RX23W Group

Bluetooth Low Energy Application Developer's Guide

Introduction

This application note describes how to develop a Bluetooth® Low Energy (Hereinafter referred to as "Bluetooth LE" or "BLE") application.

"BP:" in the text describes recommendations and risks based on the guideline (Bluetooth® Security and Privacy Best Practices Guide) published by the Bluetooth SIG so that implementers can select best practices for security and privacy. Please refer to it.

Target Device

RX23W Group

Related Documents

- Bluetooth Core Specification (https://www.bluetooth.com)
- Bluetooth® Security and Privacy Best Practices Guide (https://www.bluetooth.com)
- RX23W Group Bluetooth Low Energy Profile Developer’s Guide (R01AN6459)
- Bluetooth Low Energy Protocol Stack Basic Package: User's Manual (R01UW0205)
- BLE Module Firmware Integration Technology (R01AN4860)

Contents

1. Overview 7
1.1 Development Bluetooth Low Energy Application 7
1.2 Development environment 9
1.2.1 Hardware requirements 9
1.2.2 Software requirements 10
1.2.3 Tool 11
1.3 Available communication features 12
1.4 Basic communication features 14
1.4.1 Device identification 16
1.5 Bluetooth LE Protocol Stack Operation Overview 17
1.6 Software structure 19
1.6.1 Primary functions 20
1.6.2 Surrounding functions 24
1.7 Flow of development 25
1.8 Use case of this document 27
1.9 Locating sections 28

2. Adjusting configuration option 29
2.1 Configuration Options 29
2.2 How to adjust RAM 32
2.3 How to configure BD address 33
2.3.1 Writing to data area 35
2.3.2 How to use Random Address 35
2.4 How to configure for minimum current consumption 37
2.4.1 Using MCU Low Power Consumption function 38

3. How to implement user code 41
3.1 Behavior of skeleton program 42
3.2 app_main function 43
3.2.1 Initialize process (ble_app_init function) 44
3.2.2 Main loop and scheduler (R_BLE_Execute) 47
3.2.3 End process 48
3.3 GAP event (gap_cb function) 49
3.4 GATTS event (gatts_cb function) 51
3.5 GATTC event (gattc_cb function) 52
3.6 VS event (vs_cb function) 53
3.7 Server-side Profile API event ([service_name]s_cb function) 54
3.8 Client-side Profile API event ([service_name]c_cb function) 55
3.9 L2CAP event 56
3.10 Event notification function (R_BLE_SetEvent) 57

3.11 RF communication timing notification 59

4. app_lib 63
4.1 Software Timer 63
4.2 Command line 67
4.2.1 How to use the standard commands 68
4.2.2 How to create a user command 71
4.3 Logger 75
4.4 Security data management 77
4.4.1 Initialization 77
4.4.2 Restore the local device keys 77
4.4.3 Store the local device keys 77
4.4.4 Store the remote device keys 78
4.5 Board and LED switch 79
4.5.1 Configuration for customer board 80
4.5.2 Initialization 82
4.5.3 ON or OFF Board LED 82
4.5.4 Callback for pressing SW 82
4.6 Abstraction API 83

5. Advertising 85
5.1 Connecting to smartphone 85
5.2 Advertising with GAP API 86
5.2.1 Advertising Parameter 86
5.2.2 Advertising Data / Scan Response Data 89
5.2.3 Start Advertising 89
5.2.4 Stop Advertising 89
5.3 Periodic Advertising with GAP API 90
5.3.1 Non-Connectable Advertising Parameter 91
5.3.2 Periodic Advertising Parameter 91
5.3.3 Periodic Advertising Data 92
5.3.4 Start Periodic Advertising 92
5.3.5 Stop Periodic Advertising 94
5.4 Advertising Data / Scan Response Data / Periodic Advertising Data 95
5.4.1 Format 95
5.4.2 Advertising Data Update 98
5.4.3 Periodic Advertising Data Update 98
5.4.4 Buffer Size 99
5.5 Advertising with Abstraction API 100
5.5.1 White List (Respond to a known device) 100
5.5.2 Privacy 101
5.6 Connection with Smart Phone 102

5.7 Beacon 103

6. Scan 104
6.1 Start or stop scan 104
6.2 Scan parameters 104
6.2.1 Privacy 106
6.3 Received information by scan 108
6.4 Scan filtering 110
6.4.1 Using the White List (Receiving from known devices) 110
6.4.2 Duplicate advertising filtering 111
6.4.3 Discoverable mode filtering 111
6.4.4 Advertising Data filtering 111
6.5 Periodic Advertising Synchronization 112
6.5.1 Start Scan 113
6.5.2 Detect Periodic Advertiser 113
6.5.3 Register to the Periodic Advertiser List 113
6.5.4 Establish Periodic Advertising Sync 113
6.5.5 Receive Periodic Advertising 115
6.5.6 Lost Periodic Advertising Sync 115
6.5.7 Terminate Periodic Advertising Sync 115

7. Connection 116
7.1 Requesting Connection 116
7.1.1 Using the White List (Connection to a known device) 117
7.1.2 Privacy 118
7.2 Cancelling Connection Request 119
7.3 Multiple Connection 120
7.3.1 Connecting to multiple peripheral devices 121
7.3.2 Connection to multiple central devices 126
7.3.3 Multi role connection 130
7.4 Disconnection 135

8. Communication 136
8.1 Changing PHY 136
8.2 Changing maximum transmission packet length 139
8.3 Updating connection parameter 141
8.4 Changing MTU 146
8.5 Flow control 148
8.6 High throughput communication 149

9. Security 150
9.1 Pairing 150
9.1.1 Pairing Parameters 152

9.1.2 Key generation and registration 156
9.1.3 OOB (Out of Band) data transmission and reception 156
9.1.4 Pairing request 157
9.1.5 Response to pairing request 157
9.1.6 Carrying out pairing method 158
9.1.7 Key exchange 159
9.1.8 Completion of pairing 160
9.2 Bonding 161
9.2.1 Store remote device keys 162
9.2.2 Store local device keys 166
9.2.3 Reset the stored keys 166
9.2.4 Delete the stored keys 166
9.2.5 Filtering remote devices after bonding 167
9.3 Encryption 168
9.3.1 Request Encryption 168
9.3.2 Respond to an encryption request 170
9.4 Privacy 173
9.4.1 Generate local device RPA 175
9.4.2 Resolve remote device RPA 178

10. Profile and service 185
10.1 Standard profile and Standard Service 186
10.2 APIs of GATT Procedure 192
10.2.1 Read operation 192
10.2.2 Write operation 193
10.2.3 WriteWithoutResponse operation 194
10.2.4 Notification operation 195
10.2.5 Indication operation 197
10.2.6 ReliableWrites operation 199
10.2.7 Broadcast Operation 201
10.3 Example of using GATT Procedure 203
10.3.1 Example for sending data from GATT client 203
10.3.2 Example for sending data from GATT server 206

11. Debugging 208
11.1 Using Logger function 209
11.2 Using Command line function 211
11.3 Using RF communication timing notification function 213
11.4 Checking Server operation 218
11.4.1 Using BTTS Beacon Scanning 218
11.4.2 Using BTTS Data Comm Master 219
11.4.3 Using GATT Browser 219

11.5 Checking Client operation 220
11.5.1 Using BTTS Beacon Advertising 220
11.5.2 Using BTTS Data Comm Slave 221
11.6 Others 223
11.6.1 MCU package 223
11.6.2 Generating MOT file 224
11.6.3 Outputting detail to MAP file 225
11.6.4 Optimization 225

12. Appendix A : Sample applications 226
12.1 Beacon sample 229
12.1.1 Remote devices 229
12.1.2 Operations 229
12.1.3 Advertising Data 229
12.1.4 Configuration option 231
12.1.5 Configurable parameters 231
12.1.6 Command 231
12.2 Peripheral sample 232
12.2.1 Remote devices 232
12.2.2 Operations 232
12.2.3 Configuration option 234
12.2.4 Configurable parameters 234
12.3 Central sample 235
12.3.1 Remote devices 235
12.3.2 Operations 235
12.3.3 Configuration option 235
12.3.4 Configurable parameters 236
12.4 Multi-role sample 237
12.4.1 Topology 237
12.4.2 Remote devices 238
12.4.3 Operations 238
12.4.4 Configuration option 241
12.4.5 Configurable parameters 242

Revision History 243

1. Overview

1.1 Development Bluetooth Low Energy Application

There are two methods of data communication using Bluetooth Low Energy: connectionless manner and connection manner. For mesh communication using connectionless manner, refer to "Bluetooth Mesh Stack Package Startup Guide (R01AN4874)" and “Bluetooth Mesh Developer Guide (R01AN4875)”.

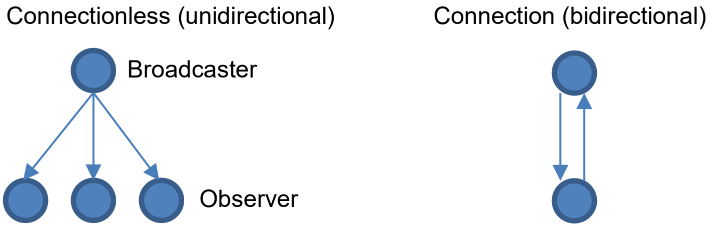


Figure 1-1 Image of Bluetooth LE data communication method

In the connectionless manner, the application data is sent in an advertisement packet. The receiving device receives the advertisement packet by scanning. The Application perform this communication with the Generic Access Profile (GAP) for device detection and connection. With this method, the data is unidirectional communication from the broadcaster to the observer. Since no device is connected, the advertisement packet can be received by any device.

The connection manner is used for bidirectional communication. The connection manner connects devices by GAP. Application data is sent and received by Generic Attribute Profile (GATT). GATT provides communication by the server-client architecture on the communication path of GAP.GATT performs data communication according to the application profile.

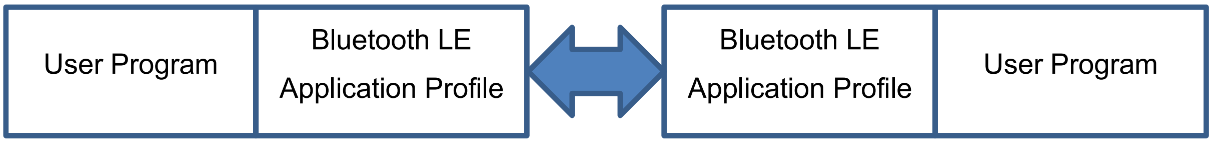


Figure 1-2 Bluetooth LE bidirectional communication

For the application that assumes using Bluetooth LE, Bluetooth SIG publishes the application profiles as specifications. By implementing this application profile, a device can interconnect with existing devices that are already working. When developing a new bidirectional communication application, design the application profile as well as the user program.

The application profile defines the structure of application data exchanged between GATT server and clients and the method of accessing the database, the setting of communication parameters by GAP, the method of connecting devices, and the setting of security.

BP: Support for authentication and encryption is recommended when there are modifiable GATT characteristics (e.g. a door lock mechanism where the remote device manipulates the lock state by changing the value of the attribute), including custom profile.

This document describes how to implement a program for performing Bluetooth LE communication and information that is a hint for application profile developing.

Renesas provides tools to assist with Bluetooth LE application development.

(1) BLE FIT Module

It provides the Bluetooth LE feature that complies with the Bluetooth Core Specification version 5.0 defined by Bluetooth SIG. You can add to your project from Smart Configurator on e^{2}studio and start Bluetooth LE application development.
The Bluetooth LE feature is provided in library format as a Bluetooth LE Protocol Stack. Bluetooth LE operation is performed by using the API. The Bluetooth LE Protocol Stack notifies the application of events related to Bluetooth LE by a callback function to reduce power consumption.
BLE FIT module provides application library (app_lib) to assist application development in addition to Bluetooth LE Protocol Stack. By using app_lib, you can easily realize the basic operation of Bluetooth LE.

(2) QE for BLE, QE Utility Module

QE for BLE is a QE tool for designing application profiles with GUI and code generation. Code generation is performed based on the template file provided by the QE Utility module. The QE Utility module is provided in the FIT module format or the bundled format for QE for BLE V1.40 or later.
By using these tools, the GATT part of the application profile is designed from the GUI and the API (service API) for realizing the profile is generated. It is possible to generate not only the designed profile but also the application profile API exposed to the Bluetooth SIG.

Finally, an example of the Bluetooth LE application development process and use of the Renesas tool is shown.

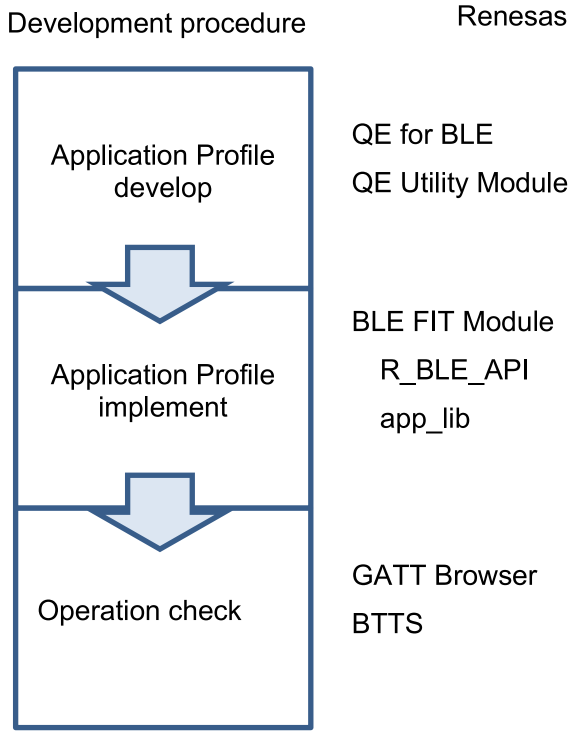


Figure 1-3 Bluetooth LE application development procedure and auxiliary tools

1.2 Development environment

1.2.1 Hardware requirements

Table 1.1 shows the hardware requirements for building and debugging the application.

Table 1.1 Hardware requirements

Hardware	Description
Host PC	Windows PC with USB interface.
MCU board	The board with RX23W Target Board for RX23W [RTK5RX23W0C00000BJ] or RSSK RX23W [RTK5523W8AC00001BJ] or Target Board for RX23W module [RTK5RX23W0C01000BJ] Note: This document uses Target Board for RX23W for explanation.
On-chip debugging emulators	E2 emulator [RTE0T00020KCE00000R] or E2 emulator Lite [RTE0T0002LKCE00000R] or E20 emulator [R0E000200KCT00] or E1 emulator [R0E000010KCE00] When using the RSSK, either emulator is required. The Target Board has an on-board debugger equivalent to the E2 emulator Lite, so there is no need to prepare an emulator. Note: The E1 emulator is discontinued. Note: This document uses E2 emulator Lite for explanation.
USB cables	Used to connect the PC to the emulator and RX23W board. E2 or E1 emulator: 1 USB A ↔ mini-B cable Target Board: 2 USB A ↔ micro-B cable RSSK: 1 USB A ↔ micro-B cable

1.2.2 Software requirements

Table 1.2 shows the software requirements for building and debugging the application.

Table 1.2 Software requirements

Software		Version	Description
CC-RX environment	e^2 studio	v7.6.0 or later	Integrated development environment (IDE) for Renesas devices. Note: This document uses e^2studio for explanation.
	CS+ for CC	V8.02.00 or later	Integrated development environment (IDE) for Renesas devices. Note: It is recommended using e^2studio because CS+ is not support QE for BLE.
	CC-RX compiler	V2.08.00 or later	C/C++ compiler. (Download from e^2 studio installer)
	QE for BLE[RX]	v1.0.0 or later	A plugin for e^2studio to generate skeleton programs for application and profile development. Note: QE for BLE[RX] has been integrated into QE for BLE[RA,RE,RX].
	QE for BLE[RA,RE,RX] QE for BLE[RA,RE,RX] Utility	V1.4.0 or later	
	BLE FIT module (r_ble_rx23w)	v1.10 or later	Required to develop Bluetooth Low Energy applications with Renesas MCUs. Note: When using BLE FIT module v2.50 or later, use QE for BLE[RA,RE,RX] Utility v1.6.0 or later.
	BSP FIT module (r_bsp)	v5.40 or later	Required to use BLE FIT module. When using BLE FIT module version 1.01 or later, it is necessary to change the version of r_bsp to 5.40 or later.
	CMT FIT module (r_cmt_rx)	v4.10 or later	Required to use BLE FIT module. Bluetooth LE Protocol Stack uses CMT2 and CMT3. When using the software timer function of app_lib/timer, more 1 channel is used.
	LPC FIT module (r_lpc_rx)	v1.42 or later	When using the MCU low power consumption function with the BLE FIT module, use v1.42 or later.
	Flash FIT module (r_flash_rx)	v4.10 or later	When using the device-specific data management function of the optional function with the BLE FIT module, use v4.10 or later.
IAR environment	IAR Embedded Workbench for Renesas RX	v4.12.1 or later	Integrated development environment (IDE) for Renesas devices made by IAR Systems. Note: Supported by Bluetooth LE Protocol Stack v1.10 or later.
	IAR C/C++ Compiler for Renesas RX version	v4.12.1 or later	C/C++ compiler made by IAR Systems.
	QE for BLE[RX] or QE for BLE[RA,RE,RX]	Same with CC-RX environment	Used by changing from the e^2studio created project to IAR project. As for procedure, refer to “4.9 Create a project on the IAR development environments” in “BLE Module Firmware Integration Technology Application Note (R01AN4860)”.
Renesas Flash Programmer		V3.06.00 or later	Tool for programming the on-chip flash memory of Renesas microcontrollers.
Integer types			Uses ANSI C99 “Exact width integer types” in order to make the code clearer and more portable. These types are defined in stdint.h.
Endian			Little endian.

1.2.3 Tool

Application development is supported by the following tools.

Table 1.3 Supporting tools for application development

Tool	Description
GATT Browser	Smartphone application to access to GATT Server. Bluetooth Low Energy basic communication operation and GATT database structure and so on can be confirmed by smartphone.
BTTS	Tool suite to control RX23W connected with Windows PC and USB Serial and evaluate three functions of RF, Beacon and Data Communication in Bluetooth Core Specification 5.0. It can be also used when getting the Radio Law Certification for the device.

1.3 Available communication features

RX23W supports Bluetooth Low Energy (LE) features shown in Table 1.4 and can communicate with the devices that have LE features.

Table 1.4 LE features

Bluetooth version	LE features and description	Remark
5.0	LE 2M PHY (2 Msym/s PHY for LE) 2Mbps PHY data rate.	High data throughput. Low power consumption by short communication time.
5.0	LE Coded PHY (LE Long Range) 500kbps/125kbps PHY data rate.	Extend communication distance.
5.0	LE Advertising Extensions Enable Advertising by secondary channel. (Up to 4 independent Advertising can be executed simultaneously in RX23W.) Expansion of Advertising Data/Scan Response Data size up from 31 bytes to 1650 bytes. Advertising by Long Range. Periodic Advertising is possible.	Wireless interference reduction. Beacon information expansion. Establishing connection in long-distance. Utilization of secondary channel.
5.0	LE Channel Selection Algorithm #2 Improving the channel hopping algorithm.	Wireless interference reduction.
5.0	High Duty Cycle Non-Connectable Advertising Shorten minimum Advertising Interval (100ms→20ms).	Shortening the time to connect. Higher frequency of beacon transmission.
4.2	LE Data Packet Length Extension Expand the data communication packet size (27 bytes→251 bytes).	High data throughput. Low power consumption by short communication time.
4.2	LE Secure Connections Support the pairing with the Elliptic curve Diffie-Hellman (ECDH) key exchange for passive eavesdropping protection.	Enhanced security.
4.2	Link Layer Privacy Link Layer supports address resolution of Privacy feature.	Faster address resolution.
4.2	Link Layer Extended Scanner Filter Policies	
4.1	Low Duty Cycle Directed Advertising Support Low Duty Cycle Advertising for reconnection with known devices.	
4.1	32-bit UUID Support in LE Support 32-bit UUID (extended to 128-bit when used by GATT).	
4.1	LE L2CAP Connection-Oriented Channel Support Support the communication using L2CAP credit based flow control channel.	
4.1	LE Privacy v1.1 Avoid the tracking from other LE devices by changing the BD Address periodically.	Enhanced security.
4.1	LE Link Layer Topology Support both Central and Peripheral roles, and can operate as Central when connecting to one remote device and as Peripheral when connecting to another remote device.	Enhanced topology.
4.1	LE Ping Checks whether connection is maintained by a packet transmission request including MIC field after connection encryption.	
Addendum 2	Appearance Data Type Appearance characteristic can be used in GAP service.	

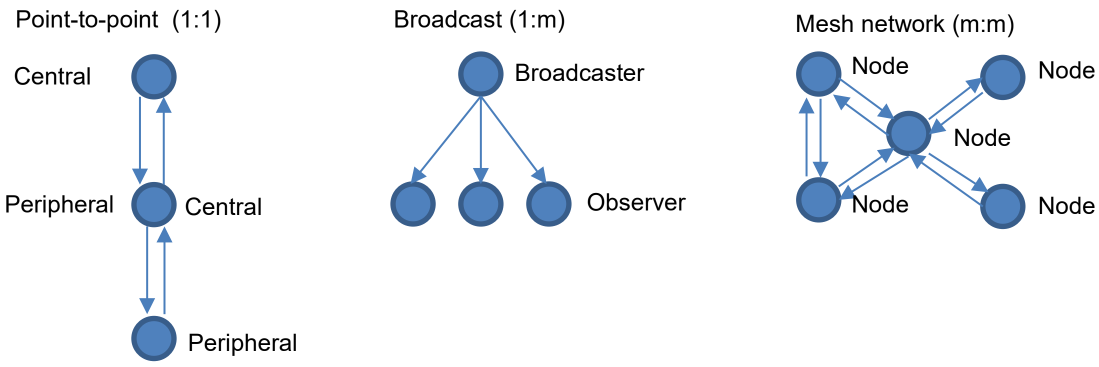
Bluetooth version	LE features and description	Remark
4.0	Bluetooth Low Energy - Low Energy Controller Low Energy Physical Layer (PHY) Low Energy Link Layer (LL) - Low Energy Host Enhancements to L2CAP for Low Energy Security Manager (SM) - Enhancements to HCI for Low Energy - Low Energy Direct Test Mode - AES Encryption - Enhancements to GAP for Low Energy - Attribute Protocol (ATT) - Generic Attribute profile (GATT)	Low Energy Controller is mandatory feature. Low Energy Host is mandatory feature. ATT is mandatory feature. GATT is mandatory feature.

Note: BR/EDR (Basic Rate/Enhanced Data Rate) is not supported.

Note: The features except mandatory feature is optional feature (vendor dependent), so they may be not supported by devices such as smartphone and so on.

1.4 Basic communication features

The communication topology that can be constructed by the device that have LE features shown in Figure 1-4.

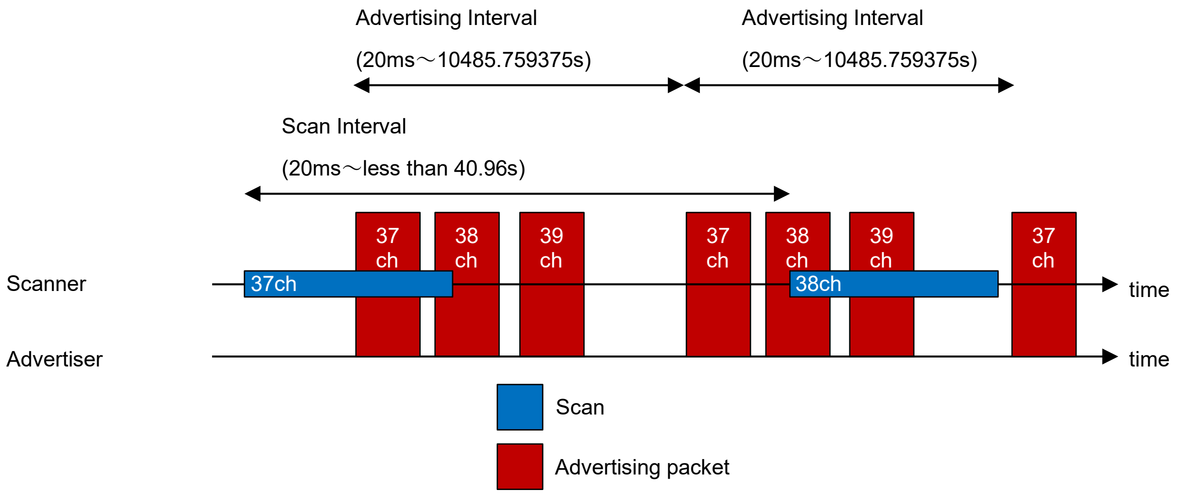


Note: Up to 7 units can be connected simultaneously in RX23W.

Note: Up to 4 sets can be advertised simultaneously in RX23W.

Figure 1-4 Communication topology

In Broadcast, the communication is performed without establishing Connection. Broadcaster (Advertiser) executes Advertising and sends packets, and Observer (Scanner) executes Scan and receives packets.



Note: In actually, the random delay of 0 to 10 ms is added to Advertising Interval for each Advertising.

Figure 1-5 Advertising and Scan

In Point-to-point, the communication is performed with establishing Connection. Peripheral (Advertiser) executes Advertising and sends packets, and Central (Scanner) executes Scan and receives packets. One device requests Connection to the device wanted to connect to as the Initiator, and the other device accepts and Connection is established. Initiator becomes Central and the other becomes Peripheral. Once Connection is established, Data communication is possible.

GAP (Generic Access Profile) commands control from Advertising and Scan to establishing Connection. GATT (Generic Attribute Profile) commands control Data communication after establishing Connection. In GATT, the side that provides services by storing the sensor data and so on as GATT database is called Server, and the side that requests the service is called Client. Client can read and write to Server that has the database. Server can do Indication and Notification to Client. When Client receives Indication, Client returns the response by executing Confirmation. The following is an example when Central is Client and Peripheral is Server.

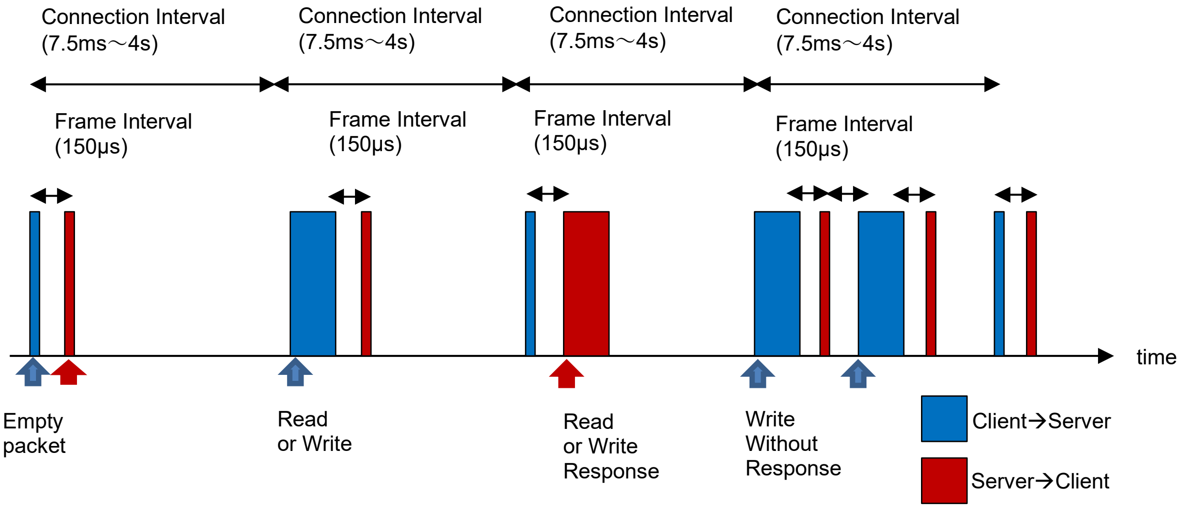


Figure 1-6 Read and Write

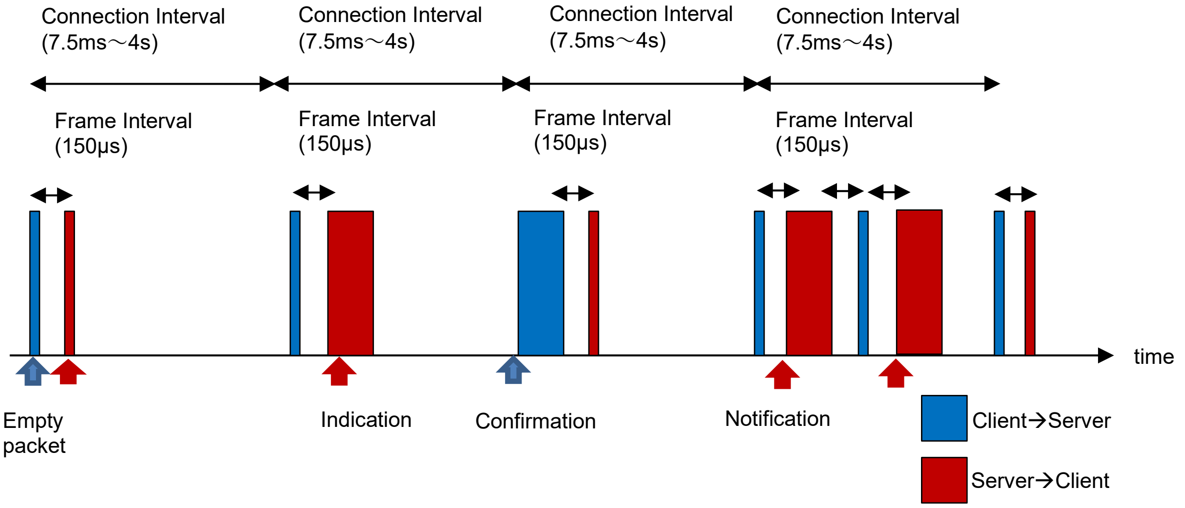


Figure 1-7 Indication and Notification

Advertising is described in “5 Advertising”. Scan is described in “6 Scan”. Connection is described in “7 Connection”. Data communication is described in “8 Communication”.

Note: As for Mesh network, refer to “Bluetooth Mesh Stack Package Startup Guide (R01AN4874)”.

1.4.1 Device identification

Devices are identified using Bluetooth Device address (BD address). BD address is described in “2.3 How to configure BD address”.

You can additionally use Local Name of Advertising Data and Device Name of GAP service. Local Name is shown in “Table 5.7”. Device Name is shown in “Table 10.2”.

1.5 Bluetooth LE Protocol Stack Operation Overview

The Bluetooth LE Protocol Stack controls the BLE peripheral functions and manages the execution of RF events. RF event refers to one communication operation at each interval in the following four operation states specified by Bluetooth LE.

- Advertising
- Scanning
- Initiating
- Connection

The Bluetooth LE Protocol Stack provides the control interface for Bluetooth LE operation as R_BLE API. The BLE peripheral functions generate an interrupt (BLEIRQ) corresponding to an RF event to the MCU. When BLEIRQ occurs, it is necessary to call *R_BLE_Execute* and perform task processing according to the RF event status. Also, when various R_BLE APIs are called, it is necessary to call *R_BLE_Execute* to perform API task processing of the Bluetooth LE Protocol Stack.

When BLE_CFG_RF_DEEP_SLEEP_EN is set to 1 in “2.1 Configuration Options”, when there is no task to be executed by the Bluetooth LE Protocol Stack, and when there is a time of 80ms or more before the start of the next RF event time, transition to RF sleep mode to reduce the current consumption of the RF part. This time does not mean the "interval time" of an RF event, but the "RF idle time" between the completion of one RF event and the start of the next RF event. Therefore, it is necessary to set the RF event interval to 100ms or more in consideration of the processing time of each layer in order to shift the RF part to sleep mode. In Scanning operation, the time difference between the Scan interval and Scan window must also be set to 100ms or more.

The Bluetooth LE Protocol Stack performs RF sleep processing and RF wake-up processing to transition the RF part to sleep mode. Figure 1-8 shows MCU/RF operation overview with RF sleep.

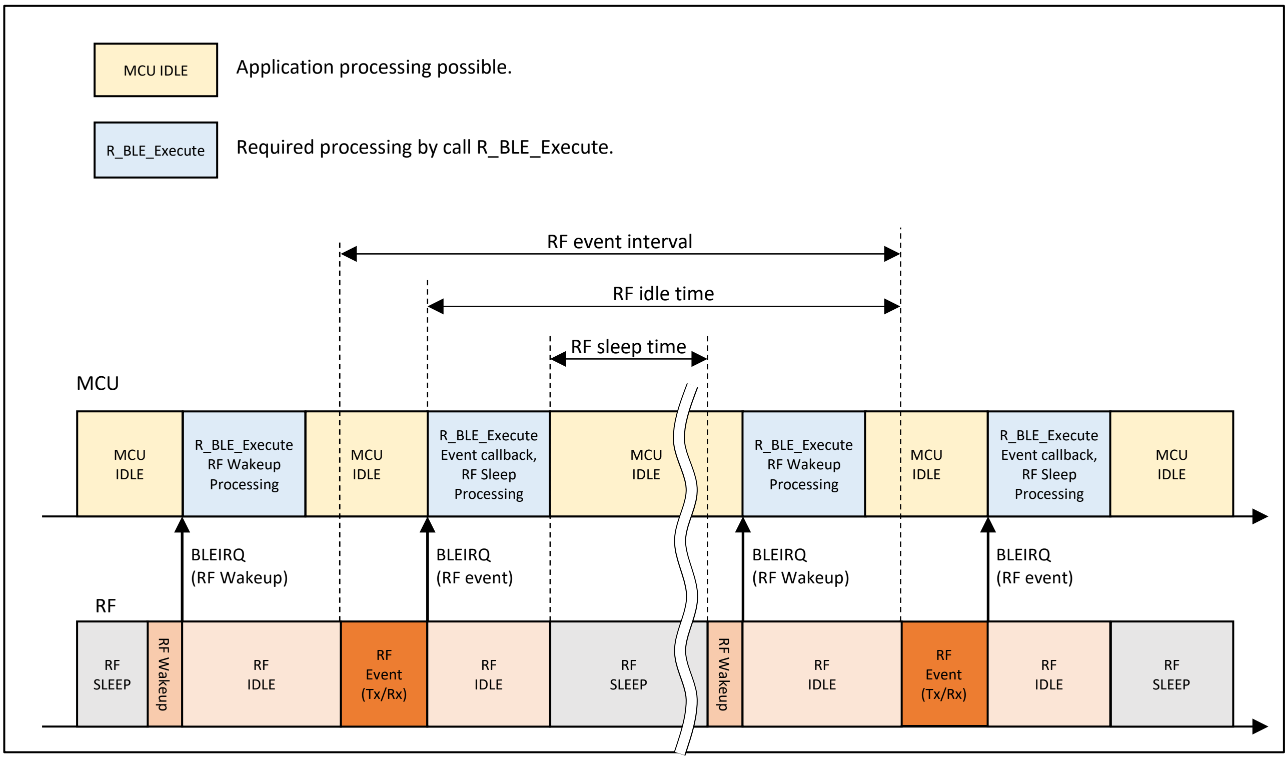


Figure 1-8 MCU/RF operation overview with RF sleep

While the MCU is idle, it is possible to transition the MCU to the low power consumption mode or execute processing of the other application. However, if the RF wakeup process by *R_BLE_Execute* is not performed before the RF event starts, the RF event cannot be executed. Therefore, application processing must be implemented so as not to interfere with the *R_BLE_Execute* call.

When BLE_CFG_RF_DEEP_SLEEP_EN is set to 0 in “2.1 Configuration Options ”, or when BLE_CFG_RF_DEEP_SLEEP_EN is set to 1 but the RF sleep transition condition is not satisfied, the Bluetooth LE Protocol Stack dose not transition RF part to sleep mode. In this case, the current consumption during RF idle time increases, but the MCU idle time that can be used by the application increases because RF sleep processing and RF wakeup processing are not performed. Figure 1-9 shows MCU/RF operation without RF sleep.

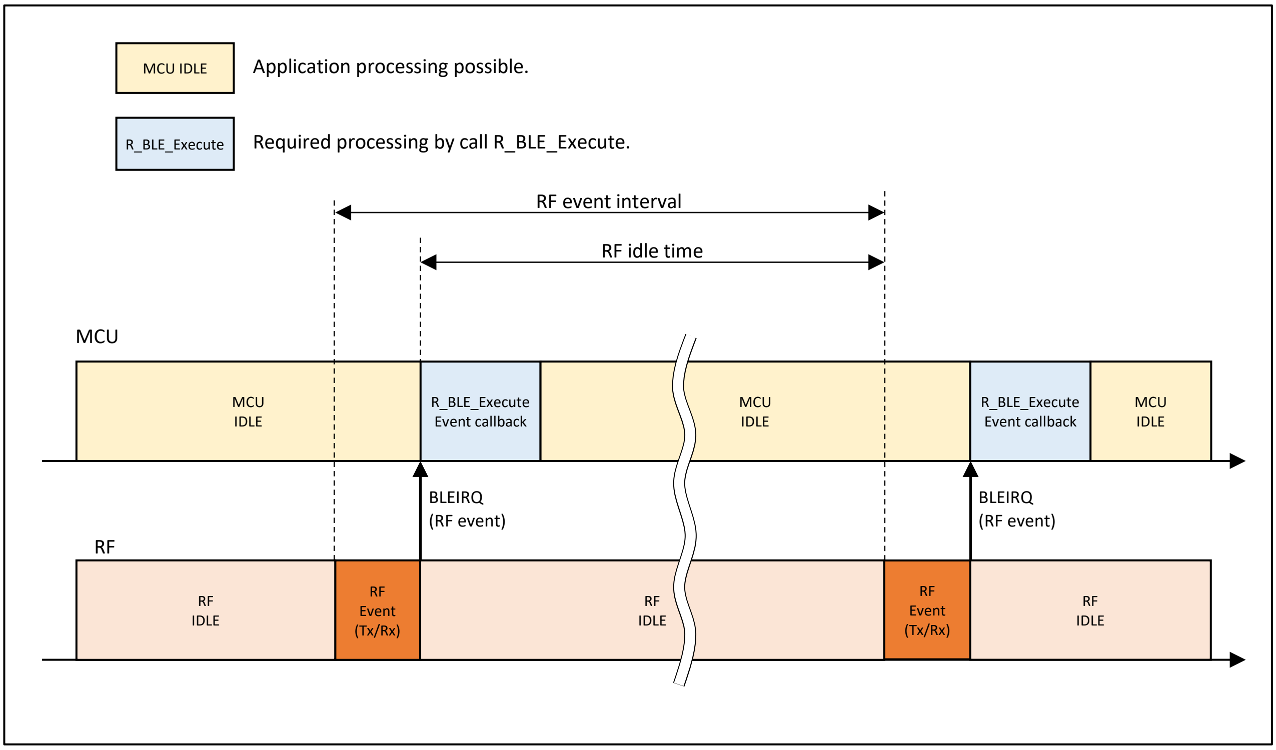


Figure 1-9 MCU/RF operation overview without RF sleep

Regardless of the RF sleep state, if the application process continuously occupies the MCU and *R_BLE_Execute* is not called, the connection may not be maintained. Therefore, it is recommended that the application is processed in short time. For processing that takes a long time, refer to "3.10 Event notification function (R_BLE_SetEvent)" and execute the processing by dividing it into multiple times.

Note: The Bluetooth LE Protocol Stack initializes the RF hardware state by *R_BLE_Open*. If the software is reset during RF communication operation, call *R_BLE_Open* to initialize the RF hardware state and stop RF operation.

1.6 Software structure

To develop the RX23W Bluetooth LE application, it is necessary to develop the application part and profile part shown in Figure 1-10.

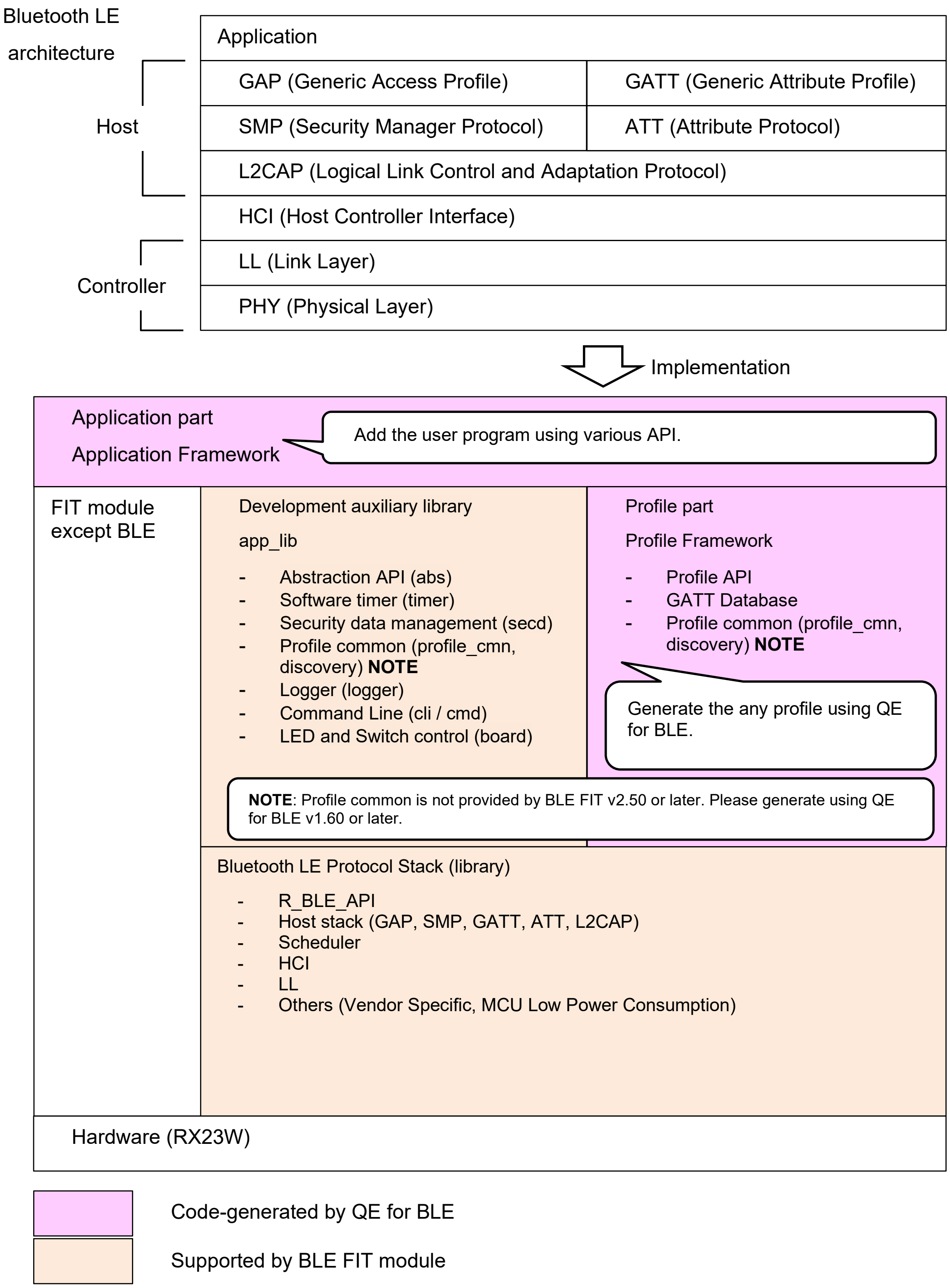

Bluetooth LE
architecture
Host
Controller
Application
GAP (Generic Access Profile)
GATT (Generic Attribute Profile)
SMP (Security Manager Protocol)
ATT (Attribute Protocol)
L2CAP (Logical Link Control and Adaptation Protocol)
HCI (Host Controller Interface)
LL (Link Layer)
PHY (Physical Layer)
Implementation
Application part
Application Framework
Add the user program using various API.
FIT module except BLE
Development auxiliary library
app_lib
- Abstraction API (abs)
- Software timer (timer)
- Security data management (secd)
- Profile common (profile_cmn, discovery) NOTE
- Logger (logger)
- Command Line (cli / cmd)
- LED and Switch control (board)
Profile part
Profile Framework
- Profile API
- GATT Database
- Profile common (profile_cmn, discovery) NOTE
Generate the any profile using QE for BLE.
NOTE: Profile common is not provided by BLE FIT v2.50 or later. Please generate using QE for BLE v1.60 or later.
Bluetooth LE Protocol Stack (library)
- R_BLE_API
- Host stack (GAP, SMP, GATT, ATT, L2CAP)
- Scheduler
- HCI
- LL
- Others (Vendor Specific, MCU Low Power Consumption)
Hardware (RX23W)
Code-generated by QE for BLE
Supported by BLE FIT module

Figure 1-10 Software structure

1.6.1 Primary functions

Constituting BLE FIT module into the project in the integrated development environment e^{2}studio enables to use the library supporting Bluetooth LE protocol and driver. The skeleton program of the application part (Application Framework) and the profile part (Profile Framework) can be code-generated by QE for BLE. As for details of each function block, refer to the document shown in Table 1.5.

Table 1.5 Function blocks

Function blocks	Reference document
BLE FIT module	BLE Module Firmware Integration Technology (R01AN4860)
Bluetooth LE Protocol Stack	Bluetooth Low Energy Protocol Stack Basic Package User's Manual (R01UW0205)
app_lib	
Profile Framework	Bluetooth Low Energy Profile Developer’s Guide (R01AN6459)
Application Framework	This document
Mesh Stack	Bluetooth Mesh Module Using Firmware Integration Technology (R01AN4930) Bluetooth Mesh Stack Package Startup Guide (R01AN4874) Bluetooth Mesh Stack Package Development Guide (R01AN4875) Note: For your information.

The functions provided by Bluetooth LE Protocol Stack library and the development auxiliary library are shown in Table 1.6.

Table 1.6 Functions provided by libraries

Functions	API/Macro name	Include header and Use
Bluetooth LE	R_BLE_XXX R_BLE_GAP_XXX R_BLE_GATT_GetMtu R_BLE_GATTS_XXX R_BLE_GATTC_XXX R_BLE_L2CAP_XXX	#include "r_ble_rx23w_if.h" Mandatory ● R_BLE_GAP_XXX Once registering callback function using R_BLE_GAP_Init, API result can be received as BLE_GAP_EVENT_XXX as event. ● R_BLE_GATTS_XXX Once registering callback function using R_BLE_GATTS_RegisterCb, API result can be received as BLE_GATTS_EVENT_XXX event. ● R_BLE_GATTC_XXX Once registering callback function using R_BLE_GATTC_RegisterCb, API result can be received as BLE_GATTC_EVENT_XXX event. ● R_BLE_L2CAP_XXX Once registering callback function using R_BLE_L2CAP_RegisterCfPsm, API result can be received as BLE_L2CAP_EVENT_XXX event. No need to register for R_BLE_XXX and R_BLE_GATT_GetMtu. API result can be received immediately. R_BLE_XXX_Init, R_BLE_XXX_RegisterCb, R_BLE_GAP_SetPairingParams can also receive API result immediately.

Functions	API/Macro name	Include header and Use
Vendor Specific (VS)	R_BLE_VS_XXX	#include "r_ble_rx23w_if.h" ● Flow control function is available. ● Device specific data management function is disabled in default. (BLE_CFG_DEV_DATA_DF_BLOCK) Note: Function to manage self BD address by using data flash. R_BLE_VS_SetBdAddr and R_BLE_VS_GetBdAddr are available. Once registering callback function using R_BLE_VS_Init, API result can be received as BLE_VS_EVENT_XXX event.
MCU Low Power Consumption (LPC)	R_BLE_LPC_XXX	#include "r_ble_rx23w_if.h" Enabled in default (BLE_CFG_MCU_LPC_EN) No need to register callback function. API result can be received immediately.
Abstraction API	R_BLE_ABS_XXX	#include "abs/r_ble_abs_api.h" Enabled in default (BLE_CFG_ABS_API_EN) Once registering callback function using R_BLE_ABS_Init, API result can be received as BLE_GAP_EVENT_XXX / BLE_GATTS_EVENT_XXX / BLE_GATTC_EVENT_XXX / BLE_VS_EVENT_XXX event. Do not change the Abstraction API codes.
Software timer	R_BLE_TIMER_XXX	#include "timer/r_ble_timer.h" Enabled in default (BLE_CFG_SOFT_TIMER_EN) If using Abstraction API, enable this function. Once registering callback function using R_BLE_TIMER_Create, timing notification can be received when interrupting by timer. Note: Use In app_main.c, call R_BLE_TIMER_Init, R_BLE_TIMER_Create.
Security data management	R_BLE_SECD_XXX	#include "sec_data/r_ble_sec_data.h" Disabled in default (BLE_CFG_EN_SEC_DATA) Note: Function to manage the bonding information by using data flash when pairing. No need to register callback function. API result can be received immediately. Note: Use In [Component] tab of Smart Configurator, add r_flash_rx. Set [r_ble_rx23w] → [Store Security Data in DataFlash.] to ”Enable”, Set [Data Flash Block for Security Data Management.] to 0～7. (Set different block from [Device specific data block on E2 Data Flash.])

Functions	API/Macro name	Include header and Use
Profile common	R_BLE_DISC_XXX R_BLE_SERVC_XXX R_BLE_SERVS_XXX	#include "discovery/r_ble_disc.h" #include "profile_cmn/r_ble_servc_if.h" #include "profile_cmn/r_ble_servs_if.h" Generated by QE for BLE. ● R_BLE_DISC_XXX Once registering callback function using R_BLE_DISC_Start, Service Discovery result can be received. ● R_BLE_SERVC_XXX Once registering callback function using R_BLE_SERVC_GattcCb, API result can be received. ● R_BLE_SERVS_XXX Once registering callback function using R_BLE_SERVS_GattsCb, API result can be received as event. ● Function to receive VS event in SERVS It is necessary to passing the event data from callback function registered by R_BLE_VS_Init or R_BLE_ABS_Init to R_BLE_SERVS_VsCb as it is. Note: Profile common is not provided by BLE FIT v2.50 or later. Please generate using QE for BLE v1.60 or later.
Logger	BLE_BD_ADDR_STR BLE_UUID_STR BLE_LOG BLE_LOG_ERR BLE_LOG_WRN BLE_LOG_DBG	#include "logger/r_ble_logger.h" Enabled in default (BLE_CFG_LOG_LEVEL) No need to register callback function.
Command Line	R_BLE_CLI_XXX R_BLE_CMD_AbsGapCb R_BLE_CMD_VsCb R_BLE_CMD_SetResetCb	#include "cli/r_ble_cli.h" #include "cmd/r_ble_cmd_abs.h" #include "cmd/r_ble_cmd_vs.h" #include "cmd/r_ble_cmd_sys.h" Disabled in default (BLE_CFG_CMD_LINE_EN) Once registering callback function using R_BLE_CLI_RegisterCmds, event can be received when interrupting by command line input. ● Function to output log Abstraction API It is necessary to passing the event data from GAP callback function registered by R_BLE_GAP_Init or R_BLE_ABS_Init to R_BLE_CMD_AbsGapCb as it is. ● Function to output log of VS It is necessary to passing the event data from VS callback function registered by R_BLE_VS_Init or R_BLE_ABS_Init to R_BLE_CMD_VsCb as it is. ● Function to register callback function notifying reset Once registering callback function using R_BLE_CMD_SetResetCb, timing notification can be received after Bluetooth LE Protocol Stack is reset by “ble reset” command or R_BLE_ABS_Reset. Note: Use in Target Board In [Component] tab of Smart Configurator, check that r_bsp is v5.40 or later, add r_sci_rx and r_byteq, set [r_ble_rx23w] → [Enabled/Disabled command line function] to "Enabled", set [SCI CH for command line function] to "8". (In order to using SCI8) Set [r_sci_rx] → [Include software support for channel 8] to ”Include”, set [ASYNC mode TX queue buffer size for channel 8] to ”160”, set [Transmit end interrupt] to ”Enable”, [Resources] → [SCI] → [SCI8] → [RXD8/SMISO8 Pin] and [TXD8/SMOSI8 Pin] to ”Used”. In app_main.c, define gsp_cmds. In app_main function, call R_BLE_CLI_Init, R_BLE_CLI_RegisterCmds, R_BLE_CMD_SetResetCb. In main loop, call R_BLE_CLI_Process.

Functions	API/Macro name	Include header and Use
LED and Switch control	R_BLE_BOARD_XXX	#include "board/r_ble_board.h" Disabled in default (BLE_CFG_BOARD_LED_SW_EN) Once registering callback function using R_BLE_BOARD_RegisterSwitchCb, timing notification can be received when interrupting by pushing switch and so on. Note: Use in Target Board In [Component] tab of Smart Configurator, add r_gpio_rx, r_irq_rx. Set [r_ble_rx23w] → [Enabled/Disabled board LED and Switch control support.] to ”Enable”, [Board Type] to ”Target board”. Set [r_irq_rx] → [Filter for IRQ5] to ”Enable”, [Filter clock divisor for IRQ5] to ”Divisor1”, [Resources] → [ICU] → [IRQ5 Pin] to ”Used”. In app_main function, call R_BLE_BOARD_Init and R_BLE_BOARD_RegisterSwitchCb.
Profile API	R_BLE_[service name]_XXX	#include "r_ble_[service name].h" Generated by QE for BLE. Once registering callback function using R_BLE_[service name]_Init, event can be received when receiving Write, Read, Indication, Notification from remote device.

The type of Bluetooth LE Protocol Stack library is selectable according to the feature used in the application. It is selectable by “2.1 Configuration Options”. The ROM/RAM code size can be reduced by selecting the type limited features. The features supported by each type are shown in Table 1.7.

Table 1.7 Bluetooth LE Protocol Stack types and its supporting features

Bluetooth LE Feature	Bluetooth LE Protocol Stack type		
	All features	Balance	Compact
LE 2M PHY	Yes	Yes	No
LE Coded PHY	Yes	Yes	No
LE Advertising Extensions	Yes	No	No
LE Channel Selection Algorithm #2	Yes	Yes	No
High Duty Cycle Non-Connectable Advertising	Yes	Yes	Yes
LE Data Packet Length Extension	Yes	Yes	Yes
LE Secure Connections	Yes	Yes	Yes
Link Layer privacy	Yes	Yes	Yes
Link Layer Extended Scanner Filter policies	Yes	Yes	No
Low Duty Cycle Directed Advertising	Yes	Yes	Yes
32-bit UUID Support in LE	Yes	Yes	Yes
LE L2CAP Connection Oriented Channel Support	Yes	No	No
LE Link Layer Topology	Yes	Yes	No
LE Ping	Yes	Yes	Yes
Bluetooth Low Energy - Enhancements to GAP for Low Energy - - GAP Role	Central Peripheral Observer Broadcaster	Central Peripheral Observer Broadcaster	Peripheral Broadcaster
Bluetooth Low Energy - Generic Attribute profile (GATT) - - GATT Role	Sever Client	Sever Client	Sever Client

1.6.2 Surrounding functions

Constituting FIT modules except BLE enables to use the MCU functions except BLE more easily. FIT modules used mainly are shown in Table 1.8.

Table 1.8 FIT modules

FIT module name	Component name	Comment
BLE	r_ble_rx23w	Bluetooth LE basic function **Mandatory** for Bluetooth LE software Interrupt priority used by Bluetooth LE Protocol Stack such as BLEIRQ, CMT2, CMT3 is fixed to 14 (cannot be changed by SC)
BLE QE Utility	r_ble_qe_utility	Profile generation function **Mandatory** for QE for BLE **Note: Included in QE for BLE instead of FIT module in QE for BLE V1.40 or later**
Board Support Package (BSP)	r_bsp	Basic setting for MCU **Mandatory** for clock setting and so on Enable to use RX23W flash memory protection function (Enable protect the block which device specific data is written, not erase it when writing firmware by flash memory writing tool)
IRQ	r_irq_rx	Set and notify interruption event Used by LED and Switch control function Enable to notify to application by detecting interrupt from switch, sensor and so on. IRQ interrupt priority is default 15 (can be changed by SC)
GPIO	r_gpio_rx	Set and use general I/O pin Used by LED and Switch control function Enable to use I/O such as LED and switch and so on assigned to Pin.
LPC	r_lpc_rx	Low Power Consumption Used by MCU Low Power Consumption function
Flash	r_flash_rx	Rewrite internal flash memory Used by Security data management function Used by Device specific data management function Used as option in HCI mode (v4.10 or later)
SCI	r_sci_rx	Set and use action mode of SCI Used by Command Line function SCI interrupt priority is default 15 (can be changed by SC)
CMT	r_cmt_rx	Generate timer event **Mandatory** for controlling H/W(RF) Used by Software timer function too Interrupt priority of CMT0, CMT1 is default 15 (can be changed by SC)
Byte type queue buffer (BYTEQ)	r_byteq	Set and manage byte type ring buffer Used by Command Line function
Bluetooth Mesh	r_mesh_rx23w	Support Mesh topology Used by Mesh function

1.7 Flow of development

Develop as the following steps. This section describes the procedure for creating a minimum configuration application (application that executes Advertising). As for standard procedure, refer to “4. BLE FIT module project” in “BLE Module Firmware Integration Technology (R01AN4860)”.

(1) Install integrated development environment e^{2}studio, Smart Configurator (SC), and QE for BLE.

(2) Create a project on e^{2}studio.

When using Target Board for RX23W, specify R5F523W8AxNG.

When using Target Board for RX23W module, specify R5F523W8CxLN.

(3) Add components of FIT module and QE for BLE by SC, change settings, generate code.

The component settings for minimum configuration are shown in Table 1.9.

Table 1.9 Component settings for minimum configuration

Procedure	Standard procedure	Minimum configuration procedure
Set clock	**Mandatory**	**Mandatory**
Add r_ble_rx23w	**Mandatory**	**Mandatory**
Change r_ble_rx23w	Execute	No need if leaving the followings disabled. Command Line function (BLE_CFG_CMD_LINE_EN) Security data management function (BLE_CFG_EN_SEC_DATA) Device specific data management function (BLE_CFG_DEV_DATA_DF_BLOCK) LED and Switch control function (BLE_CFG_BOARD_LED_SW_EN) **Note: Change MCU Low Power Consumption function (BLE_CFG_MCU_LPC_EN) to disabled.**
Add r_ble_qe_utility	Option	**Mandatory** **Note: Not required for QE for BLE V1.40 or later.**
Add BLE Profile Creation	Option	**Mandatory** **Note: Not required for QE for BLE V1.40 or later.**
Change BLE Profile Creation	Option	No need because Advertising is executed in default setting.
Change r_bsp	Execute	No need if not using RX23W flash memory protection function. **Note: Change to v5.40 or later.**
Add and change r_irq_rx	Execute	No need if leaving LED and Switch control function (BLE_CFG_BOARD_LED_SW_EN) disabled.
Add r_gpio_rx	Execute	No need if leaving LED and Switch control function (BLE_CFG_BOARD_LED_SW_EN) disabled.
Add r_lpc_rx	Execute	No need if changing MCU Low Power Consumption function (BLE_CFG_MCU_LPC_EN) to disabled.
Add r_flash_rx	Execute	No need if leaving Security data management function (BLE_CFG_EN_SEC_DATA) and Device specific data management function (BLE_CFG_DEV_DATA_DF_BLOCK) disabled.
Add and change r_sci_rx	Execute	No need if leaving Command Line function (BLE_CFG_CMD_LINE_EN) disabled.
Add r_cmt_rx	**Mandatory**	**Mandatory**
Add r_byteq	Execute	No need if leaving Command Line function (BLE_CFG_CMD_LINE_EN) disabled.

(4) Settings after code-generating

If the CMT FIT module is older than v4.50, add the following after the definition of CMT_RX_NUM_CHANNELS in r_cmt_rx.c.

```
#if defined(BSP_MCU_RX23W)
#undef CMT_RX_NUM_CHANNELS
#define CMT_RX_NUM_CHANNELS (2)
#endif /* BSP_MCU_RX23W */
```

Code 1-1 CMT_RX_NUM_CHANNELS の変更

No need changing app_lib/board/r_ble_board.c in BLE FIT module if leaving LED and Switch control function (BLE_CFG_BOARD_LED_SW_EN) disabled.

(5) Linker setting and Debugging setting on e^{2}studio

Add the followings to the section setting screen in [Project] → [Properties] → [C/C++ Build] → [Settings] → [Tool Settings] → [Linker] → [Section] → [...].
RAM: BLE_B*, BLE_R*
ROM: BLE_C*, BLE_D*, BLE_W*, BLE_L, BLE_P

Add the followings to [Project] → [Properties] → [C/C++ Build] → [Settings] → [Tool Settings] → [Linker] → [Section] → [Symbol file] → [ROM to RAM mapped section].
BLE_D=BLE_R
BLE_D_1=BLE_R_1
BLE_D_2=BLE_R_2

Input the following to [Project] → [Properties] → [C/C++ Build] → [Settings] → [Build Steps] → [Pre-build steps] → [Command(s):].
..\src\smc_gen\r_ble_rx23w\lib\ble_fit_lib_selector.bat

If connecting to the board written firmware with RX23W flash memory protection enabled, change [Run] → [Debug Configurations…] → [Renesas GDB Hardware Debugging] → [(Project name) HardwareDebug] → [Debugger] → [Connection Settings] → [Flash] → [ID Code] to “45FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF”.

When using Target Board for RX23W, [Run] → [Debug Configurations…] → [Renesas GDB Hardware Debugging] → [(Project name) HardwareDebug] → [Debugger] → [Connection Settings] → [Power] → [Power Target From The Emulator (MAX 200mA)] is not needed to change.

(6) Use the generated code

Call app_main() from src\[Project name].c in your project.

```
#include "r_smc_entry.h"

void main(void);
void app_main(void);

void main(void)
{
        app_main();
}
```

Code 1-2 Call app_main() in the main function

(7) Add and change the code

Develop any application by referring to the following chapters.

1.8 Use case of this document

An application that is connected as Peripheral from Central such as a PC or smartphone and operates as a GATT server is general. Below is a basic application and its processing.

Table 1.10 Basic application and process

Application	Process	Description
GATT server	Advertising	Refer to "5 Advertising".
	Connection	Refer to "3.3 GAP event (gap_cb function)". When receiving a connection request from Central, Bluetooth LE Protocol Stack automatically establishes a connection and notifies BLE_GAP_EVENT_CONN_IND.
	Pairing	Refer to "9 Security".
	Data communication (Notification)	Refer to "8 Communication".
GATT client	Scan	Refer to "6 Scan".
	Connection	Refer to "7 Connection".
	Pairing	Refer to "9 Security".
	Data Communication (Read, Write)	Refer to "8 Communication".

Other examples of applications that use various FIT modules and Bluetooth LE functions with RX23W are shown below.

GATT Server application that collects operation logs of industrial equipment and sensor data of healthcare equipment and uploads them to Clients such as PCs and smartphones

→ Refer to "2.4 How to configure for minimum current consumption", "7.3 Multiple Connection" and "9 Security".

GATT Server application that transfers the data downloaded from Clients such as PCs and smartphones and updates the firmware

→ Refer to "8.6 High throughput communication" and "9 Security".

GATT Server application that uploads the image data such as printers and scanners, voice data and audio data of recording devices to Clients such as PCs and smartphones, and downloads the setting data from Clients.

→ Refer to "8.6 High throughput communication".

GATT Server applications for electronic locks, OA devices, consumer devices, etc. that are operated by multiple Clients such as smartphones

→ Refer to "7.3 Multiple Connection" and "9 Security".

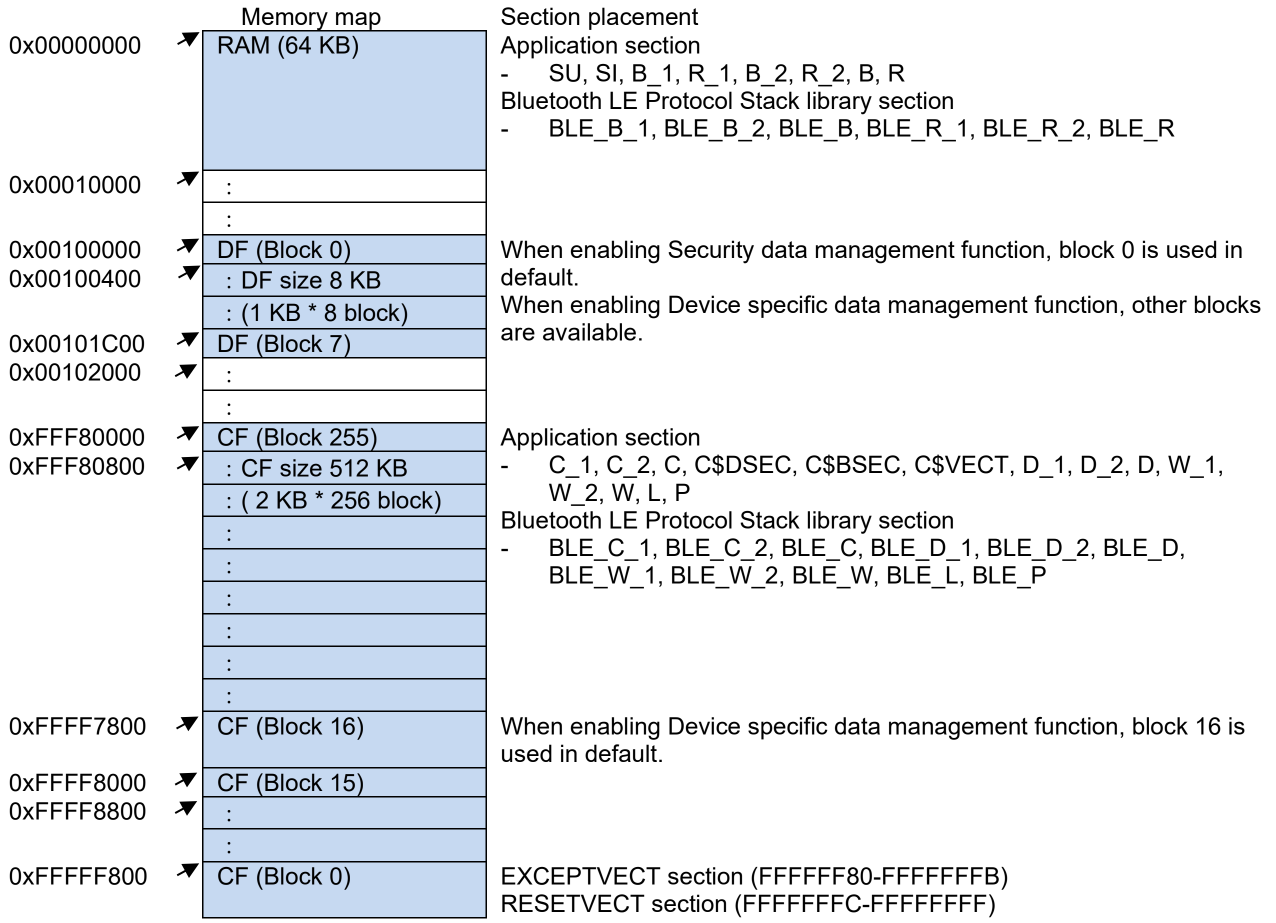
Beacon application that periodically sends out multiple sensor data

→ Refer to "5.7 Beacon".

1.9 Locating sections

The section of the library provided by the Bluetooth LE Protocol Stack and the section of the public source code used by the library is changed to the section name with the prefix "BLE_" so that the section allocation can be divided for the purpose of FW update.

Memory map as for RAM, Data Flash ROM(DF), and Code Flash ROM (CF) in demo project in RX23W(R5F523W8Axxx) and their section placement set by linker on e^{2}studio are shown in below.



	Memory map	Section placement
0x00000000	RAM (64 KB)	Application section - SU, SI, B_1, R_1, B_2, R_2, B, R Bluetooth LE Protocol Stack library section - BLE_B_1, BLE_B_2, BLE_B, BLE_R_1, BLE_R_2, BLE_R
0x00010000	:	
	:	
0x00100000	DF (Block 0)	When enabling Security data management function, block 0 is used in default. When enabling Device specific data management function, other blocks are available.
0x00100400	: DF size 8 KB	
	: (1 KB * 8 block)	
0x00101C00	DF (Block 7)	
0x00102000	:	
	:	
0xFFF80000	CF (Block 255)	Application section - C_1, C_2, C, C$DSEC, C$BSEC, C$VECT, D_1, D_2, D, W_1, W_2, W, L, P Bluetooth LE Protocol Stack library section - BLE_C_1, BLE_C_2, BLE_C, BLE_D_1, BLE_D_2, BLE_D, BLE_W_1, BLE_W_2, BLE_W, BLE_L, BLE_P
0xFFF80800	: CF size 512 KB	
	: (2 KB * 256 block)	
	:	
	:	
	:	
	:	
	:	
	:	
0xFFFF7800	CF (Block 16)	When enabling Device specific data management function, block 16 is used in default.
0xFFFF8000	CF (Block 15)	
0xFFFF8800	:	
	:	
0xFFFFF800	CF (Block 0)	EXCEPTVECT section (FFFFFF80-FFFFFFFB) RESETVECT section (FFFFFFFC-FFFFFFFF)

Figure 1-11 Locating sections

As for linker setting, refer to “1.7 Flow of development”.

It can be confirming actual section placement by map file. As for map file, refer to “11.6.3 Outputting detail to MAP file”.

If using RX23W Start-Up Program Protection function, block 0 to 15 are protected. Therefore, block (BLE_CFG_DEV_DATA_CF_BLOCK) where device specific data such as BD address is written are specified as block 16 in default. As for BD address, refer to “2.3 How to configure BD address”.

2. Adjusting configuration option

2.1 Configuration Options

The configuration options of the BLE FIT module are located in the r_ble_rx23w_config.h. The options are able to be configured in Smart Configurator (SC). The changed options are automatically reflected when adding the BLE FIT module to the project. The macro and SC display name and setting range are listed in Table 2.1.

Note: If you edit r_ble_rx23w_config.h directly, depending on the settings in [Project]-[Properties]-[Builders]-[SC Code Generation Builder], the codes generated when building the project will overwrite the edited contents. It is recommended to change the setting from SC.

Table 2.1 Configuration Options

Macro (SC display name)	Setting range (default)	Description
BLE_CFG_LIB_TYPE (Type of Bluetooth LE Protocol Stack library)	0: All features, 1: Balance, 2: Compact (0)	Type of the Bluetooth LE Protocol Stack.
BLE_CFG_RF_DBG_PUB_ADDR (Initial Public Address)	Set any value. ({0xFF,0xFF,0xFF,0x50,0x90,0x74})	Initial Public Address.
BLE_CFG_RF_DBG_RAND_ADDR (Initial Static Address)	Set any value. ({0xFF,0xFF,0xFF,0xFF,0xFF,0xFF})	Initial Static Address.
BLE_CFG_RF_CONN_MAX (Maximum number of connections)	1 - 7 (7)	Maximum number of simultaneous connections.
BLE_CFG_RF_CONN_DATA_MAX (Maximum connection data length)	27 - 251 (251)	Maximum packet data length (bytes).
BLE_CFG_RF_ADV_DATA_MAX (Maximum advertising data length)	31 - 1650 (1650)	Maximum advertising data length (bytes).
BLE_CFG_RF_ADV_SET_MAX (Maximum advertising set number)	1 - 4 (4)	Maximum number of the advertising set.
BLE_CFG_RF_SYNC_SET_MAX (Maximum advertising set number)	1 - 2 (2)	Maximum number of periodic sync set.
BLE_CFG_EVENT_NOTIFY_CONN_START (Connection event start notify)	0 - 1 (0)	Enable or disable start interrupt notification of a connection complete event.
BLE_CFG_EVENT_NOTIFY_CONN_CLOSE (Connection event close notify)	0 - 1 (0)	Enable or disable end interrupt notification of a connection complete event.
BLE_CFG_EVENT_NOTIFY_ADV_START (Advertising event start notify)	0 - 1 (0)	Enable or disable the advertising event start interrupt notification.
BLE_CFG_EVENT_NOTIFY_ADV_CLOSE (Advertising event close notify)	0 - 1 (0)	Enable or disable the advertising event complete interrupt notification.
BLE_CFG_EVENT_NOTIFY_SCAN_START (Scanning event start notify)	0 - 1 (0)	Enable or disable the scan start interrupt notification.
BLE_CFG_EVENT_NOTIFY_SCAN_CLOSE (Scanning event close notify)	0 - 1 (0)	Enable or disable the scan complete interrupt notification.
BLE_CFG_EVENT_NOTIFY_INIT_START (Initiating event start notify)	0 - 1 (0)	Enable or disable the notification that the scan start interrupt has occurred in sending a connection request.
BLE_CFG_EVENT_NOTIFY_INIT_CLOSE (Initiating event close notify)	0 - 1 (0)	Enable or disable the notification that the scan complete interrupt has occurred in sending a connection request.
BLE_CFG_EVENT_NOTIFY_DS_START (RF_DEEP_SLEEP start notify)	0 - 1 (0)	Enable or disable the RF_DEEP_SLEEP start notification.

Macro (SC display name)	Setting range (default)	Description
BLE_CFG_EVENT_NOTIFY_DS_WAKEUP (RF_DEEP_SLEEP wakeup notify)	0 - 1 (0)	Enable or disable the RF_DEEP_SLEEP wakeup notification.
BLE_CFG_RF_CLVAL (Capacity adjustment of 32MHz crystal resonator)	0 - 15 (6)	Adjustment value of the 32MHz crystal oscillator.
BLE_CFG_RF_DDC_EN (DC-DC converter configuration for RF part)	0 - 1 (0)	Enable or disable the DC-DC on the RF.
BLE_CFG_RF_EXT32K_EN (Slow clock source for RF part)	0 - 1 (0)	Slow clock source to the RF.
BLE_CFG_RF_MCU_CLKOUT_PORT (MCU CLKOUT port)	0 - 1 (0)	Port of the MCU CLKOUT.
BLE_CFG_RF_MCU_CLKOUT_FREQ (MCU clock frequency)	0 - 1 (0)	Output frequency from the MCU CLKOUT.
BLE_CFG_RF_SCA (Sleep Clock Accuracy (SCA) for RF slow clock)	0 - 500 (250)	Sleep Clock Accuracy (SCA) for the RF slow clock.
BLE_CFG_RF_MAX_TX_POW (Transmission power maximum value)	0 - 1 (1)	Maximum transmit power configuration.
BLE_CFG_RF_DEF_TX_POW (Default transmit power)	0 - 1 (0)	Default transmit power level.
BLE_CFG_RF_CLKOUT_EN (CLKOUT_RF output setting)	Select one of the followings. 0: No output 5: 4MHz output 6: 2MHz output 7: 1MHz output (0)	CLKOUT_RF output.
BLE_CFG_RF_DEEP_SLEEP_EN (RF_DEEP_SLEEP transition)	0 - 1 (1)	Enable or disable the RF Deep Sleep.
BLE_CFG_MCU_MAIN_CLK_KHZ (MCU Main Clock Frequency (kHz))	If the HOCO is used, this option is ignored. If the Main Clock is used, set a value within the range between 1000 and 20000. If the PLL Circuit is used, set a value within the range between 4000 and 12500. (4000)	MCU main clock frequency (kHz).
BLE_CFG_DEV_DATA_CF_BLOCK (Device specific data block on Code Flash (ROM))	-1 - 255 (16)	The Code Flash (ROM) block stored the device specific data.
BLE_CFG_DEV_DATA_DF_BLOCK (Device specific data block on E2 Data Flash)	-1 - 7 (-1)	The E2 Data Flash block stored the device specific data.
BLE_CFG_GATT_MTU_SIZE (MTU Size configured by GATT MTU exchange procedure)	23 - 247 (247)	The MTU size (bytes) for the GATT communication.
BLE_CFG_NUM_BOND (Number of remote device bonding information)	1 - 7 (7)	Maximum number of the bonding information stored in the Data Flash.
BLE_CFG_EN_SEC_DATA (Store Security Data in DataFlash)	0 - 1 (0)	Enable or disable the security data management.
BLE_CFG_SECD_DATA_DF_BLOCK (Data Flash Block for Security Data Management)	0 - 7 (0)	The Data Flash block for the security data management to store the bonding information.
BLE_CFG_CMD_LINE_EN (Enabled/Disabled command line function)	0 - 1 (0)	Enable or disable the command line function.
BLE_CFG_CMD_LINE_CH (SCI CH for command line function)	1 or 5 or 8 (1)	SCI Channel for the command line function.
BLE_CFG_BOARD_LED_SW_EN (Enabled/Disabled board LED and Switch control support)	0 - 1 (0)	Enable or disable support the board LED & Switch control.

Macro (SC display name)	Setting range (default)	Description
BLE_CFG_BOARD_TYPE (Board Type)	0 - 3 (0)	Board type.
BLE_CFG_LOG_LEVEL (Log level)	0 - 3 (3)	Log level.
BLE_CFG_ABS_API_EN (Abstraction API support)	0 - 1 (1)	Enable or disable support the Abstraction API.
BLE_CFG_SOFT_TIMER_EN (Software Timer support)	0 - 1 (1)	Enable or disable support the software time in app_lib.
BLE_CFG_MCU_LPC_EN (MCU low power consumption control support)	0 - 1 (1)	Enable or disable support the MCU low power consumption control.
BLE_CFG_HCI_MODE_EN (HCI mode support)	0 - 1 (0)	Select start in HCI mode or not.

2.2 How to adjust RAM

Some configuration options affect the RAM size. Table 2.2 shows the additional RAM size if one is added to the configuration option.

Table 2.2 Additional RAM size per configuration option

Configuration Options		Setting range (default)	Library	Additional Size (bytes)
SC display name	Macro			
Maximum number of connections	BLE_CFG_RF_CONN_MAX	1 - 7 (7)	All features	1094
			Balance	1086
			Compact	1074
Maximum connection data length	BLE_CFG_RF_CONN_DATA_MAX	27 - 251 (251)	All libraries	9
Maximum advertising data length	BLE_CFG_RF_ADV_DATA_MAX	31 - 1650 (1650)	All features	Described in Table 2.3
Maximum advertising set number *1	BLE_CFG_RF_ADV_SET_MAX	1 - 4 (4)	All features	308
Maximum periodic sync set number *2	BLE_CFG_RF_SYNC_SET_MAX	1 - 2 (2)	All features	66

*1 : Simultaneous advertising number.

*2 : Maximum periodic synchronization number.

The additional RAM size of BLE_CFG_RF_ADV_DATA_MAX depends on BLE_CFG_RF_ADV_SET_MAX. Table 2.3 shows the additional RAM size where BLE_CFG_RF_ADV_DATA_MAX is changed from the RAM size when BLE_CFG_RF_ADV_DATA_MAX is set to 0-252 bytes.

Table 2.3 Additional RAM size per BLE_CFG_RF_ADV_DATA_MAX and BLE_CFG_RF_ADV_SET_MAX

Maximum advertising set number	1	BLE_CFG_RF_ADV_DATA_MAX	0-252	253-504	505-756	757-1008	1009-1260	1261-1512	1513-1650
		Additional size (bytes)	0	512	1024	1536	2048	2560	3072
	2	BLE_CFG_RF_ADV_DATA_MAX	0-252	253-504	505-756	757-1008	1009-1260	1261-1512	1513-1650
		Additional size (bytes)	0	1024	2048	3072	4096	5120	6144
	3	BLE_CFG_RF_ADV_DATA_MAX	0-252	253-504	505-756	757-1008	1009-1260	1261-1650	
		Additional size (bytes)	0	1536	3072	4608	6144	7680	
	4	BLE_CFG_RF_ADV_DATA_MAX	0-252	253-504	505-756	757-1008	1009-1650		
		Additional size (bytes)	0	2048	4096	6144	7168		

Set the values of maximum advertising data length and maximum advertising set number so that they fall within the following range.

4250 >= Maximum advertising data length * Maximum number of advertising sets

2.3 How to configure BD address

Bluetooth Device address (BD address) has the following types.

Table 2.4 BD address types

BD address type			Description
Public device address			Public Address gotten upper 24 bits from IEEE. Note: Refer to Bluetooth Core Specification Vol 2, PartB, " 1.2 Bluetooth Device Addressing". It can be obtained in the same way as the MAC address.
Random device address	Static address		Random Address where the most significant bit starts with 11 and the remaining bits can be set randomly to be used. Cx:xx:xx:xx:xx:xx or Dx:xx:xx:xx:xx:xx or Ex:xx:xx:xx:xx:xx or Fx:xx:xx:xx:xx:xx Note: Refer to Bluetooth Core Specification Vol 6, PartB, "1.3.2 Random Device Address". Note: Bluetooth LE Protocol Stack does not check address format. Note: A static address consists of random numbers. The possibility of duplicate values with other devices is not zero.
	Private address	Non-resolvable private address	Random Address where the most significant bit starts with 00 and the remaining bits can be dynamically regenerated. 0x:xx:xx:xx:xx:xx or 1x:xx:xx:xx:xx:xx or 2x:xx:xx:xx:xx:xx or 3x:xx:xx:xx:xx:xx
		Resolvable private address (RPA)	Random Address where the most significant bit starts with 01 and the remaining bits can be dynamically regenerated and enhanced with privacy feature. 4x:xx:xx:xx:xx:xx or 5x:xx:xx:xx:xx:xx or 6x:xx:xx:xx:xx:xx or 7x:xx:xx:xx:xx:xx

Bluetooth devices have an Identity address. Identity address is either Public device address or Static Address. The device using Privacy function requires an Identity address.

RX23W provides the function to store the static BD address such as Public device address and Static Address in the data area and user area of the internal ROM. Data flash (DF) can be used as the data area and code flash (CF) can be used as the user area. They are set as follows in default by Configuration option of [r_ble_rx23w].

Table 2.5 BD address configurations

Configuration option	Initial value
BLE_CFG_DEV_DATA_DF_BLOCK	-1 (DF is not used)
BLE_CFG_DEV_DATA_CF_BLOCK	16 (CF block 16 is used)
BLE_CFG_RF_DBG_PUB_ADDR	74:90:50:FF:FF:FF (Firmware initial value of Public Address)
BLE_CFG_RF_DBG_RAND_ADDR	FF:FF:FF:FF:FF:FF (Firmware initial value of Random Address)

The BD address can be used by selecting either Public Address or Random Address when starting Advertising. For details on how to use the set Random Address, refer to "2.3.2 How to use Random Address".

The BD address is determined as below in R_BLE_Open at application startup according to "5.4.6 BD address adoption flow" in "Bluetooth Low Energy Protocol Stack Basic Package User's Manual (R01UW0205)", and stored in the managed RAM of Bluetooth LE Protocol Stack.

Table 2.6 BD address adoption method

Priority	BD address adoption method	Initial value	Description
1	DF is used. (BLE_CFG_DEV_DATA_DF_BLOCK is set 0 to 7) Note: Set the different block from the block set by BLE_CFG_SECD_DATA_DF_BLOCK of default 0. Note: DF is not used when setting -1.	For flash initialization: Public Address FF:FF:FF:FF:FF:FF Random Address FF:FF:FF:FF:FF:FF Note: ALL 0x00 or 0xFF is disable.	Used if writing BD address by outer I/F such as UART after product shipment. It can be rewriting by either of the following methods. - Rewrite by specifying DF in R_BLE_VS_SetBdAddr. - Rewrite by BDAddrWriter, in case of HCI mode firmware. Note: Enable Flash FIT module. Note: Changes are reflected by resetting RX23W.
2	CF is used. (BLE_CFG_DEV_DATA_CF_BLOCK is set 0 to 255.) Note: Because 0 to 15 are Start-Up Program Protection blocks, when using Start-Up Program Protection function, do not set 0 to15. Note: CF is not used when setting -1.	For flash initialization: Public Address FF:FF:FF:FF:FF:FF Random Address FF:FF:FF:FF:FF:FF Note: ALL 0x00 or 0xFF is disable.	Used if writing BD address together with the firmware at the time of product shipment. It can be rewriting by following methods. - Rewrite firmware by using unique code function of Renesas Flash Programmer(RFP). Note: By using RX23W memory protection function, it can be guarded against being rewritten by third parties.
3	Firmware initial value is used. BLE_CFG_RF_DBG_PUB_ADDR BLE_CFG_RF_DBG_RAND_ADDR	Public Address 74:90:50:FF:FF:FF Random Address FF:FF:FF:FF:FF:FF Note: ALL 0x00 or 0xFF is disable.	Used if changing BD address on debug temporarily.
4	Static value is used	Public Address 74:90:50:FF:FF:FF Random Address XX:XX:XX:XX:XX:XX	Used when all of the above are disabled. Random Address is generated by MCU unique ID.
Other	The managed RAM of Bluetooth LE Protocol Stack is used by rewriting.	Public Address XX:XX:XX:XX:XX:XX Random Address XX:XX:XX:XX:XX:XX	Used if managing BD address by application dynamically. After BLE_GAP_EVENT_STACK_ON, rewrite by specifying Current register in R_BLE_VS_SetBdAddr.

As for details on rewriting BD address to data area, refer to "2.3.1 Writing to data area".

As for details on rewriting to user area and RX23 memory protection function, refer to "5.4.3 Writing to user area (ROM)" and "5.4.5 RX23W flash memory protection function" in "Bluetooth Low Energy Protocol Stack Basic Package User's Manual (R01UW0205)".

2.3.1 Writing to data area

Use R_BLE_VS_SetBdAddr() to write to the data area. If writing HCI mode firmware, use public BD address writing tool (BDAddrWriter). As for BDAddrWriter, refer to "5.4.4.2 Write to data area using BDAddrWriter tool" in "Bluetooth Low Energy Protocol Stack Basic Package: User's Manual (R01UW0205)". The written BD address is used by resetting RX23W once.

2.3.2 How to use Random Address

The following is a sample code for advertising with a Random Address determined by R_BLE_Open.

Get the Random Address selected with R_BLE_VS_GetBdAddr and call R_BLE_ABS_StartLegacyAdv with the Random Address obtained with the BLE_VS_EVENT_GET_ADDR_COMP event.

```
static st_ble_abs_legacy_adv_param_t gs_adv_param =
{
    (OMISSION)
    .o_addr_type     = BLE_GAP_ADDR_PUBLIC,
};

static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_VS_GetBdAddr(BLE_VS_ADDR_AREA_REG, BLE_GAP_ADDR_RAND);
        } break;
        (OMISSION)

static ble_status_t ble_app_init(void);
static void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
    (OMISSION)
    switch (type)
    {
        case BLE_VS_EVENT_GET_ADDR_COMP:
        {
            st_ble_vs_get_bd_addr_comp_evt_t * p_get_addr =
                (st_ble_vs_get_bd_addr_comp_evt_t *)p_data->p_param;
            memcpy(gs_adv_param.o_addr, p_get_addr->addr.addr, BLE_BD_ADDR_LEN);
            R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
        } break;
        (OMISSION)
```

Code 2-1 Sample of using Random Address

Using Command line function, BD address of DF can be check and rewritten with "vs addr get df" and "vs addr set df" commands. BD address of the managed RAM of Bluetooth LE Protocol Stack can be checked and rewritten with "vs addr get curr" and "vs addr set curr".

```
$ vs addr get curr pub
$ BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8
 addr:36:35:34:33:32:31 pub on current register

$ vs addr get df pub
$ BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8
 addr:36:35:34:33:32:31 pub on data flash

$ vs addr get curr rnd
$ BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8
 addr:D9:7C:E6:81:83:35 rnd on current register

$ vs addr get df rnd
$ BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8
 addr:FF:FF:FF:FF:FF:FF rnd on data flash
```

2.4 How to configure for minimum current consumption

The following configurations make the current consumption minimize.

Table 2.7 Configurations for minimum current consumption

Configuration options		Comments
MCU clock set	HOCO clock: Enable Frequency: 32MHz	Note: Configure on [Clocks] tab in Smart configurator. Note: Disable non-used clocks or set minimum clock frequency. Note: R_BLE_Open optimizes the operation of R_BLE_Execute according to the clock frequency. If you change the clock frequency dynamically, call R_BLE_Open again after R_BLE_Close.
	FCLK : x1 (32MHz)	
	ICLK : x1 (32MHz)	
	PCLKB : x1 (32MHz)	
r_ble_rx23w component set	DC-DC on the RF: Enable (BLE_CFG_RF_DDC_EN=1)	Note: Refer to “Guidelines for Bluetooth Board Design Application Note (R01AN4534)”.
	RF Deep Sleep: Enable (BLE_CFG_RF_DEEP_SLEEP_EN=1)	
	MCU Low Power Consumption: Enable (BLE_CFG_MCU_LPC_EN=1)	Note: Need to call R_BLE_LPC_EnterLowPowerMode API after calling R_BLE_Execute API in main loop.
	CLKOUT_RF: No output (BLE_CFG_RF_CLKOUT_EN=0)	
	Command line function: Disable (BLE_CFG_CMD_LINE_EN=0)	
	LED and Switch control function: Disable (BLE_CFG_BOARD_LED_SW_EN=0)	
	RF maximum transmit power: +4dBm → +0dBm (BLE_CFG_RF_MAX_TX_POW=0)	
	RF default transmit power: High → Mid → Low (BLE_CFG_RF_DEF_TX_POW=2)	Note: The transmit current can be reduced by lowering the RF transmit power, but the communication range will be shortened accordingly.

2.4.1 Using MCU Low Power Consumption function

The MCU can be shifted to the low power consumption state even when using the BLE function. The basic policy of the transition to Low power consumption state is as below.

- After completing the execution of *R_BLE_Execute()*, until the next *R_BLE_Execute()* is executed, Bluetooth LE Protocol Stack does not prevent MCU from the transitioning to Low power consumption state.
- After confirming that all the used components (including the BLE function) can shift MCU to Low power consumption state, the application shifts MCU to Low power consumption state.

As a sample program code for low power consumption, a program code (r_ble_pf_lowpower.c) with the following functions is provided.

- Use LPC FIT module to shifts MCU to Low power consumption state.
- Sleep mode, Deep sleep mode, and Software standby mode are available as Low power consumption state.
- Use *R_BLE_LPC_Init()* to initialize Low power consumption function.
- Use *R_BLE_LPC_EnterLowPowerMode()* to shift to Low power consumption state.
  - Disable MCU interrupts
  - Check that there is no problem even if each component shifts to Low power consumption state
  - Execute the transition processing to Low power consumption state of each component
  - Enter MCU to Low power consumption state
  - After MCU wakes-up from Low power consumption state, resume each component to the normal state
- When BLE communication occurs, it resumes from Low power consumption state by RF interrupt. However, since there is a possibility that RF interrupt may occur during processing for disabling interrupts, check the status of BLE task once after disabling interrupts, If BLE task state is not free, skip transition to Low power consumption state of MCU.

The operation status of each component in each low power consumption state is listed “11. Low Power Consumption Table 11-2” in “RX23W Group User's Manual: Hardware (R01UH0823)”.

As for components other than the BLE function, if adding processing for transition and resume to Low power consumption state, change the following locations of “r_ble_pf_lowpower.c”.

(1) Checking transition to Low power consumption state

- Software standby mode
  In *check_software_standby()* function, add processing to check if there is no problem even if the component enters to Software standby mode. Add processing to the location of “/* add check for other components */” comment in Code 2-2.

```
static bool check_software_standby(void)
{
    if (g_inhibit_software_standby)
    {
        return false;
    }

    /* (OMISSION) */

    /* If DTC/DMAC/DataFlash is in active, MCU can not enter software standby.
       This code is copied from r_lpc_rx23w.c lpc_lowpower_activate_check. */
    if ((0x0000 != (FLASH.FENTRYR.WORD & 0x0081)) ||
        ((0 == SYSTEM.MSTPCRA.BIT.MSTPA28) &&
         ((1 == DTC.DTCST.BIT.DTCST) || (1 == DMAC.DMAST.BIT.DMST))))
    {
        return false;
    }

    /* add check for other components */

    return true;
}
```

Code 2-2 Location to check for transition to Software standby mode

- Deep Sleep mode
  In *check_deep_sleep()* function, add processing to check if there is no problem even if entering to Deep sleep mode when using the Watchdog Timer (WDT). There is no need to add any components other than the Watchdog Timer (WDT) for checking the transition to Deep sleep mode. Add processing to the location of “/* add check for other components */” in Code 2-3.

```
static bool check_deep_sleep(void)
{
    /* If DTC/DMAC/DataFlash is in active, MCU can not enter deep sleep.
       This code is copied from r_lpc_rx23w.c lpc_lowpower_activate_check. */
    if ((0x0000 != (FLASH.FENTRYR.WORD & 0x0081)) ||
        (0 == SYSTEM.MSTPCRA.BIT.MSTPA28))
    {
        return false;
    }

    /* add check for other components */

    return true;
}
```

Code 2-3 Location to check for transition to Deep sleep mode

(2) Transition preparation processing to Low power consumption state

In *suspend_peripherals()* function, add the preparation processing for transition to Low power consumption state of each component. Add the transition preparation processing according to each low power consumption state to the location of "/* add implementation for transiting xxx mode */" in Code 2-4.

```
static void suspend_peripherals(lpc_low_power_mode_t mode)
{
    if (LPC_LP_SW_STANDBY == mode)
    {
        R_BLE_CLI_Terminate();

        /* add implementation for transiting the software standby mode. */
    }
    else if(LPC_LP_DEEP_SLEEP == mode)
    {
        /* add implementation for transiting the deep sleep mode. */
    }
    else if(LPC_LP_SLEEP == mode)
    {
        /* add implementation for transiting the sleep mode. */
    }
    else
    {

    }

}
```

Code 2-4 Location to add transition preparation for each low power consumption state

(3) Resume processing from Low power consumption state

In *resume_peripherals()* function, add the resume processing from Low power consumption state of each component. Add the resume process according to each low power consumption state to the location of "/* add implementation for transiting the active state. */" in Code 2-5.

```
static void resume_peripherals(lpc_low_power_mode_t mode)
{
    if (LPC_LP_SW_STANDBY == mode)
    {
        R_BLE_CLI_Init();

        /* add implementation for transiting the active state. */
    }
    else if(LPC_LP_DEEP_SLEEP == mode)
    {
        /* add implementation for transiting the active state. */
    }
    else if(LPC_LP_SLEEP == mode)
    {
        /* add implementation for transiting the active state. */
    }
    else
    {

    }
}
```

Code 2-5 Location to add resume processing from each low power consumption state

3. How to implement user code

QE for BLE does not only code-generate the designed profile part but also the application processing corresponding to GAP role (Central/Peripheral) into src\smc_gen\Config_BLE_Profile. Once just only creating a project for each GAP role and code-generating, the mutually connectable programs are generated as app_main.c. As for the behavior of the generated program, refer to "3.1 Behavior of skeleton program".

Applications can be developed by adding/modifying user code using various APIs in the following functions of app_main.c. Some APIs, once are executed, immediately returns the result as the return value of the function, but most APIs are queued and executed as events in the scheduler, and the execution results are returned as event notifications to the event handler.

Table 3.1 Functions of app_main.c

Function of app_main.c		Description
app_main function		The app_main function has a main loop. The following API is used before the main loop. - Initialization API This is a required function. Register an event handler or interrupt handler as a callback function in the Bluetooth LE Protocol Stack scheduler. Also register the GATT database. It is explained in "3.2.1 Initialization process (ble_app_init function) (* transfer)". The following API is used in the main loop. - R_BLE_Execute This is a required function. Run the scheduler to handle the event and return the result to your callback function. It is explained in "3.2.2 Main loop and scheduler (R_BLE_Execute) (* transfer)". - R_BLE_CLI_Process Used when the command line function is enabled. - R_BLE_LPC_EnterLowPowerMode This is used when the MCU low power consumption function is enabled.
Callback function	Event handler	Called when GAP / GATTS / GATTC / VS / Profile Server / Profile Client / L2CAP / DISC event occurs. Note: The RF communication timing notification is called from the LL → scheduler when an RF interrupt occurs. Note: The software timer is called from the CMT FIT module → Bluetooth LE Protocol Stack scheduler when a timer interrupt occurs. [Note] LED of LED and Switch control does not use notification. Switch is called from the IRQ FIT module → Bluetooth LE Protocol Stack scheduler when a switch press interrupt occurs.
	Interrupt handler	When the command line function is enabled, it is called from the SCI FIT module when a UART transmission/reception interrupt occurs.

If dividing the process of app_main.c into another file, right-click the Config_BLE_Profile folder on e^{2}studio and add another file by the following procedure.
Add [arbitrary name].c from [New] → [Source File].
Add [arbitrary name].h from [New] → [Header File].

As for details of API parameters, refer to R_BLE API document (r_ble_api_spec.chm).

3.1 Behavior of skeleton program

The behavior of the skeleton program generated by QE for BLE and the main events to be notified are shown below. The Bluetooth LE Protocol Stack automatically handles the dotted line responses and operations, so no code is required.

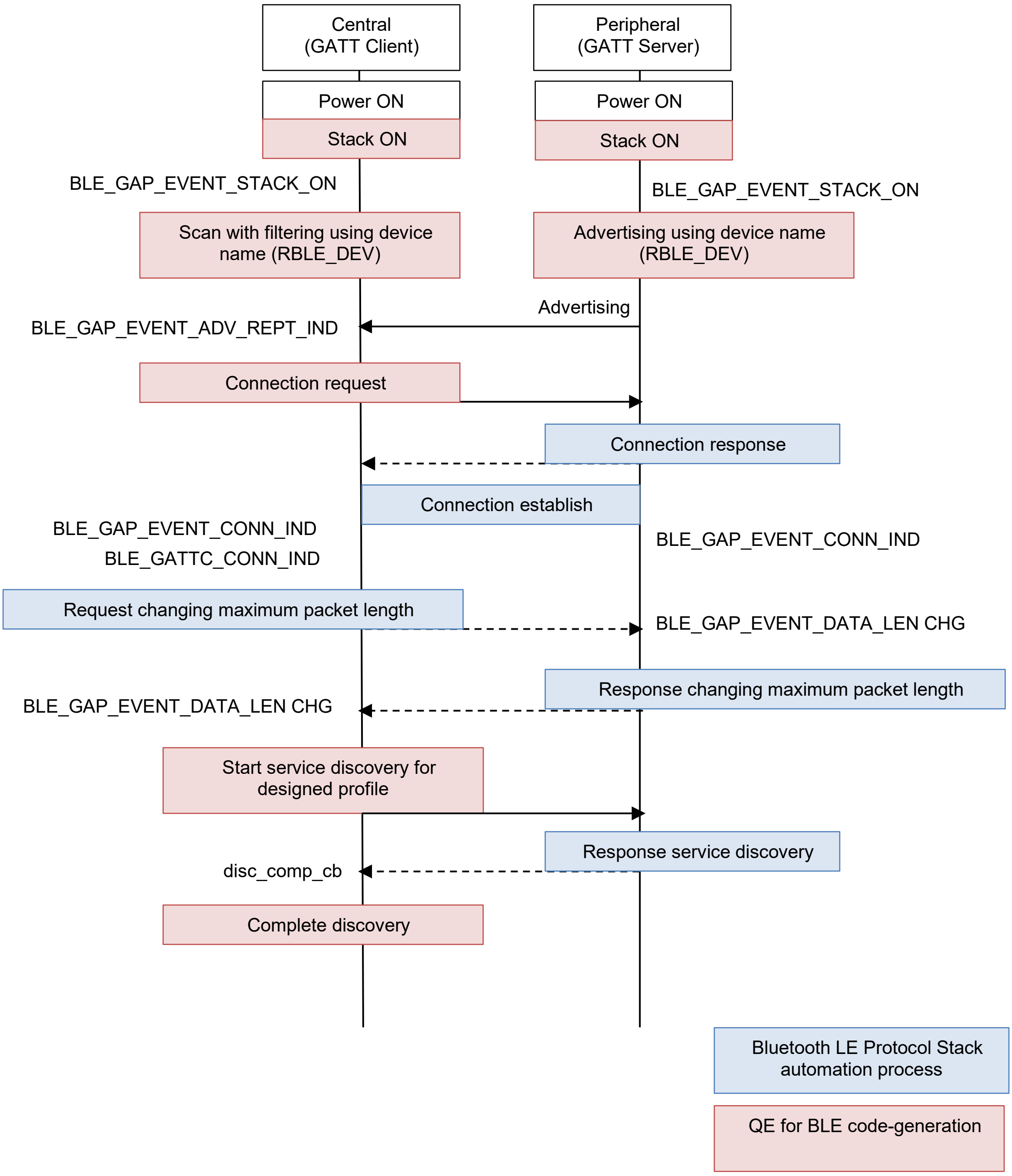

Central
(GATT Client)
Peripheral
(GATT Server)
Power ON
Stack ON
Power ON
Stack ON
BLE_GAP_EVENT_STACK_ON
BLE_GAP_EVENT_STACK_ON
Scan with filtering using device name (RBLE_DEV)
Advertising using device name (RBLE_DEV)
Advertising
BLE_GAP_EVENT_ADV_REPT_IND
Connection request
Connection response
Connection establish
BLE_GAP_EVENT_CONN_IND
BLE_GATTC_CONN_IND
BLE_GAP_EVENT_CONN_IND
Request changing maximum packet length
BLE_GAP_EVENT_DATA_LEN CHG
Response changing maximum packet length
BLE_GAP_EVENT_DATA_LEN CHG
Start service discovery for designed profile
Response service discovery
disc_comp_cb
Complete discovery
Bluetooth LE Protocol Stack automation process
QE for BLE code-generation

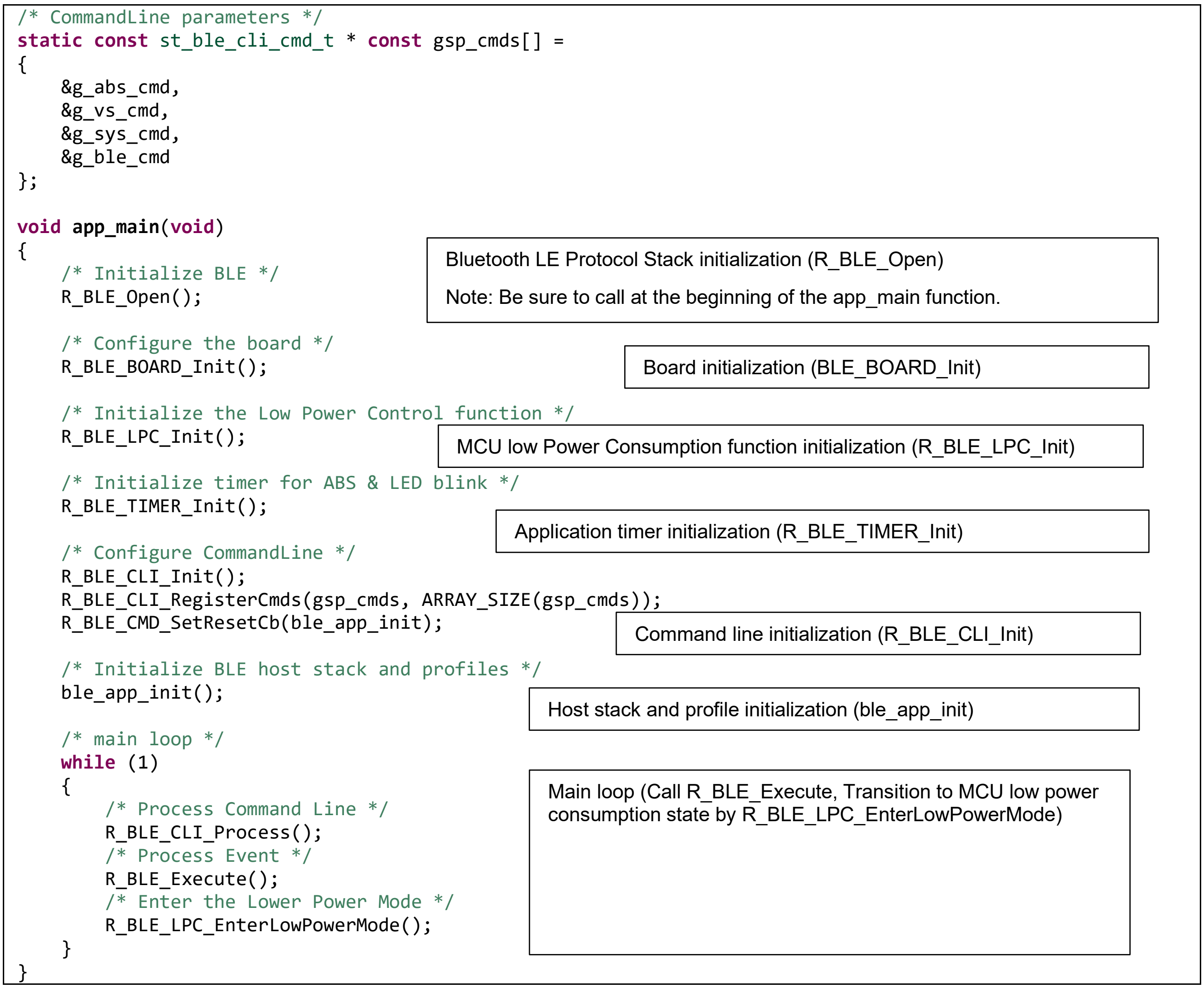
Figure 3-1 Behaviour of skeleton program generated by QE for BLE

3.2 app_main function

The app_main function performs initialization processing and implementation of the main loop. When using the timer, board setting, command line, etc., the initialization process is performed by the app_main function.

Note: When using QE for BLE, the source code of the app_main function is automatically generated.

Code 3-1 shows an example of the app_main function.



```
/* CommandLine parameters */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_abs_cmd,
    &g_vs_cmd,
    &g_sys_cmd,
    &g_ble_cmd
};

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

    /* Configure the board */
    R_BLE_BOARD_Init();

    /* Initialize the Low Power Control function */
    R_BLE_LPC_Init();

    /* Initialize timer for ABS & LED blink */
    R_BLE_TIMER_Init();

    /* Configure CommandLine */
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));
    R_BLE_CMD_SetResetCb(ble_app_init);

    /* Initialize BLE host stack and profiles */
    ble_app_init();

    /* main loop */
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
        /* Process Event */
        R_BLE_Execute();
        /* Enter the Lower Power Mode */
        R_BLE_LPC_EnterLowPowerMode();
    }
}
```

Bluetooth LE Protocol Stack initialization (R_BLE_Open)

Note: Be sure to call at the beginning of the app_main function.

Board initialization (BLE_BOARD_Init)

MCU low Power Consumption function initialization (R_BLE_LPC_Init)

Application timer initialization (R_BLE_TIMER_Init)

Command line initialization (R_BLE_CLI_Init)

Host stack and profile initialization (ble_app_init)

Main loop (Call R_BLE_Execute, Transition to MCU low power consumption state by R_BLE_LPC_EnterLowPowerMode)

Code 3-1 app_main function example

3.2.1 Initialize process (ble_app_init function)

The ble_app_init function initializes the host stack and profile. Register the callback function and GATT database.
Note: When using QE for BLE, the source code of the ble_app_init function is automatically generated.

Code 3-2 shows an example of the ble_app_init function.

```
static ble_status_t ble_app_init(void)
{
    ble_status_t status;

    g_conn_hdl  = BLE_GAP_INVALID_CONN_HDL;
    gs_timer_hdl = BLE_TIMER_INVALID_HDL;

    /* Initialize host stack */
    status = R_BLE_ABS_Init(&gs_abs_init_param);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize GATT Database */
    status = R_BLE_GATTS_SetDbInst(&g_gatt_db_table);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize GATT Server */
    status = R_BLE_SERVS_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize GATT client */
    status = R_BLE_SERVC_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize GATT Discovery Library */
    status = R_BLE_DISC_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize LED and Switch Service */
    status = R_BLE_LSC_Init(lss_cb);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Create timer for LED blink */
    status = R_BLE_TIMER_Create(&gs_timer_hdl, 1, BLE_TIMER_PERIODIC, timer_cb);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    return status;
}
```

Host stack initialization (R_BLE_ABS_Init)
Note: When not using Abstraction API, use below.
R_BLE_GAP_Init
R_BLE_VS_Init
R_BLE_GATTS_Init
R_BLE_GATTC_Init

GATT database registration (R_BLE_GATTS_SetDbInst)
Note: Code-generated when GATT role is set as whichever Server and Client by QE for BLE.

GATT Server function initialization (R_BLE_SERVS_Init)
Note: Code-generated when GATT role is set as whichever Server and Client by QE for BLE.

GATT Client function initialization (R_BLE_SERVC_Init)
Note: Code-generated when GATT role is set as whichever Server and Client by QE for BLE.

Service Discovery function initialization (R_BLE_DISC_Init)
Note: Code-generated when GAP role is set as Central by QE for BLE.

Service initialization
(R_BLE_[service name]S_Init or R_BLE_[service name]C_Init)
Note: Code-generated as R_BLE_[service name]S_Init) when GATT role is set as Server by QE for BLE.
Note: Code-generated as R_BLE_[service name]C_Init) when GATT role is set as Client by QE for BLE.

Software timer creation (R_BLE_TIMER_Create)

Code 3-2 ble_app_init function example

3.2.1.1 Registering callback function

By registering the callback function in the application, it is possible to process at the reception timing of various events. Table 3.2 shows the callback registration API of each function block.

Table 3.2 Callback registration API

Function block	Callback registration API	Comment
GAP	R_BLE_ABS_Init or R_BLE_GAP_Init	Registered callback function is called when receiving the result of R_BLE_GAP_XXX such as Advertising, Scan, Connection establishment and so on.
GATT Server (Profile common)	R_BLE_ABS_Init or R_BLE_GATTS_RegisterCb	Registered callback function is called when accessed from GATT Client.
GATT Client (Profile common)	R_BLE_ABS_Init or R_BLE_GATTC_RegisterCb	Registered callback function is called when accessed from GATT Server.
Service Discovery (Profile common)	R_BLE_DISC_Start()	Registered callback function is called when completing Service Discovery.
Vendor Specific	R_BLE_ABS_Init or R_BLE_VS_Init	Registered callback function is called when receiving the result of R_BLE_VS_XXX.
L2CAP	R_BLE_L2CAP_RegisterCfPsm()	Registered callback function is called when receiving the result of R_BLE_L2CAP_XXX such as that the response of L2CAP Credit-Based Flow Control request is received, L2CAP Credit-Based Flow Control is received and so on. Note: Not code-generated by QE for BLE.
LED and Switch control	R_BLE_BOARD_RegisterSwitchCb()	Registered callback function is called when receiving the result of R_BLE_BOARD_XXX such as that the board switch is pushed and so on. Note: Not code-generated by QE for BLE.
Software timer	R_BLE_TIMER_Create()	Registered callback function is called when receiving the result of R_BLE_TIMER_XXX such as that the specifying time has passed and so on. Note: Not code-generated by QE for BLE.
Server-side profile API	R_BLE_XXXS_Init() (XXX is Service name)	Registered callback function is called when accessed from Client.
Client-side profile API	R_BLE_XXXC_Init() (XXX is Service name)	Registered callback function is called when accessed from Server.

Note: R_BLE_ABS_Init can register GAP, GATT Server, GATT Client, and VS callback functions together.

3.2.1.2 Registering GATT database (R_BLE_GATTS_SetDbInst)

When creating a GATT service application that operates as a GATT server, QE for BLE generates a service database code in the following file.

- gatt_db.c
- gatt_db.h

This GATT database is registered in the application by R_BLE_GATTS_SetDbInst.

3.2.2 Main loop and scheduler (R_BLE_Execute)

Bluetooth LE Protocol Stack uses a scheduler to process the API called by the application. Please call R_BLE_Execute in the main loop to operate the scheduler. The event that occurred is notified to the registered callback function.

The scheduler processes the task according to the message queue sent to the task of each layer of Bluetooth LE Protocol Stack by R_BLE_Execute. Figure 3-2 shows the basic sequence chart of Bluetooth LE Protocol Stack.

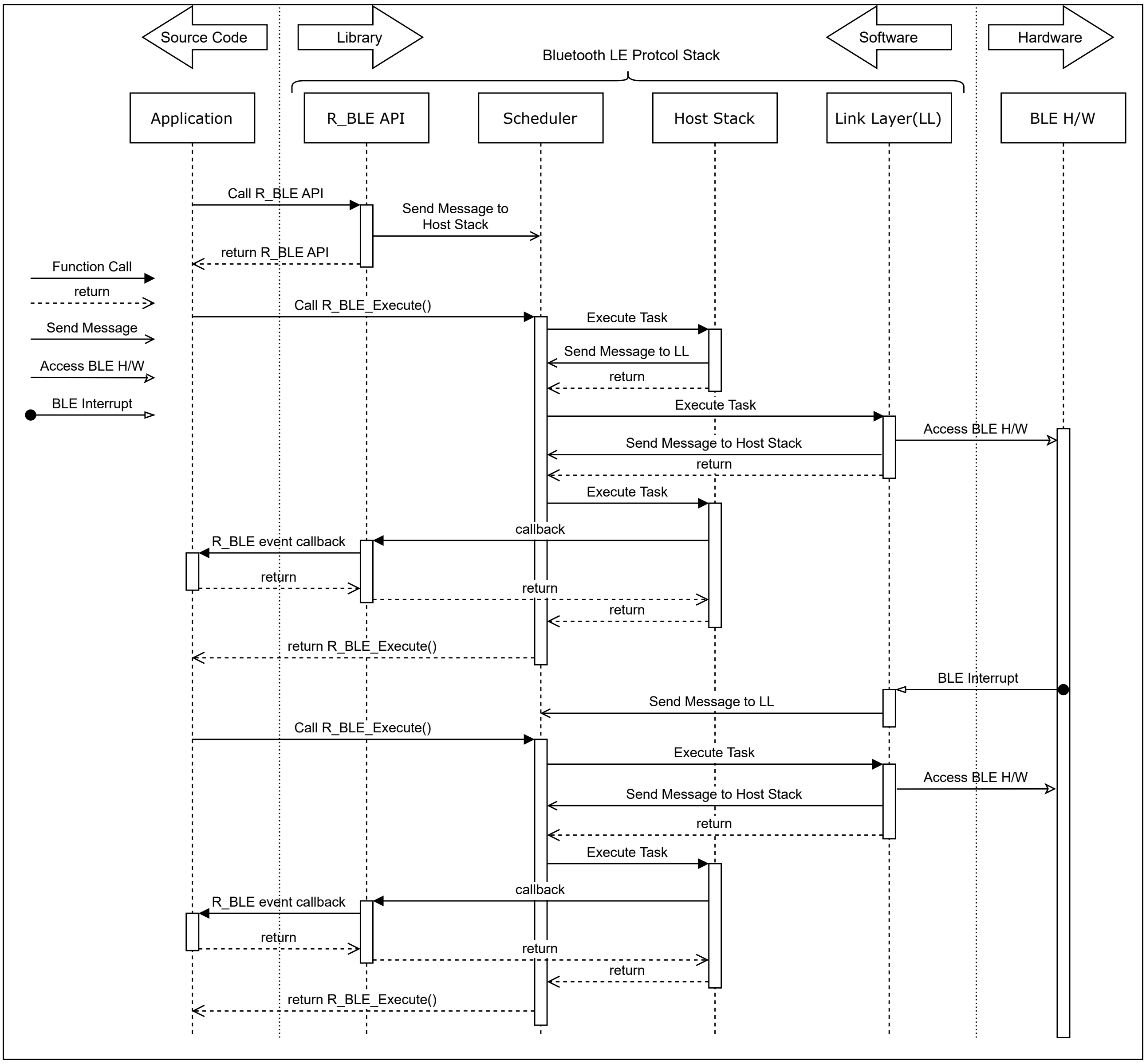


Figure 3-2 Basic sequence chart of Bluetooth LE Protocol Stack

3.2.3 End process

If your application exits from the main loop, call the following APIs to terminate the Bluetooth LE Protocol Stack and app_lib.

[Bluetooth LE Protocol Stack]

R_BLE_Close()

R_BLE_GAP_Terminate()

[Software Timer]

R_BLE_TIMER_Terminate()

[Command Line]

R_BLE_CLI_Terminate()

The Abstraction API provides R_BLE_ABS_Reset() as the finalization sample. This API calls the above APIs.

Note: Since R_BLE_Close() stops RF H/W, when resetting software during RF communication, be sure to call R_BLE_Close() before resetting.

3.3 GAP event (gap_cb function)

GAP callback function receives following events.

```
enum   e_ble_gap_evt_t {
  BLE_GAP_EVENT_INVALID = 0x1001,
  BLE_GAP_EVENT_STACK_ON,
  BLE_GAP_EVENT_STACK_OFF,
  BLE_GAP_EVENT_LOC_VER_INFO,
  BLE_GAP_EVENT_HW_ERR,
  BLE_GAP_EVENT_CMD_ERR = 0x1101,
  BLE_GAP_EVENT_ADV_REPT_IND,
  BLE_GAP_EVENT_ADV_PARAM_SET_COMP,
  BLE_GAP_EVENT_ADV_DATA_UPD_COMP,
  BLE_GAP_EVENT_ADV_ON,
  BLE_GAP_EVENT_ADV_OFF,
  BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP,
  BLE_GAP_EVENT_PERD_ADV_ON,
  BLE_GAP_EVENT_PERD_ADV_OFF,
  BLE_GAP_EVENT_ADV_SET_REMOVE_COMP,
  BLE_GAP_EVENT_SCAN_ON,
  BLE_GAP_EVENT_SCAN_OFF,
  BLE_GAP_EVENT_SCAN_TO,
  BLE_GAP_EVENT_CREATE_CONN_COMP,
  BLE_GAP_EVENT_CONN_IND,
  BLE_GAP_EVENT_DISCONN_IND,
  BLE_GAP_EVENT_CONN_CANCEL_COMP,
  BLE_GAP_EVENT_WHITE_LIST_CONF_COMP,
  BLE_GAP_EVENT_RAND_ADDR_SET_COMP,
  BLE_GAP_EVENT_CH_MAP_RD_COMP,
  BLE_GAP_EVENT_CH_MAP_SET_COMP,
  BLE_GAP_EVENT_RSSI_RD_COMP,
  BLE_GAP_EVENT_GET_REM_DEV_INFO,
  BLE_GAP_EVENT_CONN_PARAM_UPD_COMP,
  BLE_GAP_EVENT_CONN_PARAM_UPD_REQ,
  BLE_GAP_EVENT_AUTH_PL_TO_EXPIRED,
  BLE_GAP_EVENT_SET_DATA_LEN_COMP,
  BLE_GAP_EVENT_DATA_LEN_CHG,
  BLE_GAP_EVENT_RSLV_LIST_CONF_COMP,
  BLE_GAP_EVENT_RPA_EN_COMP,
  BLE_GAP_EVENT_SET_RPA_TO_COMP,
  BLE_GAP_EVENT_RD_RPA_COMP,
  BLE_GAP_EVENT_PHY_UPD,
  BLE_GAP_EVENT_PHY_SET_COMP,
  BLE_GAP_EVENT_DEF_PHY_SET_COMP,
  BLE_GAP_EVENT_PHY_RD_COMP,
  BLE_GAP_EVENT_SCAN_REQ_RECV,
  BLE_GAP_EVENT_CREATE_SYNC_COMP,
  BLE_GAP_EVENT_SYNC_EST,
  BLE_GAP_EVENT_SYNC_TERM,
  BLE_GAP_EVENT_SYNC_LOST,
  BLE_GAP_EVENT_SYNC_CREATE_CANCEL_COMP,
  BLE_GAP_EVENT_PERD_LIST_CONF_COMP,
  BLE_GAP_EVENT_PRIV_MODE_SET_COMP,
  BLE_GAP_EVENT_PAIRING_REQ = 0x1401,
  BLE_GAP_EVENT_PASSKEY_ENTRY_REQ,
  BLE_GAP_EVENT_PASSKEY_DISPLAY_REQ,
  BLE_GAP_EVENT_NUM_COMP_REQ,
  BLE_GAP_EVENT_KEY_PRESS_NTF,
  BLE_GAP_EVENT_PAIRING_COMP,
  BLE_GAP_EVENT_ENC_CHG,
  BLE_GAP_EVENT_PEER_KEY_INFO,
  BLE_GAP_EVENT_EX_KEY_REQ,
  BLE_GAP_EVENT_LTK_REQ,
  BLE_GAP_EVENT_LTK_RSP_COMP,
  BLE_GAP_EVENT_SC_OOB_CREATE_COMP
}
```

Reception condition of the frequently occurring events are shown below.

Table 3.3 Frequently occurring event of GAP callback

Event	Reception condition
BLE_GAP_EVENT_STACK_ON(0x1001)	Complete R_BLE_GAP_Init
BLE_GAP_EVENT_ADV_PARAM_SET_COMP(0x1003)	Complete R_BLE_GAP_SetAdvParam
BLE_GAP_EVENT_ADV_DATA_UPD_COMP (0x1004)	Complete R_BLE_GAP_SetAdvSresData
BLE_GAP_EVENT_ADV_ON (0x1005)	Start Advertising
BLE_GAP_EVENT_ADV_OFF (0x1006)	End Advertising
BLE_GAP_EVENT_SCAN_ON (0x1111)	Start Scan
BLE_GAP_EVENT_SCAN_OFF (0x1112)	End Scan
BLE_GAP_EVENT_CONN_IND (0x1115)	Start Connection
BLE_GAP_EVENT_CONN_IND (0x1115)	End Connection
BLE_GAP_EVENT_DISCONN_IND (0x1116)	End Disconnection

GAP callback function is following.

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    ble_app_gapcb(type, result, p_data);

    switch(type)
    {
        /* TODO: Set callback events of GAP. Check BLE API reference for events. */
    Note: Add processing when an event is received here.
```

Code 3-3 GAP callback function

Note: If the result is other than BLE_SUCCESS, the data notified by p_data will be an undefined value.

3.4 GATTS event (gatts_cb function)

GATT server (GATTS) callback function receives following events.

```
enum   e_r_ble_gatts_evt_t {
   BLE_GATTS_EVENT_EX_MTU_REQ = 0x3002,
   BLE_GATTS_EVENT_READ_BY_TYPE_RSP_COMP = 0x3009,
   BLE_GATTS_EVENT_READ_RSP_COMP = 0x300B,
   BLE_GATTS_EVENT_READ_BLOB_RSP_COMP = 0x300D,
   BLE_GATTS_EVENT_READ_MULTI_RSP_COMP = 0x300F,
   BLE_GATTS_EVENT_WRITE_RSP_COMP = 0x3013,
   BLE_GATTS_EVENT_PREPARE_WRITE_RSP_COMP = 0x3017,
   BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP = 0x3019,
   BLE_GATTS_EVENT_HDL_VAL_CNF = 0x301E,
   BLE_GATTS_EVENT_DB_ACCESS_IND = 0x3040,
   BLE_GATTS_EVENT_CONN_IND = 0x3081,
   BLE_GATTS_EVENT_DISCONN_IND = 0x3082,
   BLE_GATTS_EVENT_INVALID = 0x30FF
}
```

Reception condition of frequently occurring events are shown below.

Table 3.4 Frequently occurring events of GATTS callback

Event	Reception condition
BLE_GATTS_EVENT_CONN_IND(0x3081)	Establish Connection
BLE_GATTS_EVENT_EX_MTU_REQ(0x3002)	Changing MTU is requested from GATT Client after Connection
BLE_GATTS_EVENT_DB_ACCESS_IND(0x3040)	Accessed to GATT database
BLE_GATTS_EVENT_READ_BY_TYPE_RSP_COMP(0x3009)	Complete sending Read By Type Response
BLE_GATTS_EVENT_WRITE_RSP_COMP(0x3013)	Complete sending Write Response
BLE_GATTS_EVENT_HDL_VAL_CNF(0x301E)	Complete receiving Confirmation from GATT Client
BLE_GATTS_EVENT_DISCONN_IND(0x3082)	End Disconnection

GATTS callback function is following.

```
static void gatts_cb(uint16_t type, ble_status_t result, st_ble_gatts_evt_data_t *p_data)
{
    R_BLE_SERVS_GattsCb(type, result, p_data);

    switch(type)
    {
        /* TODO: Set callback events of GATTS. Check BLE API reference for events. */
    Note: Add processing when an event is received here.
```

Code 3-4 GATTS callback function

Note: If the result is other than BLE_SUCCESS, the data notified by p_data will be an undefined value.

3.5 GATTC event (gattc_cb function)

GATT client (GATTC) callback function receives following events.

```
enum   e_r_ble_gattc_evt_t {
   BLE_GATTC_EVENT_ERROR_RSP = 0x4001,
   BLE_GATTC_EVENT_EX_MTU_RSP = 0x4003,
   BLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP = 0x4009,
   BLE_GATTC_EVENT_CHAR_READ_RSP = 0x400B,
   BLE_GATTC_EVENT_CHAR_PART_READ_RSP = 0x400D,
   BLE_GATTC_EVENT_MULTI_CHAR_READ_RSP = 0x400F,
   BLE_GATTC_EVENT_CHAR_WRITE_RSP = 0x4013,
   BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP = 0x4017,
   BLE_GATTC_EVENT_HDL_VAL_NTF = 0x401B,
   BLE_GATTC_EVENT_HDL_VAL_IND = 0x401D,
   BLE_GATTC_EVENT_CONN_IND = 0x4081,
   BLE_GATTC_EVENT_DISCONN_IND = 0x4082,
   BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND = 0x40E0,
   BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND = 0x40E1,
   BLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP = 0x40E2,
   BLE_GATTC_EVENT_PRIM_SERV_DISC_COMP = 0x40E3,
   BLE_GATTC_EVENT_SECOND_SERV_16_DISC_IND = 0x40E4,
   BLE_GATTC_EVENT_SECOND_SERV_128_DISC_IND = 0x40E5,
   BLE_GATTC_EVENT_ALL_SECOND_SERV_DISC_COMP = 0x40E6,
   BLE_GATTC_EVENT_INC_SERV_16_DISC_IND = 0x40E7,
   BLE_GATTC_EVENT_INC_SERV_128_DISC_IND = 0x40E8,
   BLE_GATTC_EVENT_INC_SERV_DISC_COMP = 0x40E9,
   BLE_GATTC_EVENT_CHAR_16_DISC_IND = 0x40EA,
   BLE_GATTC_EVENT_CHAR_128_DISC_IND = 0x40EB,
   BLE_GATTC_EVENT_ALL_CHAR_DISC_COMP = 0x40EC,
   BLE_GATTC_EVENT_CHAR_DISC_COMP = 0x40ED,
   BLE_GATTC_EVENT_CHAR_DESC_16_DISC_IND = 0x40EE,
   BLE_GATTC_EVENT_CHAR_DESC_128_DISC_IND = 0x40EF,
   BLE_GATTC_EVENT_ALL_CHAR_DESC_DISC_COMP = 0x40F0,
   BLE_GATTC_EVENT_LONG_CHAR_READ_COMP = 0x40F1,
   BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP = 0x40F2,
   BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP = 0x40F3,
   BLE_GATTC_EVENT_RELIABLE_WRITES_COMP = 0x40F4,
   BLE_GATTC_EVENT_INVALID = 0x40FF
}
```

Reception condition of frequently occurring events are shown below.

Table 3.5 Frequently occurring events of GATTC callback

Event	Reception condition
BLE_GATTC_EVENT_CONN_IND(0x4081)	Establish Connection
BLE_GATTC_EVENT_EX_MTU_RSP(0x4003)	Request Changing MTU to GATT Server after Connection and receive normal response
BLE_GATTC_EVENT_ERROR_RSP(0x4001)	Receive error response from GATT Server
BLE_GATTC_EVENT_HDL_VAL_NTF(0x401B)	Complete receiving Notification
BLE_GATTC_EVENT_HDL_VAL_IND(0x401D)	Complete receiving Indication
BLE_GATTC_EVENT_DISCONN_IND(0x4082)	End Disconnection

GATTC callback function is following.

```
static void gattc_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
    R_BLE_SERVC_GattcCb(type, result, p_data);

    switch(type)
    {
        /* TODO: Set callback events of GATTC. Check BLE API reference for events. */
    Note: Add processing when an event is received here.
```

Code 3-5 GATTC callback function

Note: If the result is other than BLE_SUCCESS, the data notified by p_data will be an undefined value.

3.6 VS event (vs_cb function)

Vender specific (VS) callback function receives following events.

```
enum   e_r_ble_vs_evt_t {
   BLE_VS_EVENT_SET_TX_POWER = 0x8001,
   BLE_VS_EVENT_GET_TX_POWER = 0x8002,
   BLE_VS_EVENT_TX_TEST_START = 0x8003,
   BLE_VS_EVENT_TX_TEST_TERM = 0x8004,
   BLE_VS_EVENT_RX_TEST_START = 0x8005,
   BLE_VS_EVENT_TEST_END = 0x8006,
   BLE_VS_EVENT_SET_CODING_SCHEME_COMP = 0x8007,
   BLE_VS_EVENT_RF_CONTROL_COMP = 0x8008,
   BLE_VS_EVENT_SET_ADDR_COMP = 0x8009,
   BLE_VS_EVENT_GET_ADDR_COMP = 0x800A,
   BLE_VS_EVENT_GET_RAND = 0x800B,
   BLE_VS_EVENT_TX_FLOW_STATE_CHG = 0x800C,
   BLE_VS_EVENT_FAIL_DETECT = 0x800D,
   BLE_VS_EVENT_SET_SCAN_CH_MAP = 0x800E,
   BLE_VS_EVENT_GET_SCAN_CH_MAP = 0x800F,
   BLE_VS_EVENT_INVALID = 0x80FF
}
```

Reception condition of frequently occurring events are shown below.

Table 3.6 Frequently occurring events of VS callback

Event	Reception condition
BLE_VS_EVENT_SET_TX_POWER(0x8001)	Complete R_BLE_VS_SetTxPower
BLE_VS_EVENT_GET_TX_POWER(0x8002)	Complete R_BLE_VS_GetTxPower
BLE_VS_EVENT_SET_ADDR_COMP(0x8009)	Complete R_BLE_VS_SetBdAddr
BLE_VS_EVENT_GET_ADDR_COMP(0x800A)	Complete R_BLE_VS_GetBdAddr

VS callback function is following.

```
static void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
    R_BLE_SERVS_VsCb(type, result, p_data);

    switch(type)
    {
        /* TODO: Set callback events of VS. Check BLE API reference for events. */
    Note: Add processing when an event is received here.
```

Code 3-6 VS callback function

Note: If the result is other than BLE_SUCCESS, the data notified by p_data will be an undefined value.

3.7 Server-side Profile API event ([service_name]s_cb function)

Callback function of the server-side Profile API receives following events.

```
enum   e_ble_servs_event_t {
   BLE_SERVS_WRITE_REQ = 0x00,
   BLE_SERVS_WRITE_CMD = 0x01,
   BLE_SERVS_WRITE_COMP = 0x02,
   BLE_SERVS_READ_REQ = 0x03,
   BLE_SERVS_HDL_VAL_CNF = 0x04
}

enum   e_ble_[service name]s_event_t {
   BLE_[service name]S_EVENT_[characteristic name]_WRITE_REQ = 0xXX00,
   BLE_[service name]S_EVENT_[characteristic name]_WRITE_CMD= 0xXX01,
   BLE_[service name]S_EVENT_[characteristic name]_WRITE_COMP = 0xXX02,
   BLE_[service name]S_EVENT_[characteristic name]_READ_REQ = 0xXX03,
   BLE_[service name]S_EVENT_[characteristic name]_HDL_VAL_CNF = 0xXX04,
   BLE_[service name]S_EVENT_[characteristic name]_[descriptor name]_WRITE_REQ = 0xYY00,
   BLE_[service name]S_EVENT_[characteristic name]_[descriptor name]_READ_REQ = 0xYY03,
   :
   :
}

Note: The 10th to 15th bits are serial numbers that distinguish attributes (characteristics and descriptors). XX and YY are 00, 04, 08,
10, ..., FC.
```

Reception condition of frequently occurring events are shown below.

Table 3.7 Frequently occurring events of Profile Server callback

Event	Reception condition
XXX_WRITE_REQ (0xXXX0)	Complete receiving Write Request
XXX_WRITE_CMD (0xXXX1)	Complete receiving Write Without Response
XXX_WRITE_COMP (0xXXX2)	Complete sending Write Response
XXX_READ_REQ (0xXXX3)	Complete receiving Read Request
XXX_HDL_VAL_CNF (0xXXX4)	Complete receiving Confirmation

Callback function of server-side profile API is following. (Example of ls service)

```
static void lss_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
    switch(type)
    {
    Note: Add processing when an event is received here.
```

Code 3-7 Profile Server callback function

Note: If the result is other than BLE_SUCCESS, the data notified by p_data will be an undefined value.

3.8 Client-side Profile API event ([service_name]c_cb function)

Callback function of the client-side profile API receives following events.

```
enum   e_ble_servc_event_t {
  BLE_SERVC_WRITE_RSP,
  BLE_SERVC_READ_RSP,
  BLE_SERVC_HDL_VAL_NTF,
  BLE_SERVC_HDL_VAL_IND
}

enum   e_ble_[service name]c_event_t {
  BLE_[service name]C_EVENT_[characteristic name]_WRITE_RSP = 0xXX00,
  BLE_[service name]C_EVENT_[characteristic name]_READ_RSP= 0xXX01,
  BLE_[service name]C_EVENT_[characteristic name]_HDL_VAL_NTF = 0xXX02,
  BLE_[service name]C_EVENT_[characteristic name]_HDL_VAL_IND = 0xXX03,
  BLE_[service name]C_EVENT_[characteristic name]_[descriptor name]_WRITE_RSP = 0xYY00,
  BLE_[service name]C_EVENT_[characteristic name]_[descriptor name]_READ_RSP = 0xYY01,
  :
  :
}
```

Note: The 10th to 15th bits are serial numbers that distinguish attributes (characteristics and descriptors). XX and YY are 00, 04, 08, 10, ..., FC.

Reception condition of the frequently occurring events are shown below.

Table 3.8 Frequently occurring events of Profile Client callback

Event	Reception condition
XXX_WRITE_RSP (0xXXX0)	Complete receiving Write Response
XXX_READ_RSP (0xXXX1)	Complete receiving Read Response
XXX_HDL_VAL_NTF (0xXXX2)	Complete receiving Notification
XXX_HDL_VAL_IND (0xXXX3)	Complete receiving Indication

Callback function of client-side profile API is following. (Example of ls service)

```
static void lsc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
    (void)result;
    (void)p_data;

    switch (type)
    {
        /* TODO: Set callback events of lsc. Check BLE API reference or e_ble_lsc_event_t for events. */
    Note: Add processing when an event is received here.
```

Code 3-8 Profile Client callback function

Note: If the result is other than BLE_SUCCESS, the data notified by p_data will be an undefined value.

3.9 L2CAP event

L2CAP callback function receives following events.

```
enum   e_r_ble_l2cap_cf_evt_t {
   BLE_L2CAP_EVENT_CF_CONN_CNF = 0x5001,
   BLE_L2CAP_EVENT_CF_CONN_IND = 0x5002,
   BLE_L2CAP_EVENT_CF_DISCONN_CNF = 0x5003,
   BLE_L2CAP_EVENT_CF_DISCONN_IND = 0x5004,
   BLE_L2CAP_EVENT_CF_RX_DATA_IND = 0x5005,
   BLE_L2CAP_EVENT_CF_LOW_RX_CRD_IND = 0x5006,
   BLE_L2CAP_EVENT_CF_TX_CRD_IND = 0x5007,
   BLE_L2CAP_EVENT_CF_TX_DATA_CNF = 0x5008,
   BLE_L2CAP_EVENT_CMD_REJ = 0x5009
}
```

L2CAP callback function is following.

```
static void l2cap_cb(uint16_t type, ble_status_t result, st_ble_l2cap_cf_evt_data_t *p_data)
{
    switch (type)
    {
    Note: Add processing when an event is received here.
```

Code 3-9 L2CAP callback function

Note: If the result is other than BLE_SUCCESS, the data notified by p_data will be an undefined value.

3.10 Event notification function (R_BLE_SetEvent)

Calling R_BLE_SetEvent enables to queue events to the scheduler of Bluetooth LE Protocol Stack.

Bluetooth LE Protocol Stack checks the event status with R_BLE_Execute, if the event is queued, calls the callback function registered with R_BLE_SetEvent.

Note: The maximum number of events that can be queued is 8.

This function is mainly used in the following cases.

- The time-consuming processing in the interrupt handler should be executed outside the interrupt handler.
  Note: RF control processing of Bluetooth LE Protocol Stack is processed by MCU with high priority. To reduce the impact on RF control processing, it is recommended that the application processing time is short (recommended time is within 30% of RF idle time after completion of RF event processing). Use R_BLE_SetEvent to divide long-time processing into multiple callbacks and execute.
- The function that cannot be executed in the interrupt handler should be called outside the interrupt handler.

The sequence chart of R_BLE_SetEvent is shown below.

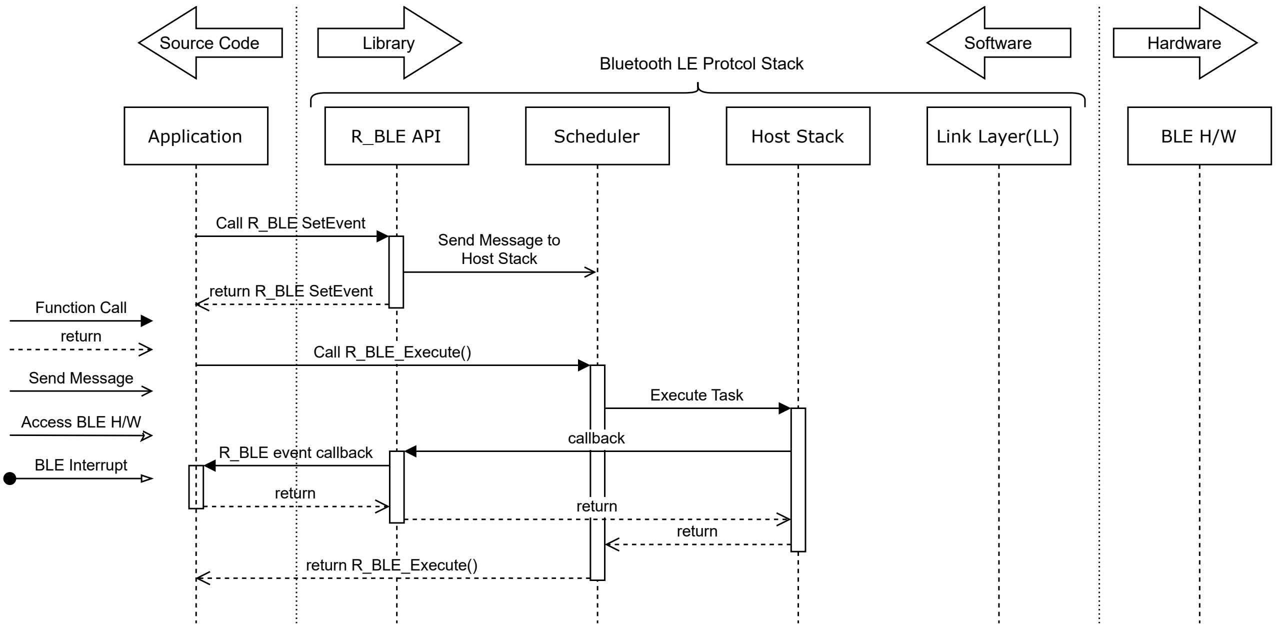


Figure 3-3 Sequence chart of R_BLE_SetEvent

Code 3-10 shows a sample of turning on/off the LED between RF control processing (Advertising). In order to reduce the influence on the next Advertising, the event is queued at the end of Advertising and the LED is turned on and off.

```
[src\smc_gen\r_ble_rx23w\src\platform\r_ble_pf_functions.c]
extern void sw_ntf_recv_event(void);
BLE_SECTION_P void r_ble_rf_notify_event_close(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( (uint16_t)(param>>16) )
    {
        case BLE_EVENT_TYPE_ADV:
        {
            R_BLE_SetEvent( sw_ntf_recv_event );
        } break;
    }
}

[app_main.c]
#include "board/r_ble_board.h"

void sw_ntf_recv_event(void)
{
    R_BLE_BOARD_ToggleLEDState(BLE_BOARD_LED1);
}

void app_main(void)
{
    /* Configure the board */
    R_BLE_BOARD_Init();
```

Note: LED and Switch control function is used. Refer to "1.5.1 Main Functions" to enable the functions and generate code.
Note: RF communication timing notification function is used. Double-click [project name].scfg in e^{2}studio, change [Component] → [Middleware] → [Generic] → [r_ble_rx23w] → [Advertising event close notify.] to “Enable”, and generate code.

Code 3-10 Event notification example (1)

Code 3-11 shows a sample of queuing an event from the interrupt handler sw_cb and turning on/off the LED when SW1 is pressed.

```
[app_main.c]
#include "board/r_ble_board.h"

void sw_ntf_recv_event(void)
{
    R_BLE_BOARD_ToggleLEDState(BLE_BOARD_LED1);
}

static void sw_cb(void)
{
    R_BLE_SetEvent( sw_ntf_recv_event );
}

void app_main(void)
{
    /* Configure the board */
    R_BLE_BOARD_Init();
    R_BLE_BOARD_RegisterSwitchCb(BLE_BOARD_SW2, sw_cb);
```

Note: LED and Switch control function is used. Refer to "1.5.1 Main Functions" to enable the functions and generate code.

Code 3-11 Event notification example (2)

3.11 RF communication timing notification

In order to perform application development synchronized with RF event shown in "1.5 Bluetooth LE Protocol Stack Operation Overview", it is necessary to use the RF communication timing notification function and "3.10 Event notification function (R_BLE_SetEvent)". The following shows how to use RF communication timing notification function.

Select the communication timing wanted to notify from the following settings and set it to "Enable".

Table 3.9 Configuration of RF communication timing notification

Configuration option	Value
BLE_CFG_EVENT_NOTIFY_CONN_START	1: Enable
BLE_CFG_EVENT_NOTIFY_CONN_CLOSE	1: Enable
BLE_CFG_EVENT_NOTIFY_ADV_START	1: Enable
BLE_CFG_EVENT_NOTIFY_ADV_CLOSE	1: Enable
BLE_CFG_EVENT_NOTIFY_SCAN_START	1: Enable
BLE_CFG_EVENT_NOTIFY_SCAN_CLOSE	1: Enable
BLE_CFG_EVENT_NOTIFY_INIT_START	1: Enable
BLE_CFG_EVENT_NOTIFY_INIT_CLOSE	1: Enable
BLE_CFG_EVENT_NOTIFY_DS_START	1: Enable
BLE_CFG_EVENT_NOTIFY_DS_WAKEUP	1: Enable

The sequence chart of RF communication timing notification is shown below.

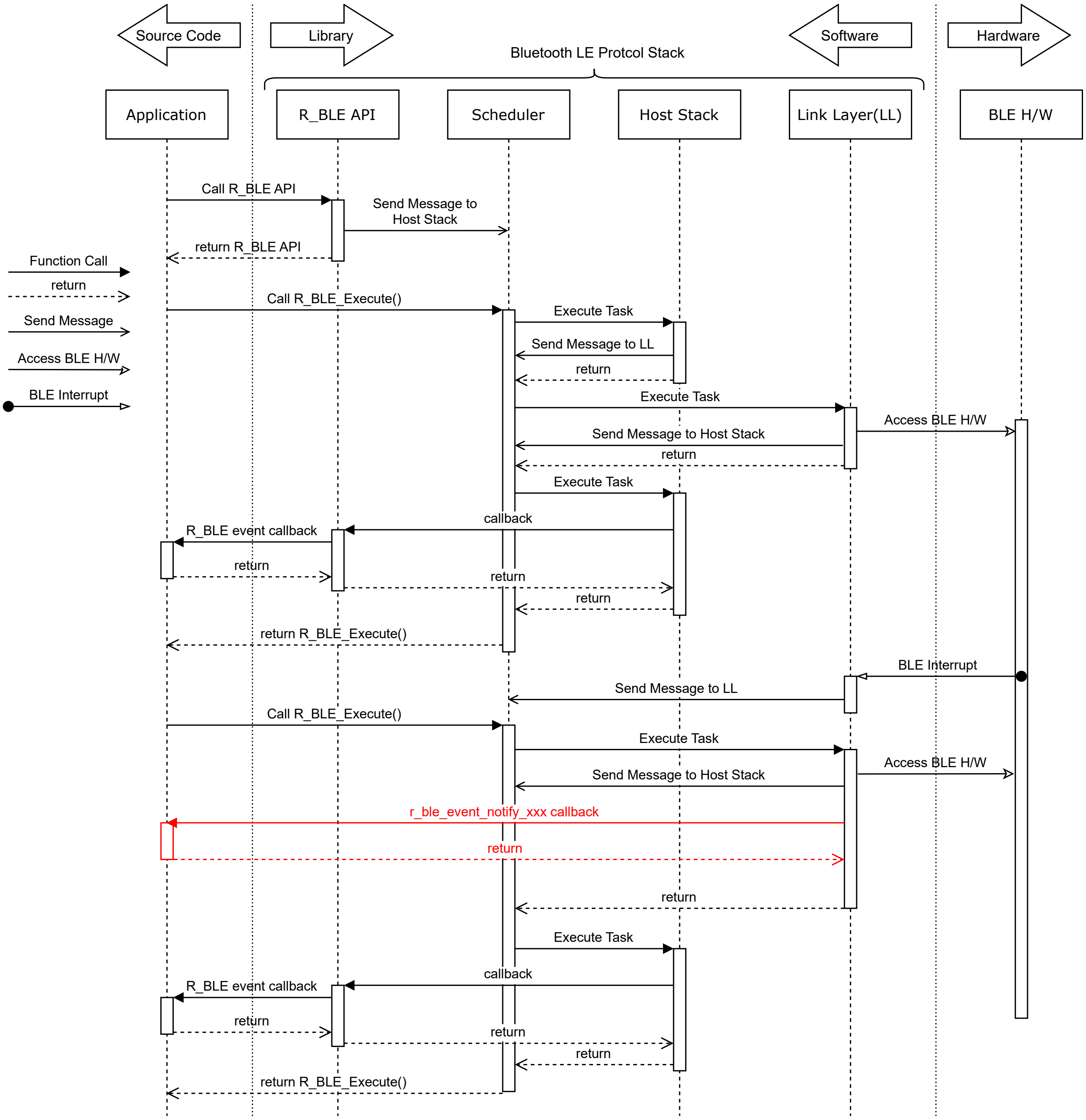


Figure 3-4 Sequence chart of RF communication timing notification

The following is the sample that displays the log on the command line using R_BLE_SetEvent in the reception of RF communication timing. This sample uses Command line function. Enable the function and code-generate, referring to "1.6.1 Primary functions".

The following code makes logs of RF communication timing notification outputted.

```
[src\smc_gen\r_ble_rx23w\src\platform\r_ble_pf_functions.c]
(OMISSION)

#include "cli/r_ble_cli.h"
#define pf R_BLE_CLI_Printf
void rf_ntf_recv_event(void)
{
        pf("RF event has come!!\n");
}

(OMISSION)

BLE_SECTION_P void r_ble_rf_notify_event_start(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( (uint16_t)(param>>16) )
    {
        case 0x0000:/*BLE_EVENT_TYPE_CONN*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0001:/*BLE_EVENT_TYPE_ADV*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0002:/*BLE_EVENT_TYPE_SCAN*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0003:/*BLE_EVENT_TYPE_INITIATOR*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
    }
}

(OMISSION)

BLE_SECTION_P void r_ble_rf_notify_event_close(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( (uint16_t)(param>>16) )
    {
        case 0x0000:/*BLE_EVENT_TYPE_CONN*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0001:/*BLE_EVENT_TYPE_ADV*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0002:/*BLE_EVENT_TYPE_SCAN*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case 0x0003:/*BLE_EVENT_TYPE_INITIATOR*/
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
    }
}

(OMISSION)

BLE_SECTION_P void r_ble_rf_notify_deep_sleep(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( param )
    {
        case BLE_EVENT_TYPE_RF_DS_START:
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
        case BLE_EVENT_TYPE_RF_DS_CLOSE:
        {
            R_BLE_SetEvent( rf_ntf_recv_event );
        } break;
    }
}

(OMISSION)
```

Code 3-12 Sample log display of RF communication timing notification (r_ble_pf_functions.c)

The following code operates only the input and output of Command line function.

```
[app_main.c]
(OMISSION)

#include "cli/r_ble_cli.h"

(OMISSION)

void app_main(void)
{
    (OMISSION)
    /* Configure CommandLine */
    R_BLE_CLI_Init();
    (OMISSION)
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
        (OMISSION)
```

Code 3-13 Sample log display of RF communication timing notification (app_main.c)

4. app_lib

The app_lib is the supplementary library to assist application development. By using app_lib, you can easily realize the basic operation of Bluetooth LE.

4.1 Software Timer

Software Timer provides timer functionality to applications.

The features of Software Timer are as follows:

- Software Timer uses one channel of Compare Match Timer (CMT). To control CMT, CMT FIT Module (r_cmt_rx) is used. A CMT channel used is allocated by the r_cmt_rx dynamically, then the channel is released when application finishes using Software Timer.
- Timeout time of Software Timer is specified in units of milliseconds. Callback function is invoked when Software Timer is triggered and the timeout time expires.
- Software Timer has two operation modes.
  - ➢ Periodic Notification mode (BLE_TIMER_PERIODIC): Once application starts operation of Software Timer channel, expiration of timeout time is notified periodically.
  - ➢ One-Shot Notification mode (BLE_TIMER_ONE_SHOT): When application starts operation of Software Timer channel, expiration of timeout time is notified only once.
- Software Timer has multiple channels. Timeout time, operation mode, and callback function can be set independently for each channel.
- The number of Software Timer channels is defined by the BLE_TIMER_NUM_OF_SLOT macro (default is 10) and can be changed. Note that 24bytes management area is required on RAM per channel.

Notes for using Software Timer are as follows:

- If a long timeout time is specified, or if multiple channels are used, timeout is delayed because CMT operation is started and stopped repeatedly.
- CMT suspends its operation on Software Standby Mode, which is low power consumption mode of CPU. Do not transition to Software Standby Mode while Software Timer is running.
- Abstraction API uses Software Timer. If application uses Abstraction API, invoke the R_BLE_TIMER_Init(), which is the initialization function of Software Timer, before invoking R_BLE_ABS_Init(), which is the initialization function of Abstraction API.

Software Timer provides the following API to applications. For more information about API specification, refer to the R_BLE API documentation.

Table 4.1 Software Timer API

Software Timer API	Description
R_BLE_TIMER_Init	Initialize Software Timer
R_BLE_TIMER_Terminate	Terminate Software Timer
R_BLE_TIMER_Create	Allocate Software Timer channel and set operation parameters
R_BLE_TIMER_Delete	Release Software Timer channel
R_BLE_TIMER_Start	Start operation of Software Timer channel
R_BLE_TIMER_Stop	Stop operation of Software Timer channel
R_BLE_TIMER_UpdateTimeout	Update timeout time and start operation of Software Timer channel

The state transition of Software Timer is shown in Figure 4-1.

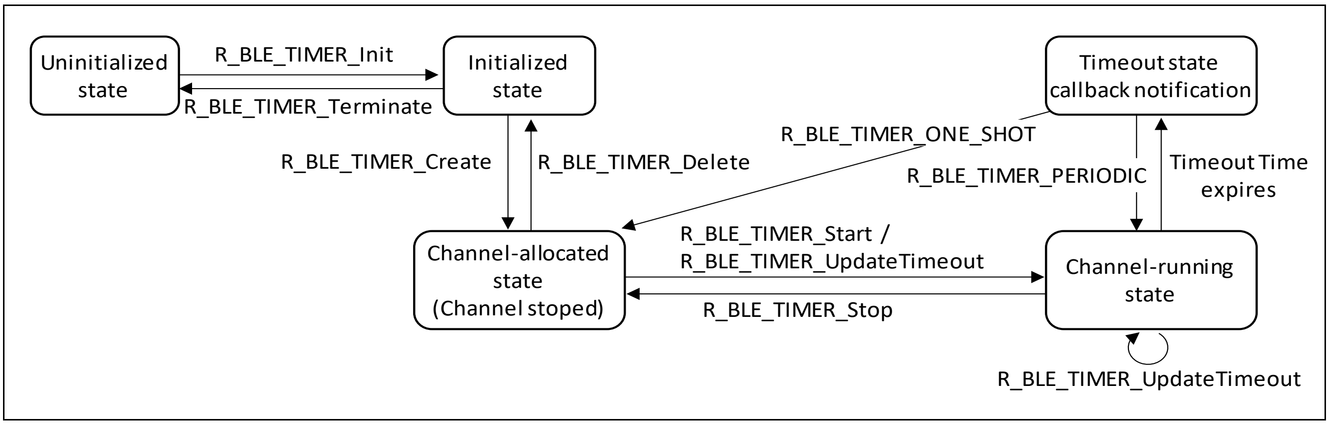


Figure 4-1 State Transition of Software Timer

- R_BLE_TIMER_UpdateTimeout() can be invoked on the Timeout state too.
- R_BLE_TIMER_Delete() can be invoked on the Running state and the Timeout state too.

Example implementation of Software Timer is shown as below.

- Include header file of Software Timer and initialize Software Timer with the R_BLE_TIMER_Init().

```
/* Include Software Timer header */
#include "timer/r_ble_timer.h"


{
        /* Initialize Software Timer */
        R_BLE_TIMER_Init();
}
```

Code 4-1 Initializing Software Timer

- Allocate a channel of Software Timer and specify the following operation parameters with the R_BLE_TIMER_Create(). Also, the R_BLE_TIMER_Create() returns handle value to identify the Software Timer channel allocated.
  - ➢ Timeout Time (in units of milliseconds)
  - ➢ Callback function to notify timeout
  - ➢ Operation Mode: BLE_TIMER_PERIODIC or BLE_TIMER_ONE_SHOT

Note that application attempts to allocate channels more than the number of channels (BLE_TIMER_NUM_OF_SLOT) that can be allocated, the R_BLE_TIMER_Create() returns the BLE_ERR_LIMIT_EXCEEDED error code.

```
static void timer_cb(uint32_t timer_hdl)
{
}

{
        /* Allocate Software Timer channel */
        ble_status_t status;
        status = R_BLE_TIMER_Create(&gs_timer_hdl, 1000, BLE_TIMER_PERIODIC, timer_cb);

        /* Start operation of Software Timer channel */
        R_BLE_TIMER_Start(gs_timer_hdl);
}
```

Code 4-2 Allocating and Starting Software Timer Channel

- Start operation of Software Time channel with either the R_BLE_TIMER_Start() or the R_BLE_TIMER_UpdateTimeout(). When timeout time expires, callback function which is registered with the R_BLE_TIMER_Create() is invoked.
- Operation of Software Timer channel can be stopped with the R_BLE_TIMER_Stop().
- Software Timer channel allocated can be used any number of times.

```
/* Handle of Software Timer Handle */
static uint32_t gs_timer_hdl;

{
        /* Start Operation of Software Timer Channel */
        R_BLE_TIMER_Start(gs_timer_hdl);

        /* Update timeout time and start operation of Software Time channel */
        R_BLE_TIMER_UpdateTimeout(gs_timer_hdl, 500);

        /* Stop operation of Software Timer Channel */
        R_BLE_TIMER_Stop(gs_timer_hdl);
}
```

Code 4-3 Starting, Updating, and Stopping Operation of Software Timer Channel

- If Software Timer channel allocated is no longer needed, it can be released with the R_BLE_TIMER_Delete().

```
{
        /* Release Software Timer Channel */
        R_BLE_TIMER_Delete(&gs_timer_hdl);
}
```

Code 4-4 Releasing Software Timer Channel

- If Software Timer is no longer used, it can be terminated with the R_BLE_TIMER_Terminate(). Note that the R_BLE_TIMER_Terminate() must be invoked after releasing all Software Timer channels.

```
{
        /* Terminate Software Timer */
        R_BLE_TIMER_Terminate();
}
```

Code 4-5 Terminating Software Timer

4.2 Command line

The command line feature provides a function to execute BLE control commands through a terminal emulator that supports VT100 emulation. If you use the command line feature, add the SCI FIT module and the BYTE Queue FIT module in Table 1.8 to your project. Set the configuration options as Table 4.2.

Table 4.2 Configuration options for the command line feature

Configuration option	Value
BLE_CFG_CMD_LINE_EN	1: Enable
BLE_CFG_CMD_LINE_CH	SCI channel for the command line feature. Select one of the following. 1: SCI1 5: SCI5 8: SCI8 12: SCI12 (Only BGA 85pin)

By default, the commands in Table 4.3 are supported. For more information about the commands refer to “Bluetooth Low Energy Protocol Stack Basic Package User’s Manual (R01UW0205)”.

Table 4.3 Supported Command List

Standard Command	Subcommand	Description
gap	adv	Start Advertising.
	scan	Start Scan.
	conn	Send a Connection Request.
	disconn	Disconnect
	device	Display the connecting device list.
	priv	Enable privacy feature in the local device.
	conn_cfg	Configure a connection.
	wl	Register a remote device in the White List.
	auth	Start pairing or encryption.
	sync	Establish a Periodic Sync.
	ver	Display the version information.
vs	txp	Set /Get the transmit power.
	scheme	Set the Coding Scheme of the Coded PHY.
	test	Operate the Direct Test Mode (DTM) to test the RF.
	addr	Set / Get the local BD_ADDR.
	rand	Generate a random number.
sys	stby	Set software standby mode.
ble	reset	Reset the Bluetooth LE Protocol Stack.
	close	Terminate the Bluetooth LE Protocol Stack.

The following sections describe how to change the code to add the command line feature to your application.

4.2.1 How to use the standard commands

(1) Include Header file

Include the below header files for the standard commands.

```
/* Include the header files for standard commands. */
#include "cmd/r_ble_cmd_abs.h"
#include "cmd/r_ble_cmd_vs.h"
#include "cmd/r_ble_cmd_sys.h"
```

Code 4-6 Header files for the standard commands

(2) Initialization and registration of the commands

To use the command line feature, call the APIs in Table 4.4 in application initialization.

Table 4.4 APIs called in the command line feature initialization

API	Description
R_BLE_CLI_Init	Initialize the SCI FIT module.
R_BLE_CLI_RegisterCmds	Register the commands.
R_BLE_CMD_SetResetCb	Register a callback that restarts the Bluetooth LE Protocol Stack after reset.

An example of adding the command line APIs to application initialization is shown in below.

```
/** some code is omitted  **/

/* CommandLine parameters */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_abs_cmd,
    &g_vs_cmd,
    &g_sys_cmd,
    &g_ble_cmd
};

/** some code is omitted  **/
/* Reset BLE Protocol Stack */
static void ble_host_stack_init(void)
{
    ble_app_init();
}

/** some code is omitted  **/

/* Initialize BLE Protocol Stack */
static ble_status_t ble_app_init(void)
{
    ble_status_t status;

    /* Initialize host stack */
    status = R_BLE_ABS_Init(&gs_abs_init_param);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /** some code is omitted  **/
}
/** some code is omitted  **/

void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();
    /** some code is omitted  **/

    /* Configure CommandLine */
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));
    R_BLE_CMD_SetResetCb(ble_host_stack_init);
    /** some code is omitted  **/
}
```

Code 4-7 Sample of adding the command line initialization

(3) Callback

Add the functions in Table 4.5 to the callbacks to process the BLE events in executing command.

Table 4.5 Command line functions added to the callbacks

Callback	Function	Description
GAP Callback	R_BLE_CMD_AbsGapCb	Process the events generated by the gap command.
VS Callback	R_BLE_CMD_VsCb	Process the events generated by the vs command.

An example of adding the command line functions in Table 4.5 to the callback is shown in below.

```
/** some code is omitted  **/
/* GAP Callback */
void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    R_BLE_CMD_AbsGapCb(type, result, p_data);
    /** some code is omitted  **/
}

/** some code is omitted  **/
/* Vendor Specific Callback */
void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
    R_BLE_CMD_VsCb(type, result, p_data);
    /** some code is omitted  **/
}
/** some code is omitted  **/
```

Code 4-8 Sample of adding the command line function to the callbacks

(4) Main loop

To execute a command, add the below function to the application main loop.

Table 4.6 Command line function added to the main loop

API	Description
R_BLE_CLI_Process	Process the characters input through a terminal emulator.

An example of adding the command line function in Table 4.6 to the main loop is shown in below.

```
/* main loop */
void app_main(void)
{
    /** some code is omitted  **/
    /* main loop */
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
        /* Process Event */
        R_BLE_Execute();
        /** some code is omitted  **/
     }
}
```

Code 4-9 Sample of adding the command line to the main loop

4.2.2 How to create a user command

In the command line feature, you can create your own commands by defining commands in the st_ble_cli_cmd_t type variable. This section describes an example of creating a new command to operate the custom profile LED Switch service Client (hereafter “lsc”) provided in the demo project.

(1) Include header files

Include r_ble_cmd.h and r_ble_clli.h for the command line interface.

```
/* Include the header files for command line. */
#include "cmd/r_ble_cmd.h"
#include "cli/r_ble_cli.h"
```

Code 4-10 Command line header files

(2) Command definition

Define command name, subcommand group, number of subcommands, and the message string output by “help” command. For “lsc” command, define a command structure variable as shown below.

```
/* Command definition */
const st_ble_cli_cmd_t g_lsc_cmd =
{
    .p_name      = "lsc",                    /* Command name */
    .p_cmds      = lsc_sub_cmds,             /* Subcommand group */
    .num_of_cmds = ARRAY_SIZE(lsc_sub_cmds), /* Number of subcommands */
    .p_help       = "Sub Command: set_switch_state_ntf, write_led_blink_rate\n"
                    "Try 'lsc sub-cmd help' for more information", /* Message for help */
};
```

Code 4-11 Sample of command definitions

(3) Subcommand definition

Define subcommand. For “lsc” command, define a subcommand structure variable as shown in below. If you want to create a command such as the "Connection command” or "Scan command” that manually abort the process, you need to set an abort handler. During execution of a command for which the abort handler is set, no other command input will be accepted until the command execution is aborted by pressing Ctrl + C key.

```
/* Subcommand definition */
static const st_ble_cli_cmd_t lsc_set_switch_state_ntf_cmd =
{
    .p_name = "set_switch_state_ntf",                         /* Subcommand name */
    .exec = cmd_lsc_set_switch_state_ntf,                    /* Subcommand function */
    .p_help = "Usage: lsc set_switch_state_ntf conn_hdl value",  /* Message for help */
};

/** some code is omitted  **/

/* Subcommand definition */
static const st_ble_cli_cmd_t lsc_write_led_blink_rate_cmd =
{
    .p_name = "write_led_blink_rate",                           /* Subcommand name */
    .exec = cmd_lsc_write_led_blink_rate,                       /* Subcommand function */
    .p_help = "Usage: lsc write_led_blink_rate conn_hdl blink_rate", /* Message for help */
};

/** some code is omitted  **/

/* Subcommand definition */
static const st_ble_cli_cmd_t lsc_conn_lss_cmd =
{
    .p_name = "conn_lss",                                        /* Subcommand name */
    .exec = cmd_lsc_conn_lss,                                    /* Subcommand function */
    .abort = abort_lsc_conn,                                      /* Abort handler */
    .p_help = "Usage: lsc conn_lss XX:XX:XX:XX:XX:XX addr_type", /* Message for help */
};

/** some code is omitted  **/

/* Subcommand group */
static const st_ble_cli_cmd_t * const lsc_sub_cmds[] =
{
    &lsc_set_switch_state_ntf_cmd,  /* Subcommand */
    &lsc_write_led_blink_rate_cmd,  /* Subcommand */
    &lsc_conn_lss_cmd,              /* Subcommand */
};
```

Code 4-12 Sample of Subcommand definitions

(4) Subcommand function definition

Define the function to be processed when the subcommand is executed. For “lsc” command, define a subcommand function as shown in below.

```
/*----------------------------------------------------------------------
lsc set_switch_state_ntf command
----------------------------------------------------------------------*/
static void cmd_lsc_set_switch_state_ntf(int argc, char *argv[])
{
    if (argc != 3)
    {
        pf("lsc %s: unrecognized operands\n", argv[0]);
        return;
    }

    uint16_t conn_hdl;
    conn_hdl = (uint16_t)strtol(argv[1], NULL, 0);

    long value = strtol(argv[2], NULL, 0);
    ble_status_t ret;
    ret = R_BLE_LSC_WriteSwitchStateCliCnfg(conn_hdl, (uint16_t *)&value);

    if (ret != BLE_SUCCESS)
    {
        pf("lsc %s: failed with 0x%04X\n", argv[0], ret);
        return;
    }
}
```

Code 4-13 Sample of Subcommand function definition

(5) Abort handler

Define a function to stop by pressing Ctrl + C key in executing subcommand. An example of an abort handler is shown below.

```
/*----------------------------------------------------------------------
lsc connect lss abort handler
----------------------------------------------------------------------*/
static void abort_lsc_conn(void)
{
    R_BLE_GAP_CancelCreateConn();
}
```

Code 4-14 Sample of Abort handler

(6) Registering commands

After defining the command and subcommand, register the command using R_BLE_CLI_RegisterCmds() API as shown in below so that it can be used as an application-specific command.

```
/* Registering commands */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_abs_cmd,
    &g_vs_cmd,
    &g_lsc_cmd  /* Command to be added */
};

/** some code is omitted  **/

void app_main(void)
{
    /** some code is omitted  **/

    R_BLE_CLI_Init();                                        /* Initialize the command line */
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));  /* Register commands */

    /** some code is omitted  **/
    /* main loop */
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
        /* Process Event */
        R_BLE_Execute();
        /** some code is omitted  **/
     }
}
```

Code 4-15 Sample of initialization and command registration

4.3 Logger

The Logger function provides the following log message output functions.

- Three log levels (ERROR, WARNING, DEBUG)
- Log message output is written in the same format as *printf()* function.
- Provision of functions that convert BD addresses and UUID into character strings.

Log messages are output on “Renesas Debug Virtual Console” (debug console) feature in e^{2} studio.

Therefore, it is possible to display arbitrary character strings even in an application environment that does not use a terminal emulator such as a command line interface feature. However, if there are many outputs to the debug console, the MCU processing may be occupied and BLE communication may not be performed normally. If a phenomenon such as BLE communication disconnection occurs while using the logger function, disable the logger function or reduce the output information.

The log level is set using BLE_CFG_LOG_LEVEL configuration option that specifies the log level for the entire project. Table 4.7 shows the setting values of BLE_CFG_LOG_LEVEL and log output settings.

Table 4.7 Setting BLE_CFG_LOG_LEVEL

BLE_CFG_LOG_LEVEL value	Description
0	No log message output
1	ERROR log message output
2	ERROR and WARNING log message output
3	ERROR and WARNING and DEBUG log message output

The log output uses the macro for log output shown in Table 4.7, which is defined in r_ble_logger.h.

Table 4.8 Setting BLE_CFG_LOG_LEVEL

Macro Name	LOG_LABEL	Description
BLE_LOG_ERR	ERR	For ERROR log message output
BLE_LOG_WRN	WRN	For WARNING log message output
BLE_LOG_DBG	DBG	For DEBUG log message output

Use the log message output macro to set the log in the same format as *printf()* as follows.

```
BLE_LOG_DBG("BLE_GAP_EVENT_STACK_ON \n");
```

Log messages are output in the following format.

```
module_tag: [LOG_LABEL] (function:line) log_body \n
```

“module_tag” can specify the tag to be added to the log for each module by “BLE_LOG_TAG” macro. Define “BLE_LOG_TAG” macro before including r_ble_logger.h.

```
#define BLE_LOG_TAG “app_main”
#include “logger/r_lib_logger.h”
```

Code 4-16 Inclusion of logger header

In the previous example, the log is output as follows: “module_tag” is set to “app_main”.

```
app_main: [DBG] (ble_app_gapcb:238) BLE_GAP_EVENT_STACK_ON
```

In addition, BLE_BD_ADDR_STR() and BLE_UUID_STR() functions are provided for output of BD address and 16-bit/128-bit UUID log messages. BLE_BD_ADDR_STR() function returns string in BD address format when BD address byte array and address type are specified as parameters. BLE_UUID_STR() function returns a UUID format string when UUID byte array and UUID type are specified as parameters. Refer to “R_BLE API document (r_ble_api_spec.chm)” for details.

BLE_ADDR_STR() and BLE_UUID_STR() functions are used as follows:

```
BLE_LOG_DBG(“Connected to %s\n”, BLE_ADDR_STR(addr, addr_type));
BLE_LOG_DBG(“UUID: %s\n”, BLE_UUID_STR(uuid, uuid_type));
```

Code 4-17 Sample of using BLE_ADDR_STR() and BLE_UUID_STR()

4.4 Security data management

BLE application can use the security data management to manage the keys exchanged by pairing. This feature stores the keys in RX23W E2 Data Flash. If you use the security data management, add the Flash FIT module in Table 1.8. Set the configuration options in Table 4.9.

Table 4.9 The security data management configuration option

Configuration option	Description
BLE_CFG_EN_SEC_DATA	1: Enable
BLE_CFG_SECD_DATA_DF_BLOCK	The number of the Data Flash block that the security data management uses to store the keys. Range : 0 to 7 Set other than block number for other use.
BLE_CFG_NUM_BOND	Set the number of the bonding information. Range : 1 to 7 When you change this value during development, after writing the firmware, delete the bonding information by R_BLE_GAP_DeleteBondInfo() or “gap auth del remote all” command.

Because the Abstraction API use the security management data, it does not need to implement the following if it is enabled.

4.4.1 Initialization

The security data management is initialized by R_BLE_SECD_Init. The initialization restores the keys in Data Flash and reset those to the Bluetooth LE Protocol Stack.

4.4.2 Restore the local device keys

Restore the local device keys (IRK, CSRK) in Data Flash by R_BLE_SECD_ReadLocInfo. The keys are reset to the Bluetooth LE Protocol Stack by the APIs in Table 9.10. LTK is the common between the local device and the remote device.

4.4.3 Store the local device keys

After generating the local IRK and CSRK, those are stored by R_BLE_SECD_WriteLocInfo.

4.4.4 Store the remote device keys

To store the keys and the key information distributed by the remote device in pairing, call the APIs in Table 4.10 in the GAP Callback.

Table 4.10 APIs used to store the remote device keys.

Security data management API	Description
R_BLE_SECD_RecvRemKeys	Store the keys distributed by remote device.
R_BLE_SECD_WriteRemKeys	Store the key information received from remote device.

An example of storing the keys and the key information received from a remote device is shown below.

```
/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_event_data)
{
    switch(event_type)
    {
        /** some code is omitted  **/
        case BLE_GAP_EVENT_PAIRING_COMP :
            {
                if(BLE_SUCCESS == event_result)
                {
                    st_ble_gap_pairing_info_evt_t * p_param;
                    p_param = (st_ble_gap_pairing_info_evt_t *)p_event_data->p_param;
                    R_BLE_SECD_WriteRemKeys(&p_param->bd_addr, &p_param->auth_info);
                }
            }
        break;

        case BLE_GAP_EVENT_PEER_KEY_INFO :
            {
                st_ble_gap_peer_key_info_evt_t * p_param;
                p_param = (st_ble_gap_peer_key_info_evt_t *)p_event_data->p_param;
                R_BLE_SECD_RecvRemKeys(&p_param->bd_addr, &p_param->key_ex_param);
            }
        break;
         /** some code is omitted  **/
    }
}
```

Code 4-18 Sample of storing received keys and key information

4.5 Board and LED switch

LED and Push-switch on the board can be controlled by setting the configuration options shown in Table 4.11 according to the board environment.

Table 4.11 LED and Push-switch Configuration Options

Configuration Options	Set Value
BLE_CFG_BOARD_LED_SW_EN	1
BLE_CFG_BOARD_TYPE	0 (Customer board) 1 (Target Board) 2 (RSSK)

Include the below header file to control LED and Push-switch.

```
/* Include LED and Push-switch control header file */
#include "board/r_ble_board.h"
```

Code 4-19 Inclusion of LED and Switch control header file

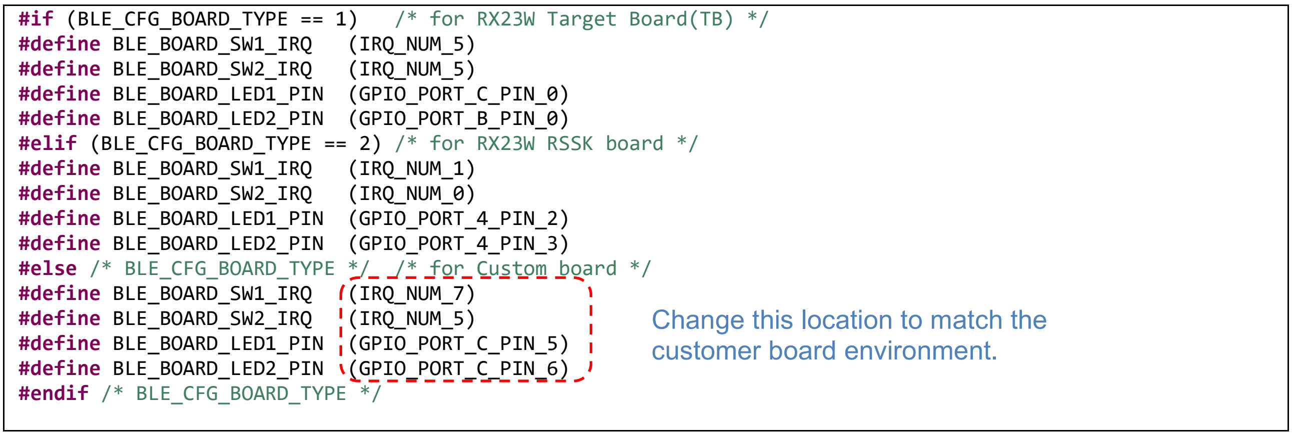
4.5.1 Configuration for customer board

If you use your original board, set the configuration option in Table 4.11 and change the following points in app_lib/board/r_ble_board.c .

(1) Macro definition of LED and Push-Switch (SW)

Change the following macro definition to match the Customer board environment.

- BLE_BOARD_SW1_IRQ
- BLE_BOARD_SW2_IRQ
- BLE_BOARD_LED1_PIN
- BLE_BOARD_LED2_PIN



```
#if (BLE_CFG_BOARD_TYPE == 1)   /* for RX23W Target Board(TB) */
#define BLE_BOARD_SW1_IRQ   (IRQ_NUM_5)
#define BLE_BOARD_SW2_IRQ   (IRQ_NUM_5)
#define BLE_BOARD_LED1_PIN  (GPIO_PORT_C_PIN_0)
#define BLE_BOARD_LED2_PIN  (GPIO_PORT_B_PIN_0)
#elif (BLE_CFG_BOARD_TYPE == 2) /* for RX23W RSSK board */
#define BLE_BOARD_SW1_IRQ   (IRQ_NUM_1)
#define BLE_BOARD_SW2_IRQ   (IRQ_NUM_0)
#define BLE_BOARD_LED1_PIN  (GPIO_PORT_4_PIN_2)
#define BLE_BOARD_LED2_PIN  (GPIO_PORT_4_PIN_3)
#else /* BLE_CFG_BOARD_TYPE */  /* for Custom board */
#define BLE_BOARD_SW1_IRQ   (IRQ_NUM_7)
#define BLE_BOARD_SW2_IRQ   (IRQ_NUM_5)
#define BLE_BOARD_LED1_PIN  (GPIO_PORT_C_PIN_5)
#define BLE_BOARD_LED2_PIN  (GPIO_PORT_C_PIN_6)
#endif /* BLE_CFG_BOARD_TYPE */
```

Change this location to match the customer board environment.

Code 4-20 Changes to LED and SW macro definitions

In the above example, IRQ7 is assigned to SW1 and IRQ5 is assigned to SW2, LED1 is set to PC5 pin, and LED2 is set to PC6 pin.

(2) Register setting in irq_pin_set()

In order to set the pins used in the IRQ for the Push-Switch (SW) on the customer board, change code the MCU register setting location in the *irq_pin_set()* function to match the customer board environment.

```
#if   (BLE_CFG_BOARD_TYPE == 1)
    /*Set IRQ5 pin */
    PORT1.PMR.BIT.B5 = 0U;
    PORT1.PDR.BIT.B5 = 0U;
    MPC.P15PFS.BYTE  = 0x40U;
#elif (BLE_CFG_BOARD_TYPE == 2)
    /*Set IRQ0 pin */
    PORT3.PMR.BIT.B0 = 0U;
    PORT3.PDR.BIT.B0 = 0U;
    MPC.P30PFS.BYTE  = 0x40U;
    /*Set IRQ1 pin */
    PORT3.PMR.BIT.B1 = 0U;
    PORT3.PDR.BIT.B1 = 0U;
    MPC.P31PFS.BYTE  = 0x40U;
#else /* (BLE_CFG_BOARD_TYPE == x) */
    /*Set IRQ5 pin */
    PORT1.PMR.BIT.B5 = 0U;
    PORT1.PDR.BIT.B5 = 0U;
    MPC.P15PFS.BYTE  = 0x40U;
    /*Set IRQ7 pin */
    PORT1.PMR.BIT.B7 = 0U;
    PORT1.PDR.BIT.B7 = 0U;
    MPC.P17PFS.BYTE  = 0x40U;
#endif /* (BLE_CFG_BOARD_TYPE == x) */
```

Change this location to match the customer board environment.

Code 4-21 Changes to irq_pin_set()

In the above example, P17 pin is set for IRQ7 for SW1, and P15 pin is set for IRQ5 for SW2.

4.5.2 Initialization

To control LED and Push-switch, R_BLE_BOARD_Init is call in application initialization.

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

    /* Configure the board */
    R_BLE_BOARD_Init();
    /* some code is omitted.  */

}
```

Code 4-22 Led and SW control initialization

4.5.3 ON or OFF Board LED

The following APIs turns the LED on the board on or off.

- R_BLE_BOARD_SetLEDState
- R_BLE_BOARD_ToggleLEDState

R_BLE_BOARD_SetLEDState specifies the state to be set.

R_BLE_BOARD_ToggleLEDState reverses the LED state.

4.5.4 Callback for pressing SW

Call R_BLE_BOARD_RegisterSwitchCb to register a function to process after pressing SW.

An example of the sw_cb callback called by pressing SW2 is shown below.

```
static void sw_cb(void)
{
    R_BLE_BOARD_ToggleLEDState(BLE_BOARD_LED1);
}

/** some code is omitted  **/

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

    /* Configure the board */
    R_BLE_BOARD_Init();
    R_BLE_BOARD_RegisterSwitchCb(BLE_BOARD_SW2, sw_cb);
    /* some code is omitted. */
}
```

Code 4-23 Sample of callback allocated for SW press

4.6 Abstraction API

The Abstraction API is intended to make it easier to use the functions often used in the Bluetooth LE Protocol Stack. The Abstraction API internally uses GAP, GATT server, GATT client, and Vendor Specific API to realize each function. Table 4.12 shows the APIs called by the Abstraction APIs and the events notified as a result. Refer to the R_BLE API document (r_ble_api_spec.chm) for detailed specifications of each Abstraction API. Do not change the Abstraction API codes.

Table 4.12 APIs and Events used by the Abstraction API

Abstraction API	Description	API to use	Events
R_BLE_ABS_Init	The initialization process is as follows. 1. Initialize the host stack 2. GAP, GATTS, GATTC, Notify VS event For the callback of Register 3. Pairing parameters Configuration	R_BLE_GAP_Init R_BLE_GAP_SetPairingParams R_BLE_VS_Init R_BLE_GATTS_SetDbInst R_BLE_GATTS_Init R_BLE_GATTS_RegisterCb R_BLE_GATTC_Init R_BLE_GATTC_RegisterCb R_BLE_GAP_GetVerInfo R_BLE_SECD_Init R_BLE_SECD_ReadLocInfo R_BLE_GAP_SetLocIdInfo	BLE_GAP_EVENT_STACK_ON BLE_GAP_EVENT_LOC_VER_INFO
R_BLE_ABS_Reset	Bluetooth LE Protocol Stack Perform a reset.	R_BLE_Close R_BLE_GAP_Terminate R_BLE_Open R_BLE_SetEvent	
R_BLE_ABS_StartLegacyAdv	Set the parameters and Advertising Data for Legacy Advertising, and start Advertising.	R_BLE_GAP_SetAdvParam R_BLE_GAP_SetAdvSresData R_BLE_GAP_StartAdv	BLE_GAP_EVENT_ADV_PARAM_SET_COMP BLE_GAP_EVENT_ADV_DATA_UPD_COMP BLE_GAP_EVENT_ADV_ON BLE_GAP_EVENT_ADV_OFF
R_BLE_ABS_StartExtAdv	Set parameters for Extended Advertising and Advertising Data, and start Advertising	R_BLE_GAP_SetAdvParam R_BLE_GAP_SetAdvSresData R_BLE_GAP_StartAdv	BLE_GAP_EVENT_ADV_PARAM_SET_COMP BLE_GAP_EVENT_ADV_DATA_UPD_COMP BLE_GAP_EVENT_ADV_ON BLE_GAP_EVENT_ADV_OFF
R_BLE_ABS_StartNonConnAdv	Set the parameters and Advertising Data for Non-Connectable Advertising and start Advertising.	R_BLE_GAP_SetAdvParam R_BLE_GAP_SetAdvSresData R_BLE_GAP_StartAdv	BLE_GAP_EVENT_ADV_PARAM_SET_COMP BLE_GAP_EVENT_ADV_DATA_UPD_COMP BLE_GAP_EVENT_ADV_ON BLE_GAP_EVENT_ADV_OFF
R_BLE_ABS_StartPerdAdv	Set parameters for Periodic Advertising and Periodic Advertising Data, and start Advertising.	R_BLE_GAP_SetAdvParam R_BLE_GAP_SetAdvSresData R_BLE_GAP_SetPerdAdvParam R_BLE_GAP_StartPerdAdv R_BLE_GAP_StartAdv	BLE_GAP_EVENT_ADV_PARAM_SET_COMP BLE_GAP_EVENT_ADV_DATA_UPD_COMP BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP BLE_GAP_EVENT_PERD_ADV_ON BLE_GAP_EVENT_ADV_ON BLE_GAP_EVENT_ADV_OFF

Abstraction API	Description	API to use	Events
R_BLE_ABS_StartScan	Set up Scan and start.	R_BLE_GAP_StartScan	BLE_GAP_EVENT_SCAN_ON BLE_GAP_EVENT_SCAN_OFF BLE_GAP_EVENT_SCAN_TO BLE_GAP_EVENT_ADV_REPT_IND
R_BLE_ABS_CreateConn	Create a connection request.	R_BLE_TIMER_Create R_BLE_GAP_CreateConn R_BLE_TIMER_Start R_BLE_GAP_CancelCreateConn R_BLE_TIMER_Delete R_BLE_TIMER_Stop R_BLE_TIMER_Delete	BLE_GAP_EVENT_CREATE_CONN_COMP BLE_GAP_EVENT_CONN_CANCEL_COMP BLE_GAP_EVENT_CONN_IND
R_BLE_ABS_SetLocPrivacy	Sets the privacy of the local device.	R_BLE_GAP_EnableRpa R_BLE_VS_GetRand R_BLE_GAP_SetLocIdInfo R_BLE_GAP_ConfRslvList R_BLE_GAP_SetPrivMode	BLE_GAP_EVENT_RPA_EN_COMP BLE_VS_EVENT_GET_RAND BLE_GAP_EVENT_RSLV_LIST_CONF_COMP BLE_GAP_EVENT_PRIV_MODE_SET_COMP
R_BLE_ABS_StartAuth	The pairing will start. If it is already paired, encryption will start.	R_BLE_GAP_GetDevSecInfo R_BLE_GAP_StartPairing R_BLE_GAP_ReplyPasskeyEntry R_BLE_GAP_ReplyNumComp R_BLE_GAP_ReplyExKeyInfoReq R_BLE_GAP_StartEnc R_BLE_GAP_ReplyLtkReq	BLE_GAP_EVENT_PAIRING_REQ BLE_GAP_EVENT_PASSKEY_ENTRY_REQ BLE_GAP_EVENT_PASSKEY_DISPLAY_REQ BLE_GAP_EVENT_NUM_COMP_REQ BLE_GAP_EVENT_KEY_PRESS_NTF BLE_GAP_EVENT_PEER_KEY_INFO BLE_GAP_EVENT_EX_KEY_REQ BLE_GAP_EVENT_PAIRING_COMP BLE_GAP_EVENT_LTK_REQ BLE_GAP_EVENT_LTK_RSP_COMP BLE_GAP_EVENT_ENC_CHG

5. Advertising

Bluetooth LE device sends data to nearby scanning devices by advertising.

5.1 Connecting to smartphone

Figure 5-1 shows the advertising procedure in an application. Details of each step are explained in the following chapters. If you use the Abstraction API, the procedure from 5.2 to 5.2.3 are performed by an Abstraction advertising API call. Regarding to the way of using the API, refer to 5.5.

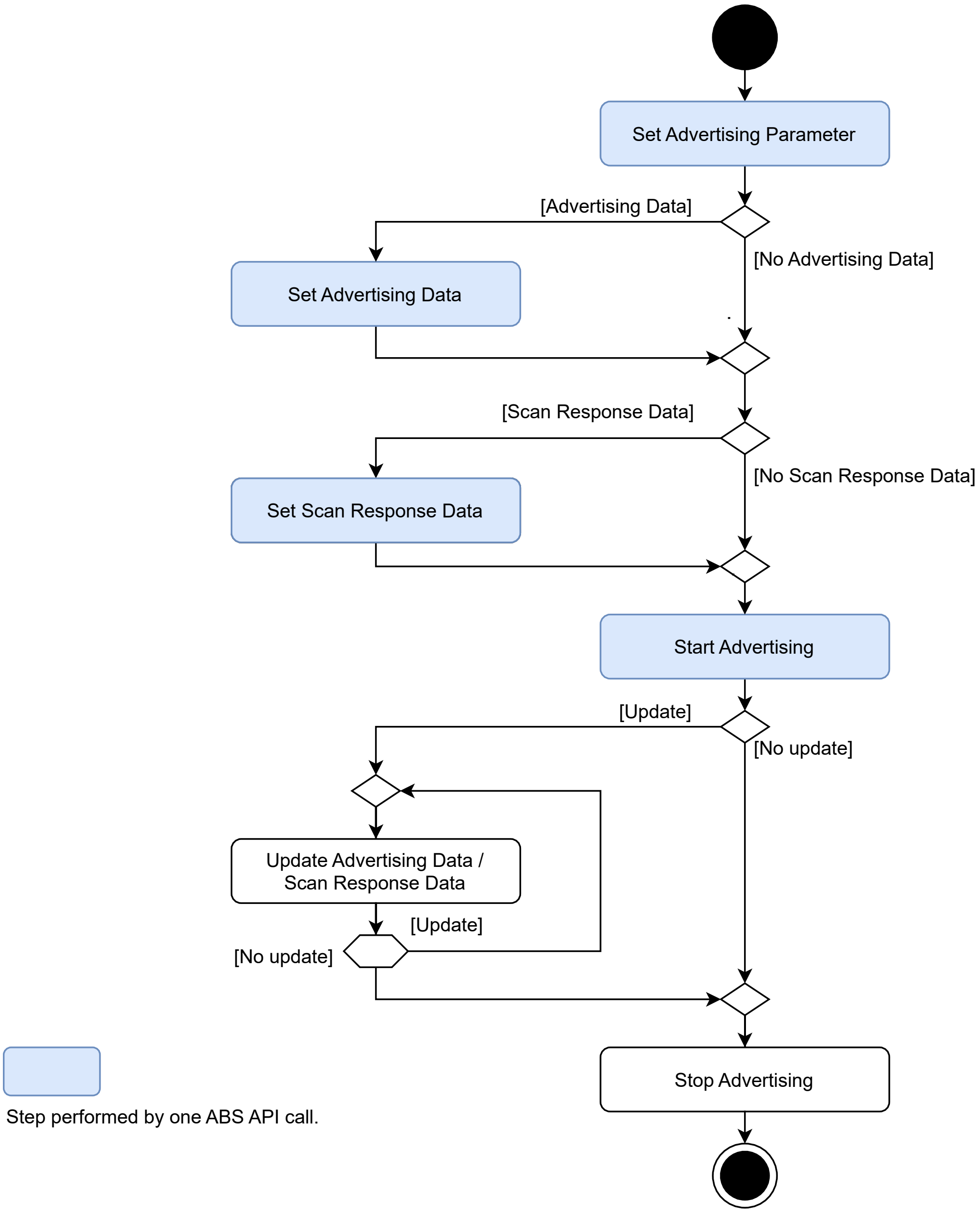


Figure 5-1 Advertising Procedure

5.2 Advertising with GAP API

5.2.1 Advertising Parameter

It is necessary to set the advertising parameters by R_BLE_GAP_SetAdvParam to starting advertising. These parameters cannot be changed during advertising. If you use the Abstraction API, the procedure does not need. The following sections describe the parameter settings for some Use Cases.

5.2.1.1 Adverting Type

Select the advertising type from the below items and set a value in Figure 5-1 to the adv_prop_type field in the st_ble_gap_adv_param_t structure.

- Response to a connection request from remote device (Connectable or Non-Connectable)
- Response to a scan request from remote device (Scannable or Non-Scannable)
- Designation of remote address (Direct or Undirect)
- Type of advertising that a remote device supports (legacy or extended advertising)
- Maximum size of the Advertising Data

Table 5.1 Advertising type and the adv_prop_type field

Advertising Type	Advertising PDU	The adv_prop_type field value	legacy or extended	Max Size(byte)
Connectable and Scannable Undirected *5	ADV_IND	BLE_GAP_LEGACY_PROP_ADV_IND	legacy	31
Connectable Undirected	ADV_EXT_IND AUX_ADV_IND	BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_UNDIRECT	extended	245*1*4
Connectable Directed	ADV_DIRECT_IND	BLE_GAP_LEGACY_PROP_ADV_DIRECT_IND or BLE_GAP_LEGACY_PROP_ADV_HDC_DIRECT_IND	legacy	0
	ADV_EXT_IND AUX_ADV_IND	BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_DIRECT or BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_HDC_DIRECT	extended	239*1*4
Non-Connectable and Non-Scannable Undirected	ADV_NONCONN_IND	BLE_GAP_LEGACY_PROP_ADV_NONCONN_IND	legacy	31
	ADV_EXT_IND AUX_ADV_IND AUX_CHAIN_IND*2	BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_UNDIRECT	extended	BLE_CFG_RF_ADV_DATA_MAX*4
Non-Connectable and Non-Scannable Directed	ADV_EXT_IND AUX_ADV_IND AUX_CHAIN_IND*3	BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_DIRECT or BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_HDC_DIRECT	extended	BLE_CFG_RF_ADV_DATA_MAX*4
Scannable Undirected *5	ADV_SCAN_IND	BLE_GAP_LEGACY_PROP_ADV_SCAN_IND	legacy	31
	ADV_EXT_IND AUX_ADV_IND	BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_UNDIRECT	extended	0
Scannable Directed *5	ADV_EXT_IND AUX_ADV_IND	BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_DIRECT or BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_HDC_DIRECT	extended	0

*1 : If the BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER is added to adv_prop_type, it's Max Size -1 byte.

*2 : If the size of Advertising Data is 245 bytes or less (It's reduced -18 bytes when using Periodic advertising. It's reduced -1 byte when using BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER), since Advertising Data can be sent only with AUX_ADV_IND, AUX_CHAIN ID is not used.

*3 : If the size of Advertising Data is 239 bytes or less (It's reduced -18 bytes when using Periodic advertising. It's reduced -1 byte when using BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER), since Advertising Data can be sent only with AUX_ADV_IND, AUX_CHAIN ID is not used.

*4 : If the size of Advertising Data is 230 bytes or more, since Advertising Data is divided according to Bluetooth specification, combine them on the receiver if necessary.

*5 : The relationship between Scan Response Data and PDU and type is shown in Figure 5-3.

The supported advertising type depends on the Bluetooth LE Protocol Stack library type. All features library supports legacy and extended advertising. Balance and Compact libraries only support the legacy advertising. If a scanner supports only the legacy advertising, it cannot receive extended advertising packets. If the advertising type is extended and non-scannable, each PDU is sent in order shown in Figure 5-2. The advDelay is a random delay from 0 to 10ms.

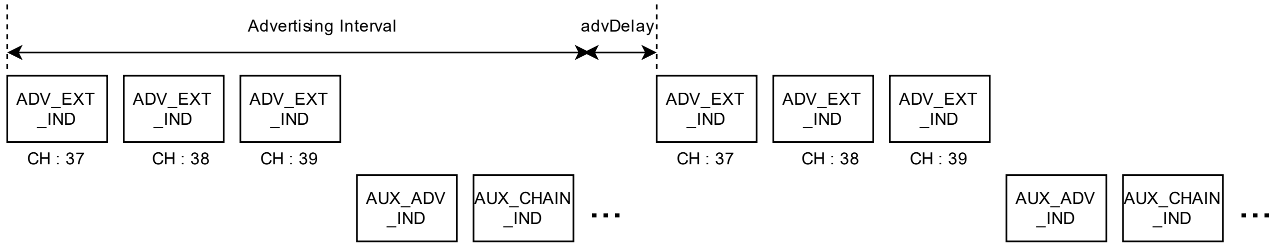


Figure 5-2 Extended Advertising PDU

If the advertising type is scannable and the Scan Response Data is set, the Scan Response Data shown in Table 5.2 are sent as Figure 5-3 against a scan request.

Table 5.2 Scan Response Data

Value set to the adv_prop_type field	Scan Response Data PDU	legacy or extended	Max Size (Byte)
BLE_GAP_LEGACY_PROP_ADV_IND BLE_GAP_LEGACY_PROP_ADV_SCAN_IND	SCAN_RSP	legacy	31
BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_UNDIRECT BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_DIRECT BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_HDC_DIRECT	AUX_SCAN_RSP AUX_CHAIN_IND[*1]	extended	BLE_CFG_RF_ADV_DATA_MAX[*2][*3]

[*1] : If the Scan Response Data is 253 bytes or less (It’s reduced -1 byte when using BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER), since Scan Response Data can be sent only with AUX_SCAN_RSP, AUX_CHAIN ID is not used.

[*2] : If the BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER is added to adv_prop_type, it’s Max Size -1 byte.

[*3] : If the size of Scan Response Data is 230 bytes or more, since Scan Response Data is divided according to Bluetooth specification, combine them on the receiver if necessary.

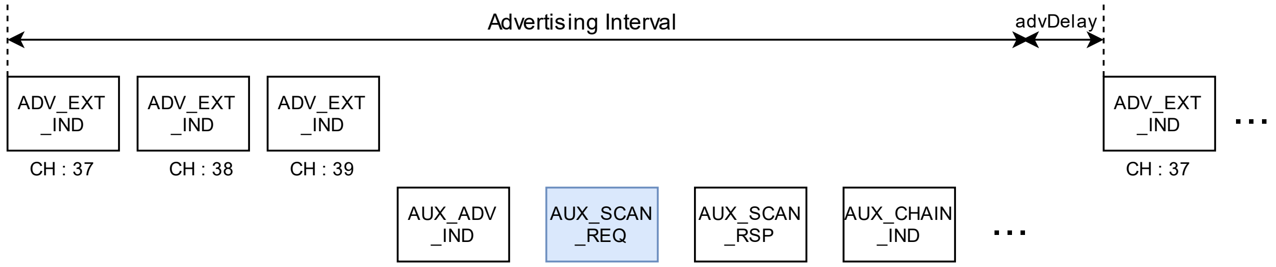


Figure 5-3 Scannable Advertising PDU

The blue box shows the PDU from a remote device.

If the advertising type is Direct, set a remote device address to the p_addr_type and the p_addr field in the st_ble_gap_adv_param_t structure.

If the advertising type is Extended, set the PHY that sends Advertising to the adv_phy and the sec_adv_phy field in the st_ble_gap_adv_param_t structure. Specify the PHY (1M PHY or Coded PHY) of the primary channel (CH:37/38/39) for adv_phy. Specify the PHY (1M PHY, 2M PHY or Coded PHY) of the secondary channel (other than CH:37/38/39) for sec_adv_phy.

5.2.1.2 Using the White List (Respond to a known device)

If the advertising type is Connectable and Scannable, using the White List can filter remote devices that sends a request. If the requesting device BD_ADDR is known to the local device, perform the 1, 2 steps.

1. Register a known device BD_ADDR to the White List
   Call R_BLE_GAP_ConfWhiteList to register a known device.

Note: The White List cannot be added/deleted when the White List filter enabled operation (advertising, scanning, connection request) is executed.

2. Set the Advertising filter policy
   Set the value in Table 5.3 to the filter_policy field in the st_ble_gap_adv_param_t structure.

Table 5.3 The value set to the filter_policy field

Value set to the filter_policy field	Description
BLE_GAP_SCAN_ALLOW_ ADV_ALL(0x00)	Respond to scan requests and connection requests from all devices.
BLE_GAP_ADV_ALLOW_ SCAN_WLST_CONN_ANY(0x01)	Respond to scan requests from whitelisted devices and respond to connection requests from all devices.
BLE_GAP_ADV_ALLOW_ SCAN_ANY_CONN_WLST(0x02)	Respond to scan requests from all devices and respond to connection requests from whitelisted devices.
BLE_GAP_ADV_ALLOW_ SCAN_WLST_CONN_WLST(0x03)	Respond to scan requests and connection requests from whitelisted devices.

5.2.1.3 Privacy

The privacy feature is available to prevent the other devices from tracing the advertising packets. Prepare for the privacy feature in advance according to “9.4.1 Generate local device RPA”. Set the value in Table 5.4 to the field in the st_ble_gap_adv_param_t structure and the address included in the advertising packets are changed to a different address periodically (the default update interval is 900 seconds. Interval can be changed with R_BLE_GAP_SetRpaTo()).

Table 5.4 The parameters used for the privacy feature

Field	Value	Description
o_addr_type	BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Public Address.
	BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Static Address.
o_addr	Specify the Static Address registered by R_BLE_GAP_SetLocIdInfo.	Specify the value if the o_addr_type is BLE_GAP_ADDR_RPA_ID_RANDOM(0x03).
p_addr_type p_addr	Specify the remote device Identity Address registered by R_BLE_GAP_ConfRslvList().	—

BP: Including advertising data that uniquely identifies a device may defeat the purpose of using private addresses to hide the device. Advertisement payloads obfuscation is recommended when using private addressing.

BP: The advertising data can be used to track the device even if the device address changes periodically to different addresses. It is therefore recommended to update the address and data at the same time.

5.2.1.4 Concurrent Execution

If All features library is used, the number of the BLE_CFG_RF_ADV_SET_MAX value advertisings are available concurrently. The advertisings are identified by the advertising handle shown by the adv_hdl field in the st_ble_gap_ext_adv_param_t structure. In each of the procedures in Figure 5-1, the target advertising is specified by the advertising handle.

Balance and Compact libraries are available only one advertising concurrently.

If the Abstraction API and the GAP API are simultaneously used, note that the advertising handle is not available during advertising.

5.2.2 Advertising Data / Scan Response Data

For details about setting Advertising Data / Scan Response Data, refer to “5.4 Advertising Data / Scan Response Data / Periodic Advertising Data”.

For details updating Advertising Data / Scan Response Data setting, refer to “5.4.2 Advertising Data Update”.

5.2.3 Start Advertising

When starting advertising, call the following API.

```
ble_status_t R_BLE_GAP_StartAdv  ( uint8_t  adv_hdl,
                                   uint16_t  duration,
                                   uint8_t  max_extd_adv_evts)
```

If using the All features library, the API specifies the advertising continuing period (duration x 10ms) or the number of sending advertising packets (max_extd_adv_evts).

5.2.4 Stop Advertising

Connectable advertising terminates when the local device connects to a remote device.

The API for stopping advertising is as follows.

```
ble_status_t R_BLE_GAP_StopAdv ( uint8_t  adv_hdl )
```

If 252 bytes or more Extended Advertising data is to be updated, because it cannot be updated due to Bluetooth specification, the advertising needs to be stopped before update.

5.3 Periodic Advertising with GAP API

Periodic Advertising is used in case of sending at a fixed interval. The All features library supports Periodic Advertising. Figure 5-4 shows the procedure for Periodic Advertising in application. The following sections describes the details of Periodic Advertising procedure.

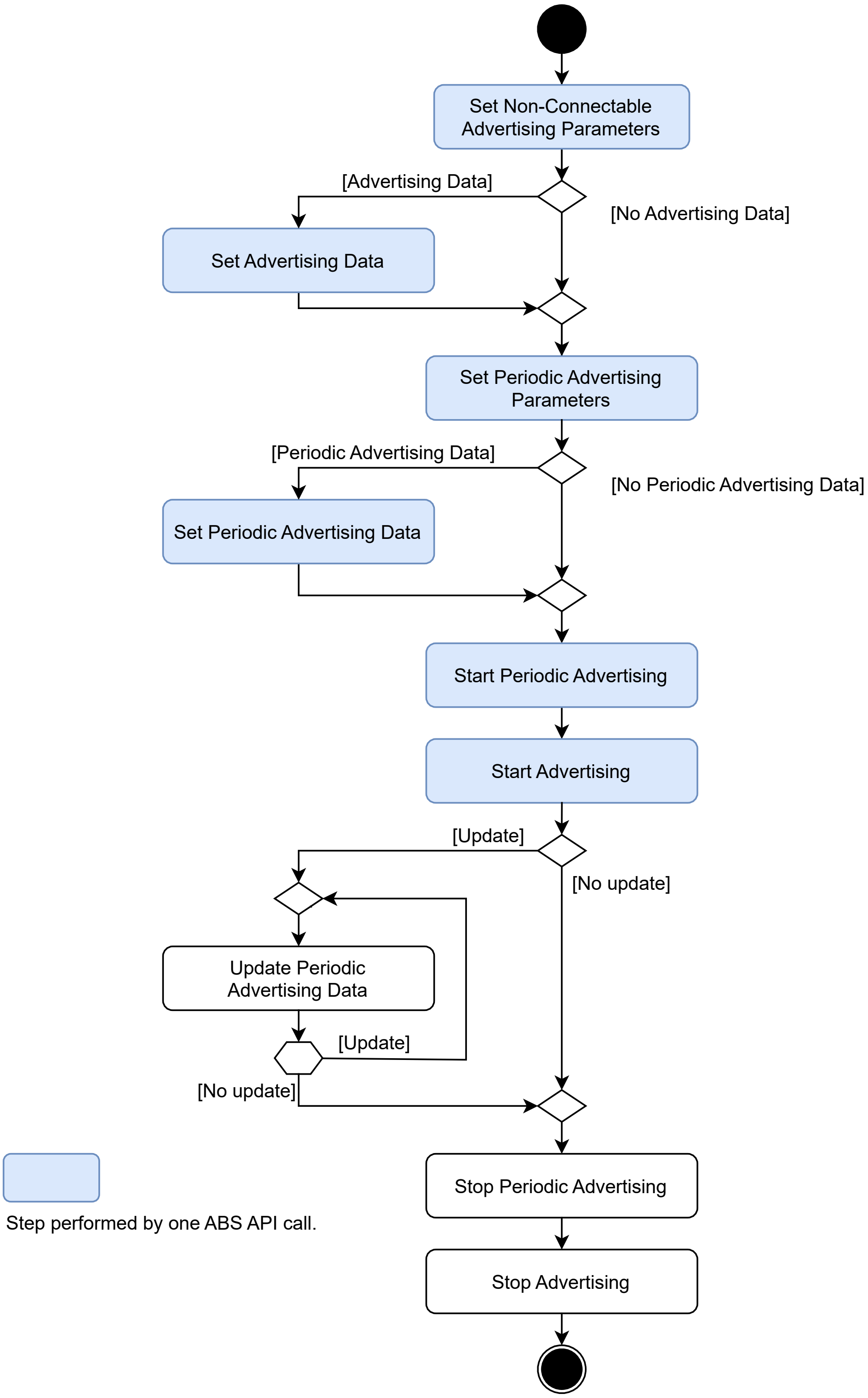

Set Non-Connectable Advertising Parameters
[Advertising Data]
[No Advertising Data]
Set Advertising Data
Set Periodic Advertising Parameters
[Periodic Advertising Data]
[No Periodic Advertising Data]
Set Periodic Advertising Data
Start Periodic Advertising
Start Advertising
[Update]
[No update]
Update Periodic Advertising Data
[Update]
[No update]
Stop Periodic Advertising
Stop Advertising
Step performed by one ABS API call.

Figure 5-4 Periodic Advertising procedure

5.3.1 Non-Connectable Advertising Parameter

Set the advertising parameters by R_BLE_GAP_SetAdvParam to start Periodic Advertising. Non-Connectable advertising in Table 5.1 is used for Periodic Advertising.

- BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_UNDIRECT
- BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_DIRECT
- BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_HDC_DIRECT

5.3.2 Periodic Advertising Parameter

When setting the Periodic Advertising parameters, call the following API.

```
ble_status_t R_BLE_GAP_SetPerdAdvParam(st_ble_gap_perd_adv_param_t * p_perd_adv_param)
```

Setting the Periodic Advertising parameters, AUX_SYNC_IND and AUX_CHAIN_IND PDUs in Table 5.5 follows the Non-Connectable Advertising PDUs (ADV_EXT_INDs and AUX_ADV_IND) the PDUs. Figure 5-5 shows the difference of the intervals by R_BLE_GAP_SetAdvParam and R_BLE_GAP_SetPerdAdvParam.

Table 5.5 Periodic Advertising PDU

Advertising Type	Periodic Advertising PDU	legacy or extended	Maximum Size (Bytes)
Periodic Advertising	AUX_SYNC_IND AUX_CHAIN_IND*1	extended	BLE_CFG_RF_ADV_DATA_MAX*2 *3

*1 : If the size of Periodic Advertising Data is 253 bytes or less (It's reduced -1 byte when using BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER), since Periodic Advertising Data can be sent only with AUX_SYNC_IND, AUX_CHAIN ID is not used.

*2 : If the BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER is added to adv_prop_type, it's Max Size -1 byte.

*3 : If the size of Periodic Advertising Data is 248 bytes or more, since Periodic Advertising Data is divided according to Bluetooth specification, combine them on the receiver if necessary.

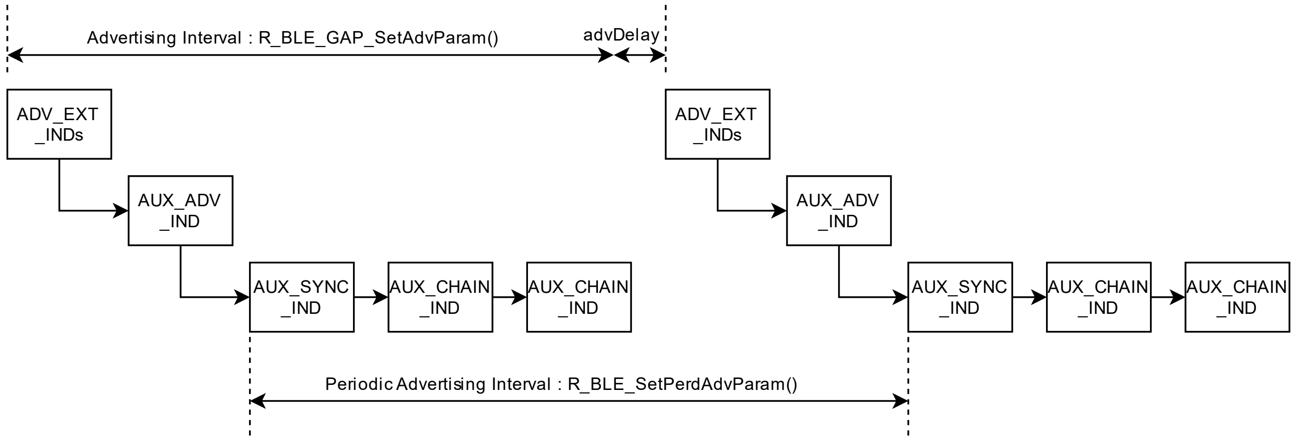


Figure 5-5 Periodic Advertising PDUs

5.3.3 Periodic Advertising Data

For details about setting the Periodic Advertising Data, refer to “5.4 Advertising Data / Scan Response Data / Periodic Advertising Data”.

For details updating Periodic Advertising Data, refer to “5.4.3 Periodic Advertising Data Update”.

5.3.4 Start Periodic Advertising

When starting Periodic Advertising, call the following API.

ble_status_t R_BLE_GAP_StartPerdAdv (uint8_t adv_hdl)

If the Non-Connectable advertising has not been started and the advertising PDUs has not been sent, the Periodic Advertising PDU is not sent by calling this API.

An example of starting Periodic Advertising is shown below.

```
/* Advertising data */
static uint8_t gs_adv_data[] =
{

    /* Flag (mandatory) */
    2,          /* Data Size */
    0x01,       /* Data Type: Flag */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE |
     BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED),    /* Data */

    /* Complete Local Name */
    9,          /* Data Size */
    0x09,       /* Data Type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /* Data */
};

/* Periodic Advertising Data */
static uint8_t gs_perd_adv_data[] =
{

    /* Complete Local Name */
    9,         /* Data Size */
    0xFF,      /* Data Flag: Manufacturer Specific data type  */
    0x36, 0x00,/* Company ID: Renesas Electronics Corporation */
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05,  /* Data */

};

/* some code is omitted. */
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    ble_app_gapcb(type, result, p_data);
    st_ble_gap_adv_set_evt_t * p_adv_set_param;

    switch(type)
    {
        case BLE_GAP_EVENT_STACK_ON :
        {
                st_ble_gap_adv_param_t adv_param =
                {
                    .adv_hdl       = 0x02,
                    .adv_prop_type = BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_UNDIRECT,
                    .adv_intv_min  = 0x0200,
                    .adv_intv_max  = 0x0200,
                    .adv_ch_map    = BLE_GAP_ADV_CH_ALL,
                    .o_addr_type   = BLE_GAP_ADDR_PUBLIC,
                    .filter_policy = BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_ANY,
                    .adv_phy       = BLE_GAP_ADV_PHY_1M,
                    .sec_adv_phy   = BLE_GAP_ADV_PHY_1M,
                };
                /* Set Advertising parameter */
                R_BLE_GAP_SetAdvParam(&adv_param);
        }
        break;

        case BLE_GAP_EVENT_ADV_PARAM_SET_COMP :
        {
                p_adv_set_param = (st_ble_gap_adv_set_evt_t *)p_data->p_param;
                st_ble_gap_adv_data_t adv_data_param = {
                        .adv_hdl        = 0x02,
                        .data_type      = BLE_GAP_ADV_DATA_MODE,
                        .data_length    = ARRAY_SIZE(gs_adv_data),
                        .p_data         = gs_adv_data  ,
                };
                /* Set Advertising Data */
                R_BLE_GAP_SetAdvSresData(&adv_data_param);
        }
        break;

        case BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP :
        {
                /* Periodic Advertising Data parameter */
                st_ble_gap_adv_data_t perd_adv_data_param = {
                        .adv_hdl        = 0x02,
                        .data_type      = BLE_GAP_PERD_ADV_DATA_MODE,
                        .data_length    = ARRAY_SIZE(gs_perd_adv_data),
                        .p_data         = gs_perd_adv_data  ,
                };

                /* Set Periodic Advertising Data */
                R_BLE_GAP_SetAdvSresData(&perd_adv_data_param);
        }
        break;

        case BLE_GAP_EVENT_PERD_ADV_ON :
        {
                p_adv_set_param = (st_ble_gap_adv_set_evt_t *)p_data->p_param;
                /* Start Advertising */
                R_BLE_GAP_StartAdv(0x02, 0, 0);
        }
        break;

        case BLE_GAP_EVENT_ADV_DATA_UPD_COMP :
        {
            st_ble_gap_adv_data_evt_t * p_adv_data_set_param;
            p_adv_data_set_param = (st_ble_gap_adv_data_evt_t *)p_data->p_param;
                if(BLE_GAP_ADV_DATA_MODE == p_adv_data_set_param->data_type)
                {
                        st_ble_gap_perd_adv_param_t perd_param =
                        {
                                .adv_hdl       = 0x02,
                                .prop_type     = 0x0000,
                                .perd_intv_min = 0x0100,
                                .perd_intv_max = 0x0100,
                        };
                        /* Set Periodic Advertising parameter */
                        R_BLE_GAP_SetPerdAdvParam(&perd_param);
                }
                else
                {
                        if(BLE_GAP_PERD_ADV_DATA_MODE == p_adv_data_set_param->data_type)
                        {
                                /* Start Periodic Advertising parameter */
                                R_BLE_GAP_StartPerdAdv(0x02);
                        }
                }
        }
        break;

        default:
                break;
    }
}
```

Code 5-1 Sample of starting Periodic Advertising

5.3.5 Stop Periodic Advertising

The API for stopping Periodic Advertising is as follows.

```
ble_status_t R_BLE_GAP_StopPerdAdv(uint8_t adv_hdl)
```

This API stops only the PDUs in Table 5.5.

If 253 bytes or more Periodic Advertising data is to be updated, because it cannot be updated due to Bluetooth specification, the Periodic Advertising needs to be stopped before update.

5.4 Advertising Data / Scan Response Data / Periodic Advertising Data

Setting Advertising Data / Scan Response Data / Periodic Advertising Data and updating those use R_BLE_GAP_SetAdvSresData. The format of Advertising Data, Scan Response Data and Periodic Advertising Data are same. The data_type field in the st_ble_gap_adv_data_t structure varies as Table 5.6.

Table 5.6 Value set to the data_type field

Data Type	Value set to the data_type field
Advertising Data	BLE_GAP_ADV_DATA_MODE(0x00)
Scan Response Data	BLE_GAP_SCAN_RSP_DATA_MODE(0x01)
Periodic Advertising Data	BLE_GAP_PERD_ADV_DATA_MODE(0x02)

If Scan Response data setting follows Advertising data setting, after calling R_BLE_GAP_SetAdvSresData to set Advertising Data, confirm the Advertising Data setting completion and call R_BLE_GAP_SetAdvSresData to set Scan Response Data in the GAP callback.

5.4.1 Format

Figure 5-6 shows the data format.

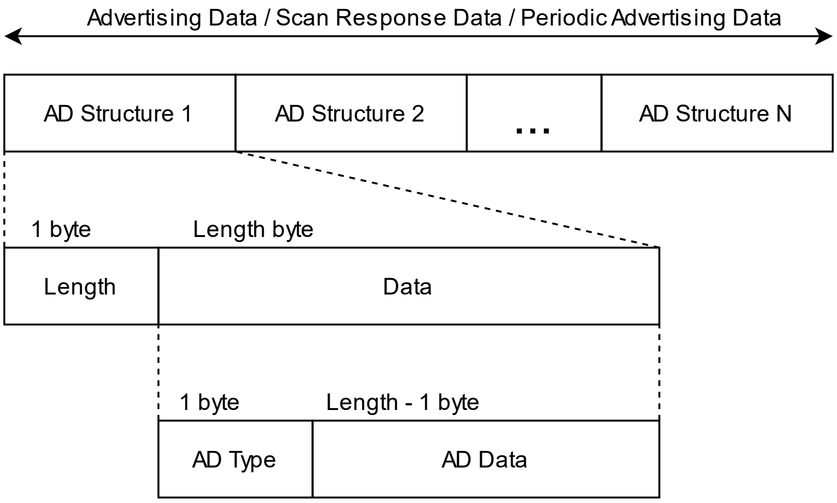


Figure 5-6 Advertising Data / Scan Response Data / Periodic Advertising Data format

Advertising Data / Scan Response Data / Periodic Advertising Data includes one more AD structures. Each AD structure consists of Length and AD Type and AD Data. The Length is the sum of the size of AD type (1 byte) and the size of the AD Data. The AD Type defined by Bluetooth SIG is written in “Supplement to the Bluetooth Core Specification (CSS)”. Table 5.7 shows the AD type often used.

Table 5.7 AD Type and AD Data

<table>
<tr><th colspan="2">Data type</th><th>AD Type</th><th>AD Data</th></tr>
<tr><td colspan="2">Flags</td><td>0x01</td><td>Used for Connectable advertising.
The Flags value used for Bluetooth LE is as follows.
<table><tr><th>Octet</th><th>Bit</th><th>Description</th></tr><tr><td>0</td><td>0</td><td>LE Limited Discoverable Mode</td></tr><tr><td>0</td><td>1</td><td>LE General Discoverable Mode</td></tr><tr><td>0</td><td>2</td><td>BR/EDR Not Supported.</td></tr></table>A scanner is available Discoverable Mode for filtering by the mode.
If adding Discoverable Mode, select Limited or General.</td></tr>
<tr><td rowspan="6">Service UUID</td><td>Incomplete List of 16-bit Service UUIDs</td><td>0x02</td><td rowspan="6">UUID List.
The AD Type varies depending on the size.
If the AD Data includes all UUIDs, select Complete List.
If the AD Data include not all UUIDs, select Incomplete List.</td></tr>
<tr><td>Complete List of 16-bit Service UUIDs</td><td>0x03</td></tr>
<tr><td>Incomplete List of 32-bit Service UUIDs</td><td>0x04</td></tr>
<tr><td>Complete List of 32-bit Service UUIDs</td><td>0x05</td></tr>
<tr><td>Incomplete List of 128-bit Service UUIDs</td><td>0x06</td></tr>
<tr><td>Complete List of 128-bit Service UUIDs</td><td>0x07</td></tr>
<tr><td rowspan="2">Local Name</td><td>Shortened Local Name</td><td>0x08</td><td>Strings that show the first half of the device name.</td></tr>
<tr><td>Complete Local Name</td><td>0x09</td><td>Complete Device Name.</td></tr>
<tr><td colspan="2">Manufacturer Specific Data</td><td>0xFF</td><td>More than 2 bytes manufacturer specific data.
First 2 bytes shows the Company ID.
For details of the Company ID, refer to Assigned Number (https://www.bluetooth.com/specifications/assigned-numbers/)</td></tr>
</table>

An example of setting the Advertising Data including Flags and Complete Local Name and the Scan Response Data including Complete Local Name is shown below.

```
/* Advertising Data */
uint8_t gs_adv_data[] =
{
    /* Flags */
    2,          /*  Data Size: 2byte */
    0x01,       /*  AD type: Flags */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE |
     BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED), /* Data */

    /* Complete Local Name */
    9,         /*  Data Size: 9byte */
    0x09,      /*  AD type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V',  /* Data */
};

/* Scan_Response Data */
uint8_t gs_sres_data[] =
{
    /* Complete Local Name */
    9,         /*  Data Size: 9byte */
    0x09,      /*  AD type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /* Data */

};
/* some code is omitted.  */

/* Advertising Data parameter */
st_ble_gap_adv_data_t adv_data_param = {
        .adv_hdl        = 0x00,
        .data_type      = BLE_GAP_ADV_DATA_MODE,
        .data_length    = ARRAY_SIZE(gs_adv_data),
        .p_data         = gs_adv_data  ,
};

/* Scan_Response Data parameter */
st_ble_gap_adv_data_t sres_data_param = {
        .adv_hdl        = 0x00,
        .data_type      = BLE_GAP_SCAN_RSP_DATA_MODE,
        .data_length    = ARRAY_SIZE(gs_sres_data),
        .p_data         = gs_sres_data,
};
```

```
/* some code is omitted. */

/* Set Advertising Data */
R_BLE_GAP_SetAdvSresData(&adv_data_param);

/* some code is omitted. */

/* GAP Callback  */
void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch(type)
    {
        /* some code is omitted.  */
        case BLE_GAP_EVENT_ADV_DATA_UPD_COMP :
            st_ble_gap_adv_data_evt_t * p_adv_data_set_param;
            p_adv_data_set_param = (st_ble_gap_adv_data_evt_t *)p_data->p_param;
            if((0x00 == p_adv_data_set_param->adv_hdl) &&
               (BLE_GAP_ADV_DATA_MODE == p_adv_data_set_param->data_type))
            {
                R_BLE_GAP_SetAdvSresData(&sres_data_param);
            }
        break;

        /* some code is omitted. */

    }
}
```

Code 5-2 : Sample of setting Advertising Data and Scan Response Data

5.4.2 Advertising Data Update

If the requirement in Table 5.8 is fulfilled, the Advertising Data or the Scan Response Data can be updated in advertising.

Table 5.8 Requirement for updating Advertising Data or Scan Response Data in advertising

Advertising type	Requirement
Legacy advertising	No requirement
Extended advertising	The data length is 251 bytes or less.

Set the following parameters and call R_BLE_GAP_SetAdvSresData to update Advertising Data or Scan Response Data.

```
st_ble_gap_adv_data_t adv_data_param = {
    .adv_hdl        = “Advertising handle of the advertising data to be update”,
    .data_type      = “BLE_GAP_ADV_DATA_MODE or BLE_GAP_SCAN_RSP_DATA_MODE”,
    .data_length    = “Size of the data to be updated”,
    .p_data         = “Pointer to the data to be updated”,
};
```

Code 5-3 Parameters for updating Advertising Data / Scan Response Data

If updating 252 bytes or more Advertising Data in extended advertising, stop the advertising according to “5.2.4” and update the data by R_BLE_GAP_SetAdvSresData.

5.4.3 Periodic Advertising Data Update

If the requirement in Table 5.9 is fulfilled, Periodic Advertising Data can be updated in advertising.

Table 5.9 Requirement for updating Periodic Advertising Data

Advertising type	Requirement
Periodic Advertising	The data length is 252 bytes or less.

Set the following parameters and call R_BLE_GAP_SetAdvSresData to update Periodic Advertising Data.

```
st_ble_gap_adv_data_t adv_data_param = {
    .adv_hdl        = “Advertising handle of the Periodic Advertising Data to be update”,
    .data_type      = BLE_GAP_PERD_ADV_DATA_MODE,
    .data_length    = “Size of the data to be updated”,
    .p_data         = “Pointer to the data to be updated”,
};
```

Code 5-4 Parameters for updating Periodic Advertising Data

If updating 253 bytes or more Periodic Advertising Data in Periodic Advertising, stop the Periodic Advertising according to “5.3.5” and update the data by R_BLE_GAP_SetAdvSresData.

5.4.4 Buffer Size

The size of the buffer for Advertising Data / Scan Response Data in the Bluetooth LE Protocol Stack is 4250 bytes. As shown in Table 5.1, extended advertising can be set Advertising Data or Scan Response Data up to the BLE_CFG_RF_ADV_DATA_MAX value. The sum of Advertising Data / Scan Response Data in advertising simultaneously needs to be 4250 bytes or less.

The size of the buffer for Periodic Advertising Data in the Bluetooth LE Protocol Stack is 4306 bytes. Periodic Advertising can be set Periodic Advertising Data up to the BLE_CFG_RF_ADV_DATA_MAX value. The sum of Periodic Advertising Data in Periodic Advertising simultaneously needs to be 4306 bytes or less.

Figure 5-7 and Figure 5-8 show a sample of Advertising Data in advertising simultaneously. Here the BLE_CFG_RF_ADV_DATA_MAX value is 1650. R_BLE_GAP_GetRemainAdvBufSize() gets the free sizes of the buffer for Advertising Data / Scan Response Data.

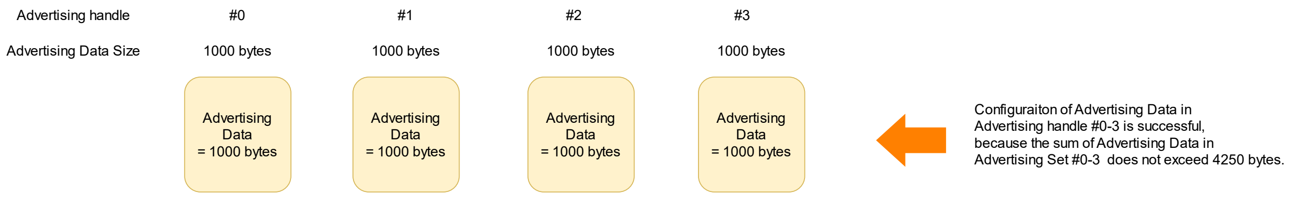

Advertising handle
#0
#1
#2
#3
Advertising Data Size
1000 bytes
1000 bytes
1000 bytes
1000 bytes
Advertising Data = 1000 bytes
Advertising Data = 1000 bytes
Advertising Data = 1000 bytes
Advertising Data = 1000 bytes
Configuraiton of Advertising Data in Advertising handle #0-3 is successful, because the sum of Advertising Data in Advertising Set #0-3 does not exceed 4250 bytes.

Figure 5-7 Successful sample of setting Advertising Data

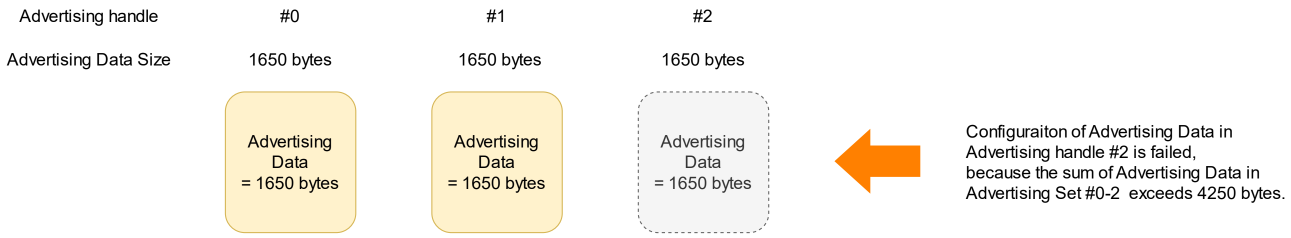

Advertising handle
#0
#1
#2
Advertising Data Size
1650 bytes
1650 bytes
1650 bytes
Advertising Data = 1650 bytes
Advertising Data = 1650 bytes
Advertising Data = 1650 bytes
Configuraiton of Advertising Data in Advertising handle #2 is failed, because the sum of Advertising Data in Advertising Set #0-2 exceeds 4250 bytes.

Figure 5-8 Failed sample of setting Advertising Data

5.5 Advertising with Abstraction API

If you use the Abstraction API, the procedure from setting advertising parameters to starting advertising are performed by an Abstraction API call. Table 5.10 shows the advertising type supported by the Abstraction API.

Table 5.10 Advertising type supported by the Abstraction API

Abstraction API	Legacy or Extended	Advertising Type	Advertising PDU	Advertising handle	Maximum Advertising Data Size (Bytes)
R_BLE_ABS_StartLegacyAdv	Legacy	Connectable and Scannable Undirected	ADV_IND	0	31
R_BLE_ABS_StartExtAdv	Extended	Connectable Undirected	ADV_EXT_IND AUX_ADV_IND	1	245
		Connectable Directed	ADV_EXT_IND AUX_ADV_IND		239
R_BLE_ABS_StartNonConnAdv	Legacy	Non-Connectable and Non-Scannable Undirected	ADV_NONCONN_IND	2	31
	Extended		ADV_EXT_IND AUX_ADV_IND AUX_CHAIN_IND		BLE_CFG_RF_ADV_DATA_MAX
		Non-Connectable and Non-Scannable Directed	ADV_EXT_IND AUX_ADV_IND AUX_CHAIN_IND		BLE_CFG_RF_ADV_DATA_MAX
R_BLE_ABS_StartPerdAdv	Extended	Periodic	ADV_EXT_IND AUX_ADV_IND AUX_SYNC_IND AUX_CHAIN_IND	3	BLE_CFG_RF_ADV_DATA_MAX

If the Abstraction API and the GAP API are simultaneously used, note that the advertising handle is not available during advertising.

5.5.1 White List (Respond to a known device)

The White List is available by R_BLE_ABS_StartLegacyAdv and R_BLE_ABS_StartExtAdv. According to the following procedure, the White List can filter remote devices that sends a request.

1. Register a known device BD_ADDR to the White List
   Call R_BLE_GAP_ConfWhiteList to register a known device.

Note: The White List cannot be added/deleted when the White List filter enabled operation (advertising, scanning, connection request) is executed.

2. Set the Advertising filter policy
   Set the value in Table 5.3 to the filter field in the st_ble_abs_legacy_adv_param_t (if using R_BLE_ABS_StartLegacyAdv) or the st_ble_abs_ext_adv_param_t (if using R_BLE_ABS_StartExtAdv) structure .

5.5.2 Privacy

The privacy feature is available by R_BLE_ABS_StartLegacyAdv, R_BLE_ABS_StartExtAdv, R_BLE_ABS_StartNonConnAdv, R_BLE_ABS_StartPerdAdv. Prepare for the privacy feature in advance according to "9.4.1 Generate local device RPA". Set the value in Table 5.11 to the fields in the st_ble_abs_legacy_adv_param_t or the st_ble_abs_ext_adv_param_t or the st_ble_abs_non_conn_adv_param_t structure. The address included in advertising packets is RPA and are changed to a different address periodically (the default update interval is 900 seconds. Interval can be changed with R_BLE_GAP_SetRpaTo()).

Table 5.11 The parameters used for the privacy feature

Field	Value	Description
o_addr_type	BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Public Address.
	BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Public Address.
o_addr	Specify the Static Address registered by R_BLE_GAP_SetLocIdInfo.	Specify the value if the o_addr_type is BLE_GAP_ADDR_RPA_ID_RANDOM(0x03).
p_addr	Specify the remote device Identity Address registered by R_BLE_GAP_ConfRslvList().	—

5.6 Connection with Smart Phone

Call R_BLE_ABS_StartLegacyAdv to send connectable Legacy Advertising packets to connect to a Smart Phone. An example of sending advertising packets to connect with Smart Phone is shown below.

```
/* Advertising Data */
static uint8_t gs_adv_data[] =
{
    /* Flag (mandatory) */
    2,          /**< Data Size */
    0x01,       /**< Data Flag: Flag */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE | BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED),  /**< Data Value */

    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Flag: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */

};

/* Scan_Response Data */
static uint8_t gs_sres_data[] =
{
    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Flag: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */

};

/* Advertising parameters */
static st_ble_abs_legacy_adv_param_t gs_adv_param =
{
    .slow_adv_intv   = 0x00A0,
    .slow_period     = 0,
    .p_adv_data      = gs_adv_data,
    .adv_data_length = ARRAY_SIZE(gs_adv_data),
    .p_sres_data     = gs_sres_data,
    .sres_data_length = ARRAY_SIZE(gs_sres_data),
    .adv_ch_map      = BLE_GAP_ADV_CH_ALL,
    .filter          = BLE_ABS_ADV_ALLOW_CONN_ANY,
    .o_addr_type     = BLE_GAP_ADDR_PUBLIC,
    .o_addr          = {0},
};

/** some code is omitted  **/

/* Start Advertising */
R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
```

Code 5-5 Sample of advertising for connecting with Smart Phone

When starting advertising, the BLE_GAP_EVENT_ADV_ON event is notified. After the event notification, Smart Phones can detect the device to connect.

5.7 Beacon

An example of sending non-connectable advertising packets as beacon by calling R_BLE_ABS_StartNonConnAdv is shown below.

```
/* Advertising Data */
static uint8_t gs_adv_data[] =
{
    /* Flag */
    2,          /**< Data Size */
    0x01,       /**< Data Flag: Flag */
    BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED,    /**< Data Value */

    /* Complete Local Name */
    9,         /* Data Size */
    0x09,      /* Data Flag: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /* Data */

};

/* Advertising parameters */
static st_ble_abs_non_conn_adv_param_t gs_non_conn_adv_param =
{
    .p_addr          = NULL,
    .p_adv_data      = gs_adv_data,
    .adv_intv        = 0x00A0,
    .duration        = 0,
    .adv_data_length = ARRAY_SIZE(gs_adv_data),
    .adv_ch_map      = BLE_GAP_ADV_CH_ALL,
    .o_addr_type     = BLE_GAP_ADDR_PUBLIC,
    .adv_phy         = BLE_GAP_ADV_PHY_1M,
    .sec_adv_phy     = BLE_GAP_ADV_PHY_1M,
    .o_addr          = {0},
};

/** some code is omitted  **/

/* Start Advertising */
R_BLE_ABS_StartNonConnAdv (&gs_non_conn_adv_param);
```

Code 5-6 Sample of using R_BLE_ABS_StartNonConnAdv

When starting advertising, the BLE_GAP_EVENT_ADV_ON event is notified. After the event notification, a remote device can detect the beacon by scan.

Smart Phone may support only the legacy advertising type of non-connectable advertising packet. Send advertising packets which the scanner can detect the packets.

If you use iBeacon (Apple Inc) or Eddystone (Google), use non-connectable advertising. For more information, refer to the following.

iBeacon	: https://developer.apple.com/ibeacon/
Eddystone	: https://github.com/google/eddystone

6. Scan

Bluetooth LE device receives advertising packets from other devices by scan. If your device scan, use the All features or Balance type Bluetooth LE Protocol Stack library. The All features library can receive the extended advertising and legacy advertising packets. The Balance library receives only the legacy advertising packet.

6.1 Start or stop scan

Scan starts by calling one of the following APIs.

Start Scan API :

- R_BLE_GAP_StartScan
- R_BLE_ABS_StartScan

If the period parameter of the above APIs is set to other than 0, the scan stops after the period is expired. Otherwise scan stops by calling the following API. If the target device is found or you want to change the scan parameters, stop the scan.

Stop Scan API:

- R_BLE_GAP_StopScan

6.2 Scan parameters

Table 6.1 –Table 6.5 show the Start Scan APIs parameters.

[R_BLE_GAP_StartScan]: parameter 1(st_ble_gap_scan_param_t *), parameter 2(st_ble_gap_scan_on_t *)

Table 6.1 st_ble_gap_scan_param_t structure

Type	Field	Description
uint8_t	o_addr_type	Address type included in a scan request packet with active scan.
uint8_t	filter_policy	The filter policy which packets from what kind of device can be received.
st_ble_gap_scan_phy_param_t *	p_phy_param_1M	1MPHY scan parameters.
st_ble_gap_scan_phy_param_t *	p_phy_param_coded	Coded PHY scan parameters.

Table 6.2 st_ble_gap_scan_phy_param_t structure

Type	Field	Description
uint8_t	scan_type	Select active or passive scan. If you use Scan Response Data, select active scan.
uint16_t	scan_intv	Scan interval.
uint16_t	scan_window	Scan window.

Table 6.3 st_ble_gap_scan_on_t structure

Type	Field	Description
uint8_t	proc_type	Scan procedure type.
uint8_t	filter_dups	Specify whether receiving the same advertising packet from the same device or not.
uint16_t	duration	Scan duration.
uint16_t	period	Scan period.

[R_BLE_ABS_StartScan]

Table 6.4 st_ble_abs_scan_param_t structure

Type	Field	Description
st_ble_abs_scan_phy_param_t *	p_phy_param_1M	1MPHY scan parameters.
st_ble_abs_scan_phy_param_t *	p_phy_param_coded	Coded PHY scan parameters.
uint8_t *	p_filter_data	Scan Filtering Data.
uint16_t	fast_period	Fast scan period.
uint16_t	slow_period	Slow scan period.
uint16_t	filter_data_length	Scan Filtering Data size.
uint8_t	dev_filter	The filter policy which packets from what kind of device can be received.
uint8_t	filter_dups	Specify whether receiving the same advertising packet from the same device or not.
uint8_t	filter_ad_type	AD_TYPE of Scan Filtering Data.

Table 6.5 st_ble_abs_scan_phy_param_t structure

Type	Field	Description
uint16_t	fast_intv	Fast scan interval.
uint16_t	slow_intv	Fast scan window.
uint16_t	fast_window	Slow scan interval.
uint16_t	slow_window	Slow scan window.
uint8_t	scan_type	Select active or passive scan. If you use Scan Response Data, select active scan.

The p_phy_param_1M and the p_phy_param_coded field specify the PHY of scan. Setting the p_phy_param_1M is required to receive Advertising that the primary channels (CH:37/38/39) use 1M PHY. Setting the p_phy_param_coded is required to receive Advertising that primary channels use Coded PHY.

The scan interval, scan window, duration and period field specify the interval and period of scan.

Figure 6-1 shows those parameters relationship.

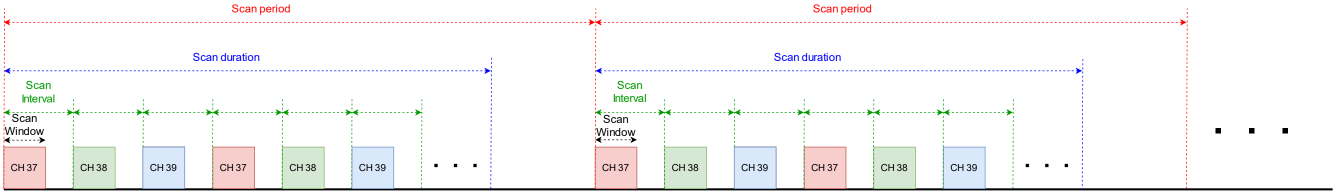


Figure 6-1 The relationship of scan interval, window, duration, period

The “fast_xxx” and “slow_xxx” fields of R_BLE_ABS_StartScan are set to change the scan frequency. As use case, the fast scan increases a detection probability of the target device and the slow scan decreases the scan frequency. Figure 6-2 shows the relationship between the fast scan and slow scan. Table 6.6 shows the event regarding the fast scan and slow scan.

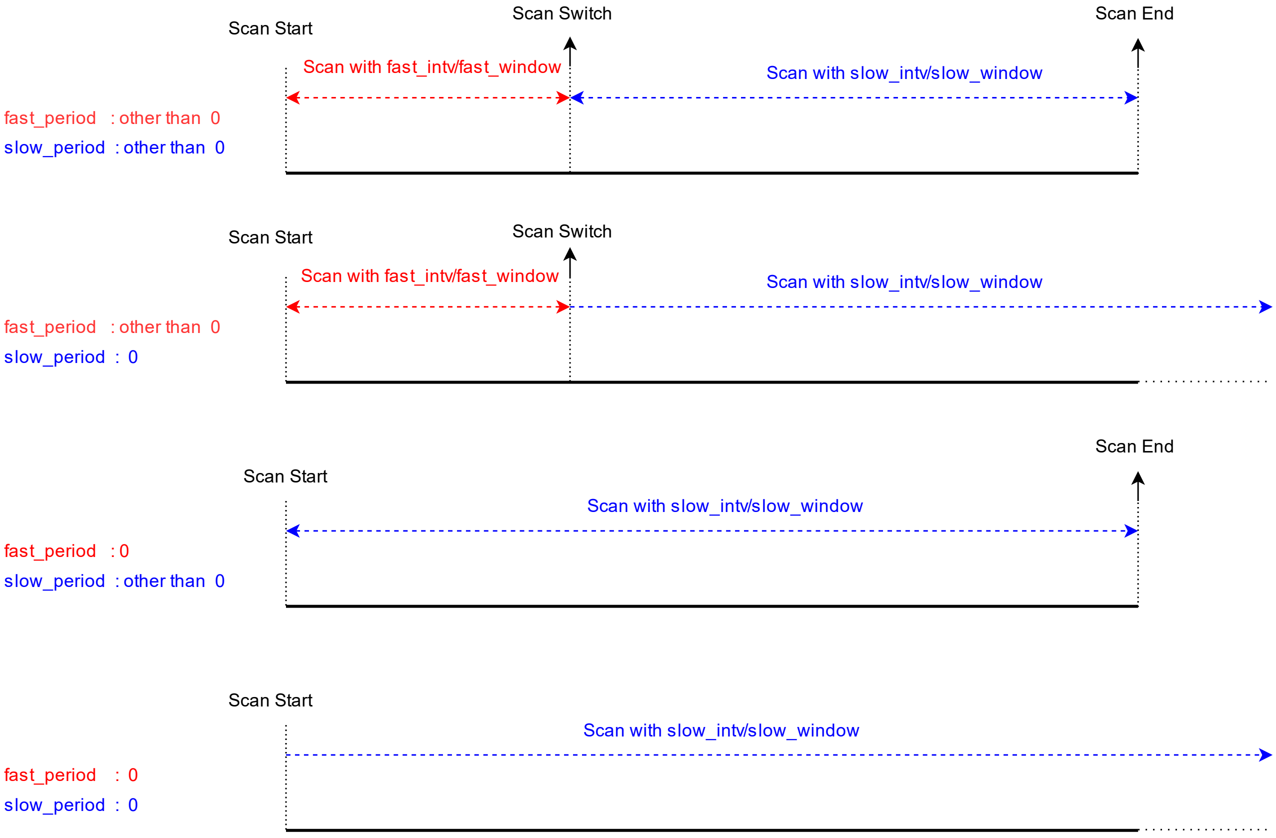

Scan Start
Scan Switch
Scan End
Scan with fast_intv/fast_window
Scan with slow_intv/slow_window
fast_period : other than 0
slow_period : other than 0
Scan Start
Scan Switch
Scan with fast_intv/fast_window
Scan with slow_intv/slow_window
fast_period : other than 0
slow_period : 0
Scan Start
Scan End
Scan with slow_intv/slow_window
fast_period : 0
slow_period : other than 0
Scan Start
Scan with slow_intv/slow_window
fast_period : 0
slow_period : 0

Figure 6-2 The relationship between the fast scan and slow scan

Table 6.6 The event regarding the fast scan and slow scan

Library Type	Scan Start	Scan Switch	Scan End
All features	BLE_GAP_EVENT_SCAN_ON	BLE_GAP_EVENT_SCAN_TO BLE_GAP_EVENT_SCAN_ON	BLE_GAP_EVENT_SCAN_TO
Balance	BLE_GAP_EVENT_SCAN_ON	BLE_GAP_EVENT_SCAN_OFF BLE_GAP_EVENT_SCAN_ON	BLE_GAP_EVENT_SCAN_OFF

6.2.1 Privacy

The privacy feature can set the address in a scan request to RPA. According to “9.4.1 Generate local device RPA”, prepare for the privacy feature in advance. If the local device uses RPA by R_BLE_GAP_StartScan, the following field in the st_ble_gap_scan_param_t structure(Table 6.1) needs to be set to the value shown in Table 6.7 to enable the privacy feature.

Table 6.7 The parameters used for the privacy feature (R_BLE_GAP_StartScan)

Field	Value	Description
o_addr_type	BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Public Address.
	BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Static Address.

If the local device uses RPA by R_BLE_ABS_StartScan, the following field in the st_ble_abs_scan_param_t structure(Table 6.4) needs to be set to the value shown in Table 6.8 to enable the privacy feature. For more details about this field, see API document.

Table 6.8 The parameters used for the privacy feature (R_BLE_ABS_StartScan)

Field	Value	Description
dev_filter	BLE_ABS_SCAN_ALL_RPA_PUBLIC (BLE_GAP_SCAN_ALLOW_ADV_ALL | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Public Address.
	BLE_ABS_SCAN_WLST_RPA_PUBLIC (BLE_GAP_SCAN_ALLOW_ADV_WLST | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	
	BLE_ABS_SCAN_EXC_DIR_RPA_PUBLIC (BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	
	BLE_ABS_SCAN_EXC_DIR_WLST_RPA_PUBLIC (BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	
	BLE_ABS_SCAN_ALL_RPA_STATIC (BLE_GAP_SCAN_ALLOW_ADV_ALL | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Static Address.
	BLE_ABS_SCAN_WLST_RPA_STATIC (BLE_GAP_SCAN_ALLOW_ADV_WLST | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	
	BLE_ABS_SCAN_EXC_DIR_RPA_STATIC (BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	
	BLE_ABS_SCAN_EXC_DIR_WLST_RPA_STATIC (BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	

If RPA generation and resolution are enabled by dev_filter a(), active scan will not be performed on remote devices that use Non-resolvable private address.

BP: When supporting privacy, it is recommended to limit the use of scannable advertisements and active scanning to avoid the risk of devices being tracked.

6.3 Received information by scan

After calling the Start Scan API, the Bluetooth LE Protocol Stack notifies receiving an advertising packet from another device using the BLE_GAP_EVENT_ADV_REPT_IND event. If the sender uses AUX_CHAIN_IND, Advertising Data will be notified separately. Furthermore, since the size of Advertising Data that can be notified to upper layer according to Bluetooth specification is 229 byte or less, 230 byte or more Advertising Data will be notified separately. Combine them on the receiver if necessary.

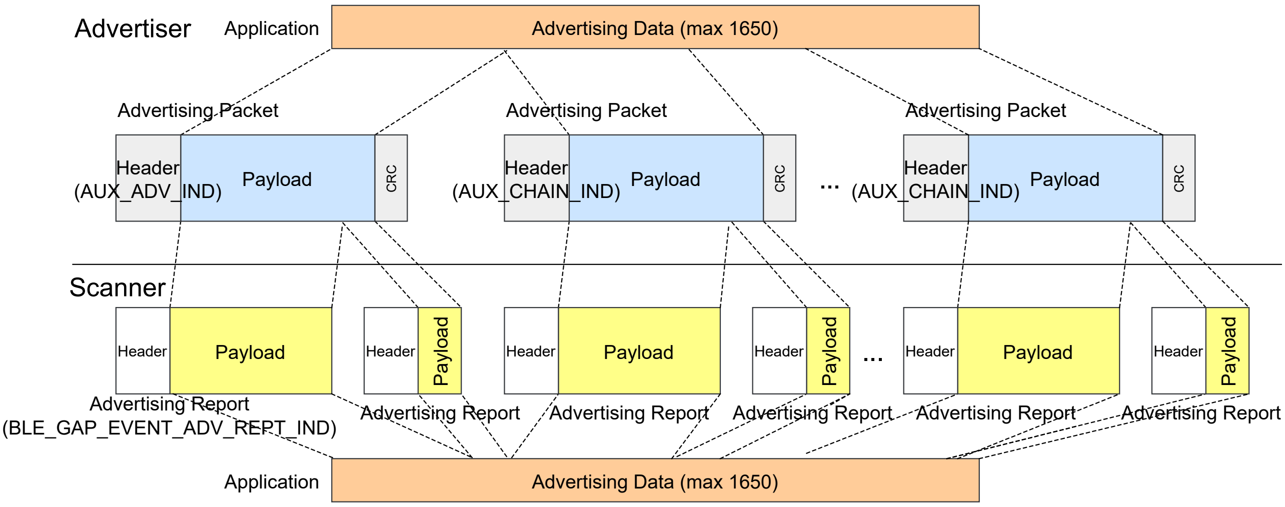


Figure 6-3 Dividing and combining Advertising Data

Received advertising packet is stored in a st_ble_gap_adv_rept_evt_t structure variable. Table 6.9 shows st_ble_gap_adv_rept_evt_t structure.

Table 6.9 st_ble_gap_adv_rept_evt_t structure

Type	Field	Description
uint8_t	adv_rpt_type	Advertising type.
union {		
st_ble_gap_adv_rept_t *	p_adv_rpt	If the Balance library is used, a received advertising packet is notified by this field.
st_ble_gap_ext_adv_rept_t *	p_ext_adv_rpt	If the All features library is used, a received advertising packet is notified by this field.
st_ble_gap_perd_adv_rept_t *	p_per_adv_rpt	A received periodic advertising packet is notified by this field. Only the All features library can use the field.
} param;		

Depending on the Bluetooth LE Protocol Stack, the field of advertising varies. Table 6.10 and Table 6.11 show the advertising field.

Table 6.10 st_ble_gap_adv_rept_t structure

Type	Field	Description
uint8_t	num	Number of received advertising. This field is always 1.
uint8_t	adv_type	Advertising packet type.
uint8_t	addr_type	Address type of received advertising packet.
uint8_t *	p_addr	Address of received advertising packet.
uint8_t	len	Size of received advertising data.
int8_t	rssi	Received advertising RSSI.
uint8_t *	p_data	Received advertising data.

Table 6.11 st_ble_gap_ext_adv_rept_t structure

Type	Field	Description
uint8_t	num	Number of received advertising. This field is always 1.
uint8_t	adv_type	Advertising packet type.
uint8_t	addr_type	Address type of received advertising packet.
uint8_t *	p_addr	Address of received advertising packet.
uint8_t	adv_phy	Primary PHY for Advertising.
uint8_t	sec_adv_phy	Secondary PHY for Advertising.
uint8_t	adv_sid	Advertising SID.
int8_t	tx_pwr	Tx power.
int8_t	rssi	Received advertising RSSI.
uint16_t	perd_adv_intv	Periodic advertising interval.
uint8_t	dir_addr_type	Address type included in Direct Advertising packet.
uint8_t *	p_dir_addr	Address included in Direct Advertising packet.
uint8_t	len	Size of received advertising data.
uint8_t *	p_data	Received advertising data.

For more information about the above structures, refer to the API document.

An example of displaying the RSSI included in a received advertising packet is shown below.

```
/* GAP callback function */
void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
        /** some code is omitted  **/
        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
            st_ble_gap_adv_rept_evt_t *adv_rept_evt_param =
                (st_ble_gap_adv_rept_evt_t *)data->p_param;

            switch (adv_rept_evt_param->adv_rpt_type)
            {
                /* receive legacy advertising PDU */
                case 0x00:
                {
                    st_ble_gap_adv_rept_t *adv_rept_param =
                        (st_ble_gap_adv_rept_t *)adv_rept_evt_param->param.p_adv_rpt;

                    printf("RSSI : %d \n", adv_rept_param->rssi);
                } break;

                /* receive extended advertising PDU */
                case 0x01:
                {
                    st_ble_gap_ext_adv_rept_t *ext_adv_rept_param =
                        (st_ble_gap_ext_adv_rept_t *)ext_adv_rept_param->
                                                         param.p_ext_adv_rpt;

                    printf("RSSI : %d \n", ext_adv_rept_param->rssi);
                } break;
                /** some code is omitted  **/
```

Code 6-1 Sample of displaying the RSSI included in a received advertising packet

6.4 Scan filtering

It is possible to filter received advertising packets by scan. The filtering can be used if you want to notify the essential advertising packets to your application.

The filtering by the APIs is as follows.

- Using the White List
- Duplicate advertising filtering
- Discoverable mode filtering
- Advertising Data filtering

6.4.1 Using the White List (Receiving from known devices)

If the BD_ADDR of the device which of advertising packets are to received is known, filter advertising packets by this method. Before starting scan, perform the 1, 2 steps.

1. Register the BD_ADDR of the remote device which sends advertising packets by the White List. Call R_BLE_GAP_ConfWhiteList to register a known device.

Note: The White List cannot be added/deleted when the White List filter enabled operation (advertising, scanning, connection request) is executed.

2. Set the below Scan Filter Policy parameters of the Start Scan API parameter to BLE_GAP_SCAN_ALLOW_ADV_WLST(0x01).
    - The filter_policy field of the st_ble_gap_scan_param_t structure (R_BLE_GAP_StartScan)
    - The dev_filter field of the st_ble_abs_scan_param_t structure (R_BLE_ABS_StartScan)

6.4.2 Duplicate advertising filtering

If you do not want to receive duplicate advertising packets from same device, set the duplicate filtering. Set the below Scan Filter Policy parameters of the Start Scan API parameter to BLE_GAP_SCAN_ALLOW_ADV_WLST(0x01).

- The filter_policy field of the st_ble_gap_scan_param_t structure (R_BLE_GAP_StartScan)
- The dev_filter field of the st_ble_abs_scan_param_t structure (R_BLE_ABS_StartScan)

The duplicate filtering can filter same advertising packets from 8 devices at most. If there are more than 9 advertising devices, same advertising packets of the 9th and subsequent devices cannot be filtered and the application receives those.

6.4.3 Discoverable mode filtering

Advertising packets are filtered with Discoverable Mode because of the Flag AD_TYPE included in advertising data. The Abstraction API does not support this feature. Table 6.12 shows the value to be set to the proc_type field in the st_ble_gap_scan_on_t structure of R_BLE_GAP_StartScan.

Table 6.12 The value to be set for filtering with Discoverable Mode

Value	Description
BLE_GAP_SC_PROC_OBS(0x00)	Receive advertising packets without regard to Discoverable Mode.
BLE_GAP_SC_PROC_LIM(0x01)	Receive advertising packets in LE Limited Discoverable Mode.
BLE_GAP_SC_PROC_GEN(0x02)	Receive advertising packets in LE General Discoverable Mode.

6.4.4 Advertising Data filtering

The Abstraction API can filter by the data included in advertising data. Specify the data for filtering to the following parameters in the st_ble_abs_scan_param_t structure.

p_filter_data: The filtered data.

filter_data_length: The filtered data size.

filter_ad_type: The AD_TYPE of the filtered data.

```
/* Scan filter data */
static uint8_t gs_filter_data[] =
{
    /* Complete Local Name */
    9,          /**< Data Size */
    0x09,        /**< Data Type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */
};

/* Scan parameters */
static st_ble_abs_scan_param_t gs_scan_param =
{
    .p_phy_param_1M         = &gs_scan_phy_param,
    .p_filter_data          = gs_filter_data,
    .slow_period            = 0,
    .filter_data_length    = ARRAY_SIZE(gs_filter_data),
    .dev_filter             = BLE_GAP_SCAN_ALLOW_ADV_ALL,
    .filter_dups            = BLE_GAP_SCAN_FILT_DUPLIC_ENABLE,
};
```

Code 6-2 Sample of advertising data filtering

6.5 Periodic Advertising Synchronization

A scanner can establish a Periodic Advertising Synchronization (Sync) with an advertiser due to the AUX_ADV_IND information. Figure 6-4 shows the procedure that a scanner establishes a Periodic Advertising Sync in application. The following sections describes the details of Periodic Advertising Sync procedure.

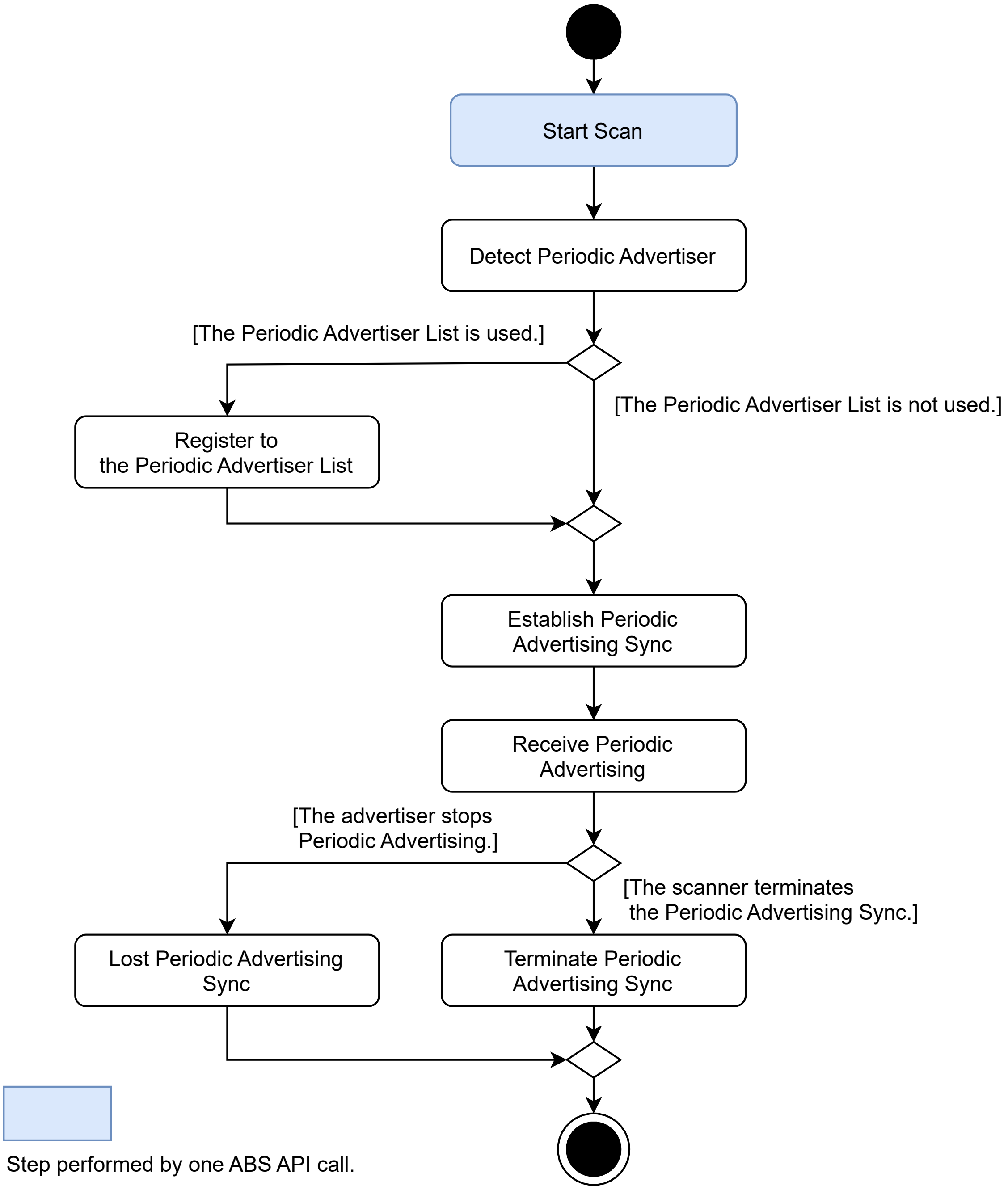

Start Scan
Detect Periodic Advertiser
[The Periodic Advertiser List is used.]
[The Periodic Advertiser List is not used.]
Register to the Periodic Advertiser List
Establish Periodic Advertising Sync
Receive Periodic Advertising
[The advertiser stops Periodic Advertising.]
[The scanner terminates the Periodic Advertising Sync.]
Lost Periodic Advertising Sync
Terminate Periodic Advertising Sync
Step performed by one ABS API call.

Figure 6-4 Periodic Advertising Sync procedure

6.5.1 Start Scan

Start scan according to “6.1 Start or stop scan”.

6.5.2 Detect Periodic Advertiser

The scanner can establish a Periodic Advertising Sync with the advertiser if the perd_adv_intv (shown in Table 6.11) included in a received advertising packet is not 0. Specify the advertiser with the addr_type, p_addr, adv_sid field in Table 6.11 according to “6.5.3 Register to the Periodic Advertiser List“ or “6.5.4 Establish Periodic Advertising Sync”.

6.5.3 Register to the Periodic Advertiser List

Select using the Periodic Advertiser List or the remote device address to point to the advertiser for establishing a Periodic Advertising Sync. If using the Periodic Advertiser List, call R_BLE_GAP_ConfPerdAdvList to register a known device.

6.5.4 Establish Periodic Advertising Sync

Call R_BLE_GAP_CreateSync to establish a Periodic Advertising Sync. When a Periodic Advertising Sync has been established, the BLE_GAP_EVENT_SYNC_EST event is notified. To cancel establishing a Periodic Advertising Sync after calling R_BLE_GAP_CreateSync, call R_BLE_GAP_CancelCreateSync. When the cancellation has been completed, the BLE_GAP_EVENT_SYNC_EST event that the result is BLE_ERR_NOT_YET_READY(0x0012) is notified.

The maximum number of Periodic Advertising Syncs is the value of the BLE_CFG_RF_SYNC_SET_MAX option. An example of from starting scan to establishing a Periodic Advertising Sync is shown below.

```
/** some code is omitted  **/

static st_ble_dev_addr_t gs_sync_advr;
static uint8_t gs_adv_sid;

static st_ble_abs_scan_phy_param_t gs_phy_param_1M =
{
    .fast_intv                  = 0x0200,
    .slow_intv                  = 0x0800,
    .fast_window                = 0x0100,
    .slow_window                = 0x0100,
    .scan_type                  = BLE_GAP_SCAN_PASSIVE,
};

static st_ble_abs_scan_param_t gs_scan_param =
{
    .p_phy_param_1M             = &gs_phy_param_1M,
    .p_phy_param_coded          = NULL,
    .p_filter_data              = NULL,
    .fast_period                = 0x0100,
    .slow_period                = 0x0000,
    .filter_data_length         = 0,
    .dev_filter                 = BLE_GAP_SCAN_ALLOW_ADV_ALL,
    .filter_dups                = BLE_GAP_SCAN_FILT_DUPLIC_DISABLE,
};

static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    /** some code is omitted  **/
    switch(type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_ABS_StartScan(&gs_scan_param);
        } break;

        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
            st_ble_gap_adv_rept_evt_t * p_adv_rept_evt_param =
                (st_ble_gap_adv_rept_evt_t *)p_data->p_param;

            switch (p_adv_rept_evt_param->adv_rpt_type)
            {
                case 0x01:
                {
                    st_ble_gap_ext_adv_rept_t * p_ext_adv_rept_param =
                        (st_ble_gap_ext_adv_rept_t *)p_adv_rept_evt_param->param.p_ext_adv_rpt;

                    if(0x0000 != p_ext_adv_rept_param->perd_adv_intv)
                    {
                        /* found */
                        memcpy(gs_sync_advr.addr, p_ext_adv_rept_param->p_addr,
                               BLE_BD_ADDR_LEN);
                        gs_sync_advr.type = p_ext_adv_rept_param->addr_type;
                        gs_adv_sid = p_ext_adv_rept_param->adv_sid;
                        R_BLE_GAP_ConfPerdAdvList(BLE_GAP_LIST_ADD_DEV,
                                                     &gs_sync_advr,
                                                     &gs_adv_sid,
                                                     1);
                    }

                } break;
                /** some code is omitted  **/
            }
        } break;

        case BLE_GAP_EVENT_PERD_LIST_CONF_COMP:
        {
            R_BLE_GAP_CreateSync(NULL, 0, 100, 100);
        } break;

        case BLE_GAP_EVENT_SYNC_EST:
        {
            if(BLE_SUCCESS == result)
            {
                R_BLE_CLI_Printf("sync established.\n");
            }
        } break;

        /** some code is omitted  **/
    }
}

/** some code is omitted  **/
```

Code 6-3 Sample of establishing a Periodic Advertising Sync

6.5.5 Receive Periodic Advertising

After the Periodic Advertising Sync has been established with the advertiser, receiving a Periodic Advertising packet is notified by the BLE_GAP_EVENT_ADV_REPT_IND event. A received Periodic Advertising packet is stored in a st_ble_gap_adv_rept_evt_t type (Table 6.9) variable. Table 6.13 shows the st_ble_gap_perd_adv_rept_t structure in case of Periodic Advertising.

Table 6.13 st_ble_gap_perd_adv_rept_t structure

Type	Field	Description
uint16_t	sync_hdl	Sync handle identifying an Established Periodic Advertising Sync.
int8_t	tx_pwr	Tx power
int8_t	rssi	RSSI
uint8_t	rfu	Reserved for future use
uint8_t	data_status	Status of Periodic Advertising Data
uint8_t	len	Periodic Advertising Data Size
uint8_t *	p_data	Periodic Advertising Data

6.5.6 Lost Periodic Advertising Sync

If the advertiser stops Periodic Advertising, loss of the Periodic Advertising Sync is notified through the BLE_GAP_EVENT_SYNC_LOST event is notified.

6.5.7 Terminate Periodic Advertising Sync

By calling BLE_GAP_TerminateSync, the scanner terminates the Periodic Advertising Sync. When the Periodic Advertising Sync has been terminated, user application is notified through the BLE_GAP_EVENT_SYNC_TERM event.

7. Connection

7.1 Requesting Connection

Central device sends a connection request to Peripheral device running Connectable Advertising.by the below APIs. Peripheral device that receives a connection request terminates Connectable Advertising and responds to the connection request.

Connection Request API:

- R_BLE_GAP_CreateConn
- R_BLE_ABS_CreateConn

For more information about the above APIs parameters, refer to the following items in the API document.

R_BLE_GAP_CreateConn:
st_ble_gap_create_conn_param_t

R_BLE_ABS_CreateConn:
st_ble_abs_conn_param_t

Setting the following is required to connect with the remote device that the primary channels (CH:37/38/39) of Advertising use 1M PHY.
The p_conn_param_1M field in st_ble_gap_create_conn_param_t structure used by R_BLE_GAP_CreateConn.
The p_conn_1M field in st_ble_abs_conn_param_t structure used by the R_BLE_ABS_CreateConn.

Setting the following is required to connect with the remote device that primary channels of Advertising use Coded PHY.
The p_conn_param_coded field in st_ble_gap_create_conn_param_t structure used by R_BLE_GAP_CreateConn.
The p_conn_coded field in st_ble_abs_conn_param_t structure used by the R_BLE_ABS_CreateConn.

The PHY after establishing a connection will be the PHY specified in the secondary channel (other than CH:37/38/39) of Advertising.

When the connection is established, the connection handle is notified in the BLE_GAP_EVENT_CONN_IND event. Since the connection handle is assigned a number that does not overlap with other connections in the range from 0 to BLE_CFG_RF_CONN_MAX-1 in the lower 3 bits, it can be masked as shown in the sample below and used as the index of the management array of the connection handle.

```
#define BLE_APP_CONN_HDL_MASK               (0x0007)
uint16_t g_conn_hdl[BLE_CFG_RF_CONN_MAX];
void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
/** some code is omitted  **/
        case BLE_GAP_EVENT_CONN_IND:
        {
            if (BLE_SUCCESS == result)
            {
                /* Store connection handle */
                st_ble_gap_conn_evt_t *p_gap_conn_evt_param = (st_ble_gap_conn_evt_t *)p_data->p_param;
                uint16_t index = p_gap_conn_evt_param->conn_hdl & BLE_APP_CONN_HDL_MASK;
                g_conn_hdl[index] = p_gap_conn_evt_param->conn_hdl;
```

7.1.1 Using the White List (Connection to a known device)

It is possible to send a connection request after registering a known device in the White List. If reconnecting to the known device, use the White List. The procedure is as follows.

1. Register the BD_ADDR of the remote device which is reconnected by the White List.
   Call R_BLE_GAP_ConfWhiteList to register a known device.

Note: The White List cannot be added/deleted when the White List filter enabled operation (advertising, scanning, connection request) is executed.

2. Set the following connection parameters
   - The init_filter_policy field in st_ble_gap_create_conn_param_t structure used by R_BLE_GAP_CreateConn.
   - The filter field in st_ble_abs_conn_param_t structure used by the R_BLE_ABS_CreateConn.

   Set the above parameters to BLE_GAP_INIT_FILT_USE_WLST(0x01) to send a connection request to a known device in the White List.

An example of connecting a remote device registered in the White List is shown below.

```
/* remote device address */
dev.addr = {"Remote device BD_ADDR" };
dev.type = BLE_GAP_ADDR_PUBLIC;

/* register remote device to white list */
R_BLE_GAP_ConfWhiteList(BLE_GAP_LIST_ADD_DEV, &dev, 1);

/** some code is omitted  **/

/* reconnect */
st_ble_gap_conn_param_t conn_1M = {
    .conn_intv_min  = 0x0100,
    .conn_intv_max  = 0x0100,
    .conn_latency   = 0x0000,
    .sup_to         = 0x03BB,
    .min_ce_length  = 0xFFFF,
    .max_ce_length  = 0xFFFF,
};

st_ble_gap_create_conn_param_t conn_param;
conn_param.init_filter_policy = BLE_GAP_INIT_FILT_USE_WLST;
conn_param.own_addr_type = BLE_GAP_ADDR_PUBLIC;

/*  set connection parameters for 1M */
st_ble_gap_conn_phy_param_t conn_phy_1M = {
    .scan_intv   = 0x0300,
    .scan_window = 0x0300,
    p_conn_param = &conn_1M,
};

conn_param.p_conn_param_1M = &conn_phy_1M;

R_BLE_GAP_CreateConn(&conn_param);

/** some code is omitted  **/
```

Code 7-1 Connection Request using the White List

7.1.2 Privacy

The privacy feature can set the address in a connection request to RPA. According to “9.4.1 Generate local device RPA”, prepare for the privacy feature in advance. If the local device uses RPA by R_BLE_GAP_CreateConn, the following fields in the st_ble_gap_create_conn_param_t structure need to be set to the values shown in Table 7.1 to enable the privacy feature.

Table 7.1 The parameters used for the privacy feature (R_BLE_GAP_CreateConn)

Field	Value	Description
own_addr_type	BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Public Address.
	BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Static Address.
remote_bd_addr_type	Specify the remote device address registered by R_BLE_GAP_ConfRslvList.	—

If the local device uses RPA by R_BLE_ABS_CreateConn, the following fields in the st_ble_abs_conn_param_t structure need to be set to the values shown in Table 7.2 to enable the privacy feature.

Table 7.2 The parameters used for the privacy feature (R_BLE_ABS_CreateConn)

Field	Value	Description
filter	BLE_ABS_CONN_USE_ADDR_RPA_PUBLIC (BLE_GAP_INIT_FILT_USE_ADDR | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Public Address.
	BLE_ABS_CONN_USE_WLST_RPA_PUBLIC (BLE_GAP_INIT_FILT_USE_WLST | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))	
	BLE_ABS_CONN_USE_ADDR_RPA_STATIC (BLE_GAP_INIT_FILT_USE_ADDR | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is Public Address.
	BLE_ABS_CONN_USE_WLST_RPA_STATIC (BLE_GAP_INIT_FILT_USE_WLST | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))	
remote_bd_addr_type	Specify the remote device address registered by R_BLE_GAP_ConfRslvList.	—
remote_bd_addr		

7.2 Cancelling Connection Request

A connection request cannot be sent until the connection is established by a previous connection request or until the connection request is cancelled. After sending a connection request, if you want to send another connection request, cancel the previous connection request by BLE_GAP_CancelCreateConn. After cancelling the request, the BLE_GAP_EVENT_CONN_IND event is notified with the result BLE_ERR_INVALID_HDL(0x000E).

7.3 Multiple Connection

This chapter describes how to connect to multiple devices at the same time and the precautions to be taken when doing so. With the Bluetooth LE Protocol Stack, up to 7 devices can be connected simultaneously. The connection procedure is the same as for one-to-one communication. The application specifies the connection device using the connection handle that is notified when connecting. The connection handle is allocated for the connection, so even if it is the same device, it will change when reconnecting.

The attribute handle for accessing the characteristic in the GATT database is device specific. When connecting to multiple devices as a GATT client, it is necessary to hold an attribute handle for each GATT server. By using Profile Common of app_lib, you can hold the attribute handle for each device up to 10 in the order of connection.

When connecting from multiple devices as a GATT server, there are some such as Client Configuration Characteristic Descriptor whose specifications hold values for each device. If accessed from multiple clients, set the GATT database properties to hold the respective values.

An implementation example of application code that connects multiple devices for each expected use case is explained.

7.3.1 Connecting to multiple peripheral devices

It communicates with multiple peripheral devices, with itself as the central. For example, assume an application that aggregates multiple sensor data. Here, the central device is the GATT client.

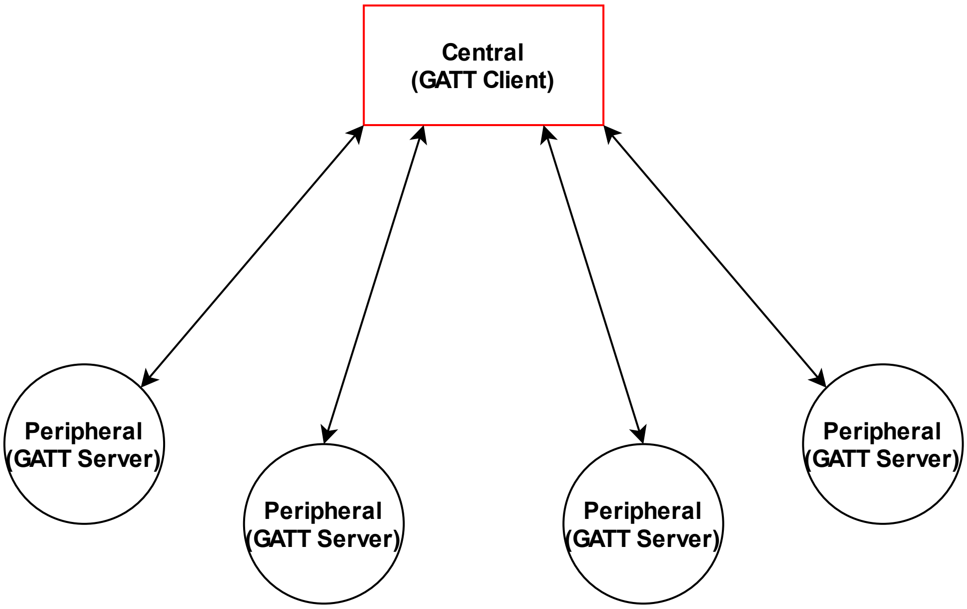


Figure 7-1 Connection with multiple peripheral devices

To ensure a reliable connection one by one, the central device connects in sequence with the completion of service discovery as a break. Below shows a sequence chart and an implementation example when connecting using app_lib of the Bluetooth LE Protocol Stack. Repeat this procedure to connect multiple peripheral devices.

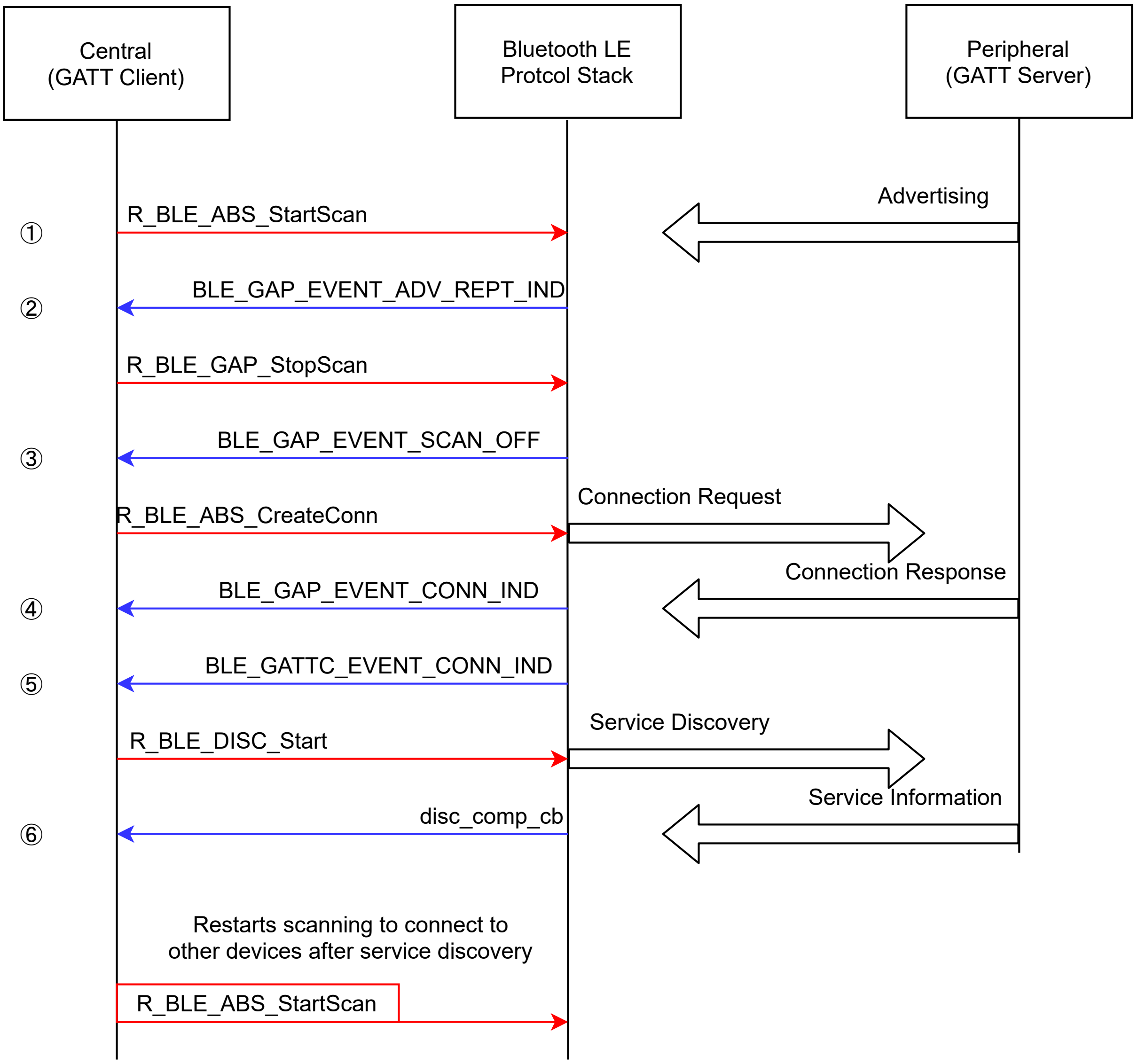


Figure 7-2 Sequence chart when connecting to a peripheral device (The circled numbers in the chart correspond to the numbers in Code 7-3 below.)

```c
/* Scan phy parameters */
static st_ble_abs_scan_phy_param_t gs_scan_phy_param =
{
    /* TODO: Modify scan phy parameter. */
    .fast_intv   = 0x200,
    .fast_window = 0x100,
    .slow_intv   = 0x200,
    .slow_window = 0x100,
    .scan_type   = BLE_GAP_SCAN_PASSIVE,
};

/* Scan filter data */
static uint8_t gs_filter_data[] =
{
    /* TODO: Modify filter of advertise data. Value of Data Flag is defined in
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */
};

/* Scan parameters */
static st_ble_abs_scan_param_t gs_scan_param =
{
    /* TODO: Modify scan parameter. */
    .p_phy_param_1M     = &gs_scan_phy_param,
    .p_filter_data      = gs_filter_data,
    .slow_period        = 0,
    .filter_data_length = ARRAY_SIZE(gs_filter_data),
    .dev_filter         = BLE_GAP_SCAN_ALLOW_ADV_ALL,
    .filter_dups        = BLE_GAP_SCAN_FILT_DUPLIC_ENABLE,
};

/* Connection phy parameters */
static st_ble_abs_conn_phy_param_t gs_conn_phy_param =
{
    /* TODO: Modify connection phy parameter. */
    .conn_intv    = 0x0130,
    .conn_latency = 0x0000,
    .sup_to       = 0x03BB,
};

/*  Connection device address */
static st_ble_dev_addr_t gs_conn_bd_addr;

/* Connection parameters */
static st_ble_abs_conn_param_t gs_conn_param =
{
    .p_conn_1M = &gs_conn_phy_param,
    .p_addr    = &gs_conn_bd_addr,     /**< Set BD address of connecting device. */
    .filter    = BLE_GAP_INIT_FILT_USE_ADDR,
    .conn_to   = 5,
};
```

Code 7-2 Setting initial values for scan parameters and connection parameters

```
/* Connection handle */
uint16_t g_conn_hdl[BLE_CFG_RF_CONN_MAX];
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON: /* (1) */
        {
            R_BLE_ABS_StartScan(&gs_scan_param);
        } break;

        case BLE_GAP_EVENT_CONN_IND: /* (4) */
        {

            if (BLE_SUCCESS == result)
            {
                st_ble_gap_conn_evt_t *p_gap_conn_evt_param =
                    (st_ble_gap_conn_evt_t *)p_data->p_param;

                for(uint8_t i=0;i<BLE_CFG_RF_CONN_MAX;i++)
                {
                if(g_conn_hdl[i] == BLE_GAP_INVALID_CONN_HDL)
                {
                        g_conn_hdl[i] = p_gap_conn_evt_param->conn_hdl;
                }
                }

            }
        } break;

        case BLE_GAP_EVENT_DISCONN_IND:
        {
        st_ble_gap_disconn_evt_t *p_gap_disconn_evt_param =
                         (st_ble_gap_disconn_evt_t*)p_data->p_param;


        for(uint8_t i=0;i<BLE_CFG_RF_CONN_MAX;i++)
        {
                if(g_conn_hdl[i] == p_gap_disconn_evt_param->conn_hdl)
                {
                        g_conn_hdl[i] = BLE_GAP_INVALID_CONN_HDL;
                }
        }
        } break;

        case BLE_GAP_EVENT_ADV_REPT_IND: /* (2) */
        {
            st_ble_gap_adv_rept_evt_t *p_adv_rept_param = (st_ble_gap_adv_rept_evt_t *)p_data->p_param;
            st_ble_gap_ext_adv_rept_t *p_ext_adv_rept_param = (st_ble_gap_ext_adv_rept_t
*)p_adv_rept_param->param.p_ext_adv_rpt;
            gs_conn_param.p_addr->type = p_ext_adv_rept_param->addr_type;
            memcpy(gs_conn_param.p_addr->addr, p_ext_adv_rept_param->p_addr, BLE_BD_ADDR_LEN)

            R_BLE_GAP_StopScan();
        } break;

        case BLE_GAP_EVENT_SCAN_OFF: /* (3) */
        {
            R_BLE_ABS_CreateConn(&gs_conn_param);
        }
        default:
        {
            /* Do nothing. */
        } break;
    }
}
```

Code 7-3 Implementation example of GAP callback function when connecting multiple units

```
/* XXX Service UUID */
static uint8_t XXXC_UUID[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 
0x00, 0x00, 0x00, 0x00 };

/* Service discovery parameters */
static st_ble_disc_entry_t gs_disc_entries[] = {

    {
        .p_uuid      = XXXC_UUID,
        .uuid_type   = BLE_GATT_128_BIT_UUID_FORMAT,
             .serv_cb     = R_BLE_XXXC_ServDiscCb,
    },
};

static void disc_comp_cb(uint16_t conn_hdl)
{
    /* TODO: Add function after discovery completed */
    BLE_ABS_StartScan(&gs_scan_param); /* (6) */
    return;
}

static void gattc_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
    R_BLE_SERVC_GattcCb(type, result, p_data);

    switch(type)
    {
        /* TODO: Set callback events of GATTC. Check BLE API reference for events. */

        case BLE_GATTC_EVENT_CONN_IND: /* (5) */
        {
            R_BLE_DISC_Start(p_data->conn_hdl, gs_disc_entries, ARRAY_SIZE(gs_disc_entries), disc_comp_cb);
        } break;

        default:
        {
            /* Do nothing. */
        } break;
    }
}
```

Code 7-4 Implementation example of service discovery using Profile Common Library

If you register R_BLE_XXXC_ServDiscCb of Service API (r_ble_xxxc.c) generated by QE for BLE in Discovery in Profile Common of app_lib (bold frame in **Code 7-4**), attribute handle of each device is retained in Service API through Profile Common. By using the Service API, the application can access the GATT database of each device using the connection handle without managing the attribute handle of each device.

7.3.2 Connection to multiple central devices

It uses itself as a peripheral to communicate with multiple central devices. For example, it is assumed that home appliances are controlled from multiple smartphones. Here, the peripheral device is the GATT server.

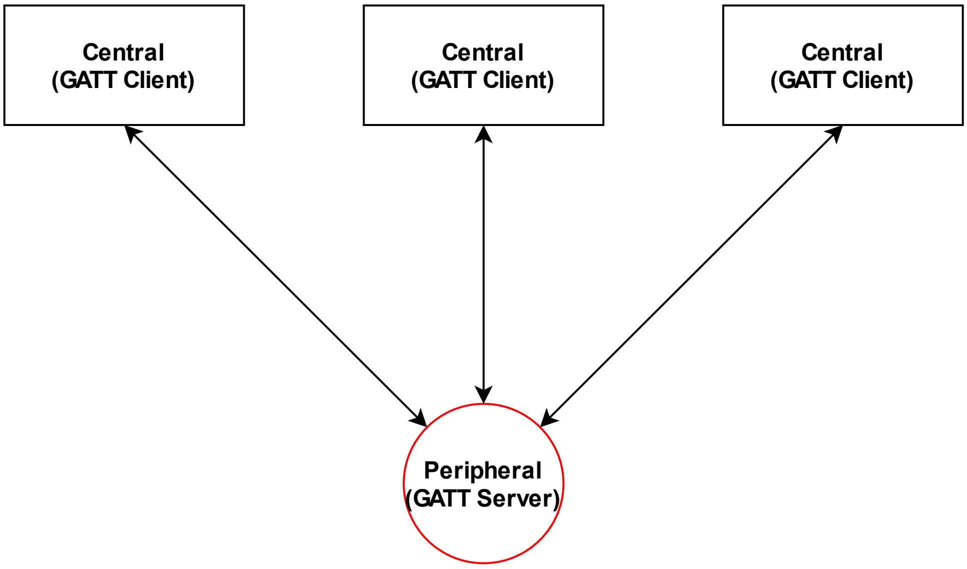


Figure 7-3 Connection with multiple central devices

Advertising stops when connected from Central. After connecting, it resumes advertising and accepts the connection from another device.

Below show a sequence chart and an implementation example when connecting using app_lib of the Bluetooth LE Protocol Stack. Repeat this procedure to accept connections from multiple central devices.

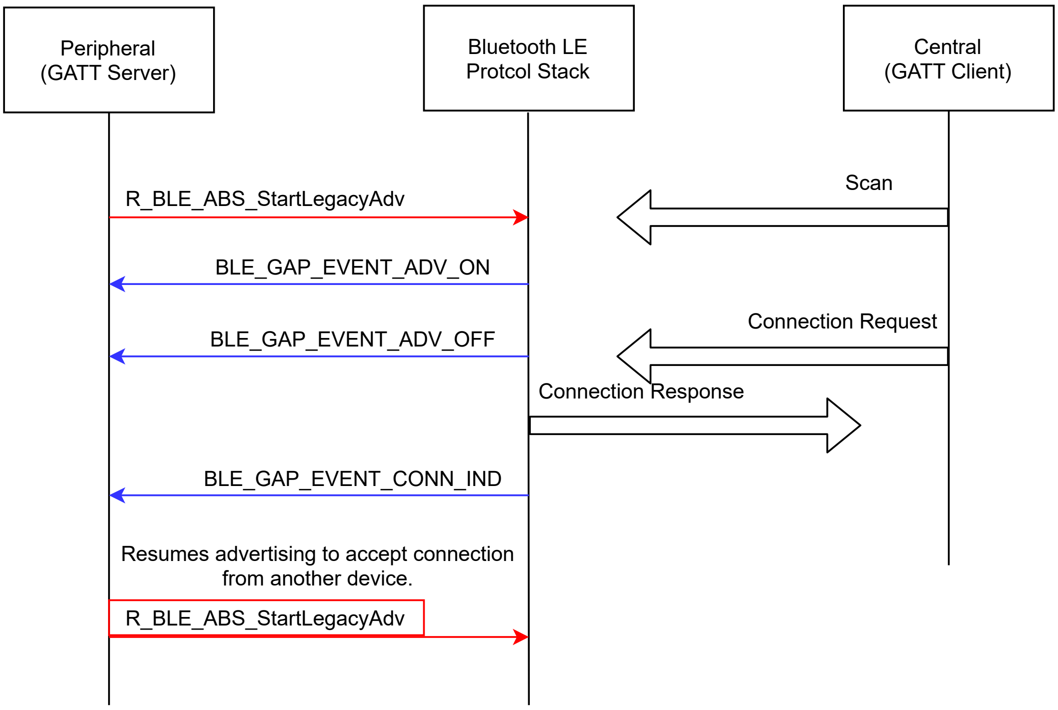


Figure 7-4 Sequence chart when connecting to a central device

```
/* Advertising data */
static uint8_t gs_adv_data[] =
{
    /* TODO: Modify advertise data. Value of Data Flag is defined in
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Flag (mandatory) */
    2,          /**< Data Size */
    0x01,       /**< Data Type: Flag */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE | BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED),    /**< Data Value */

    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */
};

/* Scan response Data */
static uint8_t gs_sres_data[] =
{
    /* TODO: Modify scan response data. Value of Data Flag is defined in
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Type: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'D', 'E', 'V', /**< Data Value */
};

/* Advertising parameters */
static st_ble_abs_legacy_adv_param_t gs_adv_param =
{
    /* TODO: Modify advertise parameters. */
    .slow_adv_intv   = 0x300,
    .slow_period     = 0,
    .p_adv_data      = gs_adv_data,
    .adv_data_length = ARRAY_SIZE(gs_adv_data),
    .p_sres_data     = gs_sres_data,
    .sres_data_length= ARRAY_SIZE(gs_sres_data),
    .adv_ch_map      = BLE_GAP_ADV_CH_ALL,
    .filter          = BLE_ABS_ADV_ALLOW_CONN_ANY,
    .o_addr_type     = BLE_GAP_ADDR_PUBLIC,
};
```

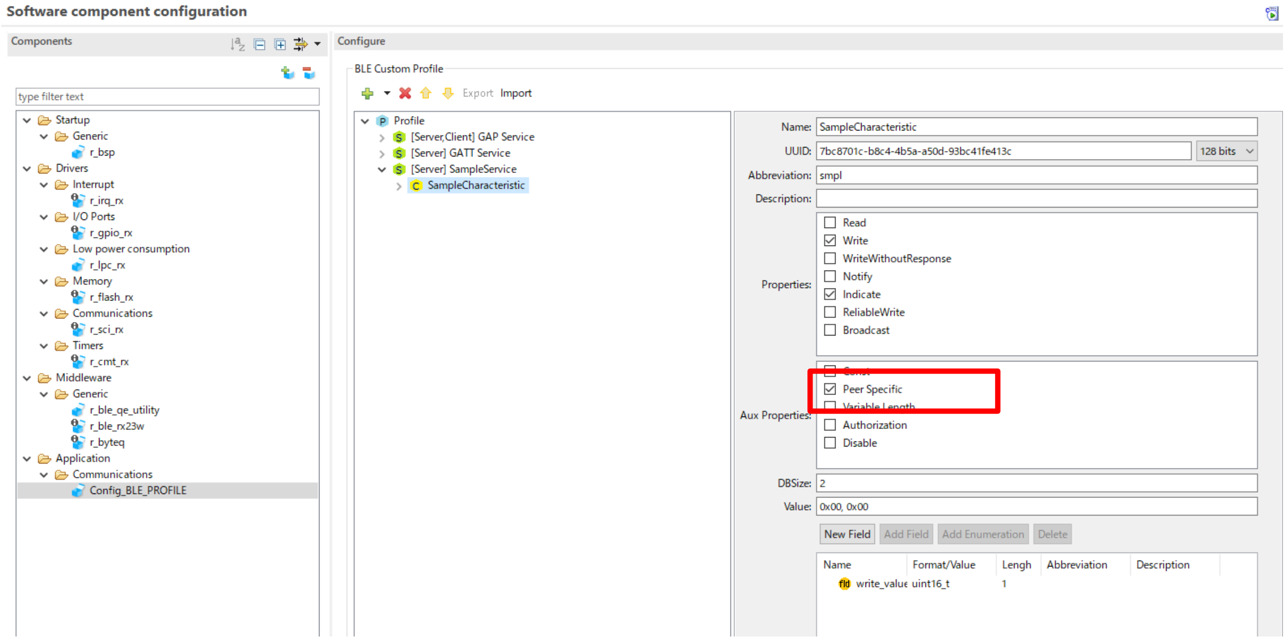
Code 7-5 Advertise packet and parameter settings

```
uint16_t g_conn_hdl[BLE_CFG_RF_CONN_MAX];

static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
        } break;

        case BLE_GAP_EVENT_CONN_IND:
        {

            if (BLE_SUCCESS == result)
            {
                st_ble_gap_conn_evt_t *p_gap_conn_evt_param =
                    (st_ble_gap_conn_evt_t *)p_data->p_param;
                R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
                for(uint8_t i=0;i<BLE_CFG_RF_CONN_MAX;i++)
                {
                if(g_conn_hdl[i] == BLE_GAP_INVALID_CONN_HDL)
                {
                        g_conn_hdl[i] = p_gap_conn_evt_param->conn_hdl;
                }

                }
            }
        } break;

        case BLE_GAP_EVENT_DISCONN_IND:
        {
        st_ble_gap_disconn_evt_t *p_gap_disconn_evt_param = (st_ble_gap_disconn_evt_t*)p_data->p_param;


        for(uint8_t i=0;i<BLE_CFG_RF_CONN_MAX;i++)
        {
                if(g_conn_hdl[i] == p_gap_disconn_evt_param->conn_hdl)
                {
                        g_conn_hdl[i] = BLE_GAP_INVALID_CONN_HDL;
                }
        }
        } break;

        default:
        {
            /* Do nothing. */
        } break;
    }
}
```

Code 7-6 Example implementation of GAP callback function when accepting connections from multiple centrals

In Bluetooth Low Energy, the Central (central device) controls the communication timing. Therefore, when multiple central devices are connected, the communication timing may accidentally collide and disconnect early. To prevent this, it is recommended to update the connection parameters so that there is a margin in peripheral latency and supervision timeout time. For updating the connection parameters, refer to "8.3 Updating connection parameter".

The GATT server may expose a common characteristic value to all connected GATT clients, or may expose a different value for each client. For example, when exposing different values for each client such as Client Configuration Characteristic Descriptor, check “Peer Specific” of Aux Properties on the characteristic screen of QE for BLE. As a result, the table of values and options held in the GATT database of the Bluetooth LE Protocol Stack are changed, and different values are held for up to 7 clients. A database value is returned for each client accessed.



Name: SampleCharacteristic

UUID: 7bc8701c-b8c4-4b5a-a50d-93bc41fe413c 128 bits

Abbreviation: smpl

Description:

Properties:
- ☐ Read
- ☑ Write
- ☐ WriteWithoutResponse
- ☐ Notify
- ☑ Indicate
- ☐ ReliableWrite
- ☐ Broadcast

Aux Properties:
- ☐ Const
- ☑ Peer Specific
- ☐ Variable Length
- ☐ Authorization
- ☐ Disable

DBSize: 2

Value: 0x00, 0x00

Name	Format/Value	Lengh	Abbreviation	Description
write_value	uint16_t	1		

Figure 7-5 Setting to retain the value of characteristic for each device

7.3.3 Multi role connection

In Bluetooth Low Energy communication, different GAP roles can be implemented for multiple devices that connect at the same time. It communicates centrally to one device and as a peripheral to another device. Here, the local device is the GATT server for the central device and the GATT client for the peripheral device.

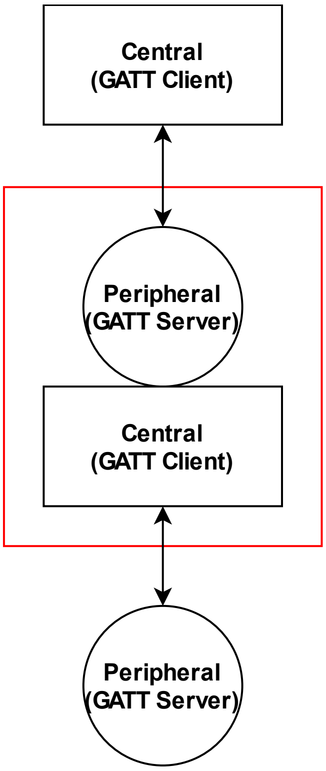


Figure 7-6 Multi roll connection example

Multi roll connections both advertise and scan to connect to both central and peripheral devices. Applications that make multi roll connections retain the connection handle and GAP role. GAP role of Local Device for the connection is posted in the BLE_GAP_EVENT_CONN_IND event. Below shows an implementation example of the GAP callback function when connecting as a central and peripheral. GAP callback function is implemented for each role. For scan and advertising settings, refer to **Code 7-4**(Scan) and **Code 7-5**(Advertise) above.

```
/* Connection handle */
uint16_t g_central_conn_hdl;

static void ble_central_gapcb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
        switch (type)
        {
                case BLE_GAP_EVENT_STACK_ON:
                {
                        R_BLE_ABS_StartScan(&gs_scan_param);
                } break;

                case BLE_GAP_EVENT_CONN_IND:
                {
                        if (BLE_SUCCESS == result)
                        {
                                st_ble_gap_conn_evt_t *p_gap_conn_evt_param =
                                         (st_ble_gap_conn_evt_t *)p_data->p_param;
                                if(0x00 == p_gap_conn_evt_param->role)
                                {
                                        g_central_conn_hdl = p_gap_conn_evt_param->conn_hdl;
                                }
                        }
                } break;

                case BLE_GAP_EVENT_DISCONN_IND:
                {
                        st_ble_gap_disconn_evt_t *p_gap_disconn_evt_param =
                                 (st_ble_gap_disconn_evt_t *)p_data->p_param;
                        if(p_gap_disconn_evt_param->conn_hdl == g_central_conn_hdl)
                        {
                                g_central_conn_hdl = BLE_GAP_INVALID_CONN_HDL;
                        }
                } break;

                case BLE_GAP_EVENT_CONN_PARAM_UPD_REQ:
                {
                        st_ble_gap_conn_upd_req_evt_t *p_conn_upd_req_evt_param =
                                 (st_ble_gap_conn_upd_req_evt_t *)p_data->p_param;
                        if(p_conn_upd_req_evt_param->conn_hdl == g_central_conn_hdl)
                        {
                                st_ble_gap_conn_param_t conn_updt_param = {
                                        .conn_intv_min = p_conn_upd_req_evt_param->conn_intv_min,
                                        .conn_intv_max = p_conn_upd_req_evt_param->conn_intv_max,
                                        .conn_latency  = p_conn_upd_req_evt_param->conn_latency,
                                        .sup_to        = p_conn_upd_req_evt_param->sup_to,
                                };

                                R_BLE_GAP_UpdConn(p_conn_upd_req_evt_param->conn_hdl,
                                                                  BLE_GAP_CONN_UPD_MODE_RSP,
                                                                  BLE_GAP_CONN_UPD_ACCEPT,
                                                                  &conn_updt_param);
                        }
                } break;
                case BLE_GAP_EVENT_ADV_REPT_IND:
                {
                        st_ble_gap_adv_rept_evt_t *p_adv_rept_param =
                                 (st_ble_gap_adv_rept_evt_t *)p_data->p_param;
                        st_ble_gap_ext_adv_rept_t *p_ext_adv_rept_param =
                                 (st_ble_gap_ext_adv_rept_t *)p_adv_rept_param->param.p_ext_adv_rpt;

                        gs_conn_param.p_addr->type = p_ext_adv_rept_param->addr_type;
                        memcpy(gs_conn_param.p_addr->addr, p_ext_adv_rept_param->p_addr, BLE_BD_ADDR_LEN);

                        R_BLE_GAP_StopScan();
                } break;

                case BLE_GAP_EVENT_SCAN_OFF:
                {
                        R_BLE_ABS_CreateConn(&gs_conn_param);
                }break;

                default:
                {
                        /* Do nothing. */
                } break;
        }
}
```

Code 7-7 Example of GAP callback function when connecting as a central role

```
/* Connection handle */
uint16_t g_peripheral_conn_hdl;

static void ble_peripheral_gapcb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
        } break;

        case BLE_GAP_EVENT_CONN_IND:
        {
            if (BLE_SUCCESS == result)
            {
                st_ble_gap_conn_evt_t *p_gap_conn_evt_param = (st_ble_gap_conn_evt_t *)p_data->p_param;
                if(0x01 == p_gap_conn_evt_param->role)
                {
                g_peripheral_conn_hdl = p_gap_conn_evt_param->conn_hdl;
                }
            }
        } break;

        case BLE_GAP_EVENT_CONN_PARAM_UPD_REQ:
        {
            st_ble_gap_conn_upd_req_evt_t *p_conn_upd_req_evt_param =
                  (st_ble_gap_conn_upd_req_evt_t *)p_data->p_param;

            if(p_conn_upd_req_evt_param->conn_hdl == g_peripheral_conn_hdl)
            {
                st_ble_gap_conn_param_t conn_updt_param = {
                    .conn_intv_min = p_conn_upd_req_evt_param->conn_intv_min,
                    .conn_intv_max = p_conn_upd_req_evt_param->conn_intv_max,
                    .conn_latency  = p_conn_upd_req_evt_param->conn_latency,
                    .sup_to        = p_conn_upd_req_evt_param->sup_to,
                };

                R_BLE_GAP_UpdConn(p_conn_upd_req_evt_param->conn_hdl,
                                  BLE_GAP_CONN_UPD_MODE_RSP,
                                  BLE_GAP_CONN_UPD_ACCEPT,
                                  &conn_updt_param);
            }
        } break;

        case BLE_GAP_EVENT_DISCONN_IND:
        {
            st_ble_gap_disconn_evt_t *p_gap_disconn_evt_param =
                  (st_ble_gap_disconn_evt_t *)p_data->p_param;
            if(p_gap_disconn_evt_param->conn_hdl == g_peripheral_conn_hdl)
            {
                g_peripheral_conn_hdl = BLE_GAP_INVALID_CONN_HDL;
            }
        } break;


        default:
        {
        /* Do Nothing */
        }break;
}
```

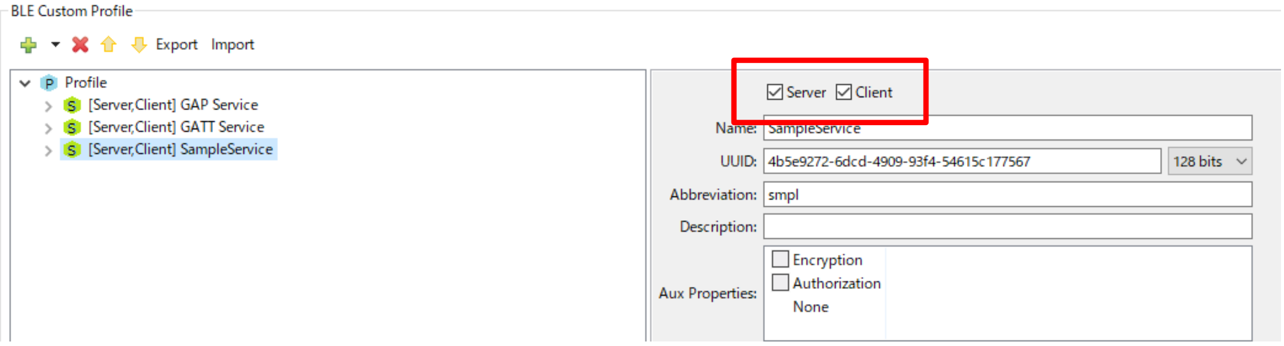
Code 7-8 Example of GAP callback function when connected as a peripheral device

GAP callback function is implemented for each role.

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
        ble_peripheral_gapcb(type, result, p_data);
        ble_central_gapcb(type, result, p_data);
}
```

Code 7-9 Call GAP callback function for each role

Applications with multi role connections may implement both GATT clients and GATT servers. Use QE for BLE to generate service API for both GATT client and GATT server. On the QE for BLE service screen, check both the server and client and generate the code.



BLE Custom Profile
Export
Import
Profile
[Server,Client] GAP Service
[Server,Client] GATT Service
[Server,Client] SampleService

☑ Server ☑ Client

Name: SampleService

UUID: 4b5e9272-6dcd-4909-93f4-54615c177567 128 bits

Abbreviation: smpl

Description:

Aux Properties: □ Encryption □ Authorization None

Figure 7-7 Select GATT Role on Service Screen

This time, when it is a central device, it operates as a GATT client, so service discovery is performed when it is connected to a peripheral device.

```
/* XXX Service UUID */
static uint8_t XXXC_UUID[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00 };

/* Service discovery parameters */
static st_ble_disc_entry_t gs_disc_entries[] = {

    {
        .p_uuid     = XXXC_UUID,
        .uuid_type  = BLE_GATT_128_BIT_UUID_FORMAT,
        .serv_cb    = R_BLE_XXXC_ServDiscCb,
    },
};
static void disc_comp_cb(uint16_t conn_hdl)
{
    /* TODO: Add function after discovery completed */
    return;
}

static void gattc_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
    R_BLE_SERVC_GattcCb(type, result, p_data);

    switch(type)
    {
        /* TODO: Set callback events of GATTC. Check BLE API reference for events. */

        case BLE_GATTC_EVENT_CONN_IND:
        {
        if(g_central_conn_hdl == p_data->conn_hdl)
        {
                R_BLE_DISC_Start(p_data->conn_hdl, gs_disc_entries, ARRAY_SIZE(gs_disc_entries),
disc_comp_cb);
        }
        } break;

        default:
        {
            /* Do nothing. */
        } break;
    }
}
```

Code 7-10 Implementation example of service discovery as a central device

If you register R_BLE_XXXC_ServDiscCb of Service API (r_ble_xxxc.c) generated by QE for BLE in Discovery in Profile Common of app_lib (bold frame in **Code 7-10**), attribute handle of each device is retained in Service API through Profile Common. By using the Service API, the application can access the GATT database of each device using the connection handle without managing the attribute handle of each device.

7.4 Disconnection

If the currently established link is disconnected, call the following API.

```
ble_status_t R_BLE_GAP_Disconnect(uint16_t conn_hdl, uint8_t reason)
```

Specify the connection handle with the conn_hdl parameter and the disconnection reason with the reason parameter. Normally, 0x13 (REMOTE USER TERMINATED CONNECTION) is specified as the disconnection reason. For more information about the disconnection reason, refer to “Bluetooth Core Specification Vol. 2 Part D, 2 Error Code Descriptions”. Central and peripheral device can call this API.

When the disconnection occurs, the BLE_GAP_EVENT_DISCONN_IND event is notified to the application. The disconnection reason is notified in the reason field of the st_ble_gap_disconn_evt_t structure by the BLE_GAP_EVENT_DISCONN_IND event.

If the local device disconnects the link by R_BLE_GAP_Disconnect,
reason: 0x16 (Connection Terminated by Local Host) will be notified.

If the remote device disconnects the link,
reason: 0x13 (Remote User Terminated Connection) will be notified in most cases. Otherwise, the disconnection reason that the remote device specifies will be notified.

If no packet is received within 6 connection intervals after starting a connection,
(for example, in an environment with many active scanning devices, peripheral device is busy responding to scan requests and cannot respond to connection request)
reason: 0x3E (Connection Failed to be Established) is notified.
As for connection interval, refer to "8.3 Updating connection parameter".

After establishing a connection, if no packet is received within the supervision timeout period,
reason: 0x08 (Connection Timeout) is notified.
As for supervision timeout, refer to "8.3 Updating connection parameter".

If the LTK of the local device and the remote device do not match when encryption is started,
reason: 0x3D (Connection Terminated due to MIC Failure) is notified.
Delete the bonding information and try pairing again, as the remote device cannot be trusted.
As for LTK, refer to "9.1 Pairing".

Note: When reconnecting to a disconnected remote device, the peripheral side needs to execute Connectable Advertising again.

8. Communication

In Bluetooth Low Energy, you can adjust the communication speed and power consumption to suit your application by changing the communication parameters. Table 8.1 shows the communication parameters described in this chapter and the libraries that support the functions. The optional feature may not be supported by the remote device.

Table 8.1 Communication parameters

Communication Parameter	Feature name	Description	Library	Role
PHY	LE 2M PHY LE Coded PHY LE 1M PHY (optional other than 1M)	Determined by Central's connection request and Peripheral's Advertising parameters when connecting. This can be changed after the connection.	All features / Balance All features / Balance All libraries	Central / Peripheral
Maximums transmit packet length	LE Data Length Extension (optional)	Can extend Maximum number of transmitted bytes 27 → 251 bytes. The initial value is the value specified by BLE_CFG_RF_CONN_DATA_MAX. This can be changed after the connection.	All libraries	Central / Peripheral
Connection parameters	-	Determined by Central's connection request parameters when connecting. This can be changed after the connection.	All libraries	Central / Peripheral
MTU	-	The initial value is 23 bytes. This can be changed only once during the connection.	All libraries	Client

The following explains how to use the API to change the communication parameters. Refer to the R_BLE API document (r_ble_api_spec.chm) included in the "RX23W Group BLE Module Firmware Integration Technology Application Note (R01AN4860)" for details on the API.

8.1 Changing PHY

PHY is a parameter that indicates the physical layer modulation method and coding scheme. Changing this parameter, it is expected that throughput and radio wave reach will be improved. The modulation method and coding scheme are shown below.

- LE 1M PHY

  This is the basic modulation method of Bluetooth Low Energy. Compatible with all Bluetooth Low Energy devices. Set for applications that connect to an unspecified number of devices.

- LE 2M PHY

  This is a modulation method that doubles the symbol rate from LE 1M PHY and shortens the packet transmission time. It is used when performing high throughput communication. Since the packet transmission time is shortened, you can expect a reduction in power consumption.

- LE Coded PHY

  A modulation method in which a forward error correction code (coding scheme) of 1/2 or 1/8 is added to the header and payload of the packet. Improves packet arrival rate. It increases the certainty of data arrival and makes it possible to extend the communication distance compared to the past.

To change the PHY, use the R_BLE_GAP_SetPhy function of GAP API. For the argument, specify the connection handle whose settings you want to change, the modulation scheme for transmission (tx_phys), the modulation scheme for reception (rx_phys), and the coding scheme for transmission (phy_options). The receiving coding scheme does not change.

Figure 8-1 show the sequence chart when changing the PHY from the local device. Local device can change it from either role. The remote device can specify the PHY that allows changes with the R_BLE_GAP_SetDefPhy function. If not specified, all PHYs are accepted.

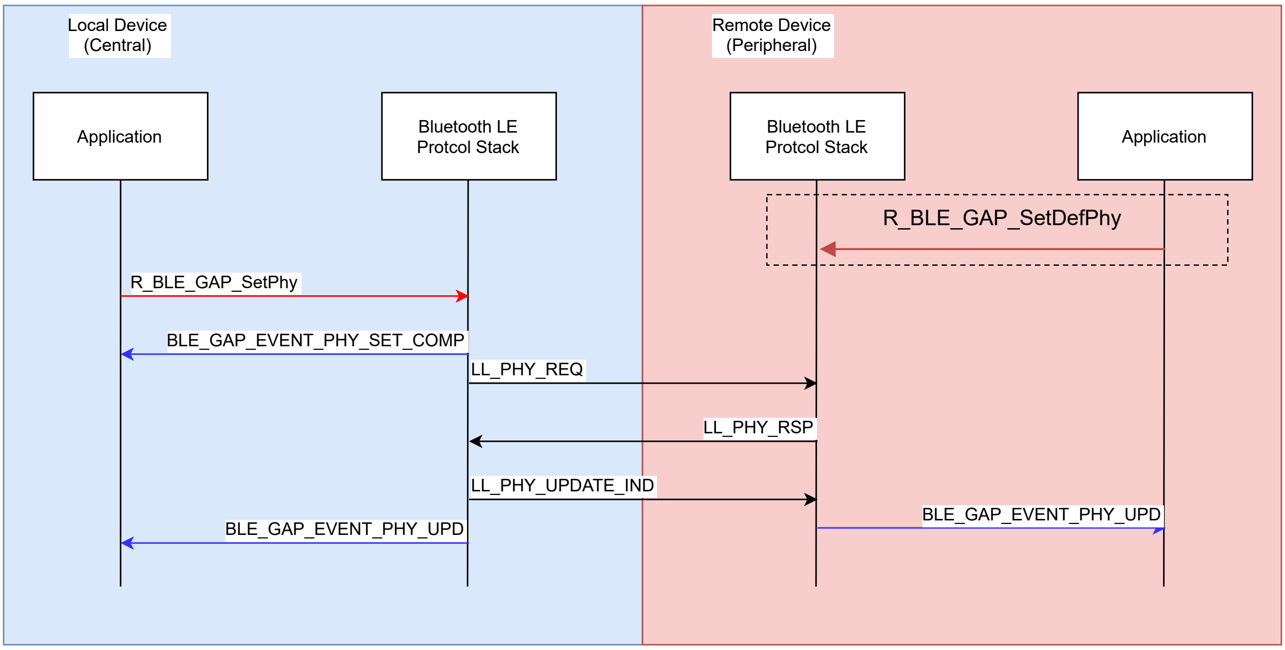


Figure 8-1 Sequence chart when changing PHY

The sample code when changing the PHY to LE Coded PHY (S=8) is shown below. Multiple PHYs can be specified by bit sum. If you specify multiple PHYs, including the PHY in use, the PHY in use will not change. If you specify multiple PHYs without including the PHY in use, the PHY will change to the fastest PHY. The PHY you specify also applies to PHYs that allow modification from remote devices. The PHY in use can be obtained with the R_BLE_GAP_ReadPhy function.

```
st_ble_gap_set_phy_param_t set_phy = {
     .tx_phys = BLE_GAP_SET_PHYS_HOST_PREF_CD,
     .rx_phys = BLE_GAP_SET_PHYS_HOST_PREF_CD,
     .phy_options = BLE_GAP_SET_PHYS_OP_HOST_PREF_S_8
};

R_BLE_GAP_SetPhy(conn_hdl, &set_phy);
```

Code 8-1 Code to change PHY to LE Coded PHY (S=8)

Due to the change of PHY, two events are notified to the application. These events are notified to the GAP callback function (gap_cb).

- **BLE _GAP_EVENT_PHY_SET_COMP**

  Notified when the controller layer of the local device accepts the PHY change.

- **BLE_GAP_EVENT_PHY_UPD**

  Notified when the remote device accepts the PHY change. The notified event data, tx_phy and rx_phy, represent the actual PHY used when transmitting from the local device to the remote device and from the remote device to the local device respectively.

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch (type)
    {
      case BLE_GAP_EVENT_PHY_SET_COMP:
      {
        if(BLE_SUCCESS == result)
        {
          st_ble_gap_conn_hdl_evt_t *event_data =
            (st_ble_gap_conn_hdl_evt_t *)p_data->p_param;
          /*PHY parameter change in event_data->conn_hdl reaches Link Layer */
        }
        else if(BLE_ERR_INVALID_HDL == result)
        {
          st_ble_gap_conn_hdl_evt_t *event_data =
              (st_ble_gap_conn_hdl_evt_t *)p_data->p_param;
          /*The connection for event_data->conn_hdl was not found.*/
        }
        else
        {
          /* Do Nothing */
        }
      } break;

      case BLE_GAP_EVENT_PHY_UPD:
      {
        st_ble_gap_phy_upd_evt_t * event_data =
            (st_ble_gap_phy_upd_evt_t *)p_data->p_param;
      } break;
    }
}
```

Code 8-2 Event that occurs when PHY is changed

When the PHY is changed, the transmission time for the transmission packet length changes. The Bluetooth LE Protocol Stack will also automatically change the maximum transmission packet length described later according to the PHY. When changed to LE Coded PHY, the maximum transmission packet length is set to 251 bytes and the transmission time is set to 27 bytes, 2704µsec. If changing the maximum send packet length to 28 bytes or more, see "8.2 Changing maximum transmission packet length" below.

8.2 Changing maximum transmission packet length

This parameter sets the maximum packet length in the Link Layer. When transmitting and receiving application data that exceeds 23 bytes, you can perform efficient communication by extending the transmitting packet length. Packet length extension requires the remote device to support the LE Data Packet Length Extension feature developed in Bluetooth 4.2.

To change the maximum transmission packet length, specify the maximum number of bytes to be transmitted and the maximum transmission time. The packet transmission time is depended on the PHY settings in the previous chapter. The maximum transmitting packet length and maximum transmit time that can be set depending on whether the LE Data Packet Length Extension and LE Coded PHY are supported are shown below.

Table 8.2 Relationship between PHY and maximum transmit packet length and maximum transmit time

LE Data Packet Length Extension	LE Coded PHY feature supported	Parameters with names ending in “Octets”		Parameters with names ending in “Time"	
		Min	Max	Min	Max
No	No	27	27	328	328
Yes	No	27	251	328	2120
No	Yes	27	27	328	2704
Yes	Yes	27	251	328	17040

Bluetooth Core Specification V5.00 Vol 6, Part B

When connected to a remote device, the Bluetooth LE Protocol Stack request to change the maximum transmission packet length to the value specified by BLE_CFG_RF_CONN_DATA_MAX.

To change the maximum transmission packet length, use the R_BLE_GAP_SetDataLen function of GAP API. For the argument, specify the connection handle whose settings you want to change, the maximum number of bytes to send, and the maximum send time. Enter the maximum transmission time in microseconds. The Bluetooth LE Protocol Stack gives priority to the smaller of the specified maximum number of transmission bytes and maximum transmission time. Figure 8-2 show the sequence chart when changing the maximum transmission packet length.

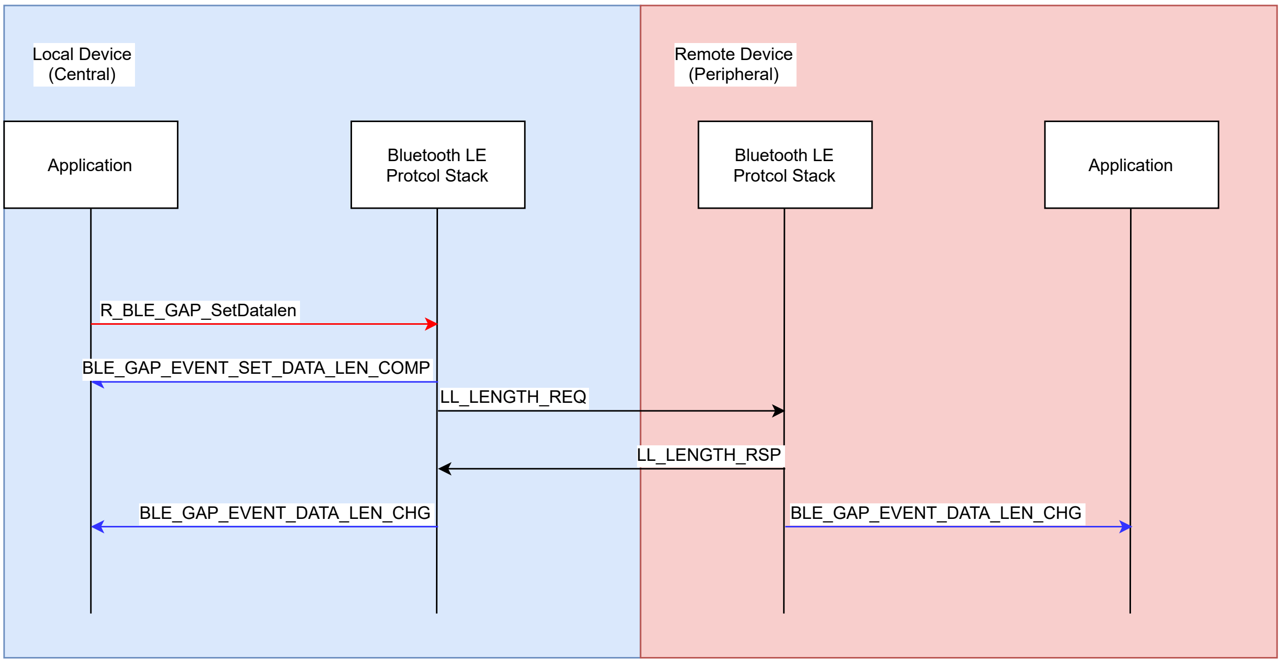


Figure 8-2 Sequence chart when changing the maximum transmission packet length

Below is an example of expanding the packet length to 251 bytes when using the LE 1M PHY.

```
uint16_t tx_octets = 251;
uint16_t tx_time = 2120;

R_BLE_GAP_SetDataLen(conn_hdl, tx_octets, tx_time);
```

Code 8-3 Example of transmit packet length change request

Two events are notified to the application by changing the transmission packet length. These events are notified to the GAP callback function (gap_cb).

- BLE_GAP_EVENT_SET_DATA_LEN_COMP
  Occurs when the change in transmitted packet length is accepted by the controller layer.

- BLE_GAP_EVENT_DATA_LEN_CHG
  Occurs when the send packet length changes with the remote device. This does not occur if the other party does not support LE Data Packet Length Extension.

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
       switch(type)
       {
              case BLE_GAP_EVENT_SET_DATA_LEN_COMP:
              {
                    st_ble_gap_conn_hdl_evt_t * event_data =
                  (st_ble_gap_conn_hdl_evt_t *)p_data->p_param;
                    /* Do Nothing */
              } break;
              case BLE_GAP_EVENT_DATA_LEN_CHG:
              {
                    st_ble_gap_data_len_chg_evt_t * event_data =
                  (st_ble_gap_data_len_chg_evt_t *)p_data->p_param;
                    /* Do Nothing */
              } break;
       }
}
```

Code 8-4 Change packet length event

8.3 Updating connection parameter

Connection parameters are parameters related to communication frequency. Setting connection parameters is important for the efficient operation of your application. The connection parameters include the following items.

- Connection Interval

  The interval between packet exchanges. Shortening the connection interval improves throughput and power consumption. On the contrary, if you lengthen the connection interval, the power consumption will decrease.

- Peripheral Latency

  The number of times the Peripheral will ignore packets from the Central. When the Peripheral receives a packet from the Central, it returns a response. If there is no data to be transmitted from the Peripheral, the packet from the Central can be ignored for the number of times set for Peripheral Latency. The Peripheral does not have to return the response for that number of times, so the power consumption can be reduced.

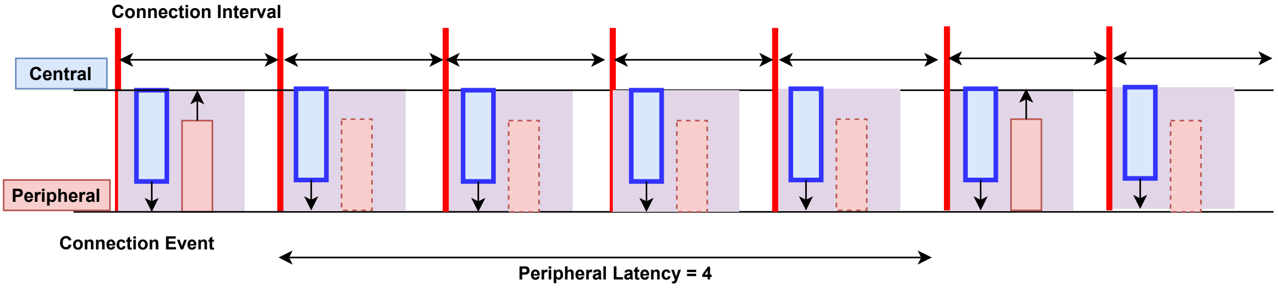


Figure 8-3 Schematic diagram of Peripheral Latency and connection event

- Supervision Timeout

  This is the time from when the packet reception is stopped until the disconnection. If no packet arrives within this time after the last packet is received, it is determined to be disconnected. Set to perform packet exchange more than once within the supervision timeout period.

$$Supervision\ Timeout(msec) > \left(1 + Peripheral\ Latency(number)\right) * Connection\ Interval(msec) * 2$$

- Connection Event Time

  Specify the connection event time that occurs at each connection interval. If 0 is set, packets will be exchanged only once for each round trip per connection event, and if 0xffff is specified, packets will be exchanged until the next connection event or until the More Data bit is not set.

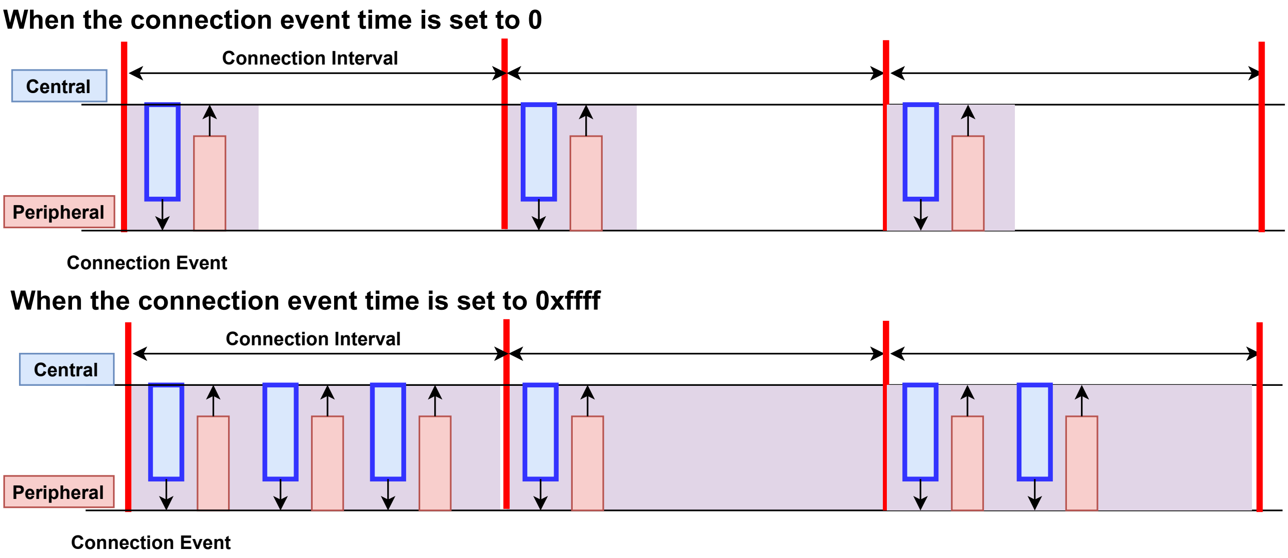


Figure 8-4 Schematic diagram of connection event time and packet exchange

The Central determines and changes the connection parameters, but Peripherals can request the changes. Also, the connection parameters can be updated any number of times during the connection. The application flexibly updates the connection parameters to achieve efficient data communication. For example, it is effective to change the connection interval at the following cases.

- **In case that application will set connection interval shorter.**

  If there is no data to send for a while

  Perform data communication simultaneously with multiple communication partners

- **In case that application will set connection interval longer.**

  Run service discovery

  Send small data in a short time at once

Figure 8-5 show the sequence chart for updating the connection parameters. The local device is the central and the remote device is the peripheral. For connection parameter updates, the PDUs that the Link Layer interacts with will depend on the role of the device being updated and support for the procedure, but at the application level, there is not much difference. For other roles, please refer to the R_BLE API document (r_ble_api_spec.chm) included in "RX23W Group BLE Module Firmware Integration Technology Application Note (R01AN4860)" for details of PDUs exchanged in Link Layer.

Note: The parameter described as Connection Latency in the R_BLE_API document is the same as Peripheral Latency. Read this as Peripheral Latency in this document.

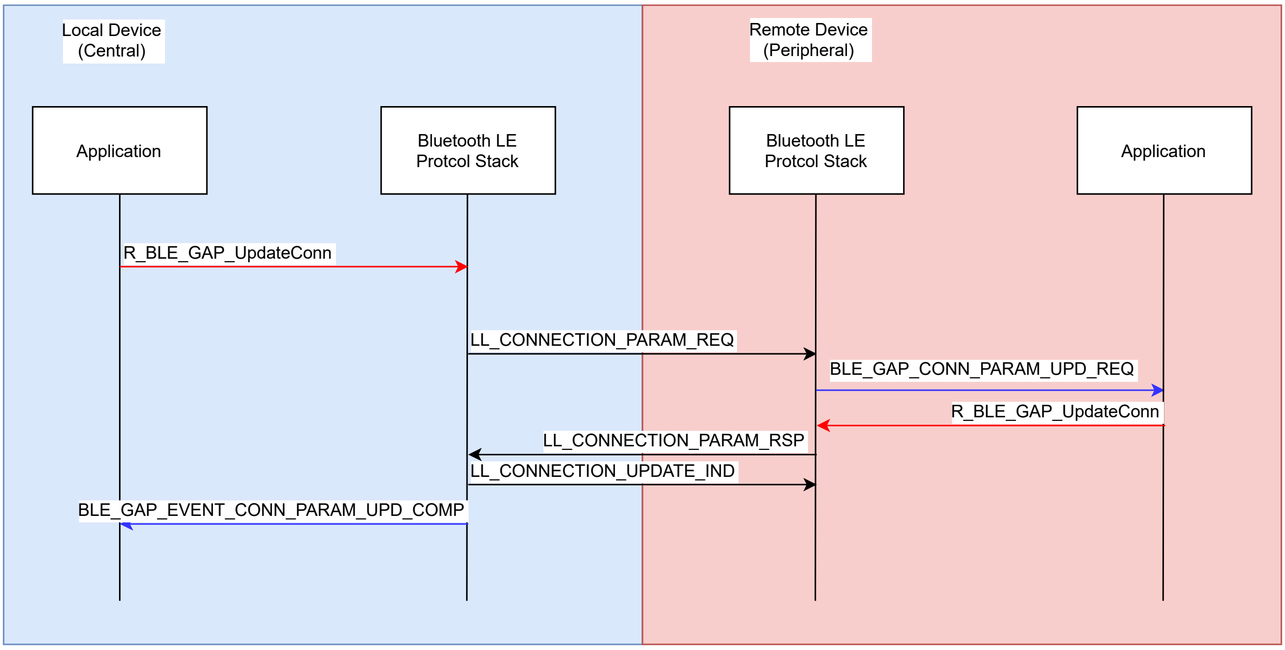


Figure 8-5 Sequence chart when updating connection parameters

Use R_BLE_GAP_UpdConn function of GAP_API for request/response of connection parameter update. The following is an example of requesting to update the connection parameters from the local device.

```
st_ble_gap_conn_param_t conn_param = {
       .conn_intv_min = 0x0006,  //Connection Interval
       .conn_intv_max = 0x0006,
       .conn_latency  = 0x0000,  //Peripheral Latency
       .sup_to         = 0x0C80, //Supervision timeout
       .max_ce_length = 0xffff,  //Connection event time
       .min_ce_length = 0xffff
};

R_BLE_GAP_UpdConn(conn_hdl , BLE_GAP_CONN_UPD_MODE_REQ , 0 , &conn_param);
```

Code 8-5 Implementation example of connection parameter update request

The application is notified of two events by updating the connection parameters. These events are notified to the GAP callback function (gap_cb).

- BLE_GAP_EVENT_CONN_PARAM_UPD_REQ

  Notified when a request to update connection parameters is received from the remote device. Implement the process of whether to accept the request.

- BLE_GAP_EVENT_CONN_PARAM_UPD_COMP

  You will be notified when the connection parameters have been updated. The result variable contains information about whether the request to update the connection parameters was accepted, and the event variable contains the connection parameters used in the actual connection.

The following is an implementation example of the response to the connection parameter update request from the remote device. In this example, it accepts all requests from remote devices. This process is implemented in app_main.c generated by QE for BLE.

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch(type)
    {
      case BLE_GAP_EVENT_CONN_PARAM_UPD_REQ:
       {
            st_ble_gap_conn_upd_req_evt_t *p_conn_upd_req_evt_param =
                              (st_ble_gap_conn_upd_req_evt_t *)p_data->p_param;

            st_ble_gap_conn_param_t conn_updt_param = {
              .conn_intv_min = p_conn_upd_req_evt_param->conn_intv_min,
              .conn_intv_max = p_conn_upd_req_evt_param->conn_intv_max,
              .conn_latency  = p_conn_upd_req_evt_param->conn_latency,
              .sup_to    = p_conn_upd_req_evt_param->sup_to,
            };

            R_BLE_GAP_UpdConn(p_conn_upd_req_evt_param->conn_hdl,
                      BLE_GAP_CONN_UPD_MODE_RSP,
                      BLE_GAP_CONN_UPD_ACCEPT,
                      &conn_updt_param);
      } break;
    }
}
```

Code 8-6 Implementation example of response to connection parameter update request event

When connecting to a smartphone, update of connection parameters may not be accepted depending on the OS. For example, for iOS, design guidelines for accessories for Apple devices (https://developer.apple.com/accessories/Accessory-Design-Guidelines.pdf)

If the remote device rejects, BLE_ERR_INVALID_ARG(0x0003) is stored in the result variable at the time of BLE_GAP_EVENT_CONN_PARAM_UPD_COMP event notification.

The following is an implementation example in which the parameters are updated and requested again after being rejected by the remote device.

```
Static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch(type)
    {
      case BLE_GAP_EVENT_CONN_PARAM_UPD_COMP:
       {
         if(BLE_ERR_INVALID_ARG == result)
         {
               st_ble_gap_conn_param_t conn_param = {
                .conn_intv_min = 0x0028,     /* Connection Interval */
                .conn_intv_max = 0x0028,
                .conn_latency  = 0x0000,     /* Peripheral Latency */
                .sup_to        = 0x0C80,     /* Supervision timeout */
                .max_ce_length = 0xffff,     /* Connection event time */
                .min_ce_length = 0xffff
                };

              R_BLE_GAP_UpdConn(conn_hdl ,
                             BLE_GAP_CONN_UPD_MODE_REQ ,
                             0 ,
                             &conn_param);

         }
       } break;
    }
}
```

Code 8-7 Request to update connection parameters after being rejected by remote device

8.4 Changing MTU

MTU represents the maximum packet length in GATT. The initial value is the minimum value of 23 bytes. This is called the default MTU. The maximum size when performing data communication by Read Characteristic Value, Write Characteristic Value, Write Without Response, Notification, and Indication operations, which are the main procedures of GATT, depends on the MTU.

When the default MTU is used, the client uses GATT Read Long Characteristic Value to read data greater than 22 bytes and Write Long Characteristic Value to write data greater than 20 bytes. These procedures have higher communication overhead than Read Characteristic Value and Write Characteristic Value. Also, with the default MTU, data greater than 20 bytes cannot be sent by Notification or Indication from the server. The MTU can be changed from the GATT client only once during the connection.

To minimize overhead, adjust the relationship between MTU and maximum send packet length to be below.

$$MTU(byte) = Maximum\ transmission\ packet\ length(byte) - 4(byte)$$

Figure 8-6 show the sequence chart when changing the MTU.

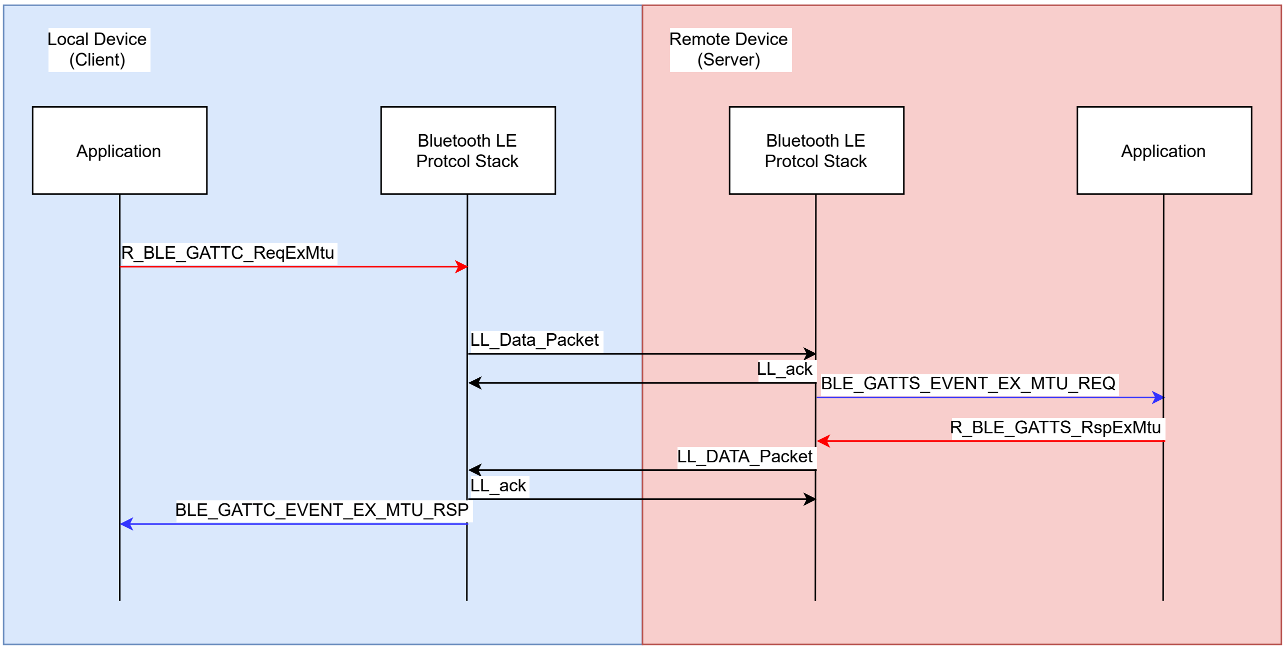


Figure 8-6 Sequence chart when changing MTU

To change the MTU, use the R_BLE_GATTC_ReqExMtu function of GATT Client API. Specify the supported MTU as an argument.

```
Uint16_t mtu = 247
R_BLE_GATTC_ReqExMtu(conn_hdl, mtu);
```

Code 8-8 MTU change request example

Two events are notified to the application by changing the MTU. These events are notified to the GATT client or GATT server callback functions (gattc_cb, gatts_cb).

- BLE_GATTS_EVENT_EX_MTU_REQ

  The server is notified when an MTU change request is received from a client device (gatts_cb). The server returns the MTU it supports in this event.

- BLE_GATTC_EVENT_EX_MTU_RSP

  The client is notified when it receives an Exchange MTU Response from the server device (gattc_cb). The smaller of the MTU supported by itself and the MTU included in the response is the actual MTU used.

Code 8-9 show an example implementation of a response to a GATT server Exchange MTU Request. For the response, use R_BLE_GATTS_RspExMtu function of GATT Server API. For the argument, specify the MTU supported by the local device. This process is implemented in R_BLE_SERVS_GattsCb function provided by Profile Common Server Library of app_lib. The size of the MTU returned by the GATT server is set in the BLE_CFG_GATT_MTU_SIZE configuration option. If you want to generate GATT server code from QE for BLE, your application does not need to implement MTU response.

```
Static void gatts_cb(uint16_t type, ble_status_t result, st_ble_gatts_evt_data_t
*p_data)
{
    switch (type)
    {
        case BLE_GATTS_EVENT_EX_MTU_REQ:
        {
            R_BLE_GATTS_RspExMtu(p_data->conn_hdl, BLE_CFG_GATT_MTU_SIZE);
        } break;
    }
}
```

Code 8-9 Example of response to MTU change request

8.5 Flow control

The Bluetooth LE Protocol Stack has a flow control function to send large application data in a short time. To realize the flow control function, the Bluetooth LE Protocol Stack has 10 send buffers for application communication. When the flow control function is enabled, the application is notified of events according to the number of empty send buffers.

The table below shows the number of empty buffers and event notification timing. The event is triggered when the application repeatedly calls the send function and the number of empty buffers decreases to the set lower limit. In response to this event, the application stops calling the send function and prevents the buffer from overflowing.

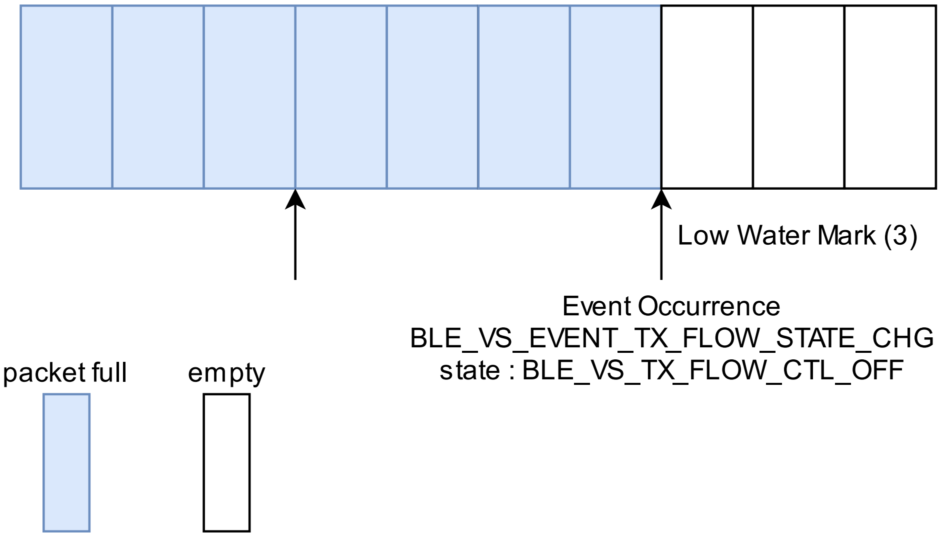

Low Water Mark (3)
Event Occurrence
BLE_VS_EVENT_TX_FLOW_STATE_CHG
state : BLE_VS_TX_FLOW_CTL_OFF
packet full
empty

Figure 8-7 Number of empty buffers and events

When the Bluetooth LE Protocol Stack transmits to the remote device, the number of empty buffers increases. An event occurs when the number of empty buffers reaches the set upper limit. The event is triggered when the application repeatedly calls the send function and the number of empty buffers decreases to the set lower limit. Upon receiving this event, the call to the send function is resumed. By repeating this, large data can be transmitted efficiently.

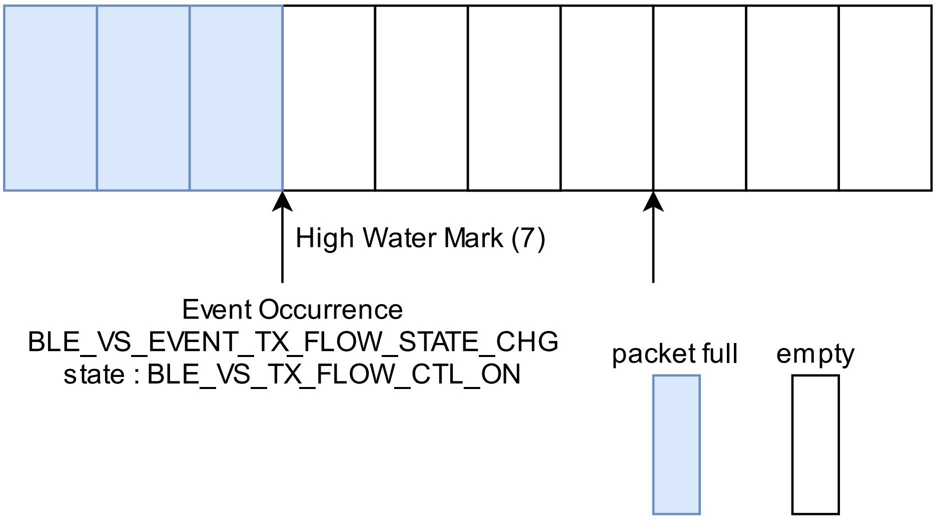


Figure 8-8 Number of empty buffers and events

The flow control function is enabled by the R_BLE_VS_SetTxLimit function and R_BLE_VS_StartTxFlowEvtNtf function of Vendor Specific API.

Use the R_BLE_VS_SetTxLimit function to set the lower limit and upper limit of the empty number of the buffer where the event occurs. Execute the R_BLE_VS_StartTxFlowEvtNtf function to enable event notification.

```
/* Enable Vender Specific Tx Flow Control */
#define LOW_WATER_MARK     (3)
#define HIGH_WATER_MARK    (7)

R_BLE_VS_SetTxLimit(LOW_WATER_MARK, HIGH_WATER_MARK);
R_BLE_VS_StartTxFlowEvtNtf();
```

Code 8-10 Start of flow control feature

The flow control feature notifies the application of the BLE_VS_EVENT_TX_FLOW_STATE_CHG event.

Information indicating the current buffer status is stored in this event variable. An example of using the flow control function is shown below. In this example, when the empty number in the buffer recovers to the High Water Mark, the send function is called only (10-Low Water Mark) times and continuous transmission is performed so that the buffer does not overflow. R_BLE_ServsCharNotification function is a sample. Please rewrite to the function of the service used.

```
Static void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
      R_BLE_SERVS_VsCb(type, result, p_data);

      switch(type)
      {
          case BLE_VS_EVENT_TX_FLOW_STATE_CHG:
          {
              /* Apprize TxFlowState changed to txflow API */
            st_ble_vs_tx_flow_chg_evt_t * evt_data=
             (st_ble_vs_tx_flow_chg_evt_t *)p_data->p_param;
              if(BLE_VS_TX_FLOW_CTL_ON == evt_data->state)
              {
                for (int i=0; i<(10-LOW_WATER_MARK); i++)
               {
                   R_BLE_ServsCharNotification(conn_hdl, &app_data);
               }
              }
              else
              {
                /* Do Nothing */
              }
          } break;

}
```

Code 8-11 Implementation example of sending by flow control feature event

8.6 High throughput communication

When performing high-throughput communication using Bluetooth Low Energy, it is important to set the communication parameters to optimal values and call the send function continuously using the flow control function. Please refer to the application note “Sample program for high-speed communication (R01AN5437)” for details on high-throughput communication.

9. Security

This section describes the security functions provided by Bluetooth Low Energy.

In order to select the necessary security functions for the final product, it is important to determine the product use case and clarify the security requirements.

9.1 Pairing

Pairing procedure is necessary to use Bluetooth security features. In the case such as the following, pairing is necessary.

- There is GATT characteristics that can be changed.
- Requires address resolving for private addresses.
- Protect a device from malicious attacks such as data eavesdropping, falsification, and device tracking.

Pairing exchanges the keys with a remote device. The keys to be exchanged are followings. The remote device keys are notified by BLE_GAP_EVENT_PEER_KEY_INFO event. For more information about how to get the keys, see “9.1.7 Key exchange”.

- LTK (Long Term Key), EDIV, Rand
  Communication data encryption uses LTK (Key exchange is not enforced in LE Secure Connections).

- IRK (Identity Resolving Key), Identity Address
  Privacy function uses IRK.

- CSRK (Connection Signature Resolving Key)
  Signed data send/receive uses CSRK.

Pairing mechanism has LE Legacy pairing and LE Secure Connections.

LE Secure Connections is supported from Bluetooth version 4.2. LE legacy pairing is the paring mechanism is used by the device which does not support LE Secure Connections.

If a remote device supports LE Secure Connections, the Bluetooth LE Protocol Stack performs LE Secure Connections. If a remote device does not support LE Secure Connections, the Bluetooth LE Protocol Stack performs LE Legacy Pairing.

BP: LE Secure Connections is the most secure pairing and encryption mechanism where LTK is not sent over the air. LTK also has enough entropy at 16 octets to protect encrypted links from brute force attacks. Because of these things, for those designing more secure products, recommend supporting LE Secure Connections unless there are restrictions on the remote device side.
In order to avoid attacks from attackers who aim at the security down mechanism, when performing pairing only with LE Secure Connections (not accept LE legacy pairing), set BLE_GAP_SC_STRICT(0x01) in Table 9.9 to the sec_conn_only member of the parameter structure of BLE_GAP_SetPairingParams or R_BLE_ABS_Init API.

BP: Signed data is provided for devices that want to avoid the overhead of encryption, and by ensuring data integrity, data falsification by attackers can be avoided. However, since it does not support protection against eavesdropping through encryption, it is possible for attackers to use replay attacks (unauthorized access by spoofing the user by using the data obtained by eavesdropping on communication). Therefore, it is recommended to avoid using signed data and support encryption.

The pairing procedure in an application is shown in Figure 9-1. The following sections describe the details of pairing steps.

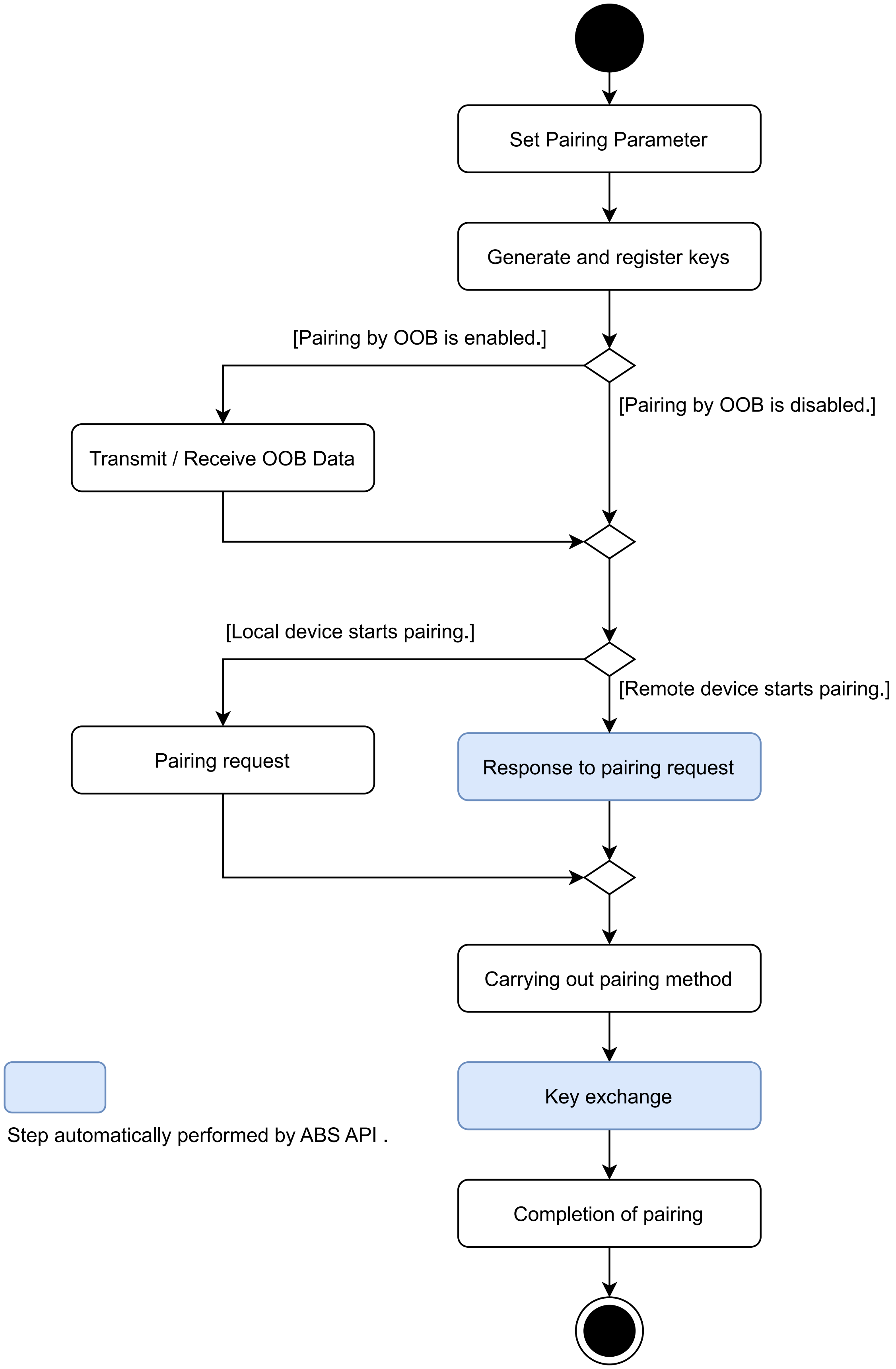

Set Pairing Parameter
Generate and register keys
[Pairing by OOB is enabled.]
[Pairing by OOB is disabled.]
Transmit / Receive OOB Data
[Local device starts pairing.]
[Remote device starts pairing.]
Pairing request
Response to pairing request
Carrying out pairing method
Key exchange
Completion of pairing
Step automatically performed by ABS API .

Figure 9-1 Pairing procedure in application

9.1.1 Pairing Parameters

Set the pairing parameters before starting the pairing procedure. The pairing parameters are set with the following APIs. Call the API before starting pairing.

R_BLE_GAP_SetPairingParams

R_BLE_ABS_Init

Table 9.1 shows the pairing parameters. The following sections describes the details of the parameters.

Table 9.1 Pairing Parameters

	API	R_BLE_ABS_Init	R_BLE_GAP_SetPairingParams	Value Range	QE for BLE default settings R_BLE_ABS_Init
	Parameter Structure	st_ble_abs_pairing_param_t	st_ble_gap_pairing_param_t		
1. Input Output capabilities		iocap	iocap	BLE_GAP_IOCAP_DISPLAY_ ONLY(0x00)	BLE_GAP_IOCAP_NOINPUT_ NOOUTPUT(0x03)
				BLE_GAP_IOCAP_DISPLAY_ YESNO(0x01)	
				BLE_GAP_IOCAP_KEYBOARD_ ONLY(0x02)	
				BLE_GAP_IOCAP_NOINPUT_ NOOUTPUT(0x03)	
				BLE_GAP_IOCAP_KEYBOARD_ DISPLAY(0x04)	
2. MITM Protection Request		mitm	mitm	BLE_GAP_SEC_MITM_BEST_ EFFORT(0x00)	BLE_GAP_SEC_MITM_BEST_ EFFORT(0x00)
				BLE_GAP_SEC_MITM_STRICT(0x01)	
3. Bonding		No parameter Fixed to BLE_GAP_ BONDING(0x01)	bonding	BLE_GAP_BONDING_NONE(0x00)	BLE_GAP_BONDING(0x01)
				BLE_GAP_BONDING(0x01)	
4. Encryption Key Size	Max Size	No parameter Fixed to 16	max_key_size	7 to 16	16
	Min Size	max_key_size	min_key_size		16
5. Exchange Key type	Keys that local device distributes	loc_key_dist	loc_key_dist	0(Keys are not distributed.)	BLE_GAP_KEY_DIST_ ENCKEY(0x01)
				BLE_GAP_KEY_DIST_ENCKEY(0x01)	
	Keys that local device requests to distribute	rem_key_dist	rem_key_dist	BLE_GAP_KEY_DIST_IDKEY(0x02)	0
				BLE_GAP_KEY_DIST_SIGNKEY(0x04)	
6. Key Press Notification Support		No parameter Fixed to BLE_GAP_ SC_KEY_PRESS_ NTF_NOT_SPRT	key_notf	BLE_GAP_SC_KEY_PRESS_NTF_ NOT_SPRT(0x00)	BLE_GAP_SC_KEY_PRESS_ NTF_NOT_SPRT(0x00)
				BLE_GAP_SC_KEY_PRESS_NTF_SPRT(0x01)	
7. LE Secure Connections Request		sec_conn_only	sec_conn_only	BLE_GAP_SC_BEST_ EFFORT(0x00)	BLE_GAP_SC_BEST_ EFFORT(0x00)
				BLE_GAP_SC_STRICT(0x01)	

1. Input Output capabilities

Table 9.4 shows the input capability (Table 9.2) and the output capability (Table 9.3) that local device supports.

Table 9.2 Input capability

Input capability	Description
No Input	Device cannot indicate “Yes” and “No”.
Yes / No	Device can indicate “Yes” and “No”.
Keyboard	Device can indicate “Yes” and “No” and input numbers 0 through 9.

Table 9.3 Output capability

Output capability	Description
No Output	Device cannot display 6-digit number.
Numeric output	Device can display 6-digit number.

Table 9.4 Input Output capability

		Output	
		No output	Numeric output
Input	No input	NoInputNoOutput BLE_GAP_IOCAP_NOINPUT_NOOUTPUT(0x03)	DisplayOnly BLE_GAP_IOCAP_DISPLAY_ONLY(0x00)
	Yes / No	NoInputNoOutput BLE_GAP_IOCAP_NOINPUT_NOOUTPUT(0x03)	DisplayYesNo BLE_GAP_IOCAP_DISPLAY_YESNO(0x01)
	Keyboard	KeyboardOnly BLE_GAP_IOCAP_KEYBOARD_ONLY(0x02)	KeyboardDisplay BLE_GAP_IOCAP_KEYBOARD_DISPLAY(0x04)

2. MITM(Man-In-The-Middle) protection

Table 9.5 shows settings for the MITM protection request parameter.

Table 9.5 MITM Protection

MITM Protection	Settings
Depending on remote device	BLE_GAP_SEC_MITM_BEST_EFFORT(0x00)
Required	BLE_GAP_SEC_MITM_STRICT(0x01)

Completing pairing with any pairing method other than Just Works according to “9.1.6 Carrying out pairing method” enables the MITM protection. If the MITM Protection is “Required” and the combination of the devices Input Output capabilities results in “Just Works” described in Table 9.13, the pairing is failed.

BP: For LE Secure Connections, it is recommended to design devices that support authenticated pairing using input, output or OOB mechanism to reduce the chances of a MITM obtaining a shared secret key during pairing.

3. Bonding

Table 9.6 shows the bonding parameter settings for whether the local device perform bonding or not. For more details about bonding, refer to “9.2 Bonding”.

Table 9.6 Bonding

Bonding Type	Settings
No bonding	BLE_GAP_BONDING_NONE(0x00)
Bonding	BLE_GAP_BONDING(0x01)

If the application uses R_BLE_ABS_Init, the bonding type is fixed to “Bonding”.

4. Encryption Key Size

Select encryption key size between 7 to 16 bytes. It is recommended that the encryption key size is 16 bytes because a short encryption key size can cause an access rejection to the remote device.

BP: Recommend setting it to maximum entropy (16 octets) to protect encrypted links from brute force attacks.

5. Type of key exchanged by pairing

Table 9.7 shows the type of keys which local device distributes and requests the remote device to distribute in pairing.

Table 9.7 Key Type

Key type	Settings
LTK, EDIV, Rand	BLE_GAP_KEY_DIST_ENCKEY(0x01)
IRK, Identity Address	BLE_GAP_KEY_DIST_IDKEY(0x02)
CSRK	BLE_GAP_KEY_DIST_SIGNKEY(0x04)

6. Key Press Notification support

Key Press Notification is used when Passkey Entry is selected according to “9.1.6 Carrying out pairing method”. If Key Press Notification is supported, the event is notified to the remote device when the local device key is pressed. Specify the feature support with the value in Table 9.8.

Table 9.8 Key Press Notification support

Key Press Notification Support	Value
Not Support	BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT(0x00)
Support	BLE_GAP_SC_KEY_PRESS_NTF_SPRT(0x01)

If the Abstraction API is enabled, the Key Press Notification support is fixed to “Not Support”.

7. LE Secure Connections Requirement

Determine whether pairing is permitted by only LE Secure Connections or not with the parameter in Table 9.9.

Table 9.9 Secure Connections Only Requirement

LE Secure Connections Only Requirement	Value
Depending on the remote device	BLE_GAP_SC_BEST_EFFORT(0x00)
Required	BLE_GAP_SC_STRICT(0x01)

When LE legacy pairing starts with BLE_GAP_SC_STRICT specified, BLE_ERR_SMP_LE_AUTH_REQ_NOT_MET(0x2003) is notified by result in BLE_GAP_EVENT_PAIRING_COMP event.

An example of setting the pairing parameters by R_BLE_GAP_SetPairingParams is shown below.

```
st_ble_gap_pairing_param_t pairing_param = {
    .iocap         = BLE_GAP_IOCAP_NOINPUT_NOOUTPUT,
    .mitm          = BLE_GAP_SEC_MITM_BEST_EFFORT,
    .bonding       = BLE_GAP_BONDING,
    .max_key_size  = 16,
    .min_key_size  = 16,
    .loc_key_dist  = BLE_GAP_KEY_DIST_ENCKEY | BLE_GAP_KEY_DIST_IDKEY,
    .rem_key_dist  = BLE_GAP_KEY_DIST_ENCKEY | BLE_GAP_KEY_DIST_IDKEY,
    .key_notf      = BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT,
    .sec_conn_only = BLE_GAP_SC_BEST_EFFORT,
};

R_BLE_GAP_SetPairingParams(&pairing_param);
```

Code 9-1 An example of setting pairing parameter

9.1.2 Key generation and registration

Generate IRK and CSRK distributed by “9.1.7 Key exchange”. The random number generated by R_BLE_VS_GetRand can be used as IRK or CSRK. The generated keys are registered by the APIs in Table 9.10.

Table 9.10 The APIs used for key generation

Key	API for key generation
IRK, Identity Address	R_BLE_ABS_SetLocPrivacy[*1] or R_BLE_GAP_SetLocIdInfo
CSRK	R_BLE_GAP_SetLocCsrk

[*1] : R_BLE_ABS_SetLocPrivacy generates and registers the local device IRK and uses current Public Address or Static Address as Identity Address.

An example of key generation and registry is shown below.

```
/** some code is omitted  **/
/* IRK generation */
R_BLE_VS_GetRand(0x10);
/** some code is omitted  **/

/* Vendor Specific Callback function */
void vs_cb(uint16_t event_type, ble_status_t result,
           st_ble_vs_evt_data_t * p_event_data)
{
    /** some code is omitted  **/
    case BLE_VS_EVENT_GET_RAND
    {
        st_ble_vs_get_rand_comp_evt_t * p_rand_param;
        p_rand_param = (st_ble_vs_get_rand_comp_evt_t *)p_event_data->p_param;
        /* register local IRK and identity address */
        R_BLE_GAP_SetLocIdInfo(&loc_bd_addr, p_rand_param);
    } break;
    /** some code is omitted  **/
}
```

Code 9-2 An example of key generation and registry

If the application does not use RPA (Resolvable Private Address), it does not need to generate and register the local device IRK. If the application does not sends/receives with the signed data, it does not need to generate and register the local device CSRK. The local device LTK (case of LE legacy pairing includes EDIV, Rand) is generated by the protocol stack as needed, so there is no need to generate and register on the application before start pairing.

BP: EDIV is included in the data exchanged between devices paired with LE legacy pairing. EDIV is unique to a particular pair of devices, allowing tracking of paired devices using EDIV when using private addresses. It is recommended to periodically establish new pairings/bonds between devices to update the EDIV to prevent long-term tracking.

9.1.3 OOB (Out of Band) data transmission and reception

If local device and remote device have a common means of communications except Bluetooth (OOB) , the data for pairing can be transmitted and received through OOB. The data consists of confirm value (16 bytes) and random value (16 bytes). It needs to meet the condition in Table 9.11 to do pairing by OOB. If OOB is available, the data is transmitted and received before starting pairing.

Table 9.11 The condition to do pairing by OOB

Pairing mechanism	Condition
LE Secure Connections	The one device can transmit the data for pairing by OOB and the other can receive it.
LE legacy pairing	Both devices can transmit and receive the data for pairing by OOB.

When pairing data is received from the remote device by OOB, register the remote device address and received data with R_BLE_GAP_SetRemOobData. This informs the remote device that OOB was able to receive the data when exchanging the pairing parameters.

If the local device sends data by OOB, call R_BLE_GAP_CreateScOobData. This API generates confirm value (16 bytes) and random value (16 bytes) according to SMP specifications. When data generation is complete, the BLE_GAP_EVENT_SC_OOB_CREATE_COMP event is notified. Send the generated data in OOB to the remote device.

BP: The TK (Temporary Key) exchanged in OOB has a maximum entropy of 128 bits and is the most resistant to MITM attacks. Support for the OOB mechanism is recommended if LE legacy pairing needs to be supported.

9.1.4 Pairing request

Call the below APIs to request to start pairing from local device.

R_BLE_ABS_StartAuth
R_BLE_GAP_StartPairing

The APIs can be called from both a Central and a Peripheral.

9.1.5 Response to pairing request

If a pairing request is received from a remote device, BLE_GAP_EVENT_PAIRING_REQ event is notified. Respond to the request event by R_BLE_GAP_ReplyPairing.

An example of a response to a pairing request is shown as below.

```
/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_event_data)
{
    /** some code is omitted  **/
    case BLE_GAP_EVENT_PAIRING_REQ :
    {
        st_ble_gap_pairing_info_evt_t * p_param;
        p_param = (st_ble_gap_pairing_info_evt_t *)p_event_data->p_param;
        R_BLE_GAP_ReplyPairing(p_param->conn_hdl, BLE_GAP_PAIRING_ACCEPT);
    }
    break;
    /** some code is omitted  **/
```

Code 9-3 Response to a pairing request

If the Abstraction API is enabled, when receiving BLE_GAP_EVENT_PAIRING_REQ event, call R_BLE_GAP_ReplyPairing to automatically respond to a pairing request.

9.1.6 Carrying out pairing method

By starting pairing or responding to pairing request, local device, and the remote device exchange pairing parameters. After exchanging the parameters, both devices select a pairing method from Table 9.12 and perform the pairing method.

Table 9.12 Pairing Method

Pairing Method	Description	MITM Protection
OOB	The application does not need to manage pairing, because the Bluetooth LE Protocol Stack processes the OOB data previously received/transmitted.	Enable
Passkey Entry	The one device displays a 6-digit number, the other inputs the number.	Enable
Numeric Comparison	Both devices display a 6-digit number. Check if two numbers are same.	Enable
Just Works	The application does not need to manage pairing, because it is automatically performed.	Disable

According to 1-3, the pairing method is determined.

1. If the OOB data is received/transmitted before pairing, the OOB pairing method is selected.
2. If the OOB data is not received/transmitted and both devices do not require the MITM protection, the Just Works pairing method is selected.
3. If the OOB data is not received/transmitted and which device requires the MITM protection, the pairing method is determined according to Table 9.13.

Table 9.13 Pairing Method Selection

Peripheral	Central				
	DisplayOnly	DisplayYesNo	KeyboardOnly	NoInputNoOutput	KeyboardDisplay
DisplayOnly	Just Works	Just Works	Passkey Entry	Just Works	Passkey Entry
DisplayYesNo	Just Works	Just Works (LE legacy pairing)	Passkey Entry	Just Works	Passkey Entry (LE legacy pairing)
		Numeric Comparison (LE Secure Connections)			Numeric Comparison (LE Secure Connections)
KeyboardOnly	Passkey Entry	Passkey Entry	Passkey Entry	Just Works	Passkey Entry
NoInputNoOutput	Just Works	Just Works	Just Works	Just Works	Just Works
KeyboardDisplay	Passkey Entry	Passkey Entry (LE legacy pairing)	Passkey Entry	Just Works	Passkey Entry (LE legacy pairing)
		Numeric Comparison (LE Secure Connections)			Numeric Comparison (LE Secure Connections)

The pairing events and the API used for the response, depend on from the selected pairing method.

- Just Works, OOB
  No events are notified to an application. It is not necessary to respond with APIs.

- Passkey Entry
  [Input device]
  BLE_GAP_EVENT_PASSKEY_ENTRY_REQ event which requires to input 6-digit number is notified to an application. If the application receives the event and the remote device displays a 6-digit number, the application inputs the number by R_BLE_GAP_ReplyPasskeyEntry. By input “gap auth passkey xxxxxx(6-digit passkey)”, the command line feature calls R_BLE_GAP_ReplyPasskeyEntry to respond to BLE_GAP_EVENT_PASSKEY_ENTRY_REQ event.
  If the Key Press Notification support is ON(Table 9.8), the type of the input keys is notified to the remote device.

  [Display device]
  BLE_GAP_EVENT_PASSKEY_DISPLAY_REQ event which requires to display 6-digit number is notified to an application. If the application receives the event, display the number. When the command line is enabled, the 6-digit number is shown. If remote device supports the Key Press Notification feature, the input key information is notified to the application with BLE_GAP_EVENT_KEY_PRESS_NTF event. After the remote device has completed to input the keys, continue to the next section.

- Numeric Comparison
  BLE_GAP_EVENT_NUM_COMP_REQ event which requires to check whether the number displayed on both devices are same. If the application receives the event, display the number. After checking the number displayed on the remote device, send the result by R_BLE_GAP_ReplyNumComp.

BP: When using except for OOB pairing methods, having the UX/UI inform the end-user that there are certain security or privacy risks is beneficial in mitigating security or privacy risks.

9.1.7 Key exchange

After the completion of the pairing method, both devices exchange keys. The link with the remote device is encrypted before key exchange and the completion is notified by BLE_GAP_EVENT_ENC_CHG event.

When the keys are distributed from the remote device, BLE_GAP_EVENT_PEER_KEY_INFO event is notified. Refer to “9.2.1 Store remote device keys ” for storing the keys received in the event.

When the local device is required to distribute the keys, BLE_GAP_EVENT_EX_KEY_REQ event is notified. The local device responds to the request with R_BLE_GAP_ReplyExKeyInfoReq. An example of the response to the key distribution request is shown below.

```
/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_event_data)
{
    /** some code is omitted  **/
    case BLE_GAP_EVENT_EX_KEY_REQ :
        {
            st_ble_gap_conn_hdl_evt_t * p_param;
            p_param = (st_ble_gap_conn_hdl_evt_t *)p_event_data->p_param;
            R_BLE_GAP_ReplyExKeyInfoReq(p_param->conn_hdl);
        }
    break;
    /** some code is omitted  **/
```

Code 9-4 Sample of responding to a key distribute request

If the Abstraction API is enabled, when BLE_GAP_EVENT_EX_KEY_REQ is notified, call R_BLE_GAP_ReplyExKeyInfoReq to automatically respond to the key distribution request.

9.1.8 Completion of pairing

When the pairing has been completed, the BLE_GAP_EVENT_PAIRING_COMP event is notified. If the pairing is successful, the event result is BLE_SUCCESS(0x0000). Any other value indicates a pairing failure.

If pairing is not completed within 30 seconds,
result: BLE_ERR_SMP_LE_TO(0x2011)
will be notified. Try pairing again.

If the bonding information about the remote device is lost, but the information remains in the Resolving List,
result: BLE_ERR_SMP_LE_DHKEY_CHECK_FAIL(0x200B)
will be notified. Delete the information about that device from the Resolving List as well using R_BLE_GAP_ConfRslvList().

If the bonding information was lost and the encryption could not be requested,
result: BLE_ERR_SMP_LE_LOC_KEY_MISSING(0x2014)
will be notified. Refer to "9.3.1 Request Encryption".

BP: If the pairing procedure fails, a waiting interval must elapse before initiating the next pairing with the same remote address. The wait interval increases exponentially with each repeated pairing procedure failure (maximum wait interval is implementation dependent). Introducing a wait interval reduces the ability of an attacker to repeatedly attempt the pairing procedure using different keys.

9.2 Bonding

The bonding process stores the keys exchanged during pairing. Because of bonding, pairing does not need to be done in reconnecting a paired device. Figure 9-2 shows the procedure of bonding and reset the keys to the Bluetooth LE Protocol Stack.

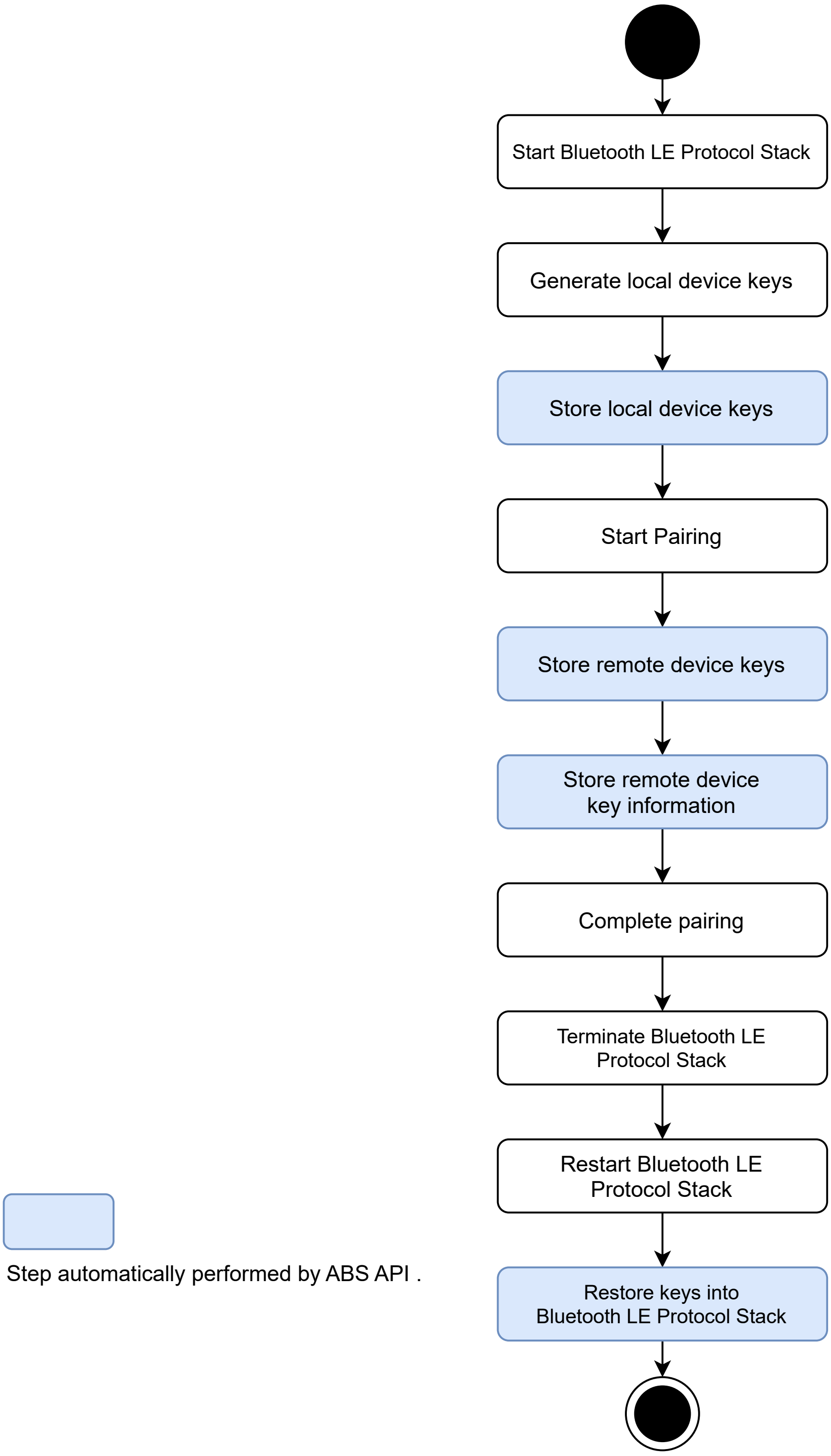

Start Bluetooth LE Protocol Stack
Generate local device keys
Store local device keys
Start Pairing
Store remote device keys
Store remote device key information
Complete pairing
Terminate Bluetooth LE Protocol Stack
Restart Bluetooth LE Protocol Stack
Restore keys into Bluetooth LE Protocol Stack
Step automatically performed by ABS API .

Figure 9-2 Bonding procedure

9.2.1 Store remote device keys

Store remote device keys and key information received by the following events in the Data Flash.

BLE_GAP_EVENT_PEER_KEY_INFO (key)

BLE_GAP_EVENT_PAIRING_COMP (key information)

An example of storing remote device keys is shown in below.

```
case BLE_GAP_EVENT_PAIRING_COMP :
    {
        if(BLE_SUCCESS == event_result)
        {
            st_ble_gap_pairing_info_evt_t * p_param;
            p_param = (st_ble_gap_pairing_info_evt_t *)p_event_data->p_param;
            /* Add code storing p_param->auth_info into the Data Flash. */
        }
    }
    break;

case BLE_GAP_EVENT_PEER_KEY_INFO :
    {
        st_ble_gap_peer_key_info_evt_t * p_param;
        p_param = (st_ble_gap_peer_key_info_evt_t *)p_event_data->p_param;
        /* Add code storing p_param->key_ex_param into the Data Flash. */
    }
    break;
```

Code 9-5 Sample of storing received keys

If you want to resolve the bonded remote device address, you have to also register the remote device IRK and Identity Address to Resolving List.

If the Abstraction API and the security data management are enabled, the keys received by BLE_GAP_EVENT_PEER_KEY_INFO event and the key information received by BLE_GAP_EVENT_PAIRING_COMP event are automatically stored.

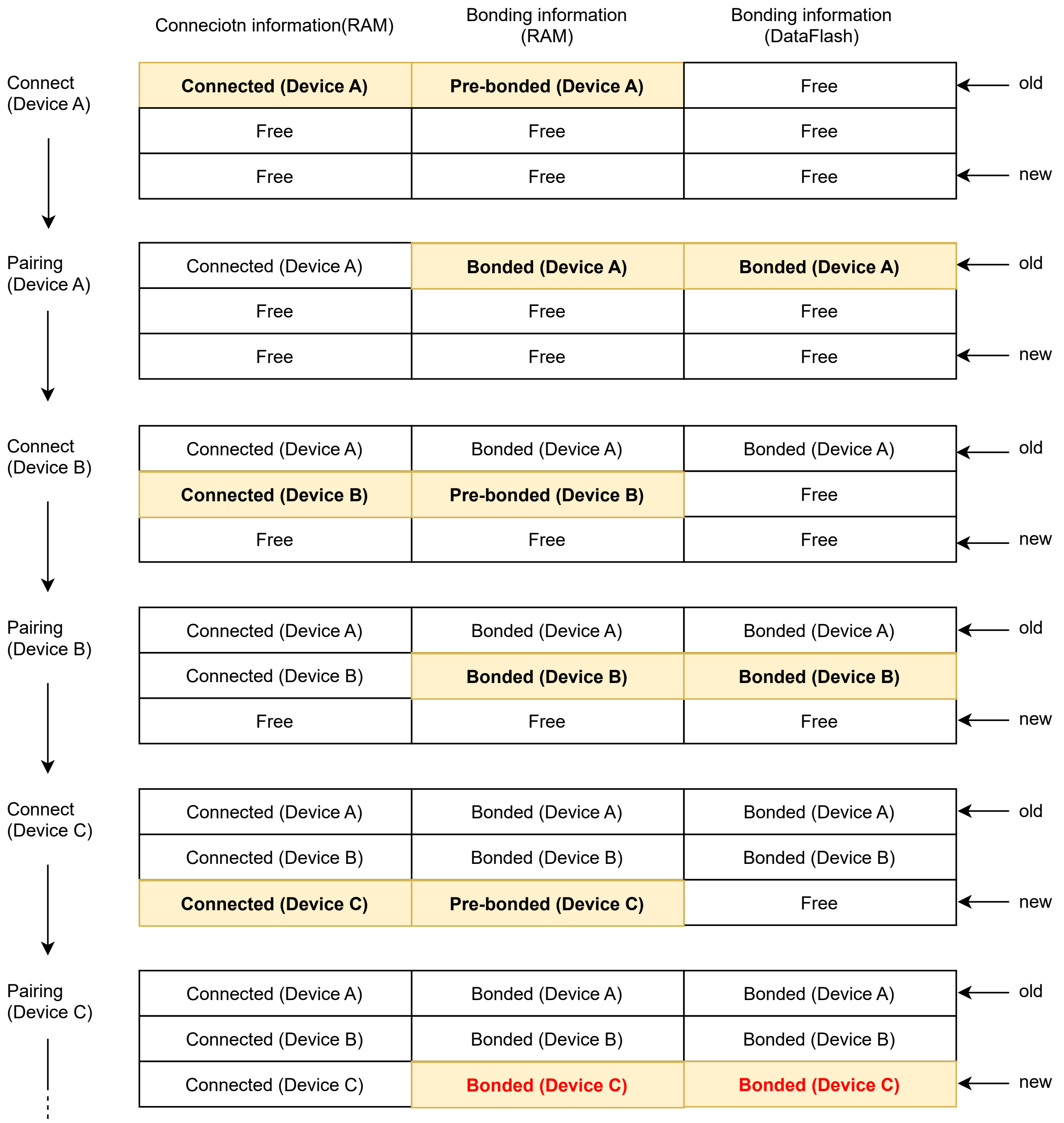
If the Abstraction API and the security data management are disabled, the keys and the key information are not stored automatically.

Information (remote device keys) are stored in RAM and DataFlash. Bonding information can be stored in RAM up to BLE_CFG_RF_CONN_MAX and stored in DataFlash up to BLE_CFG_NUM_BOND. A bonding information is stored in order of bonding and if a new one is stored over the upper limit, the stack library deletes the oldest one and stores the new one according to the following policies.

- In RAM, the oldest bonding information of which the device is not connected is first automatically deleted.
- In DataFlash, the oldest bonding information is first automatically deleted regardless of the connection state.

If you do not want to automatically delete the bonding information, you have to check a bonding except for desired device and do not allow to bond beyond the desired bonding number in your application. For example, if you want to connect with only bonded devices, the bonding information control with White List such as “9.2.5 Filtering remote devices after bonding” is effective. It is recommended that BLE_CFG_RF_CONN_MAX and BLE_CFG_NUM_BOND are set to the same number and the desired bonding number + 1.

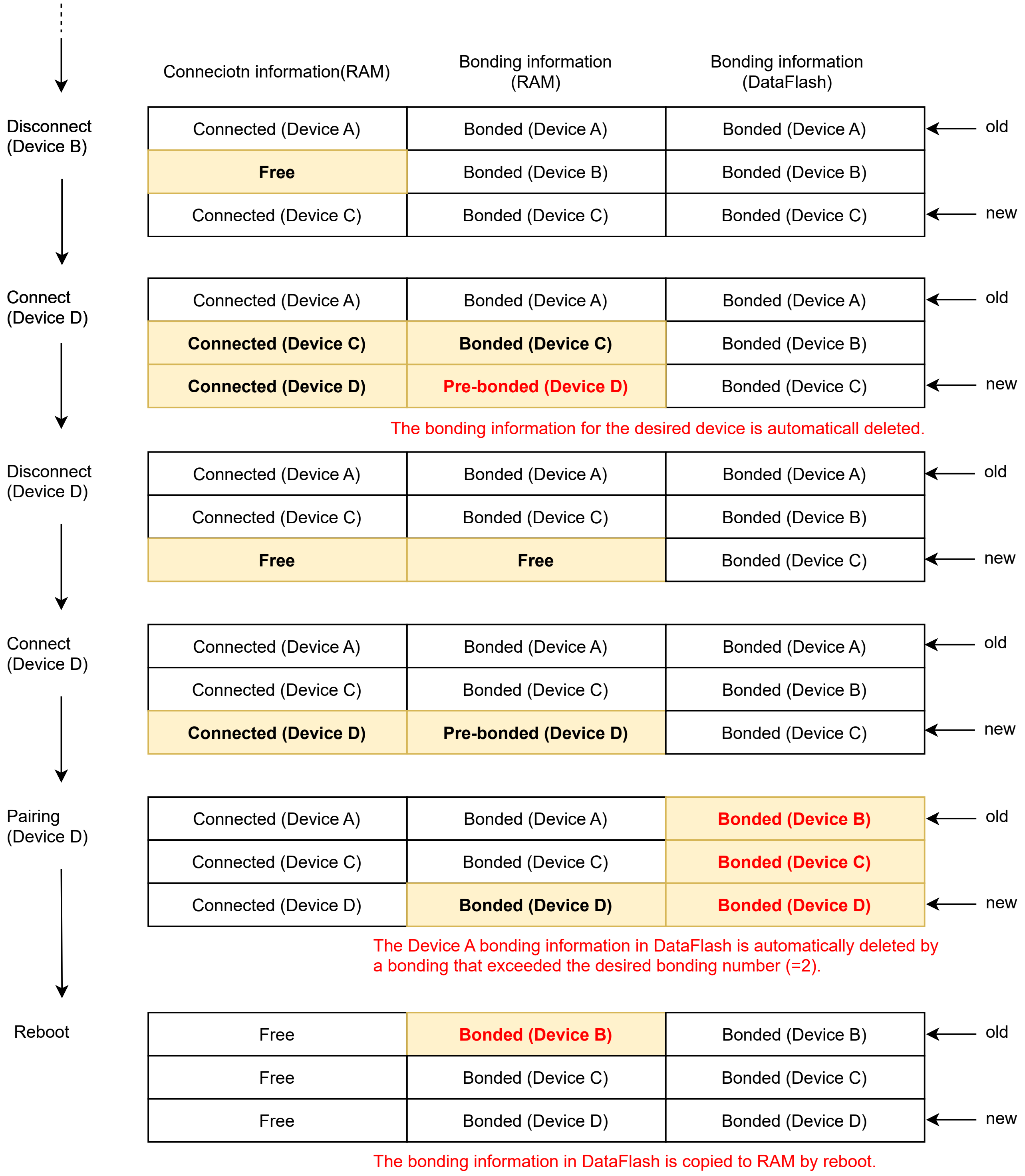
An example that the desired bonding number is 2 (BLE_CFG_RF_CONN_MAX=3, BLE_CFG_NUM_BOND=3) is shown in Figure **9-3** and Figure **9-4**. The changes are shown in bold text.



	Conneciotn information(RAM)	Bonding information (RAM)	Bonding information (DataFlash)	
Connect (Device A)	**Connected (Device A)**	**Pre-bonded (Device A)**	Free	old
	Free	Free	Free	
	Free	Free	Free	new
Pairing (Device A)	Connected (Device A)	**Bonded (Device A)**	**Bonded (Device A)**	old
	Free	Free	Free	
	Free	Free	Free	new
Connect (Device B)	Connected (Device A)	Bonded (Device A)	Bonded (Device A)	old
	Connected (Device B)	**Pre-bonded (Device B)**	Free	
	Free	Free	Free	new
Pairing (Device B)	Connected (Device A)	Bonded (Device A)	Bonded (Device A)	old
	Connected (Device B)	**Bonded (Device B)**	**Bonded (Device B)**	
	Free	Free	Free	new
Connect (Device C)	Connected (Device A)	Bonded (Device A)	Bonded (Device A)	old
	Connected (Device B)	Bonded (Device B)	Bonded (Device B)	
	Connected (Device C)	**Pre-bonded (Device C)**	Free	new
Pairing (Device C)	Connected (Device A)	Bonded (Device A)	Bonded (Device A)	old
	Connected (Device B)	Bonded (Device B)	Bonded (Device B)	
	Connected (Device C)	**Bonded (Device C)**	**Bonded (Device C)**	new

The stored area is full by a bonding that exceeded the desired bonding number (=2).

Figure 9-3 Bonding information management in RAM, DataFlash (1)



Disconnect (Device B)

Conneciotn information(RAM)	Bonding information (RAM)	Bonding information (DataFlash)	
Connected (Device A)	Bonded (Device A)	Bonded (Device A)	old
Free	Bonded (Device B)	Bonded (Device B)	
Connected (Device C)	Bonded (Device C)	Bonded (Device C)	new

Connect (Device D)

Connected (Device A)	Bonded (Device A)	Bonded (Device A)	old
Connected (Device C)	**Bonded (Device C)**	Bonded (Device B)	
Connected (Device D)	**Pre-bonded (Device D)**	Bonded (Device C)	new

The bonding information for the desired device is automaticall deleted.

Disconnect (Device D)

Connected (Device A)	Bonded (Device A)	Bonded (Device A)	old
Connected (Device C)	Bonded (Device C)	Bonded (Device B)	
Free	**Free**	Bonded (Device C)	new

Connect (Device D)

Connected (Device A)	Bonded (Device A)	Bonded (Device A)	old
Connected (Device C)	Bonded (Device C)	Bonded (Device B)	
Connected (Device D)	**Pre-bonded (Device D)**	Bonded (Device C)	new

Pairing (Device D)

Connected (Device A)	Bonded (Device A)	**Bonded (Device B)**	old
Connected (Device C)	Bonded (Device C)	**Bonded (Device C)**	
Connected (Device D)	**Bonded (Device D)**	**Bonded (Device D)**	new

The Device A bonding information in DataFlash is automatically deleted by a bonding that exceeded the desired bonding number (=2).

Reboot

Free	**Bonded (Device B)**	Bonded (Device B)	old
Free	Bonded (Device C)	Bonded (Device C)	
Free	Bonded (Device D)	Bonded (Device D)	new

The bonding information in DataFlash is copied to RAM by reboot.

Figure 9-4 Bonding information management in RAM, DataFlash (2)

9.2.1.1 Bonding information in RAM

Because bonding information in RAM is managed in the same area as the connection information, even if pairing or bonding is not done, the oldest bonding information of which the device is not connected is automatically deleted, and the area is released.

For more information about how to reconfigure the deleted bonding information from RAM, see section “9.2.3 Reset the stored keys”.

9.2.1.2 Bonding information in DataFlash

If a bonding information is stored in DataFlash exceeding BLE_CFG_NUM_BOND, the oldest bonding information is automatically deleted regardless of the connection status and the new one is overwritten as shown in Figure 9-3,Figure 9-4.

If you do not want to automatically delete the bonding information in DataFlash, your application has to monitor the bonding number not to exceed BLE_CFG_NUM_BOND by deleting the bonding information first and then doing pairing or bonding and so on. For more information about how to delete the bonding information, see section “9.2.4 Delete the stored keys”.

9.2.2 Store local device keys

If the local device uses the privacy feature, the IRK and the Identity Address are registered by R_BLE_GAP_SetLocIdInfo or R_BLE_ABS_SetLocPrivacy need to be stored.

If the local device sends/receives signed data packets, the CSRK registered by R_BLE_GAP_SetLocCsrk needs to be stored.

When the security data management configuration option is enabled, R_BLE_SECD_WriteLocInfo described in “4.4.3 Store the local device keys” can store the local device IRK and CSRK in the Data Flash.

When the Abstraction API and security data management configuration options are enabled, the local device IRK generated by R_BLE_ABS_SetLocPrivacy is automatically stored in the Data Flash with R_BLE_SECD_WriteLocInfo.

9.2.3 Reset the stored keys

When the Bluetooth LE Protocol Stack restarts, the stored keys in the device need to be reset to the stack by R_BLE_GAP_SetBondInfo.

If the Abstraction API and security data management configuration options are enabled, the stored keys are automatically reset to the Bluetooth LE Protocol Stack in restarting.

If you want to resolve the bonded remote device address, you must also reconfigure the remote device IRK and Identity Address in Resolving List.

9.2.4 Delete the stored keys

When the bonding information in the remote device deleted, delete the one in the local device by specifying the remote device address as the second parameter of R_BLE_GAP_DeleteBondInfo.

Likewise, when the bonding information (the remote device security data) in the local device deleted, it is also necessary to delete the bonding information in the remote device.

If only one of the devices deleted the bonding information, the following issues will occur, and the security feature cannot be used.

- The device cannot access to the GATT service that the encryption security requirement is configured to due to the loss of LTK.
- The device cannot resolve the remote device address and cannot connect to the remote device Identity Address due to the loss of IRK.

The bonding information that needs to be deleted is shown in Figure **9-5** if ether a RX23W device or a remote device deletes the bonding information.

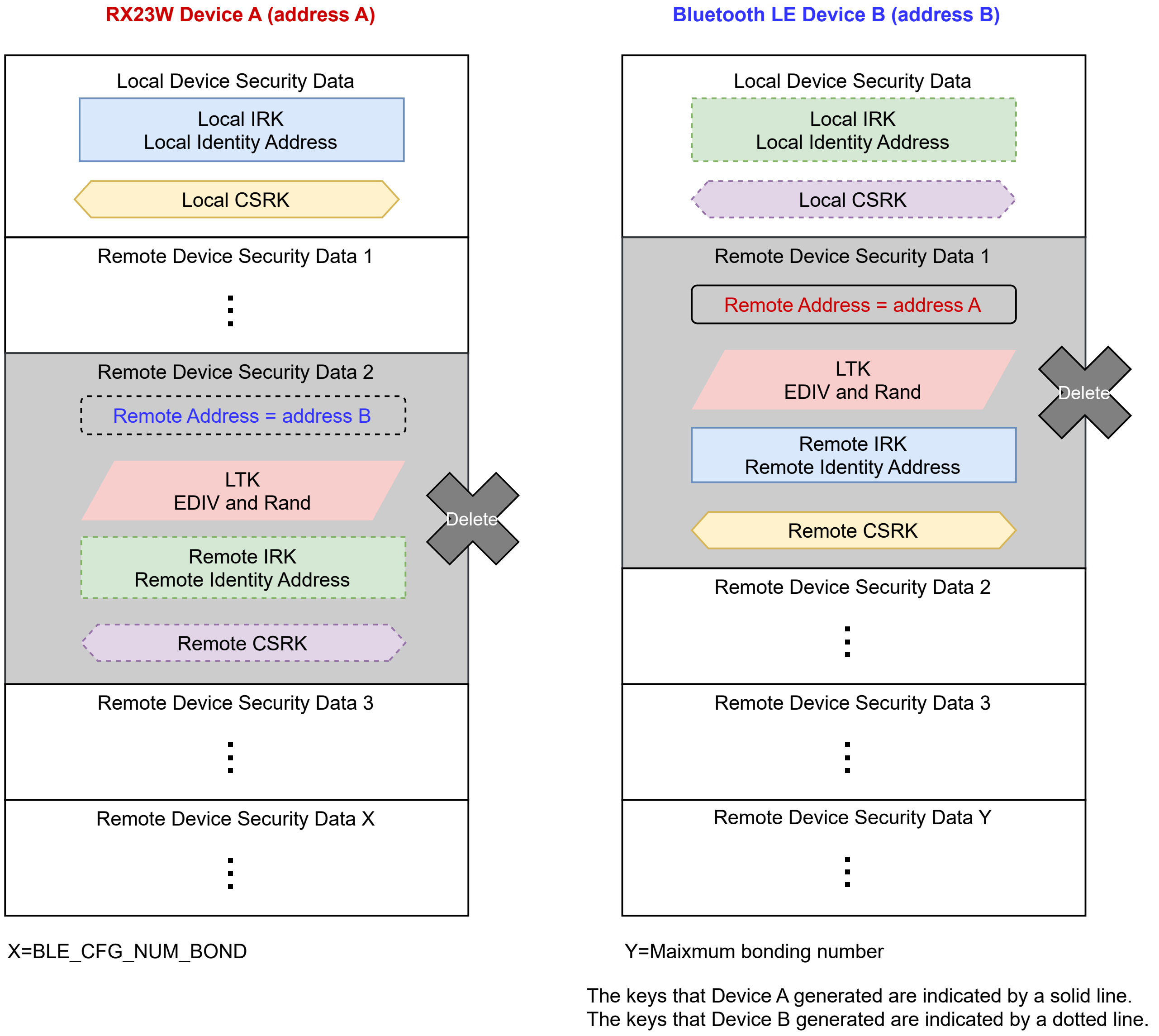

RX23W Device A (address A)
Local Device Security Data
Local IRK
Local Identity Address
Local CSRK
Remote Device Security Data 1
Remote Device Security Data 2
Remote Address = address B
LTK
EDIV and Rand
Remote IRK
Remote Identity Address
Remote CSRK
Delete
Remote Device Security Data 3
Remote Device Security Data X
X=BLE_CFG_NUM_BOND
Bluetooth LE Device B (address B)
Local Device Security Data
Local IRK
Local Identity Address
Local CSRK
Remote Device Security Data 1
Remote Address = address A
LTK
EDIV and Rand
Remote IRK
Remote Identity Address
Remote CSRK
Delete
Remote Device Security Data 2
Remote Device Security Data 3
Remote Device Security Data Y
Y=Maixmum bonding number
The keys that Device A generated are indicated by a solid line.
The keys that Device B generated are indicated by a dotted line.

Figure 9-5 The bonding information that needs to be deleted if either device deletes it.

If the bonded remote device address was resolved, you must also delete the device IRK and Identity Address from Resolving List.

9.2.5 Filtering remote devices after bonding

If you want to connect or communicate to the bonded device, you should register the remote device address to the White List. White List can register 4 devices in case of All features library and register 8 devices in case of Balance and Compact library. For more information about the use of White List, see 5.2.1.2 and 5.5.1 for Peripheral and 6.4.1 and 7.1.1 for Central.

When the RX23W device reboots, it is necessary to reconfigure the remote device addresses to White List.

If the local device deletes the stored key, delete the remote device address from White List.

9.3 Encryption

Bluetooth LE enables secure communication by encrypting data packets. The encryption in reconnection after pairing uses the key exchanged by pairing.

BP: If the encryption procedures fails, do not attempt to circumvent the failure or connect by other means. Switching to less secure options for convenience is the desired outcome for attackers. Failures at any stage should be aware of the potential for attackers to gain access through vulnerabilities or repeated attempts.

9.3.1 Request Encryption

After pairing and bonding, call the one of the following APIs to request encryption when the local device reconnects with the remote device. A peripheral sends Security Request packet and a central sends LL_ENC_REQ packet to request the link encryption.

R_BLE_ABS_StartAuth
R_BLE_GAP_StartEnc

If the encryption has been completed successfully,
BLE_GAP_EVENT_ENC_CHG
Result: BLE_SUCCESS (0x0000)
event will be notified.

Depending on the remote device implementation, the remote device does not respond to an encryption request from a peripheral device. In this case, if the above API is called, pairing may start.

The encryption request sequence is shown below.

(1) Encryption request from local device(Central)

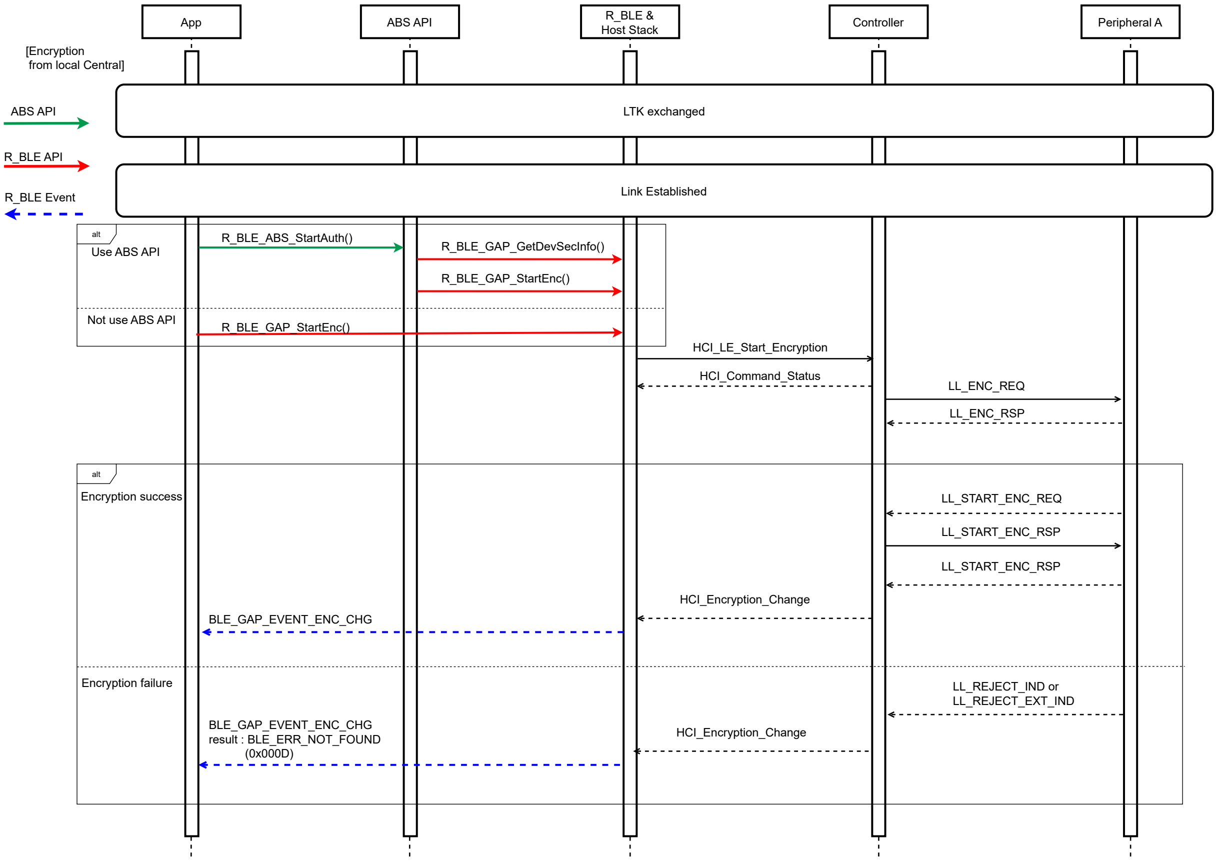


Figure 9-6 Sequence of encryption request from local device(Central)

If the remote device (Peripheral) lost the pair configuration (bonding information) before the local device (Central) sends an encryption request,
BLE_GAP_EVENT_ENC_CHG
Result: BLE_SUCCESS (0x000D)
event will be notified, and the encryption will fail. Although the link is still established, the device (Central) cannot access the service for which the encryption security requirement has been configured.
In this case, the local device (Central) also needs to delete the bonding information and perform pairing procedures again to access the service.

If the local device (Central) lost the bonding information before sending an encryption request, the local device (Central) will send a pairing request and then start encryption.

(2) Encryption request from local device(Peripheral)

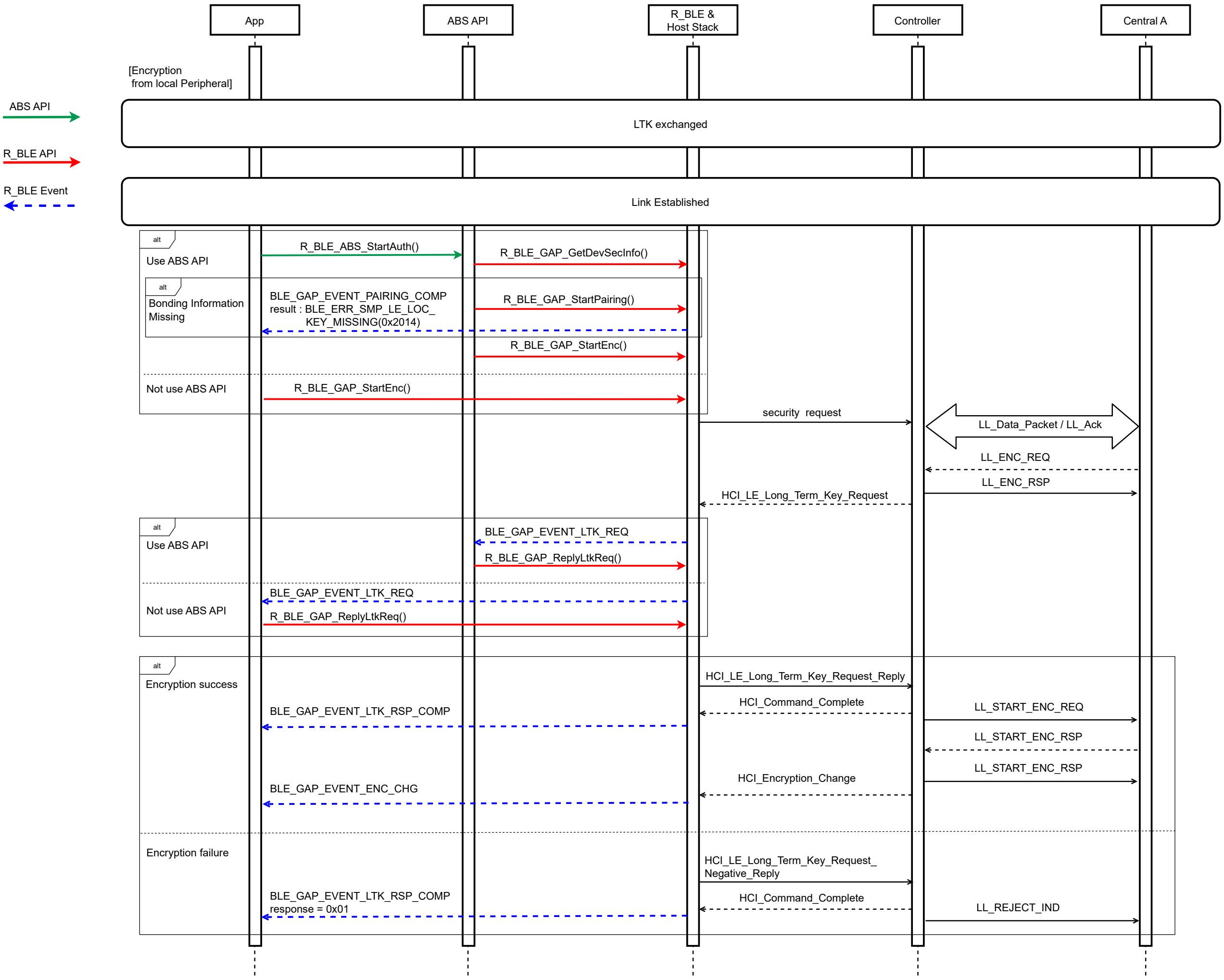


Figure 9-7 Sequence of encryption request from local device(Peripheral)

If the local device (Peripheral) lost the bonding information before sending an encryption request,
BLE_GAP_EVENT_PAIRING_COMP
Result: BLE_ERR_SMP_LE_LOC_KEY_MISSING(0x2014)
event will be notified, and the encryption will fail. Although the link is still established, the device (Central) cannot access the service for which the encryption security requirement has been configured.

In this case, the remote device (Central) also needs to delete the bonding information and perform pairing procedures again to access the service.

If the remote device (Central) lost the bonding information before the local device (Peripheral) sends an encryption request, the remote device (Central) will send a pairing request and then start encryption.

9.3.2 Respond to an encryption request

Response to encryption request differs depending on the role.
If you use the Abstraction API, it automatically replies to the remote device.
The response to encryption request sequence is shown below.

(1) Response to an encryption request from remote device(Central)

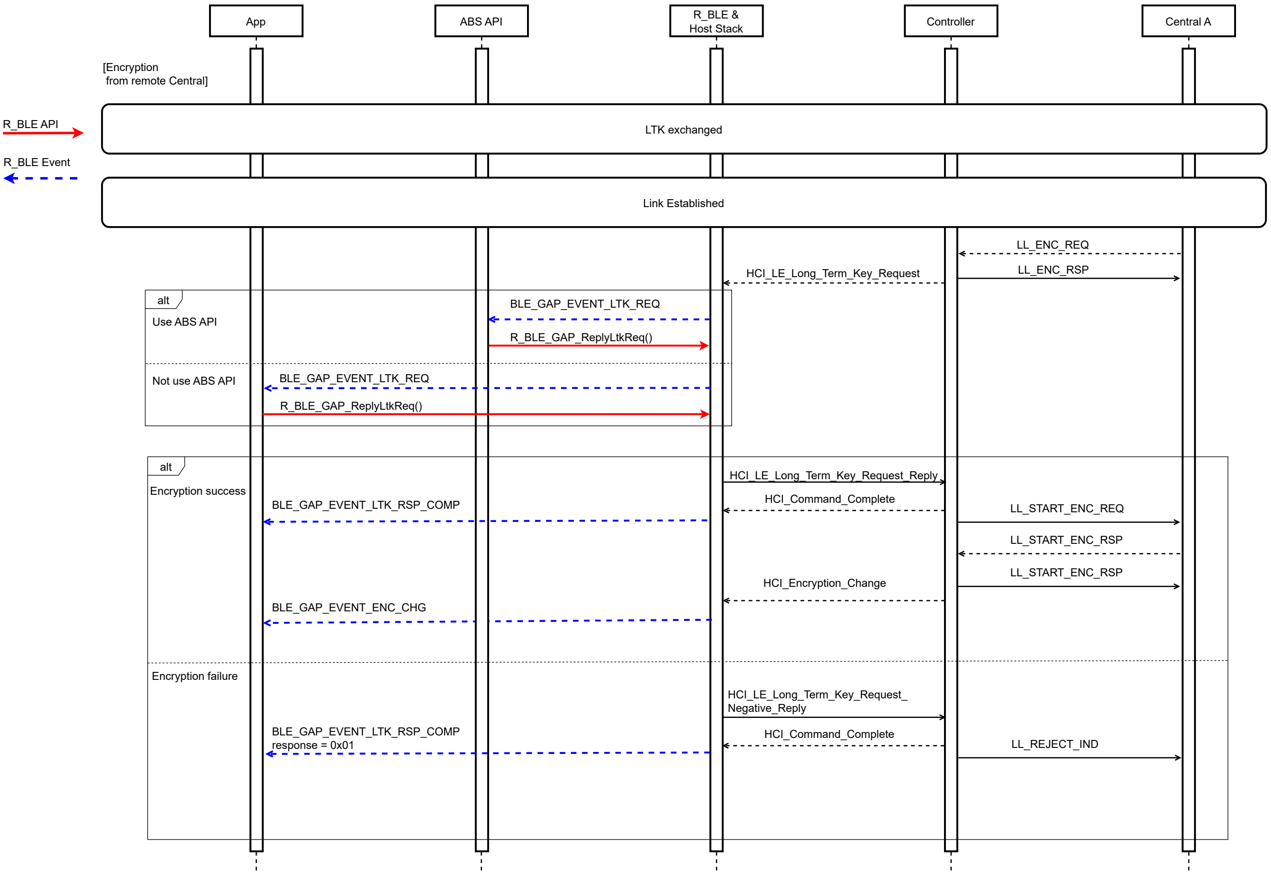


Figure 9-8 Sequence of response to an encryption request from remote device(Central)

When receiving an encryption request from a remote device, BLE_GAP_EVENT_LTK_REQ event will be notified. Local device (Peripheral) needs to reply to the encryption request by using R_BLE_GAP_ReplyLtkReq API with the parameter of BLE_GAP_EVENT_LTK_REQ event as an argument. When the LTK exchange is successfully completed, BLE_GAP_EVENT_LTK_RSP_COMP event will be notified.
If the encryption has been completed successfully,
BLE_GAP_EVENT_ENC_CHG
Result: BLE_SUCCESS (0x0000)
event will be notified.

If the local device (Peripheral) lost the bonding information before the remote device (Central) sends an encryption request and then the local device (Peripheral) calls R_BLE_GAP_ReplyLtkReq,
BLE_GAP_EVENT_LTK_RSP_COMP
response (Event data): 0x01
event will be notified, and the encryption will fail. Although the link is still established, the device (Central) cannot access the service for which the encryption security requirement has been configured.

In this case, the remote device (Central) also needs to delete the bonding information and perform pairing procedures again to access the service.

If the remote device (Central) lost the bonding information before sending an encryption request, the remote device (Central) will send a pairing request and then start encryption.
When the local device (Peripheral) connects to a smart phone (Central) for the first time, the local device is required to respond to a pairing request but is not required to respond to an encryption request. When the local device (Peripheral) connects to the paired smart phone, the local device (Peripheral) is required to respond to an encryption request.

An example of an encryption request from the remote device (Central) event and the response API is shown below.

```
/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result,
            st_ble_evt_data_t * p_event_data)
{
    /** some code is omitted **/
    /* Receive encryption request from a remote device */
    case BLE_GAP_EVENT_LTK_REQ :
        {
            st_ble_gap_ltk_req_evt_t * p_param;
            p_param = (st_ble_gap_ltk_req_evt_t *)p_event_data->p_param;
            R_BLE_GAP_ReplyLtkReq(p_param->conn_hdl, p_param->ediv,
                                  p_param->p_peer_rand, BLE_GAP_LTK_REQ_ACCEPT);
        }
        break;
        /** some code is omitted **/
```

Code 9-6 Sample of responding an encryption request from the remote device(Central) in the event

(2) Response to an encryption request from remote device(Peripheral)

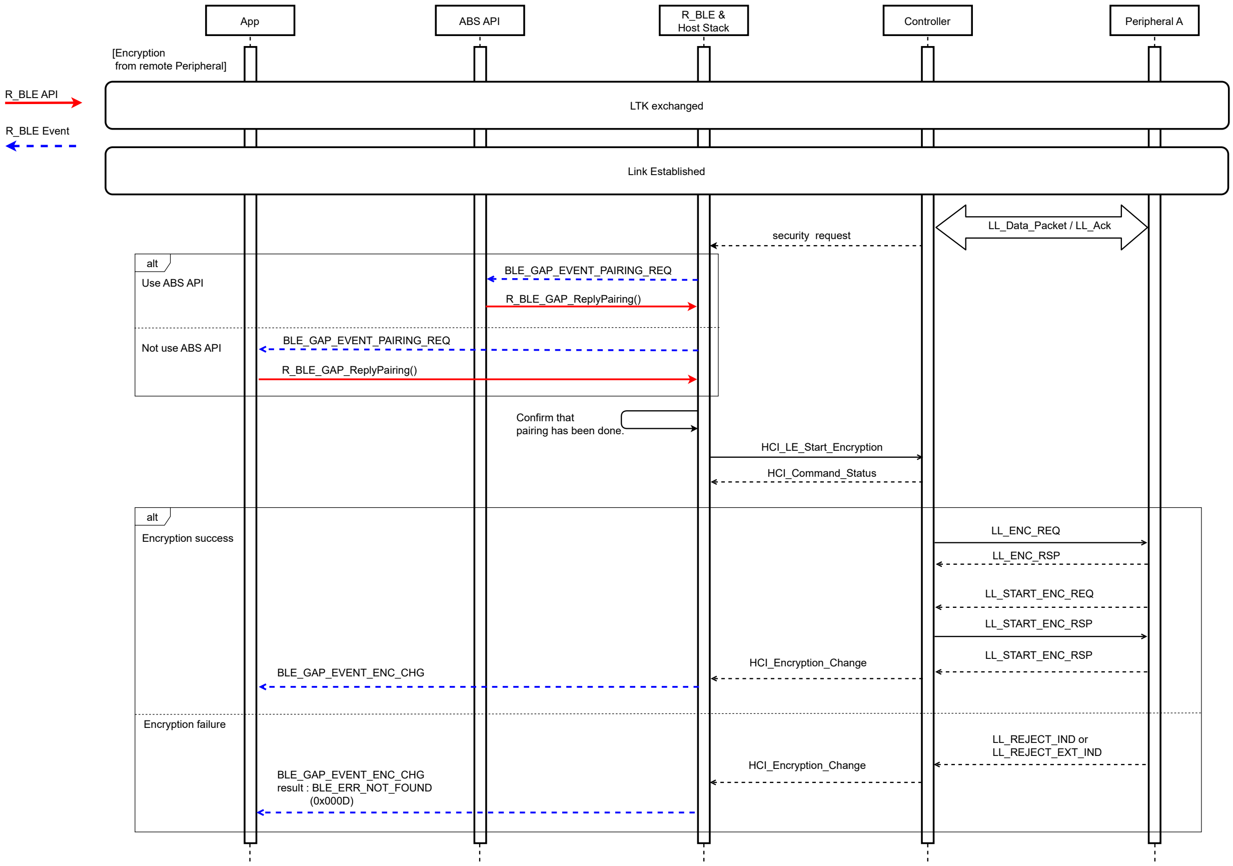


Figure 9-9 Sequence of response to an encryption request from remote device(Peripheral)

When receiving an encryption request from a remote device (Peripheral),
BLE_GAP_EVENT_PAIRING_REQ event will be notified. Local device (Central) need to reply the encryption request by using R_BLE_GAP_ReplyPairing API with the parameter of BLE_GAP_EVENT_PAIRING_REQ event as an argument. If the bonding has been done, this API responds to the encryption request.
If the encryption has been completed successfully,
BLE_GAP_EVENT_ENC_CHG
Result: BLE_SUCCESS (0x0000)
event will be notified.

If the remote device (Peripheral) lost the bonding information before sending an encryption request,
BLE_GAP_EVENT_ENC_CHG
Result: BLE_SUCCESS (0x000D)
event will be notified, and the encryption will fail. Although the link is still established, the device (Central) cannot access the service for which the encryption security requirement has been configured.
In this case, the local device (Central) also needs to delete the bonding information and perform pairing procedures again to access the service.

If the local device (Central) lost the bonding information before a remote device (Peripheral) sends an encryption request, the local device (Central) sends a pairing request and then start encryption.

An example of an encryption request from a remote device (Peripheral) event and the response API is the same as Code 9-3.

9.4 Privacy

The privacy feature allows the local device to periodically change the address used by Advertising, Scan Request and Connection Request to another address to avoid being tracked by other devices. There are two privacy mode: Network Privacy Mode and Device Privacy Mode. In Device Privacy Mode, if the local device uses RPA (Resolvable Private Address), the local device will accept Advertising, Scan Request and Connection Request regardless of the remote address type. In Network Privacy Mode, if the local device uses RPA, the local device will not accept Advertising, Scan Request and Connection Request including identity address of the remote device. By default, the local device is Network Privacy Mode. RPA is generated and resolved by Resolving List in the local device.

Up to 8 sets of the IRK (Remote IRK) and Identity Address (ID) of the remote device and the IRK (Local IRK) of the local device can be registered in the Resolving List.

If the local device generates an RPA to initiate Advertising, scanning, or connection, the Local IRK and the ID of the remote device are registered to the Resolving List in advance, and the Resolving List is searched by the specified ID of the remote device. Details on how to generate RPA are given in 9.4.1.

When resolving RPA included in Advertising, Scan Request, and Connection Request from a remote device, you need to register the Remote IRK and ID obtained by pairing procedure with the remote device together with the Local IRK to the Resolving List. The local device will search a set which the received RPA match the RPA calculated from the IRK and ID of the Resolving List. Details on how to resolve RPA are given in 9.4.2.

Figure 9-10 shows an image of generating and resolving RPA using the Resolving List in Advertising.

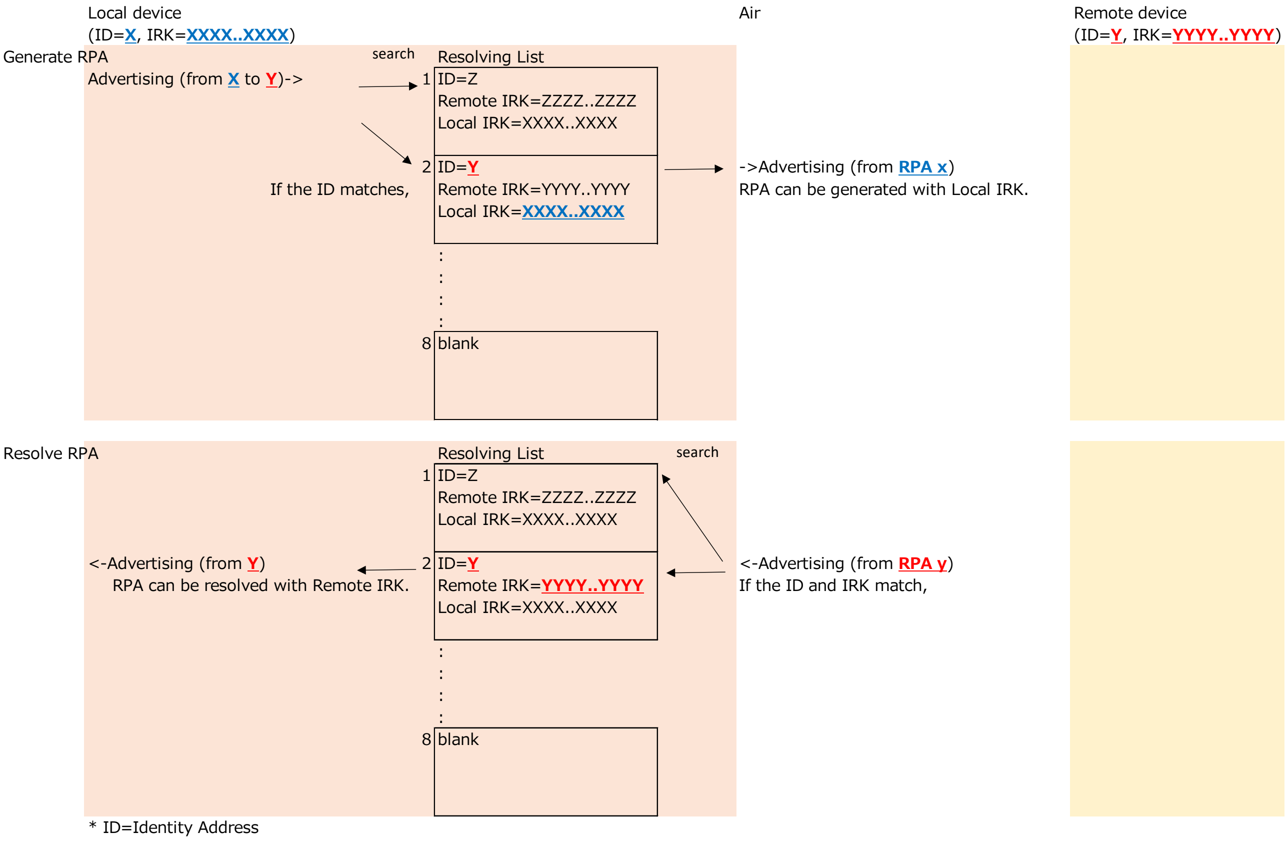


Figure 9-10 Image of Resolving List

The pairing procedure in an application is shown in Figure 9-11. The following sections describe the details of pairing steps.

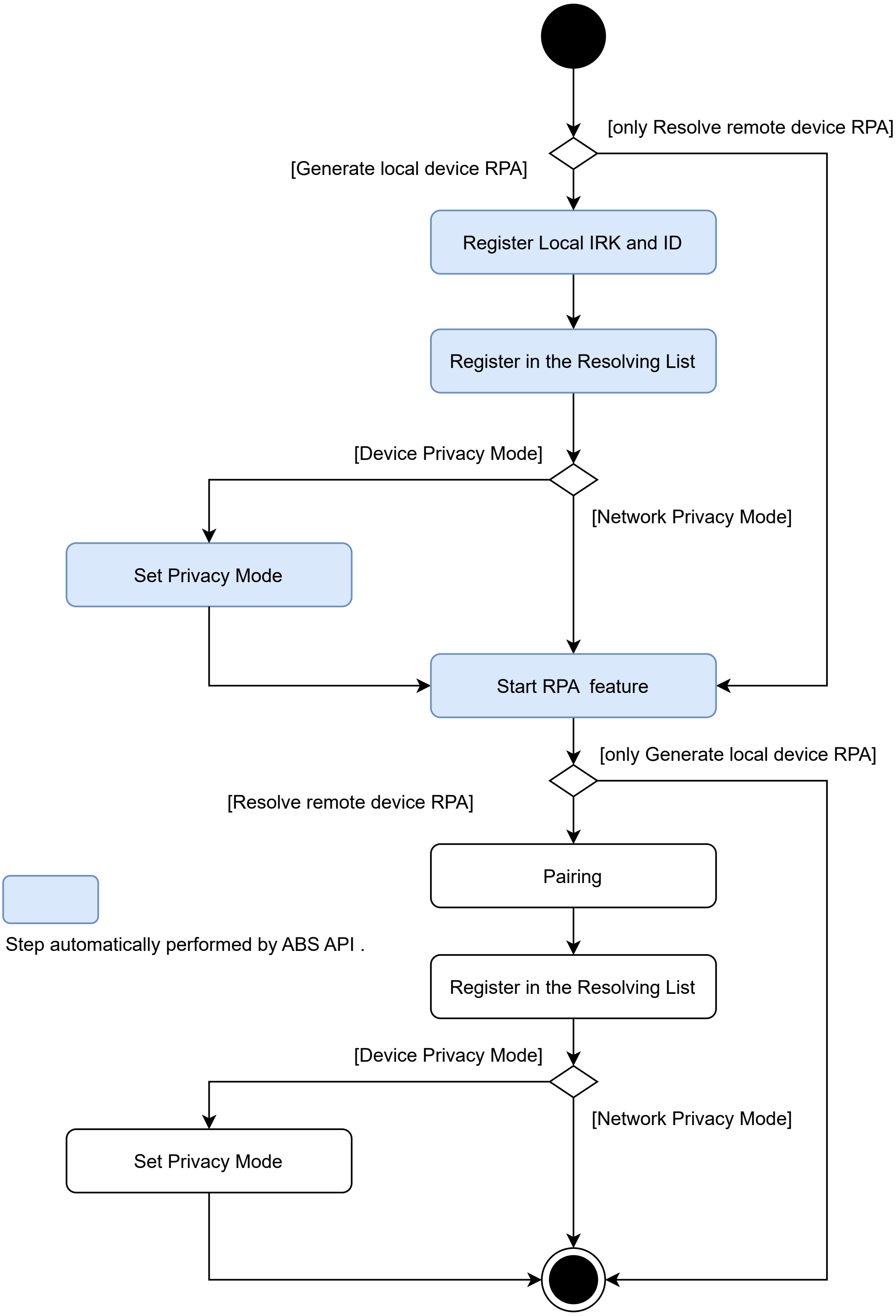

[only Resolve remote device RPA]
[Generate local device RPA]
Register Local IRK and ID
Register in the Resolving List
[Device Privacy Mode]
[Network Privacy Mode]
Set Privacy Mode
Start RPA feature
[only Generate local device RPA]
[Resolve remote device RPA]
Pairing
Register in the Resolving List
[Device Privacy Mode]
[Network Privacy Mode]
Set Privacy Mode
Step automatically performed by ABS API .

Figure 9-11 Privacy procedure in application

9.4.1 Generate local device RPA

Before local device uses RPA, perform the following step1-4. The API calls in step 1-4 can replace R_BLE_ABS_SetLocPrivacy.

1. Register local device key (IRK) and BD address
   Call R_BLE_VS_GetRand to generate the random value (16 bytes) notified by BLE_VS_EVENT_GET_RAND event as IRK. The IRK and Identity Address are registered by R_BLE_GAP_SetLocIdInfo into the Bluetooth LE Protocol Stack. The IRK is distributed to the remote device in pairing.

2. Register the IRK in the Resolving List
   Call R_BLE_GAP_ConfRslvList to register the IRK generated by 1 in the Resolving List. A set of Identity Address and IRK of a remote device needs to be registered to associate with the local device IRK. Set the Identity Address and IRK of the remote device to all 0x00 to associate with the local device IRK.. The completion is notified by BLE_GAP_EVENT_RSLV_LIST_CONF_COMP event.

3. Set Privacy Mode
   If Network Privacy Mode which is the default is used, the procedure does not need to be done.
   Call R_BLE_GAP_SetPrivMode to set the privacy mode. The completion is notified by BLE_GAP_EVENT_PRIV_MODE_SET_COMP event.

4. Start RPA feature
   Call R_BLE_GAP_EnableRpa to enable the RPA generation and resolution. The completion is notified by BLE_GAP_EVENT_RPA_EN_COMP event.

An example of the procedures 1-4 is shown below. If the Local device generates RPA, destination address must match the Identity Address in the Resolving List.

```
/** some code is omitted  **/
#include "sec_data/r_ble_sec_data.h"
/** some code is omitted  **/
st_ble_dev_addr_t gs_loc_bd_addr;
st_ble_dev_addr_t gs_rem_bd_addr;

/* Advertising parameters */
static st_ble_abs_legacy_adv_param_t gs_adv_param =
{
    /* TODO: Modify advertise parameters. */
    .p_addr          = &gs_rem_bd_addr,
    .o_addr_type     = BLE_GAP_ADDR_RPA_ID_PUBLIC,
    /** some code is omitted  **/
};
/** some code is omitted  **/

/* Vendor Specific callback function */
void vscb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    switch(event_type)
    {
        /** some code is omitted  **/
        case BLE_VS_EVENT_GET_RAND :
        {
            st_ble_vs_get_rand_comp_evt_t * p_rand_param;
            p_rand_param = (st_ble_vs_get_rand_comp_evt_t *)p_data->p_param;
            R_BLE_GAP_SetLocIdInfo(&gs_loc_bd_addr, p_rand_param->p_rand);

            /*  store local id info */
            R_BLE_SECD_WriteLocInfo(&gs_loc_bd_addr, p_rand_param->p_rand, NULL);

            /* Set all zero remote address & remote IRK */
            st_ble_gap_rslv_list_key_set_t peer_irk;

            memset(peer_irk.remote_irk, 0x00, BLE_GAP_IRK_SIZE);
            peer_irk.local_irk_type = BLE_GAP_RL_LOC_KEY_REGISTERED;
            memset(gs_rem_bd_addr.addr, 0x00, BLE_BD_ADDR_LEN);
            gs_rem_bd_addr.type = BLE_GAP_ADDR_PUBLIC;

            /* Add local IRK to resolving list */
            R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, &gs_rem_bd_addr, &peer_irk, 1);
        }
        break;
        /** some code is omitted  **/
    }
}

/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    switch(event_type)
    {
        /** some code is omitted  **/
        case BLE_GAP_EVENT_RSLV_LIST_CONF_COMP :
            {
                st_ble_gap_rslv_list_conf_evt_t * p_rslv_list_conf;
                p_rslv_list_conf = (st_ble_gap_rslv_list_conf_evt_t *)p_data->p_param;
                if(BLE_GAP_LIST_ADD_DEV == p_rslv_list_conf->op_code)
                {
                    uint8_t priv_mode;
                    priv_mode = BLE_GAP_NET_PRIV_MODE ;

                    /* Set Network Privacy Mode. */
                    R_BLE_GAP_SetPrivMode(&gs_rem_bd_addr, &priv_mode, 1);
                }
            }
            break;

        case BLE_GAP_EVENT_PRIV_MODE_SET_COMP :
            {
                /* Enable RPA. */
                R_BLE_GAP_EnableRpa(BLE_GAP_RPA_ENABLED);
            }
            break;

        case BLE_GAP_EVENT_LOC_VER_INFO:
            {
                st_ble_gap_loc_dev_info_evt_t * ev_param;
                ev_param = (st_ble_gap_loc_dev_info_evt_t *)p_data->p_param;
                gs_loc_bd_addr = ev_param->l_dev_addr;
                /* Generate IRK */
                R_BLE_VS_GetRand(BLE_GAP_IRK_SIZE);
            } break;

        case BLE_GAP_EVENT_RPA_EN_COMP:
            {
                 /* Start advertising */
                 R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
            } break;
        /** some code is omitted  **/
    }
}
```

Code 9-7 Prepare for using RPA in the local device (1)

An example using R_BLE_ABS_SetLocPrivacy is shown below.

```
/** some code is omitted */
st_ble_dev_addr_t gs_rem_bd_addr;

/* Advertising parameters */
static st_ble_abs_legacy_adv_param_t gs_adv_param =
{
    /* TODO: Modify advertise parameters. */
    .p_addr          = &gs_rem_bd_addr,
    .o_addr_type     = BLE_GAP_ADDR_RPA_ID_PUBLIC,
    /** some code is omitted */
};
/** some code is omitted */

/* GAP Callback */
void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    switch(event_type)
    {
        case BLE_GAP_EVENT_LOC_VER_INFO:
            {
                R_BLE_ABS_SetLocPrivacy(NULL, BLE_GAP_DEV_PRIV_MODE);
            } break;

        case BLE_GAP_EVENT_RPA_EN_COMP:
            {
                /* Start advertising */
                memset(gs_adv_param.p_addr->addr, 0x00, BLE_BD_ADDR_LEN);
                gs_adv_param.p_addr->type = BLE_GAP_ADDR_PUBLIC;
                R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
            } break;
        /** some code is omitted */
    }
}
```

Code 9-8 Prepare for using RPA in the local device (2)

When the local device Advertising or Scan Request or Connection Request operation with specified the RPA as own address, the packet includes the RPA.

[Advertising]

When setting the advertising parameters by R_BLE_GAP_SetAdvParam, configure the parameters in Table 5.4.

[Scan]

When setting the scan parameters by R_BLE_GAP_StartScan, configure RPA as own address type. Refer to “6.2.1 Privacy”.

[Connection]

When create a connection by R_BLE_GAP_CreateConn, configure RPA as own address type. Refer to “7.1.2 Privacy”.

9.4.2 Resolve remote device RPA

Remote device RPA is resolved according to the following procedure.

1. Start RPA feature
   Call R_BLE_GAP_EnableRpa to enable the RPA generation and resolution. The completion is notified by BLE_GAP_EVENT_RPA_EN_COMP event. This step can be replaced by R_BLE_ABS_SetLocPrivacy. If the local device does not use RPA, set a local IRK of the first parameter of R_BLE_ABS_SetLocPrivacy to all zeros.

2. Pairing
   Receive the remote device IRK and Identity Address by pairing. For more detail about pairing, see “9.1 Pairing”.

3. Register remote device key (IRK) and BD address
   Call R_BLE_GAP_ConfRslvList to register the remote device IRK and Identity Address in the Resolving List. The local device IRK is also registered at that time. If the local device does not use RPA, set a local IRK to all zeros by setting the local_irk_type in st_ble_gap_rslv_list_key_set_t type array of third parameter to BLE_GAP_RL_LOC_KEY_ALL_ZERO. The completion of the registry is notified by BLE_GAP_EVENT_RSLV_LIST_CONF_COMP event.

4. Set Privacy Mode
   If Network Privacy Mode which is the default is used, the procedure does not need to be done. Call the R_BLE_GAP_SetPrivMode to set the privacy mode. The completion is notified by BLE_GAP_EVENT_PRIV_MODE_SET_COMP event.

5. Resolve RPA
   After the procedures 1-3, the Bluetooth LE Protocol Stack can resolve the remote device RPA included in the received packet. Because of RPA resolution, the remote device address included in the event notified to the application becomes Identity Address.

An example of the procedures 1-5 is shown below. If the Local device resolves the RPA of the Remote device, the Identity Address and the IRK in the Resolving List must match Identity Address and IRK of the Remote device.

```
/** some code is omitted  **/
static st_ble_abs_scan_phy_param_t gs_phy_param_1M =
{
    .fast_intv                 = 0x0200,
    .slow_intv                 = 0x0800,
    .fast_window               = 0x0100,
    .slow_window               = 0x0100,
    .scan_type                 = BLE_GAP_SCAN_PASSIVE,
};
static st_ble_abs_scan_param_t gs_scan_param =
{
    .p_phy_param_1M            = &gs_phy_param_1M,
    .p_phy_param_coded         = NULL,
    .p_filter_data             = NULL,
    .fast_period               = 0x0100,
    .slow_period               = 0x0000,
    .filter_data_length        = 0,
    .dev_filter                = BLE_ABS_SCAN_ALL_STATIC,
    .filter_dups               = BLE_GAP_SCAN_FILT_DUPLIC_DISABLE,
};
static st_ble_abs_conn_phy_param_t gs_conn_phy_param =
{
    .conn_intv = 0x0130,
    .conn_latency = 0x0000,
    .sup_to = 0x03BB,
};
static st_ble_dev_addr_t gs_conn_bd_addr;
static st_ble_abs_conn_param_t gs_conn_param =
{
    .p_conn_1M = &gs_conn_phy_param,
    .p_addr = &gs_conn_bd_addr, /**< Set BD address of connecting device. */
    .filter = BLE_ABS_CONN_USE_ADDR_STATIC,
    .conn_to = 5,
};
static st_ble_abs_pairing_param_t gs_abs_pairing_param =
{
    .iocap          = BLE_GAP_IOCAP_NOINPUT_NOOUTPUT,
    .mitm           = BLE_GAP_SEC_MITM_BEST_EFFORT,
    .sec_conn_only  = BLE_GAP_SC_BEST_EFFORT,
    .loc_key_dist   = BLE_GAP_KEY_DIST_ENCKEY,
    .rem_key_dist   = BLE_GAP_KEY_DIST_ENCKEY | BLE_GAP_KEY_DIST_IDKEY,
    .max_key_size   = 16,
};
st_ble_dev_addr_t r_id_addr;
/** some code is omitted  **/

void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    /** some code is omitted  **/
    switch(event_type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_VS_GetBdAddr(BLE_VS_ADDR_AREA_REG, BLE_GAP_ADDR_RAND);
        } break;
        case BLE_GAP_EVENT_RPA_EN_COMP:
        {
            R_BLE_ABS_StartScan(&gs_scan_param);
        } break;
        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
            st_ble_gap_adv_rept_evt_t *p_adv_rept_param = (st_ble_gap_adv_rept_evt_t *)p_data->p_param;
            st_ble_gap_ext_adv_rept_t *p_ext_adv_rept_param = (st_ble_gap_ext_adv_rept_t
*)p_adv_rept_param->param.p_ext_adv_rpt;
            gs_conn_param.p_addr->type = p_ext_adv_rept_param->addr_type;
            memcpy(gs_conn_param.p_addr->addr, p_ext_adv_rept_param->p_addr, BLE_BD_ADDR_LEN);
            R_BLE_GAP_StopScan();
        } break;
        case BLE_GAP_EVENT_SCAN_OFF:
        {
            R_BLE_ABS_CreateConn(&gs_conn_param);
        } break;
        case BLE_GAP_EVENT_CONN_IND:
        {
            st_ble_gap_conn_evt_t *p_gap_conn_evt_param =
                (st_ble_gap_conn_evt_t *)p_data->p_param;
            R_BLE_ABS_StartAuth(p_gap_conn_evt_param->conn_hdl);
        } break;
        case BLE_GAP_EVENT_PEER_KEY_INFO:
        {
            st_ble_gap_peer_key_info_evt_t *p_peer_key_info_evt_param =
                (st_ble_gap_peer_key_info_evt_t *)p_data->p_param;
            st_ble_gap_key_dist_t * key_info;
            st_ble_gap_rslv_list_key_set_t  key_set;
            key_info = p_peer_key_info_evt_param->key_ex_param.p_keys_info;
            R_BLE_CLI_Printf("keys : 0x%02x\n", p_peer_key_info_evt_param->key_ex_param.keys);
            if(0 != (BLE_GAP_KEY_DIST_IDKEY & p_peer_key_info_evt_param->key_ex_param.keys))
            {
                /* Add remote address & irk to the resolving list. */
                memcpy(key_set.remote_irk, key_info->id_info, BLE_GAP_IRK_SIZE);
                key_set.local_irk_type = BLE_GAP_RL_LOC_KEY_REGISTERED;
                memcpy(r_id_addr.addr, &key_info->id_addr_info[1], BLE_BD_ADDR_LEN);
                r_id_addr.type = key_info->id_addr_info[0];
                R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, &r_id_addr, &key_set, 1);
            }
        } break;
        case BLE_GAP_EVENT_RSLV_LIST_CONF_COMP :
        {
            st_ble_gap_rslv_list_conf_evt_t * p_rslv_list_conf;
            p_rslv_list_conf = (st_ble_gap_rslv_list_conf_evt_t *)p_data->p_param;
            if(BLE_GAP_LIST_ADD_DEV == p_rslv_list_conf->op_code)
            {
                uint8_t priv_mode;
                priv_mode = BLE_GAP_NET_PRIV_MODE ;
                /* Set Network Privacy Mode. */
                R_BLE_GAP_SetPrivMode(&r_id_addr, &priv_mode, 1);
            }
        }
        break;
        /** some code is omitted  **/
    }
}

/* Vendor Specific callback function */
void vs_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    switch(type)
    {
        case BLE_VS_EVENT_GET_ADDR_COMP:
        {
            /* Enable RPA. */
            R_BLE_GAP_EnableRpa(BLE_GAP_RPA_ENABLED);
        } break;
    }
}
/** some code is omitted  **/
```

Code 9-9 Sample for resolving RPA of remote device (1)

An example using R_BLE_ABS_SetLocPrivacy is shown below.

```
/** some code is omitted  **/
#include "sec_data/r_ble_sec_data.h"
typedef struct
{
    /* identity address */
    st_ble_dev_addr_t idaddr[BLE_CFG_NUM_BOND + 1];
    /* local & remote IRK set */
    st_ble_gap_rslv_list_key_set_t key_set[BLE_CFG_NUM_BOND + 1];
    /* the number of identity info stored in Data Flash */
    uint8_t gs_bond_cnt;
}  st_ble_app_idinfo_t;
static st_ble_app_idinfo_t gs_idinfo;
static st_ble_gap_rslv_list_key_set_t g_ble_peer_dummy_irk;
static st_ble_abs_scan_phy_param_t gs_phy_param_1M =
{
    .fast_intv                  = 0x0200,
    .slow_intv                  = 0x0800,
    .fast_window                = 0x0100,
    .slow_window                = 0x0100,
    .scan_type                  = BLE_GAP_SCAN_PASSIVE,
};
static st_ble_abs_scan_param_t gs_scan_param =
{
    .p_phy_param_1M             = &gs_phy_param_1M,
    .p_phy_param_coded          = NULL,
    .p_filter_data              = NULL,
    .fast_period                = 0x0100,
    .slow_period                = 0x0000,
    .filter_data_length         = 0,
    .dev_filter                 = BLE_ABS_SCAN_ALL_STATIC,
    .filter_dups                = BLE_GAP_SCAN_FILT_DUPLIC_DISABLE,
};
static st_ble_abs_conn_phy_param_t gs_conn_phy_param =
{
    .conn_intv = 0x0130,
    .conn_latency = 0x0000,
    .sup_to = 0x03BB,
};
static st_ble_dev_addr_t gs_conn_bd_addr;
static st_ble_abs_conn_param_t gs_conn_param =
{
    .p_conn_1M = &gs_conn_phy_param,
    .p_addr = &gs_conn_bd_addr, /**< Set BD address of connecting device. */
    .filter = BLE_ABS_CONN_USE_ADDR_STATIC,
    .conn_to = 5,
};
static st_ble_abs_pairing_param_t gs_abs_pairing_param =
{
    .iocap          = BLE_GAP_IOCAP_NOINPUT_NOOUTPUT,
    .mitm           = BLE_GAP_SEC_MITM_BEST_EFFORT,
    .sec_conn_only  = BLE_GAP_SC_BEST_EFFORT,
    .loc_key_dist   = BLE_GAP_KEY_DIST_ENCKEY,
    .rem_key_dist   = BLE_GAP_KEY_DIST_ENCKEY | BLE_GAP_KEY_DIST_IDKEY,
    .max_key_size   = 16,
};
static void ble_app_start_scan(void)
{
    R_BLE_ABS_StartScan(&gs_scan_param);
}
static void ble_app_conn_set_event(void)
{
    R_BLE_ABS_CreateConn(&gs_conn_param);
}
/** some code is omitted  **/

void gap_cb(uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t * p_data)
{
    /** some code is omitted  **/
    switch(event_type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_VS_GetBdAddr(BLE_VS_ADDR_AREA_REG, BLE_GAP_ADDR_RAND);
        } break;
        case BLE_GAP_EVENT_RPA_EN_COMP:
        {
            if((0 != gs_idinfo.gs_bond_cnt))
            {
                /* register remote address & irk */
                R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV,
                                       gs_idinfo.idaddr,
                                       gs_idinfo.key_set,
                                       gs_idinfo.gs_bond_cnt);
                R_BLE_SetEvent(ble_app_start_scan);
            }
            else
            {
                ble_app_start_scan();
            }
        } break;
        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
            st_ble_gap_adv_rept_evt_t *p_adv_rept_param = (st_ble_gap_adv_rept_evt_t *)p_data->p_param;
            st_ble_gap_ext_adv_rept_t *p_ext_adv_rept_param = (st_ble_gap_ext_adv_rept_t
*)p_adv_rept_param->param.p_ext_adv_rpt;
            gs_conn_param.p_addr->type = p_ext_adv_rept_param->addr_type;
            memcpy(gs_conn_param.p_addr->addr, p_ext_adv_rept_param->p_addr, BLE_BD_ADDR_LEN);
            R_BLE_GAP_StopScan();
        } break;
        case BLE_GAP_EVENT_SCAN_OFF:
        {
            st_ble_gap_rslv_list_key_set_t * p_key_set;
            uint8_t i;
            uint8_t peer_addr_type;
            peer_addr_type = gs_conn_param.p_addr->type;
            gs_conn_param.p_addr->type = gs_conn_param.p_addr->type % 2;
            if(BLE_GAP_ADDR_RAND < peer_addr_type)
            {
                /* Local device can resolve the remote device address. */
                R_BLE_SECD_GetIdInfo(gs_idinfo.idaddr, gs_idinfo.key_set, &gs_idinfo.gs_bond_cnt);
                R_BLE_CLI_Printf("Remote IRK count :0x%02x \n", gs_idinfo.gs_bond_cnt);
                p_key_set = NULL;
                if(0 != gs_idinfo.gs_bond_cnt)
                {
                    for(i=0; i<BLE_CFG_NUM_BOND + 1; i++)
                    {
                        if(0 == memcmp(gs_idinfo.idaddr[i].addr, gs_conn_param.p_addr, 
sizeof(st_ble_dev_addr_t)))
                        {
                            p_key_set = &gs_idinfo.key_set[i];
                            break;
                        }
                    }
                }
                if(NULL == p_key_set)
                {
                    p_key_set = &g_ble_peer_dummy_irk;
                }
                p_key_set->local_irk_type = BLE_GAP_RL_LOC_KEY_ALL_ZERO;
                R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, gs_conn_param.p_addr, p_key_set, 1);
                R_BLE_SetEvent(ble_app_conn_set_event);
            }
            else
            {
                /* Local device can't resolve the remote device address. */
                ble_app_conn_set_event();
            }
        } break;
        case BLE_GAP_EVENT_PEER_KEY_INFO:
        {
            st_ble_gap_peer_key_info_evt_t *p_peer_key_info_evt_param =
                (st_ble_gap_peer_key_info_evt_t *)p_data->p_param;
            st_ble_gap_key_dist_t * key_info;
            st_ble_gap_rslv_list_key_set_t  key_set;
            key_info = p_peer_key_info_evt_param->key_ex_param.p_keys_info;
            R_BLE_CLI_Printf("keys : 0x%02x\n", p_peer_key_info_evt_param->key_ex_param.keys);
            if(0 != (BLE_GAP_KEY_DIST_IDKEY & p_peer_key_info_evt_param->key_ex_param.keys))
            {
                /* Add remote address & irk to the resolving list. */
                st_ble_dev_addr_t r_id_addr;
                memcpy(key_set.remote_irk, key_info->id_info, BLE_GAP_IRK_SIZE);
                key_set.local_irk_type = BLE_GAP_RL_LOC_KEY_REGISTERED;
                memcpy(r_id_addr.addr, &key_info->id_addr_info[1], BLE_BD_ADDR_LEN);
                r_id_addr.type = key_info->id_addr_info[0];
                R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, &r_id_addr, &key_set, 1);
            }
        } break;
        /** some code is omitted  **/
    }
}
static void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_VS_EVENT_GET_ADDR_COMP:
        {
            /* Get remote irk & identity address from Data Flash. */
            R_BLE_SECD_GetIdInfo(gs_idinfo.idaddr, gs_idinfo.key_set, &gs_idinfo.gs_bond_cnt);
            R_BLE_CLI_Printf("Remote IRK count :0x%02x \n", gs_idinfo.gs_bond_cnt);
            if((0 != gs_idinfo.gs_bond_cnt))
            {
                /* Already create local irk and have remote irk. */
                R_BLE_GAP_EnableRpa(BLE_GAP_RPA_ENABLED);
            }
            else
            {
                /* Initial state or remote device did not distribute irk. */
                uint8_t irk[BLE_GAP_IRK_SIZE];
                uint8_t irk_check[BLE_GAP_IRK_SIZE];
                uint8_t * p_irk;
                st_ble_dev_addr_t idaddr;
                ble_status_t retval;
                retval = R_BLE_SECD_ReadLocInfo(&idaddr, irk, NULL);
                memset(irk_check, 0x00, BLE_GAP_IRK_SIZE);
                p_irk = NULL;
                if((BLE_SUCCESS == retval) && (0 != memcmp(irk_check, irk, BLE_GAP_IRK_SIZE)))
                {
                    p_irk = irk;
                }
                R_BLE_ABS_SetLocPrivacy(p_irk, BLE_ABS_PRIV_NET_STATIC_IDADDR);
            }
        } break;
    }
}
static void disc_comp_cb(uint16_t conn_hdl)
{
    R_BLE_CLI_Printf("disc finished\n");
    R_BLE_ABS_StartAuth(conn_hdl);
    return;
}
/** some code is omitted  **/
```

Code 9-10 Sample for resolving RPA of remote device (2)

After resolving the RPA, you will need to use the Identity Address to connect and to register the whitelist.

If you restart the Bluetooth LE Protocol Stack, you will need to reset the key stored on the device to the Resolving List by R_BLE_GAP_ConfRslvList.

Please refer to "9.2.3 Reset the stored keys ".

9.4.2.1 Not generate local device RPA

If the local device resolves remote device RPA but doesn’t generate own RPA and uses Public Address or Static Address, change app_main.c according to the following.

1. Change g_ble_peer_dummy_irk.

   Set the local_irk_type field in the g_ble_peer_dummy_irk variable defined the Abstraction API to BLE_GAP_RL_LOC_KEY_ALL_ZERO before call R_BLE_ABS_Init() such as Code 9-11.

```
/******************************************************************************
 User global variables
*******************************************************************************/
/* Start user code for global variables. Do not edit comment generated here */
extern st_ble_gap_rslv_list_key_set_t g_ble_peer_dummy_irk;
/* End user code. Do not edit comment generated here */
…

static ble_status_t ble_init(void)
{
    ble_status_t status;

/* Start user code for global value initialization. Do not edit comment generated here */
    g_ble_peer_dummy_irk.local_irk_type =  BLE_GAP_RL_LOC_KEY_ALL_ZERO;

/* End user code. Do not edit comment generated here */

    /* Initialize the Low Power Control function */
    R_BLE_LPC_Init();

    /* Initialize Timer Library */
    R_BLE_TIMER_Init();

    /* Initialize host stack */
    status = R_BLE_ABS_Init(&gs_abs_init_param);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }
```

Code 9-11 Sample for changing g_ble_peer_dummy_irk

2. Change the value obtained by R_BLE_SECD_GetIdInfo().

   If you set the value obtained by R_BLE_SECD_GetIdInfo() to Resolving List, change the local_irk_type field in the st_ble_gap_rslv_list_key_set_t type array obtained as the second parameter to BLE_GAP_RL_LOC_KEY_ALL_ZERO and then call R_BLE_GAP_ConfRslvList() such as Code 9-12.

```
static void ble_app_set_resolving_list(void)
{
    st_ble_dev_addr_t idaddr[BLE_CFG_NUM_BOND + 1] = {0};
    st_ble_gap_rslv_list_key_set_t key_set[BLE_CFG_NUM_BOND + 1] = {0};
    uint8_t cnt = 0;
    uint8_t i;

    /* Get remote irk & identity address from Data Flash. */
    R_BLE_SECD_GetIdInfo(idaddr, key_set, &cnt);

    if(0 !=cnt)
    {
        for(i=0; i<cnt; i++)
        {
           key_set[i].local_irk_type = BLE_GAP_RL_LOC_KEY_ALL_ZERO;
        }

        R_BLE_GAP_ConfRslvList(BLE_GAP_LIST_ADD_DEV, idaddr, key_set, cnt);
```

Code 9-12 Sample for changing the value obtained by R_BLE_SECD_GetIdInfo()

10. Profile and service

Profiles in Bluetooth LE communication are mechanisms for ensuring interoperability between devices by defining the services and communication protocols that application share. Profile-based data communication is achieved by accessing a common data structure called GATT database. As shown in Figure 10-1, the GATT database consists of one or more services and the characteristics they contain. Services consist of one or more characteristic that enable profile functionality, and characteristics define data structures and access procedures. The procedure for accessing characteristics is called GATT procedure, and this procedure defines how to send and receive data.

The user profile can be designed using QE for BLE. For information on how to design profiles using QE for BLE, refer “RX23W Group Bluetooth Low Energy Profile Developer’s Guide (R01AN6459)”.

This chapter introduces the profiles and services provided by Renesas and explains APIs for each GATT procedure including examples of how to use them.

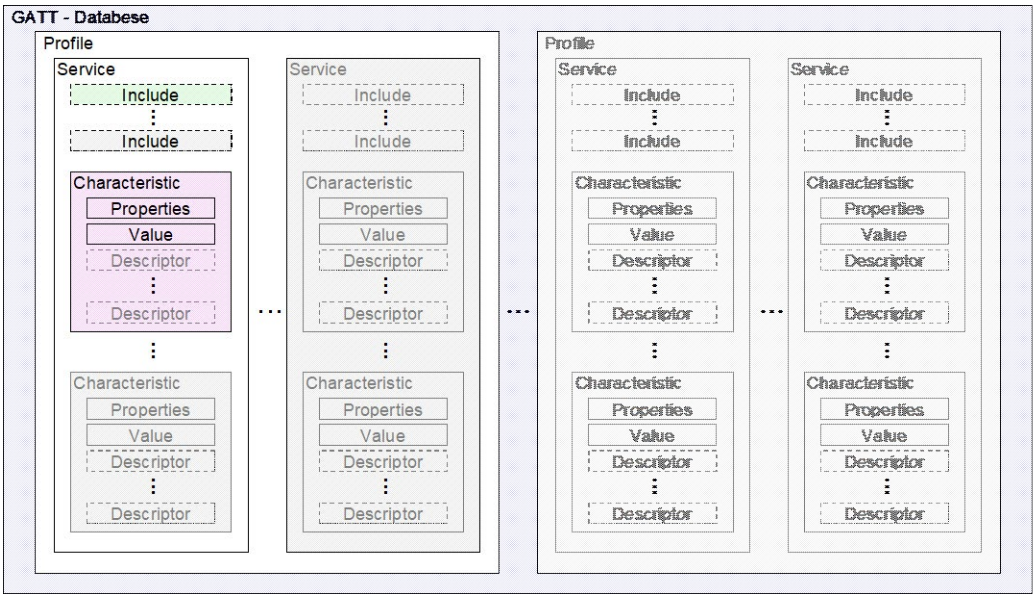


Figure 10-1 Data structure of GATT database

10.1 Standard profile and Standard Service

Standard profiles and services can be used in user applications using QE for BLE. RX23W supports the standard profiles and services listed in Table 10.1. Table 10.2 lists the characteristics that make up each standard service.

Table 10.1 Profile supported by RX23W

Usage	Profile	Service			
Healthcare	Blood Pressure Profile	BLS	DIS		
	Health Thermometer Profile	HTS	DIS		
	Heart Rate Profile	HRS	DIS		
	Glucose Profile	GLS	DIS		
	Pulse Oximeter Profile	PLXS	DIS	BAS	CTS
		BMS			
	Continuous Glucose Monitoring Profile	CGMS	DIS	BMS	
	Reconnection Configuration Profile	RCS	BMS		
	Insulin Delivery Profile	IDS	DIS	BAS	CTS
		BMS	IAS		
Sports and Fitness	Cycling Power Profile	CPS	DIS	BAS	
	Cycling Speed and Cadence Profile	CSCS	DIS		
	Running Speed and Cadence Profile	RSCS	DIS		
	Location and Navigation Profile	LNS	DIS	BAS	
	Weight Scale Profile	WSS	BCS	DIS	BAS
		CTS	UDS		
	Fitness Machine Profile	FTMS	DIS	UDS	
	Environmental Sensing Profile	ESS	DIS	BAS	
Radio tag	Find Me Profile	IAS			
	Proximity Profile	IAS	LLS	TPS	
Smartphone	Alert Notification Profile	ANS			
	Phone Alert Status Profile	PASS			
	Time Profile	CTS	NDCS	RTUS	
HID (Human Interface Device)	HID over GATT Profile	HIDS	DIS	BAS	
	Scan Parameters Profile	SCPS			
Industrial equipment	Automation IO Profile	AIOS			

Table 10.2 Structure of standard service

Service	Characteristic	GATT Procedure
Alert Notification Service **ANS**	Supported New Alert Category	Read
	New Alert	Notify
	Supported Unread Alert Category	Read
	Unread Alert Status	Notify
Automation IO Service **AIOS**	Digital 0	Read, Write, WriteWithoutResponse, Notify
	Digital 1	Read, Write, WriteWithoutResponse, Notify
	Analog 0	Read, Write, WriteWithoutResponse, Notify
	Analog 1	Read, Write, WriteWithoutResponse, Notify
	Aggregate	Read, Notify
Battery Service **BAS**	Battery Level	Read, Notify
Blood Pressure Service **BLS**	Blood Pressure Measurement	Indicate
	Intermediate Cuff Pressure	Notify
	Blood Pressure Feature	Read, Indicate
Body Composition Service **BCS**	Body Composition Feature	Read
	Body Composition Measurement	Indicate
Bond Management Service **BMS**	Bond Management Control Point	Write, ReliableWrite
	Bond Management Feature	Read, Indicate
Continuous Glucose Monitoring Service **CGMS**	CGM Measurement	Notify
	CGM Feature	Read, Indicate
	CGM Status	Read
	CGM Session Start Time	Read, Write
	CGM Session Run Time	Read
	Record Access Control Point	Write, Indicate
	CGM Specific Ops Control Point	Write, Indicate
Current Time Service **CTS**	Current Time	Read, Write, Notify
	Local Time Information	Read, Write
	Reference Time Information	Read
Cycling Power Service **CPS**	Cycling Power Measurement	Notify, Broadcast
	Cycling Power Feature	Read
	Sensor Location	Read
	Cycling Power Vector	Notify
	Cycling Power Control Point	Write, Indicate
Cycling Speed and Cadence Service **CSCS**	CSC Measurement	Notify
	CSC Feature	Read
	Sensor Location	Read
	SC Control Point	Write, Indicate

Service	Characteristic	GATT Procedure
Device Information Service **DIS**	Manufacturer Name String	Read
	Model Number String	Read
	Serial Number String	Read
	Hardware Revision String	Read
	Firmware Revision String	Read
	Software Revision String	Read
	System ID	Read
	IEEE 11073-20601 Regulatory Certification Data List	Read
	PnP ID	Read
Environmental Sensing Service **ESS**	Descriptor Value Changed	Indicate
	Temperature 0	Read, Notify
	Temperature 1	Read, Notify
	Elevation 0	Read, Notify
	Elevation 1	Read, Notify
Fitness Machine Service **FTMS**	Fitness Machine Feature	Read
	Treadmill Data	Notify
	Cross Trainer Data	Notify
	Step Climber Data	Notify
	Stair Climber Data	Notify
	Rower Data	Notify
	Indoor Bike Data	Notify
	Training Status	Read, Notify
	Supported Speed Range	Read
	Supported Inclination Range	Read
	Supported Resistance Level Range	Read
	Supported Power Range	Read
	Supported Heart Rate Range	Read
	Fitness Machine Control Point	Write, Indicate
	Fitness Machine Status	Notify
GAP Service **GAP**	Device Name	Read, Write
	Appearance	Read
	Peripheral Preferred Connection Parameters	Read
	Central Address Resolution	Read
	Resolvable Private Address Only	Read
GATT Service **GATT**	Service Changed	Indicate
Glucose Service **GLS**	Glucose Measurement	Notify
	Glucose Measurement Context	Notify
	Glucose Feature	Read, Indicate
	Record Access Control Point	Write, Indicate

Service	Characteristic	GATT Procedure
Health Thermometer Service **HTS**	Temperature Measurement	Indicate
	Temperature Type	Read
	Intermediate Temperature	Notify
	Measurement Interval	Read, Write, Indicate
Heart Rate Service **HRS**	Heart Rate Measurement	Notify
	Body Sensor Location	Read
	Heart Rate Control Point	Write
Human Interface Device Service **HIDS**	Protocol Mode	Read, WriteWithoutResponse
	Report	Read, Write, WriteWithoutResponse, Notify
	Report Map	Read
	Boot Keyboard Input Report	Read, Write, Notify
	Boot Keyboard Output Report	Read, Write, WriteWithoutResponse
	Boot Mouse Input Report	Read, Write, Notify
	HID Information	Read
	HID Control Point	WriteWithoutResponse
Immediate Alert Service **IAS**	Alert Level	WriteWithoutResponse
Insulin Delivery Service **IDS**	IDD Status Changed	Read, Indicate
	IDD Status	Read, Indicate
	IDD Annunciation Status	Read, Indicate
	IDD Features	Read, Indicate
	IDD Status Reader Control Point	Write, Indicate
	IDD Command Control Point	Write, Indicate
	IDD Command Data	InformativeText, Notify
	IDD Record Access Control Point	Write, Indicate
	IDD History Data	InformativeText, Notify
Link Loss Service **LLS**	Alert Level	Read, Write
Location and Navigation Service **LNS**	LN Feature	Read
	Location and Speed	Notify
	Position Quality	Read
	LN Control Point	Write, Indicate
	Navigation	Notify
Next DST Change Service **NDCS**	Time with DST	Read
Object Transfer Service **OTS**	OTS Feature	Read
	Object Name	Read, Write
	Object Type	Read
	Object Size	Read
	Object First-Created	Read, Write

Service	Characteristic	GATT Procedure
	Object Last-Modified	Read, Write
	Object ID	Read
	Object Properties	Read, Write
	Object Action Control Point	Write, Indicate
	Object List Control Point	Write, Indicate
	Object List Filter 0	Read, Write
	Object List Filter 1	Read, Write
	Object List Filter 2	Read, Write
	Object Changed	Indicate
Phone Alert Status Service **PASS**	Alert Status	Read, Notify
	Ringer Setting	Read, Notify
	Ringer Control point	WriteWithoutResponse
Pulse Oximeter Service **PLXS**	PLX Spot-Check Measurement	Indicate
	PLX Continuous Measurement	Notify
	PLX Features	Read, Indicate
	Record Access Control Point	Write, Indicate
Reconnection Configuration Service **RCS**	RC Feature	Read, Indicate
	RC Settings	Read, Notify
	Reconnection Configuration Control Point	Write, Indicate
Reference Time Update Service **RTUS**	Time Update Control Point	WriteWithoutResponse
	Time Update State	Read
Running Speed and Cadence Service **RSCS**	RSC Measurement	Notify
	RSC Feature	Read
	Sensor Location	Read
	SC Control Point	Write, Indicate
Scan Parameters Service **SCPS**	Scan Interval Window	WriteWithoutResponse
	Scan Refresh	Notify
Tx Power Service **TPS**	Tx Power Level	Read
User Data Service **UDS**	First Name	Read, Write
	Last Name	Read, Write
	Email Address	Read, Write
	Age	Read, Write
	Date of Birth	Read, Write
	Gender	Read, Write
	Weight	Read, Write
	Height	Read, Write
	VO2 Max	Read, Write
	Heart Rate Max	Read, Write

Service	Characteristic	GATT Procedure
	Resting Heart Rate	Read, Write
	Maximum Recommended Heart Rate	Read, Write
	Aerobic Threshold	Read, Write
	Anaerobic Threshold	Read, Write
	Sport Type for Aerobic and Anaerobic Thresholds	Read, Write
	Date of Threshold Assessment	Read, Write
	Waist Circumference	Read, Write
	Hip Circumference	Read, Write
	Fat Burn Heart Rate Lower Limit	Read, Write
	Fat Burn Heart Rate Upper Limit	Read, Write
	Aerobic Heart Rate Lower Limit	Read, Write
	Aerobic Heart Rate Upper Limit	Read, Write
	Anaerobic Heart Rate Lower Limit	Read, Write
	Anaerobic Heart Rate Upper Limit	Read, Write
	Five Zone Heart Rate Limits	Read, Write
	Three Zone Heart Rate Limits	Read, Write
	Two Zone Heart Rate Limit	Read, Write
	Database Change Increment	Read, Write, Notify
	User Index	Read
	User Control Point	Write, Indicate
	Language	Read, Write
	Registered User	Read, Write
	Preferred Units	Read, Write
	High Resolution Height	Read, Write
	Middle Name	Read, Write
	Stride Length	Read, Write
	Handedness	Read, Write
	Device Wearing Position	Read, Write
	Four Zone Heart Rate Limits	Read, Write
	High Intensity Exercise Threshold	Read, Write
	Activity Goal	Read, Write
	Sedentary Interval Notification	Read, Write
	Caloric Intake	Read, Write
Weight Scale Service **WSS**	Weight Scale Feature	Read
	Weight Measurement	Indicate

10.2 APIs of GATT Procedure

QE for BLE generates APIs depending on the GATT procedure set to the characteristic. This section describes how to implement each GATT procedure that can be configured from QE for BLE.
In the following description, we will use the function name and event name which will be generated from QE for BLE. Abbreviation of the service is set to "XXX" and abbreviation of characteristic is set to "YYY" in QE for BLE.

10.2.1 Read operation

Read operation is a procedure of the GATT client to check the data configured in the GATT database of the GATT server. Using this procedure is recommended when checking the configuration and status of the GATT server.

GATT server:
When GATT server receives "Read Request", Bluetooth LE Protocol Stack transmits "Read Response" with the value set in the GATT database. The event "BLE_XXX_EVENT_YYY_READ_REQ" occurs after receiving "Read Response" but before determining the data to be send in "Read Response". If you want to change the data to be transmitted, use function "R_BLE_XXX_SetYYY()" to change the value set in the GATT database. You can also send errors by using the function "R_BLE_GATTS_SetErrRsp()".

GATT client:
"Read Request" can be transmitted by using the function "R_BLE_XXX_ReadYYY()" in Application. The event "BLE_XXX_EVENT_YYY_READ_RSP" notifies the data received in "Read Response" to the application. The data notified in this event is in form of a structure in Field of QE for BLE because decode function is used in Bluetooth LE Protocol Stack. Read operation is completed when the event "BLE_XXX_EVENT _YYY_READ_RSP" is notified. You can start following operation after this event.

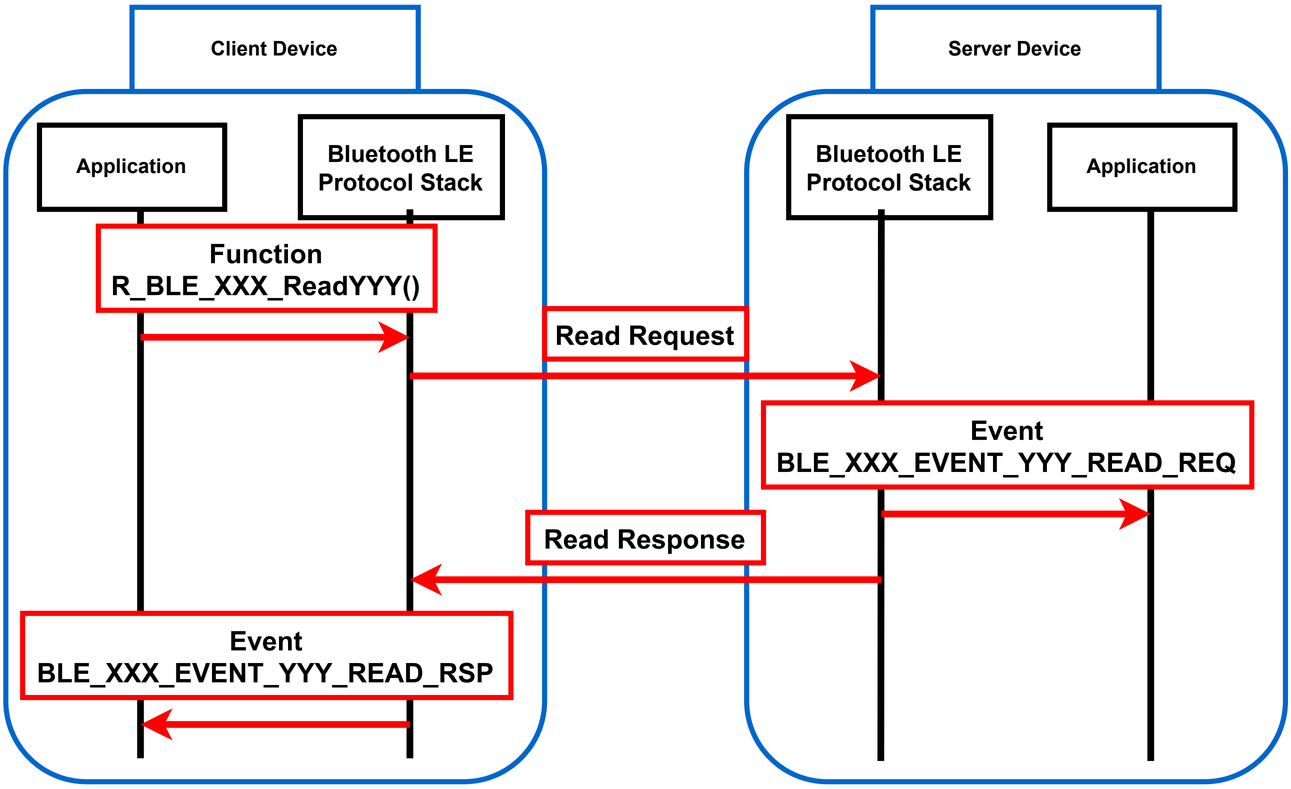

Client Device
Server Device
Application
Bluetooth LE Protocol Stack
Bluetooth LE Protocol Stack
Application
Function R_BLE_XXX_ReadYYY()
Read Request
Event BLE_XXX_EVENT_YYY_READ_REQ
Read Response
Event BLE_XXX_EVENT_YYY_READ_RSP

Figure 10-2 Flow of Read operation

10.2.2 Write operation

Write operation is a procedure to change the GATT database of the GATT server by sending data from the GATT client. GATT client can check whether the submitted data is reflected in the GATT database in response from the GATT server. Using this procedure is recommended when you want to change the settings of the GATT server.

GATT server:
Data received in “Write Request” is notified to the application by the event “BLE_XXX_EVENT_YYY_WRITE_REQ” and “BLE_XXX_EVENT_WRITE_COMP”. The data notified in this event is in form of a structure in Field of QE for BLE because decode function is used in Bluetooth LE Protocol Stack. Event “BLE_XXX_EVENT_WRITE_REQ” is an event to check the data received by “Write Request” before being written to the GATT database. If you receive invalid data, use function “R_BLE_GATTS_SetErrRsp()” to send an error and the data would not be reflected in the GATT database. If you do not send an error, Bluetooth LE Protocol Stack sends “Write Response”, so you do not need to add any process to respond in application. Event “BLE_XXX_EVENT_YYY_WRITE_COMP” is an event after the data received by “Write Request” is reflected in the GATT database and “Write Response” is sent. Process that references GATT database directly or corresponds to the data received by “Write Request” should be added after this event.

GATT client:
You can sent “Write Request” by using the function “R_BLE_XXX_WriteYYY()” in application. Result of the Write operation can be checked by the event “BLE_XXX_EVENT_YYY_WRITE_RSP”. Write operation is completed when the event “BLE_XXX_EVENT _YYY_WRITE_RSP” is notified. You can start following operation after this event.

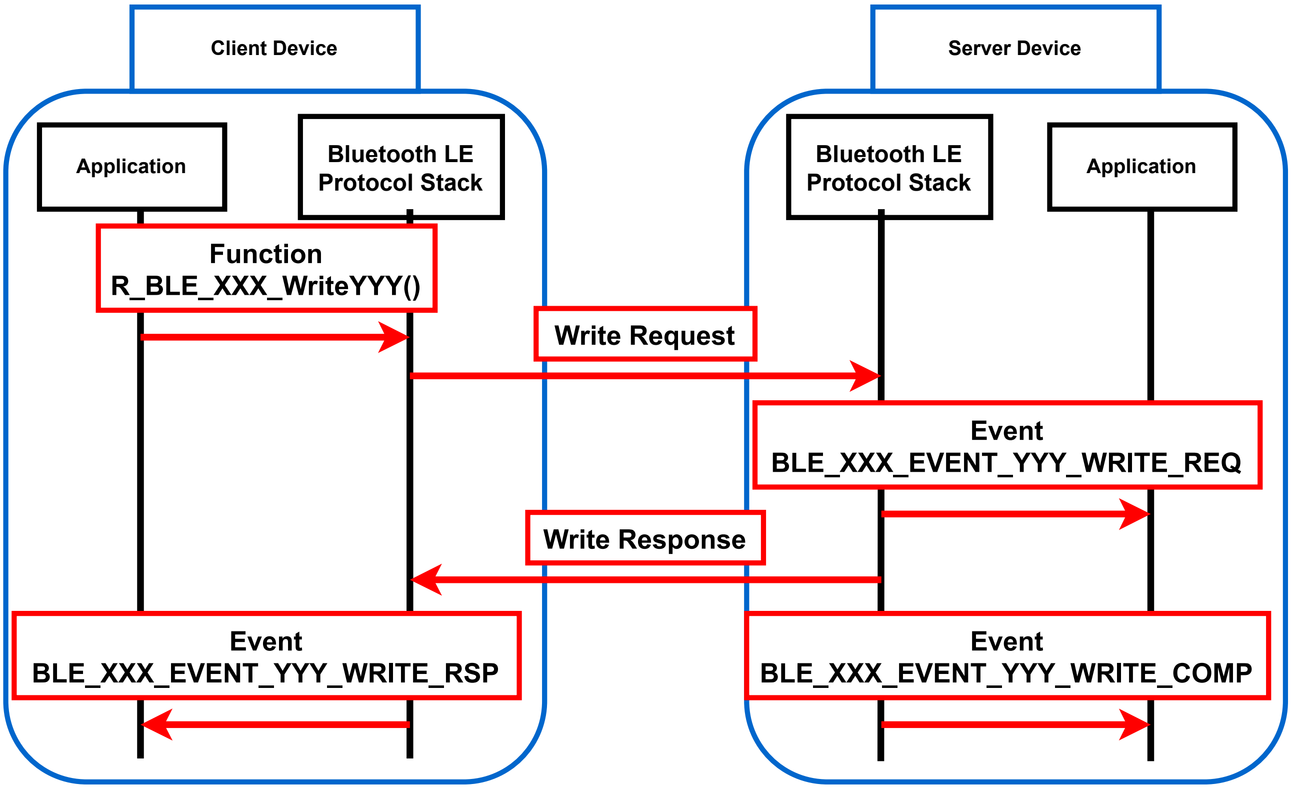

Client Device
Server Device
Application
Bluetooth LE Protocol Stack
Bluetooth LE Protocol Stack
Application
Function R_BLE_XXX_WriteYYY()
Write Request
Event BLE_XXX_EVENT_YYY_WRITE_REQ
Write Response
Event BLE_XXX_EVENT_YYY_WRITE_RSP
Event BLE_XXX_EVENT_YYY_WRITE_COMP

Figure 10-3 Flow of Write operation

10.2.3 WriteWithoutResponse operation

WriteWithoutResponse operation is a procedure to change the GATT database of the GATT server by sending data from the GATT client. Because there is no response from the GATT server, it is possible to continuously transmit data from GATT client and lower power consumption of GATT server devices, while it is not possible to verify that the data sent by GATT client is reflected in the GATT database. Using this procedure is recommended when you need low power consumption on your device, or when you need to send data continuously from GATT client.

GATT server:
Data received in "Write Command" is notified to application by the event "BLE_XXX_EVENT_YYY_WRITE_CMD". The data notified in this event is in form of a structure in Field of QE for BLE because decode function is used in Bluetooth LE Protocol Stack. When the event "BLE_XXX_EVENT_YYY_WRITE_CMD" is notified, changes to the GATT database are not reflected, so do not add any action that directly references the GATT database.

GATT client:
You can send "Write Command" by using the function "R_BLE_XXX_WriteWithoutResponseYYY()" in application. WriteWithoutResponse operation is completed when the function "R_BLE_XXX_WriteWithoutResponseYYY()" is used. You can start following operation after this event.

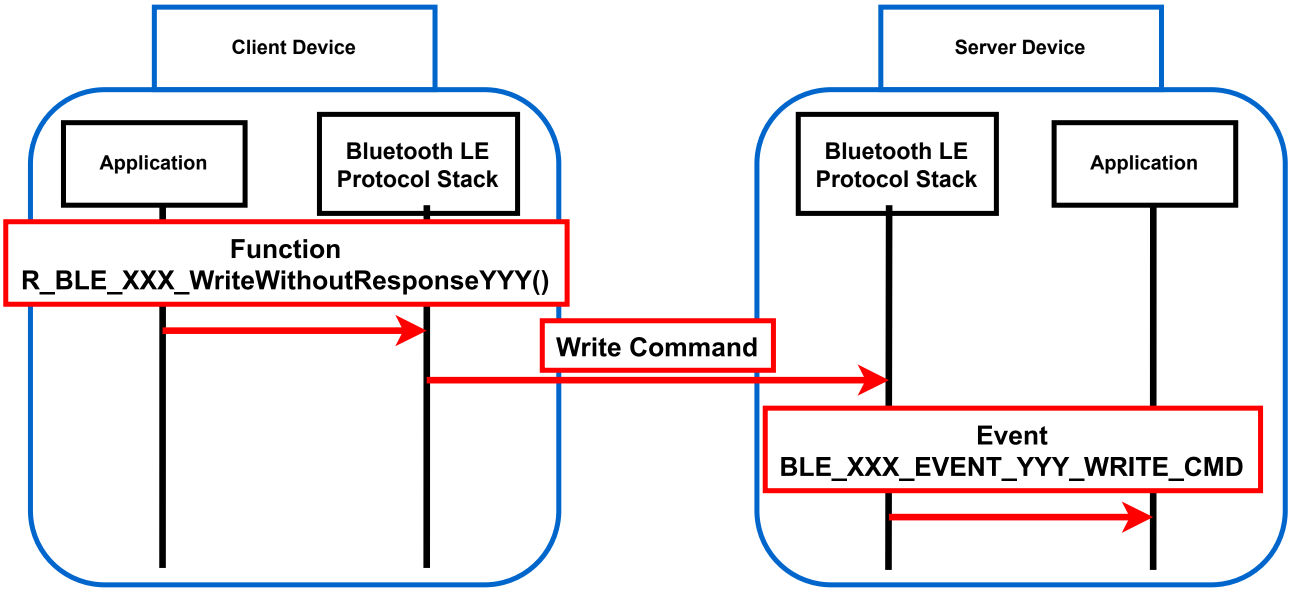

Client Device
Application
Bluetooth LE Protocol Stack
Function
R_BLE_XXX_WriteWithoutResponseYYY()
Write Command
Server Device
Bluetooth LE Protocol Stack
Application
Event
BLE_XXX_EVENT_YYY_WRITE_CMD

Figure 10-4 Flow of WriteWithoutResponse operation

10.2.4 Notification operation

Notification operation is a procedure to send data from the GATT server to the GATT client. For Notification operation, the CCCD must have been added as a descriptor. The GATT client must also set the CCCD to the appropriate value before the operation. Because there is no response from the GATT client, it is possible to send data continuously from the GATT server, but it is not possible to verify that the GATT client received the data sent from GATT server. Using this procedure is recommended when you want to send data continuously from the GATT server.

GATT server:
Before the operation, verify that the CCCD has been changed to an appropriate value. Make sure that “BLE_GATTS_CLI_CNFG_NOTIFICATION (0x0001)” is written in the event “BLE_XXX_EVENT_YYY_CLI_CNFG_WRITE_COMP”, which is the event after the Write operation of CCCD. You can send “Handle Value Notification” by using the function “R_BLE_XXX_NotifyYYY()”. If the value of CCCD has not changed, the function “R_BLE_XXX_NotifyYYY()” returns the macro ”BLE_ERR_INVALID_OPERATION” and does not send ” Handle Value Notification” from GATT server. Notification operation is completed when the function “R_BLE_XXX_NotifyYYY()” is used. You can start following operation after this event.

GATT client:
Before the operation, it is necessary to change the value of CCCD to the appropriate value. Write “BLE_GATTS_CLI_CNFG_NOTIFICATION (0x0001)” to CCCD of characteristic which performs Notification operation. Data received in “Handle Value Notification” is notified to the application by the event “BLE_XXX_EVENT_YYY_HDL_VAL_NTF”. The data notified in this event is in form of a structure in Field of QE for BLE because decode function is used in Bluetooth LE Protocol Stack.

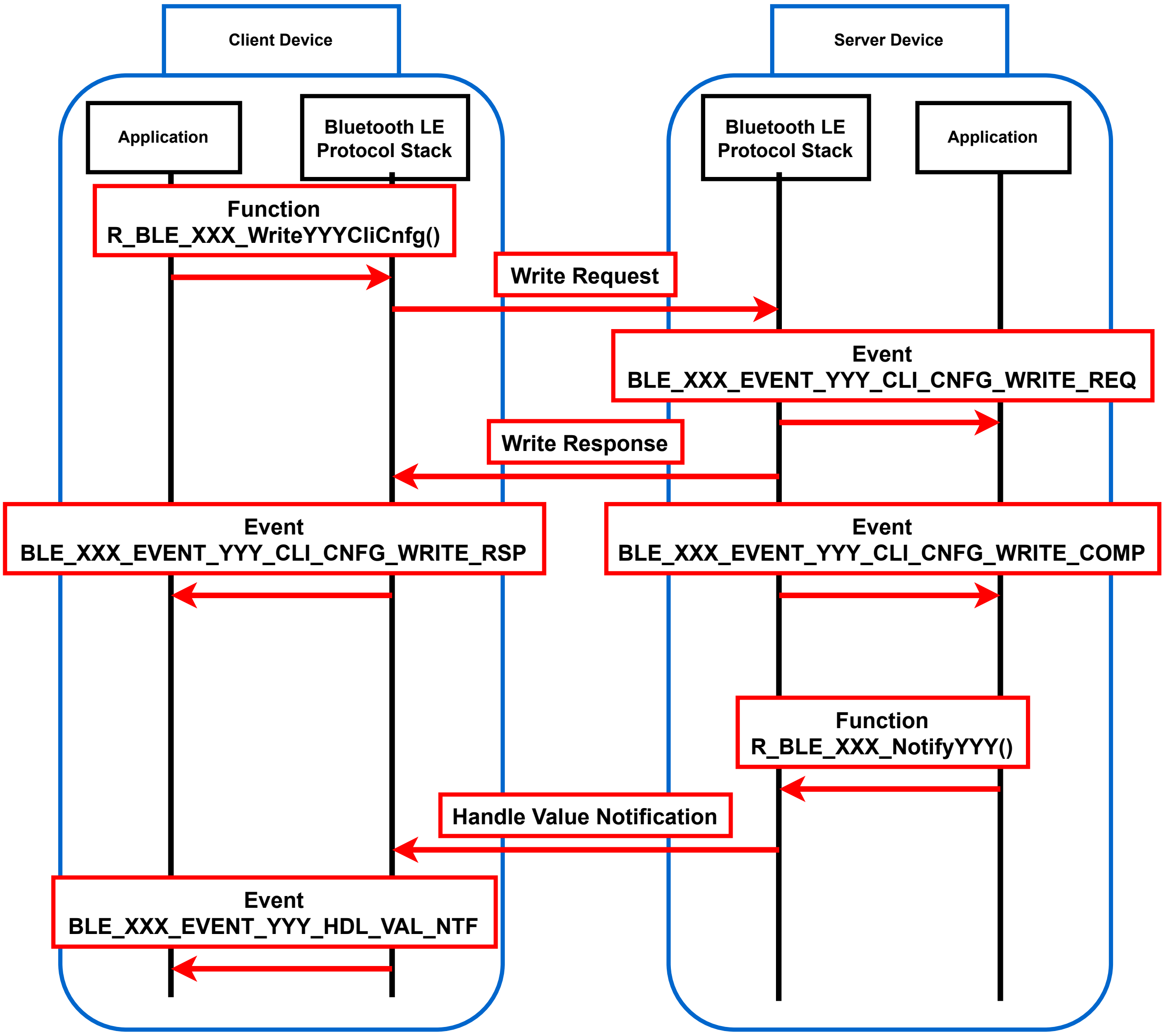

Client Device
Server Device
Application
Bluetooth LE Protocol Stack
Bluetooth LE Protocol Stack
Application
Function
R_BLE_XXX_WriteYYYCliCnfg()
Write Request
Event
BLE_XXX_EVENT_YYY_CLI_CNFG_WRITE_REQ
Write Response
Event
BLE_XXX_EVENT_YYY_CLI_CNFG_WRITE_RSP
Event
BLE_XXX_EVENT_YYY_CLI_CNFG_WRITE_COMP
Function
R_BLE_XXX_NotifyYYY()
Handle Value Notification
Event
BLE_XXX_EVENT_YYY_HDL_VAL_NTF

Figure 10-5 Flow of Notification operation

10.2.5 Indication operation

The Indication operation is a procedure to send data from GATT server to GATT client. For the Indication operation, the CCCD must have been added as a descriptor. The GATT client must also set the CCCD to the appropriate value before the operation. GATT server can verify that GATT client has received data sent from GATT server in a response from GATT client.

GATT server:
Before the operation, verify that the CCCD has been changed to appropriate value. Make sure that "BLE_GATTS_CLI_CNFG_INDICTION (0x0002)" is written in the event "BLE_XXX_EVENT_YYY_CLI_CNFG_WRITE_COMP", which is the event after the Write operation of CCCD. You can send "Handle Value Indication" by using the function "R_BLE_XXX_IndicateYYY()". If the value of the CCCD has not changed, the function "R_BLE_XXX_IndicateYYY()" returns the macro "BLE_ERR_INVALID_OPERATION" and does not send "Handle Value Indication" from GATT server. Indication operation is completed when the event "BLE_XXX_EVENT_YYY_HDL_VAL_CNF" is notified. You can start following operation after this event.

GATT client:
Before the operation, it is necessary to change the value of CCCD to the appropriate value. Write "BLE_GATTS_CLI_CNFG_INDICATION (0x0002)" to CCCD of characteristic which performs Indication operation. Data received in "Handle Value Indication" is notified to the application by the event "BLE_XXX_EVENT_YYY_HDL_VAL_IND". The data notified in this event is in form of a structure in Field of QE for BLE because decode function is used in Bluetooth LE Protocol Stack. After the event "BLE_XXX_EVENT_YYY_HDL_VAL_IND", Bluetooth LE Protocol Stack sends "Handle Value Confirmation", so you do not need to add any process to respond in application.

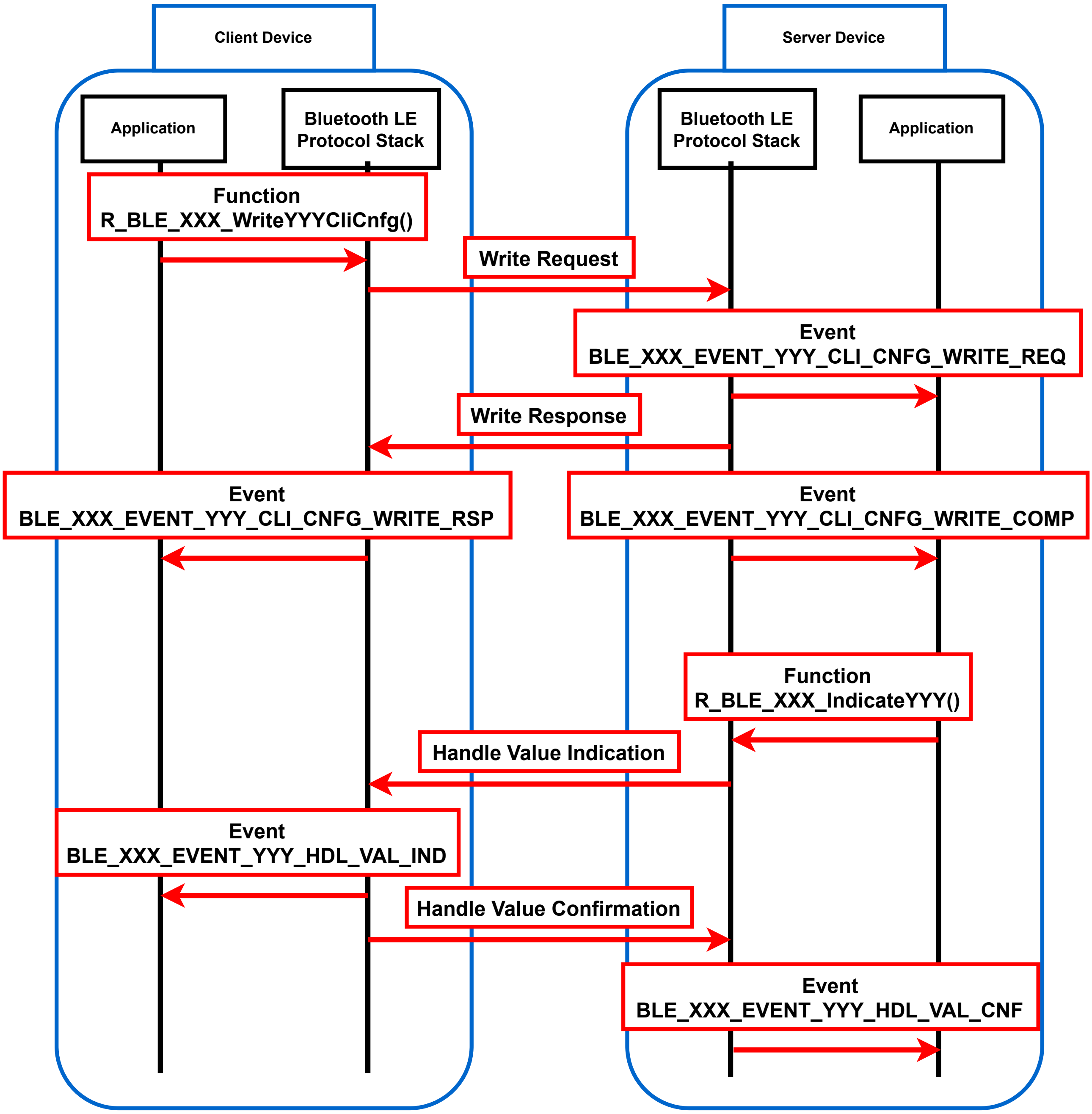

Client Device
Server Device
Application
Bluetooth LE Protocol Stack
Bluetooth LE Protocol Stack
Application
Function
R_BLE_XXX_WriteYYYCliCnfg()
Write Request
Event
BLE_XXX_EVENT_YYY_CLI_CNFG_WRITE_REQ
Write Response
Event
BLE_XXX_EVENT_YYY_CLI_CNFG_WRITE_RSP
Event
BLE_XXX_EVENT_YYY_CLI_CNFG_WRITE_COMP
Function
R_BLE_XXX_IndicateYYY()
Handle Value Indication
Event
BLE_XXX_EVENT_YYY_HDL_VAL_IND
Handle Value Confirmation
Event
BLE_XXX_EVENT_YYY_HDL_VAL_CNF

Figure 10-6 Flow of Indication operation

10.2.6 ReliableWrites operation

The ReliableWrites operation is a procedure to send data from GATT client to GATT server, ensure that the correct values are written, and then reflected it in the GATT database. There are two steps for ReliableWrites operation. In first step, GATT client sends data using “Prepare Write Request” and GATT server holds it in queue. GATT client can verify that the correct data is being written in “Prepare Write Response”. In second step, GATT server reflects the data held in queue in GATT database when receives “Execute Write Request”. Using this procedure is recommended when you want highly reliable data communication.
APIs of ReliableWrites operation is not included in the API of service generated from QE for BLE, so it must be implemented using APIs from Bluetooth LE Protocol Stack. In addition, Characteristic Extended Properties Descriptor must have been added as a descriptor for ReliableWrites operation.

GATT server:
Before the operation, reserve a queue for receiving data using function “R_BLE_GATTS_SetPrepareQueue()”. Size of the queue to be reserved should be greater than the total size of the characteristic which is able to ReliableWrites operation (if the total size is 6, specify value greater than or equal to 7). Data received in “Prepare Write Request” is notified to the application in the event “BLE_XXX_EVENT_YYY_WRITE_REQ”. The event “BLE_XXX_EVENT_YYY_WRITE_COMP” notifies the application that GATT server received “Execute Write Request” and data held in the queue is reflected in GATT database.

GATT client:
You can send “Prepare Write Request” using the function “R_BLE_GATTC_ReliableWrites()” in application. You can receive “Prepare Write Response” for each data transmitted, and you can check the data in the event “BLE_GATTC_EVENT_RELIABLE_WRITE_TX_COMP”. After verifying that GATT server is receiving the correct data, use the function “R_BLE_GATTC_ExecWrite()” to send “Execute Write Request” for reflecting data in GATT database. If confirmed data is incorrect, use the function “R_BLE_GATTC_ExecWrite()” to send “Execute Write Request” to discard the data held by GATT server.

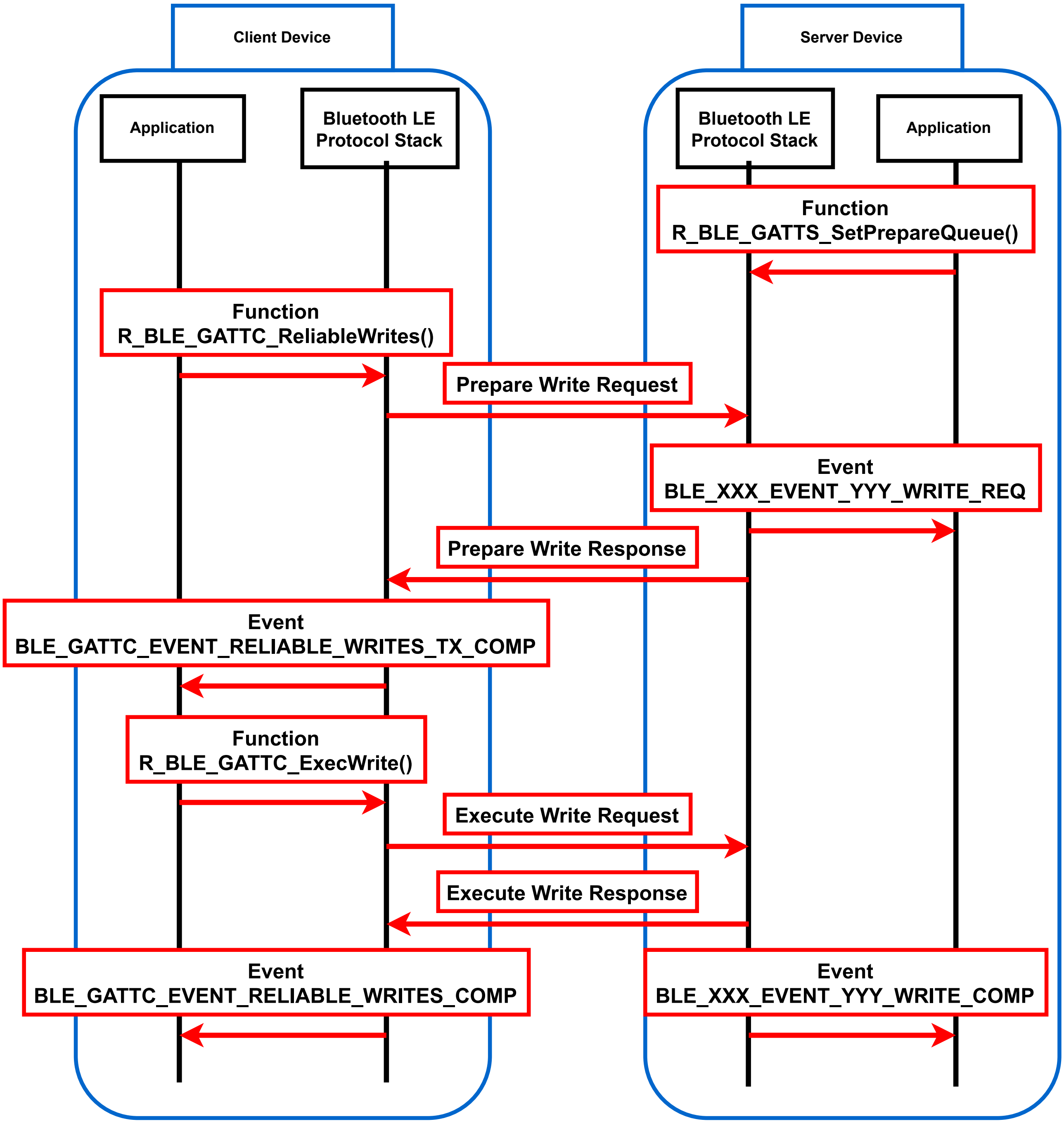

Client Device
Server Device
Application
Bluetooth LE Protocol Stack
Bluetooth LE Protocol Stack
Application
Function R_BLE_GATTS_SetPrepareQueue()
Function R_BLE_GATTC_ReliableWrites()
Prepare Write Request
Event BLE_XXX_EVENT_YYY_WRITE_REQ
Prepare Write Response
Event BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP
Function R_BLE_GATTC_ExecWrite()
Execute Write Request
Execute Write Response
Event BLE_GATTC_EVENT_RELIABLE_WRITES_COMP
Event BLE_XXX_EVENT_YYY_WRITE_COMP

Figure 10-7 Flow of ReliableWrites operation

10.2.7 Broadcast Operation

Broadcast operation is a procedure for transmitting data without connection to an unspecified number of devices. The sender device is called Broadcaster and uses the Advertising operation. The receiver device is called Observer and uses the Scan operation. Because of the communication without a connection, there is no limit in number of devices that the Broadcaster can communicate at once, but it cannot be guaranteed that the receiver device is receiving data.

APIs of Broadcast operation is not included in the API of service generated from QE for BLE, so it must be implemented using APIs from Bluetooth LE Protocol Stack. In addition, Server Characteristic Configuration Properties Descriptor must be added as a descriptor for Broadcast operation.

GATT server (Broadcaster):
Advertising operation is used for sending data. For an overview of advertising operation, refer to "5. Advertising".
Note that when Advertising as Broadcast operation, there are following limitations:

- For the advertising type specification (5.2.1.1), set adv_prop_type field with value indicated in "Non-Connectable and Non-Scannable Undirected" or "Non-Connectable and Non-Scannable Directed" in Table 5.1.
- For Advertising Data configuration (5.4), you can communicate service data by setting AD structure which has "service Data (0x16 for 16-bit UUIDs, 0x21 for 128-bit UUIDs)" for AD type and service UUIDs and data for AD data. If you want to configure AD structure with AD type of "Flags (0x01)", do not set "LE Limited Discoverable Mode" or "LE General Discoverable Mode".

GATT client (Observer):
Scan operation is used for receiving data. For an overview of scan operation, refer to "6. Scan". There are no restrictions on the scan operation but set scan parameters so that you can receive the Advertising Event sent by Broadcaster.

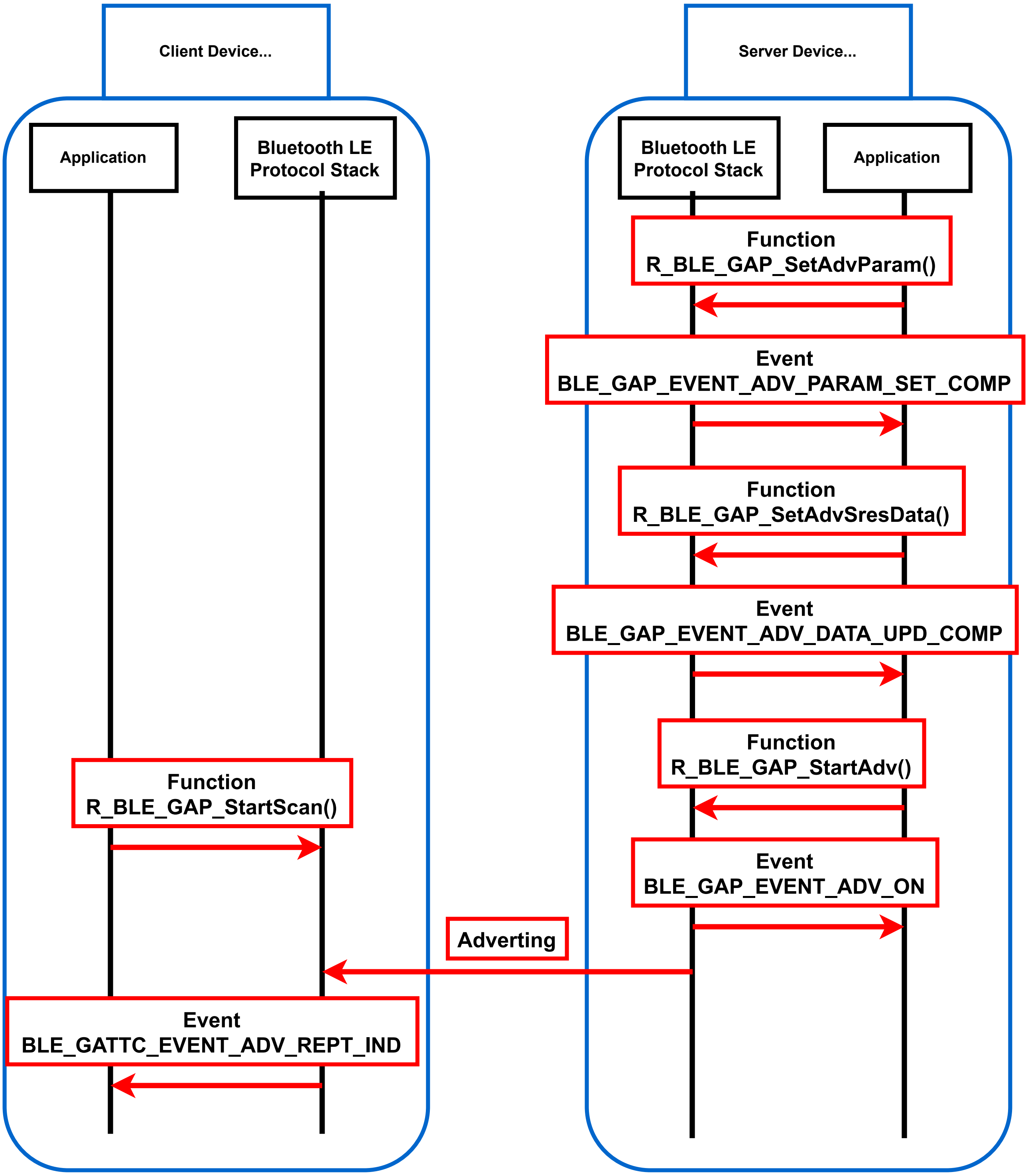

Client Device...
Application
Bluetooth LE Protocol Stack
Server Device...
Bluetooth LE Protocol Stack
Application
Function R_BLE_GAP_SetAdvParam()
Event BLE_GAP_EVENT_ADV_PARAM_SET_COMP
Function R_BLE_GAP_SetAdvSresData()
Event BLE_GAP_EVENT_ADV_DATA_UPD_COMP
Function R_BLE_GAP_StartAdv()
Function R_BLE_GAP_StartScan()
Event BLE_GAP_EVENT_ADV_ON
Adverting
Event BLE_GATTC_EVENT_ADV_REPT_IND

Figure 10-8 Flow of Broadcast operation

10.3 Example of using GATT Procedure

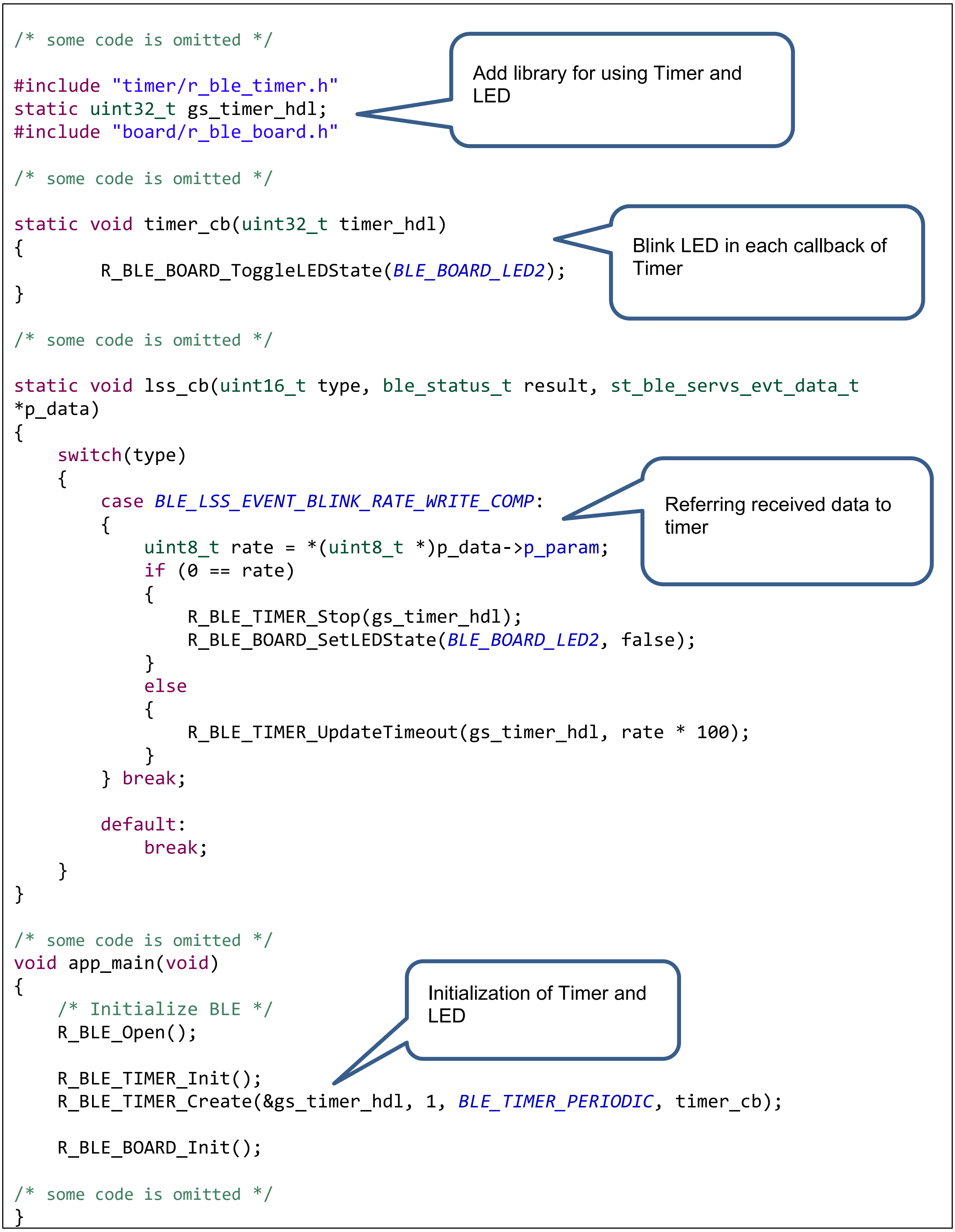
In this section, we will show how to implement GATT procedure in user application with use cases using LED Switch Service used in the demo application. **Table 10.3** shows the configuration of the LED Switch Service.

Table 10.3 Structure of LED Switch Service

Service	Characteristic	GATT Procedure
LED Switch Service **LSS**	LED Blink Rate	Read, Write
	Switch State	Notify

10.3.1 Example for sending data from GATT client

Use case: Change GATT server device's LED blink rate by pushing GATT client device's switch
Use LSS LED Blink Rate characteristic to change the blinking speed of the GATT server-side LED when the switch on the GATT client-side board is pressed. After the switch is pressed, GATT client uses Read operation to check the current LED Blink Rate value, and then uses Write operation to send the new value. The GATT server changes the LED Blink speed by using received value.



```
/* some code is omitted */

#include "timer/r_ble_timer.h"
static uint32_t gs_timer_hdl;
#include "board/r_ble_board.h"

/* some code is omitted */

static void timer_cb(uint32_t timer_hdl)
{
        R_BLE_BOARD_ToggleLEDState(BLE_BOARD_LED2);
}

/* some code is omitted */

static void lss_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t
*p_data)
{
    switch(type)
    {
        case BLE_LSS_EVENT_BLINK_RATE_WRITE_COMP:
        {
            uint8_t rate = *(uint8_t *)p_data->p_param;
            if (0 == rate)
            {
                R_BLE_TIMER_Stop(gs_timer_hdl);
                R_BLE_BOARD_SetLEDState(BLE_BOARD_LED2, false);
            }
            else
            {
                R_BLE_TIMER_UpdateTimeout(gs_timer_hdl, rate * 100);
            }
        } break;

        default:
            break;
    }
}

/* some code is omitted */
void app_main(void)
{
    /* Initialize BLE */
    R_BLE_Open();

    R_BLE_TIMER_Init();
    R_BLE_TIMER_Create(&gs_timer_hdl, 1, BLE_TIMER_PERIODIC, timer_cb);

    R_BLE_BOARD_Init();

/* some code is omitted */
}
```

Add library for using Timer and LED

Blink LED in each callback of Timer

Referring received data to timer

Initialization of Timer and LED

Code 10-1 Implementation in app_main.c for GATT server

```
/* some code is omitted */

#include "board/r_ble_board.h"
#define LED_RATE_LOW  (0x01)
#define LED_RATE_HIGH (0xff)

/* some code is omitted */

static void sw_cb(void)
{
    R_BLE_LSC_ReadBlinkRate(g_conn_hdl);
}

/* some code is omitted */

static void lsc_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t
*p_data)
{
    switch(type)
    {
        case BLE_LSC_EVENT_BLINK_RATE_READ_RSP:
        {
            uint8_t read_rate = *(uint8_t *)p_data->p_param;
            uint8_t write_rate = 0;
            if (LED_RATE_LOW == read_rate)
            {
                write_rate = LED_RATE_HIGH;
            }
            else
            {
                write_rate = LED_RATE_LOW;
            }

            R_BLE_LSC_WriteBlinkRate(g_conn_hdl, &write_rate);
        } break;

        default:
            break;
    }
}

/* some code is omitted */

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


    R_BLE_BOARD_Init();
    R_BLE_BOARD_RegisterSwitchCb(BLE_BOARD_SW2, sw_cb);

/* some code is omitted */
}
```

Add library for using switch

Start Read operation in callback of switch input

Start Write operation depending on received value

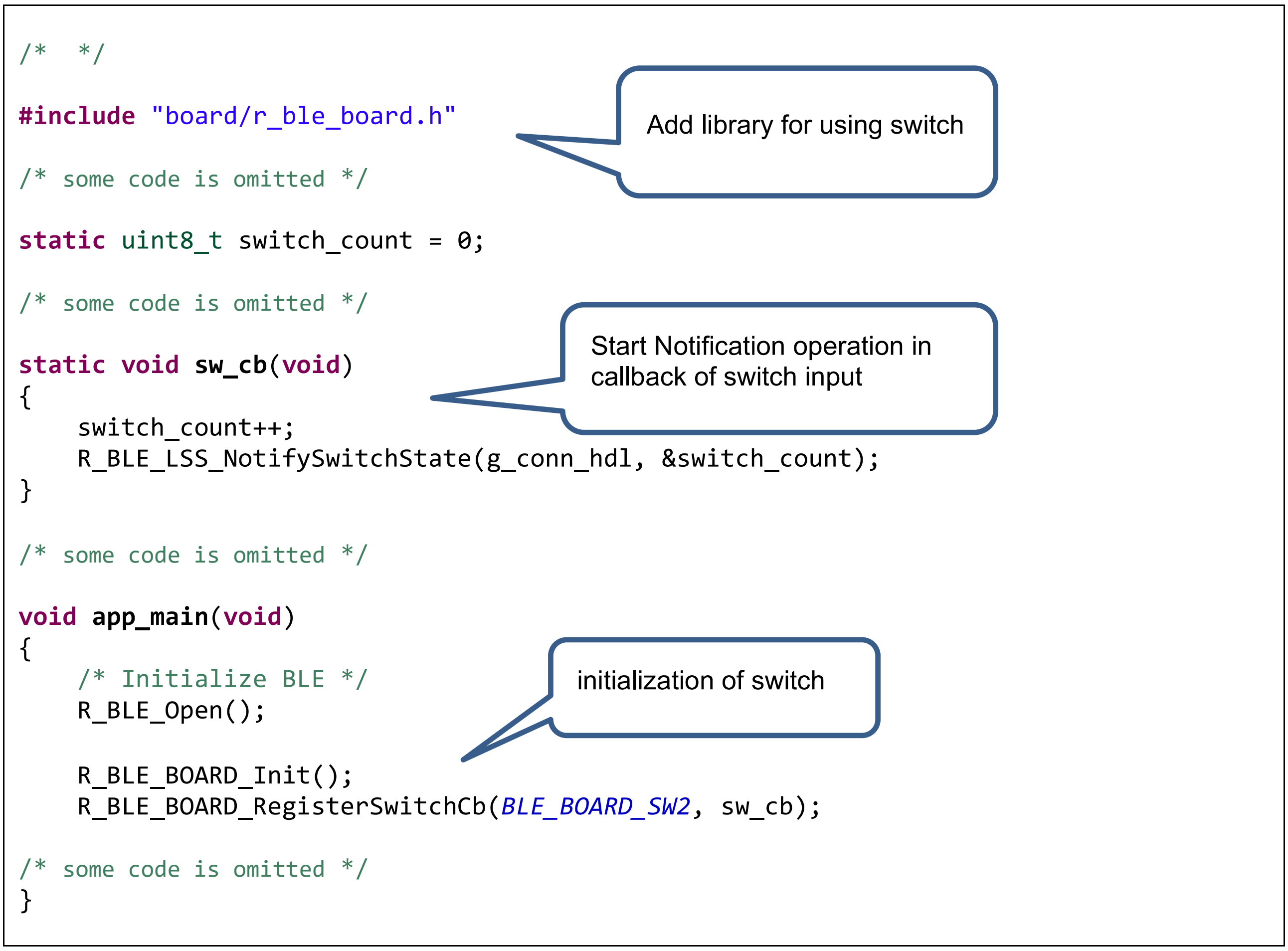
Initialization of switch

Code 10-2 Implementation in app_main.c for GATT client

10.3.2 Example for sending data from GATT server

Use case: Blink GATT client device’s LED by pressing GATT server device’s switch

Blink the GATT client-side LED using LSS Switch State characteristic each time a switch on the GATT server-side board is pressed. GATT server sends the number of times it was pressed using the Notification operation each time the switch is pressed. The GATT client side lights up when received value is odd number and turns off received value is even number.



```
/*  */

#include "board/r_ble_board.h"

/* some code is omitted */

static uint8_t switch_count = 0;

/* some code is omitted */

static void sw_cb(void)
{
    switch_count++;
    R_BLE_LSS_NotifySwitchState(g_conn_hdl, &switch_count);
}

/* some code is omitted */

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

    R_BLE_BOARD_Init();
    R_BLE_BOARD_RegisterSwitchCb(BLE_BOARD_SW2, sw_cb);

/* some code is omitted */
}
```

Add library for using switch

Start Notification operation in callback of switch input

initialization of switch

Code 10-3 Implementation in app_main.c for GATT server

```
/* some code is omitted */

#include "board/r_ble_board.h"
```

Add library for using LED

```
/* some code is omitted */

static void lsc_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t
*p_data)
{
    switch(type)
    {
        case BLE_LSC_EVENT_SWITCH_STATE_HDL_VAL_NTF:
        {
            uint8_t ntf_state = *(uint8_t *)p_data->p_param;
            if (ntf_state % 2 == 0)
            {
                R_BLE_BOARD_SetLEDState(BLE_BOARD_LED2, false);
            }
            else
            {
                R_BLE_BOARD_SetLEDState(BLE_BOARD_LED2, true);
            }

        } break;

        default:
            break;
    }
}
```

Blink LED depending on received value

```
/* some code is omitted */

static void disc_comp_cb(uint16_t conn_hdl)
{
    /* TODO: Add function after discovery completed */
    static uint16_t s_cccd_req;
    s_cccd_req = BLE_GATTS_CLI_CNFG_NOTIFICATION;
    R_BLE_LSC_WriteSwitchStateCliCnfg(g_conn_hdl, &s_cccd_req);
    return;
}
```

Write CCCD after discovery is completed

```
/* some code is omitted */

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

    R_BLE_BOARD_Init();

/* some code is omitted */

}
```

Initialization of LED

Code 10-4 Implementation in app_main.c for GATT client

11. Debugging

GATT Server application needs to confirm Advertising, Connection, GATT database, Indication, Notification, Read Response, Write Response. Beacon Scanning and Data Comm Master of BTTS, and GATT Browser are available.

The GATT Client application needs to confirm Scan, Connection, Service Discovery, Read Request, Write Request, and Confirmation. Beacon Advertising and Data Comm Slave of BTTS are available.

Note: Not all functions can be evaluated with GATT Browser or BTTS.

Logger function is available for application survey. Using Logger function enables to output logs to the debug console on e^{2}studio or IAR.

As for GATT Browser, refer to "GATTBrowser for Android Smartphone Application Instruction manual (R01AN3802)" or "GATTBrowser for iOS Smartphone Application Instruction manual (R21AN0017)".

As for BTTS, refer to "Bluetooth Test Tool Suite operating instructions (R01AN4554)". As for Logger function details, refer to "5.2 Logger" in "Bluetooth Low Energy Protocol Stack Basic Package: User's Manual (R01UW0205)".

11.1 Using Logger function

If changing BLE_DEFAULT_LOG_LEVEL or BLE_LOG_LEVEL before including r_ble_logger.h, the log level can be changed. If both are defined, the one defined last will be adopted. If either of them is set to 0, the log output will be disabled. If the log level is set as 1, BLE_LOG_ERR, if set as 2, BLE_LOG_ERR / BLE_LOG_WRN, if set as 3, BLE_LOG_ERR / BLE_LOG_WRN / BLE_LOG_DBG macro functions are enabled, if setting as 4 or more and using BLE_LOG macro function, the log level can be expanded.

If changing BLE_LOG_TAG before including r_ble_logger.h, the log tag can be extended.

The following is an example of code that extends the log level and checks arguments of R_BLE_ABS_StartLegacyAdv. Logger function is used in app_main.c and the newly created source file (appapp.c).

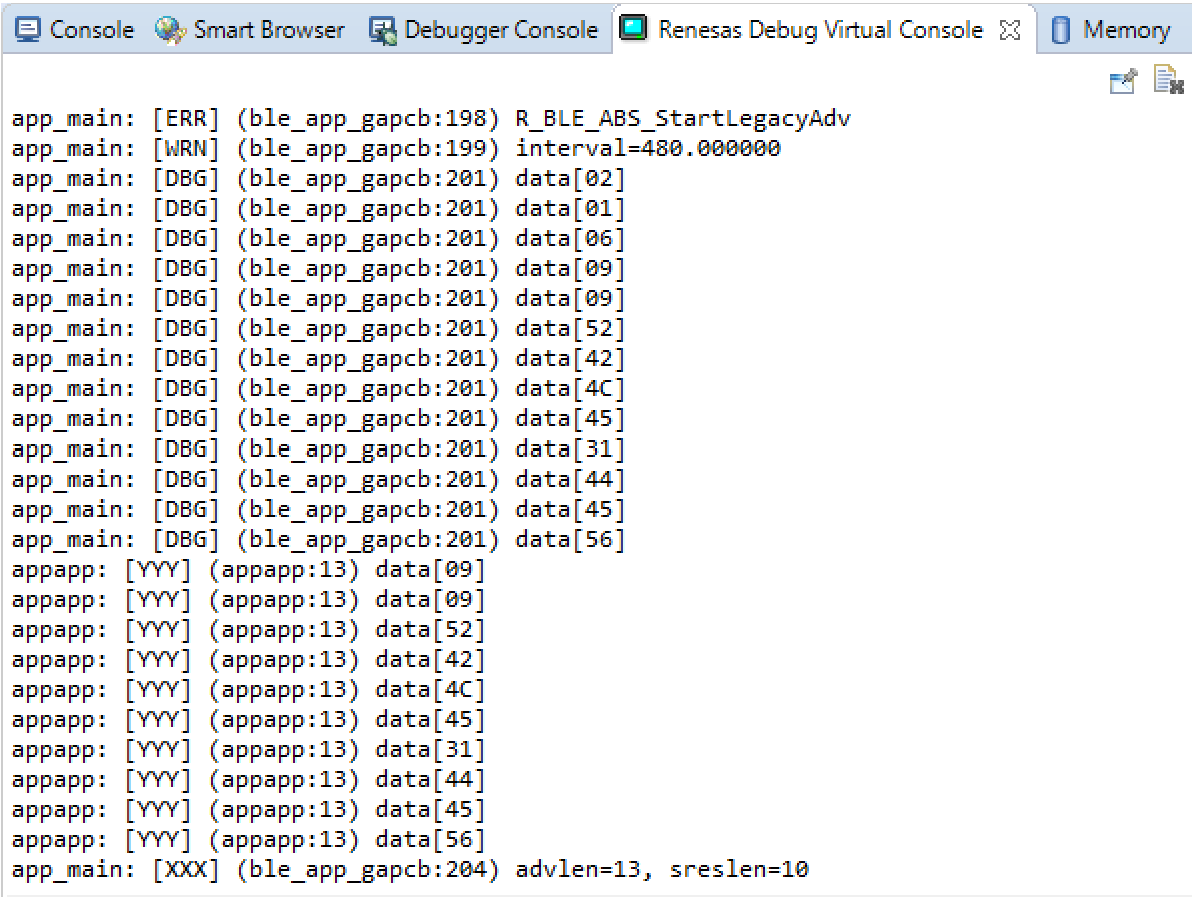
```
[app_main.c]
#define BLE_DEFAULT_LOG_LEVEL (1)
#define BLE_LOG_LEVEL (4)
#define BLE_LOG_TAG "app_main"
#include "logger/r_ble_logger.h"
#define BLE_LOG_XXX(...) BLE_LOG(4, "XXX", __VA_ARGS__)
extern void appapp( void );

//static st_ble_abs_legacy_adv_param_t gs_adv_param =
st_ble_abs_legacy_adv_param_t gs_adv_param =
(OMISSION)

    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {

            BLE_LOG_ERR("R_BLE_ABS_StartLegacyAdv");
            BLE_LOG_WRN("interval=%f", gs_adv_param.slow_adv_intv * 0.625 );
            for( int i=0; i<gs_adv_param.adv_data_length; i++ ){
                BLE_LOG_DBG("data[%02X]", gs_adv_param.p_adv_data[i] );
            }
            appapp();
            BLE_LOG_XXX("advlen=%d, sreslen=%d", gs_adv_param.adv_data_length,
gs_adv_param.sres_data_length );
            R_BLE_ABS_StartLegacyAdv(&gs_adv_param);
(OMISSION)
```

Code 11-1 Code example for checking arguments of R_BLE_ABS_StartLegacyAdv (app_main.c)

```
[appapp.c]
#include "r_ble_rx23w_if.h"
#include "abs/r_ble_abs_api.h"
#define BLE_DEFAULT_LOG_LEVEL (1)
#define BLE_LOG_LEVEL (5)
#define BLE_LOG_TAG "appapp"
#include "logger/r_ble_logger.h"
#define BLE_LOG_YYY(...) BLE_LOG(5, "YYY", __VA_ARGS__)
extern st_ble_abs_legacy_adv_param_t gs_adv_param;

void appapp( void )
{
        for( int i=0; i<gs_adv_param.sres_data_length; i++ ){
                BLE_LOG_YYY("data[%02X]", gs_adv_param.p_sres_data[i] );
        }
}
```

Code 11-2 Code example for checking arguments of R_BLE_ABS_StartLegacyAdv (appapp.c)

Logs of Logger function are displayed in [Renesas Views] → [Debug] → [Renesas Debug Virtual Console] on e^{2}studio. One line is displayed by one logger call, therefore line breaks are not required. The logs displayed in the debug console are not cleared automatically at the next debug execution. Clear by [Clear] from right-click.



```
app_main: [ERR] (ble_app_gapcb:198) R_BLE_ABS_StartLegacyAdv
app_main: [WRN] (ble_app_gapcb:199) interval=480.000000
app_main: [DBG] (ble_app_gapcb:201) data[02]
app_main: [DBG] (ble_app_gapcb:201) data[01]
app_main: [DBG] (ble_app_gapcb:201) data[06]
app_main: [DBG] (ble_app_gapcb:201) data[09]
app_main: [DBG] (ble_app_gapcb:201) data[09]
app_main: [DBG] (ble_app_gapcb:201) data[52]
app_main: [DBG] (ble_app_gapcb:201) data[42]
app_main: [DBG] (ble_app_gapcb:201) data[4C]
app_main: [DBG] (ble_app_gapcb:201) data[45]
app_main: [DBG] (ble_app_gapcb:201) data[31]
app_main: [DBG] (ble_app_gapcb:201) data[44]
app_main: [DBG] (ble_app_gapcb:201) data[45]
app_main: [DBG] (ble_app_gapcb:201) data[56]
appapp: [YYY] (appapp:13) data[09]
appapp: [YYY] (appapp:13) data[09]
appapp: [YYY] (appapp:13) data[52]
appapp: [YYY] (appapp:13) data[42]
appapp: [YYY] (appapp:13) data[4C]
appapp: [YYY] (appapp:13) data[45]
appapp: [YYY] (appapp:13) data[31]
appapp: [YYY] (appapp:13) data[44]
appapp: [YYY] (appapp:13) data[45]
appapp: [YYY] (appapp:13) data[56]
app_main: [XXX] (ble_app_gapcb:204) advlen=13, sreslen=10
```

Figure 11-1 Logs displayed by Logger function

11.2 Using Command line function

Enable Command line function and code-generate, referring to "1.6.1 Primary functions". The code for using the standard Command line function built into the library is below.

```
[app_main.c]
#include "cli/r_ble_cli.h"
#include "cmd/r_ble_cmd_abs.h"
#include "cmd/r_ble_cmd_vs.h"
#include "cmd/r_ble_cmd_sys.h"
/* CommandLine parameters */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_abs_cmd,
    &g_vs_cmd,
    &g_sys_cmd,
    &g_ble_cmd
};

(OMISSION)

static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    R_BLE_CMD_AbsGapCb(type, result, p_data);
(OMISSION)

static void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
    R_BLE_CMD_VsCb(type, result, p_data);
(OMISSION)

void app_main(void)
{
    (OMISSION)
    /* Configure CommandLine */
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));
    R_BLE_CMD_SetResetCb(ble_app_init);
    (OMISSION)

    /* main loop */
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
        (OMISSION)
```

Code 11-3 Example of using the command line function

From the terminal, the input example of Scan → Connect → Disconnect is below.

```
$
$ gap scan 0x09 0x52
74:90:50:FF:FF:FF pub ff 0000
74:90:50:FF:FF:FF pub ff 0000
74:90:50:FF:FF:FF pub ff 0000

$ receive BLE_GAP_EVENT_SCAN_OFF result : 0x0000

$ gap conn 74:90:50:ff:ff:ff pub
receive BLE_GAP_EVENT_CONN_IND result : 0x0000
gap: connected conn_hdl:0x0020, addr:74:90:50:FF:FF:FF pub

$ receive BLE_GAP_EVENT_DATA_LEN_CHG result : 0x0000, conn_hdl : 0x0020
tx_octets : 0x00fb
tx_time   : 0x0848
rx_octets : 0x00fb
rx_time   : 0x0848

$ gap disconn 0x20
$ receive BLE_GAP_EVENT_DISCONN_IND result : 0x0000
gap: disconnected conn_hdl:0x0020, addr:74:90:50:FF:FF:FF pub, reason:0x16

$
$
```

Only Advertising that includes data whose Complete Local Name (0x09) starts with "R" (0x52) will be scanned.
Note: When "gap scan 0x09 0x52,0x42" is specified, only Advertising that includes data whose Complete Local Name (0x09) starts with "RB" will be scanned.
Note: The gap scan command is stopped by pressing [Ctrl]+[c] or gap scan stop.

Specify the BD address and address type to connect.

Specify the connection handle and disconnect.

11.3 Using RF communication timing notification function

The sample displaying logs with "rf log on" command to check the RF communication timing is below. This sample uses Command line function and RF communication timing notification function. Enable these functions and code-generate, referring to "1.6.1 Primary functions" and "3.11 RF communication timing". Newly create r_ble_cmd_rf.h in “src” folder.

```
[src\r_ble_cmd_rf.h]
#include "r_ble_rx23w_if.h"
#include "cli/r_ble_cli.h"

#ifndef R_BLE_CMD_RF_H_
#define R_BLE_CMD_RF_H_

typedef struct
{
    uint32_t elapsed_time;
    uint16_t event_type;
    uint16_t event_data;
    uint8_t start_end;
} st_ble_rf_log_t;

#define BLE_RF_LOG_NUM_MAX 1000
extern st_ble_rf_log_t gs_rf_log[BLE_RF_LOG_NUM_MAX];
extern uint32_t gs_rf_log_idx;
extern uint32_t gs_timer_elapsed_time;
extern const st_ble_cli_cmd_t g_rf_cmd;
extern void save_rf_log( uint16_t event_type, uint16_t event_data, uint8_t start_end );

#endif /* R_BLE_CMD_RF_H_ */
```

Code 11-4 Sample to display log of RF communication timing (r_ble_cmd_rf.h)

Newly create r_ble_cmd_rf.c in “src” folder.

```
[src\r_ble_cmd_rf.c]
#include "r_ble_rx23w_if.h"
#include "cmd/r_ble_cmd.h"
#include "r_ble_cmd_rf.h"

#if (BLE_CFG_CMD_LINE_EN == 1) && (BLE_CFG_HCI_MODE_EN == 0)

#define pf R_BLE_CLI_Printf
st_ble_rf_log_t gs_rf_log[BLE_RF_LOG_NUM_MAX];
uint32_t gs_rf_log_idx = 0;
uint32_t gs_timer_elapsed_time;

void save_rf_log( uint16_t event_type, uint16_t event_data, uint8_t start_end )
{
    gs_rf_log[gs_rf_log_idx].elapsed_time = gs_timer_elapsed_time;
    gs_rf_log[gs_rf_log_idx].event_type = event_type;
    gs_rf_log[gs_rf_log_idx].event_data = event_data;
    gs_rf_log[gs_rf_log_idx].start_end = start_end;
    gs_rf_log_idx++;
    if( gs_rf_log_idx >= BLE_RF_LOG_NUM_MAX ){
        gs_rf_log_idx = 0;
    }
}

static void show_rf_log( uint32_t elapsed_time, uint16_t event_type, uint16_t event_data, uint8_t
start_end )
{
        switch( event_type )
        {
                case 0x0000:/*BLE_EVENT_TYPE_CONN*/
                {
                        if( start_end == 1 ){ pf("%010d,ConnS,%d\n", elapsed_time, event_data ); }
                        if( start_end == 2 ){ pf("%010d,ConnE,%d\n", elapsed_time, event_data ); }
                } break;
                case 0x0001:/*BLE_EVENT_TYPE_ADV*/
                {
                        if( start_end == 1 ){ pf("%010d,AdvS,%d\n", elapsed_time, event_data ); }
                        if( start_end == 2 ){ pf("%010d,AdvE,%d\n", elapsed_time, event_data ); }
                } break;
                case 0x0002:/*BLE_EVENT_TYPE_SCAN*/
                {
                        if( start_end == 1 ){ pf("%010d,ScanS,%d\n", elapsed_time, event_data ); }
                        if( start_end == 2 ){ pf("%010d,ScanE,%d\n", elapsed_time, event_data ); }
                } break;
                case 0x0003:/*BLE_EVENT_TYPE_INITIATOR*/
                {
                        if( start_end == 1 ){ pf("%010d,InitS,%d\n", elapsed_time, event_data ); }
                        if( start_end == 2 ){ pf("%010d,InitE,%d\n", elapsed_time, event_data ); }
                } break;
                case 0x0004:/*BLE_EVENT_TYPE_RF_DS_START*//*BLE_EVENT_TYPE_RF_DS_CLOSE*/
                {
                        if( start_end == 1 ){ pf("%010d,SleepS,%d\n", elapsed_time, event_data ); }
                        if( start_end == 2 ){ pf("%010d,SleepE,%d\n", elapsed_time, event_data ); }
                } break;
                default:
                {
                } break;
        }
}

static void exec_rf_log(int argc, char *argv[])
{
    ble_status_t status;
    if (strcmp(argv[1], "on") == 0)
    {
        R_BLE_CLI_Printf("time,type,data\n");
        for(int i=0; i<BLE_RF_LOG_NUM_MAX; i++){
                show_rf_log( gs_rf_log[i].elapsed_time, gs_rf_log[i].event_type, gs_rf_log[i].event_data,
gs_rf_log[i].start_end );
        }
    }
    else
    {
        pf("rf %s: unrecognized operands\n", argv[0]);
    }
}

static const st_ble_cli_cmd_t rf_log_cmd = {
    .p_name = "log",
    .exec   = exec_rf_log,
    .p_help = "Usage: rf log (on)\n"
              "Show rf_event or not",
};

static const st_ble_cli_cmd_t * const rf_sub_cmds[] = {
    &rf_log_cmd,
};

const st_ble_cli_cmd_t g_rf_cmd = {
    .p_name      = "rf",
    .p_cmds      = rf_sub_cmds,
    .num_of_cmds = ARRAY_SIZE(rf_sub_cmds),
    .p_help      = "Sub Command: log\n"
                   "Try 'rf sub-command help' for more information",
};

const st_ble_cli_cmd_t g_rf_cmd;

#endif /* (BLE_CFG_CMD_LINE_EN == 1) && (BLE_CFG_HCI_MODE_EN == 0) */
```

Code 11-5 Sample to display log of RF communication timing (r_ble_cmd_rf.c)

The following code saves the RF communication timing notification as logs.

```
[src\smc_gen\r_ble_rx23w\src\platform\r_ble_pf_functions.c]
extern uint32_t gs_timer_elapsed_time;
#include "../../../../../src/r_ble_cmd_rf.h"

BLE_SECTION_P void r_ble_rf_notify_event_start(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( (uint16_t)(param>>16) )
    {
        case 0x0000:/*BLE_EVENT_TYPE_CONN*/
        {
            save_rf_log( BLE_EVENT_TYPE_CONN, 0x0000, 0x01 );
        } break;
        case 0x0001:/*BLE_EVENT_TYPE_ADV*/
        {
            save_rf_log( BLE_EVENT_TYPE_ADV, 0x0000, 0x01 );
        } break;
        case 0x0002:/*BLE_EVENT_TYPE_SCAN*/
        {
            save_rf_log( BLE_EVENT_TYPE_SCAN, 0x0000, 0x01 );
        } break;
        case 0x0003:/*BLE_EVENT_TYPE_INITIATOR*/
        {
            save_rf_log( BLE_EVENT_TYPE_INITIATOR, 0x0000, 0x01 );
        } break;
    }
}
BLE_SECTION_P void r_ble_rf_notify_event_close(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( (uint16_t)(param>>16) )
    {
        case 0x0000:/*BLE_EVENT_TYPE_CONN*/
        {
            save_rf_log( BLE_EVENT_TYPE_CONN, 0x0000, 0x02 );
        } break;
        case 0x0001:/*BLE_EVENT_TYPE_ADV*/
        {
            save_rf_log( BLE_EVENT_TYPE_ADV, 0x0000, 0x02 );
        } break;
        case 0x0002:/*BLE_EVENT_TYPE_SCAN*/
        {
            save_rf_log( BLE_EVENT_TYPE_SCAN, 0x0000, 0x02 );
        } break;
        case 0x0003:/*BLE_EVENT_TYPE_INITIATOR*/
        {
            save_rf_log( BLE_EVENT_TYPE_INITIATOR, 0x0000, 0x02 );
        } break;
    }
}

BLE_SECTION_P void r_ble_rf_notify_deep_sleep(uint32_t param)
{
    /* Note: Do not processing long time here. */
    switch( param )
    {
        case BLE_EVENT_TYPE_RF_DS_START:
        {
            save_rf_log( 0x0004, 0x0000, 0x01 );
        } break;
        case BLE_EVENT_TYPE_RF_DS_CLOSE:
        {
            save_rf_log( 0x0004, 0x0000, 0x02 );
        } break;
    }
}
```

Code 11-6 Sample to display RF communication timing log (save log)

The following code uses Software timer and increments the timer count in 1ms cycles.

```
[app_main.c]
#include "timer/r_ble_timer.h"
/* timer handle */
static uint32_t gs_timer_hdl;

#include "cli/r_ble_cli.h"
#include "../../../src/r_ble_cmd_rf.h"
/* CommandLine parameters */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_rf_cmd,
};

(OMISSION)

static void timer_cb(uint32_t timer_hdl)
{
    gs_timer_elapsed_time++;
}

void app_main(void)
{
    (OMISSION)
    /* Initialize timer */
    R_BLE_TIMER_Init();
    /* Create timer */
    gs_timer_hdl = BLE_TIMER_INVALID_HDL;
    gs_timer_elapsed_time = 0;
    R_BLE_TIMER_Create(&gs_timer_hdl, 1, BLE_TIMER_PERIODIC, timer_cb);
    R_BLE_TIMER_Start(gs_timer_hdl);
    /* Configure CommandLine */
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));
    R_BLE_CMD_SetResetCb(ble_app_init);
    while (1)
    {
        /* Process Command Line */
        R_BLE_CLI_Process();
    (OMISSION)
```

Code 11-7 Sample to display RF communication timing log (timer count increment)

When inputting "rf log on" command, the following logs will be outputted. If inputting "rf log on" command during connection, the loop processing for log output occupies CPU and does not finish its processing within RF idle time, therefore connection may not be maintained. For the outline of MCU/RF operation, refer to "1.5 Bluetooth LE Protocol Stack Operation Overview".

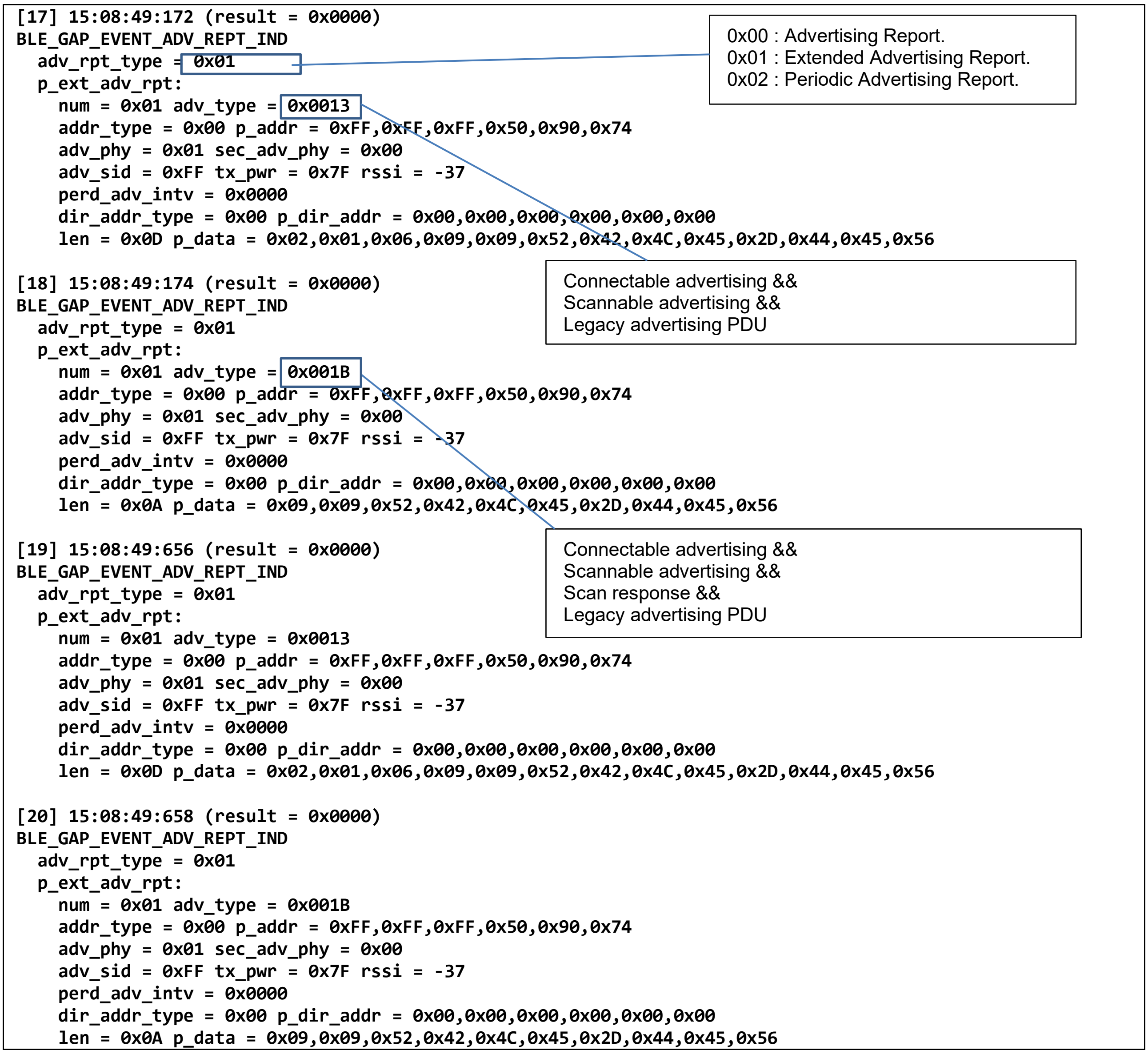
```
[Log of Advertising→Connection]
0000019851,AdvS,0
0000019854,AdvE,0
0000019854,SleepS,0
0000020286,SleepE,0
0000020289,AdvS,0
0000020292,AdvE,0
0000020293,SleepS,0
0000020728,SleepE,0
0000020731,AdvS,0
0000021069,ConnS,0
0000021070,ConnE,0
0000021392,ConnS,0
0000021394,ConnE,0
0000021715,ConnS,0
0000021715,ConnE,0
0000022038,ConnS,0
0000022038,ConnE,0
0000022360,ConnS,0
0000022361,ConnE,0
0000022683,ConnS,0
0000022684,ConnE,0
0000022686,SleepS,0
0000023025,SleepE,0
0000023028,ConnS,0
0000023029,ConnE,0
0000023029,SleepS,0
0000023370,SleepE,0
0000023373,ConnS,0
0000023374,ConnE,0
```

```
[Log of Scan→Connection]
0000002629,ScanS,0
0000002776,ScanE,0
0000002776,SleepS,0
0000002918,SleepE,0
0000002920,ScanS,0
0000003067,ScanE,0
0000003067,SleepS,0
0000003209,SleepE,0
0000003211,ScanS,0
0000003234,InitS,0
0000003261,InitE,0
0000003287,InitS,0
0000003314,InitE,0
0000003341,InitS,0
0000003368,InitE,0
0000003395,InitS,0
0000003442,ConnS,0
0000003442,ConnE,0
0000003761,ConnS,0
0000003763,ConnE,0
0000004081,ConnS,0
0000004082,ConnE,0
0000004401,ConnS,0
0000004402,ConnE,0
0000004405,SleepS,0
0000004734,SleepE,0
0000004736,ConnS,0
0000004737,ConnE,0
0000004737,SleepS,0
0000005080,SleepE,0
```

11.4 Checking Server operation

11.4.1 Using BTTS Beacon Scanning

Using Beacon Scanning enables to output the Advertising reception status as logs from Peripheral. In the example below, Advertising where Advertising Interval is 480 ms is received. It can be seen being received at intervals of 484 ms from 49 seconds 172 to 49 seconds 656. It can be also see receiving Scan response data after each Advertising.



```
[17] 15:08:49:172 (result = 0x0000)
BLE_GAP_EVENT_ADV_REPT_IND
  adv_rpt_type = 0x01
  p_ext_adv_rpt:
    num = 0x01 adv_type = 0x0013
    addr_type = 0x00 p_addr = 0xFF,0xFF,0xFF,0x50,0x90,0x74
    adv_phy = 0x01 sec_adv_phy = 0x00
    adv_sid = 0xFF tx_pwr = 0x7F rssi = -37
    perd_adv_intv = 0x0000
    dir_addr_type = 0x00 p_dir_addr = 0x00,0x00,0x00,0x00,0x00,0x00
    len = 0x0D p_data = 0x02,0x01,0x06,0x09,0x09,0x52,0x42,0x4C,0x45,0x2D,0x44,0x45,0x56

[18] 15:08:49:174 (result = 0x0000)
BLE_GAP_EVENT_ADV_REPT_IND
  adv_rpt_type = 0x01
  p_ext_adv_rpt:
    num = 0x01 adv_type = 0x001B
    addr_type = 0x00 p_addr = 0xFF,0xFF,0xFF,0x50,0x90,0x74
    adv_phy = 0x01 sec_adv_phy = 0x00
    adv_sid = 0xFF tx_pwr = 0x7F rssi = -37
    perd_adv_intv = 0x0000
    dir_addr_type = 0x00 p_dir_addr = 0x00,0x00,0x00,0x00,0x00,0x00
    len = 0x0A p_data = 0x09,0x09,0x52,0x42,0x4C,0x45,0x2D,0x44,0x45,0x56

[19] 15:08:49:656 (result = 0x0000)
BLE_GAP_EVENT_ADV_REPT_IND
  adv_rpt_type = 0x01
  p_ext_adv_rpt:
    num = 0x01 adv_type = 0x0013
    addr_type = 0x00 p_addr = 0xFF,0xFF,0xFF,0x50,0x90,0x74
    adv_phy = 0x01 sec_adv_phy = 0x00
    adv_sid = 0xFF tx_pwr = 0x7F rssi = -37
    perd_adv_intv = 0x0000
    dir_addr_type = 0x00 p_dir_addr = 0x00,0x00,0x00,0x00,0x00,0x00
    len = 0x0D p_data = 0x02,0x01,0x06,0x09,0x09,0x52,0x42,0x4C,0x45,0x2D,0x44,0x45,0x56

[20] 15:08:49:658 (result = 0x0000)
BLE_GAP_EVENT_ADV_REPT_IND
  adv_rpt_type = 0x01
  p_ext_adv_rpt:
    num = 0x01 adv_type = 0x001B
    addr_type = 0x00 p_addr = 0xFF,0xFF,0xFF,0x50,0x90,0x74
    adv_phy = 0x01 sec_adv_phy = 0x00
    adv_sid = 0xFF tx_pwr = 0x7F rssi = -37
    perd_adv_intv = 0x0000
    dir_addr_type = 0x00 p_dir_addr = 0x00,0x00,0x00,0x00,0x00,0x00
    len = 0x0A p_data = 0x09,0x09,0x52,0x42,0x4C,0x45,0x2D,0x44,0x45,0x56
```

0x00 : Advertising Report.
0x01 : Extended Advertising Report.
0x02 : Periodic Advertising Report.

Connectable advertising &&
Scannable advertising &&
Legacy advertising PDU

Connectable advertising &&
Scannable advertising &&
Scan response &&
Legacy advertising PDU

11.4.2 Using BTTS Data Comm Master

Using Data Comm Master enables to check Write Response by executing consecutive Write Request to Server application that added the following Throughput service with QE for BLE.

CUSTOM SERVICE
UUID: 9CEF3D10-7FAB-49DC-AB89-762C9079FE96
PRIMARY SERVICE

CUSTOM CHARACTERISTIC
UUID: 9CEF3D11-7FAB-49DC-AB89-762C9079FE96
Properties: Write / Write Without Response

CUSTOM CHARACTERISTIC
UUID: 9CEF3D12-7FAB-49DC-AB89-762C9079FE96
Properties: Indicate / Notify
Descriptors:
Client Characteristic Configuration
UUID: 0x2920

In the following example, Write Request with Connection Interval of 1000 ms is sent. Since Write Response is received at the next connection event and Write Request is sent at the next connection event, it can be seen transmitting at about 2000 ms intervals from 16 seconds 332 to 18 seconds 349.

```
[61] 16:58:16:332 (result = 0x0000)
R_BLE_GATTC_WriteChar
       conn_hdl : 0x0020
    write_data ->
           attr_hdl : 0x0012
             value ->
                    value_len : 0x00F4
                        value : (OMISSION because of long data)

[62] 16:58:18:348 (result = 0x0000)
BLE_GATTC_EVENT_CHAR_WRITE_RSP
  value_hdl : 0x0012

[63] 16:58:18:349 (result = 0x0000)
R_BLE_GATTC_WriteChar
       conn_hdl : 0x0020
    write_data ->
           attr_hdl : 0x0012
             value ->
                    value_len : 0x00F4
                        value : (OMISSION because of long data)
[64] 16:58:20:365 (result = 0x0000)
BLE_GATTC_EVENT_CHAR_WRITE_RSP
  value_hdl : 0x0012
```

11.4.3 Using GATT Browser

It enables to check the GATT database, Indication, Notification, Read Response, Write Response by connecting to Client application.

11.5 Checking Client operation

11.5.1 Using BTTS Beacon Advertising

Using Beacon Advertising enables to send Advertising to Client. If using Command line function on Client side, Scan is checked.

If adding the following code, start of Scan and reception of Advertising are displayed.

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_GAP_EVENT_SCAN_ON:
        {
            R_BLE_CLI_Printf("receive BLE_GAP_EVENT_SCAN_ON result : 0x%04x\n", result );
        } break;
        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
           R_BLE_CLI_Printf("receive BLE_GAP_EVENT_ADV_REPT_IND result : 0x%04x\n", result );
        } break;
        (OMISSION)
```

Code 11-8 Display example of starting Scan and receiving Advertising on client side

The following is the execution result. Since Advertising by Beacon Advertising is non-connectable, Connection will fail.

```
$ gap scan 0x09 0x52
receive BLE_GAP_EVENT_SCAN_ON result : 0x0000
74:90:50:FF:FF:FF pub ff 0000
receive BLE_GAP_EVENT_ADV_REPT_IND result : 0x0000
receive BLE_GAP_EVENT_SCAN_OFF result : 0x0000

$ receive BLE_GAP_EVENT_CONN_IND result : 0x000e

$
```

11.5.2 Using BTTS Data Comm Slave

Using Data Comm Slave enables to check Confirmation by executing continuous Indication to Client application that added the following Throughput service with QE for BLE. Connection, Service Discovery, and Write Request are also checked.

CUSTOM SERVICE (Please set the abbreviation of this service to “th”)
UUID: 9CEF3D10-7FAB-49DC-AB89-762C9079FE96
PRIMARY SERVICE

CUSTOM CHARACTERISTIC
UUID: 9CEF3D11-7FAB-49DC-AB89-762C9079FE96
Properties: Write / Write Without Response

CUSTOM CHARACTERISTIC (Character abbreviation should be thin)
UUID: 9CEF3D12-7FAB-49DC-AB89-762C9079FE96
Properties: Indicate / Notify
Descriptors:
Client Characteristic Configuration
UUID: 0x2920

When Connection parameter update request is notified by the remote device, the local device must return Response. Add the following code inside GAP callback in app_main.c.

```
static void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
    switch(type)
    {
        case BLE_GAP_EVENT_CONN_PARAM_UPD_REQ:
        {
            st_ble_gap_conn_upd_req_evt_t *p_conn_upd_req_evt_param =
                                      (st_ble_gap_conn_upd_req_evt_t *)p_data->p_param;

            st_ble_gap_conn_param_t conn_updt_param = {
                .conn_intv_min = p_conn_upd_req_evt_param->conn_intv_min,
                .conn_intv_max = p_conn_upd_req_evt_param->conn_intv_max,
                .conn_latency  = p_conn_upd_req_evt_param->conn_latency,
                .sup_to        = p_conn_upd_req_evt_param->sup_to,
            };

            R_BLE_GAP_UpdConn(p_conn_upd_req_evt_param->conn_hdl,
                              BLE_GAP_CONN_UPD_MODE_RSP,
                              BLE_GAP_CONN_UPD_ACCEPT,
                              &conn_updt_param);
        } break;
        (OMISSION)
```

Code 11-9 Sample response to connection parameter update request

It is necessary to execute Write Request to enable Indication to Throughput characteristic of Throughput service of Data Comm Slave. Add the following code inside disc callback in app_main.c. It is called when Service Discovery discovers Server side Throughput service.

```
static void disc_comp_cb(uint16_t conn_hdl)
{
    /* TODO: Add function after discovery completed */
    {
        uint16_t s_cccd_req;
        s_cccd_req = BLE_GATTS_CLI_CNFG_NOTIFICATION | BLE_GATTS_CLI_CNFG_INDICATION;
        R_BLE_THC_WriteThinCliCnfg(g_conn_hdl, &s_cccd_req);
    }
    (OMISSION)
```

Code 11-10 Example of enabling Indication in disc callback

In the following example, Indication with Connection Interval of 1000 ms is sent. Since Confirmation is received at the next connection event and Indication is sent at the next connection event, it can be seen transmitting data at the interval of about 2000 ms from 25.266 seconds to 27.286 seconds.

```
[62] 19:03:25:266 (result = 0x0000)
R_BLE_GATTS_Indication
  conn_hdl : 0x0060
  ind_data ->
      attr_hdl : 0x0005
         value ->
             value_len : 0x0014
                 value : 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
0x10 0x11 0x12 0x13

[63] 19:03:27:286 (result = 0x0000)
BLE_GATTS_EVENT_HDL_VAL_CNF
  attr_hdl : 0x0005

[64] 19:03:27:286 (result = 0x0000)
R_BLE_GATTS_Indication
  conn_hdl : 0x0060
  ind_data ->
      attr_hdl : 0x0005
         value ->
             value_len : 0x0014
                 value : 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
0x10 0x11 0x12 0x13

[65] 19:03:29:207 (result = 0x0000)
BLE_GATTS_EVENT_HDL_VAL_CNF
  attr_hdl : 0x0005
```

11.6 Others

11.6.1 MCU package

In e^2studio, [Renesas Views] → [Smart Configurator] → [MCU Package] will show the pin layout of RX23W. It enables to check whether the settings match to the connection to such as LED and Switch and so on.

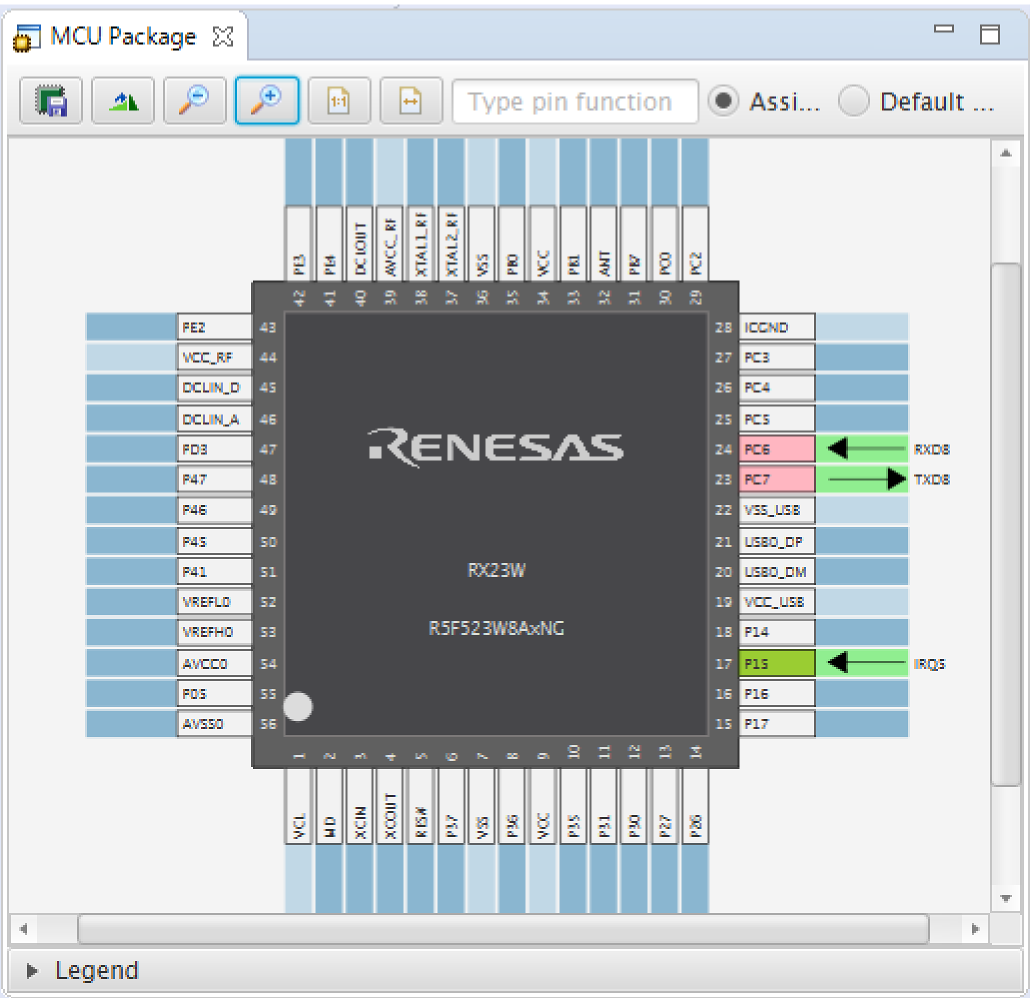


Figure 11-2 RX23W pinout

In Target Board, it is connected as follows.

Table 11.1 Target Board Pin Connection

Pin No.	Pin name	Comment
1	VCL	Digital power supply
2	MD	MCU header CN3 (can be used arbitrarily)
3	XCIN	
4	XCOUT	
5	RES#	MCU header CN3 (RES)
6	P37	MCU header CN3 (can be used arbitrarily)
7	VSS	MCU header CN3 (GND)
8	P36	MCU header CN3 (can be used arbitrarily)
9	VCC	MCU header CN3 (TGV)
10	P35	MCU header CN3 (can be used arbitrarily)
11	P31	MCU header CN3 (can be used arbitrarily), Pmod™ connector
12	P30	MCU header CN3 (can be used arbitrarily), Pmod™ connector
13	P27	MCU header CN3 (can be used arbitrarily), Pmod™ connector
14	P26	MCU header CN3 (can be used arbitrarily), Pmod™ connector

Pin No.	Pin name	Comment
15	P17	MCU header CN3 (can be used arbitrarily)
16	P16	MCU header CN3 (can be used arbitrarily)
17	P15	SW1 (used as IRQ5), MCU header CN3 (can be used arbitrarily)
18	P14	MCU header CN3 (can be used arbitrarily)
19	VCC_USB	
20	USB0_DM	MCU header CN3 (can be used arbitrarily)
21	USB0_DP	MCU header CN3 (can be used arbitrarily)
22	VSS_USB	
23	PC7	UART communication (used as TXD8), MCU header CN3 (can be used arbitrarily)
24	PC6	UART communication (used as RXD8), MCU header CN3 (can be used arbitrarily)
25	PC5	MCU header CN3 (can be used arbitrarily)
26	PC4	MCU header CN3 (can be used arbitrarily)
27	PC3	MCU header CN3 (can be used arbitrarily)
28	ICGND	GND, Bluetooth Low Energy hardware
29	PC2	MCU header CN4 (can be used arbitrarily)
30	PC0	LED1, MCU header CN4 (can be used arbitrarily)
31	PB7	MCU header CN4 (can be used arbitrarily), Pmod™ connector
32	ANT	Bluetooth Low Energy hardware
33	PB1	MCU header CN4 (can be used arbitrarily), Pmod™ connector, IRQ4
34	VCC	Digital power supply, MCU header CN4(VCC)
35	PB0	LED2, MCU header CN4 (can be used arbitrarily)
36	VSS	Digital power supply, MCU header CN4(GND)
37	XTAL2_RF	
38	XTAL1_RF	
39	AVCC_RF	Bluetooth Low Energy hardware
40	DCLOUT	Bluetooth Low Energy hardware
41	PE4	MCU header CN4 (can be used arbitrarily)
42	PE3	MCU header CN4 (can be used arbitrarily)
43	PE2	MCU header CN4 (can be used arbitrarily)
44	VCC_RF	Bluetooth Low Energy hardware
45	DCLIN_D	Bluetooth Low Energy hardware
46	DCLIN_A	Bluetooth Low Energy hardware
47	PD3	Bluetooth Low Energy hardware (CLKOUT_RF), MCU header CN4 (can be used arbitrarily), Pmod™ connector
48	P47	MCU header CN4 (can be used arbitrarily)
49	P46	MCU header CN4 (can be used arbitrarily)
50	P45	MCU header CN4 (can be used arbitrarily)
51	P41	MCU header CN4 (can be used arbitrarily)
52	VREFL0	Analog power supply, MCU header CN4 (VRL)
53	VREFH0	Analog power supply, MCU header CN4 (VRH)
54	AVCC0	Analog power supply, MCU header CN4 (AVC)
55	P05	MCU header CN4 (can be used arbitrarily), Pmod™ connector
56	AVSS0	Analog power supply, MCU header CN4 (AVS)

11.6.2 Generating MOT file

When checking [Project] → [Properties] → [C/C++ Build] → [Settings] → [Tool Settings] → [Converter] → [Output] → [Output hex file] to ON, and setting [Output file type] to "Motorola S format file", MOT file is generated.

11.6.3 Outputting detail to MAP file

When checking [Project] → [Properties] → [C/C++ Build] → [Settings] → [Tool Settings] → [Linker] → [List] → [Generate list file] to ON, and setting [Specify listfile features] to “Specify all contents”, the details of MAP file are outputted.

11.6.4 Optimization

When setting [Project] → [Properties] → [C/C++ Build] → [Settings] → [Tool Settings] → [Compiler] → [Optimization] → [Optimization level] to "Level 0: Do not perform optimization", the memory contents can be confirmed during debugging.

12. Appendix A : Sample applications

Table 12.1 shows the sample applications for Target Board for RX23W attached to this APN.

Table 12.1 Sample applications

Application	Project	Reference
Beacon sample	ble_demo_tbrx23w_beacon_sample	4.2.2 How to create a user command 5.5 Advertising with Abstraction API 5.7 Beacon
Peripheral sample	ble_demo_tbrx23w_peripheral_sample	5.5 Advertising with Abstraction API 5.6 Connection with Smart Phone 7.3.2 Connection to multiple central devices 8.4 Changing MTU 9.1.1 Pairing Parameters 9.4 Privacy 10.2.4 Notification operation 10.3.2 Example for sending data from GATT server
Central sample	ble_demo_tbrx23w_central_sample	6.1 Start or stop scan 6.2 Scan parameters 6.3 Received information by scan 6.4.4 Advertising Data filtering 7.1 Requesting Connection 7.3.1 Connecting to multiple peripheral devices 8.1 Changing PHY 8.4 Changing MTU 9.1.1 Pairing Parameters 9.1.4 Pairing request 9.3.1 Request Encryption 9.4 Privacy 10.2.2 Write operation 10.3.1 Example for sending data from GATT client
Multi-role sample	ble_demo_tbrx23w_multirole_sample	5.5 Advertising with Abstraction API 5.6 Connection with Smart Phone 6.1 Start or stop scan 6.2 Scan parameters 6.3 Received information by scan 6.4.4 Advertising Data filtering 7.1 Requesting Connection 7.3 Multiple Connection 8.1 Changing PHY 8.4 Changing MTU 9.1.1 Pairing Parameters 9.1.4 Pairing request 9.3.1 Request Encryption 9.4 Privacy 10.2.2 Write operation 10.2.4 Notification operation 10.3 Example of using GATT Procedure

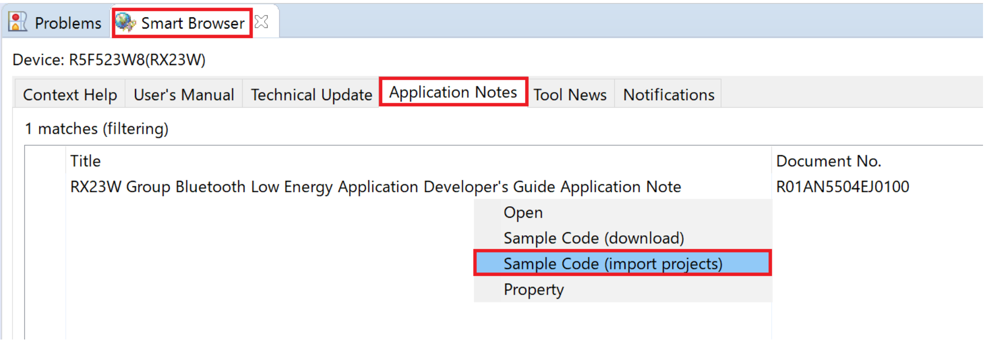
Table 12.2 shows the sample applications environment. The sample applications use the FIT modules in the RX Driver Package v1.29 (https://www.renesas.com/software-tool/rx-driver-package) except the BLE FIT and the QE utility.

Table 12.2 Sample Applications environment

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.8.0 Renesas Electronics e^2 studio 2021-04
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.08.00
Board	Target Board for RX23W
BLE FIT version	2.11
QE utility version	1.10
BSP version	5.63
LPC FIT version	2.01
CMT FIT version	4.70
IRQ FIT version	3.60
GPIO FIT version	3.70
SCI FIT version	3.70
BYTE_Q FIT version	1.82

How to import the sample application in e^2 studio is described below.

(1) Right click the application developer’s guide (Title : RX23W Group Bluetooth Low Energy Application Developer’s Guide Application Note, Document No : R01AN5504EJYYYY (YYYY : version)) on Smart Browser and select “Sample Code (import projects)”.



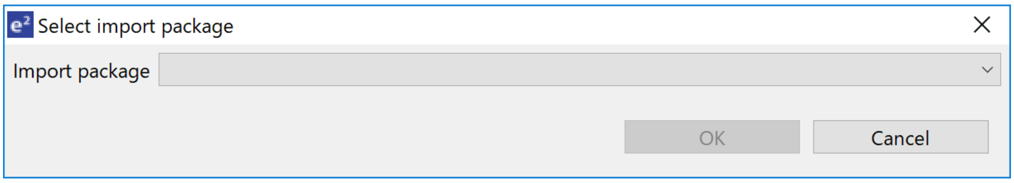
Device: R5F523W8(RX23W)

1 matches (filtering)

Title	Document No.
RX23W Group Bluetooth Low Energy Application Developer's Guide Application Note	R01AN5504EJ0100

Figure 12-1 Sample application project import

(2) Download r01an5504xxYYYY-rx23w-ble-adev.zip(YYYY : version) to the desired location. After the download has been completed, “Select import package” window is displayed. Select a sample application project.

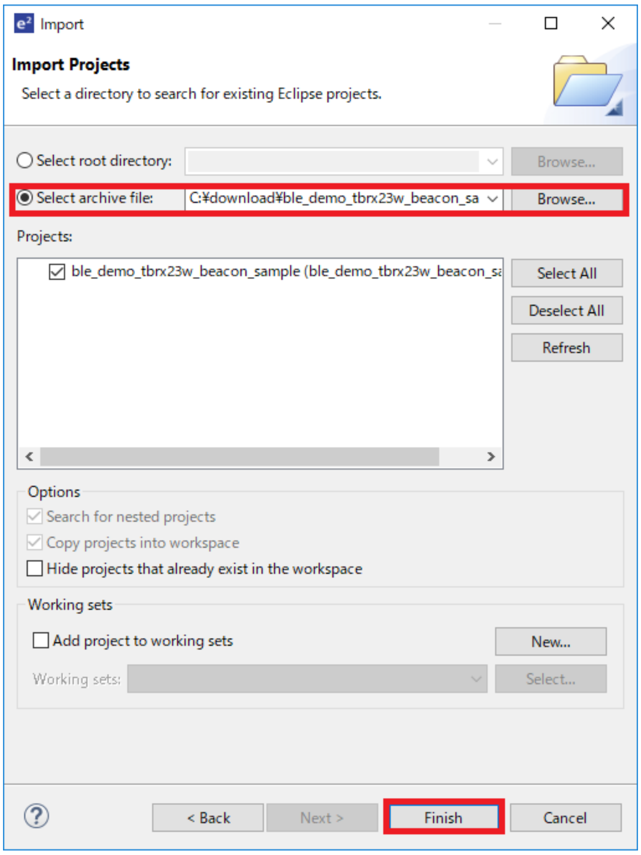


Select import package

Import package

Figure 12-2 Import package selection

(3) When “Finish” button on “Import” window has been pressed, the sample application project is imported.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

○ Select root directory:

● Select archive file: C:¥download¥ble_demo_tbrx23w_beacon_sa

Projects:

☑ ble_demo_tbrx23w_beacon_sample (ble_demo_tbrx23w_beacon_s

Options

☑ Search for nested projects

☑ Copy projects into workspace

□ Hide projects that already exist in the workspace

Working sets

□ Add project to working sets

Working sets:

Figure 12-3 Project import

12.1 Beacon sample

12.1.1 Remote devices

Advertising packets from the beacon sample can be received with scan on the following remote devices.

- iOS device
- Android device

12.1.2 Operations

The beacon sample starts Non connectable undirected advertising (ADV_NONCONN_IND) after the boot.

12.1.3 Advertising Data

Specify the following Advertising Data type with BLE_APP_BEACON_TYPE in app_main.c.

0: iBeacon (default)

1: Eddystone

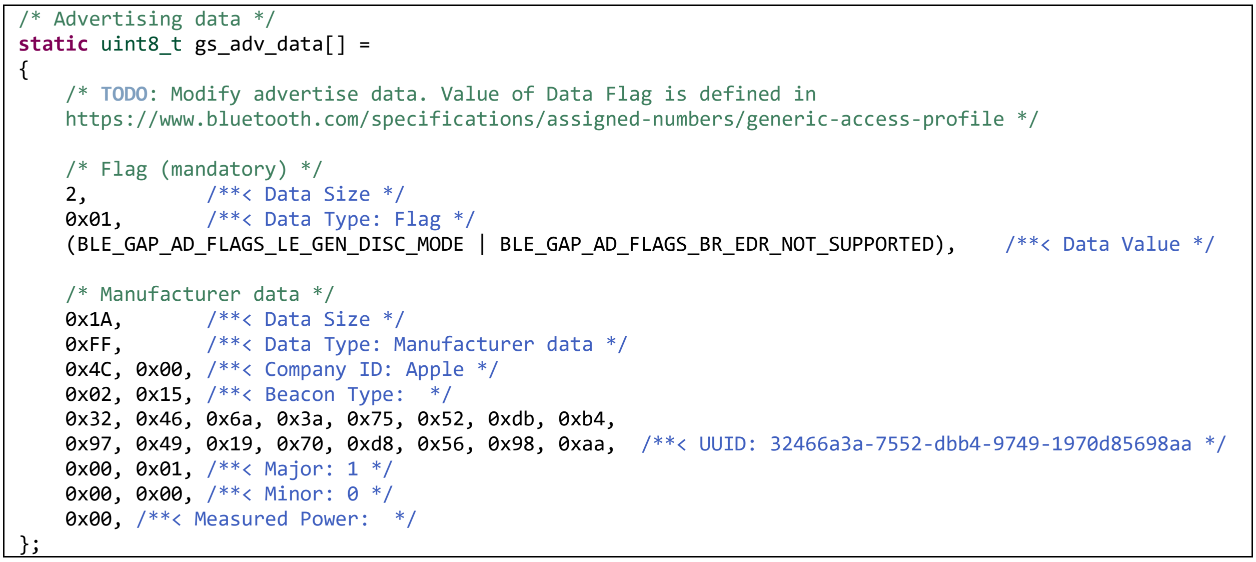
2: broadcast mode

Each Advertising Data type is described below.

- iBeacon

  The iBeacon specification is published in https://developer.apple.com/ibeacon.

  The details of the Advertising Data are follows.



```
/* Advertising data */
static uint8_t gs_adv_data[] =
{
    /* TODO: Modify advertise data. Value of Data Flag is defined in
    https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Flag (mandatory) */
    2,          /**< Data Size */
    0x01,       /**< Data Type: Flag */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE | BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED),    /**< Data Value */

    /* Manufacturer data */
    0x1A,       /**< Data Size */
    0xFF,       /**< Data Type: Manufacturer data */
    0x4C, 0x00, /**< Company ID: Apple */
    0x02, 0x15, /**< Beacon Type:  */
    0x32, 0x46, 0x6a, 0x3a, 0x75, 0x52, 0xdb, 0xb4,
    0x97, 0x49, 0x19, 0x70, 0xd8, 0x56, 0x98, 0xaa,  /**< UUID: 32466a3a-7552-dbb4-9749-1970d85698aa */
    0x00, 0x01, /**< Major: 1 */
    0x00, 0x00, /**< Minor: 0 */
    0x00, /**< Measured Power:  */
};
```

Code 12-1 Default Advertising Data for iBeacon

You need to change the above UUID to your application specific value and the Major and Minor version to your application version. Measured Power are needed to change the power value measured according to iBeacon specification.

- Eddystone

The Eddystone specification is published in https://github.com/google/eddystone.

The beacon sample provides the Eddystone-URL type. This sample does not support the Eddystone Configuration GATT Service. The details of the Advertising Data are follows.

```
/* Advertising data */
static uint8_t gs_adv_data[] =
{
    /* TODO: Modify advertise data. Value of Data Flag is defined in
    https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Flag (mandatory) */
    2,          /**< Data Size */
    0x01,       /**< Data Type: Flag */
    (BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE | BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED),    /**< Data Value */

    /* Complete list of 16bit Service UUIDs */
    3,          /**< Data Size */
    0x03,       /**< Data Type: Complete list of 16bit Service UUIDs */
    0xAA, 0xFE, /**< 16bit Eddystone UUID */

    /* Service Data */
    14,         /**< Data Size */
    0x16,       /**< Data Type: Service Data */
    0xAA, 0xFE, /**< 16bit Eddystone UUID */
    0x10,       /**< Frame Type: URL */
    0x00,       /**< Tx power: 0 dBm */
    0x01,       /**< URL Scheme: https://www. */
    0x72, 0x65, 0x6E, 0x65, 0x73, 0x61, 0x73, 0x07 /**< Encoded URL: renesas.com */
};
```

Code 12-2 Default Advertising Data for Eddystone

You need to change the above Data Size, URL Scheme and Encoded URL in Service Data to suit to your application.

- Broadcast mode

The broadcast mode Advertising Data does not include “LE General Discoverable Mode” and “LE Limited Discoverable Mode” in the Flags to meet the Broadcast Mode condition defined in Bluetooth Core Specification.

```
/* Advertising Data */
static uint8_t gs_adv_data[] =
{
    /* TODO: Modify advertise data. Value of Data Flag is defined in
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile */

    /* Flag (mandatory) */
    2,          /**< Data Size */
    0x01,       /**< Data Flag: Flag */
    BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED,    /**< Data Value */

    /* Complete Local Name */
    9,         /**< Data Size */
    0x09,       /**< Data Flag: Complete Local Name */
    'R', 'B', 'L', 'E', '-', 'A', 'D', 'V', /**< Data Value */

};
```

Code 12-3 Default Advertising Data for broadcast mode

12.1.4 Configuration option

Table 12.3 shows the BLE FIT configuration options changed from the default for the beacon sample.

Table 12.3 Changed configuration options

Macro (SC display name)	Value
BLE_CFG_LIB_TYPE (Type of Bluetooth LE Protocol Stack library)	2: Compact
BLE_CFG_RF_CONN_MAX (Maximum number of connections)	1
BLE_CFG_CMD_LINE_EN (Enabled/Disabled command line function)	1: Enable
BLE_CFG_CMD_LINE_CH (SCI CH for command line function)	8
BLE_CFG_BOARD_LED_SW_EN (Enabled/Disabled board LED and Switch control support)	1: Enable
BLE_CFG_BOARD_TYPE (Board Type)	1: Target Board

12.1.5 Configurable parameters

(1) Address

Specify the following address type with BLE_BEACON_ADDR_TYPE in app_main.c.

BLE_GAP_ADDR_RAND : Static Address (default)
BLE_GAP_ADDR_RPA_ID_RANDOM : RPA(static) address

(2) Advertising Data, Advertising parameters

The gs_adv_data (Advertising Data), gs_adv_param (Advertising parameters) variables in app_main.c are configurable according to each beacon specification in “12.1.3 Advertising Data”.

12.1.6 Command

If the beacon sample uses RPA, it supports the following command to display the current RPA.

```
beacon lrpa
```

12.2 Peripheral sample

12.2.1 Remote devices

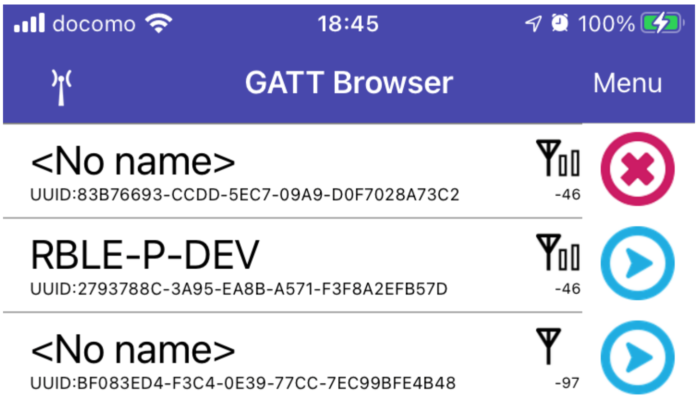
The peripheral sample supports connections with the following remote devices.

- Central sample
- Multi-role sample
- iOS device
- Android device

12.2.2 Operations

The peripheral sample works as follows.

- The peripheral sample starts Connectable undirected advertising (ADV_IND) after the boot.
  It starts fast advertising (interval: 30ms) in the first 30s and changes to slow advertising (interval: 1000ms) in the next 30s.
- By scanning from a remote device, it is detected as the “RBLE-P-DEV” device name.



docomo 18:45 100%

GATT Browser

Menu

<No name>
UUID:83B76693-CCDD-5EC7-09A9-D0F7028A73C2
-46

RBLE-P-DEV
UUID:2793788C-3A95-EA8B-A571-F3F8A2EFB57D
-46

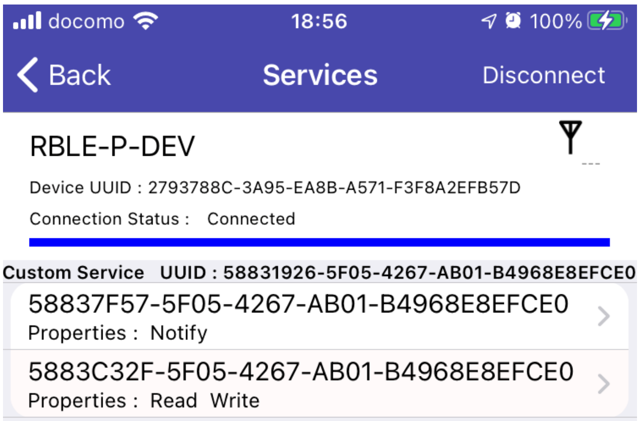
<No name>
UUID:BF083ED4-F3C4-0E39-77CC-7EC99BFE4B48
-97

Figure 12-4 Scan result on central device

- After connection establishment, if the simultaneous multiple connections feature is not supported, the peripheral sample stops the advertising. Otherwise, it continues the advertising. For changing the simultaneous multiple connections feature, refer “12.2.4(3)”.
- If a remote device searches GATT Services in the peripheral sample, the following service and characteristics are detected.

Table 12.4 Detected service and characteristics

Service, characteristic	UUID
LED Switch service	58831926-5F05-4267-AB01-B4968E8EFCE0
Switch State characteristic	58837F57-5F05-4267-AB01-B4968E8EFCE0
LED Blink Rate characteristic	5883C32F-5F05-4267-AB01-B4968E8EFCE0



RBLE-P-DEV

Device UUID : 2793788C-3A95-EA8B-A571-F3F8A2EFB57D

Connection Status : Connected

Custom Service UUID : 58831926-5F05-4267-AB01-B4968E8EFCE0

58837F57-5F05-4267-AB01-B4968E8EFCE0
Properties : Notify

5883C32F-5F05-4267-AB01-B4968E8EFCE0
Properties : Read Write

Figure 12-5 Detected GATT service and characteristics

- The peripheral sample sets BLE_GATT_DB_SER_SECURITY_UNAUTH | BLE_GATT_DB_SER_SECURITY_ENC to the second parameter which indicates LED Switch service security requirement in the gs_gatt_service variable in gatt_db.c. Therefore, if a remote device accesses a characteristic in LED Switch service, it requires pairing.

```
static const st_ble_gatts_db_serv_cfg_t gs_gatt_service[] =
{
    /* some code is omitted */
    /* LED Switch Service */
    {
        /* Num of Services */
        {
            1,
        },
        /* Description */
        BLE_GATT_DB_SER_SECURITY_UNAUTH | BLE_GATT_DB_SER_SECURITY_ENC,
        /* Service Start Handle */
        0x0010,
        /* Service End Handle */
        0x0015,
        /* Characteristic Start Index */
        6,
        /* Characteristic End Index */
        7,
    },

};
```

Code 12-4 LED Switch service security requirement

- After the remote device enables the Switch State characteristic Notification, the peripheral sample sends a Notification after SW1 is pressed on the board.
- If the remote device writes a value to the LED Blink Rate characteristic, the LED on the board blinks at the value x 100ms interval. When writing zero to the characteristic, the LED will turn off.
- If the link is disconnected, the peripheral sample restarts advertising.
- When press the reset button while pressing SW1, the bonding information is deleted.

12.2.3 Configuration option

Table 12.5 shows the BLE FIT configuration options changed from the default for the peripheral sample.

Table 12.5 Changed configuration options

Macro (SC display name)	Value
BLE_CFG_LIB_TYPE (Type of Bluetooth LE Protocol Stack library)	1: Balance
BLE_CFG_RF_CONN_MAX (Maximum number of connections)	3
BLE_CFG_NUM_BOND (Number of remote device bonding information)	3
BLE_CFG_EN_SEC_DATA (Store Security Data in DataFlash)	1: Enable
BLE_CFG_CMD_LINE_EN (Enabled/Disabled command line function)	1: Enable
BLE_CFG_CMD_LINE_CH (SCI CH for command line function)	8
BLE_CFG_BOARD_LED_SW_EN (Enabled/Disabled board LED and Switch control support)	1: Enable
BLE_CFG_BOARD_TYPE (Board Type)	1: Target Board

12.2.4 Configurable parameters

(1) Address

Specify the following address type with BLE_PERIPHERAL_ADDR_TYPE in app_main.c.

BLE_GAP_ADDR_RAND	: Static Address (default)
BLE_GAP_ADDR_RPA_ID_RANDOM	: RPA(static) address

(2) Advertising Data, Scan Response Data, Advertising parameters

The gs_adv_data (Advertising Data), gs_sres_data (Scan Response Data) and gs_adv_param (Advertising parameters) variables in app_main.c are configurable. If you change the device name included in gs_adv_data or gs_sres_data and the peripheral sample connects with a Central sample, change the scan filter in the Central sample.

(3) Simultaneous multiple connections feature

If the BLE_PERIPHERAL_MULTI_CONNS macro in app_main.c is enabled, the peripheral sample supports the simultaneous multiple connections feature.

12.3 Central sample

12.3.1 Remote devices

The central sample supports connection with the following remote device.

- Peripheral sample

12.3.2 Operations

The central sample works as follows.

- The central sample starts scan to detect a peripheral sample after booting.
  It starts fast scan (interval: 60ms, window: 30ms) in a first 10s and then continues slow scan (interval: 1200ms, window: 11.25ms) for 10s. After stopping scan, the central sample restarts scan by pressing SW1.
- After detecting a peripheral sample, the central sample stops scan. It sends a connection request to the detected peripheral sample.
- After connection establishment, the packet length is updated.
- After packet length update, a MTU change request is sent to the remote device.
  If the PHY is changed, a PHY change request is sent to the remote device before sending a MTU change request.
- When receiving a MTU change response from the remote device, the central sample discovers LED Switch service in Table 12.4.
- After service discovery, the central sample writes 1 to the CCCD of the Switch State characteristic.
- If pairing is not completed, the peripheral sample returns an error. When the central sample receives the error, it starts pairing. If pairing is completed but encryption is not completed, the peripheral sample returns an error. When the central sample receives the error, it starts encryption.
- When pairing and encryption are completed, the central sample writes 1 to the CCCD again.
- Then after pressing SW1 on the peripheral sample board, the switch state characteristic is sent to the central sample as Notification.
- When press the reset button while pressing SW1, the bonding information is deleted.

12.3.3 Configuration option

Table 12.6 shows the BLE FIT configuration options changed from the default for the central sample.

Table 12.6 Changed configuration options

Macro (SC display name)	Value
BLE_CFG_LIB_TYPE (Type of Bluetooth LE Protocol Stack library)	1: Balance
BLE_CFG_RF_CONN_MAX (Maximum number of connections)	3
BLE_CFG_NUM_BOND (Number of remote device bonding information)	3
BLE_CFG_EN_SEC_DATA (Store Security Data in DataFlash)	1: Enable
BLE_CFG_CMD_LINE_EN (Enabled/Disabled command line function)	1: Enable

Macro (SC display name)	Value
BLE_CFG_CMD_LINE_CH (SCI CH for command line function)	8
BLE_CFG_BOARD_LED_SW_EN (Enabled/Disabled board LED and Switch control support)	1: Enable
BLE_CFG_BOARD_TYPE (Board Type)	1: Target Board

12.3.4 Configurable parameters

(1) Address

Specify the following address type with BLE_CENTRAL_ADDR_TYPE in app_main.c.

BLE_GAP_ADDR_RAND	: Static Address (default)
BLE_GAP_ADDR_RPA_ID_RANDOM	: RPA(static) address

(2) Scan parameters, connection parameters

The gs_scan_phy_param, gs_scan_param (scan parameters) and gs_conn_phy_param, gs_conn_param (connection parameters) variables in app_main.c are configurable. If you change the peripheral sample device name included in the advertising data or scan response data, change the gs_filter_data as scan filter.

(3) PHY

If you want to change PHY from 1M to 2M after connection establishment, set the BLE_APP_CHANGE_PHY_2M macro in app_main.c to 1. The default value is zero.

12.4 Multi-role sample

12.4.1 Topology

The multi-role sample connects with a central device and a peripheral sample and bridges the characteristic written from the central device or notified from the peripheral sample. Figure 12-6 shows the multi-role sample topology.

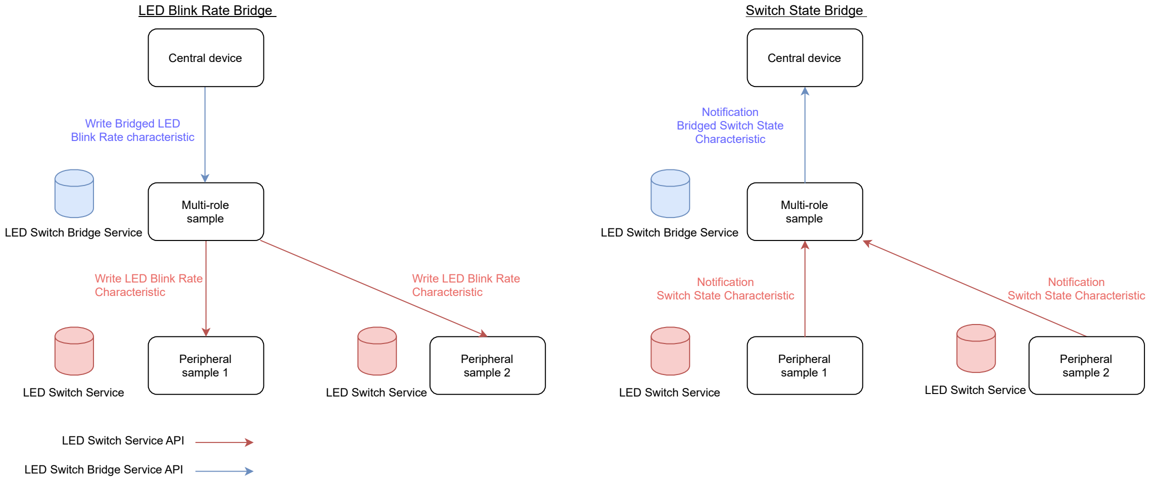
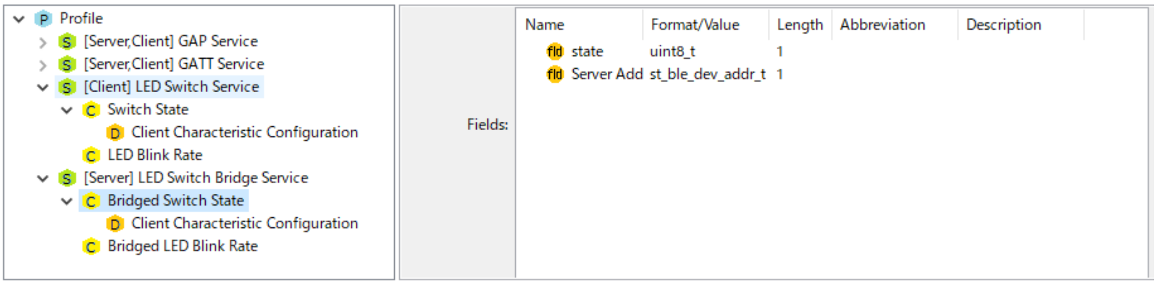


Figure 12-6 Multi-role sample topology

The multi-role sample adds LED Switch Bridge service which includes Bridged Switch State characteristic (Figure 12-7) and Bridged LED Blink Rate characteristic (Figure 12-8) to implement the bridge between the central device and the peripheral sample. The Bridged Switch State characteristic includes a Bluetooth device address to indicate which peripheral sample sends a notification. After conversion from the LED Switch Bridge service characteristic to the LED Switch service characteristic, the multi-role sample sends write request for the LED Switch service to the peripheral samples. Similarly, after conversion from the LED Switch service characteristic to the LED Switch Bridge service characteristic, the multi-role sample sends notification for the LED Switch Bridge service to the central device.

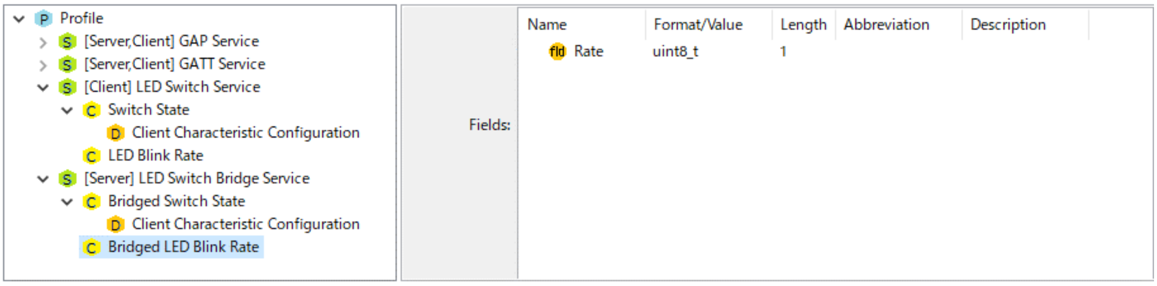


- P Profile
  - S [Server,Client] GAP Service
  - S [Server,Client] GATT Service
  - S [Client] LED Switch Service
    - C Switch State
      - D Client Characteristic Configuration
    - C LED Blink Rate
  - S [Server] LED Switch Bridge Service
    - C Bridged Switch State
      - D Client Characteristic Configuration
    - C Bridged LED Blink Rate

Fields:

Name	Format/Value	Length	Abbreviation	Description
fld state	uint8_t	1		
fld Server Add	st_ble_dev_addr_t	1		

Figure 12-7 Bridged Switch State characteristic



- P Profile
  - S [Server,Client] GAP Service
  - S [Server,Client] GATT Service
  - S [Client] LED Switch Service
    - C Switch State
      - D Client Characteristic Configuration
    - C LED Blink Rate
  - S [Server] LED Switch Bridge Service
    - C Bridged Switch State
      - D Client Characteristic Configuration
    - C Bridged LED Blink Rate

Fields:

Name	Format/Value	Length	Abbreviation	Description
fld Rate	uint8_t	1		

Figure 12-8 Bridged LED Blink Rate characteristic

12.4.2 Remote devices

The multi-role sample supports connection with the following remote device.

[Peripheral role] :

- iOS device
- Android device

[Central role] :

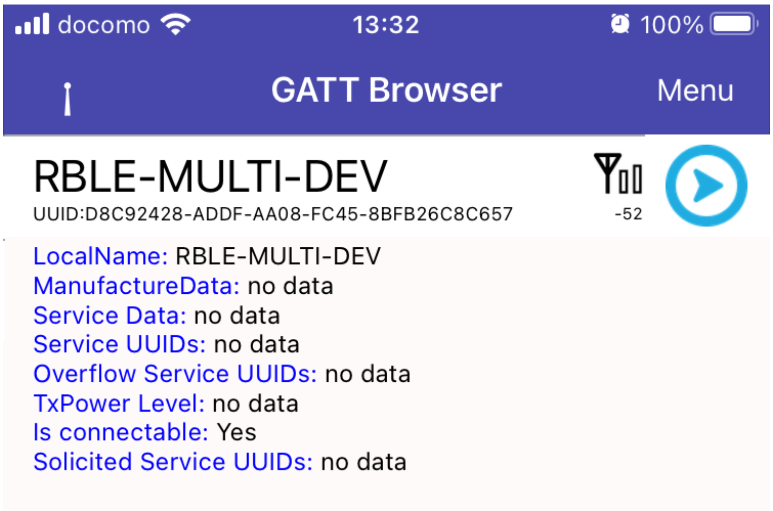
- Peripheral sample

12.4.3 Operations

The multi-role sample works as follows.

[Connection to the central device] :

- The multi-role sample starts Connectable undirected advertising (ADV_IND) after the boot. It starts fast advertising (interval: 30ms) in the first 30s and changes to slow advertising (interval: 1000ms) in the next 30s.
- By scanning from a remote device, it is detected as the “RBLE-MULTI-DEV” device name.



RBLE-MULTI-DEV
UUID:D8C92428-ADDF-AA08-FC45-8BFB26C8C657

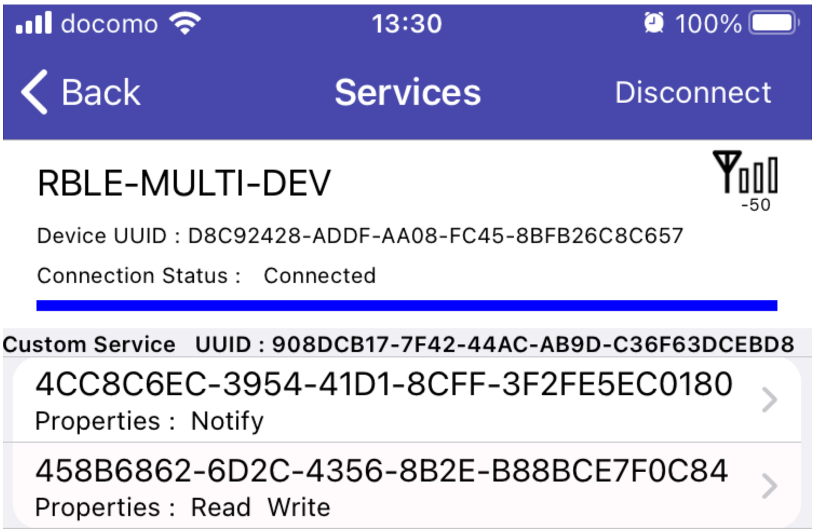
LocalName: RBLE-MULTI-DEV
ManufactureData: no data
Service Data: no data
Service UUIDs: no data
Overflow Service UUIDs: no data
TxPower Level: no data
Is connectable: Yes
Solicited Service UUIDs: no data

Figure 12-9 Scan result on central device

- After connection establishment, the multi-role sample stops advertising. The multi-role sample connects simultaneously to only one central device.
- If a remote device searches GATT Services in the multi-role sample, the following service and characteristics are detected.

Table 12.7 Detected service and characteristics

Service, characteristic	UUID
LED Switch Bridge service	908DCB17-7F42-44AC-AB9D-C36F63DCEBD8
Bridged Switch State characteristic	4CC8C6EC-3954-41D1-8CFF-3F2FE5EC0180
Bridged LED Blink Rate characteristic	458B6862-6D2C-4356-8B2E-B88BCE7F0C84



docomo 13:30 100%

Back Services Disconnect

RBLE-MULTI-DEV -50

Device UUID : D8C92428-ADDF-AA08-FC45-8BFB26C8C657

Connection Status : Connected

Custom Service UUID : 908DCB17-7F42-44AC-AB9D-C36F63DCEBD8

4CC8C6EC-3954-41D1-8CFF-3F2FE5EC0180
Properties : Notify

458B6862-6D2C-4356-8B2E-B88BCE7F0C84
Properties : Read Write

Figure 12-10 Detected GATT service and characteristics

- The multi-role sample sets BLE_GATT_DB_SER_SECURITY_UNAUTH | BLE_GATT_DB_SER_SECURITY_ENC to the second parameter which indicates LED Switch Bridge service security requirement in the gs_gatt_service variable in gatt_db.c. Therefore, if a remote device accesses a characteristic in LED Switch Bridge service, it requires pairing.

```
static const st_ble_gatts_db_serv_cfg_t gs_gatt_service[] =
{
    /* Some code is omitted */
    /* LED Switch Bridge Service */
    {
        /* Num of Services */
        {
            1,
        },
        /* Description */
        BLE_GATT_DB_SER_SECURITY_UNAUTH | BLE_GATT_DB_SER_SECURITY_ENC,
        /* Service Start Handle */
        0x0010,
        /* Service End Handle */
        0x0015,
        /* Characteristic Start Index */
        6,
        /* Characteristic End Index */
        7,
    },
};
```

Code 12-5 LED Switch Bridge service security requirement

- If the link with the central device is disconnected, the multi-role sample restarts advertising.

[Connection to the peripheral sample] :

- The multi-role sample starts scan to detect a peripheral sample after pressing SW1.
  It starts fast scan (interval: 60ms, window: 30ms) in a first 10s and then continues slow scan (interval: 1200ms, window: 11.25ms) for 10s. After stopping scan, the central sample restarts scan by pressing SW1.
- After detecting a peripheral sample, the multi-role sample stops scan. It sends a connection request to the detected peripheral sample.
- After connection establishment, the packet length is updated.

- After packet length update, a MTU change request is sent to the remote device.
  If the PHY is changed, a PHY change request is sent to the remote device before sending a MTU change request.
- When receiving a MTU change response from the remote device, the multi-role sample discovers LED Switch service.
- After service discovery, the multi-role sample writes 1 to the CCCD of the Switch State characteristic.
- If pairing is not completed, the peripheral sample returns an error. When the multi-role sample receives the error, it starts pairing. If pairing is completed but encryption is not completed, the peripheral sample returns an error. When the multi-role sample receives the error, it starts encryption.
- When pairing and encryption are completed, the multi-role sample writes 1 to the CCCD again.
- Then after pressing SW1 on the peripheral sample board, the switch state characteristic is sent to the multi-role sample as Notification.

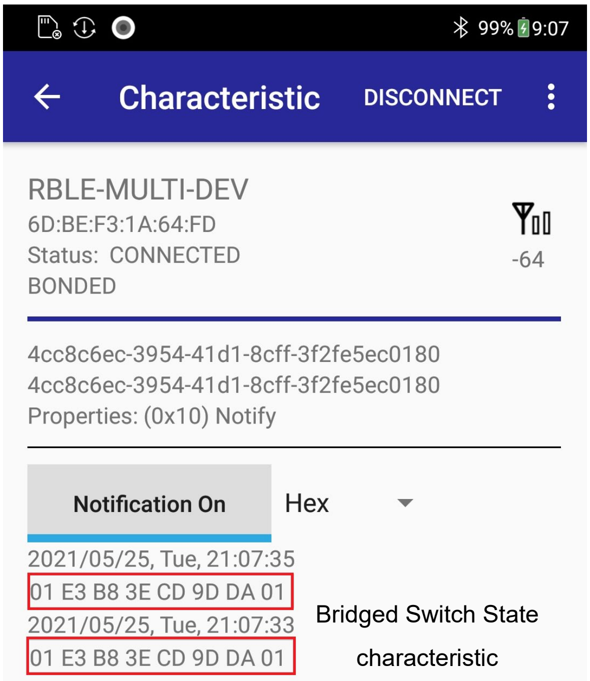
[Delete the bonding information] :

- When press the reset button while pressing SW1, the bonding information is deleted.

[Switch State characteristic bridge] :

After the following configurations are complete, when pressing SW1, the multi-role sample converts the Switch State characteristic in the LED Switch State service to the Bridged Switch State characteristic in the LED Switch Bridged service and sends write request including the data. Therefore, the Bridged Switch State characteristic is notified to the central device as shown Figure 12-11 . This characteristic includes the Switch State and the peripheral sample address as shown Figure 12-7 .

- After connection with the central device, set 1 to the CCCD of the Bridged Switch State characteristic from the central device.
- After connection with the peripheral sample, the multi-role sample automatically sets 1 the CCCD of the Switch State characteristic.



RBLE-MULTI-DEV
6D:BE:F3:1A:64:FD
Status: CONNECTED
BONDED
-64

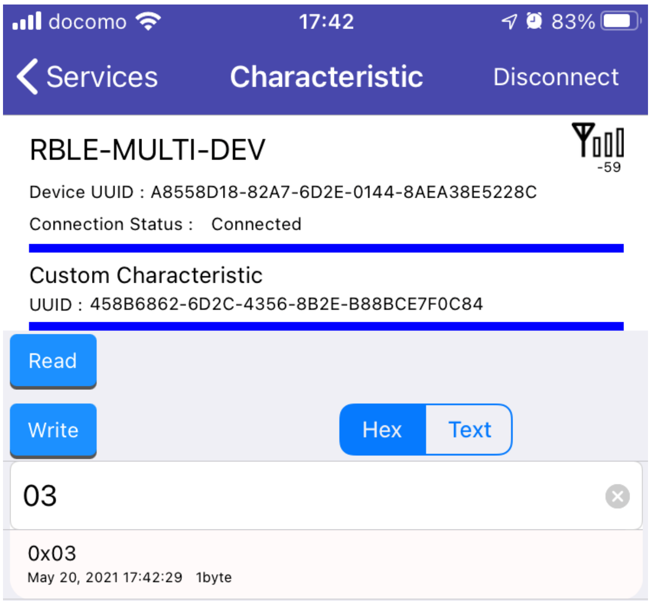
4cc8c6ec-3954-41d1-8cff-3f2fe5ec0180
4cc8c6ec-3954-41d1-8cff-3f2fe5ec0180
Properties: (0x10) Notify

2021/05/25, Tue, 21:07:35
01 E3 B8 3E CD 9D DA 01
2021/05/25, Tue, 21:07:33
01 E3 B8 3E CD 9D DA 01

Figure 12-11 Bridged Switch State characteristic notification to central device

[LED Blink Rate characteristic bridge] :

After connection with the central device and the peripheral sample, when the central device writes a value to the Bridged LED Blink Rate characteristic as shown Figure 12-12, the LED blinks at the value x 100ms interval.



RBLE-MULTI-DEV

Device UUID : A8558D18-82A7-6D2E-0144-8AEA38E5228C

Connection Status : Connected

Custom Characteristic

UUID : 458B6862-6D2C-4356-8B2E-B88BCE7F0C84

03

0x03

May 20, 2021 17:42:29 1byte

Figure 12-12 Bridged LED Blink Rate characteristic write from central device

12.4.4 Configuration option

Table 12.8 shows the BLE FIT configuration options changed from the default for the multi-role sample.

Table 12.8 Changed configuration options

Macro (SC display name)	Value
BLE_CFG_LIB_TYPE (Type of Bluetooth LE Protocol Stack library)	1: Balance
BLE_CFG_RF_CONN_MAX (Maximum number of connections)	3
BLE_CFG_NUM_BOND (Number of remote device bonding information)	3
BLE_CFG_EN_SEC_DATA (Store Security Data in DataFlash)	1: Enable
BLE_CFG_CMD_LINE_EN (Enabled/Disabled command line function)	1: Enable
BLE_CFG_CMD_LINE_CH (SCI CH for command line function)	8
BLE_CFG_BOARD_LED_SW_EN (Enabled/Disabled board LED and Switch control support)	1: Enable
BLE_CFG_BOARD_TYPE (Board Type)	1: Target Board

12.4.5 Configurable parameters

(1) Address

Specify the following address type with BLE_MULTIROLE_ADDR_TYPE in app_main.c.

BLE_GAP_ADDR_RAND	: Static Address (default)
BLE_GAP_ADDR_RPA_ID_RANDOM	: RPA(static) address

(2) PHY

If you want to change PHY from 1M to 2M after connection establishment as central role, set the BLE_APP_CHANGE_PHY_2M macro in app_main.c to 1. The default value is zero.

(3) Advertising Data, Scan Response Data, Advertising parameters

The gs_adv_data (Advertising Data), gs_sres_data (Scan Response Data) and gs_adv_param (Advertising parameters) variables in app_main.c are configurable.

(4) Scan parameters, connection parameters

The gs_scan_phy_param, gs_scan_param (scan parameters) and gs_conn_phy_param, gs_conn_param (connection parameters) variables in app_main.c are configurable. If you change the peripheral sample device name included in the advertising data or scan response data, change the gs_filter_data as scan filter.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul.28.2020	–	First edition issued.
1.10	May.31.2021	84, 87, 89, 92, 96, 96, 98, 105	Fixed the size of Advertising Data.
		149	Fixed Table 9.5. Added that the pairing will fail if the input/output functions of both devices are Just Works when MITM protection is required.
		167	Fixed that if local device does not use RPA and resolves a remote device address, set all zeros IRK in resolving list.
		214	Added “12. Appendix A : Sample applications”.
		Program	Added sample applications.
1.20	Jun.30.2022	21, 83	Added “Do not change the Abstraction API codes.”.
		106-107	Added how to enable privacy feature by the scan Abstraction API.
		116	Added how to retain the connection handles.
		118-118	Added how to enable privacy feature by the connection Abstraction API.
		162-165	Added the relationship between the bonding number and the maximum simultaneous connection number.
		166-167	Added the description about deleting the bonding information.
		167	Added how to connect to only the bonded device.
		168-172	Added the sequence if encryption was failed.
		173-184	Added the supplementary description of RPA.
1.30	Dec.27.2022	1, 7, 88, 107, 150, 153, 154, 156, 157, 159, 160, 168	Added recommendations and risk descriptions based on the "Bluetooth® Security and Privacy Best Practices Guide" published by the Bluetooth SIG so that implementers can select the best practices for security and privacy.
		88, 101	Added about RPA default update interval and API to change update interval.
		154	Added about notification information when LE legacy pairing starts with Secure Connection Only specified.
		175	When the local device uses RPA, the setting value of Identity Address and IRK of the remote device to be associated is changed from "dummy" to "all 0x00", and the registration condition ("only the local device is uses RPA or it is in unpaired state") removed. Also changed the related example code from "0xAA" and "0x55" to "0x00".
		8, 10, 25	Added the provision format of QE Utility for QE for BLE V1.40 or later.
		16	Added “1.4.1 Device identification”.
		18	Added the explanation about initializing RF hardware state.
		23, 29	Added how to select library type.
		24	Added the explanation about interrupt priority.
		33	Added note about Public device address and Static address.
		37	Added the explanation about R_BLE_Open and clock frequency.
		48	Added the explanation about R_BLE_Close and RF H/W stop.
		49-56	Added p_data when result is other than BLE_SUCCESS.
		86	Added that parameters cannot be changed during Advertising.
		87, 105, 116	Added the explanation about PHY of Advertising / Scan / Connection.
		88, 100, 110, 117	Added that White List cannot be set while it is used.
		88, 101	Added explanation about o_addr for Privacy.
		135	Added an explanation for the reason for disconnection.
		136	Added libraries and roles that can change communication parameters.

		137	Added the explanation about PHY that allows modification.
		160	Added the explanation about result when pairing fails.
		174	Added a figure for privacy procedures.

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

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

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

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

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

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

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

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

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

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

Contact information

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