI hardware in this example start reception by writing the receive buffer address, receive data size and start command into CSI registers, and generates an interrupt after the reception of specified data is completed. It is assumed that an input port of APP MCU are connected to the SDIR signal and also an interrupt is generated by dual edge detection (both the falling edge and rising edge) of the SDIR signal.

(a) 3.7.2.8. Receive Procedure Example using the CSI 4-wire and 5-wire Connection Method

- **Receive starts.**
  - **serial_read** [Software]
    - Save the receive buffer address and receive data size given by arguments to global variables.
    - **Port** [Hardware]
      - A falling edge of the SDIR signal (Pulse), that notifies transmit request, is detected.
      - The falling edge detection is notified by an interrupt.
  - Is communication state variable IDLE state ?
    - No
      - **Edge detection interrupt service routine** [Software]
        - Set communication state variable to RX state.
        - Write the start address of transmit data buffer containing ACK byte (0x88) and size (1 byte) and write start command into CSI registers.
      - **CSI** [Hardware]
        - Transmit an ACK byte (0x88).
        - A raising edge of the SDIR signal (Low), that notifies ready to transmit, is detected.
        - The falling edge detection is notified by an interrupt.
    - Yes
      - **Edge detection interrupt service routine** [Software]
        - Write the receive buffer address and receive data size saved in (a) and write start command into the CSI registers to start transmission.
      - **CSI** [Hardware]
        - The CSI receives specified size of data continuously.
        - The CSI notifies the completion by an interrupt.
      - **Receive End Interrupt Routine** [Software]
        - Call RSCIP_Uart_Rx.Done.
          - Does RSCIP_Uart_Rx_Idle return true ?
            - No
              - **Port** [Hardware]
                - A raising edge of the SDIR signal (High), that notifies transmit completion, is detected.
                - The raising edge detection is notified by an interrupt.
            - Yes
              - **Edge detection interrupt service routine** [Software]
                - Set communication state variable to IDLE state.
                - Transmission ends.
7.2.9. Transmit Procedure Example using the IIC 3-wire Connection Method

The following flowchart shows an example of receive procedure using the IIC 3-wire connection method.

As a prerequisite, IIC hardware to generate an interrupt after reception of 1 byte data.

In addition, for reliable communication, it is necessary to add the timeout process, in which carry out monitoring during handshake procedure and re-execute the handshake procedure if a timeout occurs.

7.2.10. Receive Procedure Example using the IIC 3-wire Connection Method

The following flowchart shows an example of transmit procedure using the IIC 3-wire connection method.

As a prerequisite, IIC hardware to generate an interrupt after transmission of 1 byte data. It is assumed that an input port of APP MCU are connected to the REQ signal and also an interrupt is generated by falling edge detection of the REQ signal.
Bluetooth® Low Energy Protocol Stack Sample Program

serial_read
[Software]
- Save the receive buffer address and receive data size given by arguments to global variables.

Port
[Hardware]
- A falling edge of the REQ signal is detected.
- The falling edge detection is notified by an interrupt.

Edge detection interrupt service routine
[Software]
- Is communication state variable IDLE state?
  - Yes: Set communication state variable to RX_ACK wait state.
  - No: Start transmission of the slave address with Read-bit data byte.

IIC
[Hardware]
- The IIC transmits the one byte in the transmit buffer.
- The notification by IIC interrupt after ACK clock output.

IIC interrupt service routine
[Software]
- Detected ACK?
  - Yes: Set communication state variable to RX going state.
  - No: Set to IIC 8 clocks interrupt mode.

IIC
[Hardware]
- The IIC read clock to generate, and notified by the interrupt after the data reception.

IIC interrupt service routine
[Software]
- REQ signal equal High?
  - Yes: Set to automatic ACK disabled (receiving end) and IIC 9 clocks interrupt mode.
  - No: Reception buffer full?
    - Yes: Call RSCIP_Uart_Rx_Done
    - No: Reception last byte?
      - Yes: Create the Stop Condition.
      - No: Waiting to transmission?

serial_read
[Software]
- Save the receive buffer address and receive data size given by arguments to global variables.

Reception starts.

Yes
No

No

Yes
7.3. Porting of the Sample Program

When porting the sample program to APP MCU, there are modules that should be newly developed by the user and modules that can be reused directly.

Table 7-1 shows the classification of them.

<table>
<thead>
<tr>
<th>Folder Name</th>
<th>Classification</th>
<th>Porting Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLE_Sample\src\Platform\G1D_cs_ iar\</td>
<td>new development</td>
<td>With reference to the sample program, this module should be newly developed by the user, to meet the resources in the APP MCU.</td>
</tr>
<tr>
<td>BLE_Sample\src\rBLE\src\host</td>
<td>reuse</td>
<td>This module can be reused directly.</td>
</tr>
<tr>
<td>BLE_Sample\src\rBLE\src\include</td>
<td>reuse</td>
<td>This module can be reused directly.</td>
</tr>
<tr>
<td>BLE_Sample\src\rBLE\src\rscip</td>
<td>reuse</td>
<td>This module can be reused directly.</td>
</tr>
<tr>
<td>BLE_Sample\src\rBLE\src\sample_app</td>
<td>new development</td>
<td>With reference to API usage in the sample program, this module should be newly developed by the user.</td>
</tr>
</tbody>
</table>

It is to be noted that the reference value of the size of the reusable sample program are shown in Table 7-2. These values are the result of compiling for the RL78/G1D.

Build Environment: CS+ for CC V4.00.00 / RL78 compiler CC-RL V1.03.00

<table>
<thead>
<tr>
<th>Components</th>
<th>ROM size</th>
<th>RAM size</th>
</tr>
</thead>
<tbody>
<tr>
<td>rBLE (BLE_Sample\src\rBLE\src\host)</td>
<td>52,519 bytes</td>
<td>2,898 bytes</td>
</tr>
<tr>
<td>RSCIP (BLE_Sample\src\rBLE\src\rscip)</td>
<td>4,279 bytes</td>
<td>1,268 bytes</td>
</tr>
</tbody>
</table>

When you implement the following measures, it is possible to reduce the RAM size about 2KB.

1) BLE_Sample\src\rBLE\src\host\rble_host.c
   Before changing: #define MAX_BUFF_NUM  8
   (Follows)
   After changing: #define MAX_BUFF_NUM  2
   [Note] You cannot call the command more than MAX_BUFF_NUM continuously.

2) BLE_Sample\src\rBLE\src\host\rble_if_api_c.b.c
   Before changing: static uint8_t rBLE_Over_Packet_Temp[ 0x256 ];
   (Follows)
   After changing: static uint8_t rBLE_Over_Packet_Temp[ 1 ];
   [Note] You cannot handle data more than 128-byte in RBLE_VS_Flash_Access API. If you call the API, illegal memory access occurs.
7.4. How to use the Direct Test Mode

Direct Test Mode is performed by the Vendor Specific (VS) command. Figure 7-5 shows the Vendor Specific (VS) command menu. The menu items from 2 to 5 are related to the Direct Test Mode.

After this section, the commands related to the Direct Test Mode are explained.

[Note]

About the details of Direct Test Mode, refer to the ‘Chapter 8. Vendor Specific’ in the ‘API Reference Manual: Basics’.
7.4.1. Direct Test Mode (Receiver)

Using the VS menu number 2 ‘VS Test_Rx_Start’, you can start the Direct Test Mode (Receiver).

![Figure 7-6 Log of Direct Test Mode (Receiver) Start](image1)

Using the VS menu number 2 ‘VS Test_Rx_Start’, you can set the receive frequency (channel number). If no argument is given, it displays the usage of this command. Figure 7-6 shows the log of execution, when the receive frequency is channel 39 (2,480MHz).

If you want to terminate the execution of Direct Test Mode (Receiver), use the VS menu number 4 ‘VS Test_End’. Figure 7-7 shows the log after execution of Direct Test Mode (Receiver). The number of received packets is displayed after this test. The data have been received zero times in Figure 7-7 and 3,235 times in Figure 7-8.

![Figure 7-7 Log of Direct Test Mode (Receiver) End](image2)
7.4.2. Direct Test Mode (Transmitter)

Using the VS menu number 3 ‘VS Test_Tx_Start’, you can start the Direct Test Mode (Transmitter).

Using the VS menu number 3 ‘VS Test_Tx_Start’, you can set the transmit frequency (channel number), data size and data type as arguments. If no argument is given, it displays the usage of this command. Figure 7-9 shows the log of execution, when the transmit frequency is channel 0 (2,420MHz), data size is 27 bytes and data type is ALL0.

If you want to terminate the execution of Direct Test Mode (Transmitter), use the VS menu number 4 ‘VS Test_End’. Figure 7-10 shows the log after execution of Direct Test Mode (Transmitter). The number of received packets is
displayed after this test and it is always 0.

7.4.3. Direct Test Mode (Parameter Set)

Using the VS menu number 5 ‘VS Test_Set_Parameter’, you can set the parameters for the Direct Test Mode (Receiver) and Direct Test Mode (Transmitter) menu items.
Using the VS menu number 5 ‘VS Test_Set_Parameter’, you can set the number of packet receptions, the number of packet transmissions, enable or disable of burst transfer as arguments. If no argument is given, it displays the usage of this command. Figure 7-11 shows the log of execution, when the number of packet receptions is 10000 times, the number of packet transmissions is 20 time and burst transfer is disabled.

![Figure 7-12 Log of Direct Test Mode (Receiver) after Direct Test Mode Parameter Set](image1)

Figure 7-12 Log of Direct Test Mode (Receiver) after Direct Test Mode Parameter Set

Figure 7-13 shows the log of the Direct Test Mode (Transmitter) after setting of the above parameters. The direct test mode has automatically finished after sending 20 packets.

![Figure 7-13 Log of Direct test mode (transmitter) after Direct Test Mode Parameter Set](image2)

Figure 7-13 Log of Direct test mode (transmitter) after Direct Test Mode Parameter Set
7.5. Sample Custom Profile

This section explains the Sample Custom Profile (SCP) by using the GATT API below.

To use Sample Custom Profile (SCP), add "USE_SAMPLE_PROFILE" to the macro definition in the compile option of a project.

7.5.1. Sample Custom Profile Specification

Sample Custom Profile (SCP) defines two roles: Client Role and Server Role.

Table 7-3 shows the service characteristics of the SCP.

<table>
<thead>
<tr>
<th>Characteristic Name</th>
<th>Properties</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notify Characteristic</td>
<td>Notify</td>
<td>uint8_t[]</td>
<td>This characteristic is used to send any notification. The length of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>notification is from 0 to 20 bytes and can be specified by the Notify</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Length Characteristic.</td>
</tr>
<tr>
<td>Notify Characteristic - Client</td>
<td>Read/Write</td>
<td>uint16_t</td>
<td>This characteristic descriptor is used to specify ON/OFF of notification.</td>
</tr>
<tr>
<td>Characteristic Configuration descriptor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indicate Characteristic</td>
<td>Indicate</td>
<td>uint8_t[]</td>
<td>This characteristic is used to send any indication. The length of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>indication is from 0 to 20 bytes and can be specified by the Indication</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Length Characteristic.</td>
</tr>
<tr>
<td>Indicate Characteristic - Client</td>
<td>Read/Write</td>
<td>uint16_t</td>
<td>This characteristic descriptor is used to specify ON/OFF of indication.</td>
</tr>
<tr>
<td>Characteristic Configuration descriptor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interval Characteristic</td>
<td>Read/Write</td>
<td>uint16_t</td>
<td>This characteristic is used to specify the transmit interval of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>indication/notification. (unit: 10 ms)</td>
</tr>
<tr>
<td>Notify Length Characteristic</td>
<td>Read/Write</td>
<td>uint8_t</td>
<td>This characteristic is used to specify the transmit data size of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>notification.</td>
</tr>
<tr>
<td>Indicate Length Characteristic</td>
<td>Read/Write</td>
<td>uint8_t</td>
<td>This characteristic is used to specify the transmit data size of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>indication.</td>
</tr>
</tbody>
</table>
### 7.5.2. File Structure Corresponding to Sample Custom Profile

The following figure shows the file structure corresponding to the sample custom profile.

```
Renrays
  ├ BLE_Software_Ver_X_XX
  │   └ BLE_Sample
  │       └ src
  │           └ rBLE
  │               └ include
  │                   ├ rble_api_custom.h
  │                   │ rble_app.h
  │                   │ sample_profile
  │                   │   └ db_handle.h
  │                   │       └ scp
  │                   │           └ scpc.c
  │                   │         └ scps.c
  │                   │               └ sample_app
  │                   │                   └ rble_sample_app.c
  │                   └ rble_sample_custom.c
  └ RL78_GID
      └ Project_Source
          └ rBLE
              └ src
                  └ include
                      ├ rble_api_custom.h
                      └ rble_app.h
                      └ sample_profile
                          └ scp
                              └ scpc.c
                              └ scps.c
                      └ sample_app
                          └ rble_sample_app.c
                              └ rble_sample_custom.c
  └ rneas
      └ arch
          └ RL78
              └ prf_config.c
              └ prf_config.h
              └ prf_sel.h
                  └ db_handle.h
Sample program folder for PC
BLE sample program folder
Custom profile additional API header file
Sample program header file
Sample profile folder
Attribute database handles header file
Sample custom profile folder
Sample custom profile client file
Sample custom profile server file
Sample program file
Sample program file (Sample Custom Profile)
BLE software folder for BLE MCU
rBLE folder
Custom profile additional API header file
Sample program file
Sample profile folder
Sample custom profile folder
Sample custom profile client file
Sample custom profile server file
Sample program file
Sample program file (Sample Custom Profile)
Parameter file for profile
Parameter header file for profile
Profile selection configuration header file
Attribute database handles header file
```
7.5.3. API Functions defined for Sample Custom Profile

This section describes the API functions defined for the SCP (Sample Custom Profile) in detail.

7.5.3.1. RBLE_SCP_Clinet_Enable

This function is used to enable the SCP Client role.

When connecting to the SCP Server device for the first time, set con_type to RBLE_SCP_CON_CFG, and perform the configuration connection to discover service on the SCP Server device.

The result is notified by the client role enable completion event RBLE_SCP_EVENT_CLIENT_ENABLE_COMP, save the obtained service information at this time.

When connecting to the SCP Server device for the second or subsequent time, set con_type to RBLE_SCP_CON_NORMAL, and perform the normal connection by using saved service information. The service discovery is skipped and the Client role can be enabled in shorter time.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conhdl</td>
<td>Connection handle</td>
</tr>
<tr>
<td>con_type</td>
<td>Connection type</td>
</tr>
<tr>
<td>scs</td>
<td>SCP handle information</td>
</tr>
</tbody>
</table>

(call_back) Callback function for event notification

Return:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_OK</td>
<td>Success</td>
</tr>
<tr>
<td>RBLE_PARAM_ERR</td>
<td>Failure (Wrong parameter)</td>
</tr>
<tr>
<td>RBLE_STATUS_ERROR</td>
<td>Failure (The state of the SCP Client is not “Disabled”)</td>
</tr>
</tbody>
</table>

7.5.3.2. RBLE_SCP_Clinet_Disable

This function is used to disable the SCP Client role.

The result is notified by the client role disable completion event RBLE_SCP_EVENT_CLIENT_DISABLE_COMP.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conhdl</td>
<td>Connection handle</td>
</tr>
</tbody>
</table>

Return:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_OK</td>
<td>Success</td>
</tr>
<tr>
<td>RBLE_PARAM_ERR</td>
<td>Failure (Wrong parameter)</td>
</tr>
<tr>
<td>RBLE_STATUS_ERROR</td>
<td>Failure (The state of the SCP Client is not “Enabled”)</td>
</tr>
</tbody>
</table>
### 7.5.3.3. RBLE_SCP_Clinet_Read_Char

**Function Signature:**

```c
RBLE_STATUS RBLE_SCP_Client_Read_Char ( uint16_t conhdl, uint8_t char_code )
```

This function is used to read characteristic value or descriptor specified by `char_code`. The result is notified by the read characteristic response event `RBLE_SCP_EVENT_CLIENT_READ_CHAR_RESPONSE`.

**Parameters:**

- **conhdl**: Connection handle
- **char_code**: Characteristic value or configuration descriptor to be read:
  - `RBLE_SCP_SCS_NTF_CFG`: Read Notify ClientConfiguration descriptor
  - `RBLE_SCP_SCS_IND_CFG`: Read Indicate ClientConfiguration descriptor
  - `RBLE_SCP_SCS_INTERVAL`: Read Interval characteristic value
  - `RBLE_SCP_SCS_NTF_LEN`: Read Notify Length characteristic value
  - `RBLE_SCP_SCS_IND_LEN`: Read Indicate Length characteristic value

**Return:**

- `RBLE_OK`: Success
- `RBLE_PARAM_ERR`: Failure (Wrong parameter)
- `RBLE_STATUS_ERROR`: Failure (The state of the SCP Client is not “Enabled”)

### 7.5.3.4. RBLE_SCP_Clinet_Write_Char

**Function Signature:**

```c
RBLE_STATUS RBLE_SCP_Client_Write_Char ( uint16_t conhdl, uint8_t char_code, uint8_t *write_value )
```

This function is used to write characteristic value or descriptor specified by `char_code`. The result is notified by the write characteristic response event `RBLE_SCP_EVENT_CLIENT_WRITE_CHAR_RESPONSE`.

**Parameters:**

- **conhdl**: Connection handle
- **char_code**: Characteristic value or configuration descriptor to be written:
  - `RBLE_SCP_SCS_NTF_CFG`: Write Notify ClientConfiguration descriptor
  - `RBLE_SCP_SCS_IND_CFG`: Write Indicate ClientConfiguration descriptor
  - `RBLE_SCP_SCS_INTERVAL`: Write Interval characteristic value
  - `RBLE_SCP_SCS_NTF_LEN`: Write Notify Length characteristic value
  - `RBLE_SCP_SCS_IND_LEN`: Write Indicate Length characteristic value

**Return:**

- `RBLE_OK`: Success
- `RBLE_PARAM_ERR`: Failure (Wrong parameter)
- `RBLE_STATUS_ERROR`: Failure (The state of the SCP Client is not “Enabled”)

---

<table>
<thead>
<tr>
<th>conhdl</th>
<th>Connection handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>char_code</td>
<td>Characteristic value or configuration descriptor to be read:</td>
</tr>
<tr>
<td></td>
<td><code>RBLE_SCP_SCS_NTF_CFG</code>: Read Notify ClientConfiguration descriptor</td>
</tr>
<tr>
<td></td>
<td><code>RBLE_SCP_SCS_IND_CFG</code>: Read Indicate ClientConfiguration descriptor</td>
</tr>
<tr>
<td></td>
<td><code>RBLE_SCP_SCS_INTERVAL</code>: Read Interval characteristic value</td>
</tr>
<tr>
<td></td>
<td><code>RBLE_SCP_SCS_NTF_LEN</code>: Read Notify Length characteristic value</td>
</tr>
<tr>
<td></td>
<td><code>RBLE_SCP_SCS_IND_LEN</code>: Read Indicate Length characteristic value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RBLE_OK</code>: Success</td>
</tr>
<tr>
<td><code>RBLE_PARAM_ERR</code>: Failure (Wrong parameter)</td>
</tr>
<tr>
<td><code>RBLE_STATUS_ERROR</code>: Failure (The state of the SCP Client is not “Enabled”)</td>
</tr>
</tbody>
</table>
7.5.3.5. RBLE_SCP_Server_Enable

```
RBLE_STATUS RBLE_SCP_Server_Enable( uint16_t conhdl, uint8_t con_type, 
    RBLE_SCP_SERVER_PARAM *param, RBLE_SCPS_EVENT_HANDLER call_back )
```

This function is used to enable the SCP Server role.
If the Client will write the notification/indication configuration descriptor later, set RBLE_SCP_CON_CFG to the
con_type and perform the configuration connection.
If the Server writes (initializes) the notification/indication configuration descriptor, set RBLE_SCP_CON_NORMAL
to the con_type, set the initial value to the param and perform the normal connection.
The result is notified by the server role enable completion event
RBLE_SCP_EVENT_SERVER_ENABLE_COMP.

Parameters:

<table>
<thead>
<tr>
<th>Conhdl</th>
<th>Connection handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>con_type</td>
<td>Connection type</td>
</tr>
<tr>
<td>Param</td>
<td>Initial value</td>
</tr>
<tr>
<td></td>
<td>(This parameter is valid if the con_type is RBLE_SCP_CON_NORMAL)</td>
</tr>
<tr>
<td>data_ntf_en</td>
<td>Initial value for Notify ClientConfiguration descriptor</td>
</tr>
<tr>
<td>data_ind_en</td>
<td>Initial value for Indicate ClientConfiguration descriptor</td>
</tr>
<tr>
<td>call_back</td>
<td>Callback function for event notification</td>
</tr>
</tbody>
</table>

Return:

<table>
<thead>
<tr>
<th>RBLE_OK</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_PARAM_ERR</td>
<td>Failure (Wrong parameter)</td>
</tr>
<tr>
<td>RBLE_STATUS_ERROR</td>
<td>Failure (The state of the SCP Server is not “Disabled”)</td>
</tr>
</tbody>
</table>

7.5.3.6. RBLE_SCP_Server_Disable

```
RBLE_STATUS RBLE_SCP_Server_Disable( uint16_t conhdl )
```

This function is used to disable the SCP Server role.
The result is notified by the server role disable completion event
RBLE_SCP_EVENT_SERVER_DISABLE_COMP.

Parameters:

| Conhdl     | Connection handle |

Return:

<table>
<thead>
<tr>
<th>RBLE_OK</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_PARAM_ERR</td>
<td>Failure (Wrong parameter)</td>
</tr>
<tr>
<td>RBLE_STATUS_ERROR</td>
<td>Failure (The state of the SCP Server is not “Enabled”)</td>
</tr>
</tbody>
</table>
7.5.3.7. **RBLE_SCP_Server_Send_Notify**

```c
RBLE_STATUS RBLE_SCP_Server_Send_Notify ( uint16_t conhdl, 
    RBLE_SCP_NOTIFY_INFO *notify_info )
```

This function is used for the Server to send the notification data. The result is notified by the server role send notification completion event RBLE_SCP_EVENT_SERVER_SEND_NOTIFY_COMP.

**Parameters:**

<table>
<thead>
<tr>
<th>conhdl</th>
<th>Connection handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify_info</td>
<td>Notification data</td>
</tr>
<tr>
<td>data_len</td>
<td>Data size</td>
</tr>
<tr>
<td>data[ ]</td>
<td>Data</td>
</tr>
</tbody>
</table>

**Return:**

<table>
<thead>
<tr>
<th>RBLE_OK</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_PARAM_ERR</td>
<td>Failure (Wrong parameter)</td>
</tr>
<tr>
<td>RBLE_STATUS_ERROR</td>
<td>Failure (The state of the SCP Server is not “Enabled”)</td>
</tr>
</tbody>
</table>

7.5.3.8. **RBLE_SCP_Server_Send_Indicate**

```c
RBLE_STATUS RBLE_SCP_Server_Send_Indicate ( uint16_t conhdl, 
    RBLE_SCP_IND_INFO *ind_info )
```

This function is used for the Server to send the indication data. The result is notified by the server role send indication completion event RBLE_SCP_EVENT_SERVER_SEND_IND_COMP.

**Parameters:**

<table>
<thead>
<tr>
<th>conhdl</th>
<th>Connection handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>ind_info</td>
<td>Indication data</td>
</tr>
<tr>
<td>data_len</td>
<td>data_len</td>
</tr>
<tr>
<td>data[ ]</td>
<td>data[ ]</td>
</tr>
</tbody>
</table>

**Return:**

<table>
<thead>
<tr>
<th>RBLE_OK</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_PARAM_ERR</td>
<td>Failure (Wrong parameter)</td>
</tr>
<tr>
<td>RBLE_STATUS_ERROR</td>
<td>Failure (The state of the SCP Server is not “Enabled”)</td>
</tr>
</tbody>
</table>
### 7.5.4. Events defined for Sample Custom Profile

This section describes the events defined for the SCP (Sample Custom Profile) in detail.

<table>
<thead>
<tr>
<th>Role</th>
<th>Event Name</th>
<th>Description</th>
<th>Parameter Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>RBLE_SCP_EVENT_SERVER _ENABLE_COMP</td>
<td>Enable Completion Event</td>
<td>struct RBLE_SCP_Server_Enable_t{ uint16_t conhdl; RBLE_STATUS status; uint8_t reserved; } server_enable;</td>
</tr>
<tr>
<td>Server</td>
<td>RBLE_SCP_EVENT_SERVER _DISABLE_COMP</td>
<td>Disable Completion Event</td>
<td>struct RBLE_SCP_Server_Disable_t{ uint16_t conhdl; RBLE_STATUS status; uint8_t reserved; RBLE_SCP_SERVER_PARAM server_info; } server_disable;</td>
</tr>
<tr>
<td>Server</td>
<td>RBLE_SCP_EVENT_SERVER _ERROR_IND</td>
<td>Error Indication Event (Unused)</td>
<td>struct RBLE_SCP_Server_Error_Ind_t{ uint16_t conhdl; RBLE_STATUS status; uint8_t reserved; } error_ind;</td>
</tr>
<tr>
<td>Server</td>
<td>RBLE_SCP_EVENT_SERVER _SEND_NOTIFY_COMP</td>
<td>Notification Send Completion Event</td>
<td>struct RBLE_SCP_Server_Send_Notify_t{ uint16_t conhdl; RBLE_STATUS status; uint8_t reserved; } send_notify;</td>
</tr>
<tr>
<td>Server</td>
<td>RBLE_SCP_EVENT_SERVER _SEND_IND_COMP</td>
<td>Indication Send Completion Event</td>
<td>struct RBLE_SCP_Server_Send_Indicate_t{ uint16_t conhdl; RBLE_STATUS status; uint8_t reserved; } send_ind;</td>
</tr>
<tr>
<td>Server</td>
<td>RBLE_SCP_EVENT_SERVER _CHG_INDNTF_IND</td>
<td>Client Configuration Changed Event</td>
<td>struct RBLE_SCP_Server_Cfg_Indntf_Ind_t{ uint16_t conhdl; uint8_t char_code; uint8_t reserved; uint16_t cfg_val; } cfg_indntf;</td>
</tr>
<tr>
<td>Server</td>
<td>RBLE_SCP_EVENT_SERVER _CHG_CHAR_IND</td>
<td>Characteristic Changed Event</td>
<td>struct RBLE_SCP_Server_Write_Chara_Ind_t{ uint16_t conhdl; uint8_t char_code; uint8_t reserved; uint8_t value[RBLE_SCPC_WRITE_CHAR_MAX]; } write_char;</td>
</tr>
<tr>
<td>Server</td>
<td>RBLE_SCP_EVENT_SERVER _COMMAND_DISALLOWED_IND</td>
<td>Command Disallowed Notification Event (Unused)</td>
<td>struct RBLE_SCP_Server_Command_Disallowed_Ind_t{ RBLE_STATUS status; uint8_t reserved; uint16_t opcode; } cmd_disallowed_ind;</td>
</tr>
</tbody>
</table>

---

Table 7-4 Events Used by the SCP
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event Description</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_SCP_EVENT_CLIENT _ENABLE_COMP</td>
<td>Enable Completion Event</td>
<td><code>struct RBLE_SCP_Client_Enable_t{</code>&lt;br&gt;uint16_t          conhdl;&lt;br&gt;RBLE_STATUS   status;&lt;br&gt;uint8_t           reserved;&lt;br&gt;RBLE_SCS_CONTENT    scs;&lt;br}&gt;client_enable;`</td>
</tr>
<tr>
<td>RBLE_SCP_EVENT_CLIENT _DISABLE_COMP</td>
<td>Disable Completion Event</td>
<td><code>struct RBLE_SCP_Client_Disable_t{</code>&lt;br&gt;uint16_t          conhdl;&lt;br&gt;RBLE_STATUS   status;&lt;br&gt;uint8_t           reserved;&lt;br}&gt;client_disable;`</td>
</tr>
<tr>
<td>RBLE_SCP_EVENT_CLIENT _ERROR_IND</td>
<td>Error Indication Event (Unused)</td>
<td><code>struct RBLE_SCP_Client_Error_Ind_t{</code>&lt;br&gt;uint16_t          conhdl;&lt;br&gt;RBLE_STATUS   status;&lt;br&gt;uint8_t           reserved;&lt;br}&gt;error_ind;`</td>
</tr>
<tr>
<td>RBLE_SCP_EVENT_CLIENT _NOTIFY</td>
<td>Notification Received Event</td>
<td><code>struct RBLE_SCP_Client_Notify_Ind_t{</code>&lt;br&gt;uint16_t            conhdl;&lt;br&gt;uint8_t             data_len;&lt;br&gt;uint8_t             data[];`</td>
</tr>
<tr>
<td>RBLE_SCP_EVENT_CLIENT _INDICATE</td>
<td>Indication Received Event</td>
<td><code>struct RBLE_SCP_Client_Indicate_Ind_t{</code>&lt;br&gt;uint16_t            conhdl;&lt;br&gt;uint8_t             data_len;&lt;br&gt;uint8_t             data[];`</td>
</tr>
<tr>
<td>RBLE_SCP_EVENT_CLIENT _READ_CHAR_RESPONSE</td>
<td>Read Characteristic Response Event</td>
<td><code>struct RBLE_SCP_Client_Read_Char_Response_t{</code>&lt;br&gt;uint16_t           conhdl;&lt;br&gt;uint8_t             att_code;&lt;br&gt;RBLE_ATT_INFO_DATA data;`</td>
</tr>
<tr>
<td>RBLE_SCP_EVENT_CLIENT _WRITE_CHAR_RESPONSE</td>
<td>Write Characteristic Response Event</td>
<td><code>struct RBLE_SCP_Client_Write_Char_Response_t{</code>&lt;br&gt;uint16_t           conhdl;&lt;br&gt;uint8_t             att_code;&lt;br}&gt;wr_char_resp;`</td>
</tr>
<tr>
<td>RBLE_SCP_EVENT_CLIENT _COMMAND_DISALLOWED_IND</td>
<td>Command Disallowed Notification Event (Unused)</td>
<td><code>struct RBLE_SCP_Client_Command_Disallowed_Ind_t{</code>&lt;br&gt;RBLE_STATUS   status;&lt;br&gt;uint8_t           reserved;&lt;br&gt;uint16_t         opcode;&lt;br}&gt;cmd_disallowed_ind;`</td>
</tr>
</tbody>
</table>
7.5.5. Usage of the Sample Program for Sample Custom Profile

This section explains usage of the Sample Program for Sample Custom Profile (SCP).

By default, the sample program for Server role is intended to run in the Embedded configuration and the sample program for Client role is intended to run in the Modem configuration. Therefore, the Sample program for Server role operates without any external command control.

Refer to Usage of the Sample Program for Server role in detail.

If you want to run the Sample Program for Client role in Embedded configuration, disable the definition of USE_CUSTOM_DEMO macro in prf_sel.h file.

7.5.5.1. Usage of the Sample Program for Client role

This section explains usage of the Sample Program for Client role.

After connecting to the Server device using GAP command, the following steps allow you to use commands for SCP.

(1) Select Profile Test (In case of Figure 7-14, Enter 2)

![Figure 7-14 Initial Menu (the Sample Program for Client role)](image)

(2) Select Sample Custom Profile (In case of Figure 7-15, Enter 7)

![Figure 7-15 Profile Test Menu (the Sample Program for Client role)](image)
The following table explains commands provided by the Sample Program for Client role.
<table>
<thead>
<tr>
<th>Command No.</th>
<th>Operation</th>
<th>Parameters</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Server Enable</td>
<td>-</td>
<td>Controls the Server.</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Server Disable</td>
<td>-</td>
<td>Controls the Server.</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Server Send Notify</td>
<td>-</td>
<td>Controls the Server.</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>Server Send Indicate</td>
<td>-</td>
<td>Controls the Server.</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Client Enable</td>
<td>-</td>
<td>Enables Client role, by calling RBLE_SCP_Client_Enable API. (This command always performs configuration connection using SCP_CON_CFG parameter.)</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>Client Disable</td>
<td>-</td>
<td>Disables Client role, by calling RBLE_SCP_Client_Disable API.</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>Client Read Char</td>
<td>char_code</td>
<td>Reads characteristic value, by calling RBLE_SCP_Client_Read_Char API.</td>
<td>7 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The parameter specifies characteristic value or characteristic descriptor to be read.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: Client Characteristic Configuration of Notify (RBLE_SCP_SCS_NTF_CFG)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Client Characteristic Configuration of Indicate (RBLE_SCP_SCS_IND_CFG)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2: Characteristic value of Interval (RBLE_SCP_SCS_INTERVAL)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3: Characteristic value of Notify Length (RBLE_SCP_SCS_NTF_LEN)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4: Characteristic value of Indicate Length (RBLE_SCP_SCS_IND_LEN)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Client Write Char</td>
<td>char_code</td>
<td>Writes characteristic value, by calling RBLE_SCP_Client_Write_Char API.</td>
<td>8 1 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data</td>
<td>The first parameter specifies characteristic value or characteristic descriptor to be written, and the second parameter specifies data to be written.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: Client Characteristic Configuration of Notify (RBLE_SCP_SCS_NTF_CFG)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Client Characteristic Configuration of Indicate (RBLE_SCP_SCS_IND_CFG)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2: Characteristic value of Interval (RBLE_SCP_SCS_INTERVAL)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3: Characteristic value of Notify Length (RBLE_SCP_SCS_NTF_LEN)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4: Characteristic value of Indicate Length (RBLE_SCP_SCS_IND_LEN)</td>
<td></td>
</tr>
</tbody>
</table>
7.5.5.2. Usage of the Sample Program for Server role

This section explains usage of the Sample Program for Server role.

When you power on the Server device, it waits for a connect request from the Client device automatically.

When the Client device has been connected to the Server device, the Server device enables the Server role of SCP automatically and becomes ready to accept requests from the Client device.

Write the Notify and/or Indicate characteristic from the Client device. Then, if you push SW2 switch on the RL78/G1D evaluation board, it takes effect and the Server starts sending the notification and/or indication. If you push the SW2 switch again, the Server stops sending.

The notification and/or Indication are sent depending on the Interval, Notify Length and Indicate Length characteristics respectively.

Note that the unit of Interval characteristic value is 10 milliseconds.
7.6. **Simple Sample Profile**

This section describes about the simple sample profile. To use the profile, you need to add “USE_SIMPLE_SAMPLE_PROFILE” macro definition to a project configuration.

7.6.1. **Characteristic Specification**

Table 7-5 shows the simple sample profile characteristic specification.

<table>
<thead>
<tr>
<th>Characteristic Name</th>
<th>Properties</th>
<th>Format</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch State Characteristic</td>
<td>Notify</td>
<td>uint8_t</td>
<td>Notify SW4 state (PUSH/RELEASE). 0x00 is RELEASE, 0x01 is PUSH.</td>
</tr>
<tr>
<td>- Client Characteristic Configuration</td>
<td>Read/Write</td>
<td>uint16_t</td>
<td>Enable/Disable Notify.</td>
</tr>
<tr>
<td>LED Control Characteristic</td>
<td>Read/Write</td>
<td>uint8_t</td>
<td>Set/Get LED4 state (ON/OFF). 0x00 is OFF, 0x01 is ON.</td>
</tr>
</tbody>
</table>

7.6.2. **File Structure**

Following shows the simple sample profile related files.

```
Renesas
└ BLE_Software_Ver_X_XX
    └ RL78_G1D
        └ Project_Source
            └ rBLE
                └ src
                    └ sample_simple
                        └ sams.c
                        └ sams.h
                        └ console.c
                        └ console.h
                        └ rble_sample_app_peripheral.c
                        └ rble_sample_app_peripheral.h
                └ renesas
                    └ src
                        └ arch
                            └ prf_config.c
                            └ prf_config.h
                            └ ke_conf_simple.c
                            └ db_handle.h
```

7.6.3. **Details of Simple Sample Profile**

The simple sample profile is identical with “Embedded Configuration Sample Application (r01an3319)” Peripheral role sample application. Refer “Embedded Configuration Sample Application (r01an3319)” application note for the details.
7.7. Sample Program for the Direct Test Mode with RF Tester

The BLE software includes the Sample Program that supports the Direct Test Mode (DTM).

The RL78/G1D evaluation board and the RF Conformance Tester, which is used for Bluetooth Qualification Test, are connected through a 2-wire UART interface.

To use the Sample Program for DTM:
(1) Change the following macro definition from 0 to 1
(2) Rebuild (recompile) the Sample Program.

```c
#define __DTM2WIRE_UART_USE__ 0
```

The file structure corresponding to this Sample Program is shown below.

```
Renesas
└ BLE_Software_Ver_X_XX
  └ RL78_G1D
    └ Project_Source
      └ bleip
        └ src
          └ rwble
            └ rwble_config.h
      └ renesas
        └ src
          └ arch
            └ rl78
              └ arch_main.c
              └ ke_conf.c
          └ driver
            └ DTM2Wire
              └ DTM2Wire.c
              └ DTM2Wire.h
            └ uart
              └ uart.c
              └ uart.h
```

The Sample Program for DTM automatically determines its operating mode immediately after the system reset. There are two operating modes: DTM mode and Normal operating mode. When this sample program starts with DTM mode, the baud rate of 2-wire UART interface is set 9,600 baud.

Determination conditions of the operating mode is different depending on the configurations (Embedded or Modem).

(1) In the Modem configuration
The Sample Program operates in DTM mode, if the first data is received successfully (without any errors) through the two-wire UART interface after the system reset. Otherwise the Sample Program operates in normal operation mode.

(2) In the Embedded configuration
The Sample Program operates in DTM mode, if you power on the RL78/G1D evaluation board while pressing SW2 switch on the board. Otherwise the Sample Program operates in normal operation mode.

The startup sequence in each configuration is shown in Figure 7-17, Figure 7-18 and Figure 7-19.
Figure 7-17 Startup sequence in Modem configuration

Figure 7-18 Startup sequence in Embedded configuration
Figure 7-19 Operation after operating mode determination
7.8. Printf program in the Embedded configuration

At the Sample program in the Embedded Configuration, an access of Standard IO is materialized by the “console.c”.

When "printf" function is used, "printf" function in a standard library is not called, but the "Printf" function defined in the “console.c” is called by the following macro definitions.

```
#define printf Printf
```

The “Printf” function writes a formatted string to the buffer and outputs this buffer to serial port. The size of this buffer is set by the following macro definitions.

```
#define STREAM_MEMORY_MAX_LINE_SIZE 80
```

Therefore if you need output the data which is over 80 bytes, you should adjust buffer size by this macro definition.
7.9. **FW Update Sample Program**

This section explains the FW Update Sample Program.

In FW Update, One device sends FW Update data (Sender device). The other device receives FW Update data and update FW (Receiver device).

The operation image of FW Update is shown in Figure 7-20.

![Figure 7-20 Operation image of FW Update](image)

In the FW Update Sample Program, the Sender device is Modem configuration and the Receiver device is Embedded configuration. Table 7-1

### 7.9.1. **FW Update Profile Specification**

FW Update Profile defines two roles: Sender Role and Receiver Role.

Table 7-6 shows the service characteristics of the FW update profile.

<table>
<thead>
<tr>
<th>Characteristic Name</th>
<th>Properties</th>
<th>format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Control Characteristic</td>
<td>Write</td>
<td>uint8_t [ ]</td>
<td>Control information of data transmission is written by the Write Request.</td>
</tr>
<tr>
<td>Data Characteristic</td>
<td>Write without Response</td>
<td>uint8_t [ ]</td>
<td>Update data of 1 to 20 bytes are written by the Write Command.</td>
</tr>
</tbody>
</table>
7.9.2. File Structure Corresponding to FW Update Profile

The file structure corresponding to this FW Update Sample Program is shown below.

```
BLE_Software_Ver_X_XX
    BLE_Sample
       | src
       |    rBLE
       |       include
       |       l rble_api_fwup.h
       |       l sample_profile
       |       l fwup
       |       l fwups.c
       |       l sample_app
       |       l rble_sample_app.c
       |       l rble_fw_up_sender_app.c
    Fwup
       | bin
       |    lca78k0r
       |    l RL78_G1D_CE(PXP,FMP,ANP).bin
       |    l RL78_G1D_CE(HTP,BLP,HRP).bin
       | ccrl
       |    l RL78_G1D_CCE(PXP,FMP,ANP).bin
       |    l RL78_G1D_CCE(HTP,BLP,HRP).bin
       | iar_v2
       |    l RL78_G1D_IE(PXP,FMP,ANP).bin
       |    l RL78_G1D_IE(HTP,BLP,HRP).bin
    hex
       | Sender
       |    l RL78_G1D_CM(Sender).hex
       |    l RL78_G1D_CCM(Sender).hex
       |    l RL78_G1D_IM_V2(Sender).hex
       | Receiver
       |    l ca78k0r
       |    l Embedded
       |       l RL78_G1D_CE(PXP,FMP,ANP).hex
       |       l RL78_G1D_CE(HTP,BLP,HRP).hex
       | ccrl
       |    l Embedded
       |       l RL78_G1D_CCE(PXP,FMP,ANP).hex
       |       l RL78_G1D_CCE(HTP,BLP,HRP).hex
       | iar_v2
       |    l Embedded
       |       l RL78_G1D_IE(PXP,FMP,ANP).hex
       |       l RL78_G1D_IE(HTP,BLP,HRP).hex
RL78_G1D
    | Project_Source
    |    l rBLE
    |       include
    |       l rble_api_fwup.h
    |       l sample_profile
    |       l fwup
    |       l fwups.c
    |       l sample_app
    |       l rble_fw_up_receiver_app.c
```

Sample Program folder for PC
rBLE folder
FW Update profile header file
Sample profile folder
FW Update profile folder
FW Update profile sender file
Sample Program file
Sample Program file for FW Update(Sender)
Sample folder for FW Update
Binary data folder
Folder of Binary which Base hex file that was built with CA78K0R
Binary file for Embedded configuration (PXP/FMP/ANP)
Binary file for Embedded configuration (HTP/BLP/HRP)
Folder of Binary which Base hex file that was built with CC-RL
Binary file for Embedded configuration (PXP/FMP/ANP)
Binary file for Embedded configuration (HTP/BLP/HRP)
Folder of Binary which Base hex file that was built with IAR Embedded Workbench v2
Binary file for Embedded configuration (PXP/FMP/ANP)
Binary file for Embedded configuration (HTP/BLP/HRP)
Hex data folder
ROM file for Sender device
ROM file for Embedded configuration that was built with CA78K0R
ROM file for Embedded configuration that was built with CC-RL
ROM file for Embedded configuration that was built with IAR Embedded Workbench v2
ROM file for Receiver device
Folder of ROM file that was built with CA78K0R
ROM file folder for Embedded configuration
ROM file for Embedded configuration (PXP/FMP/ANP)
ROM file for Embedded configuration (HTP/BLP/HRP)
Folder of ROM file that was built with CC-RL
ROM file folder for Embedded configuration
ROM file for Embedded configuration (PXP/FMP/ANP)
ROM file for Embedded configuration (HTP/BLP/HRP)
Folder of ROM file that was built with IAR Embedded Workbench v2
ROM file folder for Embedded configuration
ROM file for Embedded configuration (PXP/FMP/ANP)
ROM file for Embedded configuration (HTP/BLP/HRP)
BLE software folder for the BLE MCU
rBLE folder
FW Update profile header file
Sample profile folder
FW Update profile folder
FW Update profile receiver file
Sample Program file
Sample Program file for FW Update(Receiver)
Procedures of operating the FW Update Sample Program are shown in the following.

(1) Write one of the following HEX file to the RL78/G1D Evaluation Board which operate as the Sender device.
   Stored folder: BLE_Software_Ver_X_XX\BLE_Sample\Fwup\hex\Sender
   File name:
   - RL78_G1D_CM(Sender).hex
   - RL78_G1D_CCM(Sender).hex
   - RL78_G1D_IM_V2(Sender).hex

(2) Write the HEX file that are stored in following folder to the RL78/G1D Evaluation Board which operate as the Receiver device.
   Stored folder: BLE_Software_Ver_X_XX\BLE_Sample\Fwup\hex\Receiver\<development environment>
   CS+ for CA, CX (CA78K0R) : ca78k0r
   e² studio / CS+ for CC (CC-RL) : ccrl
   IAR Embedded Workbench V2 : iar_v2
   [Note] Please choose the Hex file suitable for development environment.

(3) Store the FW updates data in the following folder. The FW update data uses what converted HEX file to binary format.
   Folder name: BLE_Software_Ver_X_XX\BLE_Sample\project\windows\Exe
   [Note] HEX file uses the same development environment as what was written to Receiver device. For example, when you write 'RL78_G1D_CE(PXP,FMP,ANP).hex' to the Receiver device, you store the data which converted 'RL78_G1D_CE(HTP,BLP,HRP).hex' to binary data(RL78_G1D_CE(HTP,BLP,HRP).bin).
   Binary data which already converted are stored following folder for sample binary data.
   Folder name: BLE_Software_Ver_X_XX\BLE_Sample\Fwup\bin

(4) Start the Sample Program of the Sender device. At this time, the baud rate specifies the 76800bps according to the HEX file written to Sender device. How to start the Sample Program is shown in 5.1.

(5) Start the Sample Program of the Receiver device. How to start the Sample Program is shown in 5.3.
7.9.3. API Functions defined for FW Update Profile

This section describes the API functions defined for the FWUP (FW Update Profile) in detail.

(1) RBLE_FWUP_Sender_Enable

```c
RBLE_STATUS RBLE_FWUP_Sender_Enable ( uint16_t conhdl,
                                        uint8_t  con_type,
                                        RBLE_FWUS_CONTENT *fwus,
                                        RBLE_FWUPS_EVENT_HANDLER call_back )
```

This function is used to enable the FWUP Sender role.
When connecting to the FWUP Receiver device for the first time, set con_type to RBLE_FWUP_CON_CFG, and
perform the configuration connection to discover service on the FWUP Receiver device.
The result is notified by the Sender role enable completion event
RBLE_FWUP_EVENT_SENDER_ENABLE_COMP, save the obtained service information at this time.
When connecting to the FWUP Receiver device for the second or subsequent time, set con_type to
RBLE_FWUP_CON_NORMAL, and perform the normal connection by using saved service information. The
service discovery is skipped and the Sender role can be enabled in shorter time.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conhdl</td>
<td>Connection handle</td>
</tr>
<tr>
<td>con_type</td>
<td>Connection type</td>
</tr>
<tr>
<td>fwus</td>
<td>FWUP handle information</td>
</tr>
<tr>
<td>call_back</td>
<td>Callback function for event notification</td>
</tr>
</tbody>
</table>

Return:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_OK</td>
<td>Success</td>
</tr>
<tr>
<td>RBLE_PARAM_ERR</td>
<td>Failure (Wrong parameter)</td>
</tr>
<tr>
<td>RBLE_STATUS_ERROR</td>
<td>Failure (The state of the FWUP Sender is not &quot;Disabled&quot;)</td>
</tr>
</tbody>
</table>

(2) RBLE_FWUP_Sender_Disable

```c
RBLE_STATUS RBLE_FWUP_Sender_Disable ( uint16_t conhdl )
```

This function is used to disable the FWUP Sender role.
The result is notified by the client role disable completion event
RBLE_FWUP_EVENT_SENDER_DISABLE_COMP.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conhdl</td>
<td>Connection handle</td>
</tr>
</tbody>
</table>

Return:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_OK</td>
<td>Success</td>
</tr>
<tr>
<td>RBLE_PARAM_ERR</td>
<td>Failure (Wrong parameter)</td>
</tr>
<tr>
<td>RBLE_STATUS_ERROR</td>
<td>Failure (The state of the FWUP Sender is not &quot;Enabled&quot;)</td>
</tr>
</tbody>
</table>
### (3) RBLE_FWUP_Sender_Write_Data_Cntl

```c
RBLE_STATUS RBLE_FWUP_Sender_Write_Cntl ( uint16_t  conhdl,
                                          uint8_t  type,
                                          uint8_t  block_num,
                                          uint16_t  data_size )
```

Data Control Characteristic is set.
The block_num and the data_size are effective only if the type is set to RBLE_FWUP_DATA_SEND_START.
The result is notified by the write characteristic data response event
RBLE_FWUP_EVENT_SENDER_WRITE_CHAR_RES.

**Parameters:**

<table>
<thead>
<tr>
<th>conhdl</th>
<th>Connection handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Specifying a control command type</td>
</tr>
<tr>
<td></td>
<td>RBLE_FWUP_DATA_SEND_START Data transmission start</td>
</tr>
<tr>
<td></td>
<td>RBLE_FWUP_DATA_SEND_COMP Data transmission completion (with specified size)</td>
</tr>
<tr>
<td></td>
<td>RBLE_FWUP_DATA_CHECK_WRITE Data write confirmation</td>
</tr>
<tr>
<td></td>
<td>RBLE_FWUP_DATA_SEND_FINISH Data transmission completion (all data)</td>
</tr>
<tr>
<td></td>
<td>RBLE_FWUP_DATA_CHECK_UPDATE FW Update completion confirmation</td>
</tr>
</tbody>
</table>

| block_num | Specifying the write block number of the code flash (0 to 255) |
|           | This parameter is effective only if the type is set to BLE_FWUP_DATA_SEND_START. |

| data_size | Specifying the write data size to the code flash (4 to 1024 in increments of 4 bytes) |
|           | This parameter is effective only if the type is set to BLE_FWUP_DATA_SEND_START. |

**Return:**

<table>
<thead>
<tr>
<th>RBLE_OK</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_PARAM_ERR</td>
<td>Failure (Wrong parameter)</td>
</tr>
<tr>
<td>RBLE_STATUS_ERROR</td>
<td>Failure (The state of the FWUP Sender is not “Enabled”)</td>
</tr>
</tbody>
</table>
### (4) RBLE_FWUP_Sender_Write_Data

```c
RBLE_STATUS RBLE_FWUP_Sender_Write_Data ( uint16_t  conhdl,
                                           uint8_t  *data,
                                           uint8_t  data_size )
```

Data Characteristic is set.

**Parameters:**
- `conhdl`: Connection handle
- `data`: Specifying the beginning address of the write data to Receiver
- `data_size`: Specifying the setting data size (1 to 20 bytes)

**Return:**
- `RBLE_OK`: Success
- `RBLE_PARAM_ERR`: Failure (Wrong parameter)
- `RBLE_STATUS_ERROR`: Failure (The state of the FWUP Sender is not “Enabled”)

### (5) RBLE_FWUP_Receiver_Enable

```c
RBLE_STATUS RBLE_FWUP_Receiver_Enable ( uint16_t  conhdl,
                                        RBLE_FWUPR_EVENT_HANDLER call_back )
```

This function is used to enable the FWUP Receiver role.
The result is notified by the Receiver role enable completion event
`RBLE_FWUP_EVENT_RECEIVER_ENABLE_COMP`.

**Parameters:**
- `conhdl`: Connection handle
- `call_back`: Callback function for event notification

**Return:**
- `RBLE_OK`: Success
- `RBLE_PARAM_ERR`: Failure (Wrong parameter)
- `RBLE_STATUS_ERROR`: Failure (The state of the FWUP Receiver is not “Disabled”)

### (6) RBLE_FWUP_Receiver_Disable

```c
RBLE_STATUS RBLE_FWUP_Receiver_Disable ( uint16_t  conhdl )
```

This function is used to disable the FWUP Receiver role.
The result is notified by the Receiver role disable completion event
`RBLE_FWUP_EVENT_RECEIVER_DISABLE_COMP`.

**Parameters:**
- `conhdl`: Connection handle

**Return:**
- `RBLE_OK`: Success
- `RBLE_PARAM_ERR`: Failure (Wrong parameter)
- `RBLE_STATUS_ERROR`: Failure (The state of the FWUP Receiver is not “Enabled”)

(7) RBLE_FWUP_Receiver_Send_Data_Cntl_Res

```
RBLE_STATUS RBLE_FWUP_Receiver_Send_Data_Cntl_Res ( uint16_t  conhdl,
                                               RBLE_STATUS status )
```

This function sends response to write request of Data Control characteristic. The status is set the result in accordance with the control command that is set in Data Control characteristic.

<table>
<thead>
<tr>
<th>RBLE_FWUP_DATA_SEND_START</th>
<th>If block number and the size is correct, set to RBLE_OK. Otherwise RBLE_ERR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_FWUP_DATA_SEND_COMP</td>
<td>If specified size data is received, set to RBLE_OK. Otherwise RBLE_ERR.</td>
</tr>
<tr>
<td>RBLE_FWUP_DATA_SEND_FINISH</td>
<td>Otherwise RBLE_ERR.</td>
</tr>
<tr>
<td>RBLE_FWUP_DATA_CHECK_WRITE</td>
<td>If flash write is successfully finished, set to RBLE_OK. Otherwise RBLE_ERR.</td>
</tr>
<tr>
<td>RBLE_FWUP_DATA_CHECK_UPDATE</td>
<td>If FW Update is successfully finished, set to RBLE_OK. Otherwise RBLE_ERR.</td>
</tr>
</tbody>
</table>

**Parameters:**

<table>
<thead>
<tr>
<th>conhdl</th>
<th>Connection handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>The result for received command</td>
</tr>
<tr>
<td>RBLE_OK</td>
<td>Success</td>
</tr>
<tr>
<td>RBLE_ERR</td>
<td>Failure</td>
</tr>
</tbody>
</table>

**Return:**

<table>
<thead>
<tr>
<th>RBLE_OK</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBLE_PARAM_ERR</td>
<td>Failure (Wrong parameter)</td>
</tr>
<tr>
<td>RBLE_STATUS_ERROR</td>
<td>Failure (The state of the FWUP Receiver is not &quot;Enabled&quot;)</td>
</tr>
</tbody>
</table>
### 7.9.4. Events defined for FW Update Profile

This section describes the events defined for the FWUP (FW Update Profile) in detail.

#### Table 7-7 Events Used by the FWUP

<table>
<thead>
<tr>
<th>Role</th>
<th>Event Name</th>
<th>Description</th>
<th>Parameter Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver</td>
<td>R BLE_FWUP_EVENT_RECEIVER_ENABLE_COMP</td>
<td>Enable Completion Event</td>
<td>struct R BLE_FWUP_Receiver_Enable_t{uint16_t conhdl; R BLE_STATUS status; }receiver_enable;</td>
</tr>
<tr>
<td></td>
<td>R BLE_FWUP_EVENT_RECEIVER_DISABLE_COMP</td>
<td>Disable Completion Event</td>
<td>struct R BLE_FWUP_Receiver_Disable_t{uint16_t conhdl; R BLE_STATUS status; }receiver_disable;</td>
</tr>
<tr>
<td>Receiver</td>
<td>R BLE_FWUP_EVENT_RECEIVER_CHG_DATA_CNTL_IND</td>
<td>Data Control Change Event</td>
<td>struct R BLE_FWUP_Receiver_Chg_Data_Cntl_Ind_t{uint16_t conhdl; uint8_t type; uint8_t block_num; uint16_t data_size; }data_cntl_ind;</td>
</tr>
<tr>
<td>Receiver</td>
<td>R BLE_FWUP_EVENT_RECEIVER_CHG_DATA_IND</td>
<td>Data Change Event</td>
<td>struct R BLE_FWUP_Receiver_Chg_Data_Ind_t{uint16_t conhdl; uint8_t data_size; uint8_t data[R BLE_FWUP_DATA_MAX]; }data_ind;</td>
</tr>
<tr>
<td>Sender</td>
<td>R BLE_FWUP_EVENT_S ENDER_ENABLE_COMP</td>
<td>Enable Completion Event</td>
<td>struct R BLE_FWUP_Sender_Enable_t{uint16_t conhdl; R BLE_STATUS status; uint8_t reserved; R BLE_FWUS_CONTENT fwus; }sender_enable;</td>
</tr>
<tr>
<td>Sender</td>
<td>R BLE_FWUP_EVENT_S ENDER_DISABLE_COMP</td>
<td>Disable Completion Event</td>
<td>struct R BLE_FWUP_Sender_Disable_t{uint16_t conhdl; R BLE_STATUS status; }sender_disable;</td>
</tr>
<tr>
<td>Sender</td>
<td>R BLE_FWUP_EVENT_S ENDER_WRITE_CHAR_RES</td>
<td>Write Characteristic Response Event</td>
<td>struct R BLE_FWUP_Sender_Write_Char_Res_t{uint16_t conhdl; uint8_t att_code; }wr_char_resp;</td>
</tr>
</tbody>
</table>
7.9.5. Usage of the Sample Program for FW Update Profile

When started the Sample Program of the Sender device or the Receiver device according to 7.9.2, the following content is displayed at console.

```
> BLE Sample Program Menu Version 1.00.000 --
1. GAP & SM & GATT Test
2. Profile Test
3. Vendor Specific Test
4. PTS Test Case Select
5. FW Update Start
ESC Key: Menu exit
>> rBLE Mode (ACTIVE)
>>
```

[Note] ‘5. FW Update Start’ command is not displayed on a console of the Receiver device.

Procedures of control the Sample Program are shown in the following.

(1) The Sender device (Master) gets BD address of the Receiver device by using procedure of 5.5.

(2) Push the SW2 (red frame of Figure 7-22) for the Receiver device become FW Update mode.

[Note] Until FW Update is complete after press the SW2, the Receiver device will not be able to receive command from console.
(3) Send ‘5. FW Update Start’ command to Sender device with binary file num.

Following table is correspondence of file num and file name.

<table>
<thead>
<tr>
<th>num</th>
<th>file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RL78_G1D_CE(PXP,FMP,ANP).bin</td>
</tr>
<tr>
<td>1</td>
<td>RL78_G1D_CE(HTP,BLP,HRP).bin</td>
</tr>
<tr>
<td>2</td>
<td>RL78_G1D_IE(PXP,FMP,ANP).bin</td>
</tr>
<tr>
<td>3</td>
<td>RL78_G1D_IE(HTP,BLP,HRP).bin</td>
</tr>
<tr>
<td>4</td>
<td>RL78_G1D_CCE(PXP,FMP,ANP).bin</td>
</tr>
<tr>
<td>5</td>
<td>RL78_G1D_CCE(HTP,BLP,HRP).bin</td>
</tr>
</tbody>
</table>

Figure 7-23 is example of sending binary file of ‘RL78_G1D_CE(PXP,FMP,ANP).bin.

Figure 7-23 Console log when sending ‘FW update Start’ command.(Sender device)
(4) After sending ‘5.FW Update Start’ command, the Sample Program operates automatically until FW Update is completed.

Figure 7-24 is console log when FW Update is in operation.

[Note] While FW Update is in operation, the Sample Program repeats create connection, data send and disconnect.

![Console log when FW Update is in operation.](image)

Figure 7-24 Console log when FW Update is in operation.

(5) When FW Update is completed, ‘fw update finish’ is displayed on a console of Sender device.

The Receiver device is reset and can send a command from console.
7.10. Project Setting to use FW Update Sample Program

Procedures of setting project to use FW update sample program are shown in the following.

7.10.1. Receiver device

7.10.1.1. Project Settings of IAR Embedded Workbench V2.20.1

The setup procedures of the project in IAR Embedded Workbench V2.20.1 are shown in the following.

(1) Starting the project of Embedded or Modem configuration.
(2) Select [Project] → [Option] → [C/C++Compiler] → [Preprocessor].

Change the Defined symbol form ‘noUSE_FW_UPDATE_PROFILE’ to ‘USE_FW_UPDATE_PROFILE’.

![Figure 7-25 Setting of Defined symbols.](image-url)
(3) Select [Linker] → [Config]

- **Embedded Configuration**
  Change the linker configuration file from ‘lnkr5fllagji.cf’ to ‘lnkr5fllagj_fw.icf’.

- **Modem Configuration**
  Change the linker configuration file from ‘lnkr5fllagj.icf’ to ‘lnkr5fllagj_fw_mdm.icf’.

Figure 7-26 Setting of Linker configuration file (Embedded Configuration)
(4) If need setting of force link, Select [Input] and set function which need force link.

About force link is shown 7.10.3.

If change profile by using FW Update function, Must set following function at ‘Keep symbols’.

Embedded:

```c
?F_DIV
?F_MUL
?F_SL2F
?F_UL2F
?SL_RSH_L03
?UL_RSH_L03
?0EI_VSWITCH_L10
?0SI_VSWITCH_L10
?1EC_VSWITCH_L10
?1SI_VSWITCH_L10
?2SI_VSWITCH_L10
?3SI_VSWITCH_L10
?I_VSWITCH_L10
__iar_copy_init2
__iar_packbits_init_near_single2
```  

Modem:

```c
?I_VSWITCH_L10
?3SI_VSWITCH_L10
?2SI_VSWITCH_L10
?0SI_VSWITCH_L10
__iar_copy_init2
__iar_packbits_init_near_single2
```
Figure 7-27 Setting of Keep symbols.
(5) Select [General Options] → [Library Options]
    - Embedded Configuration
      Change the Printf formatter from 'Large' to 'Small'.

(6) Click ‘OK’ and finish option setting.

(7) Run Build.
7.10.1.2. Project Settings of e² studio
The setup procedures of the project in e² studio are shown in the following.

(1) Launch the e² studio, and open the workspace.
(2) From the [Project Explorer], right-click the project of rBLE_Emb or rBLE_Mdm, select the [Renesas Tool Settings] in the context menu.
(3) From the left tree of [Tool Settings] tab, select [Compiler]→[Source], and change the following definitions from the right pane of the [Macro definition].

```
noUSE_FW_UPDATE_PROFILE  →  USE_FW_UPDATE_PROFILE
```

![Figure 7-29 Setting of Macro definition.](image-url)
(4) From the left tree of [Tool Settings] tab, select [Linker] → [Section], and click the [Import] button of the right pane, then select the following section information file for FW update.

- **Embedded**: `renesas\tools\project\e2studio\BLE_Embedded \rBLE_Emb\sect_emb_fwup.esi`
- **Modem**: `renesas\tools\project\e2studio\BLE_Modem \rBLE_Mdm\sect_mdm_fwup.esi`

![Figure 7-30 Import of section information file](image)

(5) Click ‘OK’ and finish tool settings.

(6) Run Build.
7.10.2. Sender device

The setup procedures of the project for Sender device are shown in the following.

The Sender device received FW Update data from the Sample Program at Windows.

So change operating frequency to 32MHz and change UART baud rate to 76,800bps.

[Note] It is possible to do FW Update at low clock, but FW Update time will be long.

(1) Change UART baud rate

To change the UART driver in order to 76,800bps the UART baud rate.

The serial_init() function in ‘Renesas\RL78_G1D\Project_Source\renesas\src\driver\uart\uart.c’ is changed following processing.

[Note] red word is a changing point.

```c
#if (1)
  #ifndef CONFIG_EMBEDDED
  /* MCK = fclk/n = 1MHz */
  write_sfr(SPSOL, (uint8_t)((read_sfr(SPSOL) | UART_VAL_SPS_2MHZ)));

  /* baudrate 4800bps (when MCK = 1MHz) */
  write_sfrp(UART_TXD_SDR, (uint16_t)0x1800U);
  write_sfrp(UART_RXD_SDR, (uint16_t)0x1800U);
  #else /*CONFIG_EMBEDDED*/
    ...
    #if SERIAL_U_2WIRE
    #if (1)
      #ifndef CONFIG_EMBEDDED
        /* if baudrate is 4800bps, set enable */
        stop_flg = false;
      #else /*CONFIG_EMBEDDED*/
        ...
      #endif
    #endif
    ...
  #endif /*CONFIG_EMBEDDED*/
```

(2) Change operating frequency

To change operating frequency is shown in Bluetooth Low Energy Protocol Stack User’s Manual.
7.10.3. Notes of making FW Update Environment

- force link of function

In FW Update, update code area of Application and profile.

So it is impossible to change link of runtime library or standard library before and after the FW Update.

[Note] If link of runtime library or standard library is changed, can’t use runtime library or standard library from excluded area of FW Update.

If link of runtime library or standard library is changed, you need to link to the required function of runtime library or standard library using forced link in the FW of before FW Update.

Way of forcing link is shown in 7.10.1.
7.11. References

1. Bluetooth Core Specification v4.2, Bluetooth SIG
2. Find Me Profile Specification v1.0, Bluetooth SIG
3. Immediate Alert Service Specification v1.0, Bluetooth SIG
4. Proximity Profile Specification v1.0, Bluetooth SIG
5. Link Loss Service Specification v1.0, Bluetooth SIG
6. Tx Power Service Specification v1.0, Bluetooth SIG
7. Health Thermometer Profile Specification v1.0, Bluetooth SIG
8. Health Thermometer Service Specification v1.0, Bluetooth SIG
9. Device Information Service Specification v1.1, Bluetooth SIG
10. Blood Pressure Profile Specification v1.0, Bluetooth SIG
11. Blood Pressure Service Specification v1.0, Bluetooth SIG
12. HID over GATT Profile Specification v1.0, Bluetooth SIG
13. HID Service Specification v1.0, Bluetooth SIG
14. Battery Service Specification v1.0, Bluetooth SIG
15. Scan Parameters Profile Specification v1.0, Bluetooth SIG
16. Scan Parameters Service Specification v1.0, Bluetooth SIG
17. Heart Rate Profile Specification v1.0, Bluetooth SIG
18. Heart Rate Service Specification v1.0, Bluetooth SIG
19. Cycling Speed and Cadence Profile Specification v1.0, Bluetooth SIG
20. Cycling Speed and Cadence Service Specification v1.0, Bluetooth SIG
21. Cycling Power Profile Specification v1.0, Bluetooth SIG
22. Cycling Power Service Specification v1.0, Bluetooth SIG
23. Glucose Profile Specification v1.0, Bluetooth SIG
24. Glucose Service Specification v1.0, Bluetooth SIG
25. Time Profile Specification v1.0, Bluetooth SIG
27. Next DST Change Service Specification v1.0, Bluetooth SIG
28. Reference Time Update Service Specification v1.0, Bluetooth SIG
29. Alert Notification Service Specification v1.0, Bluetooth SIG
30. Alert Notification Profile Specification v1.0, Bluetooth SIG
31. Location and Navigation Service Specification v1.0, Bluetooth SIG
32. Location and Navigation Profile Specification v1.0, Bluetooth SIG
33. Phone Alert Status Service Specification v1.0, Bluetooth SIG
34. Phone Alert Status Profile Specification v1.0, Bluetooth SIG
37. Personal Health Devices Transcoding White Paper v1.2, Bluetooth SIG
## 7.12. Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>A service is provided from a GATT server to a GATT client. The GATT server exposes some characteristics as the interface. The service prescribes how to access the exposed characteristics.</td>
</tr>
<tr>
<td>Profile</td>
<td>A profile enables implementation of a use case by using one or more services. The services used are defined in the specifications of each profile.</td>
</tr>
<tr>
<td>Characteristic</td>
<td>A characteristic is a value used to identify services. The characteristics to be exposed and their formats are defined by each service.</td>
</tr>
<tr>
<td>Role</td>
<td>Each device takes the role prescribed by the profile or service in order to implement the specified use case.</td>
</tr>
<tr>
<td>Client Characteristic Configuration Descriptor</td>
<td>This is used to control the transmission (notification / indication) of the characteristic values from the GATT server with a client characteristic configuration descriptor.</td>
</tr>
<tr>
<td>Connection Handle</td>
<td>This is the handle determined by the controller stack and is used to identify connection with a remote device. The valid handle range is between 0x0000 and 0x0EFF.</td>
</tr>
<tr>
<td>Universally Unique Identifier (UUID)</td>
<td>This is an identifier for uniquely identifying an item. In the BLE standard, a 16-bit UUID is defined for identifying services and their characteristics.</td>
</tr>
<tr>
<td>Bluetooth Device Address (BD Address)</td>
<td>This is a 48-bit address for identifying a Bluetooth device. The BLE standard defines both public and random addresses, and at least one or the other must be supported.</td>
</tr>
<tr>
<td>Public Address</td>
<td>This is an address that includes an allocated 24-bit OUI (Organizationally Unique Identifier) registered with the IEEE.</td>
</tr>
</tbody>
</table>
| Random Address                            | This is an address that contains a random number and belongs to one of the following three categories:  
• Static Address  
• Non-Resolvable Private Address  
• Resolvable Private Address                                                                 |
| Static Address                            | This is an address whose 2 most significant bits are both 1, and whose remaining 46 bits form a random number other than all 1’s or all 0’s. This static address cannot be changed until the power is switched off. |
| Non-resolvable private Address            | This is an address whose 2 most significant bits are both 0, and whose remaining 46 bits form a random number other than all 1’s or all 0’s. Static addresses and public addresses must not be equal. This type of address is used to make tracking by an attacker difficult by changing the address frequently. |
| Resolvable private Address                | This is an address generated from an IRK and a 24-bit random number. Its 2 most significant bits are 0 and 1, and the remaining higher 22 bits form a random number other than all 1’s or all 0’s. The lower 24 bits are calculated based on an IRK and the higher random number. This type of address is used to make tracking by an attacker difficult by changing the address frequently. By allocating an IRK to the peer device, the peer device can identify the communicating device by using that IRK. |
| Broadcaster                               | This is one of the roles of GAP. It is used to transmit advertising data.                                                                     |
| Observer                                  | This is one of the roles of GAP. It is used to receive advertising data.                                                                     |
| Central                                   | This is one of the roles of GAP. It is used to establish a physical link. In the link layer, it is called Master.                                 |
| Peripheral                                | This is one of the roles of GAP. It is used to accept the establishment of a physical link. In the link layer, it is called Slave.               |
| Advertising                               | Advertising is used to transmit data on a specific channel for the purpose of establishing a connection or performing data transmission.       |
| Scan                                      | Scans are used to receive advertising data. There are two types of scans:  
Passive scan, in which data is simply received, and active scan, in which additional information is requested by sending SCAN_REQ. |
<p>| White List                                | By registering known devices that are connected or bonded to a White List, it is possible to filter devices that can accept advertising data or connection requests. |</p>
<table>
<thead>
<tr>
<th>Device Name</th>
<th>This is a user-friendly name freely assigned to a Bluetooth device to identify it. In the BLE standard, the device name is exposed to the peer device by the GATT server as a GAP characteristic.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reconnection Address</td>
<td>If a non-resolvable private address is used and the address is changed frequently, not only attackers but also the peer device will have difficulty identifying the device. Therefore, the address to be used at reconnection is reported by setting a new reconnection address as the exposed reconnection address characteristic.</td>
</tr>
<tr>
<td>Scan Interval</td>
<td>This is the interval for receiving advertising data.</td>
</tr>
<tr>
<td>Scan Window</td>
<td>This is the period of time during which advertising data is received at the scan interval.</td>
</tr>
<tr>
<td>Connection Interval</td>
<td>This is the interval for transmitting and receiving data periodically following connection establishment.</td>
</tr>
<tr>
<td>Connection Event</td>
<td>This is the period of time during which data is transmitted and received at the connection interval.</td>
</tr>
<tr>
<td>Slave Latency</td>
<td>This is the period of time during which data is transmitted and received at the connection interval.</td>
</tr>
<tr>
<td>Supervision Timeout</td>
<td>This is the timeout interval after which the link is considered to have been lost when no response is received from the peer device.</td>
</tr>
<tr>
<td>Passkey Entry</td>
<td>This is a pairing method whereby a six-digit number is input by each device to the other, or a six-digit number is displayed by one of the devices and that number is input to the other device.</td>
</tr>
<tr>
<td>Just Works</td>
<td>This is a pairing method that does not require user action.</td>
</tr>
<tr>
<td>OOB</td>
<td>This is a pairing method whereby pairing is performed by using data obtained by a communication method other than Bluetooth.</td>
</tr>
<tr>
<td>Identity Resolving Key (IRK)</td>
<td>This is a 128-bit key used to generate and resolve resolvable private addresses.</td>
</tr>
<tr>
<td>Connection Signature Resolving Key (CSRK)</td>
<td>This is a 128-bit key used to create data signatures and verify the signature of incoming data.</td>
</tr>
<tr>
<td>Long Term Key (LTK)</td>
<td>This is a 128-bit key used for encryption. The key size to be used is the size agreed on during pairing.</td>
</tr>
<tr>
<td>Short Term Key (STK)</td>
<td>This is a 128-bit key used for encryption during key exchange. It is generated using TK.</td>
</tr>
<tr>
<td>Temporary Key (TK)</td>
<td>This is a 128-bit key used required for STK generation. In the case of Just Works, the TK value is 0. In the case of Passkey Entry, it is the 6-digit number that was input, and in the case of OOB, it is the OOB data.</td>
</tr>
</tbody>
</table>
Website and support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

All trademarks and registered trademarks are the property of their respective owners.
Bluetooth is a registered trademark of Bluetooth SIG, Inc. U.S.A.
EEPROM is a trademark of Renesas Electronics Corporation.
Windows, Windows NT and Windows XP are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
PC/AT is a trademark of International Business Machines Corporation.
## Revision Record

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Issued on</th>
<th>Description</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.01</td>
<td>Feb 15, 2013</td>
<td>First edition issued</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1.10</td>
<td>Mar 27, 2013</td>
<td>The descriptions on the following topics are added:</td>
<td>-</td>
<td>The usages of the following profiles are added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Compliant with the BLE S/W Ver.2.0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Serial communication</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Custom profile</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* 2-wire DTM</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Replace the captured images from the command prompt screen</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1.11</td>
<td>Apr 12, 2013</td>
<td>Update folder organization</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1.12</td>
<td>Jun 28, 2013</td>
<td>Update folder organization</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compliant with the BLE S/W Ver.2.3</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update folder organization</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added note to Usage of Sample Program</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The usages of the following profiles are added</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Heart Rate Profile</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Cycling Speed and Cadence Profile</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Cycling Power Profile</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Alert Notification Profile</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Printf program in the Embedded configuration is added</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>References specifications are added</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1.13</td>
<td>Nov 29, 2013</td>
<td>Compliant with the BLE S/W Ver.2.3</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update folder organization</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update folder organization</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added note to Usage of Sample Program</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The usages of the following profiles are added</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Heart Rate Profile</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Cycling Speed and Cadence Profile</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Cycling Power Profile</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Alert Notification Profile</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Printf program in the Embedded configuration is added</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>References specifications are added</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compliant with the Bluetooth specification v4.1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compliant with the BLE S/W Ver.0.5</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update version of VC++</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delete folder organization</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update Operating Environment and Development Environment</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update folder pass of EXE file.</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FW Update Sample Program is added</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FW Update Environment is added</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clerical error correction</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1.14</td>
<td>Sep 19, 2014</td>
<td>Compliant with the BLE S/W Ver.2.3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compliant with the Bluetooth specification v4.1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update version of VC++</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update folder pass of EXE file.</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FW Update Sample Program is added</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clerical error correction</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1.15</td>
<td>Jan 30, 2015</td>
<td>Compliant with the BLE S/W Ver.0.9</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Changed the UART baud rate for FW Update.</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>1.16</td>
<td>Apr 17, 2015</td>
<td>Compliant with the BLE S/W Ver.1.0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Add IIC interface of serial communication</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1.17</td>
<td>Jul 10, 2015</td>
<td>Compliant with the BLE S/W Ver.1.01</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Change the mode switching specification in the Modem Configuration</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>1.18</td>
<td>Oct 30, 2015</td>
<td>Compliant with the BLE S/W Ver.1.1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The description related CS+ for CC / e² studio (CC-RL) are added.</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The description for the baud rate information of DTM mode is added.</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>1.19</td>
<td>Aug 31, 2016</td>
<td>Compliant with the BLE S/W Ver.1.2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Description is changed from CD to package.</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Version information for Renesas Flash Programmer is</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>changed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>“5.1 How to Change Parameters” is moved from Appendix.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>“UART 2-wire Branch Connection” is added in Table 5-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Heading level of “7.2 Requirements and Flow Chart of Serial Communication Driver on APP MCU” is changed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>Reference value of ROM/RAM size is added.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>File structure for Sample Custom Profile is updated.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>File structure for F/W update is updated.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>The description for F/W update of IAR V2 is added.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>1. Handling of Unused Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.</td>
</tr>
<tr>
<td>— 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2. Processing at Power-on</th>
</tr>
</thead>
<tbody>
<tr>
<td>The state of the product is undefined at the moment when power is supplied.</td>
</tr>
<tr>
<td>— The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3. Prohibition of Access to Reserved Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to reserved addresses is prohibited.</td>
</tr>
<tr>
<td>— The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. Clock Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.</td>
</tr>
<tr>
<td>— 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. Moreover, 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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5. Differences between Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before changing from one product to another, i.e., to a product with a different part number, confirm that the change will not lead to problems.</td>
</tr>
<tr>
<td>— The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the 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.</td>
</tr>
</tbody>
</table>