RL78/G1D Beacon Stack

Connecting and Updating Beacon Data Sample Program

R01AN3313EJ0111
Rev.1.11
Mar. 30, 2018

Introduction

This Sample Program runs on Bluetooth® Low Energy microcontroller RL78/G1D device, and executes Advertising for providing information and executes connecting to smart phone in order to update configuration and data.

The Sample Program switches Beacon Application and Connect Application alternately. Beacon Application executes low power consumption Advertising. Connect Application works as GAP peripheral role, and executes pairing for establishing secure connection as well as communication with custom profile. As an example, the custom profile is implemented for updating configuration and data of beacon. It is not limited to this updating beacon but also possible to extend various usage.

Target Device

RL78/G1D Evaluation Board (RTK0EN0001D01001BZ)

Related documents

Document Name	Document No.
RL78/G1D	
User's Manual: Hardware	R01UH0515E
RL78/G1D Evaluation Board	
User's Manual	R30UZ0048E
E1 Emulator	
User's Manual	R20UT0398E
Additional Document for User's Manual (Notes on Connection of RL78)	R20UT1994E
Renesas Flash Programmer V3.02 Flash memory programming software	
User's Manual	R20UT3841E
CC-RL Compiler	
User's Manual	R20UT3123E
Bluetooth Low Energy Protocol Stack	
User's Manual	R01UW0095E
API Reference Manual: Basics	R01UW0088E
RL78/G1D Beacon Stack	
User's Manual	R01UW0171E
RL78 Family Flash Self-Programing Library Type01	
User's Manual	R01US0050E
RL78 Family EEPROM Emulation Library Pack02	
User's Manual	R01US0068E

Contents

1. Overview

The Sample Program executes either Beacon Operation or RF Evaluation Operation. In the Beacon Operation, it is possible to transmit Advertising packets and update Advertising data with Custom Profile by using smart phone or other Bluetooth Low Energy device. In the RF Evaluation Operation, it is possible to evaluate RL78/G1D device RF characteristic by using RF Tester.

Figure 1-1 shows the architecture of the Sample Program. The Sample Program consists of Beacon Application, Connect Application, Direct Test Mode (DTM) Application, Beacon Stack, Bluetooth Low Energy (BLE) Protocol Stack, Code Flash Library, and Data Flash Library. The sample program works on RL78/G1D Evaluation Board.

Beacon Application executes transmitting Advertising packets by using Beacon Stack.

Connect Application executes connecting to peer device by using BLE Protocol Stack and writing data by using Code Flash Library and Data Flash Library. Beacon configuration is written to Code Flash memory and pairing information is written to Data Flash memory, so that the configuration and information data are stored after power off.

DTM Application executes Direct Test Mode for evaluating RF characteristic by using BLE Protocol Stack.

Beacon Stack provides the APIs for application to execute Advertising Function.

BLE Protocol Stack provides the APIs for applications to execute Bluetooth Low Energy Functions.

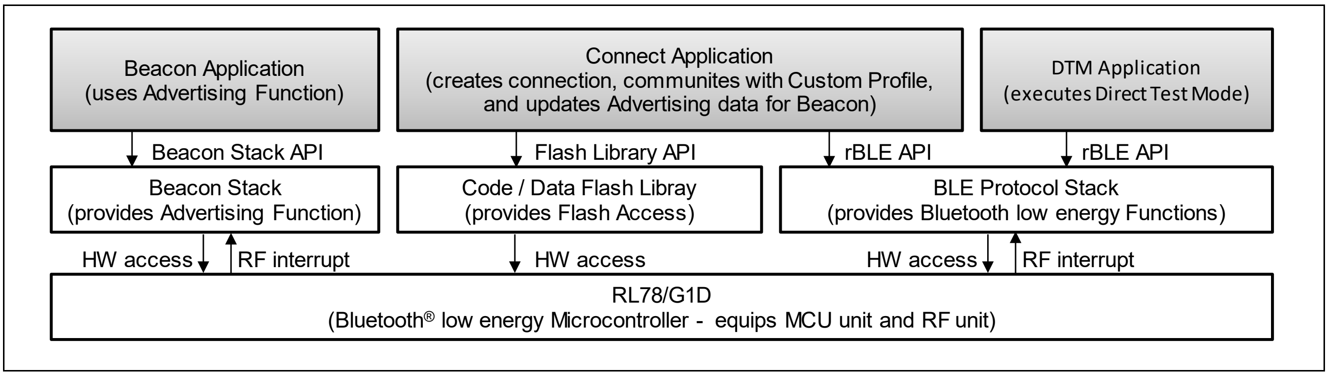


Figure 1-1 Architecture of Sample Program

Regarding to the specification of Beacon Application, Connect Application, and DTM Application, refer to chapter 5 "Specification" in this document.

Regarding to the specification of Beacon Stack, refer to RL78/G1D Beacon Stack User's Manual (R01UW0171).

Regarding to the specification of BLE Protocol Stack, refer to Bluetooth Low Energy Protocol Stack User's Manual (R01UW0095) and Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

The Sample Program runs on RL78/G1D Evaluation Board. Regarding to the details about the evaluation board, refer to RL78/G1D Evaluation Board User's Manual (R30UZ0048).

1.1 Beacon Operation

Figure 1-2 shows the overview of Beacon Operation.

When set DIP switch SW6 position-1 to ON and then power up the evaluation board, Beacon Application starts running. Beacon Application uses Beacon Stack and it transmits Non-connectable undirected Advertising packet. When switch SW2 is pushed, Beacon Application stops and Connect Application starts running. Connect Application uses BLE Protocol Stack and it transmits Connectable Undirected Advertising packet for establishing connection with peer device. When switch SW2 is pushed again or connection is not established within 30seconds, Connect Application stops and Beacon Application starts running again.

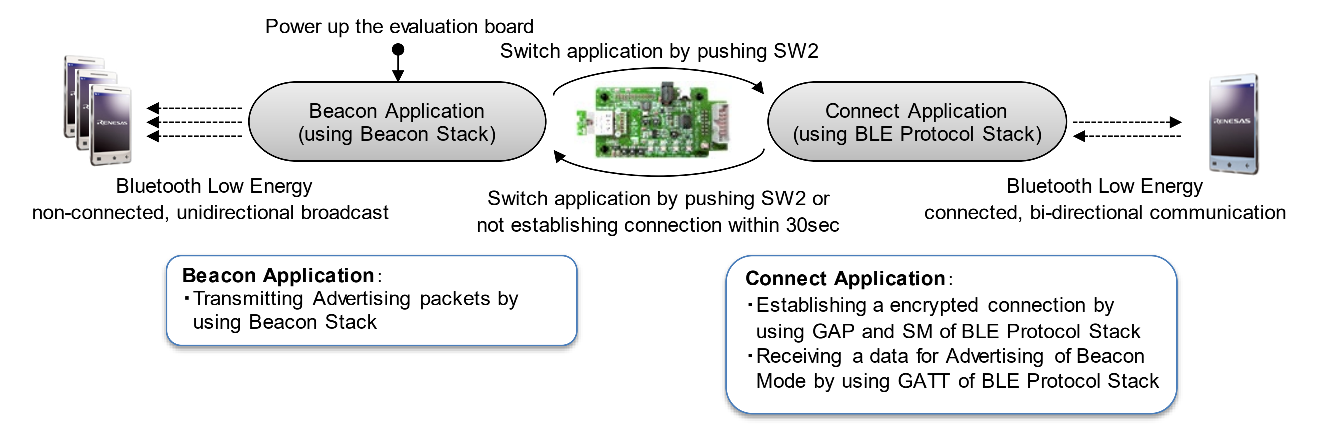


Figure 1-2 Overview of Beacon Operation

1.2 RF Evaluation Operation

Figure 1-3 shows the overview of RF Evaluation Operation.

When set DIP switch SW6 position-1 to ON and then power up the evaluation board, the Sample Program starts DTM Application. DTM Application executes Direct Test Mode corresponds to RF Test Commands, which are transferred by RF Tester through 2-wire UART. The application sends the Direct Test Mode results as RF Test Events to the RF Tester.

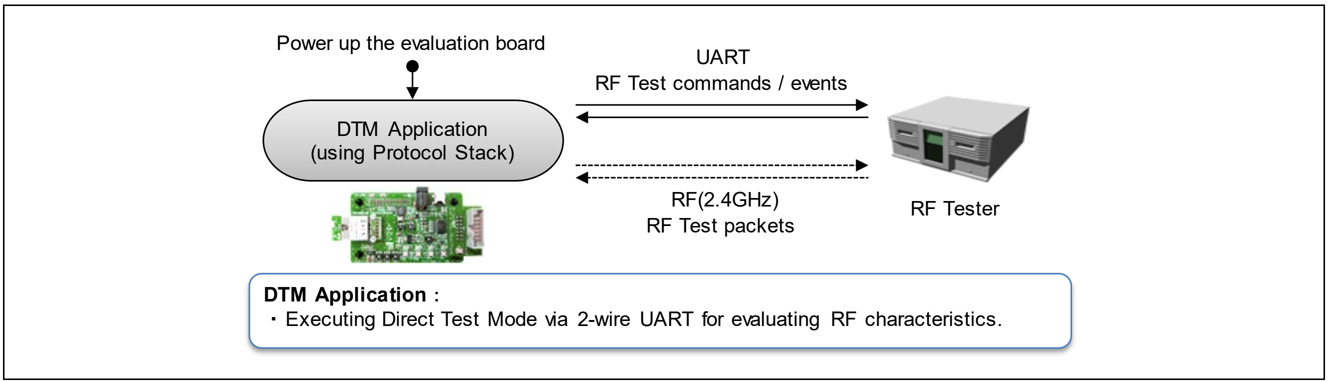


Figure 1-3 Overview of RF Evaluation Operation

2. Environment

For compiling and evaluating the Sample Program, following are the necessary environment.

- Hardware Environment
  - Host
    - PC/AT™ compatible computer
    - Processor : at least 1.6GHz
    - Main memory : at least 1Gbyte
    - Interface : USB2.0 (for connecting E1 Emulator and RL78/G1D Evaluation Board)
  - Device
    - RL78/G1D Evaluation Board (RTK0EN0001D01001BZ)
    - USB cable (A type male / mini-B type male)
    - iOS device or Android device
  - Tool
    - Renesas On-chip Debugging Emulator E1 (R0E000010KCE00)
- Software Environment
    - Windows®7 Service Pack1
    - Renesas CS+ for CC V5.00.00 / Renesas CC-RL V1.04.00
      or Renesas e² studio Version 5.3.0.023 / Renesas CC-RL V1.04.00
    - Renesas Flash Programmer v3.02.01
    - Tera Term Pro (or Terminal software which can connect to serial port)
    - UART-USB conversion device driver

  Note: It may be that device driver of UART-USB conversion IC "FT232RL" is requested is in the first connection with host. In this case, you can get the device driver from below website.

  - FTDI (Future Technology Devices International) - Drivers
    http://www.ftdichip.com/Drivers/D2XX.htm
- Software Library
    - BLE Protocol Stack : Bluetooth Low Energy Protocol Stack V1.20
    - Beacon Stack : RL78/G1D Beacon Stack V2.10
    - Code Flash Library : Flash Self Programming Library Type01 Ver2.21
    - Data Flash Library : EEPROM Emulation Library Pack02 for CC-RL Compiler Ver1.01

  It is possible to download above software libraries from Renesas Website. Regarding to the details about downloading the libraries, refer to section 4.1 "Getting Libraries" in this document.

3. File Composition

The Sample Program includes Beacon Stack library and the source code of Beacon Application, Connect Application and DTM Application. However, the following libraries are not included. So, it is necessary to download the libraries and put in suitable folders for building firmware.

- BLE Protocol Stack library
- Code Flash Library
- Data Flash Library

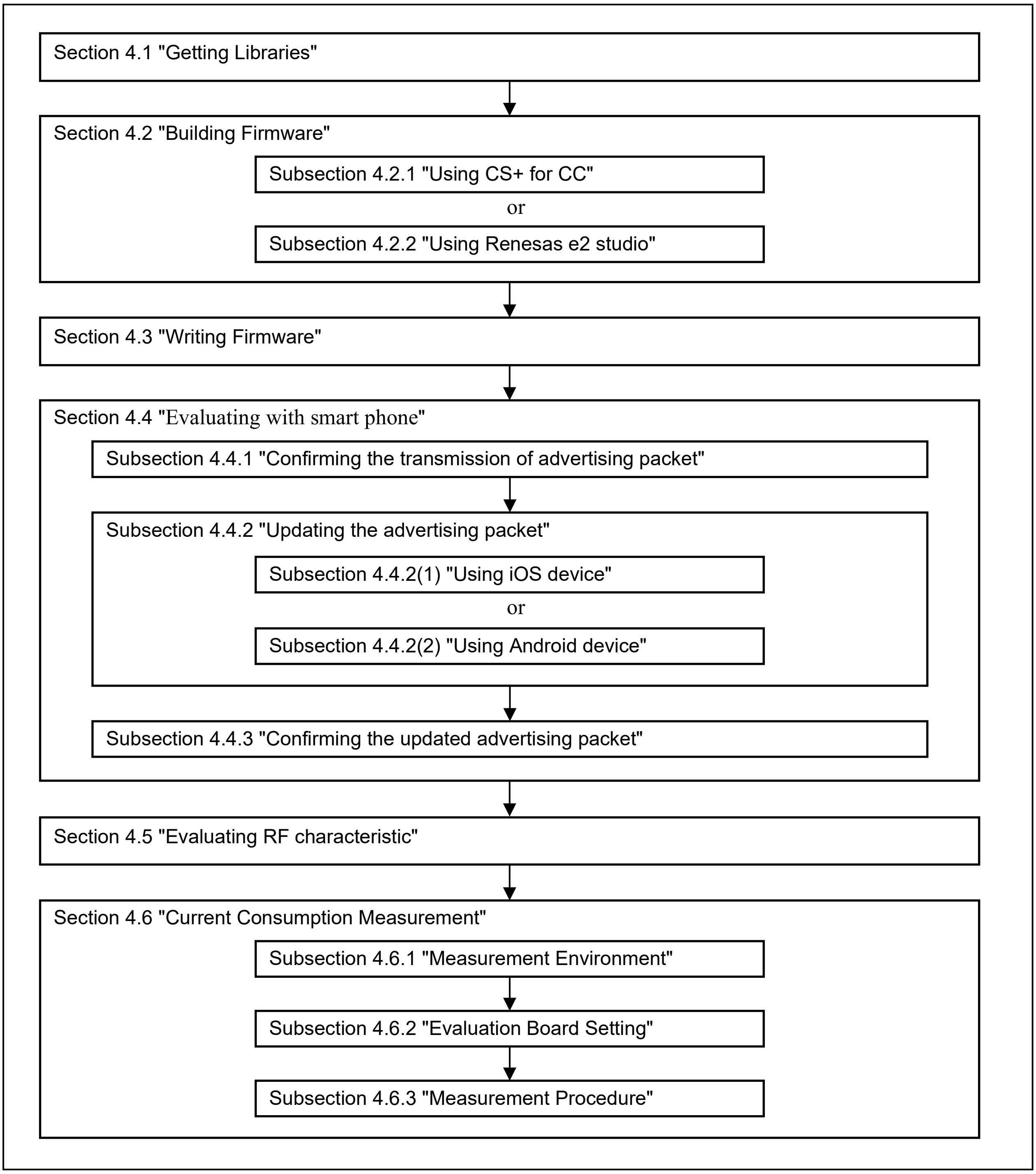
File and folder composition of the release package of the Sample Program is as shown below.

```
RL78G1D_BeaconCombination
 ├ROM_File
 │    R5F11AGJ_BcnCmb.hex                Sample Program - firmware file (R5F11AGJ)
 │    R5F11AGJ_BcnCmb_no_sw.hex          Sample Program - firmware file (R5F11AGJ) (EVABOARD_SWITCH_EN=0)
 ├RUC_File
 │    r5f11agg_syscfg.ruc                System Configuration - unique code file (R5F11AGG)
 │    r5f11agh_syscfg.ruc                System Configuration - unique code file (R5F11AGH)
 │    r5f11agj_syscfg.ruc                System Configuration - unique code file (R5F11AGJ)
 └Project_Source
   ├library
   │ │ r_arch.h                          architecture - header file
   │ │ r_compiler.h                      compiler dependent part - header file
   │ │ r_iodefine.h                      SFR definition - header file for CC-RL
   │ │ r_ll.h                            low level built-in function - header file
   │ │ r_port.h                          port access - header file
   │ ├beacon
   │ │ BLE_BEACON_CC.lib                 Beacon Stack - library for CC-RL
   │ │ r_bcn_api.h                       Beacon Stack API - header file
   │ ├protocol
   │ │ │ (empty)                         (it is necessary to put BLE Protocol Stack Library files here)
   │ │ └dummy
   │ │      types.h                      dummy header file
   │ ├codeflash
   │ │    (empty)                        (it is necessary to put Code Flash Library files here)
   │ └dataflash
   │      (empty)                        (it is necessary to put Data Flash Library files here)
   └application
     ├src
     │ │ cstart.asm                      start-up - assembly file for CC-RL
     │ │ r_config.h                      configuration - header file
     │ │ r_main.c                        entry point - header file
     │ ├beacon
     │ │ r_beacon_main.c                 Beacon Application main loop - code file
     │ │ r_beacon_isr.c                  Beacon Application interrupt - code file
     │ │ r_beacon.h                      Beacon Application - header file
     │ │ r_beacon.c                      Beacon Application - code file
     │ ├connect
     │ │ │ r_connect_main.c              Connect Application main loop - code file
     │ │ │ r_connect.h                   Connect Application - header file
     │ │ │ r_connect.c                   Connect Application - code file
     │ │ │ r_profile.h                   Custom Profile - header file
     │ │ │ r_profile.c                   Custom Profile - code file
     │ │ │ r_dtm_main.c                  DTM Application main loop - code file
     │ │ │ r_dtm.h                       DTM Application - header file
     │ │ │ r_dtm.c                       DTM Application - code file
     │ │ ├resource
     │ │ │    r_rble_core.h              BLE Protocol Stack rBLE Core Layer resource - header file
     │ │ │    r_rble_core.c              BLE Protocol Stack rBLE Core Layer resource - code file
     │ │ │    r_gatt.h                   BLE Protocol Stack GATT Layer resource - header file
```

```
│  │  │    r_gatt.c                 BLE Protocol Stack GATT Layer resource - code file
│  │  │    r_host.h                 BLE Protocol Stack Host Layer resource - header file
│  │  │    r_host.c                 BLE Protocol Stack Host Layer resource - code file
│  │  │    r_controller.c           BLE Protocol Stack Controller Layer resource - code file
│  │  │    r_kernel.h               BLE Protocol Stack Kernel resource - header file
│  │  │    r_kernel.c               BLE Protocol Stack Kernel resource - code file
│  │  │    r_stack.h                BLE Protocol Stack - header file
│  │  └optional
│  │       r_optional.c             BLE Protocol Stack optional functions - code file
│  │       r_reserved.c             BLE Protocol Stack reserved functions - code file
│  └driver
│    ├codeflash
│    │    r_codeflash.h             code flash driver - header file
│    │    r_codeflash.c             code flash driver - code file
│    ├dataflash
│    │    r_dataflash.h             data flash driver - header file
│    │    r_dataflash.c             data flash driver - code file
│    │    r_eel_descriptor_t02.h    data flash library EEPROM Emulation descriptor - header file
│    │    r_eel_descriptor_t02.c    data flash library EEPROM Emulation descriptor - code file
│    │    r_fdl_descriptor_t02.h    data flash library descriptor - header file
│    │    r_fdl_descriptor_t02.c    data flash library descriptor - code file
│    ├input
│    │    r_input.h                 external interrupt input driver - header file
│    │    r_input.c                 external interrupt input driver - code file
│    ├plf
│    │    r_plf.h                   platform driver - header file
│    │    r_plf.c                   platform driver - code file
│    └uart
│         r_uart.h                  UART driver - header file
│         r_uart.c                  UART driver - code file
└project
  ├cs_cc
  │  └BLE_Software
  │    │ BLE_Software.mtpj                project file for CS+ for CC
  │    ├R5F11AGG_BcnCmb
  │    │    R5F11AGG_BcnCmb.mtsp          subproject file for CS+ for CC (R5F11AGG)
  │    ├R5F11AGH_BcnCmb
  │    │    R5F11AGH_BcnCmb.mtsp          subproject file for CS+ for CC (R5F11AGH)
  │    └R5F11AGJ_BcnCmb
  │         R5F11AGJ_BcnCmb.mtsp          subproject file for CS+ for CC (R5F11AGJ)
  └e2_cc
    └BLE_Software
      ├R5F11AGG_BcnCmb
      │    .project                 project composition file for e2 studio (R5F11AGG)
      │    .cproject                project configuration file for e2 studio (R5F11AGG)
      │    .info                    IDE information file for e2 studio (R5F11AGG)
      │    .DefaultBuildlinker      linker configuration file for e2 studio (R5F11AGG)
      ├R5F11AGH_BcnCmb
      │    .project                 project composition file for e2 studio (R5F11AGH)
      │    .cproject                project configuration file for e2 studio (R5F11AGH)
      │    .info                    IDE information file for e2 studio (R5F11AGH)
      │    .DefaultBuildlinker      linker configuration file for e2 studio (R5F11AGH)
      └R5F11AGJ_BcnCmb
           .project                 project composition file for e2 studio (R5F11AGJ)
           .cproject                project configuration file for e2 studio (R5F11AGJ)
           .info                    IDE information file for e2 studio (R5F11AGJ)
           .DefaultBuildlinker      linker configuration file for e2 studio (R5F11AGJ)
```

4. Evaluation Procedure

This chapter describes evaluation procedure of the Sample Program. The evaluation procedure consists of six steps: Getting Libraries, Building Firmware, Writing Firmware, Evaluating with smart phone, Evaluating RF characteristic, and Current Consumption Measurement.



4.1 Getting Libraries

Before building the Sample Program firmware, it is necessary to download below libraries from Renesas Website.

- BLE Protocol Stack:
  - Bluetooth Low Energy Protocol Stack V1.20
    https://www.renesas.com/software-tool/bluetooth-low-energy-protocol-stack-rl78-family
- Code Flash Library:
  - Flash Self Programming Library Type01 Package Ver.3.00 for the RL78 Family [for the CA78K0R/CC-RL Compiler]
    https://www.renesas.com/software-tool/code-flash-libraries-flash-self-programming-libraries
- Data Flash Library:
  - EEPROM Emulation Library Pack02 Package Ver.2.00(for CA78K0R/CC-RL Compiler) for RL78 Family
    https://www.renesas.com/software-tool/data-flash-libraries

After downloading the libraries, copy respective released library files into specified folders of the Sample Program. Respective downloaded library paths are as shown below.

- Protocol Stack:
  - BLE_Software_Ver_x_xx\RL78_G1D\Project_Source\renesas\lib\BLE_rBLE_lib_CCRL.lib
  - BLE_Software_Ver_x_xx\RL78_G1D\Project_Source\renesas\lib\BLE_HOST_lib_CCRL.lib
  - BLE_Software_Ver_x_xx\RL78_G1D\Project_Source\renesas\lib\BLE_CONTROLLER_LIB_CCRL.lib
  - BLE_Software_Ver_x_xx\RL78_G1D\Project_Source\rBLE\src\include\rble_api.h
  - BLE_Software_Ver_x_xx\RL78_G1D\Project_Source\rBLE\src\include\rble.h
- Code Flash Library:
  - FSLRL78 Type01\V2.21B\CCRL_V2.21\CCRL\V2.21\librl78\fsl.lib
  - FSLRL78 Type01\V2.21B\CCRL_V2.21\CCRL\V2.21\incrl78\fsl.h
  - FSLRL78 Type01\V2.21B\CCRL_V2.21\CCRL\V2.21\incrl78\fsl_types.h
- Data Flash Library:
  - EELRL78 Pack02\V1.01\librl78\eel.lib
  - EELRL78 Pack02\V1.01\librl78\fdl.lib
  - EELRL78 Pack02\V1.01\incrl78\eel.h
  - EELRL78 Pack02\V1.01\incrl78\eel_types.h
  - EELRL78 Pack02\V1.01\incrl78\fdl.h
  - EELRL78 Pack02\V1.01\incrl78\fdl_types.h

The above files that needed to copy into folders of the Sample Program are as shown below.

```
RL78G1D_BeaconCombination
 └Project_Source
   └library
     ├protocol
     │ BLE_rBLE_lib_CCRL.lib          Protocol Stack rBLE Layer - library file
     │ BLE_HOST_lib_CCRL.lib          Protocol Stack Host Layer - library file
     │ BLE_CONTROLLER_LIB_CCRL.lib    Protocol Stack Controller Layer - library file
     │ rble_api.h                     Protocol Stack rBLE API - header file
     │ rble.h                         Protocol Stack rBLE definitions - header file
     ├codeflash
     │ fsl.lib                        Code Flash Library - library file
     │ fsl.h                          Code Flash Library - header file
     │ fsl_types.h                    Code Flash Library type definition - header file
     └dataflash
       eel.lib                        Data Flash Library EEPROM Emulation - library file
       eel.h                          Data Flash Library EEPROM Emulation - header file
       eel_types.h                    Data Flash Library EEPROM Emulation type definition - header file
       fdl.lib                        Data Flash Library - library file
       fdl.h                          Data Flash Library - header file
       fdl_types.h                    Data Flash Library type definition - header file
```

4.2 Building Firmware

After adding the necessary libraries from aforementioned section 4.1, the project for the Sample Program is ready to build firmware. Building Sample Program firmware can be used either CS+ for CC or e[2] studio as IDE (Integrated Development Environment).

By building the Sample Program with default settings, the firmware R5F11AGJ_BcnCmp.hex file that is the same HEX file included in release package is generated.

If using HEX file included in release package for evaluation, you can skip below building procedures.

4.2.1 Using CS+ for CC

1. Start CS+ for CC and open the project "BLE_Software.mtpj" from menu bar [File]→[Open File]
   - Project_Source\application\project\cs_cc\BLE_Software\
2. Select [Build]→[Rebuild project] and confirm that compiling is successful.
3. Confirm that the firmware R5F11AGJ_BcnCmb.hex is generated in the place of below path.
   - Project_Source\application\project\cs_cc\BLE_Software\R5G11AGJ_BcnCmb\DefaultBuild\

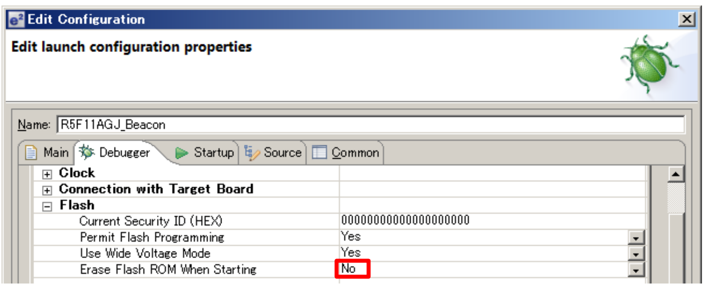
4.2.2 Using Renesas e[2] studio

1. Start Renesas e[2] studio and select below path as a workspace.
   - Project_Source\
2. Select [File]→[Import] in order to open Import dialog.
3. Select [General]→[Existing Project into Workspace] and click [Next].
4. Select below path as a root folder and confirm R5F11AGJ_BcnCmb is selected in [Projects].
   - Project_Source\
5. Click [Finish] in order to close Import dialog.
6. Close [Welcome].
7. Select R5F11AGJ_BcnCmb in the Project Explorer.
8. Select [Project]→[Build Project] and confirm that compiling is success.
9. Confirm that the firmware R5F11AGJ_BcnCmb.hex is generated in the place of below path.
   - Project_Source\application\project\e2_cc\BLE_Software\R5F11AGJ_BcnCmb\DefaultBuild\

Note: By default, debugger setting of e[2] studio erases Flash memory before writing firmware.

When developing by using e[2] studio, change the debugger setting before starting debugging, to avoid erasing Shipping Checking Flag and Device Address which already been written in RL78/G1D Module. Disconnect E1 Emulator from RL78/G1D Module when changing the debugger setting.

- Select [Debugger] tab in [Edit launch configuration properties] dialog, and set [No] in [Erase Flash ROM When Starting].



4.3 Writing Firmware

When writing the firmware of the Sample Program, Host machine and E1 Emulator is used. It is connected by USB cable between Host machine and E1 Emulator, between Host machine and the evaluation board. It is connected by User I/F cable between E1 Emulator and the evaluation board.

Regarding to the details of E1 Emulator, refer to E1 Emulator User's Manual (R20UT0398) and E1 Emulator Additional Document for User's Manual (Notes on Connection of RL78) (R20UT1994).

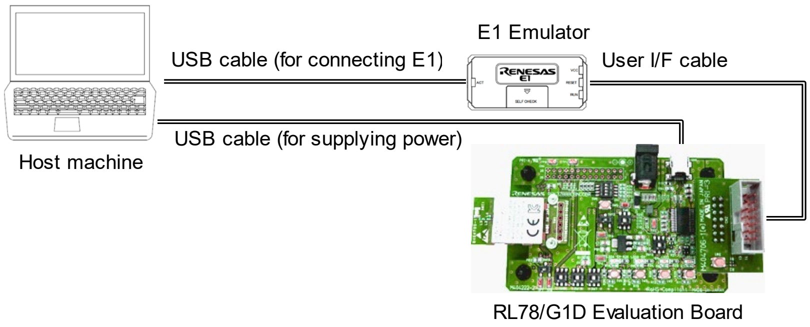


Figure 4-1 Evaluation Board Operation

Table 4-1 shows the slide switch settings for evaluating the Sample Program.

Table 4-1 Slide Switch Settings for evaluating the Sample Program

Switch	Setting	Description
SW7	2-3 connected (right) (default setting)	Power is supplied from the AC Power Supply Adapter through Power jack (J1) or USB interface (CN3) via a Regulator.
SW8	2-3 connected (right)	Power is supplied from USB. If it is necessary to supply from AC Power Supply Adapter, set 1-2 connected (left).
SW9	1-2 connected (left)	Connect to external extension interface.
SW10	1-2 connected (left) (default setting)	Power is supplied to the module.
SW11	2-3 connected (right) (default setting)	Power is supplied from a source other than the E1 Emulator
SW12	2-3 connected (right) (default setting)	(Fixed)
SW13	2-3 connected (right)	USB interface is disconnected.

For writing Sample Program firmware, Renesas Flash Programmer (RFP) is used.

There is a Unique Code Embedding Function in RFP, so that it is possible to write same firmware with each different system configuration to each individual device. Regarding to the Unique Code Embedding Function, refer to subsection 2.3.6 "[Unique Code] Tabbed Page" in Renesas Flash Programmer V3.02 Flash memory programming software User's Manual (R20UT3841).

Procedure of writing the Sample Program firmware into RL78/G1D Evaluation Board is shown below.

1. Set the switches of Evaluation Board according to **Table 4-1**.
2. Connect E1 Emulator to Evaluation Board, and connect E1 Emulator to PC.
3. Connect Evaluation Board to PC or AC-USB Power Supply Adaptor, and start to supply power.
4. Start RFP, and create project in accordance with below procedures.

   Note that after creating project once, below procedures can be skipped for next time.

   4-1. Select [File]→[Create a new project].

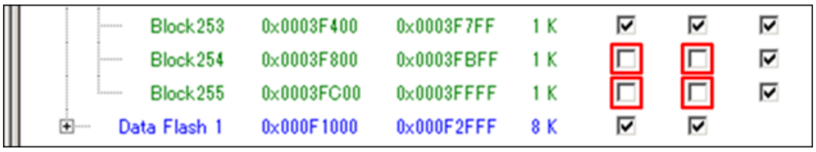
   4-2. Select [RL78] as a Microcontroller, and input a project name, and click [Connect] in [Create New Project] dialog.

   4-3. Confirm [Operation completed] message in Log output panel.
5. Select the firmware R5F11AGJ_BcnCmb.hex at [Program File].
6. Prevent erasing Block 254, 255 in Code Flash memory in accordance with below procedure.

   Note that Shipping Check Flag is written in Block 254 and Device Address is written in Block 255 in the case of RL78/G1D Module.

   6-1. Select [Operation Setting] tab, and select [Erase Selected Blocks] at [Erase Option].

   6-2. Select [Block Setting] tab, and uncheck each [Erase], [P.V] of Block 254, 255.



Block253	0x0003F400	0x0003F7FF	1 K	☑	☑	☑
Block254	0x0003F800	0x0003FBFF	1 K	☐	☐	☑
Block255	0x0003FC00	0x0003FFFF	1 K	☐	☐	☑
Data Flash 1	0x000F1000	0x000F2FFF	8 K	☑	☑	

7. (Optional) If it is necessary to change System Configuration, set unique code in accordance with below procedure.

   7-1. Select [Unique Code] tab.

   7-2. Check [Enable].

   7-3. Select the below unique code file at [Unique Code File].

   - RUC_File\r5f11agj_syscfg.ruc

   7-4. Go back to [Operation] tab.
8. Click [Start] button to start writing the firmware, and confirm [Operation completed] message.
9. Disconnect E1 Emulator and Power Supply from the Evaluation Board.

4.4 Evaluating with smart phone

To evaluate the Sample Program with smart phone, there is three steps: Confirming the transmission of advertising packet, Updating the advertising packet, and Confirming the updated advertising packet.

The Sample Program uses some Switches (SW) and LED indicators on the evaluation board for user interface. **Figure 4-2** shows the switches SW and the LED indicators used by Sample Program. The status of DIP switch SW6 position-1 is read when initialize the Sample Program, and SW2 is used external interrupt input for triggering user action Beacon Operation.

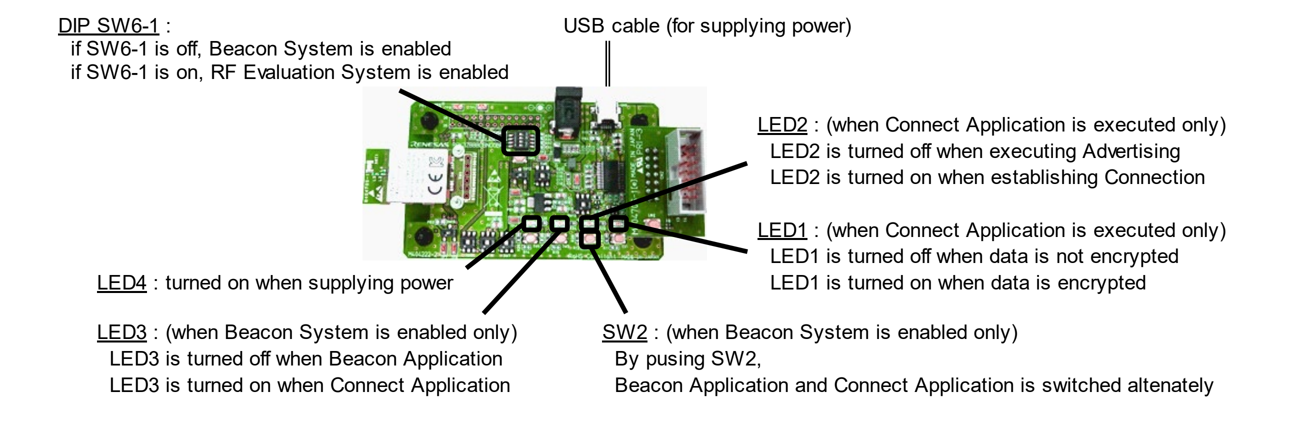
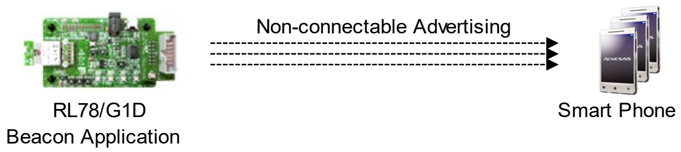


Figure 4-2 Evaluation Board Operation

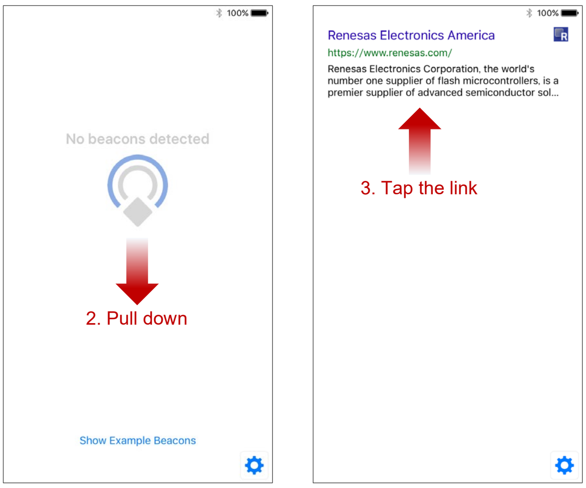
4.4.1 Confirming the transmission of advertising packet

This subsection describes procedure for enabling Beacon Application and confirming transmitted Advertising packet from the evaluation board to Smart Phone.



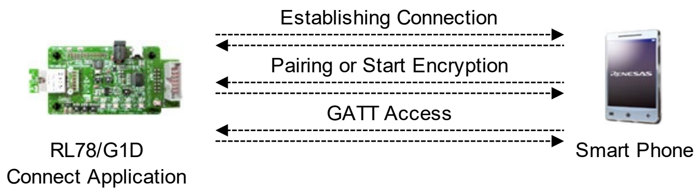
First, supply +5 DC power via Power jack (J1) or USB interface (CN3) to the evaluation board, which is programmed with the Sample Program firmware. Just after power on with setting OFF to DIP switch SW6 position-1, the Sample Program executes Beacon Application and turn on the LED4. The Beacon Application transmits Eddystone-URL packet by default. Using smart phone application, you can receive the advertising packet. The smart phone application procedure is very much similar for both iOS device and Android device.

1. In order to receive Eddystone-URL packet, install below application to smart phone.
   - for Android device, Physical Web - Google Play
     https://play.google.com/store/apps/details?id=physical_web.org.physicalweb
   - for iOS device, Physical Web - App Store
     https://itunes.apple.com/app/physical-web/id927653608
2. Run the smart phone application and search the Eddystone beacons by pulling down the display.
3. When receive the Eddystone-URL packet from the Sample Program, below URL is displayed to link the web page.
   - Renesas Electronics
     https://www.renesas.com/



4.4.2 Updating the advertising packet

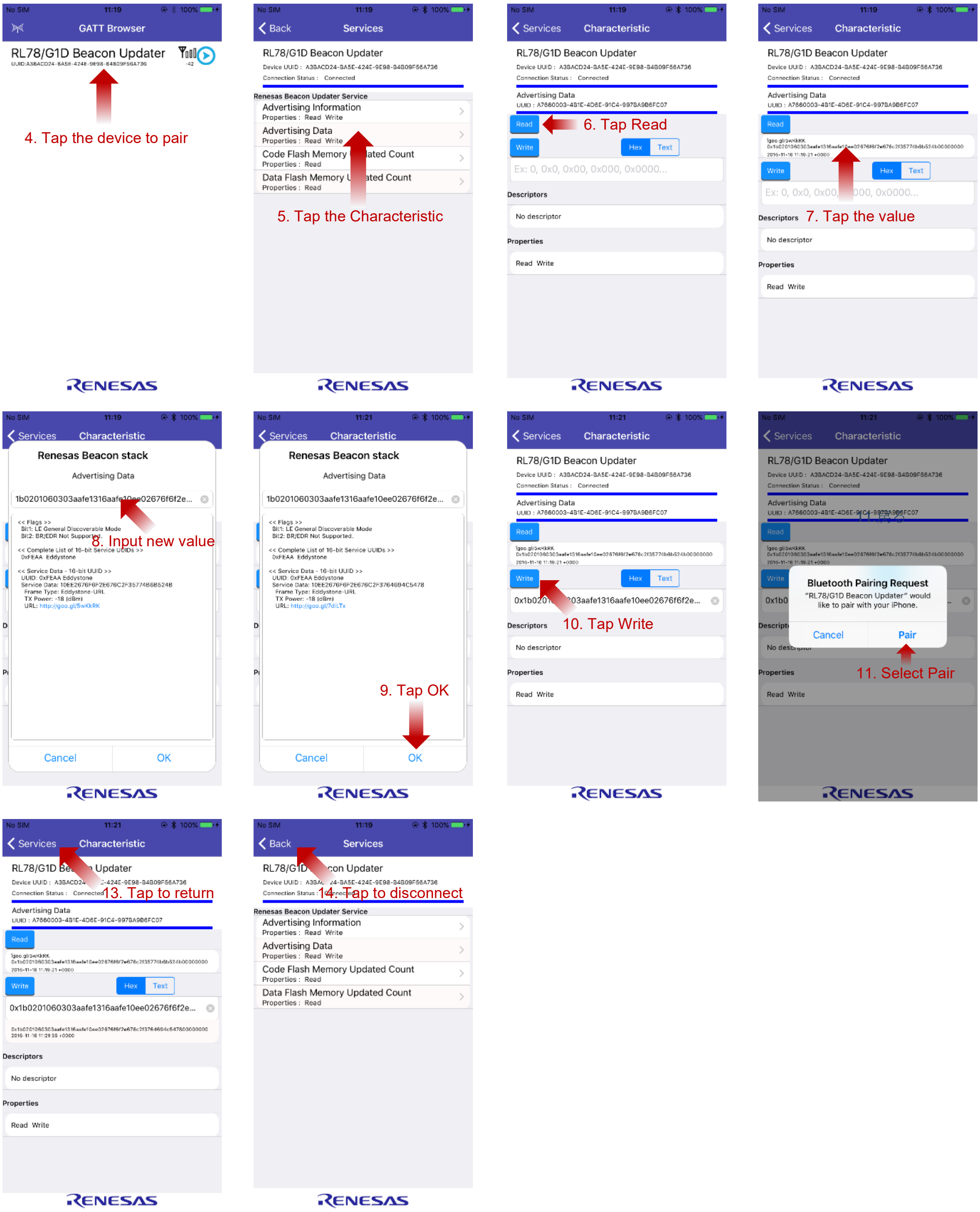
This subsection describes procedure for enabling Connect Application and updating Advertising packet by using Smart Phone.



Pushing SW2 on the evaluation board can change Beacon Application and Connect Application alternately when the Sample Program is running in Beacon Operation. When Connect Application works, not only LED4 but also LED3 are turned on. Connect Application transmits connectable advertising packet in order to connect peer device. By connecting Smart Phone to RL78/G1D device and accessing GATT, it is possible to update Advertising data Beacon Application transmits.

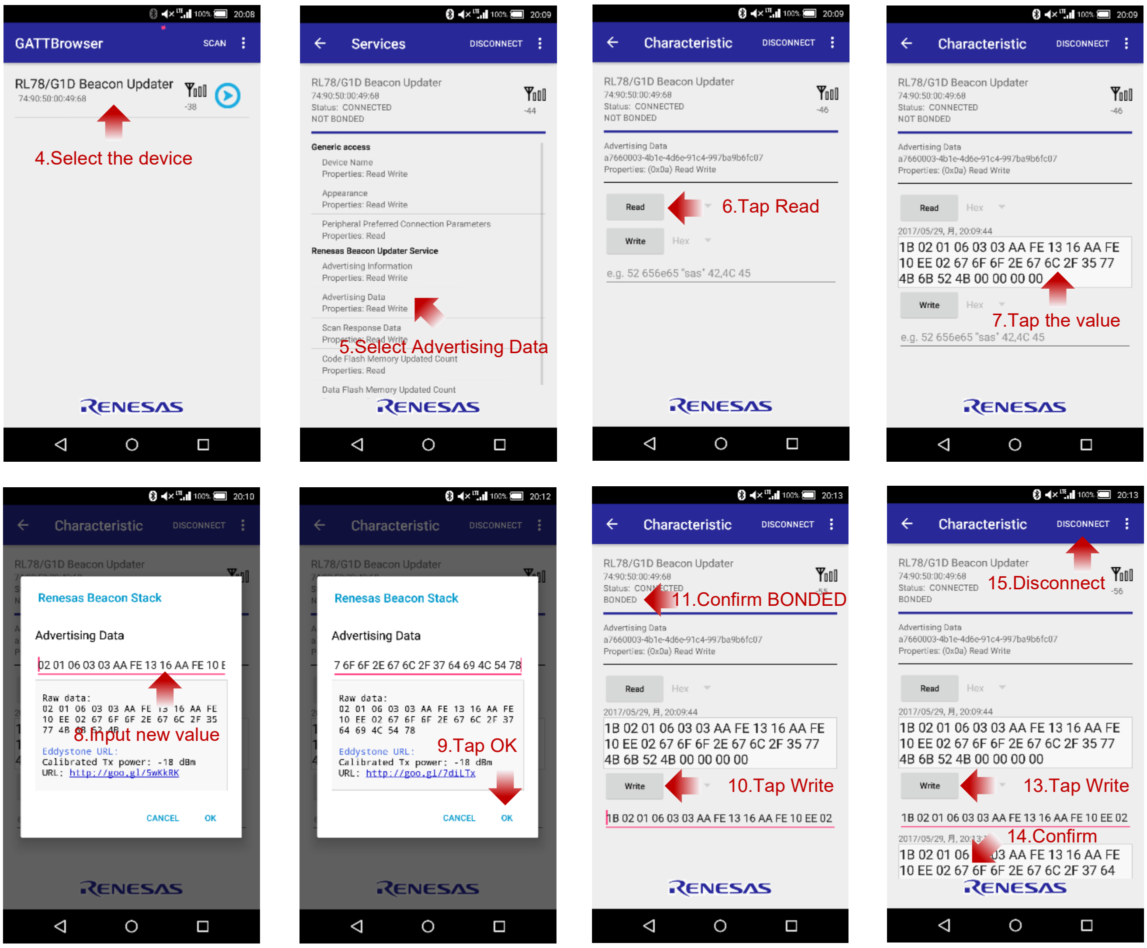
(1) **Using iOS device**

1. iOS device needs GATT Client application to access Custom Profile of Connect Application. As an example of GATT Client application, install and use below application.
   - GATTBrowser - App Store
     https://itunes.apple.com/app/gattbrowser/id1163057977
2. Push SW2 on the evaluation board and confirm LED3 is on.
3. Run the smart phone application and start scanning devices.
4. Select the "RL78/G1D Beacon Updater" to establish connection.
5. Select the Advertising Data characteristic of the Renesas Beacon Updater service.
6. Tap "Read" button.
7. Tap the value displayed below "Read" button.
8. In Advertising Data dialog, key in below value as new Advertising data.
   - Advertising Data value, Eddystone-URL including shortened URL to https://www.bluetooth.com/
     1B0201060303AAFE1316AAFE10EE02676F6F2E676C2F3764694C547800000000
9. Tap "OK" button in Advertising Data dialog.
10. Tap "Write" button.
11. Tap "Pair" button in pairing request dialog.
12. Confirm LED1 on the evaluation board is on.
13. Return to scanning view to disconnect.
14. Confirm LED1 and LED2 on the evaluation board are off.


4. Tap the device to pair
5. Tap the Characteristic
6. Tap Read
7. Tap the value
8. Input new value
9. Tap OK
10. Tap Write
11. Select Pair
13. Tap to return
14. Tap to disconnect

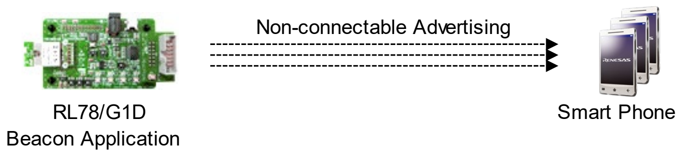
(2) **Using Android device**

1. Android device needs GATT Client application to access Custom Profile of Connect Application. As an example of GATT Client application, install and use below application.
   - GATTBrowser - Google Play
     https://play.google.com/store/apps/details?id=com.renesas.ble.gattbrowser
2. Push SW2 on the evaluation board and confirm LED3 is on.
3. Run the smart phone application and start scanning devices.
4. Select the "RL78/G1D Beacon Updater" to establish connection.
5. Select the Advertising Data characteristic of the Renesas Beacon Updater service.
6. Tap "Read" button.
7. Tap the value displayed below "Read" button.
8. In Advertising Data dialog, key in below value as new Advertising data.
   - Advertising Data value, Eddystone-URL including shortened URL to https://www.bluetooth.com/
     0x1B0201060303AAFE1316AAFE10EE02676F6F2E676C2F3764694C5478
9. Tap "OK" button in Advertising Data dialog.
10. Tap "Write" button.
11. Confirm BONDED by pairing.
12. Confirm LED1 is on.
13. Tap "Write" button again.
14. Confirm writing succeeded.
15. Tap the DISCONNECT
16. Confirm LED1 and LED2 on the evaluation board is off.


4.Select the device
5.Select Advertising Data
6.Tap Read
7.Tap the value
8.Input new value
9.Tap OK
10.Tap Write
11.Confirm BONDED
13.Tap Write
14.Confirm
15.Disconnect

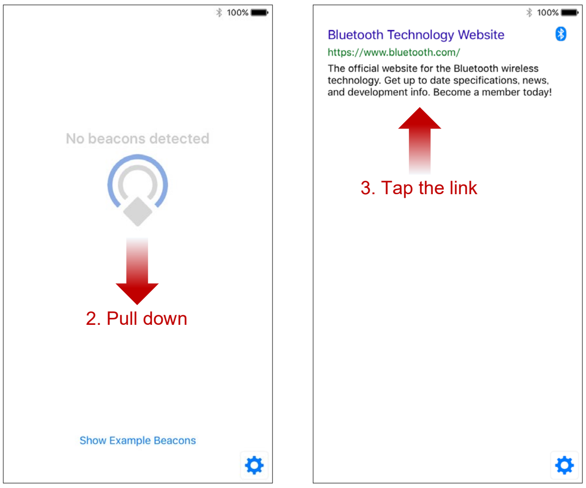
4.4.3 Confirming the updated advertising packet

This subsection describes procedure for enabling Beacon Application again and confirming that Advertising packet is updated.



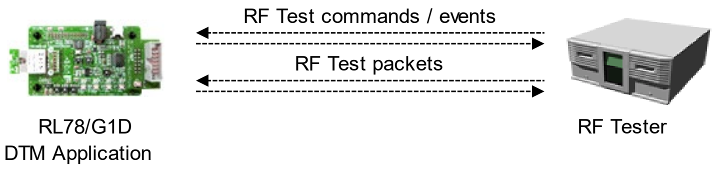
Advertising data for Beacon Application is stored into Code Flash memory after updating the characteristic value of Custom Profile and disconnection. Again, you can confirm the updated advertising data with using Beacon Application and smart phone. Here, the procedure for both iOS device and Android device are very similar.

1. Restart Beacon Application, by pushing SW2 on the evaluation board, or reset the MCU, or power-cycling to evaluation board.
2. Run the smart phone application installed in subsection 4.4.1 and search the advertising packet by pulling down the display.
3. When receive new Eddystone-URL packet from the Sample Program, link to below new URL is displayed to link the web page.
   - Bluetooth Technology Website
     https://www.bluetooth.com/



4.5 Evaluating RF characteristic

This subsection describes procedure for enabling DTM Application and evaluating RF characteristic of RL78/G1D device.



Before supplying power to the evaluation board, turn on the DIP switch SW6 position-1. Then by supplying the power to the board, the Sample Program executes the DTM Application. Now, you can test RF characteristic of RL78/G1D device by using RF Tester. Below is the procedure for testing RF characteristic of RL78/G1D device.

1. Turn on the DIP switch SW6 position-1, which is on the evaluation board.
2. Connect UART TxD0 pin, RxD0 pin, and GND pin on the evaluation board to pins of RF Tester. If logic level of the signals is different between RL78/G1D device and RF Tester, connect through logic level converter.
3. Refer to respective manuals of RF Tester and set UART settings according to **Table 4-2**.
4. Refer to respective manuals of RF Tester and start Direct Test Mode.

Table 4-2 UART Settings

Setting	Value
Baud rate	9600bps
Data bit length	8bit
Parity	None
Stop bit length	1bit
Flow control	None

4.6 Current Consumption Measurement

This section describes current consumption measurement for using RL78/G1D Evaluation Board. Regarding to the details of RL78/G1D Evaluation Board (RTK0EN0001D01001BZ), refer to RL78/G1D Evaluation Board User's Manual (R30UZ0048).

4.6.1 Measurement Environment

Table 4-3 shows the necessary equipment for current consumption measurement. Regarding to the details on how to use each equipment, refer to respective manuals of each equipment.

Table 4-3 Necessary Equipment for Current Consumption Measurement

Equipment	Role	Example Equipment
Power Source	Supplying power to RL78/G1D	Stabilized power supply or Battery Note that supply voltage shall be in the range of the RL78/G1D operation voltage
Measurement Equipment	Indicating and logging the result of measurement	Oscilloscope
Voltage Detector	Detecting the operation voltage of RL78/G1D	Voltage Probe
Current Detector	Detecting the current consumption of RL78/G1D	Current Probe with clamp, or combination of Shunt Resistor and Voltage Probe Note that recommended resistor is 10 ohm.

Figure 4-3 shows the measurement environment which uses current probe as current detector. In this environment, the current consumption of RL78/G1D is the result of measuring between terminal TP7 and TP8 of the evaluation board by current probe.

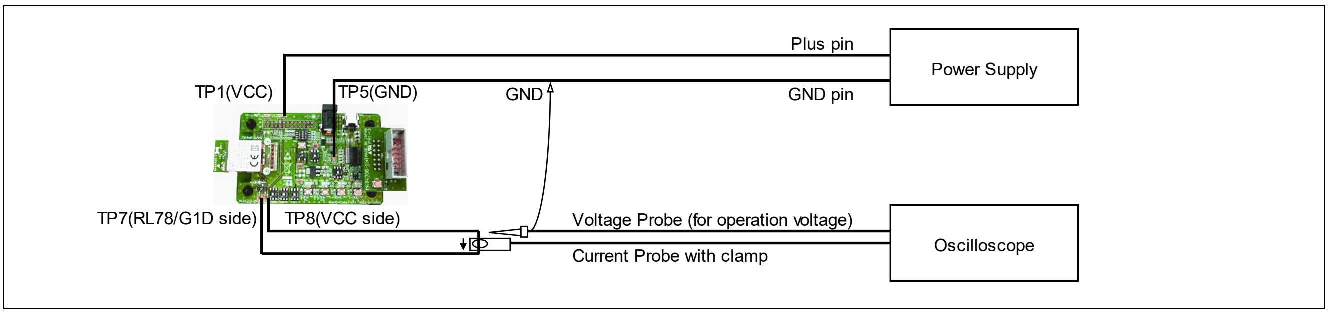


Figure 4-3 Measurement Environment which uses Current Probe

Figure 4-4 shows the measurement environment which uses the combination of shunt resistor and voltage probe as a current detector. In this environment, the resistor is inserted between terminal TP7 and TP8 of the evaluation board, and voltage drop at the resistor is measured by using two voltage probes.

Voltage drop *dV* by the resistor is difference of two voltages measured by individual voltage probe. The current consumption of RL78/G1D is the result of calculating with formula $I=dV / R$, where I is current; dV is voltage drop by the resistor; and R is resistance value.

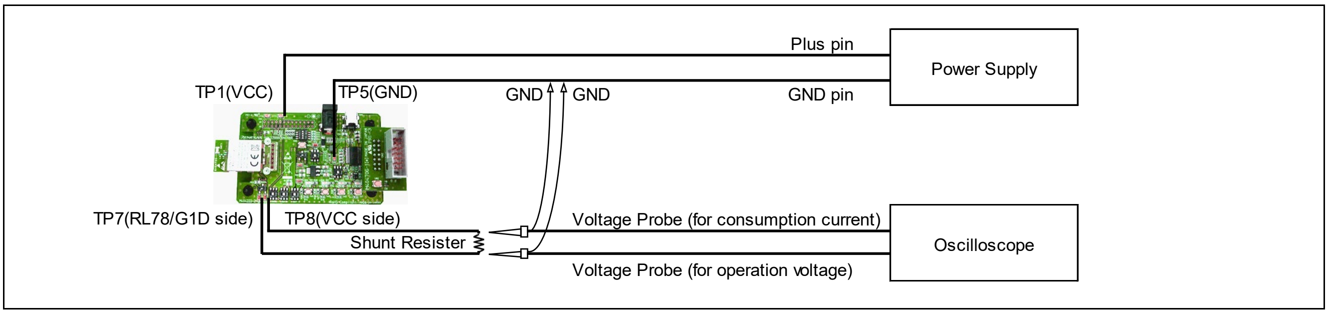


Figure 4-4 Measurement Environment which uses the combination of Shunt Resistor and Voltage Probe

4.6.2 Evaluation Board Setting

Table 4-4 shows the slide switch settings of the evaluation board for current consumption measurement.

Table 4-4 Slide Switch Settings for current consumption measurement

Switch	Setting	Description
SW7	1-2 connected (left)	Power is directly supplied from external power source (not via a regulator). If it is necessary to supply from USB, set 2-3 connected (right).
SW8	1-2 connected (left)	Power is supplied from TP1, TP5 pin or AC Power Supply Adapter. If it is necessary to supply from USB, set 2-3 connected (right).
SW9	1-2 connected (left)	Connect to an external extension interface.
SW10	2-3 connected (right)	The power supply line is left open.
SW11	2-3 connected (right) (default setting)	Power is supplied from a source other than the E1 debugger.
SW12	2-3 connected (right) (default setting)	(Fixed).
SW13	2-3 connected (right)	USB interface is disconnected.

4.6.3 Measurement Procedure

Current consumption measurement procedures are described in below steps. Note that the procedure is reference for only measuring current consumption of Beacon Application with default setting.

Regarding to the details of how to set each equipment settings, refer to the respective manuals.

(1) **Measuring Current Consumption in Periodic Packet Transmission**

1. Start supplying power and start Beacon Application.
2. Set below settings to Oscilloscope by referring to **Figure 4-5**.
   - Capture Trigger : about 0.5mA in current consumption
   - Current Measurement Range : about 10mA
   - Measurement Period : about 10msec from capture trigger
3. Start measuring by Oscilloscope by detecting the current of periodic transmitting.

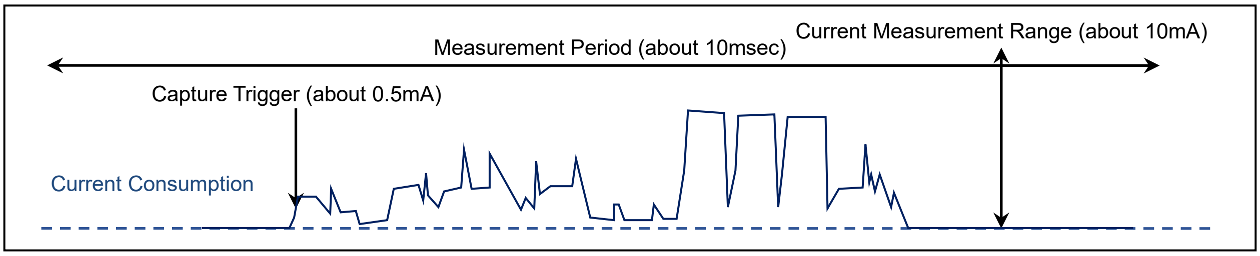


Figure 4-5 Measuring Current Consumption in Periodic Packet Transmission

5. Specification

5.1 Beacon Application

5.1.1 Non-connectable Advertising

Beacon Application loads the Advertising Information and the Advertising Data from system configuration, which stored in Code Flash memory. Then starts transmitting Non-connectable Undirected Advertising packet for broadcasting information. Peer device, like a Smart Phone, receives Advertising packet and provides each service related to the Advertising data. When request to exit the application, it stops Advertising and stops supplying power to RF unit.

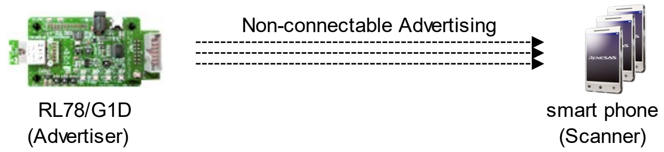


Figure 5-1 Non-connectable Advertising

Regarding to the state transition of Beacon Application and the sequence of Non-connectable Advertising, refer to subsection 8.1.1 "Beacon Application" and subsection 8.2.1(1) "Initializing & Advertising & RF Powerdown Sequence" in this document respectively.

Regarding to the details about system configuration, refer to subsection 5.4.1 "Accessing to Code Flash memory" in this document.

When only Tx is enabled as a RF Operation of Beacon Stack, Beacon Application transmits Non-connectable Undirected Advertising packet.

Table 5-1 shows the default Advertising configuration of Beacon Application.

Table 5-1 the Advertising configuration of Beacon Application when only Tx is enabled

Advertiser Address	Public Device Address 12:34:56:78:9A:B0	
Advertising Type	Non-connectable Undirected Advertising (ADV_NONCONN_IND)	
Advertising Interval	100msec	
Advertising Interval Delay	add random delay to Advertising interval	
Advertising Channel Map	All channels (37,38,39ch)	
Advertising Loop Count	transmitting indefinitely	
Advertising Transmit Power	0dBm at ANT pin of RL78/G1D	
Advertising Data Count	the number of Advertising Data is 1	
Advertising Data [0]~[9]	Advertising Data[0] (ADV_NONCONN_IND payload data)	
	Length	2byte
	AD Type	<<Flags>> (0x01)
	AD Data	LE General Discoverable Mode (bit1) BR/EDR Not Supported (bit2)
	Length	3byte
	AD Type	<<Complete List of 16-bit Service Class UUIDs>> (0x03)
	AD Data	Eddystone (0xFEAA)
	Length	19byte
	AD Type	<<Service Data>> (0x16)
	AD Data	Eddystone-URL: https://goo.gl/5wKkRK
	Advertising Data[1] to [9] are empty	
Advertising Event Permission	notify All Advertising Event	
Use White List	-	

When both Tx and Rx are enabled as a RF Operation of Beacon Stack, Beacon Application transmits Scannable Undirected Advertising packet. And If receive Scan Request packet, Beacon Application transmits Scan Response packet.

Regarding to the setting of RF Operation, refer to subsection 6.1.2 "RF Operation".
Table 5-2 shows Advertising configuration of Beacon Application when both Tx and Rx are enabled.

Table 5-2 the Advertising configuration of Beacon Application when both Tx and Rx are enabled

Advertiser Address	Public Device Address 12:34:56:78:9A:B0	
Advertising Type	Scannable Undirected Advertising (ADV_SCAN_IND)	
Advertising Interval	100msec	
Advertising Interval Delay	add random delay to Advertising interval	
Advertising Channel Map	All channels (37,38,39ch)	
Advertising Loop Count	transmitting indefinitely	
Advertising Transmit Power	0dBm at ANT pin of RL78/G1D	
Advertising Data Count	the number of Advertising Data is 2	
Advertising Data [0]～[9]	Advertising Data[0] (ADV_SCAN_IND payload data)	
	Length	2byte
	AD Type	<<Flags>> (0x01)
	AD Data	LE General Discoverable Mode (bit1) BR/EDR Not Supported (bit2)
	Length	3byte
	AD Type	<<Complete List of 16-bit Service Class UUIDs>> (0x03)
	AD Data	Eddystone (0xFEAA)
	Length	19byte
	AD Type	<<Service Data>> (0x16)
	AD Data	Eddystone-URL: https://goo.gl/5wKkRK
	Advertising Data[1] (SCAN_RSP payload data)	
	Length	24byte
	AD Type	<<Complete Local Name>> (0x09)
	AD Data	"Renesas RL78/G1D Beacon"
	Advertising Data[2] to [9] are empty	
Advertising Event Permission	notify All Advertising event	
Use White List	not use White List	

Regarding to the specification of Eddystone and Eddystone-URL, refer to below website.

- Specification for Eddystone, an open beacon format from Google
  https://github.com/google/eddystone
- Specification for Eddystone, an open beacon format from Google - Eddystone-URL
  https://github.com/google/eddystone/tree/master/eddystone-url

5.2 Connect Application

5.2.1 Connectable Advertising

First, Connect Application starts transmitting Connectable undirected advertising packet for establishing connection. Then peer device, like a smart phone, receives Advertising packet and establishes a connection with RL78/G1D device by transmitting Connection Request packet.

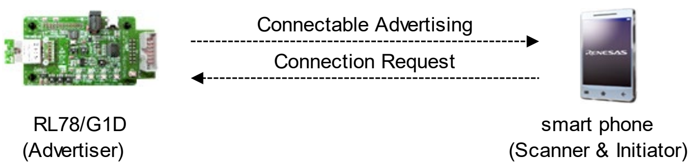


Figure 5-2 Connectable Advertising

Regarding to the state transition of Connect Application and the sequence of Connectable Advertising, refer to subsection 8.1.2 "Connect Application" and subsection 8.2.2(1) "Initializing & Advertising & Slave Connection (Configurations) Sequence" in this document respectively.

Table 5-3 shows the default advertising configuration of Connect Application.

Table 5-3 the default advertising configuration of Connect Application

Advertiser Address	Public Device Address 12:34:56:78:9A:B0	
Advertising Type	Connectable undirected Advertising(ADV_IND)	
Advertising Interval Min	30msec	
Advertising Interval Max	30msec	
Advertising Channel Map	All channels (37,38,39ch)	
Advertising Data	Length	2byte
	AD Type	<<Flags>> (0x01)
	AD Data	LE General Discoverable Mode (bit1) BR/EDR Not Supported (bit2)
	Length	24byte
	AD Type	<<Complete Local Name>> (0x09)
	AD Data	"RL78/G1D Beacon Updater"
Scan Response Data	Empty	

5.2.2 Pairing / Start Encryption

After establishing connection, Connect Application executes paring sequence or starts encryption sequence by Master's request. When the pairing sequence is completed, the subsequent transmitted data packets are encrypted.

The pairing sequence is executed in first connection to the peer device. Device exchanges the pairing information and generates the encryption key. For encrypting data in subsequent connection, need the generated encryption key. Thus, the application stores the encryption key into the Data Flash memory, by using Data Flash Library.

The start encryption sequence is executed in the connection to the peer device, which has been already executed the pairing sequence before connection. The application loads encryption key from Data Flash memory after that start to encrypt data packets.

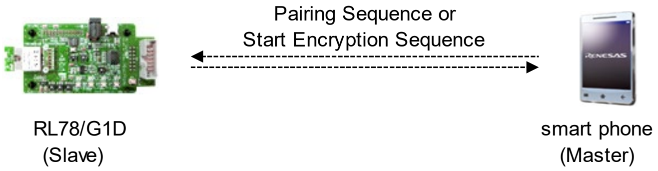


Figure 5-3 Pairing / Start Encryption

Regarding to the state transition of Connect Application and the sequence of the pairing / the start encryption, refer to subsection 8.1.2 "Connect Application", subsection 8.2.2(2) "Slave Connection (Pairing) Sequence", and subsection 8.2.2(3) "Slave Connection (Start Encryption) Sequence" in this document respectively.

Table 5-4 shows the default pairing configuration of Connect Application.

Table 5-4 the default pairing configuration of Connect Application

Bonding	Bondable Mode
Security Mode	Unauthenticated pairing with encryption
Pairing Method	Just Works
IO Capabilities	No Input No Output
OOB Flag	OOB Data not present
Authentication Requirements	No MITM Bonding
Encryption Key Size	128bit
Initiator Key Distribution	None
Responder Key Distribution	Encryption key

5.2.3 Profile Communication

In the connection, Connect Application communicates data according to the Custom Profile. First, GATT Client device gets the service composition and the characteristic composition of GATT Server device by Primary Service Discovery, Characteristic Discovery and Characteristic Descriptor Discovery.

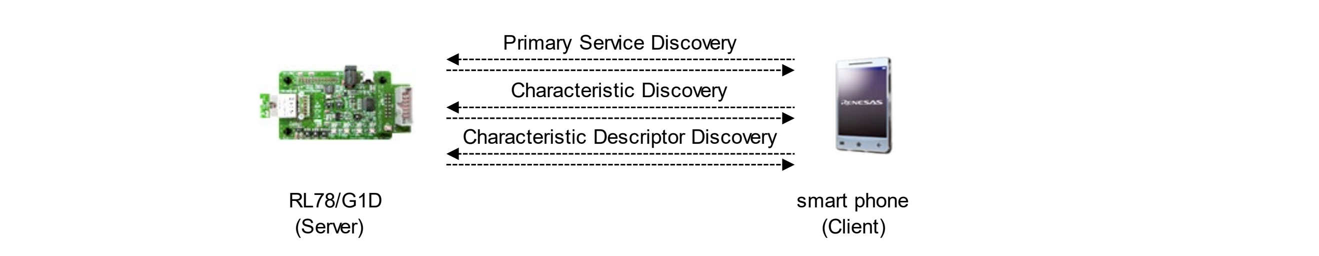


Figure 5-4 GATT Discovery

Then, GATT Client device reads and writes the characteristic value of GATT Server device by Characteristic Value Read and Characteristic Value Write respectively.

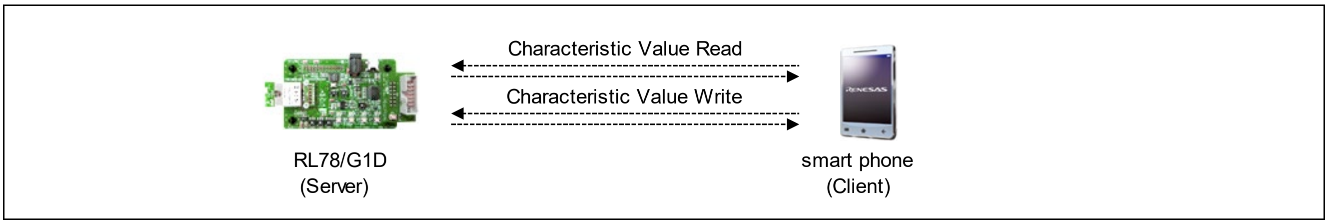


Figure 5-5 GATT Read / Write

Regarding to the state transition of Connect Application and the sequence of GATT Access, refer to subsection 8.1.2 "Connect Application" and subsection 8.2.2(4) "Slave Connection (GATT Access) Sequence" in this document respectively.

Regarding to implementing and changing Custom Profile, refer to subsection 6.2.6 "Custom Profile" in this document.

The specification of Custom Profile implemented in the Sample Program is as shown below.

- Custom Profile Role
  - Role of Beacon device is GATT Server.
  - Role of device which connects to beacon device is GATT Client.
  - GATT Server has Custom Service.
  - GATT Client gets Characteristic Value of Custom Service by Characteristic Value Read, and updates Characteristic Value of Custom Service by Characteristic Value Write.
  - GATT Server does not inform data by Notification and Indication.

- Custom Profile Scenarios
  - GATT Client device updates Advertising Information and Advertising Data for Beacon Stack in beacon device by writing Characteristic Value of Custom Profile.
  - Advertising information and Advertising data is stored by Code Flash memory of beacon device.
  - GATT Client device gets the number of updating Code Flash memory and Data Flash memory in beacon device by reading Characteristic Value of Custom Profile.

Figure 5-6 shows the default Custom Profile Role of Connect Application.

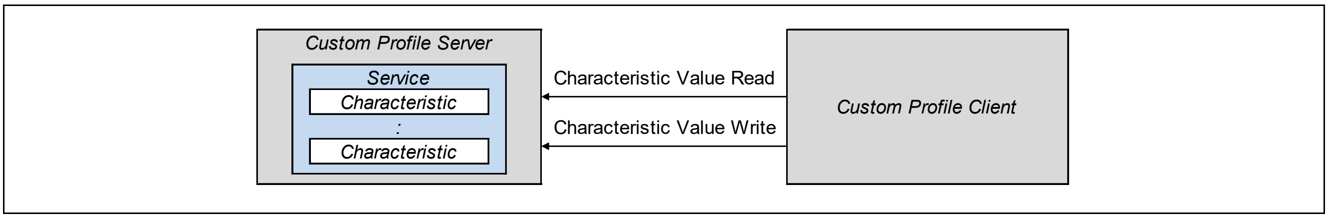


Figure 5-6 Custom Profile Role

Table 5-5 shows the default Custom Service specification of Connect Application.

Table 5-5 the default Custom Service specification of Connect Application

	Attribute Handle	Attribute Type	Attribute Value
<<Custom Service>>			
	0x000C	Primary Service Declaration (0x2800)	UUID: A7660001-4B1E-4D6E-91C4-997BA9B6FC07
<<Characteristic : Advertising Information>>			
	0x000D	Characteristic Declaration (0x2803)	Properties: Read, Write (0x0A) Value Handle: 0x000E UUID: A7660002-4B1E-4D6E-91C4-997BA9B6FC07
	0x000E	Advertising Information	Advertising Information structure defined by Beacon Stack API(18byte)
<<Characteristic : Advertising Data>>			
	0x000F	Characteristic Declaration (0x2803)	Properties: Read, Write (0x0A) Value Handle: 0x0010 UUID: A7660003-4B1E-4D6E-91C4-997BA9B6FC07
	0x0010	Advertising Data	Advertising Data structure defined by Beacon Stack API (32byte)
<<Characteristic : Scan Response Data>>			
	0x0011	Characteristic Declaration (0x2803)	Properties: Read, Write (0x0A) Value Handle: 0x0012 UUID: A7660006-4B1E-4D6E-91C4-997BA9B6FC07
	0x0012	Scan Response Data	Advertising Data structure defined by Beacon Stack API (32byte)
<<Characteristic : Code Flash Memory Updated Count>>			
	0x0013	Characteristic Declaration (0x2803)	Properties: Read (0x02) Value Handle: 0x0014 UUID: A7660004-4B1E-4D6E-91C4-997BA9B6FC07
	0x0014	Code Flash Memory Updated Count	Code Flash Memory Updated Count (2byte) Byte Order : Least Significant Byte First
<<Characteristic : Data Flash Memory Updated Count>>			
	0x0015	Characteristic Declaration (0x2803)	Properties: Read (0x02) Value Handle: 0x0016 UUID: A7660005-4B1E-4d6e-91C4-997BA9B6FC07
	0x0016	Data Flash Memory Updated Count	Data Flash Memory Updated Count (2byte) Byte Order : Least Significant Byte First

Regarding to the specification of Advertising Information structure and Advertising Data structure, refer to chapter 4 "API" in RL78/G1D Beacon Stack User's Manual (R01UW0171).

5.3 DTM Application

5.3.1 Direct Test Mode

DTM Application enables UART for communicating for RF Test commands and events. By receiving RF Test command from Tester, the application executes RF Transmitter Test and RF Receiver Test, then after the application transmits RF Test events back to Tester.

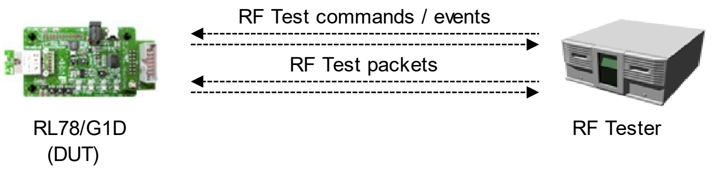


Figure 5-7 Direct Test Mode

Regarding to the state transition of DTM Application and the sequence of Direct Test Mode, refer to subsection 8.1.3 "DTM Application" and subsection 8.2.3(1) "Initializing & Transmitter Test & Receiver Test Sequence" in this document respectively.

Table 5-6 shows RF Test Commands for executing Direct Test mode.

Table 5-6 RF Test Commands

RF Test Command	Parameters
LE_RESET	Control (ignored)
LE_RECEIVER_TEST	Frequency, Length, Packet Type
LE_TRANSMITTER_TEST	Frequency, Length, Packet Type
LE_TEST_END	None

Table 5-7 shows RF Test Events for returning the result of Direct Test Mode.

Table 5-7 RF Test Events

RF Test Event	Parameters
LE_TEST_STATUS	Status(Success / Error)
LE_TEST_PACKET_REPORT	Packet Count

Regarding to the specification of Direct Test Mode, refer to [Vol. 6, Part F] Section 3.3, Bluetooth Core Specification v4.2.

5.4 Accessing to Flash memory

5.4.1 Accessing to Code Flash memory

Both Beacon Application and Connect Application uses a part of Code Flash memory, which is located outside of firmware, to store parameters as system configuration. System configuration is used to store parameters, which need to be different from each individual device.

Beacon Application only loads Device Address, Type, Advertising Information and Advertising Data from system configuration. Then start Advertising.

Connect Application loads Device Address, Type and Device Name from system configuration, and configures them to Protocol Stack. When Connect Application receives new Advertising Information or new Advertising Data from peer connected device, the application updates system configuration in Code Flash memory.

Table 5-8 shows the specification of system configuration in Code Flash memory. Regarding to the location of system configuration, refer to section 5.10 "Address Map" in this document.

Table 5-8 System Configuration in Code Flash memory

offset	data		size	read (YES:read, NO:not read)	write (YES:write, NO:not write)
0x00	Device Address (RBLE_BD_ADDR structure)		6 byte	YES	NO
0x06	Device Address Type 0x00: public, 0x01: random (uint8_t type)		1 byte	YES	NO
0x07	(reserved)		1 byte	NO	NO
0x08	Device Name (device_name structure)		66 byte		
		namelen	1 byte	YES	NO
		name	65 byte	YES	NO
0x4A	Advertising Information (RBLE_ADV_INFO structure)		18 byte		
		interval	2 byte	YES	YES
		delay	1 byte	YES	YES
		ch_map	1 byte	YES	YES
		loop_cnt	1 byte	YES	YES
		tx_pwr	1 byte	YES	YES
		own_addr	6 byte	NO	NO
		own_addr_type	1 byte	NO	NO
		data_cnt	1 byte	NO	NO
		data	2 byte	NO	NO
		evt_permit	1 byte	NO	NO
		use_wl	1 byte	NO	NO
0x5C	Non-connectable Undirected Advertising packet data (RBLE_ADV_DATA structure)		32 byte		
		len	1 byte	YES	YES
		data	31 byte	YES	YES
0x7C	Scannable Undirected Advertising packet data (RBLE_ADV_DATA structure)		32 byte		
		len	1 byte	YES	YES
		data	31 byte	YES	YES
0x9C	Scan Response packet data (RBLE_ADV_DATA structure)		32 byte		
		len	1 byte	YES	YES

		data		31 byte	YES	YES
0xAC	-		-		-	

Connect Application uses Code Flash Library for updating the Code Flash memory. Regarding to the details about Code Flash Library, refer to RL78 Family Flash Self-Programing Library Type01 User's Manual (R01US0050).

5.4.2 Accessing to Data Flash memory

Connect Application uses Data Flash memory to save parameters, which needs to be stored when power shutdown. Thus, by using Data Flash Library, Connect Application reads and writes data in Data Flash memory.

Table 5-9 shows the stored data in Data Flash memory.

Table 5-9 Stored Data in Data Flash memory

Data ID	data	size	reading	writing
0x02	Paring Information (con_pairing_t structure)		In starting Connect Application, copy to the variable in RAM.	When disconnection in Connect Application, if pairing information is updated by connecting, write to Data Flash memory.
	peer device address	6 byte		
	peer device address type	1 byte		
	security status	1 byte		
	encryption key information			
	EDIV (Encrypted Diversifier)	2 byte		
	Random Number	8 byte		
	LTK (Long Term Key)	16 byte		
0x03	Flash memory updating count (con_flashcnt_t structure)			When disconnection in Connect Application, if either Pairing information, Advertising information, or Advertising Data is updated, write flash memory.
	Code Flash memory updating count	2 byte		
	Data Flash memory updating count	2 byte		

Connect Application uses Data Flash Library for reading and writing Data Flash memory. Regarding to the details about Data Flash Library, refer to RL78 Family EEPROM Emulation Library Pack02 User's Manual (R01US0068).

Note: Specification of Stored Data in Flash memory is changed from the specification of Rev1.00. When evaluate this Sample Program Rev.1.10 by using RL78/G1D Evaluation Board which was used for evaluating the Sample Program Rev.1.00, it is necessary to erase Data Flash memory by using Renesas Flash Programmer (RFP), etc.

Regarding to erasing by using RFP, refer to subsection 2.3.2 "[Operation Setting] Tabbed Page" in Renesas Flash Programmer V3.02 Flash memory programming software User's Manual (R20UT3841).

5.5 Supporting Status of Protocol Stack Functions

This subsection shows supporting status of functions implemented in Protocol Stack for the Sample Program. The supported functions are listed in below tables.

Table 5-10 Software Configuration

Software Configuration	Status	Description
Embedded configuration	supported	-
Modem configuration	not supported	application for RSCIP is not implemented

Table 5-11 GAP Role

GAP Role	Status	Description
Broadcaster	supported	-
Observer	not supported	application for Observer is not implemented
Central	not supported	application for Central is not implemented
Peripheral	supported	-

Table 5-12 Protocol Stack Layer

Protocol Stack Layer	Status	Description
LL	supported	-
GAP	supported	-
SM	supported	-
GATT	supported	-
VS	supported	-
Adopted Profile (Note1)	not supported	application for Adopted Profile is not implemented
Custom Profile (Note2)	supported	-

Note1: Adopted Profile

It is a GATT-based profile adopted by Bluetooth SIG.

Note2: Custom Profile

It is a profile defined uniquely by user.

Regarding to the details, refer to below website.

https://www.bluetooth.com/specifications/gatt

Table 5-13 Optional Function

Optional Function	Status	Description
RWKE	supported	-
SLEEP	supported	-
RSCIP	not supported	application for RSCIP is not implemented
DTM 2Wire-UART	supported	-
Adaptable	not supported	application for Adaptable is not implemented
Peak current notification	not supported	application for peak current notification is not implemented
FW update	not supported	application for FW update is not implemented
HCI packet monitor	supported	-
DataFlash read / write	supported	-
CodeFlash write	supported	-

Table 5-14 Hardware Configuration

HW Configuration	Status	Description
RF high-speed clock output	not supported	application for RF clock output is not implemented
external clock MCU operation	not supported	not supported by Beacon Stack
RF External Power Amplifier	not supported	not supported by Beacon Stack

5.6 Hardware Resources used

Table 5-15 shows the hardware resources used by the Sample Program with default settings.

Table 5-15 Hardware Resource used

RL78/G1D MCU Unit		
	Clock generator	Common • use 8MHz from High-speed On-chip Oscillator as MCU main system clock
		Common • not use XT1 oscillator (use RF on-chip oscillator for generating RF slow clock)
	Clock output/buzzer output	Common • not output clock generated XT1 oscillation from PCLBUZ0 pin
	Timer Array Unit	Beacon Stack • use TM00, and set operation clock CK00 to 1MHz
	Serial array unit	Beacon Stack and BLE Protocol Stack • use CSI21 DTM Application • use UART0
	DMA controller	Beacon Stack and BLE Protocol Stack • use DMA2 and DMA3 DTM Application • use DMA0 and DMA1
	Interrupt	Beacon Stack • use INTRF, INTDMA2, INTDMA3, and INTTM00 Beacon Application and Connect Application • use INTP5 DTM Application • use INTDMA0, INTDMA1, INTSR0, INTSRE0, and INTST0
	Port	Common • use P10, for DIP switch SW6 position-1 input on the evaluation board • use P16, for switch SW2 input on the evaluation board • use P60, for controlling LED4 on the evaluation board • use P120, P147, P03, and P60, for controlling LED1, 2, 3, and 4 on the evaluation board
RL78/G1D RF Unit		
	DC-DC Converter	use RF on-chip DC-DC converter
	Oscillator for RF slow clock	use RF on-chip oscillator
	GPIO0	Input mode (unchangeable)
	GPIO1	Input mode (unchangeable)
	GPIO2	Input mode, RF high-speed clock output is disabled (unchangeable)
	GPIO3	When use Oscillator for RF slow clock: Output-low mode When no use Oscillator for RF slow clock: Input mode for RF slow clock input
RL78/G1D Evaluation Board		
	Input functions	Common • use DIP switch SW6 position-1, for selecting application Beacon Application and Connect Application • use push switch SW2, for switching application
	Display	Beacon Application and DTM Application • use LED4, for indicating that the Sample Program is started Connect Application • use LED1, for indicating that data is encrypted • use LED2, for indicating that connection is established • use LED3, for indicating that Connect Application is started • use LED4, for indicating that the Sample Program is started

5.7 Compiler

The library of Beacon Stack is generated by below compiler. It is necessary to use CC-RL compiler for developing application which uses Beacon Stack.

Compiler : Renesas CC-RL V1.04.00

5.8 Memory Model

The memory model of Beacon Stack is medium model. It is necessary to set below option in the compile option of application which uses Beacon Stack.

Memory Model : -memory_model=medium

5.9 Program Size

Table 5-16 shows the total memory usage in the Sample Program.

Target Device : R5F11AGJ
Compiler : Renesas CC-RL V1.04.00
Compile Configuration : default configuration of Sample Program released

Table 5-16 Sample Program Total Program Size

ROM SIZE	119,998 byte PROGRAM SECTION + ROMDATA SECTION
RAM SIZE	10,439 byte RAMDATA SECTION (not included stack memory which program consumes for calling functions and allocating auto variables)

Regarding to the section specification, refer to chapter 6 "SECTION SPECIFICATIONS" in CC-RL Compiler User's Manual (R20UT3123).

5.10 Address Map

Figure 5-8 shows the address map of the Sample Program for RL78/G1D (R5F11AGG) device.

Under-lined values are different for R5F11AGH and R5F11AGJ.

Address	Area Size	Section		Section Name
	131,072byte	Code Flash memory		
0x00000	128byte	Vector table area		.vect
0x00080	64byte	CALLT table area		.callt0
0x000C0	4byte	Option byte area		.option_byte
0x000C4	10byte	Security ID setting area		.security_id
0x000CE	129,842byte	Program area (below sections are described in no particular order)		
			OCD monitor	.monitor1, .monitor2
			Startup	BOOT0_TEXT
			Runtime library	.RLIB
			Standard library	.SLIB
			CodeFlash library	FSL_FCD, FSL_RCD, FSL_BCD, FSL_BECD
			DataFlash library	EEL_CODE, FDL_CODE
			Beacon Stack	BCN_CONST, BCN_TEXT
			Protocol Stack	RBL_CNST_n, RBL_CODE_n, RBL_CODE_f, HST_CNST_n, HST_CODE_n, HST_CODE_f, CNT_CNST_n, CNT_CODE_n, CNT_CODE_f
			Applications	.const, .constf, .data, .text, .textf
			Unused area	-
0x1FC00	156byte	System Configuration area		
0x1FC9C		Unused area		
0x20000		Reserved		
0xF0000	2048byte	Special function register(2nd SFR)		
0xF0800		Reserved		
0xF1000	8192byte	DataFlash memory		
0xF3000	40,704byte	Mirror area		
0xFCF00	12,064byte	RAM area		
		Program Resource area (below sections are described in no particular order)		
			Applications	.bss, .dataR
			Protocol Stack	RBL_DATA_n, HST_DATA_n , CNT_DATA_n
			Beacon Stack	BCN_BSS
			DataFlash library	EEL_SDAT , FDL_SDAT
			Unused area	-
			Stack area	-
0xFFEE0	32byte	General-purpose register		
0xFFF00	256byte	Special function register(SFR)		
0xFFFFF				

Figure 5-8 Address Map (R5F11AGG)

Figure 5-9 shows the address map of the Sample Program for RL78/G1D (R5F11AGH) device.

Under-lined values are different for R5F11AGG and R5F11AGJ.

Address	Area Size	Section	Section Name
	196,608byte	Code Flash memory	
0x00000	128byte	Vector table area	.vect
0x00080	64byte	CALLT table area	.callt0
0x000C0	4byte	Option byte area	.option_byte
0x000C4	10byte	Security ID setting area	.security_id
0x000CE	195,378byte	Program area (below sections are described in no particular order)	
		OCD monitor	.monitor1, .monitor2
		Startup	BOOT0_TEXT
		Runtime library	.RLIB
		Standard library	.SLIB
		CodeFlash library	FSL_FCD, FSL_RCD, FSL_BCD, FSL_BECD
		DataFlash library	EEL_CODE, FDL_CODE
		Beacon Stack	BCN_CONST, BCN_TEXT
		Protocol Stack	RBL_CNST_n, RBL_CODE_n, RBL_CODE_f, HST_CNST_n, HST_CODE_n, HST_CODE_f, CNT_CNST_n, CNT_CODE_n, CNT_CODE_f
		Applications	.const, .constf, .data, .text, .textf
		Unused area	-
0x3F400	156byte	System Configuration area	
0x3F49C		Unused area	
0x40000		Reserved	
0xF0000	2048byte	Special function register(2nd SFR)	
0xF0800		Reserved	
0xF1000	8192byte	DataFlash memory	
0xF3000	36,608byte	Mirror area	
0xFBF00	16,160byte	RAM area	
		Program Resource area (below sections are described in no particular order)	
		Applications	.bss, .dataR
		Protocol Stack	RBL_DATA_n, HST_DATA_n , CNT_DATA_n
		Beacon Stack	BCN_BSS
		DataFlash library	EEL_SDAT , FDL_SDAT
		Unused area	-
		Stack area	-
0xFFEE0	32byte	General-purpose register	
0xFFF00	256byte	Special function register(SFR)	
0xFFFFF			

Figure 5-9 Address Map (R5F11AGH)

Figure 5-10 shows the address map of the Sample Program for RL78/G1D (R5F11AGJ) device.

Under-lined values are different for R5F11AGG and R5F11AGH.

Address	Area Size	Section	Section Name
	262,144byte	Code Flash memory	
0x00000	128byte	Vector table area	.vect
0x00080	64byte	CALLT table area	.callt0
0x000C0	4byte	Option byte area	.option_byte
0x000C4	10byte	Security ID setting area	.security_id
0x000CE	258,866byte	Program area (below sections are described in no particular order)	
		OCD monitor	.monitor1, .monitor2
		Startup	BOOT0_TEXT
		Runtime library	.RLIB
		Standard library	.SLIB
		CodeFlash library	FSL_FCD, FSL_RCD, FSL_BCD, FSL_BECD
		DataFlash library	EEL_CODE, FDL_CODE
		Beacon Stack	BCN_CONST, BCN_TEXT
		Protocol Stack	RBL_CNST_n, RBL_CODE_n, RBL_CODE_f, HST_CNST_n, HST_CODE_n, HST_CODE_f, CNT_CNST_n, CNT_CODE_n, CNT_CODE_f
		Applications	.const, .constf, .data, .text, .textf
		Unused area	-
0x3F400	156byte	System Configuration area	
0x3F49C		Unused area	
0x3F800	512byte	Reserved area (RL78/G1D Module only)	
0x3FC00	6byte	User Information area	
0x3FC06		Unused area	
0x40000		Reserved	
0xF0000	2048byte	Special function register(2nd SFR)	
0xF0800		Reserved	
0xF1000	8192byte	DataFlash memory	
0xF3000	32,512byte	Mirror area	
0xFAF00	1024byte	Self RAM area (R5F11AGJ only)	
0xFB300	20,447byte	RAM area	
		Program Resource area (below sections are described in no particular order)	
		Applications	.bss, .dataR
		Protocol Stack	RBL_DATA_n, HST_DATA_n , CNT_DATA_n
		Beacon Stack	BCN_BSS
		DataFlash library	EEL_SDAT , FDL_SDAT
		Unused area	-
		Stack area	-
0xFFEE0	32byte	General-purpose register	
0xFFF00	256byte	Special function register(SFR)	
0xFFFFF			

Figure 5-10 Address Map (R5F11AGJ)

6. Configuration

This chapter describes the configurations for hardware and application of the Sample Program.

6.1 Hardware configuration

For using Protocol Stack and Beacon Stack, major hardware configurations are arranged to macro definitions in r_config.h. Regarding to the details about macro definitions, refer to following subsections.

Project_Source\application\src\r_config.h, line 34-86

```
34:    /*
35:     * CONFIGURATIONS (NEED TO CHANGE BELOW DEFINES AS NECESSARY)
36:     **************************************************************************************
37:     */
38:    /* MCU Main System Clock (either clock frequency of 4MHz,8MHz,16MHz,32MHz)              */
39:    /*  Note: It is necessary to set Option Bytes Value at Device Setting of Linker Option  */
40:    #define MCU_HOCO_CLK            (8)
41:
42:    /* RF Operation (0:enable both Tx and Rx, 1:enable Tx only) */
43:    /*   Note: This configuration is only for Beacon Stack      */
44:    #define RF_TX_ONLY              (0)
45:
46:    /* RF DC-DC Converter (0:disable DC-DC, 1:enable DC-DC) */
47:    #define RF_DCDC_EN              (1)
48:
49:    /* RF Slow Clock Source (0:RF On-Chip Oscillator, 1:MCU XT1 Oscillator) */
50:    #define RF_SLK_XT1              (0)
51:
52:    /* RF Slow Clock Calibration (0:not execute, 1:execute)                   */
53:    /*   Note: This configuration is only for Beacon Stack                    */
54:    /*       : RF Slow Clock Calibration is only for RF-On_Chip_Oscillator    */
55:    /*       : Protocol Stack always execute RF on chip oscillator calibration */
56:    #define RF_SLK_CAL              (1)
57:
58:    /* RF 32MHz Oscillation Stabilization Time (usec, at least 550usec)     */
59:    /*   Note: This configuration is only for Beacon Stack                 */
60:    /*       : Stabilization Time needs to be optimized for 32MHz resonator */
61:    #define RF_32MHZ_WAIT           (1000)
62:
63:    /* Maximum number of Simultaneous Connections (fixed 1)              */
64:    /*   Note: This configuration is only for BLE Protocol Stack         */
65:    /*       : fixed 1, Connect Application behaves as peripheral device  */
66:    #define MAX_CONNECTION          (1)
67:
68:    /* Packet Monitoring (0:disable Packet Monitor, 1:enable Packet Monitor) */
69:    /*   Note: This configuration is only for BLE Protocol Stack            */
70:    /*       : Packet Monitoring uses UART1 for using output HCI log         */
71:    #define PKTMON_EN               (0)
72:
73:    /* System Configuration Address in CodeFlash memory */
74:    #if   defined(_USE_R5F11AGG)
75:        /* System Configuration is located the last block */
76:        #define SYSCFG_ADDR         (0x1FC00)
77:    #elif defined(_USE_R5F11AGH)
78:        /* System Configuration is located the last block */
79:        #define SYSCFG_ADDR         (0x2FC00)
80:    #elif defined(_USE_R5F11AGJ)
81:        /* System Configuration is located the third last block           */
82:        /* by taking into account the location of RL78/G1D module (RY7011) */
83:        #define SYSCFG_ADDR         (0x3F400)
84:        /* In the case of RL78/G1D Module (RY7011), Device Address is located the last block */
85:        #define MODCFG_ADDR         (0x3FC00)
86:    #endif
```

6.1.1 MCU main system clock frequency

Clock generated by Hi-speed On-chip Oscillator is used as MCU main system clock, and selectable frequency of MCU main system clock is 4, 8, 16 and 32MHz. In the Sample Program, frequency of MCU main system clock is defined by the macro MCU_HOCO_CLK and Option Bytes. The default setting of clock frequency is 8 (MHz).

If changing the frequency of MCU main system clock, change the macro value to one of the values: 4 (MHz), 8 (MHz), 16 (MHz), 32 (MHz).

Project_Source\application\src\r_config.h, line 38-40

```
38:    /* MCU Main System Clock (either clock frequency of 4MHz,8MHz,16MHz,32MHz)            */
39:    /*  Note: It is necessary to set Option Bytes Value at Device Setting of Linker Option  */
40:    #define MCU_HOCO_CLK            (8)
```

Option Bytes is set to the linker option "-user_opt_byte". Regarding to the value of Option Bytes, refer to **Table 6-1**.

Table 6-1 Option Bytes value setting

Option Bytes setting			Clock frequency	Flash Operation Mode
000C0	000C1	000C2		
(any)	(any)	2B	4MHz	low-voltage main mode
		AA	8MHz	low-speed main mode
		E9	16MHz	high-speed main mode
		E8	32MHz	

Regarding to the details about Option Bytes, refer to chapter 25 "OPTION BYTE" in RL78/G1D User's Manual: Hardware (R01UH0515). CPU operation voltage varies with respect to CPU clock frequency. Regarding to the operation voltage, refer to section 30.2 "Operating Voltage" in RL78/G1D User's Manual: Hardware (R01UH0515).

(1) **Using CS+ for CC**

In the case of CS+ for CC about how to set Option Bytes, follow the below steps.

1 Right-click to [CC-RL] of the subproject "R5F11AGJ_BcnCmb" in the project tree.

2 Select [Property] in right click menu.

3 Set the Option Bytes at the [Device]→[User option byte value] of [Link Options] tab.

(2) **Using e^2 studio**

In the case of e^2 studio about how to set Option Bytes, follow the below steps.

1 Right-click to "R5F11AGJ_BcnCmb" project.

2 Select [Renesas Tool Settings] in right click menu.

3 Set the Option Bytes at the [Linker]→[Device]→[User option byte value] of [Tool Settings] tab.

6.1.2 RF Operation

It is possible to select whether to enable both Tx and Rx or only Tx when Beacon Stack works. When enabling only Tx is selected, RF initialization time is shortened. In the Sample Program, whether to enable both Tx and Rx or only Tx is defined by the macro RF_TX_ONLY. The default setting is 0, which means that RF operation is enabled both Tx and Rx.

If need to enable only Tx, change the macro value to 1.

Project_Source\application\src\r_config.h, line 42-44

```
42:    /* RF Operation (0:enable both Tx and Rx, 1:enable Tx only) */
43:    /*   Note: This configuration is only for Beacon Stack      */
44:    #define RF_TX_ONLY              (0)
```

6.1.3 RF on-chip DC-DC converter

In the Sample Program, whether to use RF on-chip DC-DC converter is defined by the macro RF_DCDC_EN. Thus, it is possible to select whether to use RF on-chip DC-DC converter or not. The default setting is 1, which means that RF on-chip DC-DC converter is used.

If not using RF on-chip DC-DC converter, change the macro value to 0.

Project_Source\application\src\r_config.h, line 46-47

```
46:    /* RF DC-DC Converter (0:disable DC-DC, 1:enable DC-DC) */
47:    #define RF_DCDC_EN              (1)
```

6.1.4 RF slow clock source

RF slow clock is needed to RF unit for counting the period, and it is possible to select as a source of RF clock from either RF on-chip oscillator or MCU unit XT1 oscillator. In the Sample Program, RF slow clock source is defined by the macro RF_SLK_XT1. The default setting is 0, which means that RF on-chip oscillator is selected as a source for RF slow clock.

If changing RF slow clock source to MCU unit XT1 oscillator, change the macro value to 1. By changing the macro to 1, clock generated by MCU unit XT1 oscillator is supplied to RF unit via EXSLK_RF pin.

Project_Source\application\src\r_config.h, line 49-50

```
49:    /* RF Slow Clock Source (0:RF On-Chip Oscillator, 1:MCU XT1 Oscillator) */
50:    #define RF_SLK_XT1              (0)
```

6.1.5 RF on-chip oscillator calibration

In the case of using RF on-chip oscillator as a source of RF slow clock, calibrating accuracy of clock generated by RF on-chip oscillator is always executed when Protocol Stack works. But it is possible to select whether to execute calibration or not when Beacon Stack works. In the Sample Program, whether to execute calibration is defined by the macro RF_SLK_CAL. The default setting is 1, which means that the calibration is executed.

Beacon Stack executes calibration only once, just after the end of the transmitting first advertising packet followed by RF initialization. By executing calibration, the accuracy of advertising interval is improved.

If not executing calibration, change the macro value to 0.

Project_Source\application\src\r_config.h, line 52-56

```
52:    /* RF Slow Clock Calibration (0:not execute, 1:execute)                */
53:    /*   Note: This configuration is only for Beacon Stack                 */
54:    /*       : RF Slow Clock Calibration is only for RF-On_Chip_Oscillator */
55:    /*       : Protocol Stack always execute RF on chip oscillator calibration */
56:    #define RF_SLK_CAL              (1)
```

6.1.6 RF base clock oscillation stabilization time

In the Sample Program, the oscillation stabilization time is defined by the macro RF_32MHZ_WAIT. Thus, it is necessary to optimize the oscillation stabilization time of XTAL_RF oscillator for using RF base clock, which is depending on the 32MHz resonator connected to XTAL1_RF and XTAL2_RF pin. The default setting is 1000 (usec) which is suitable for the particular RL78/G1D Evaluation Board.

If changing the oscillation stabilization time, change the macro value to the time, as a minimum 550 (usec).

Project_Source\application\src\r_config.h, line 58-61

```
58:    /* RF 32MHz Oscillation Stabilization Time (usec, at least 550usec)     */
59:    /*   Note: This configuration is only for Beacon Stack                  */
60:    /*       : Stabilization Time needs to be optimized for 32MHz resonator */
61:    #define RF_32MHZ_WAIT            (1000)
```

Regarding to the details about RF base clock generator, refer to subsection 15.3.9 "RF clock generator circuit block" in RL78/G1D User's Manual: Hardware (R01UH0515).

6.1.7 Maximum number of Simultaneous connection

The Sample Program performs as Peripheral Role, so maximum number of simultaneous connection is fixed to 1. In the Sample Program, the maximum number is defined by the macro MAX_CONNECTION. The default setting is 1, which means that only one connection is established to peer Central Role device.

Project_Source\application\src\r_config.h, line 63-66

```
63:    /* Maximum number of Simultaneous Connections (fixed 1)              */
64:    /*   Note: This configuration is only for BLE Protocol Stack         */
65:    /*       : fixed 1, Connect Application behaves as peripheral device  */
66:    #define MAX_CONNECTION           (1)
```

6.1.8 HCI Monitoring

BLE Protocol Stack provides monitoring HCI sequence for debugging purpose. By enabling this function, you can monitor HCI log packet through UART1, and understand how Protocol Stack works. In the Sample Program, whether to enable HCI monitoring or not is defined by the macro PKTMON_EN. The default setting is 0, which means that HCI monitoring is disabled.

If enabling HCI monitoring, change the macro value to 1.

Project_Source\application\src\r_config.h, line 68-71

```
68:    /* Packet Monitoring (0:disable Packet Monitor, 1:enable Packet Monitor) */
69:    /*   Note: This configuration is only for BLE Protocol Stack             */
70:    /*       : Packet Monitoring uses UART1 for using output HCI log         */
71:    #define PKTMON_EN                (0)
```

In order to confirm the contents of HCI log packets, PC and specific application software is needed.

Regarding to the details about how to use the HCI monitoring, refer to chapter 12 "HCI Packet Monitoring Feature" in Bluetooth Low Energy Protocol Stack User's Manual (R01UW0095).

6.1.9 System Configuration Address

In the Code Flash memory, it is possible to store information as system configuration outside of the firmware. By setting each different system configuration for different devices, it is possible to configure the information without rebuilding firmware. For example, this information includes device address, advertising data, and etc. In the Sample Program, the address of system configuration is defined by the macro SYSCFG_ADDR.

If needed to re-assign the address map, change the macro value to new address.

Project_Source\application\src\r_config.h, line 73-86

```
73:    /* System Configuration Address in CodeFlash memory */
74:    #if   defined(_USE_R5F11AGG)
75:        /* System Configuration is located the last block */
76:        #define SYSCFG_ADDR         (0x1FC00)
77:    #elif defined(_USE_R5F11AGH)
78:        /* System Configuration is located the last block */
79:        #define SYSCFG_ADDR         (0x2FC00)
80:    #elif defined(_USE_R5F11AGJ)
81:        /* System Configuration is located the third last block            */
82:        /* by taking into account the location of RL78/G1D module (RY7011) */
83:        #define SYSCFG_ADDR         (0x3F400)
84:        /* In the case of RL78/G1D Module (RY7011), Device Address is located the last block */
85:        #define MODCFG_ADDR         (0x3FC00)
86:    #endif
```

Regarding to the details about System Configuration, refer to subsection 5.4.1 "Accessing to Code Flash memory" in this document.

6.1.10 Switches on RL78/G1D Evaluation Board

For switching application, the Sample Program uses switches on the evaluation board. DIP switch SW6 position-1 switches either Beacon Operation or RF Evaluation Operation. Switch SW2 switches either Beacon Application or Connect Application alternately. In the Sample Program, whether to use switches or not is defined by the macro EVB_SW. The default setting is 1, which means that switches are used.

If need not to use switches on the evaluation board, change the macro value to 0.

Project_Source\application\src\r_main.c, line 51-61

```
51:    /* Switches on RL78/G1D Evaluation Board (0:not to use, 1:use)                     */
52:    /*   Operation:                                                                     */
53:    /*     When use Switches:                                                           */
54:    /*       - After power up, Beacon Application starts running at first               */
55:    /*       - It is possible to switch Beacon Application and Connect Application alternately */
56:    /*     When use no Switches:                                                        */
57:    /*       - After power up, Connect Application starts running at first              */
58:    /*       - If connection is not established within 30sec, Connect Application stops and    */
59:    /*         Beacon Application starts running                                        */
60:    /*       - It is not possible to switch from Beacon Application to Connect Application     */
61:    #define EVABOARD_SWITCH_EN      (1)
```

Figure 6-1 shows application switch operation when the Sample Program uses switches on the evaluation board.

The Sample Program uses Switch SW2 and SW6 position-1 for switching application. When switch SW6 position-1 is ON, the Sample Program executes DTM Application. When switch SW6 position-1 is OFF, the Sample Program executes Beacon Application and Connect Application. After power on, Beacon Application runs at first. Pushing switch SW2 can switch Beacon Application and Connect Application alternately.

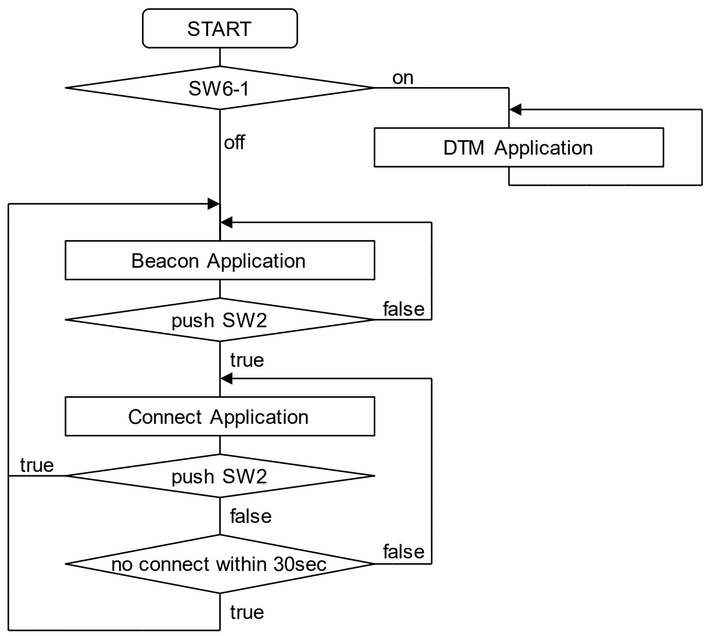


Figure 6-1 Application Switch Operation when the Sample Program uses switches on the evaluation board

Figure 6-2 shows application switch operation when the Sample Program uses no switch on the evaluation board.

The Sample Program executes Beacon Application and Connect Application. After power on, Connect Application runs at first. If connection is not established within 30seconds, Beacon Application runs. To execute Connect Application again, it is necessary to reset the MCU.

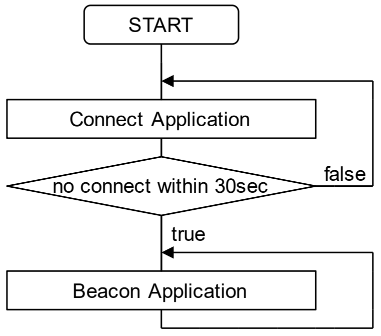


Figure 6-2 Application Switch Operation when the Sample Program uses no switch on the evaluation board

6.2 Application Configuration

6.2.1 System Configuration

System configuration is located outside of firmware in the Code Flash memory. Thus it is possible to write System configuration and firmware at the same time, by using Unique Code Embedding Function of Renesas Flash Programmer.

As a sample, the release package includes unique code file for system configuration. Regarding to the details about system configuration defined by Sample Program, refer to subsection 5.4.1 "Accessing to Code Flash memory" in this document.

RUC_File\r5f11agj_syscfg.ruc, line 1-10

```
 1:  // -----------------------------------------------------------------------------------------------
 2:  // -- System Configuration for RL78/G1D BLE Protocol / Beacon Stack Combination Sample Program --
 3:  // --   Device Part Number : R5F11AGJ                                                            --
 4:  // -----------------------------------------------------------------------------------------------
 5:  format hex
 6:  area user flash
 7:  address 0x3f400
 8:  size 156
 9:  index data
       (a)        (b)        (c)                               (d)
10:  000001 B19A7856341200001 2524C37382F47314420426561636F6E20303100000000000000000000000000000000000000
                                    (d)                                         (e)                 (f)
     00000000000000000000000000000000000000000000000000000000A00001070009B09A7856341200018EC703001B02010
                                    (f)                                              (g)
     60303AAFE1316AAFE10EE02676F6F2E676C2F3764694C5478000000001B180952656E6573617320524C37382F4731442042
               (g)
     6561636F6E303100000000
```

The sample unique code file for R5F11AGJ describes below.

line 1-4 : The lines starting with // are comment line.
line 5 : specifies the format as hexadecimal format
line 6 : specifies the area as User area
line 7 : specifies the address as 0x3F400 (block 253)
line 8 : specifies the size 124 byte
line 9 : declares the unique code data starts at the next line
line10 : specifies the index and unique code

- (a): index of unique code data
- (b): device address (6byte)
- (c): device address type (1byte), padding (1byte)
- (d): device name (66byte)
- (e): advertising information (18byte)
- (f): advertising data (32byte)
- (g): scan response data (32byte)

6.2.2 Kernel Heap Memory Configuration

BLE Protocol Stack includes a kernel called RWKE. The kernel provides below functions.

- Event Management
- Message Communication Management
- Task State Management
- Timer Management
- Memory Management.

When execute kernel functions, the kernel allocates a partial area dynamically from an area predefined as heap memory. If there are a lot of load for the kernel, heap memory may be exhausted. When run short of heap memory by enhancing Connect Application, change the macro APP_HEAP_SIZE value, which defines heap memory size.

By the way, unused area of RAM is used as stack memory. If heap memory is too big, stack-overflow of stack memory may occur.

Project_Source\application\src\connect\resource\r_kernel.c, line 56-65

```
56:    #define APP_HEAP_SIZE           (0)
57:    /* Note: When Kernel Heap size is not enough, it is necessary to incease APP_HEAP_SIZE */
58:
59:    #define BLE_HEAP_SIZE           ((MAX_CONNECTION * 256) + 512           \
60:                                      + BLE_HEAP_CONT                       \
61:                                      + (BLE_HEAP_HOST * MAX_CONNECTION)    \
62:                                      + BLE_DB_SIZE                         \
63:                                      + RBLE_TABLE_SIZE                     \
64:                                      + APP_HEAP_SIZE                       \
65:                                    )
```

Regarding to the details of the kernel, refer to chapter 9 "RWKE" in Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

6.2.3 Advertising Configuration

Default Advertising configuration for Beacon Application is defined in r_beacon.c file.

If needed to change the default Advertising information or default Advertising data, modify the value of RBLE_ADV_INFO structure or RBLE_ADV_DATA structure. Regarding of these structure specification, refer to chapter 4 "API" in RL78/G1D Beacon Stack User's Manual (R01UW0171).

Project_Source\application\src\beacon\r_beacon.c, line 42-92

```
42:    /* Advertising Data Array */
43:    static RBLE_ADV_DATA adv_data[] =
44:    {
45:        /* Advertising Data[0] */
46:        /* Eddystone-URL: https://goo.gl/5wKkRK -> https://www.renesas.com/ */
47:        {
48:            /* Advertising data length */
49:            27,
50:            /* Advertising data <<Flags>> */
51:            0x02, 0x01, 0x06,
52:            /* Advertising data <<Complete List of 16-bit Service Class UUIDs>> */
53:            0x03, 0x03, 0xAA, 0xFE,
54:            /* Advertising data <<Service Data>> */
55:            0x13, 0x16, 0xAA, 0xFE, 0x10, 0xEE, 0x02,
56:            'g', 'o', 'o', '.', 'g', 'l', '/', '5', 'w', 'K', 'k', 'R', 'K'
57:        },
58:        #if !RF_TX_ONLY
59:        /* Scan Response Data[0] */
60:        {
61:            /* Scan Response data length */
62:            25,
63:            /* Scan Response data <<Complete local name>> */
64:            0x18, 0x09,
65:            'R','e','n','e','s','a','s',' ','R','L','7','8','/','G','1','D',
66:            ' ','B','e','a','c','o','n'
67:        },
68:        #endif
69:    };
70:
71:    /* Advertising packet type */
72:    #if RF_TX_ONLY
73:    static const uint8_t adv_type = RBLE_PDU_ADV_NONCONN_IND;
74:    #else
75:    static const uint8_t adv_type = RBLE_PDU_ADV_SCAN_IND;
76:    #endif
77:
78:    /* Advertising Information */
79:    static RBLE_ADV_INFO adv_info =
80:    {
81:        0x00A0,                                        /* Advertising Interval         */
82:        true,                                          /* Advertising Interval Delay   */
83:        RBLE_ADV_ALL_CHANNELS,                         /* Advertising Channel Map      */
84:        0x00,                                          /* Advertising Transfer Count   */
85:        RBLE_TXPW_LV9,                                 /* Advertising Transfer Power   */
86:        { 0xB0, 0x9A, 0x78, 0x56, 0x34, 0x12 },        /* Own Device Address           */
87:        RBLE_ADDR_PUBLIC,                              /* Own Device Address Type      */
88:        sizeof(adv_data) / sizeof(RBLE_ADV_DATA),      /* Advertising Data Count       */
89:        &adv_data[0],                                  /* Advertising Data             */
90:        RBLE_EVT_PERMIT_ADV_ALL,                       /* Advertising Event Permission */
91:        false                                          /* Use White List               */
92:    };
```

If transmitting multiple Advertising data repeatedly, increase the number of RBLE_ADV_DATA structure array.

Example Code for transmitting multiple Advertising data

```
/* Advertising Data Array */
static RBLE_ADV_DATA adv_data[] =
{
    /* Advertising data No.1 */
    {
        /* Advertising data length */
        … ,
        /* Advertising data */
        …
    },
    /* Advertising data No.2 */
    {
        /* Advertising data length */
        … ,
        /* Advertising data */
        …
    }
    :
};
```

Default Advertising configuration for Connect Application is defined in r_connect.c file.

If needed to change the default Advertising information, default Advertising data and default Scan Response data, modify the value of RBLE_ADV_INFO structure. Regarding to the specification of its structure, refer to chapter 5 "Generic Access Profile" in Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

Project_Source\application\src\connect\r_connect.c, line 148-181

```
148:    /* Advertising Information for connection as a slave role */
149:    /*  Note : it is necessary to change configuration corresponds to each use case */
150:    static RBLE_ADV_INFO  broadcast_info =
151:    {
152:        /* Advertising Parameter structure */
153:        {
154:            0x0030,                                     /* Advertising Interval Min        */
155:            0x0030,                                     /* Advertising Interval Max        */
156:            RBLE_GAP_ADV_CONN_UNDIR,                    /* Advertising Type                */
157:            RBLE_ADDR_PUBLIC,                           /* Own Address Type                */
158:            0x00,                                       /* Direct Advertising Address Type */
159:            { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },     /* Direct Advertising Address      */
160:            RBLE_ADV_ALL_CHANNELS,                      /* Advertising Channel Map         */
161:            RBLE_ADV_ALLOW_SCAN_ANY_CON_ANY,            /* Advertising Filter Policy       */
162:            0x00,                                       /*  (reserved)                     */
163:        },
164:        /* Advertising Data structure */
165:        {
166:            /* Advertising data length (max 31byte)  */
167:            3+25,
168:            /* Advertising data <<Flags>> */
169:            2,  0x01, 0x06,
170:            /* Advertising data <<Complete Local Name>> */
171:            24, 0x09,
172:            'R','L','7','8','/','G','1','D',' ','B','e','a','c','o','n',' ','U','p','d','a','t','e','r'
173:        },
174:        /* Scan Response Data structure */
175:        {
176:            /* Scan Response data length (max 31byte) */
177:            0,
178:            /* Scan Response data */
179:            0x00
180:        }
181:    };
```

To specify peer device for establishing connection, set RBLE_ADV_ALLOW_SCAN_ANY_CON_WLST as the advertising filter policy of RBLE_ADV_INFO structure, in order to enable White List. Before starting Advertising, call RBLE_GAP_Add_To_White_List in order to add device address to the White List.

If specifying multiple peer device, call RBLE_GAP_Add_To_White_List for adding each peer device address.

Example Code for adding device address to White List

```
uint9_t wl_cnt;

/* device address list for white list */
static RBLE_DEV_ADDR_INFO wl_info[] =
{
    {RBLE_ADDR_PUBLIC, { 0x01, 0x90, 0x78, 0x56, 0x34, 0x12 }},
    :
};

/* set device address to white list */
/* it is necessary to call repeatedly until all device address of list is added */
if(wl_cnt < (sizeof(wl_info) / sizeof(RBLE_DEV_ADDR_INFO)))
{
    RBLE_GAP_Add_To_White_List(&wl_info[wl_cnt++]);
}
```

Note: White List should not be used for Resolvable Private address, so Resolvable Private address is changed regularly by generating with Identity Resolving Key.

6.2.4 No Connection Timeout Time Configuration

Connect Application finishes if connection is not established from either application start-up or previous disconnection within the no connection timeout time. The timeout time is defined by the macro CON_TIME_OUT. It is possible to set the timeout time in the range of 10 to 299,990 milli-seconds, and in increments of 10milli-seconds. The default setting is 30 seconds.

If changing the timeout time, change the CON_TIME_OUT macro value in the range of 1 to 29,999.

Project_Source\application\src\connect\r_connect.c, line 48-50

```
48:    /* Connect Application Exit Timeout Time (unit: 10msec) */
49:    /* When connection is not established within this time, Connect Application exits. */
50:    #define CON_EXIT_TIME        (3000)
```

To monitor the timeout time, the application uses the kernel timer of BLE Protocol Stack. Regarding to the specification of the kernel timer, refer to section 9.5 "Timer Management" in Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

6.2.5 Paring Configuration

Default pairing configuration is defined in r_connect.c file. The Sample Program executes Just Works pairing method in pairing sequence.

Project_Source\application\src\connect\r_connect.c, line 183-196

```
183:    /* Pairing Information for secure connection */
184:    /*  Note : it is necessary to change configuration corresponds to each secure level requested */
185:    static RBLE_BOND_RESP_PARAM bond_info =
186:    {
187:        0x0000,                                     /* Connection handle           */
188:        RBLE_OK,                                    /* accept or reject bonding    */
189:        RBLE_IO_CAP_NO_INPUT_NO_OUTPUT,             /* IO capabilities             */
190:        RBLE_OOB_AUTH_DATA_NOT_PRESENT,             /* OOB flag                    */
191:        RBLE_AUTH_REQ_NO_MITM_BOND,                 /* Authentication Requirements */
192:        RBLE_SMP_MAX_ENC_SIZE_LEN,                  /* Encryption key size         */
193:        RBLE_KEY_DIST_NONE,                         /* Initiator key distribution  */
194:        RBLE_KEY_DIST_ENCKEY,                       /* Responder key distribution  */
195:        0x00                                        /* Reserved                    */
196:    };
```

When changing pairing method from Just Works to Passkey Entry, it is necessary to provide Passkey from the application. Thus set RBLE_IO_CAP_DISPLAY_ONLY as the IO capabilities and set RBLE_AUTH_REQ_MITM_BOND as the authentication requirements of RBLE_BOND_RESP_PARAM structure.

When executing paring sequence with Passkey Entry, RBLE_SM_TK_REQ_IND event occurs. Therefore, the application is needed to call RBLE_SM_Tk_Req_Resp in order to respond Passkey as the Temporary Key. Below example code generates Passkey with rand function of standard library. After generating, it is necessary to display the passkey to user on the display like LCD screen.

Example Code for responding Temporary Key

```
    uint32_t passkey;
    uint8_t* byteptr = (uint8_t*)&passkey;
    uint8_t  idx;

    /* TK(Temporary Key) buffer */
    RBLE_KEY_VALUE tk =
    {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    };

    /* generate Passkey (range:000,000 - 999,999) */
    passkey  = (uint32_t)rand();
    passkey |= (uint32_t)rand() << 16;
    passkey %= 1000000;

    /* copy Passkey to TK(Temporary Key) buffer */
    for(idx = 0; idx < sizeof(uint32_t); idx++)
    {
        tk.key[RBLE_KEY_LEN - 1 - idx] = byteptr[idx];
    }
    RBLE_SM_Tk_Req_Resp(con_idx, RBLE_OK, &tk);
```

6.2.6 Custom Profile

In order to implement or change Custom Profile, it is necessary to implement below definitions, resources and processing. Detail will be explained in the following.

- definitions:
  - UUIDs of Custom Profile service and Characteristics, for identifying them.
  - Attribute indexes of Custom Profile Service and Characteristics, for operating by Protocol Stack.
  - Attribute handles of Custom Profile Service and Characteristics, for announcing to Client device

- resources:
  - variables of Custom Profile Characteristics, for storing each Characteristic Value
  - descriptors of Custom Profile Service and characteristics, for setting features to Attribute database.
  - descriptors of Custom Profile Characteristics, for setting Characteristic variable to Attribute database.
  - Attribute database, which is accessed by Protocol Stack.

- processing:
  - enabling GATT after stablishing connection, and registering event callback function.
  - updating Characteristic Value by Write Request from client device, and sending Write Response.

Note1: It is not necessary to implement the processing for responding Characteristic Value to the Read Request from Client device. The response is executed by Protocol Stack automatically.

Note2: If it is necessary to inform Characteristic Value in the timing that determined by server device, the processing of sending Notification or Indication is needed. In the Sample Program, processing for Notification or Indication is not implemented. If sending Notification or Indication, use RBLE_GATT_Notify_Request or RBLE_GATT_Indicate_Request of Protocol Stack API respectively.

Regarding to the specification of those functions, refer to subsection 7.2.9 "RBLE_GATT_Notify_Request" and 7.2.10 "RBLE_GATT_Indicate_Request" in Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

(1) **Definitions**

UUIDs are defined in r_profile.h file.

It is possible to generate randomly numbers UUID by using UUIDGEN Linux command. It is necessary to set UUID value in LSB order and UUIDs value are set to each descriptor of Service and Characteristic.

Project_Source\application\src\connect\r_profile.h, line 45-53

```
45:    /* Custom Profile 128bit UUID: A766xxxx-4B1E-4d6e-91C4-997BA9B6FC07 */
46:    /*  Note: randomly numbers UUID can be generated by UUIDGEN linux command */
47:    /*        regarding the specification of UUID, refer to ITU-T X.667       */
48:    #define PRF_UUID_SERVICE
                         {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x01,0x00,0x66,0xA7}
49:    #define PRF_UUID_CHAR_BCNINFO
                         {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x02,0x00,0x66,0xA7}
50:    #define PRF_UUID_CHAR_BCNDATA
                         {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x03,0x00,0x66,0xA7}
51:    #define PRF_UUID_CHAR_CFLCNT
                         {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x04,0x00,0x66,0xA7}
52:    #define PRF_UUID_CHAR_DFLCNT
                         {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x05,0x00,0x66,0xA7}
53:    #define PRF_UUID_CHAR_RSPDATA
                         {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x06,0x00,0x66,0xA7}
```

Attribute indexes and Attribute handles are defined in r_gatt.h file.

It is possible to add or delete Service or Characteristic of Custom Profile.

The Attribute index values are set directly to Attribute database, and Attribute handle values are set to each descriptor of Service and Characteristic.

Project_Source\application\src\connect\resource\r_gatt.h, line 37-113

```
 37:    /* Attribute index */
 38:    enum
 39:    {
 60:        /* offset index for Custom Profiles */
 61:        ATT_IDX_CUSTOM = 0x0200,
 :
 65:        /* Custom Profile Service */
 66:        PRF_IDX_SVC,
 67:        PRF_IDX_BCNINFO_CHAR,
 68:        PRF_IDX_BCNINFO_VAL,
 69:        PRF_IDX_BCNDATA_CHAR,
 70:        PRF_IDX_BCNDATA_VAL,
 71:        PRF_IDX_RSPDATA_CHAR,
 72:        PRF_IDX_RSPDATA_VAL,
 73:        PRF_IDX_CFLCNT_CHAR,
 74:        PRF_IDX_CFLCNT_VAL,
 75:        PRF_IDX_DFLCNT_CHAR,
 76:        PRF_IDX_DFLCNT_VAL,
 79:    };
 80:
 81:    /* Attribute handles */
 82:    enum
 83:    {
 99:        /* Custom Profile Service */
100:        PRF_HDL_SVC                   = 0x000C,
101:        PRF_HDL_BCNINFO_CHAR          = 0x000D,
102:        PRF_HDL_BCNINFO_VAL           = 0x000E,
103:        PRF_HDL_BCNDATA_CHAR          = 0x000F,
104:        PRF_HDL_BCNDATA_VAL           = 0x0010,
105:        PRF_HDL_RSPDATA_CHAR          = 0x0011,
106:        PRF_HDL_RSPDATA_VAL           = 0x0012,
107:        PRF_HDL_CFLCNT_CHAR           = 0x0013,
108:        PRF_HDL_CFLCNT_VAL            = 0x0014,
109:        PRF_HDL_DFLCNT_CHAR           = 0x0015,
110:        PRF_HDL_DFLCNT_VAL            = 0x0016,
113:    };
```

(2) Resources

Variables for storing Characteristic Value are defined in r_profile.c file.

Initial Characteristic Value are set by application. After that, Characteristic Value which has the write permission is updated by Client device. Characteristic value variables are set to each descriptor of Characteristic.

Project_Source\application\src\connect\r_profile.c, line 64-69

```
64:   /* Custom Profile characteristic variables */
65:   PRF_ADV_INFO   prf_bcninfo_val;                   /* Advertising   Information */
66:   PRF_ADV_DATA   prf_bcndata_val;                   /* Advertising   Data        */
67:   PRF_ADV_DATA   prf_rspdata_val;                   /* Scan Response Data        */
68:   uint16_t       prf_cflcnt_val;                    /* Code Flash Updated Count  */
69:   uint16_t       prf_dflcnt_val;                    /* Data Flash Updated Count  */
```

Descriptors of Custom Profile Service and Characteristic are defined in r_gatt.c file.

It is necessary to define descriptors in order to set the features like UUID, Attribute handle, Attribute permissions and the variables of Characteristic Values to the Attribute database.

Project_Source\application\src\connect\resource\r_gatt.c, line 103-179

```
103:    /* Custom Service */
104:    static const uint8_t custom_svc[RBLE_GATT_128BIT_UUID_OCTET] = PRF_UUID_SERVICE;
105:
106:    /* Advertising Information */
107:    static const struct atts_char128_desc prf_bcninfo_char =
108:    {
 :          …
114:    };
115:
116:    struct atts_elmt_128 prf_bcninfo_elmt =
117:    {
 :          …
119:    };
120:
121:    /* Advertising Data */
122:    static const struct atts_char128_desc prf_bcndata_char =
123:    {
 :          …
129:    };
130:
131:    struct atts_elmt_128 prf_bcndata_elmt =
132:    {
 :          …
134:    };
135:
136:    /* Scan Response Data */
137:    static const struct atts_char128_desc prf_rspdata_char =
138:    {
 :          …
144:    };
145:
146:    struct atts_elmt_128 prf_rspdata_elmt =
147:    {
 :          …
149:    };
150:
151:    /* Code Flash Updated Count */
152:    static const struct atts_char128_desc prf_cflcnt_char =
153:    {
 :          …
159:    };
160:
161:    struct atts_elmt_128 prf_cflcnt_elmt =
162:    {
 :          …
164:    };
165:
166:    /* Data Flash Updated Count */
167:    static const struct atts_char128_desc prf_dflcnt_char =
168:    {
 :          …
174:    };
175:
176:    struct atts_elmt_128 prf_dflcnt_elmt =
177:    {
 :          …
179:    };
```

Attribute database is defined in r_gatt.c file.

Attribute database is needed in order to set Service and Characteristics to Protocol Stack. Attribute database consists of Service descriptors and Characteristic descriptors.

Project_Source\application\src\connect\resource\r_gatt.c, line 213-243

```
213:    /* Attribute Database */
214:    const struct atts_desc atts_desc_list_prf[] =
215:    {
216:        /***********************/
217:        /* Custom Service      */
218:        /***********************/
219:        {RBLE_DECL_PRIMARY_SERVICE,  sizeof(custom_svc),       sizeof(custom_svc),             … },
220:        /* Advertising Information */
221:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_bcninfo_char), sizeof(prf_bcninfo_char),       … },
222:        { DB_TYPE_128BIT_UUID,       sizeof(PRF_ADV_INFO),     sizeof(PRF_ADV_INFO),           … },
223:        /* Advertising Data */
224:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_bcndata_char), sizeof(prf_bcndata_char),       … },
225:        { DB_TYPE_128BIT_UUID,       sizeof(PRF_ADV_DATA),     sizeof(PRF_ADV_DATA),           … },
226:        /* Scan Response Data */
227:        #if RF_TX_ONLY
228:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_rspdata_char), sizeof(prf_rspdata_char),       … },
229:        { DB_TYPE_128BIT_UUID,       sizeof(PRF_ADV_DATA),     sizeof(PRF_ADV_DATA),           … },
230:        #else
231:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_rspdata_char), sizeof(prf_rspdata_char),       … },
232:        { DB_TYPE_128BIT_UUID,       sizeof(PRF_ADV_DATA),     sizeof(PRF_ADV_DATA),           … },
233:        #endif
234:        /* Code Flash Updated Count */
235:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_cflcnt_char),  sizeof(prf_cflcnt_char),        … },
236:        { DB_TYPE_128BIT_UUID,       sizeof(uint16_t),         sizeof(uint16_t),               … },
237:        /* Data Flash Updated Count */
238:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_dflcnt_char),  sizeof(prf_dflcnt_char),        … },
239:        { DB_TYPE_128BIT_UUID,       sizeof(uint16_t),         sizeof(uint16_t),               … },
240:
241:        /* zero terminator */
242:        {0,0,0,0,0,0}
243:    };
```

(3) **Processing**

Processing for enabling GATT and updating Characteristic Values are implemented in r_profile.c file.

RBLE_GATT_Enable function is called to enable GATT and register GATT event callback function. When updating Characteristic Values is requested by Write Request from Client device, RBLE_GATT_EVENT_WRITE_CMD_IND event occurs. After updating the Characteristic Value, the application calls RBLE_GATT_Write_Response in order to send Write Response.

Project_Source\application\src\connect\r_profile.c, line 92-341

```
 92:    RBLE_STATUS PRF_Server_Enable(uint16_t conhdl, PRF_EVT_HANDLER callback)
 93:    {
101:            result = RBLE_GATT_Enable(prf_gatt_callback);
113:    }
 :
215:    static void prf_gatt_callback(RBLE_GATT_EVENT* evt)
216:    {
224:        switch(evt->type)
225:        {
226:            case RBLE_GATT_EVENT_WRITE_CMD_IND:
227:                /* reach here when client device requests to write characteristic */
235:                switch(att_hdl)
236:                {
237:                    /* Advertising information (18byte fixed) is written */
238:                    case PRF_HDL_BCNINFO_VAL:
 :
243:                            /* update characteristic value */
 :
256:                        break;
257:
258:                    /* Advertising data (2byte - 32byte variable) is written */
259:                    /* Note: when requested size is over than the size of single write request, */
260:                    /*       characteristic value is transferred separately per 18byte          */
261:                    case PRF_HDL_BCNDATA_VAL:
 :
268:                                /* update characteristic value */
 :
277:                        break;
304:                }
305:
306:                /* send the write response to client device */
307:                if(evt->param.write_cmd_ind.resp)
308:                {
309:                    prf_send_wr_resp(att_hdl, result);
310:                }
311:                break;
315:        }
316:    }
317:
 :
325:    static void prf_send_wr_resp(uint16_t att_hdl, RBLE_STATUS result)
326:    {
340:        RBLE_GATT_Write_Response(&wr_resp);
341:    }
```

6.2.7 RF Operation

If change RF Operation of Beacon Stack, the application is changed. Changes of Beacon Application and Connect Application are shown in **Table 6-2**.

Regarding to the detail of RF Operation, refer to chapter 6.1.2 "RF Operation" in this document.

Table 6-2 Software Configuration

	Both Tx and Rx enabled (RF_TX_ONLY=0)	Only Tx enabled (RF_TX_ONLY=1)
Beacon Application		
Advertising Type (r_beacon.c)	Scannable Undirected Advertising	Non-connectable Undirected Advertising
Connect Application		
Custom Profile (r_gatt.c) (r_profile.c) (r_connect.c)	Characteristics − Advertising Information − Advertising Data − Scan Response Data − Code Flash Memory Updated Count − Data Flash Memory Updated Count	Characteristics − Advertising Information − Advertising Data − Code Flash Memory Updated Count − Data Flash Memory Updated Count Note that it is impossible to access Scan Response Data Characteristic

iOS device stores services and characteristics constitution of connected device. Changing RF operation causes a difference between constitution of Connect Application and the stored constitution.

If change RF Operation, to clear the services and characteristics constitution stored by iOS device, disable and enable Bluetooth in iOS Settings.

7. Functions

This chapter describes major functions implemented in the Sample Program.

7.1 Function List

7.1.1 Switching Application

Table 7-1 shows the functions for switching application.

Table 7-1 Application Switching Functions

file	function	description
r_main.c	main	initializes MCU and executes Applications
	input_callback	calls application exit function
r_input,c	R_INPUT_Init	initializes external interrupt input
	intp5_interrupt	handler of external interrupt input
r_plf.c	R_PLF_Init	initializes MCU (ports and clock)

7.1.2 Beacon Application

Table 7-2 shows the functions of Beacon Application.

Table 7-2 Beacon Application Functions

file	function	description
r_beacon_main.c	R_BEACON_Main	main loop of Beacon Application
r_beacon.c	R_BEACON_Start	starts Beacon Application
	R_BEACON_Exit	exits Beacon Application
	R_BEACON_EventHandler	event handler

7.1.3 Connect Application

Table 7-3 shows the function of Connect Application.

Table 7-3 Connect Application Functions

file	function	description
r_connect_main.c	R_CONNECT_Main	main loop of Connect Application
r_connect.c	R_CONNECT_Start	starts Connect Application
	R_CONNECT_Exit	exits Connect Application
	con_rble_callback	event callback function : RBLE
	con_gap_callback	event callback function : Generic Access Profile
	con_sm_callback	event callback function : Security Manager
	con_profile_callback	event callback function : Custom Profile
	con_vs_callback	event callback function : Vendor Specific
	con_exit_timer_task	exits Connect Application when no connection within 30sec

7.1.4 DTM Application

Table 7-4 shows the functions of DTM Application.

Table 7-4 DTM Application Functions

module	function	description
r_dtm_main.c	R_DTM_Main	main loop of DTM Application
	R_DTM_Start	starts DTM Application
r_dtm.c	dtm_rble_callback	event callback function : RBLE
	dtm_gap_callback	event callback function : Generic Access Profile
	dtm_sm_callback	event callback function : Security Manager
	dtm_vs_callback	event callback function : Vendor Specific

7.2 Function Calling

7.2.1 Function Calling of Beacon Operation

Figure 7-1 shows the function calling graph of Beacon Operation. If DIP switch SW6 position-1 on the evaluation board is OFF, main function calls R_BEACON_Main function or R_CONNECT_Main function alternately.

R_BEACON_Main function starts Beacon Application by calling R_BEACON_Start function and executes main loop, then the main loop breaks by calling R_BEACON_Break function.

R_CONNECT_Main function starts Connect Application by calling R_CONNECT_Start function and executes main loop, then main loop exits by calling R_CONNECT_Break function.

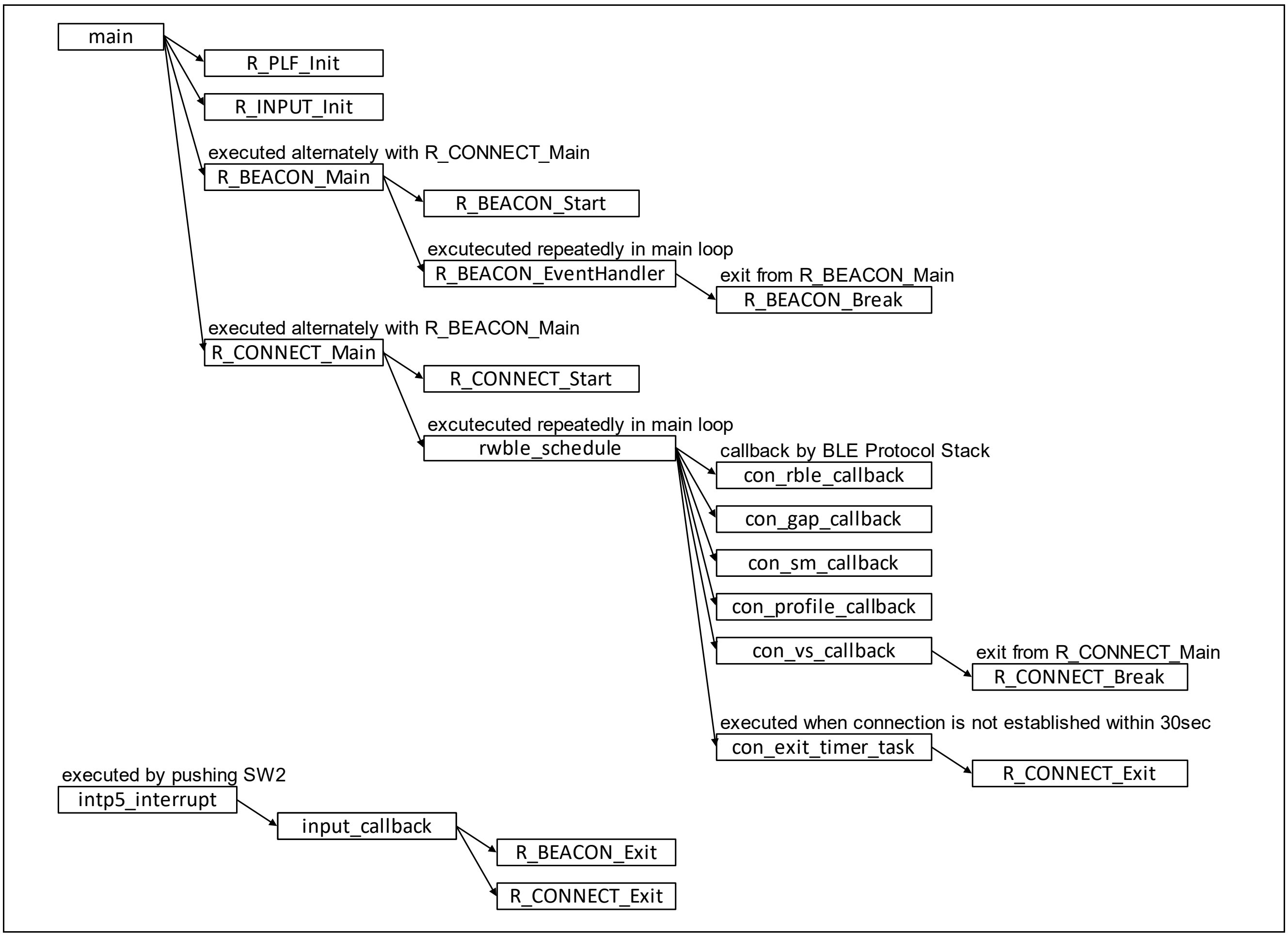


Figure 7-1 Function Calling Graph of Beacon Operation

7.2.2 Function Calling of RF Evaluation Operation

Figure 7-2 shows the function calling graph of RF Evaluation Operation. If DIP switch SW6 position-1 on the evaluation board is ON, main function calls R_DTM_Main function. R_DTM_Main function starts DTM Application by calling R_DTM_Start function and executes main loop. The main loop of DTM Application never break.

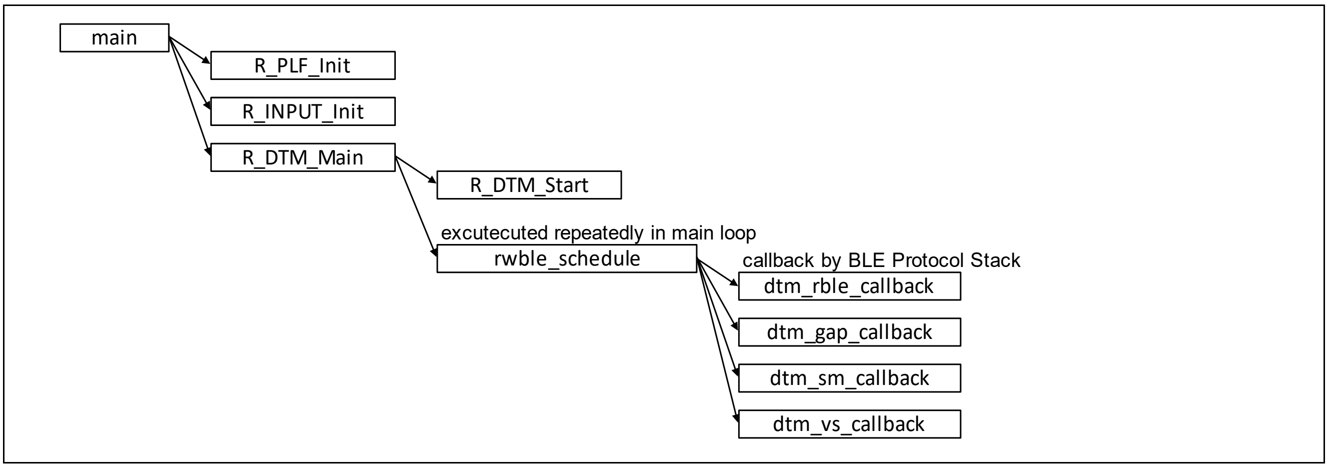


Figure 7-2 Function Calling Graph of RF Evaluation Operation

8. Operation

8.1 State Transition

There are three applications in the Sample Program: Beacon Application, Connect Application, and DTM Application. This section describes the state transition of each application.

8.1.1 Beacon Application

Figure 8-1 shows the state transition of Beacon Application.

It starts with Initializing state and then follow by Advertising state. In the Advertising state, the application executes Advertising. If receive exit request, next go to RF Powerdown state and finally exit from Beacon Application.

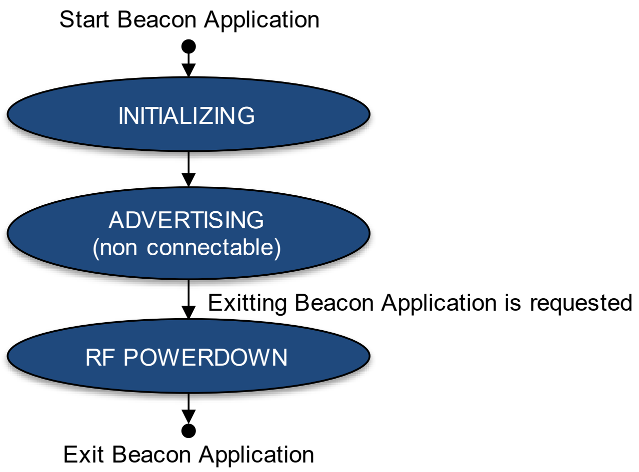


Figure 8-1 State Transition of Beacon Application

8.1.2 Connect Application

Figure 8-2 shows the state transition of Connect Application.

It starts with Initializing state and the follow by Advertising state. In the Advertising state, the application executes Advertising.

If receive connection request from peer device, go to Slave Connection state. In the Slave Connection state, the application enables GATT and checks about security status. If receive either Pairing Request or Start Encryption Request from peer connected device, the application executes pairing or start encryption respectively. And in this state, the application executes GATT Access. If disconnect from peer connected device, go back to the Advertising state.

In the Advertising state, if receive exit request or if not connect within 30seconds, the application stops Advertising and next go to RF Powerdown state and finally exit from Connect Application.

In the Slave Connection state, if receive exit request, the application requests disconnection to peer device and next go to RF Powerdown state and finally exit from Connect Application.

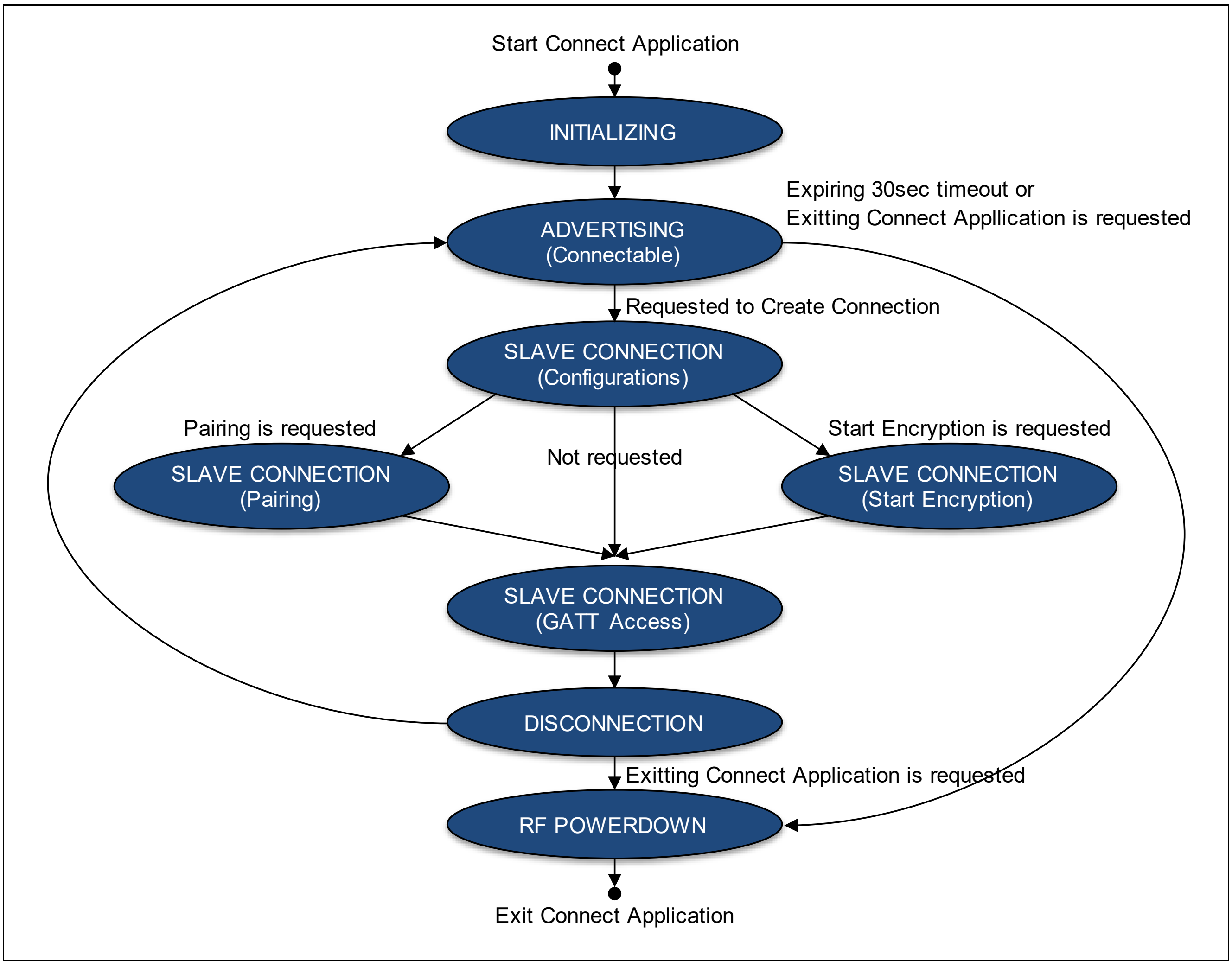


Figure 8-2 State Transition of Connect Application

8.1.3 DTM Application

Figure 8-3 shows the state transition of DTM Application.

It starts with Initializing state and the follow by Idling state. Then perform either RF Transmitter Test or RF Receiver Test. Respective test executes according the request from Tester and return to Idling state when complete the test.

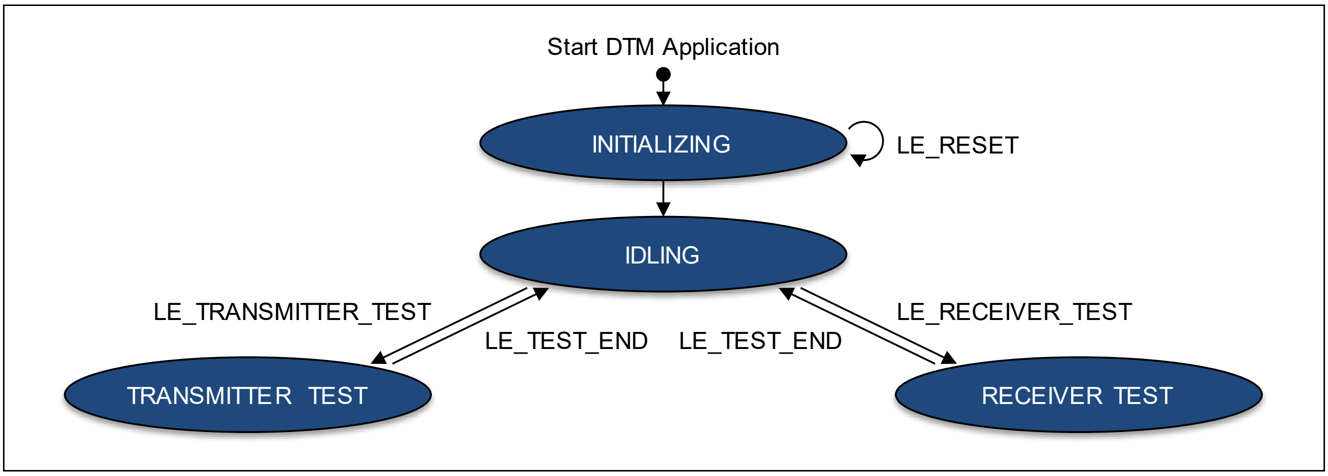


Figure 8-3 State Transition of DTM Application

8.2 Sequence

8.2.1 Beacon Application

(1) **Initializing & Advertising & RF Powerdown Sequence**

Figure 8-4 shows the sequence in Initializing state, Advertising state and RF Powerdown state of Beacon Application. Beacon Stack API is used in sequence between Beacon Application and Beacon Stack in the Sample Program.

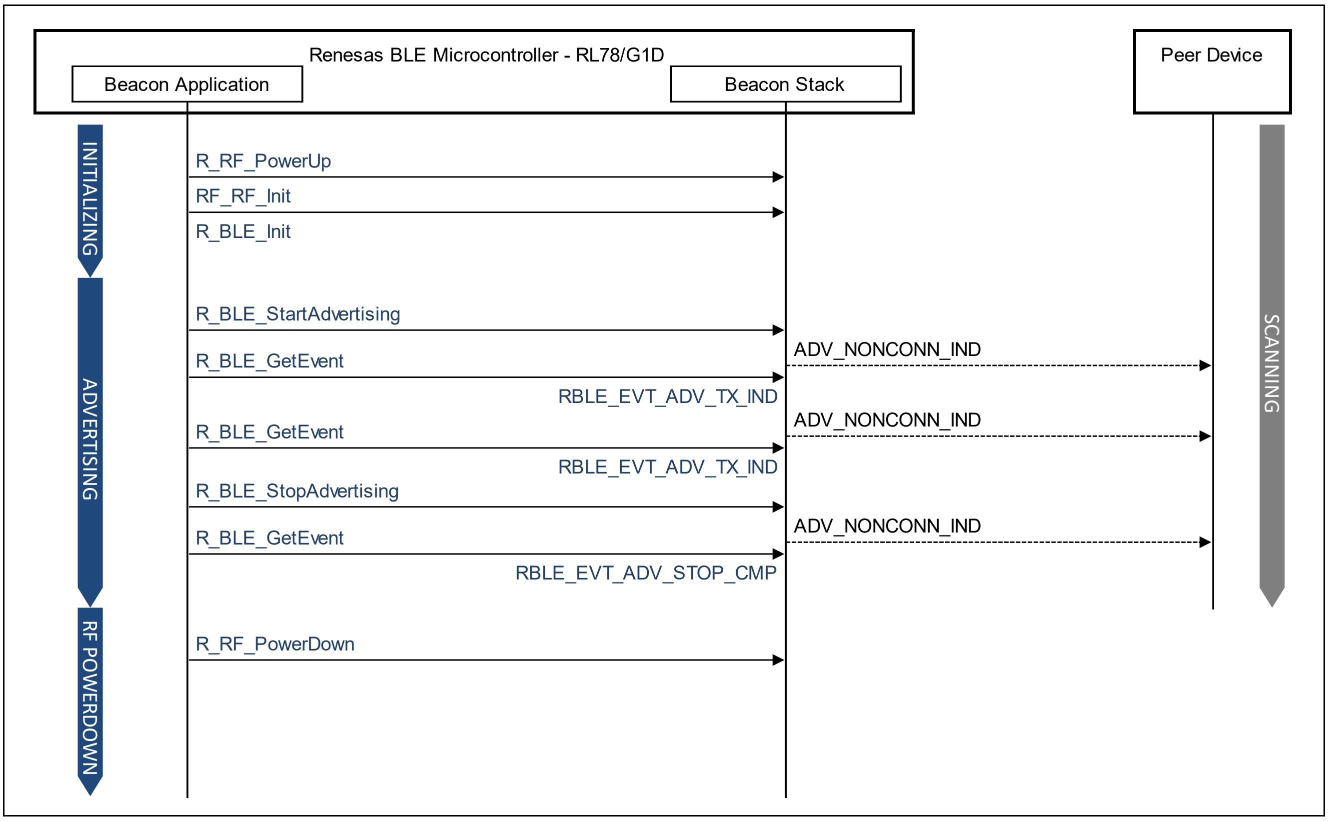


Figure 8-4 Initializing & Advertising & RF Powerdown Sequence of Beacon Application

Regarding to the specification of Beacon Stack API, refer to chapter 4 "API" in RL78/G1D Beacon Stack User's Manual (R01UW0171).

Beacon Application calls R_RF_PowerUp and R_RF_Init in order to enable RF unit. After that, the application calls R_BLE_Init in order to initialize Beacon Stack.

Project_Source\application\src\beacon\r_beacon_main.c, line 63-129

```
 63:    void R_BEACON_Main(void)
 64:    {
 72:        /*
 73:         ***************************************************************************************
 74:         * Beacon Stack Initialization
 75:         ***************************************************************************************
 76:        */
 77:        if (RBLE_OK != R_RF_PowerUp(BCN_RF_CFG, RF_32MHZ_WAIT))
 78:        {
 81:            mcu_reset();
 82:        }
 83:
 84:        if (RBLE_OK != R_RF_Init())
 85:        {
 88:            mcu_reset();
 89:        }
 90:
 91:        /* Initialize Beacon Stack */
 92:        interrupt_init();
 93:        R_BLE_Init();
129:    }
```

The application calls R_BLE_StartAdvertising in order to start Advertising for providing information.

RBLE_EVT_ADV_TX_IND event occurs after every transmitting of advertising packets.

The application calls R_BLE_StopAdvertising in order to request stopping Advertising. After stopping Advertising RBLE_EVT_ADV_STOP_CMP event occurs.

Project_Source\application\src\beacon\r_beacon.c, line 132-202

```
132:    bool R_BEACON_Start(void)
133:    {
149:        if(RBLE_OK != R_BLE_StartAdvertising(adv_type, &adv_info))
150:        {
151:            return false;
152:        }
155:    }
 :
163:    void R_BEACON_Exit(void)
164:    {
165:        R_BLE_StopAdvertising();
166:    }
 :
174:    void R_BEACON_EventHandler(void)
175:    {
176:        RBLE_EVT* evt = R_BLE_GetEvent();
177:
178:        while (evt != NULL)
179:        {
180:            switch (evt->type)
181:            {
182:                case RBLE_EVT_ADV_TX_IND:
183:                    /* reach here after transmitting Advertising packet */
184:                    bcn_adv_tx_eventhandler(evt);
185:                    break;
186:
187:                case RBLE_EVT_SCANREQ_RX_IND:
188:                    /* reach here after receiving scan request packet */
189:                    bcn_scanreq_rx_eventhandler(evt);
190:                    break;
191:
192:                case RBLE_EVT_ADV_STOP_CMP:
193:                    /* reach here when advertising is stopped */
194:                    bcn_adv_stop_eventhandler(evt);
195:                    break;
196:
197:                default:
198:                    break;
199:            }
200:            evt = R_BLE_GetEvent();
201:        }
202:    }
```

8.2.2 Connect Application

(1) **Initializing & Advertising & Slave Connection (Configurations) Sequence**

Figure 8-5 shows the sequence in Initialization state, Advertising state and Slave Connection (Configurations) state of Connect Application. rBLE API is used in sequence between Connect Application and BLE Protocol Stack in the Sample Program.

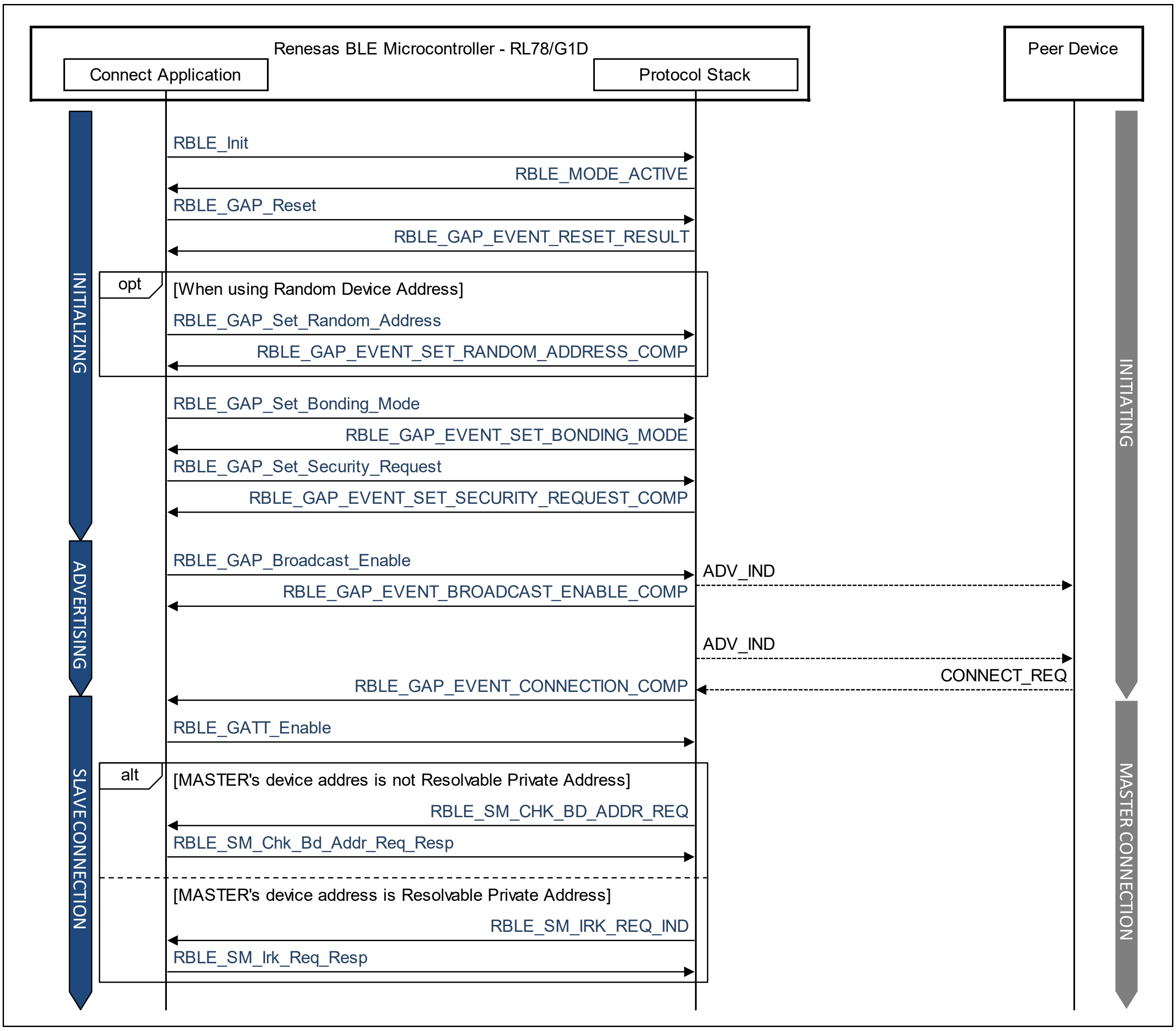


Figure 8-5 Initializing & Advertising & Slave Connection (Configurations) Sequence of Connect Application

Regarding to the specification of rBLE API, refer to Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

First of all, Connect Application calls RBLE_Init in order to activate the rBLE_Core of BLE Protocol Stack.

The application calls RBLE_GAP_Reset in order to initialize GAP layer of Protocol Stack and GAP event callback function and SM event callback function.

Project_Source\application\src\connect\r_connect.c, line 268-490

```
268:    bool R_CONNECT_Start(void)
269:    {
270:        /* initialize rBLE */
271:        if(RBLE_OK != RBLE_Init(&con_rble_callback))
272:        {
273:            return false;
274:        }
293:    }
 :
374:    static void con_rble_callback(RBLE_MODE mode)
375:    {
383:        switch(mode)
384:        {
385:            case RBLE_MODE_ACTIVE:
386:                /* reach here when activating rBLE is completed after calling RBLE_Init */
387:                con_rble_active_eventhandler();
388:                break;
391:        }
392:    }
 :
400:    static void con_rble_active_eventhandler(void)
401:    {
402:        /* reach here when activating rBLE is completed after calling RBLE_Init */
405:        RBLE_GAP_Reset(&con_gap_callback, &con_sm_callback);
406:    }
 :
414:    static void con_gap_callback(RBLE_GAP_EVENT* evt)
415:    {
440:        switch(evt->type)
441:        {
442:            case RBLE_GAP_EVENT_RESET_RESULT:
443:                /* reach here after RBLE_GAP_Reset is called */
444:                con_gap_reset_eventandler(evt);
445:                break;
446:            case RBLE_GAP_EVENT_SET_RANDOM_ADDRESS_COMP:
447:                /* reach here after RBLE_GAP_Set_Random_Address is called */
448:                con_gap_set_random_address_eventhandler(evt);
449:                break;
450:            case RBLE_GAP_EVENT_SET_BONDING_MODE_COMP:
451:                /* reach here after RBLE_GAP_Set_Bonding_Mode is called */
452:                con_gap_set_bonding_mode_eventhandler(evt);
453:                break;
454:            case RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP:
455:                /* reach here after RBLE_GAP_Set_Security_Request is called */
456:                con_gap_set_security_request_eventhandler(evt);
457:                break;
458:            case RBLE_GAP_EVENT_BROADCAST_ENABLE_COMP:
459:                /* reach here after RBLE_GAP_Broadcast_Enable is called */
460:                con_gap_broadcast_enable_eventhandler(evt);
461:                break;
466:            case RBLE_GAP_EVENT_CONNECTION_COMP:
467:                /* reach here when connection occurred */
468:                con_gap_connection_eventhandler(evt);
469:                break;
489:        }
490:    }
```

If Random Device Address is used, the application calls RBLE_GAP_Set_Random_Address in order to set random address to Protocol Stack.

To execute pairing sequence to the peer device, the application calls RBLE_GAP_Set_Bonding_Mode and RBLE_GAP_Set_Security_Request.

After above initializing is completed, the application calls RBLE_GAP_Broadcast_Enable to start Advertising for establishing connection as Slave.

Project_Source\application\src\connect\r_connect.c, line 498-609

```
498:    static void con_gap_reset_eventandler(RBLE_GAP_EVENT* evt)
499:    {
500:        /* reach here after RBLE_GAP_Reset is called */
501:
541:            if(own_type == RBLE_ADDR_RAND)
542:            {
546:                /* Set Random Device Address */
547:                RBLE_GAP_Set_Random_Address(&own_addr);
551:            }
552:            else
553:            {
554:                /* Set Bonding Mode */
555:                RBLE_GAP_Set_Bonding_Mode(RBLE_GAP_BONDABLE);
556:            }
558:    }
 :
566:    static void con_gap_set_random_address_eventhandler(RBLE_GAP_EVENT* evt)
567:    {
568:        /* reach here after RBLE_GAP_Set_Random_Address is called */
569:
572:            /* Set Bonding Mode */
573:            RBLE_GAP_Set_Bonding_Mode(RBLE_GAP_BONDABLE);
575:    }
 :
583:    static void con_gap_set_bonding_mode_eventhandler(RBLE_GAP_EVENT* evt)
584:    {
585:        /* reach here after RBLE_GAP_Set_Bonding_Mode is called */
586:
589:            /* Set Security Request */
590:            RBLE_GAP_Set_Security_Request(RBLE_GAP_SEC1_NOAUTH_PAIR_ENC);
592:    }
 :
600:    static void con_gap_set_security_request_eventhandler(RBLE_GAP_EVENT* evt)
601:    {
602:        /* reach here after RBLE_GAP_Set_Security_Request is called */
603:
606:            /* Start Broadcast for the First Connection */
607:            RBLE_GAP_Broadcast_Enable(RBLE_GAP_GEN_DISCOVERABLE, RBLE_GAP_UND_CONNECTABLE,      … );
609:    }
```

After establishing connection, RBLE_GAP_EVENT_CONNECTION_COMP event occurs.

If peer's Device Address Type is not Resolvable Private Address, RBLE_SM_CHK_BD_ADDR_REQ event occurs. The application should call RBLE_SM_Chk_Bd_Addr_Req_Resp to respond security status in previous connection with peer device. In order to execute Pairing again if peer device forgets Pairing information, the application always respond that there is no security status.

If peer's Device Address Type is Resolvable Private Address, RBLE_SM_IRK_REQ_IND occurs. The application should call RBLE_SM_Irk_Req_Resp to respond security status and Identity Resolving Key (IRK) for resolving address. But in order to execute Pairing again if peer device forgets Pairing information, the application always respond that there are no security status and no IRK.

Project_Source\application\src\connect\r_connect.c, line 815-906

```
815:    static void con_sm_callback(RBLE_SM_EVENT* evt)
816:    {
824:        switch(evt->type)
825:        {
826:            case RBLE_SM_CHK_BD_ADDR_REQ:
827:                /* reach here when connection is established to peer device that address is
828:                   public address or random address except resolvable private address       */
829:                con_sm_bdaddr_check_request_eventhandler(evt);
830:                break;
831:            case RBLE_SM_IRK_REQ_IND:
832:                /* reach here when connection is established to peer device that address is
833:                   resolvable private address                                                */
834:                /* IRK is requested for resolving peer's resolvable private address         */
835:                con_sm_irk_request_eventhandler(evt);
836:                break;
858:        }
859:    }
 :
867:    static void con_sm_bdaddr_check_request_eventhandler(RBLE_SM_EVENT* evt)
868:    {
869:        /* reach here when connection is established to peer device that address is
870:           public address or random address except resolvable private address       */
871:
876:        /* Reply BD Address Check Result */
877:        RBLE_SM_Chk_Bd_Addr_Req_Resp(evt->param.chk_bdaddr.idx,
878:                                     0,
879:                                     false,
880:                                     RBLE_SMP_SEC_NONE,
881:                                     NULL);
882:    }
 :
890:    static void con_sm_irk_request_eventhandler(RBLE_SM_EVENT* evt)
891:    {
892:        /* reach here when connection is established to peer device that address is
893:           resolvable private address                                                */
894:        /* IRK is requested for resolving peer's resolvable private address         */
895:
900:        /* Reply IRK(Identity Resolving Key) */
901:        RBLE_SM_Irk_Req_Resp(evt->param.irk_req.idx,
902:                             RBLE_ERR,
903:                            &con_env.con_addr,
904:                             NULL,
905:                             RBLE_SMP_SEC_NONE);
906:    }
```

(2) **Slave Connection (Pairing) Sequence**

Figure 8-6 shows the sequence in Slave Connection (Pairing) state of Connect Application. rBLE API is used in sequence between Connect Application and BLE Protocol Stack in the Sample Program.

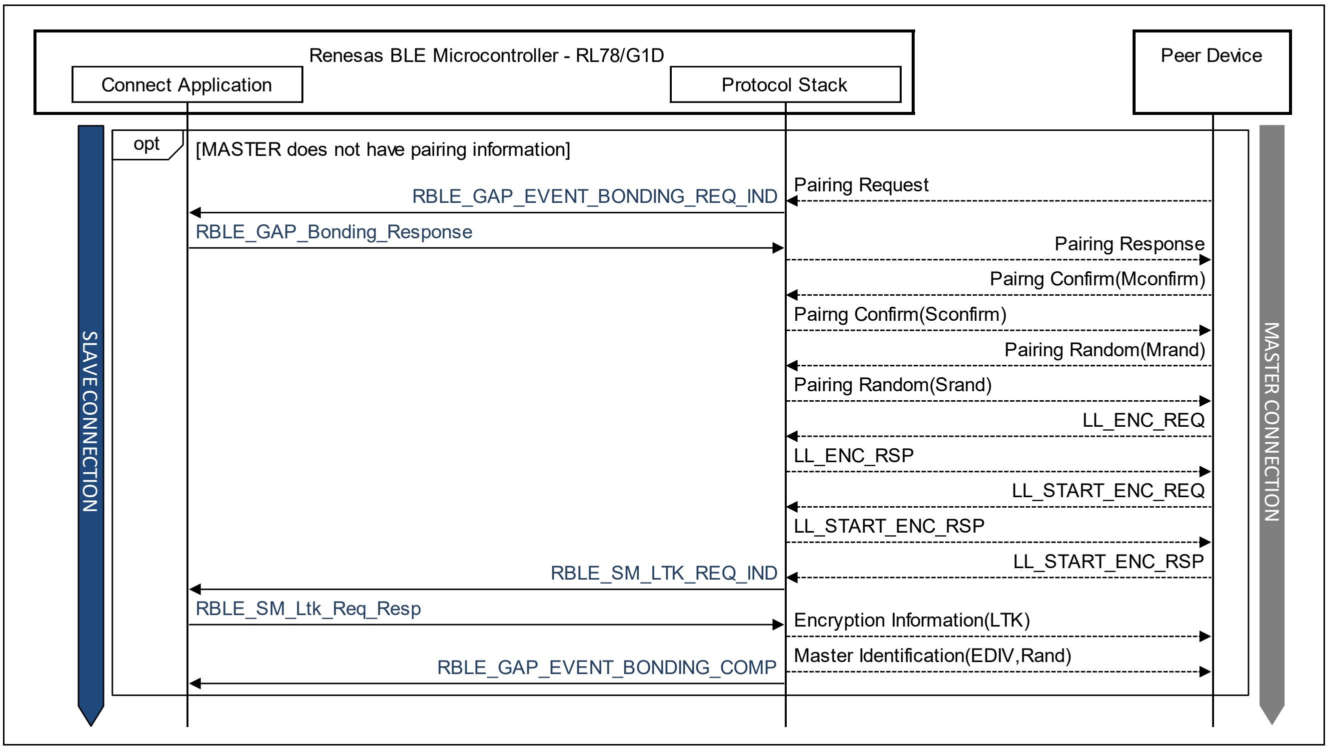


Figure 8-6 Slave Connection (Pairing) Sequence of Connect Application

Regarding to the specification of rBLE API, refer to Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

When pairing request is sent after establishing connection to the unpaired device, RBLE_GAP_BONDING_REQ_IND event occurs. The application calls RBLE_GAP_Bonding_Response in order to respond the pairing features. Pairing method is determined by exchanging pairing feature. In the Sample Program, Just Works is executed as pairing method. In the case of Just Works, it is not necessary to set Temporary Key to Protocol Stack. Following pairing process is encrypted with Short Term Key, which is generated by BLE Protocol Stack.

Project_Source\application\src\connect\r_connect.c, line 414-784

```
414:    static void con_gap_callback(RBLE_GAP_EVENT* evt)
415:    {
440:        switch(evt->type)
441:        {
477:            case RBLE_GAP_EVENT_BONDING_REQ_IND:
478:                /* reach here when bonding is requested */
479:                /* in the middle of PHASE1: PAIRING FEATURE EXCHANGE in pairing sequence */
480:                con_gap_bonding_request_eventhandler(evt);
481:                break;
482:            case RBLE_GAP_EVENT_BONDING_COMP:
483:                /* reach here bonding is completed */
484:                /* at the end of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
485:                con_gap_bonding_eventhandler(evt);
486:                break;
489:        }
490:    }
 :
776:    static void con_gap_bonding_request_eventhandler(RBLE_GAP_EVENT* evt)
777:    {
778:        /* reach here when bonding is requested */
779:        /* in the middle of PHASE1: PAIRING FEATURE EXCHANGE in pairing sequence */
780:
781:        /* Reply Bonding Response */
783:        RBLE_GAP_Bonding_Response(&bond_info);
784:    }
```

After starting encryption with Short Term Key, RBLE_SM_LTK_REQ_IND event occurs. The application generates Long Term Key (LTK) and calls RBLE_SM_Req_Resp in order to respond LTK. LTK is used as encryption key for encrypting following connection.

After providing LTK to Master device, RBLE_SM_KEY_IND event occurs and encryption key is provided from Master device, which is specified by the bonding response.

When pairing sequence is completed, RBLE_GAP_EVENT_BONDING_COMP event occurs.

Project_Source\application\src\connect\r_connect.c, line 815-940

```
815:    static void con_sm_callback(RBLE_SM_EVENT* evt)
816:    {
824:        switch(evt->type)
825:        {
837:            case RBLE_SM_LTK_REQ_IND:
838:                /* reach here when LTK is requested */
839:                /* in the first of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
840:                con_sm_ltk_request_eventhandler(evt);
841:                break;
842:            case RBLE_SM_KEY_IND:
843:                /* reach here when peer device's encryption information are provided */
844:                /* in the middle of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
845:                con_sm_key_eventhandler(evt);
846:                break;
858:        }
859:    }
 :
914:    static void con_sm_ltk_request_eventhandler(RBLE_SM_EVENT* evt)
915:    {
916:        /* reach here when LTK is requested */
917:        /* in the first of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
918:
933:        /* Reply LTK(Long Term Key) */
934:        RBLE_SM_Ltk_Req_Resp(evt->param.ltk_req.idx,
935:                             RBLE_OK,
936:                             RBLE_SMP_KSEC_NONE,
937:                             pair_info.enc_key.ediv,
938:                            &pair_info.enc_key.nb  ,
939:                            &pair_info.enc_key.ltk );
940:    }
```

(3) **Slave Connection (Start Encryption) Sequence**

Figure 8-7 shows the sequence in Slave Connection (Start Encryption) state of Connect Application. rBLE API is used in sequence between Connect Application and BLE Protocol Stack in the Sample Program.

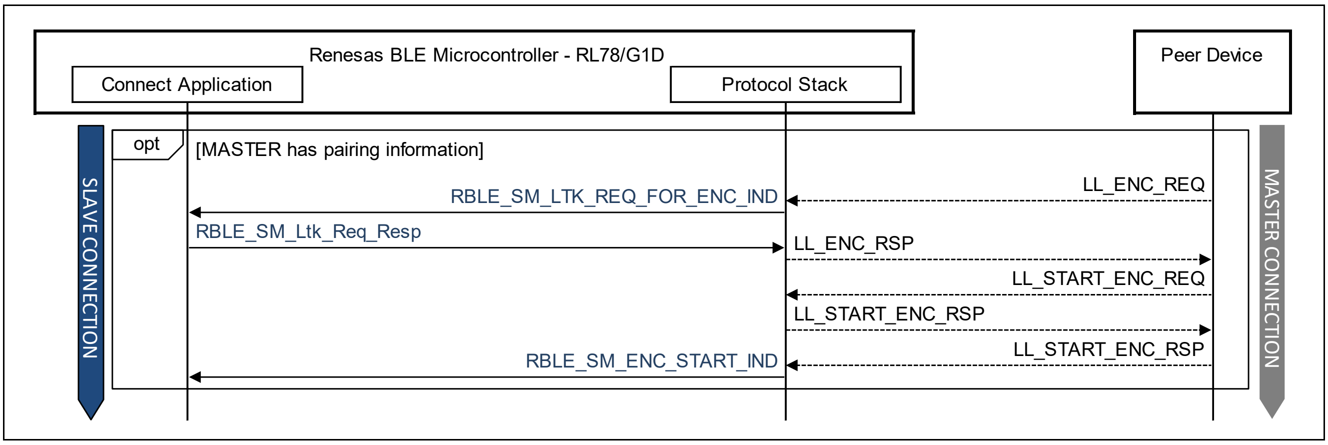


Figure 8-7 Slave Connection (Start Encryption) Sequence of Connect Application

Regarding to the specification of rBLE API, refer to Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

When the encryption request is sent after establishing connection to the paired device, RBLE_SM_LTK_REQ_FOR_ENC_IND event occurs. The application calls RBLE_SM_Ltk_Req_Resp in order to respond Long Term Key (LTK). LTK is generated and exchanged already in before connection, and used as encryption key for encrypting this connection.

When start encryption sequence is completed, RBLE_SM_ENC_START_IND event occurs.

Project_Source\application\src\connect\r_connect.c, line815-984

```
815:    static void con_sm_callback(RBLE_SM_EVENT* evt)
816:    {
824:        switch(evt->type)
825:        {
847:            case RBLE_SM_LTK_REQ_FOR_ENC_IND:
848:                /* reach here when LTK is requested */
849:                /* in the first of start encryption sequence */
850:                con_sm_ltk_request_for_enc_eventhandler(evt);
851:                break;
852:            case RBLE_SM_ENC_START_IND:
853:                /* reach here when start encryption sequence is executed */
854:                con_sm_enc_start_eventhandler(evt);
855:                break;
858:        }
859:    }
 :
960:    static void con_sm_ltk_request_for_enc_eventhandler(RBLE_SM_EVENT* evt)
961:    {
962:        /* reach here when LTK is requested */
963:        /* in the first of Start Encryption sequence */
964:
977:        /* Reply LTK(Long Term Key) */
978:        RBLE_SM_Ltk_Req_Resp(evt->param.ltk_req_for_enc.idx,
979:                             status,
980:                             pair_info.sec_prop,
981:                             pair_info.enc_key.ediv,
982:                            &pair_info.enc_key.nb  ,
983:                            &pair_info.enc_key.ltk );
984:    }
```

(4) **Slave Connection (GATT Access) Sequence**

Figure 8-8 shows the sequence in Slave Connection (GATT Access) state of Connect Application. rBLE API is used in sequence between Connect Application and BLE Protocol Stack in the Sample Program.

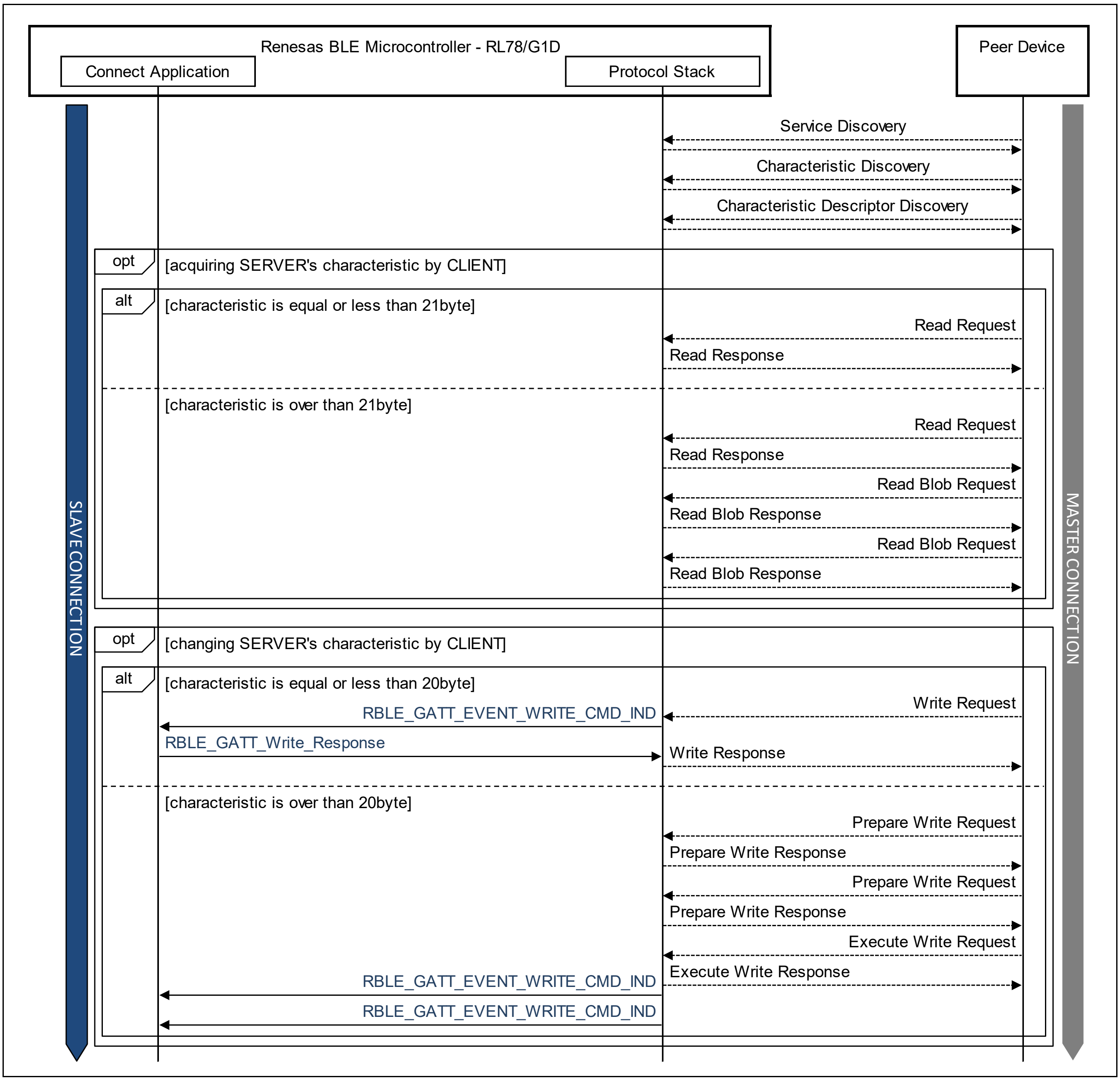


Figure 8-8 Slave Connection (GATT Access) Sequence of Connect Application

Regarding to the specification of rBLE API, refer to Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

When Service Discovery, Characteristic Discovery and Characteristic Descriptor Discovery are requested from Client device, response are executed by Protocol Stack automatically.

Similarly, when Read Request is sent from Client device, Read Response is sent by Protocol Stack automatically.

When updating characteristic values is requested by Write Request from Client device, RBLE_GATT_EVENT_WRITE_CMD_IND event occurs. After updating characteristic value, the application calls RBLE_GATT_Write_Response in order to send Write Response.

Project_Source\application\src\connect\r_profile.c, line 215-341

```
215:    static void prf_gatt_callback(RBLE_GATT_EVENT* evt)
216:    {
224:        switch(evt->type)
225:        {
226:            case RBLE_GATT_EVENT_WRITE_CMD_IND:
227:                /* reach here when client device requests to write characteristic */
228:
235:                switch(att_hdl)
236:                {
237:                    /* Advertising information (18byte fixed) is written */
238:                    case PRF_HDL_BCNINFO_VAL:
243:                                /* update characteristic value */
256:                        break;
257:
258:                    /* Advertising data (2byte - 32byte variable) is written */
259:                    /* Note: when requested size is over than the size of single write request, */
260:                    /*       characteristic value is transferred separately per 18byte          */
261:                    case PRF_HDL_BCNDATA_VAL:
268:                                    /* update characteristic value */
277:                        break;
278:
279:                    /* Scan Response data (2byte - 32byte variable) is written */
280:                    /* Note: when requested size is over than the size of single write request, */
281:                    /*       characteristic value is transferred separately per 18byte          */
282:                    #if !RF_TX_ONLY
283:                    case PRF_HDL_RSPDATA_VAL:
290:                                    /* update characteristic value */
299:                        break;
300:                    #endif
304:                }
305:
306:                /* send the write response to client device */
307:                if(evt->param.write_cmd_ind.resp)
308:                {
309:                    prf_send_wr_resp(att_hdl, result);
310:                }
311:                break;
315:        }
316:    }
 :
325:    static void prf_send_wr_resp(uint16_t att_hdl, RBLE_STATUS result)
326:    {
340:        RBLE_GATT_Write_Response(&wr_resp);
341:    }
```

(5) **Disconnection & RF Powerdown Sequence**

Figure 8-9 shows the sequence in Disconnection state and RF Powerdown state of Connect Application. rBLE API is used in sequence between Connect Application and BLE Protocol Stack in the Sample Program.

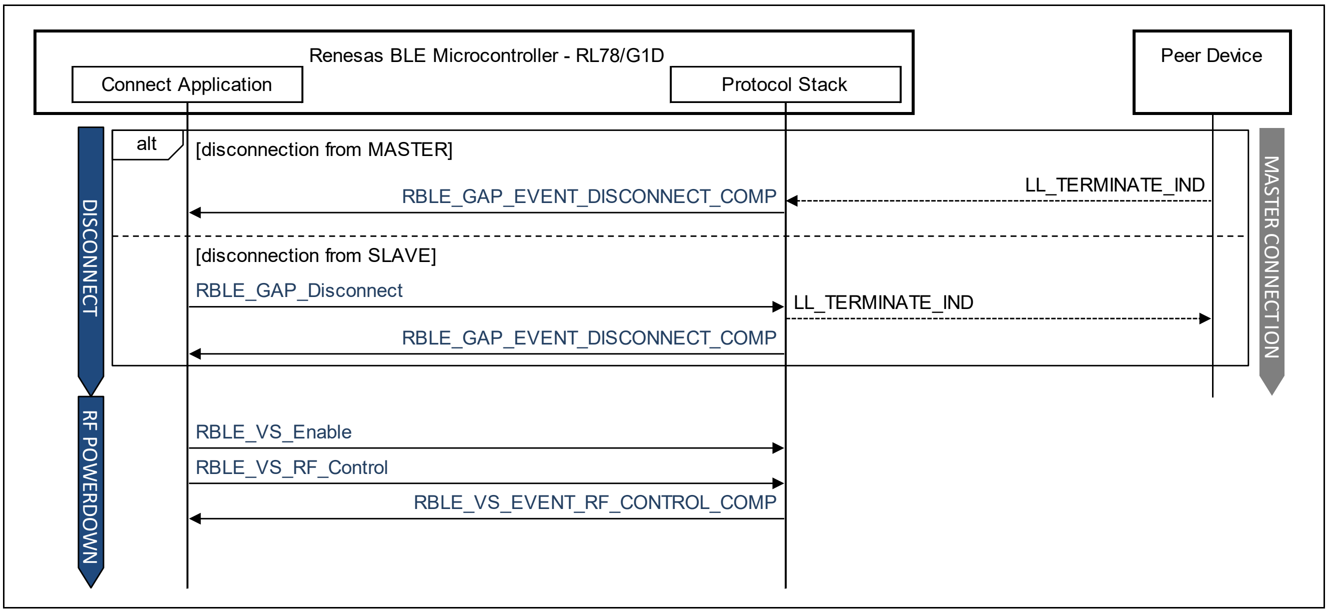


Figure 8-9 Disconnection & RF Powerdown Sequence of Connect Application

Regarding to the specification of rBLE API, refer to Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

When exiting the application is requested in Advertising, the application calls RBLE_GAP_Broadcast_Disable in order to stop Advertising.

When exiting the application is requested in establishing connection, the application calls RBLE_GAP_Disconnect in order to disconnect.

When RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP event or RBLE_GAP_EVENT_DISCONNECT_COMP event occurs, the application calls RBLE_VS_Enable and RBLE_VS_RF_Control in order to power down the RF unit.

Project_Source\application\src\connect\r_connect.c, line 414-1085

```
 414:    static void con_gap_callback(RBLE_GAP_EVENT* evt)
 415:    {
 440:        switch(evt->type)
 441:        {
 462:            case RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP:
 463:                /* reach here after RBLE_GAP_Broadcast_Disable is called */
 464:                con_gap_broadcast_disable_eventhandler(evt);
 465:                break;
 470:            case RBLE_GAP_EVENT_DISCONNECT_COMP:
 471:                /* reach here when disconnection occurred */
 472:                con_gap_disconnection_eventhandler(evt);
 473:                break;
 489:        }
 490:    }
   :
 641:    static void con_gap_broadcast_disable_eventhandler(RBLE_GAP_EVENT* evt)
 642:    {
 643:        /* reach here after RBLE_GAP_Broadcast_Disable is called */
 644:
 647:            RBLE_VS_Enable(con_vs_callback);
 648:            RBLE_VS_RF_Control(RBLE_VS_RFCNTL_CMD_POWDOWN);
 650:    }
   :
 685:    static void con_gap_disconnection_eventhandler(RBLE_GAP_EVENT* evt)
 686:    {
 687:        /* reach here when disconnection occurred */
 688:
 760:            RBLE_VS_Enable(con_vs_callback);
 761:            RBLE_VS_RF_Control(RBLE_VS_RFCNTL_CMD_POWDOWN);
 768:    }
   :
1053:    static void con_vs_callback(RBLE_VS_EVENT* evt)
1054:    {
1076:        switch(evt->type)
1077:        {
1078:            case RBLE_VS_EVENT_RF_CONTROL_COMP:
1079:                /* reach here after RBLE_VS_RF_Control is called */
1080:                con_vs_rf_control_eventhandler(evt);
1081:                break;
1084:        }
1085:    }
```

8.2.3 DTM Application

(1) **Initializing & Transmitter Test & Receiver Test Sequence**

Figure 8-10 shows the sequence in Initializing state, Transmitter Test state and Receiver Test state of DTM Application. rBLE API is used in sequence between DTM Application and BLE Protocol Stack in the Sample Program.

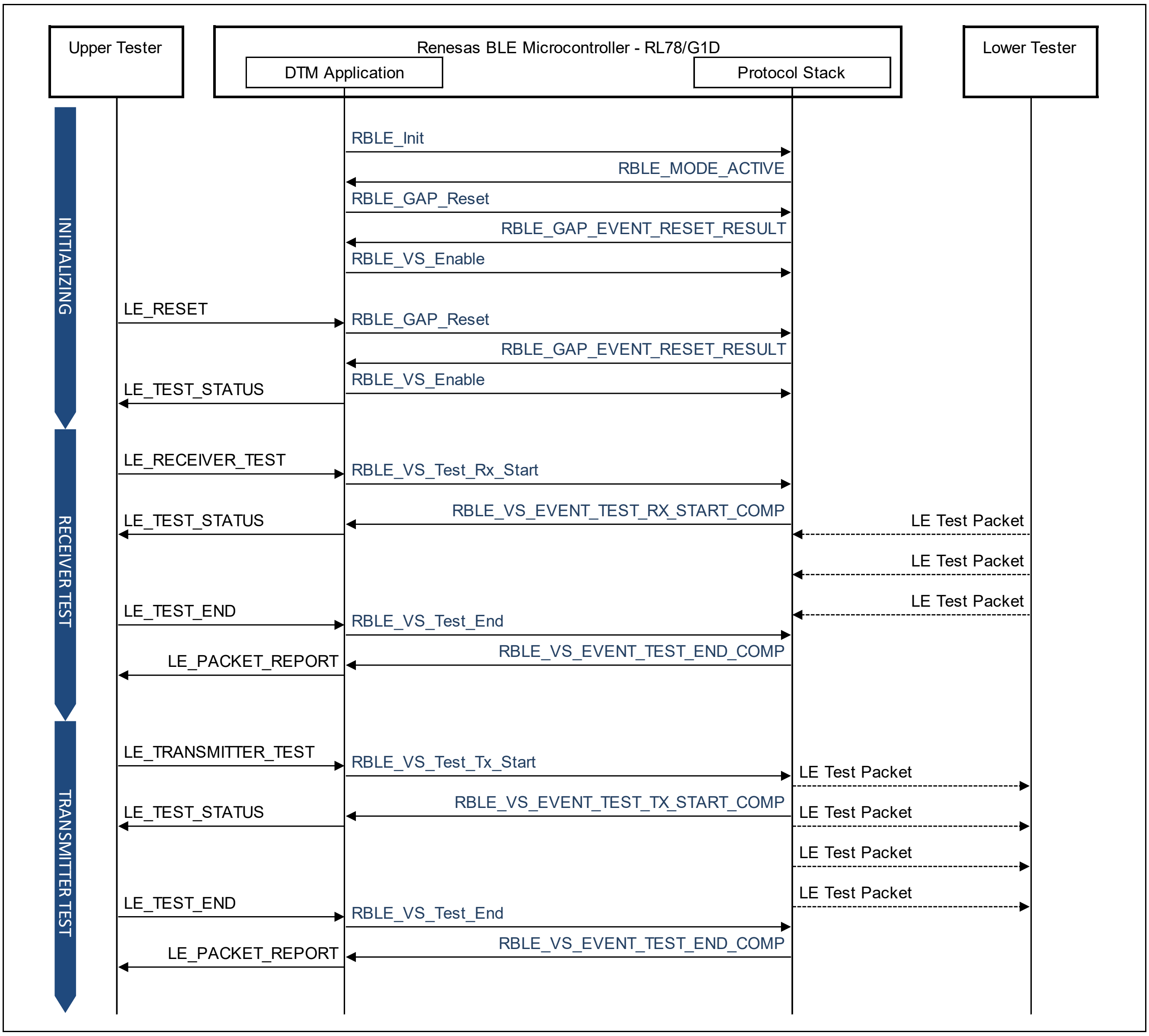


Figure 8-10 Initializing & Transmitter Test & Receiver Test Sequence of DTM Application

Regarding to the specification of rBLE API, refer to Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

DTM Application executes RF Test by receiving RF test command through UART. The application reports the result by transmitting RF test event through UART.

When LE_RECEIVER_TEST command is received, the application calls RBLE_VS_Test_Rx_Start in order to start receiving RF test packets from RF Tester.

In the same way, when LE_TRANSMITTER_TEST command is received, the application calls RBLE_VS_Test_Tx_Start in order to start transmitting RF test packets to RF Tester.

When LE_TEST_END command is received, the application calls RBLE_VS_Test_End in order to stop RF test.

Project_Source\application\src\connect\r_dtm.c, line 193-384

```
193:    static void dtm_cmdhandler(uint16_t cmd)
194:    {
201:        switch(val16 & DTM_CMD_MASK)
202:        {
203:            case DTM_CMD_RESET:
204:                if(RBLE_OK != RBLE_GAP_Reset(&dtm_gap_callback, &dtm_sm_callback))
205:                {
206:                    dtm_send_error();
207:                }
208:                break;
209:            case DTM_CMD_RX_START:
210:                if(RBLE_OK != RBLE_VS_Test_Rx_Start(DTM_GET_FREQ(val16)))
211:                {
212:                    dtm_send_error();
213:                }
214:                break;
215:            case DTM_CMD_TX_START:
216:                if(RBLE_OK != RBLE_VS_Test_Tx_Start(DTM_GET_FREQ(val16), DTM_GET_LENGTH(val16), … )
217:                {
218:                    dtm_send_error();
219:                }
220:                break;
221:            case DTM_CMD_END:
222:                if(RBLE_OK != RBLE_VS_Test_End())
223:                {
224:                    dtm_send_error();
225:                }
226:                break;
227:        }
228:    }
 :
329:    static void dtm_vs_callback(RBLE_VS_EVENT* evt)
330:    {
333:        switch(evt->type)
334:        {
335:            case RBLE_VS_EVENT_TEST_RX_START_COMP:
336:                /* reach here when RBLE_VS_Test_Rx_Start is called */
348:                break;
349:
350:            case RBLE_VS_EVENT_TEST_TX_START_COMP:
351:                /* reach here when RBLE_VS_Test_Tx_Start is called */
363:                break;
364:
365:            case RBLE_VS_EVENT_TEST_END_COMP:
366:                /* reach here when RBLE_VS_Test_End is called */
379:                break;
383:        }
384:    }
```

9. Appendix

9.1 Device Address

Device Address is 48-bit value for identifying each device. Device Address Types defined by Bluetooth Core Specification are shown as below.

- Public Device Address
    - ■ Public device address shall be created in accordance with section "48-bit universal LAN MAC addresses" of the IEEE 802-2001 standard and using a valid Organizationally Unique Identifier (OUI) obtained from the IEEE Registration Authority.
- Random Device Address
    - Static Device Address
        - ■ Static Device Address is a 48-bit randomly generated. Device may choose to initialize its address to a new value after each power cycle. And device shall not change its address value once initialized until the device is power cycled.
    - Private Device Address
        - Non-resolvable Private Address
            - ■ Non-resolvable Private Address is a 48-bit randomly generated. Its address should be changed over a period of time (recommended value of Bluetooth Core Specification is 15mins) for reducing the ability to track by other device.
        - Resolvable Private Address
            - ■ Resolvable Private Address contains 24-bit randomly generated number and 24-bit hash generated with randomly generated number and Identity Resolving Key (IRK). Its address should be changed over a period of time (recommended value of Bluetooth Core Specification is 15mins) for reducing the ability to track by other device.

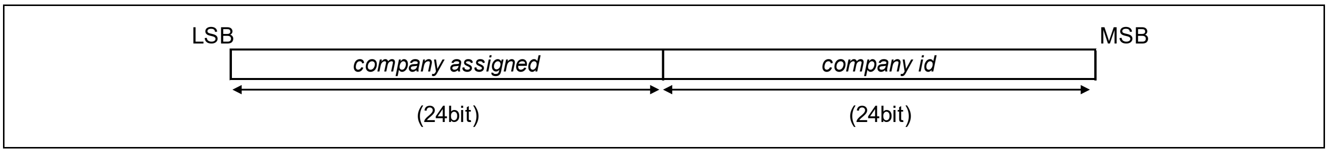


Figure 9-1 Public Device Address format

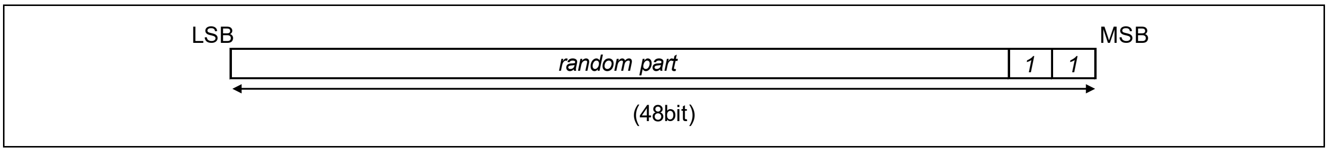


Figure 9-2 Static Device Address format

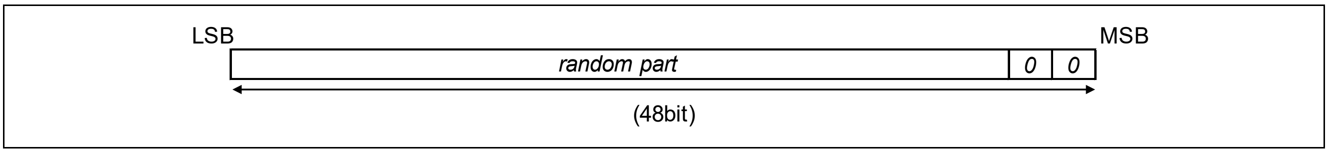


Figure 9-3 Non-resolvable Private Address format

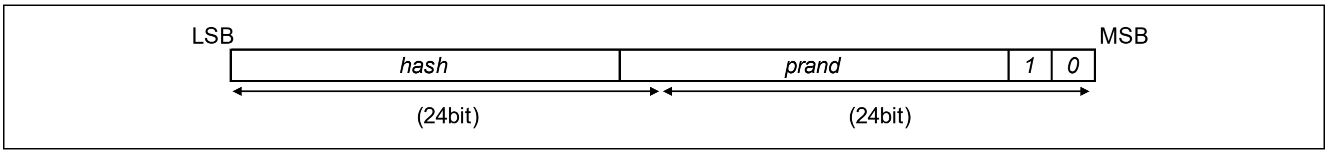


Figure 9-4 Resolvable Private Address format

Regarding to the specification of Device Address, refer to [Vol. 6, Part B] Section 1.3, Bluetooth Core Specification v4.2.

9.2 Advertising Packet Format

Beacon Application transmits non-connectable undirected advertising packet, and Connect Application transmits connectable undirected advertising packet in non-connected state. The packet format is common, and it is shown in **Figure 9-5**.

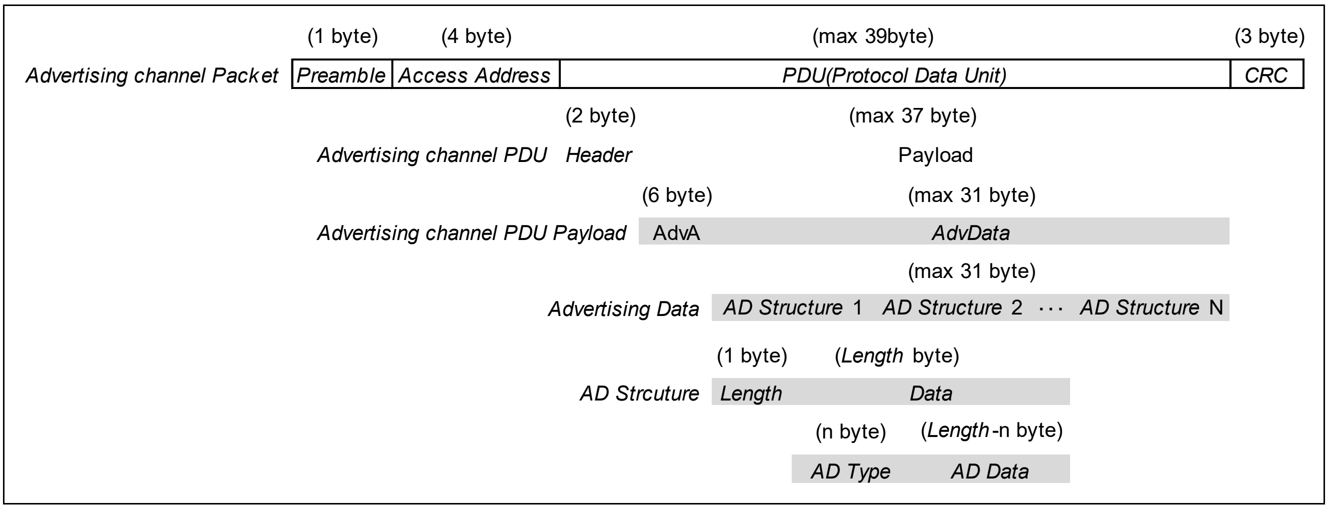


Figure 9-5 Advertising packet format

The specification of advertising packet is as shown below.

- advertising channel packet
  - Preamble : fixed 10101010b
  - Access Address : fixed 0x8E89BED6
  - Advertising channel PDU : Header and Payload
  - CRC : 24bits

Below fields are set by application.

- advertising channel PDU Payload
  - AdvA : Advertiser's Address is placed in
  - AdvData (Advertising data) : multiple AD structures are placed in, and maximum size is 31 bytes
    - AD structure : 1byte part of Length information and Length bytes part of Data
      - Data : n bytes part of AD Type and (Length-n) bytes part of AD Data

Regarding to the details, refer to below specifications respectively.

- advertising packet format : [Vol. 6, Part B] Section 2.1, Bluetooth Core Specification v4.2
- advertising channel PDU format : [Vol. 6, Part B] Section 2.3, Bluetooth Core Specification v4.2
- advertising data format : [Vol. 3, Part C] Section 11, Bluetooth Core Specification v4.2
- AD Type : Part A, Supplement to the Bluetooth Core Specification v7.

Regarding to the definitions of AD Type, refer to below website.

- Bluetooth SIG Home > Specification > Assigned Numbers > Generic Access Profile
  https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

9.3 Attribute Packet Format

Connect Application transmits Attribute packet in connected state. Attribute packet format which is used in Characteristic Value Read and Characteristic Value Write is shown in **Figure 9-6**.

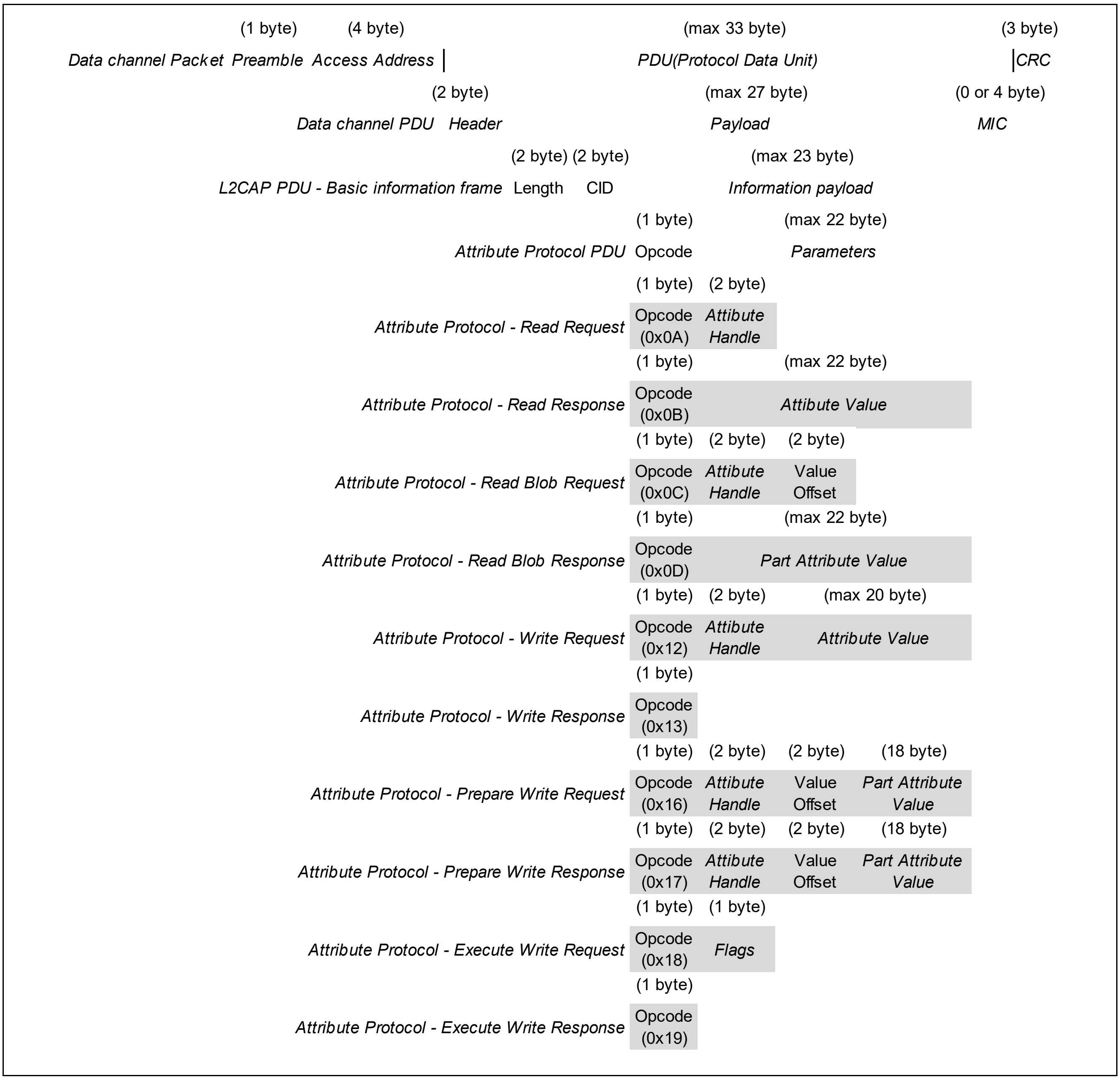


Figure 9-6 Attribute packet format

The specification of Attribute packet is as shown below.

- Data channel packet
  - Preamble : either 10101010b or 01010101b.
  - Access Address : determined by Master device when establishing connection
  - Data channel PDU : consists of Header, Payload and Message Integrity Check (MIC)
    MIC is added if data is encrypted
  - CRC : 24bits

- L2CAP PDU
  - consists of Length, Channel ID (CID) and Information payload.
  - CID is 0x0004 (Attribute Protocol)
- Attribute Protocol PDU
  - consists of Attribute Opcode and Attribute Parameters
  - Attribute Opcode specifies Attribute operation
  - Attribute Parameters is different from each Attribute operation

Regarding to the details, refer to below specification

- Data channel PDU : [Vol. 6, Part B] Section 2.4, Bluetooth Core Specification v4.2
- L2CAP PDU : [Vol. 3, Part A] Chapter 3, Bluetooth Core Specification v4.2
- Attribute Protocol PDU : [Vol. 3, Part F] Section 3.4, Bluetooth Core Specification v4.2
- GATT Features : [Vol. 3, Part G] Chapter 4, Bluetooth Core Specification v4.2

9.4 Specification Changes

Table 9-1 shows major specification changes.

Table 9-1 Specification Changes between Rev.1.00 and Rev.1.10

Item		Rev.1.00	Rev1.10	Position
Environment Version				
	CC-RL	V1.02.00	V1.04.00	Chapter2
	CS+ for CC	V3.03.00	V5.00.00	Chapter2
	e2 studio	Version 4.3.0.008	Version 5.2.0.023	Chapter2
	BLE Protocol Stack	V1.11	V1.20	Chapter2
	Beacon Stack	V1.00	V2.10	Chapter2
Beacon Application				
	RF Operation	RF Transmission only	RF Transmission only RF Transmission and Reception	Subsection 6.1.2
	Advertising Type	Non-connectable Undirected Advertising	Non-connectable Undirected Advertising Scannable Undirected Advertising	Subsection 5.1.1
Connect Application				
	Pairing Initiator Key Distribution	ID Resolving key	None	Subsection 5.2.2
	Custom Service Characteristic	Advertising Information Advertising Data Code Flash Memory Updated Count Data Flash Memory Updated Count	Advertising Information Advertising Data Scan Response data Code Flash Memory Updated Count Data Flash Memory Updated Count	Subsection 5.2.3
Application Sequence				
	Application Switching	Beacon Application starts, and then either Beacon Application and Connect Application switches alternately when switch is pushed.	Beacon Application starts, and then either Beacon Application and Connect Application switches alternately when switch is pushed. Connect Application starts, and then switch to Beacon Application after 30seconds.	Subsection 6.1.10
Stored Data in Flash memory				
	Code Flash	Device Address Device Address Type Device Name Advertising Information Non-connectable Undirected Advertising Data	Device Address Device Address Type Device Name Advertising Information Non-connectable Undirected Advertising Data Scanable Undirected Advertising Data Scan Response Data	Subsection 5.4.1
	Data Flash	Pairing Information Peer Device Address Peer Device Address Type Security Status Local Encryption Keys Remote Encryption Keys Random Seed Value Data Flash Updated Count Code Flash Updated Count	Pairing Information Peer Device Address Peer Device Address Type Security Status Local Encryption Keys Data Flash Updated Count Code Flash Updated Count	Subsection 5.4.2

Website and Support

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

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

Revision History of Preceding Editions

Rev.	Date	Description	
1.00	Jul. 14, 2016	First edition issued	
1.10	Jun. 30, 2017	Chapter 1 Overview	
		P.5	figure of Beacon System and figure of RF Evaluation System are merged into Figure 1-1
		P.6	Figure 1-2 and Figure 1-3 are changed
		-	content about the evaluation board operation is moved to Chapter 4
		Chapter 2 Environment	
		P.7	Software Environment and Software Library versions are updated
		Chapter 3 File Composition	
		P.8	R5F11AGJ_BcnCmb_no_sw.hex is added in file composition
		Chapter 4 Evaluation Procedure	
		P.10	Current Consumption Measurement is added in evaluation procedure
		P.11	Library download URLs are changed
		P.13	content about the evaluation board operation and Figure 4-1 are added
		P.15	content about the evaluation board operation and Figure 4-2 are added
		P.15	flow chart using smart phone is merged into flow chart of P.10
		P.16	figure of Confirming the transmission of advertising packet is added
		P.17	figure of Updating the advertising packet is added
		P.17	procedure is changed for evaluating with iOS GATTBrowser
		P.19	procedure is changed for evaluating with Android GATTBrowser
		P.21	figure of Confirming the updated advertising packet is added
		P.22	figure of Evaluating RF characteristic is added
		P.23	Section 4.6 is added
		Chapter 5 Specification	
		P.25	Operation when both Tx and Rx are enabled is added
		P.26	Table 5-2 is added
		P.28	Initiator Key Distribution is changed to None in Table 5-4
		P.30	Scan Response Data Characteristic is added in Table 5-5
		P.30	Attribute Handle are changed in Table 5-5
		P.31	RF Test Commands /Events Table is separated into Table 5-6 and Table 5-7
		P.31	Combinations of RF Test Commands / Events Table is removed
		P.32	format of Table 5-8 is changed
		P.32	Scannable Advertising Data and Scan Response Data are added in Table 5-8
		P.33	Structure of Pairing Information and Flash memory count are changed in Table 5-9
		P.33	Random Seed is removed in Table 5-9
		P.35	Section5.6 is added
		P.36	Compiler version is updated
		P.37	size of System Configuration is changed in Figure 5-8, Figure 5-9, and Figure 5-10
		Chapter 6 Configuration	
		P.44	Subsection 6.1.10 is added
		P.46	Scan Response Data is added in unique code file
		P.47	Subsection 6.2.2 is added
		P.51	Subsection 6.2.4 is added
		P.58	Subsection 6.2.7 is added

		Chapter 9 Appendix	
		-	content about building Android application is removed
		P.87	Section 9.4 is added
		Overall	
		-	"Beacon System" is changed into "Beacon Operation"
		-	"RF Evaluation System" is changed into "RF Evaluation Operation"
		-	Related documents to be referred are added
		-	typos of term, symbol name, and other are modified
1.11	2018.03.30	Chapter 5 Specification	
		P.34	Supplementation of Table 5-12 is changed.
		Chapter 9 Appendix	
		P.84	Description in Section 9.2 is modified.
		P.85	Description in Section 9.3 is modified.

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

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

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   ¾ 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.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   ¾ 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.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   ¾ 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.

4. Clock Signals

   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.

   ¾ 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.

5. Differences between Products

   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.

   ¾ 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.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.1