RE01B Group

Bluetooth Low Energy Application Developer's Guide

Introduction

This application note describes how to develop a Bluetooth Low Energy application.

Target Device

RE01B Group

Related Documents

- Bluetooth Core Specification (https://www.bluetooth.com)
- Supplement of Bluetooth Core Specification (https://www.bluetooth.com)
- RE01B Group Product with 1.5-Mbyte Flash Memory User's Manual: Hardware (R01UH0903)
- Getting Started Guide to Development Using CMSIS Package (R01AN5310)
- e^2 studio Getting Started Guide (R20UT4204)
- Bluetooth Low Energy Profile Developer's Guide (R01AN5638)
- Bluetooth Low Energy MCU Bluetooth Test Tool Suite operating instructions (R01AN4554)
- RE01B Group Hardware Design Guide (R01AN5471)
- Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)
- RE01B Group IAR Embedded Workbench for Arm patch Setup Guide (R20AN0596)

Contents

1. Overview

1.1 Development Bluetooth Low Energy Application

There are two methods of data communication using Bluetooth Low Energy (Bluetooth LE): broadcast method and point-to-point method.

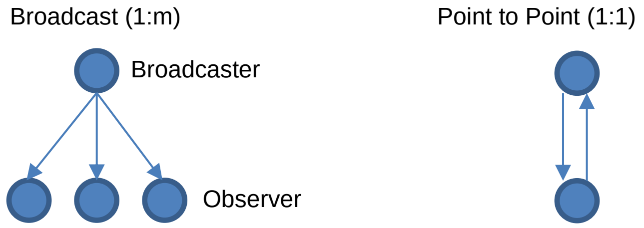

Figure 1.1 Bluetooth LE communication topology

In the broadcast method, 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.
Point-to-point communication is used for bidirectional communication. The point-to-point method 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.

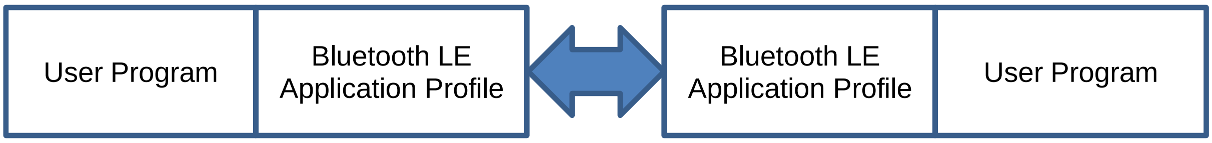

Figure 1.2 Bluetooth LE bidirectional communication

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 level.
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 Module**

   It provides the Bluetooth LE feature that complies with the Bluetooth Core Specification version 5.0 defined by Bluetooth SIG. The Bluetooth LE feature is provided in library format as a BLE protocol stack. Bluetooth LE operation is performed by using the API. The BLE protocol stack notifies the application of events related to Bluetooth LE by a callback function to reduce power consumption.

   BLE module provides application library (app_lib) to assist application development in addition to BLE protocol stack. By using app_lib, you can easily realize the basic operation of Bluetooth LE.

2. QE for BLE, QE Utility

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.

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.

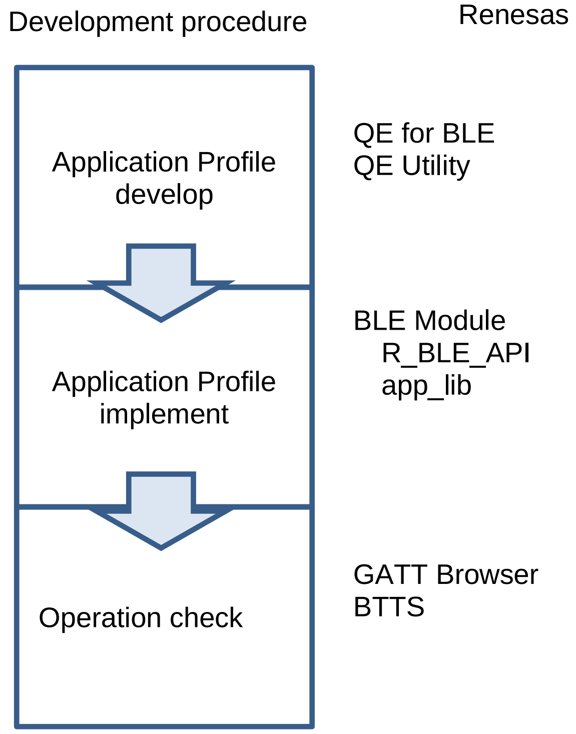

Figure 1.3 Bluetooth LE application development procedure and auxiliary tools

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® 10 PC with USB interface.
MCU board	The board with RE01B Note: This document uses EB-RE01B for explanation.
On-chip debugging emulators	GCC environment: Either of the following emulator. E2 emulator [RTE0T00020KCE00000R] E2 emulator Lite [RTE0T0002LKCE00000R] J-Link IAR environment: Either of the following emulator. I-jet® J-Link Note: This document uses E2 emulator Lite for explanation.
USB cables	Used to connect to the emulator and EB-RE01B. Emulator: 1 USB A-miniB cable Board: 1 USB A-microB cable

1.2.2 Software requirements

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

Table 1.2 Software requirements

<table>
<tr><th colspan="2">Software</th><th>Version</th><th>Description</th></tr>
<tr><td rowspan="4">GCC environment</td><td>e² studio</td><td>v21.1.0 or later</td><td>Integrated development environment (IDE) for Renesas devices.
Note: This document uses e²studio for explanation.</td></tr>
<tr><td>GCC ARM Embedded</td><td>v6.3.1 or later</td><td>C/C++ compiler. (download from e² studio installer)</td></tr>
<tr><td>CMSIS Driver Package</td><td>v1.0.0 or later</td><td>Software package for developing applications for the RE microcontroller series.</td></tr>
<tr><td>QE for BLE[RA,RE]</td><td>v1.2.0 or later</td><td>A plugin for e²studio to generate skeleton programs for application and profile development.</td></tr>
<tr><td rowspan="4">IAR environment</td><td>IAR Embedded Workbench for ARM</td><td>v8.50.5 or later</td><td>Integrated development environment (IDE) for ARM devices made by IAR Systems.
Note: RE01B device file patch setup is required for v8.50.5 and v9.10.1 environment. For the setup procedure, refer to "RE01B Group IAR Embedded Workbench for Arm patch Setup Guide (R20AN0596)".</td></tr>
<tr><td>IAR C/C++ Compiler for ARM</td><td>v8.50.5 or later</td><td>C/C++ compiler made by IAR Systems.</td></tr>
<tr><td>CMSIS Driver Package</td><td>v1.0.0 or later</td><td>Software package for developing applications for the RE microcontroller series.</td></tr>
<tr><td>QE for BLE[RA,RE]</td><td>-</td><td>Use by overwriting the code that QE for BLE in e²studio generates to IAR project.</td></tr>
<tr><td colspan="2">Renesas Flash Programmer</td><td>v3.06.01 or later</td><td>Tool for programming the on-chip flash memory of Renesas microcontrollers.</td></tr>
<tr><td colspan="2">Integer types</td><td></td><td>Uses ANSI C99 “Exact width integer types”. These types are defined in stdint.h.</td></tr>
<tr><td colspan="2">Endian</td><td></td><td>Little endian.</td></tr>
</table>

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 RE01B 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

RE01B 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 RE01B.) 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 Master and Slave roles, and can operate as Master when connecting to one remote device and as Slave 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.

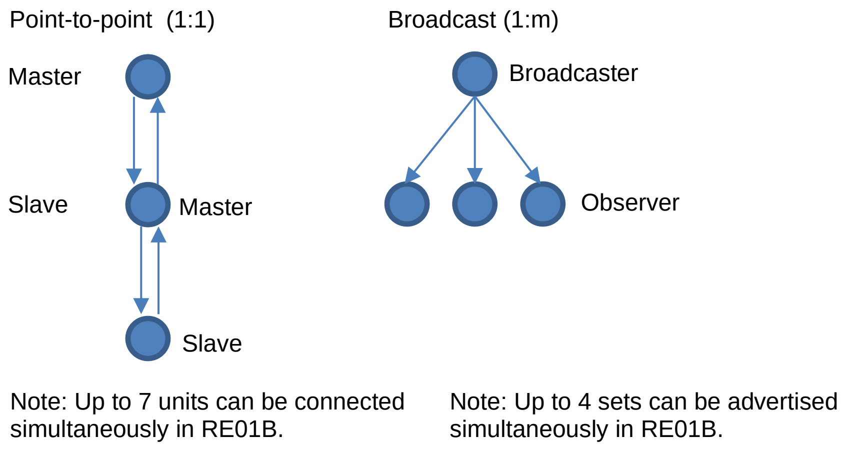

Figure 1.4 Communication topology

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

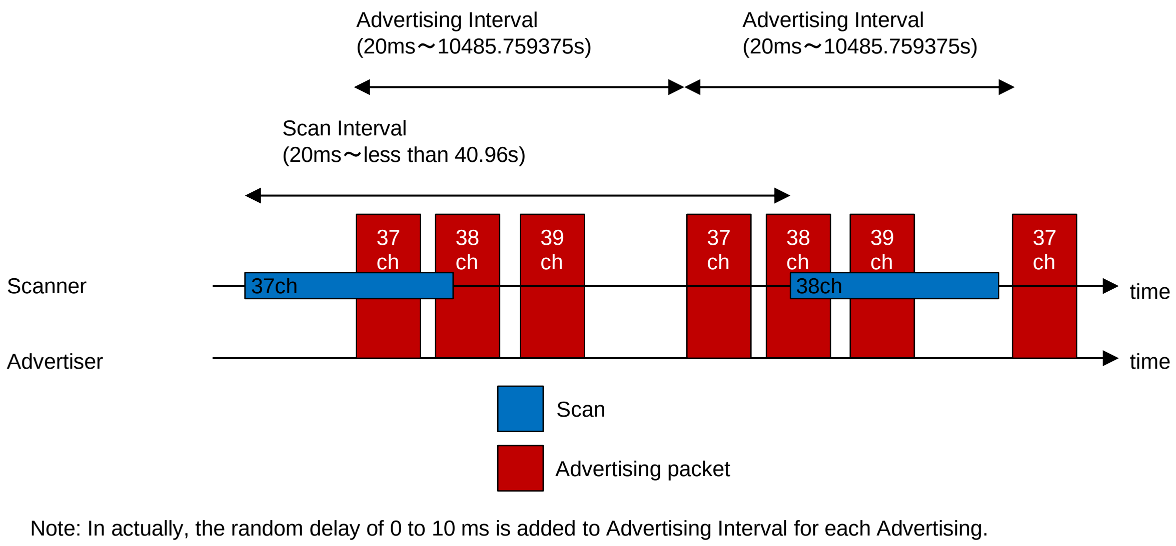

Figure 1.5 Advertising and Scan

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 Master and the other becomes Slave. 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 Master is Client and Slave is Server.

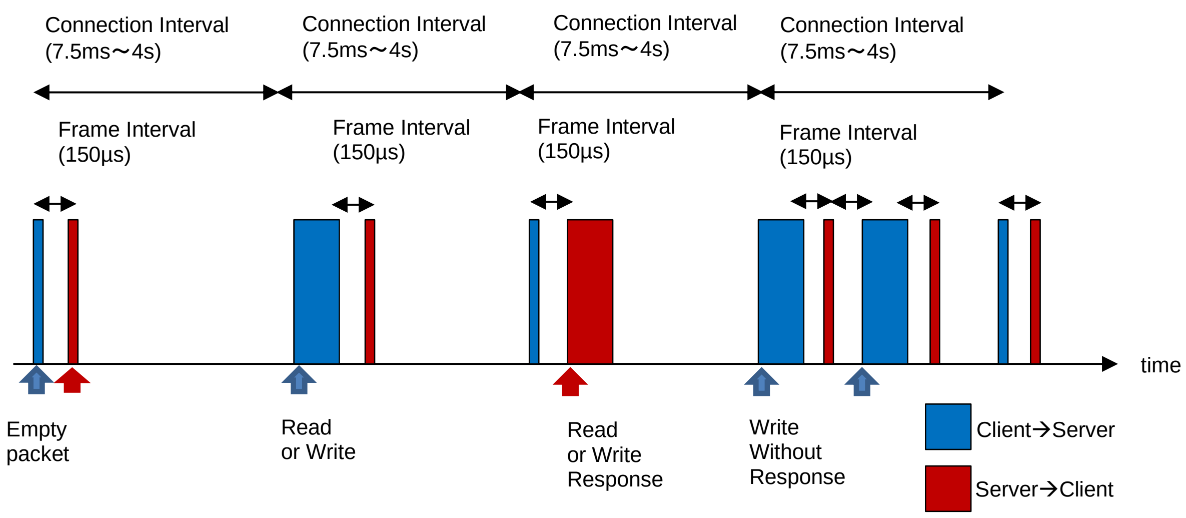

Figure 1.6 Read and Write

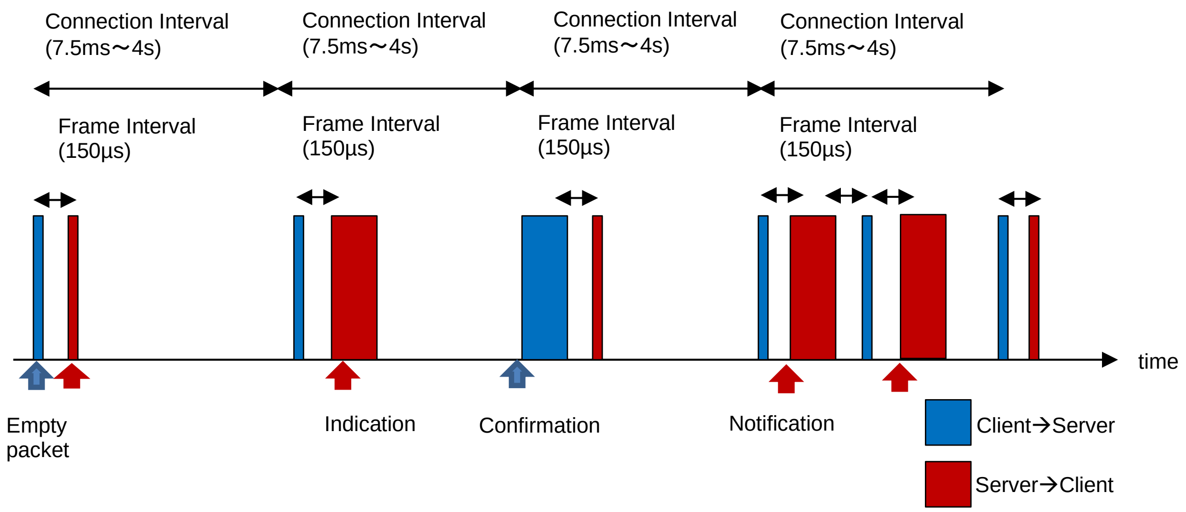

Figure 1.7 Indication and Notification

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”.

1.5 BLE Protocol Stack Operation Overview

The BLE 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 BLE 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 BLE 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 BLE protocol stack, and when there is a time of 40ms 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 60ms 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 60ms or more.
The BLE 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.

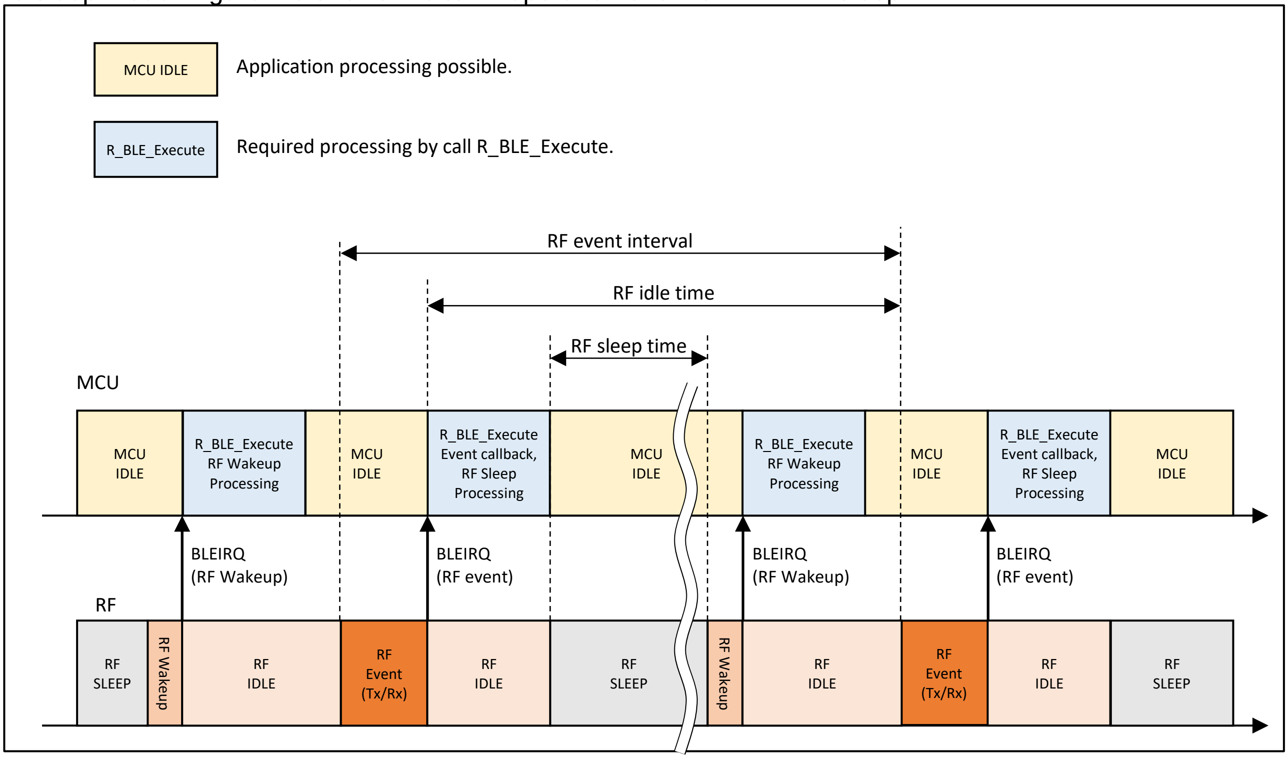

Figure 1.8 MCU/RF operation overview with RF sleep

While the MCU is idle, it is possible to transition the MCU to the low power consumption mode or execute application processing. 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 BLE 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.

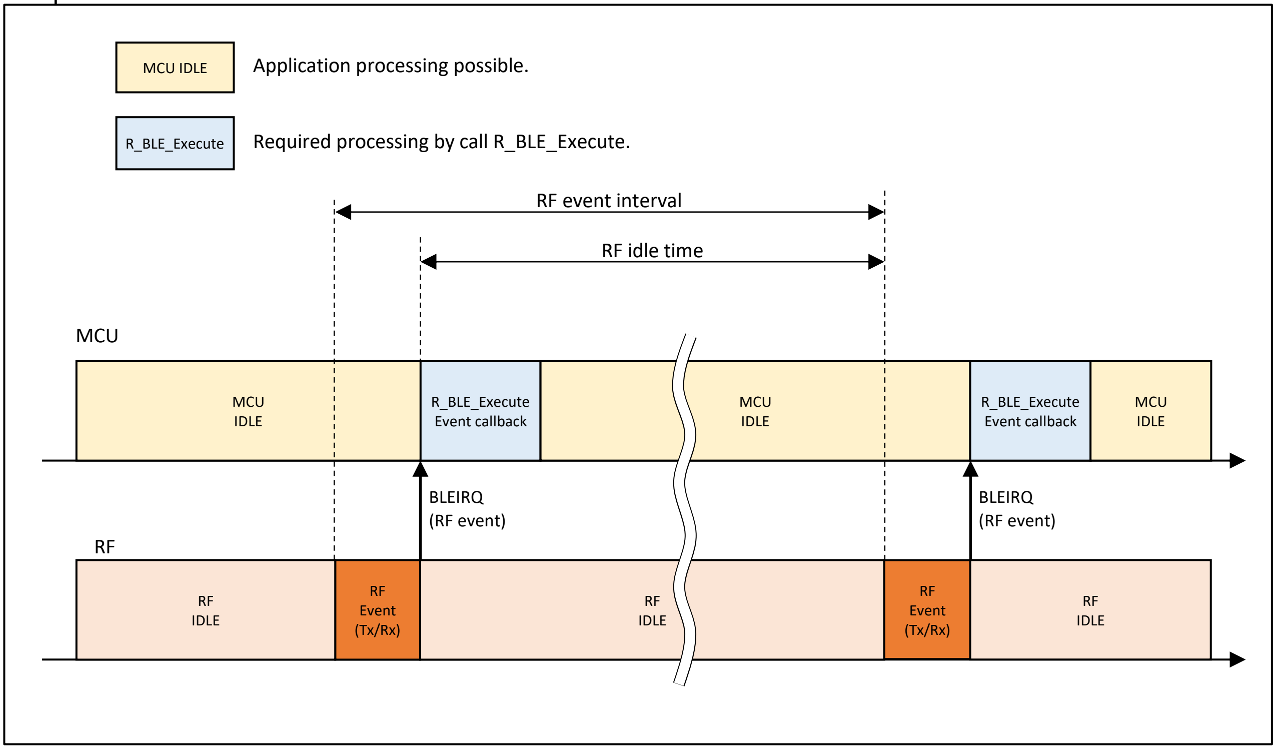

Figure 1.9 MCU/RF operation overview without RF sleep

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 processing is short time. For processing that takes a long time, refer to "3 How to implement user code" and execute the processing by dividing it into multiple times.

1.6 Software structure

To develop the RE01B Bluetooth LE application, it is necessary to develop the application part and profile part shown in Figure 1.10.

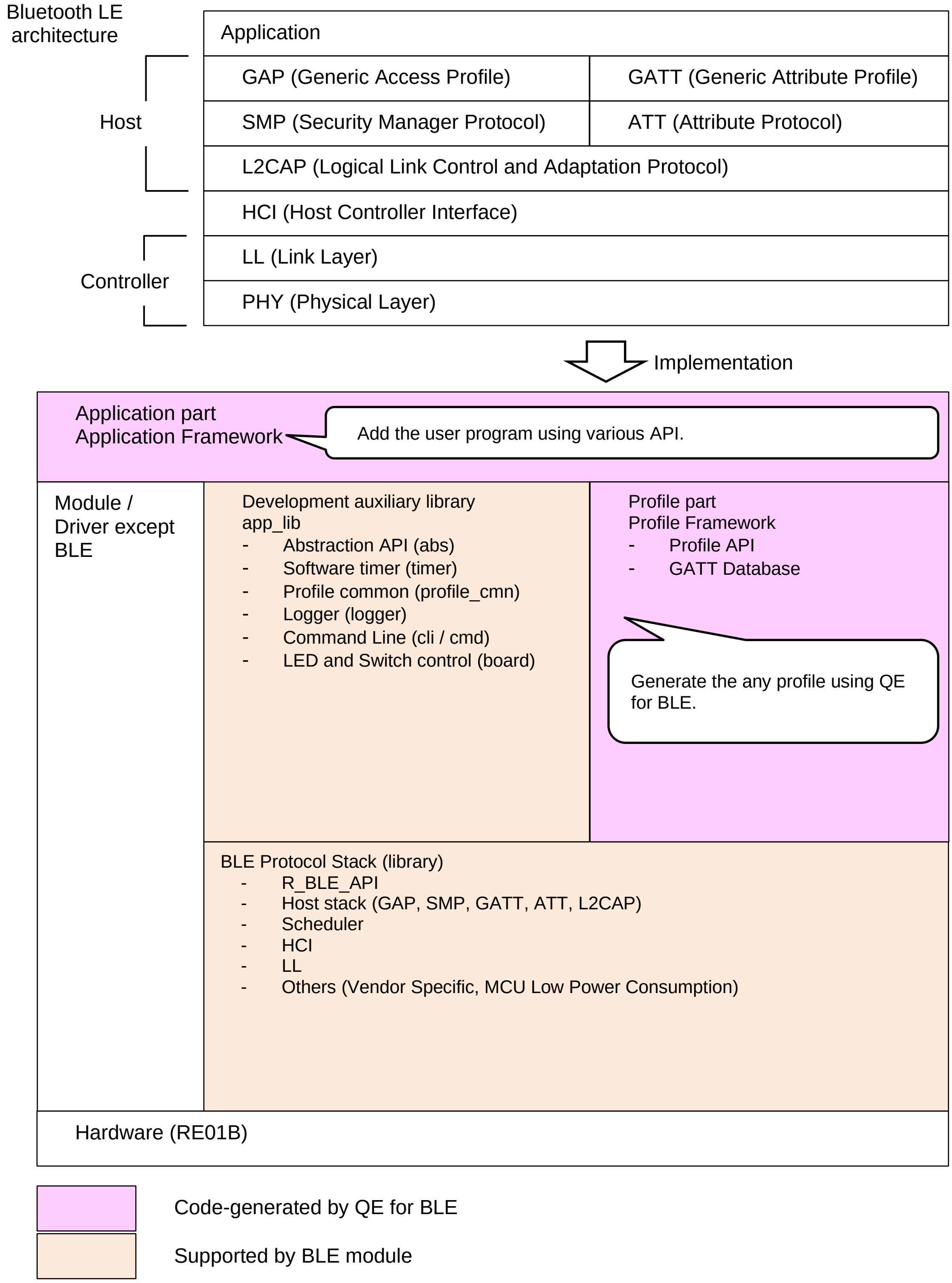

Figure 1.10 Software structure

1.6.1 Primary functions

Constituting BLE module into the project in the integrated development environment e²studio 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 module BLE Protocol Stack app_lib	Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)
Profile Framework	Bluetooth Low Energy Profile Developer's Guide (R01AN5638)
Application Framework	This document

The functions provided by BLE 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_api.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.
Vendor Specific (VS)	R_BLE_VS_XXX	#include "r_ble_api.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_api.h" Enabled in default (BLE_CFG_MCU_LPC_EN) No need to register callback function. API result can be received immediately.

Functions	API/Macro name	Include header and Use
Abstraction API	R_BLE_ABS_XXX	#include "r_ble_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.
Software timer	R_BLE_TIMER_XXX	#include "r_ble_api.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.
Profile common	R_BLE_DISC_XXX R_BLE_SERVC_XXX R_BLE_SERVS_XXX	#include "r_ble_api.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.
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 "r_ble_api.h" Enabled 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 BLE Protocol Stack is reset by “ble reset” command or R_BLE_ABS_Reset. Note: Use Set BLE_CFG_CMD_LINE_EN to ”1”. 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

Functions	API/Macro name	Include header and Use
		loop, call R_BLE_CLI_Process.
LED and Switch control	R_BLE_BOARD_XXX	#include "r_ble_api.h" Enabled 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 Set BLE_CFG_BOARD_LED_SW_EN to "1". 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 BLE Protocol Stack library is selectable according to the feature used in the application. 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 BLE Protocol Stack types and its supporting features

BLE Feature	BLE 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

Using modules / drivers except BLE enables to use the MCU functions except BLE more easily. Modules / Drivers used mainly are shown in Table 1.8.

Table 1.8 Modules / Drivers

Module / Driver name	Comment
R_BLE	BLE basic function **Mandatory** for BLE software
R_SYSTEM	Basic setting for MCU **Mandatory** for clock setting and so on 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.
R_PIN	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.
R_USART	Set and use action mode of UART serial communication Used by Command Line function
R_CORE	Set interrupt of timer and so on **Mandatory** for controlling H/W(RF) Used by Software timer function too

1.7 Flow of development

Develop as the following steps. As for detail, refer to “4. Create project” in “Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)”.

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

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

Import "ble_project_server" in case of creating an application that executes Advertising, or "ble_project_client" in case of creating an application that executes Scan.

(3) 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 Slave from Master such as a PC or smartphone and operates as a GATT server is general. Below is a basic application and its processing.

Table 1.9 Basic application and process

Application	Process	Description
GATT server	Advertising	Refer to "5 Advertising".
	Connection	When receiving a connection request from Master, BLE 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 modules and Bluetooth LE functions with RE01B 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 "5.10 Beacon".

1.9 Locating sections

Memory map as for RAM, and Code Flash ROM (CF) in demo project in RE01B(R7F0E01BD2DNB) and their section placement set by linker are shown in below.

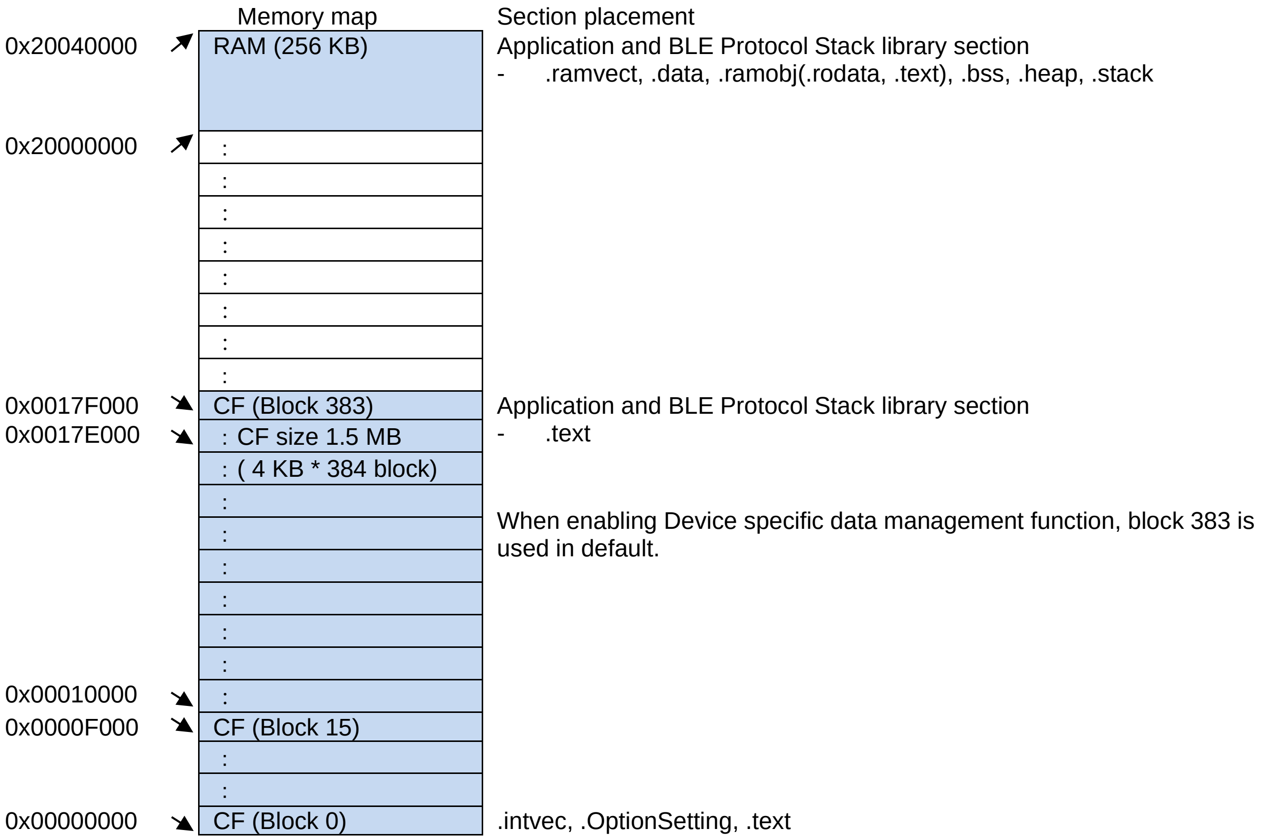

Figure 1.11 Locating sections

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 RE01B 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 383 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 module are located in the r_ble_cfg.h. The macro and setting range are listed in Table 2.1.

Table 2.1 Configuration Options

Macro	Setting range (default)	Description
BLE_CFG_LIB_TYPE	0 - 2 (1)	Type of the BLE Protocol Stack.
BLE_CFG_RF_DBG_PUB_ADDR	Set any value. ({0xFF,0xFF,0xFF,0x50,0x90,0x74})	Initial Public Address.
BLE_CFG_RF_DBG_RAND_ADDR	Set any value. ({0xFF,0xFF,0xFF,0xFF,0xFF,0xFF})	Initial Static Address.
BLE_CFG_RF_CONN_MAX	1 - 7 (2)	Maximum number of simultaneous connections.
BLE_CFG_RF_CONN_DATA_MAX	27 - 251 (251)	Maximum packet data length (bytes).
BLE_CFG_RF_ADV_DATA_MAX	31 - 1650 (252)	Maximum advertising data length (bytes).
BLE_CFG_RF_ADV_SET_MAX	1 - 4 (1)	Maximum number of the advertising set.
BLE_CFG_RF_SYNC_SET_MAX	1 - 2 (1)	Maximum number of periodic sync set.
BLE_CFG_EVENT_NOTIFY_CONN_START	0 - 1 (0)	Enable or disable start interrupt notification of a connection complete event.
BLE_CFG_EVENT_NOTIFY_CONN_CLOSE	0 - 1 (0)	Enable or disable end interrupt notification of a connection complete event.
BLE_CFG_EVENT_NOTIFY_ADV_START	0 - 1 (0)	Enable or disable the advertising event start interrupt notification.
BLE_CFG_EVENT_NOTIFY_ADV_CLOSE	0 - 1 (0)	Enable or disable the advertising event complete interrupt notification.
BLE_CFG_EVENT_NOTIFY_SCAN_START	0 - 1 (0)	Enable or disable the scan start interrupt notification.
BLE_CFG_EVENT_NOTIFY_SCAN_CLOSE	0 - 1 (0)	Enable or disable the scan complete interrupt notification.
BLE_CFG_EVENT_NOTIFY_INIT_START	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	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	0 - 1 (0)	Enable or disable the RF_DEEP_SLEEP start notification.

Macro	Setting range (default)	Description
BLE_CFG_EVENT_NOTIFY_DS_WAKEUP	0 - 1 (0)	Enable or disable the RF_DEEP_SLEEP wakeup notification.
BLE_CFG_RF_CLVAL	0 - 15 (7)	Adjustment value of the 32MHz crystal oscillator.
BLE_CFG_RF_DDC_EN	0 - 1 (1)	Enable or disable the DC-DC on the RF.
BLE_CFG_RF_SCA	250 - 500 (250)	Sleep Clock Accuracy (SCA) for the RF slow clock.
BLE_CFG_RF_MAX_TX_POW	0 - 1 (0)	Maximum transmit power configuration.
BLE_CFG_RF_DEF_TX_POW	0 - 2 (0)	Default transmit power level.
BLE_CFG_RF_DEEP_SLEEP_EN	0 - 1 (1)	Enable or disable the RF Deep Sleep.
BLE_CFG_DEV_DATA_CF_BLOCK	-1 - 383 (383)	The Code Flash (ROM) block stored the device specific data.
BLE_CFG_DEV_DATA_DF_BLOCK	- (-1)	Not support in RE01B.
BLE_CFG_GATT_MTU_SIZE	23 - 247 (247)	The MTU size (bytes) for the GATT communication.
BLE_CFG_NUM_BOND	- (7)	Not support in RE01B.
BLE_CFG_EN_SEC_DATA	- (0)	Not support in RE01B.
BLE_CFG_SECD_DATA_DF_BLOCK	- (0)	Not support in RE01B.
BLE_CFG_CMD_LINE_EN	0 - 1 (1)	Enable or disable the command line function.
BLE_CFG_BOARD_LED_SW_EN	0 - 1 (1)	Enable or disable support the board LED & Switch control.
BLE_CFG_LOG_LEVEL	0 - 3 (3)	Log level.
BLE_CFG_ABS_API_EN	0 - 1 (1)	Enable or disable support the Abstraction API.
BLE_CFG_SOFT_TIMER_EN	0 - 1 (1)	Enable or disable support the software time in app_lib.
BLE_CFG_MCU_LPC_EN	0 - 1 (1)	Enable or disable support the MCU low power consumption control.
BLE_CFG_HCI_MODE_EN	0 - 1 (0)	Select start in HCI mode or not.
BLE_CFG_SOFT_TIMER_AGT_CH	0 - 1 (1)	AGT channel for the software timer function.
BLE_CFG_AUTO_READ_ADC_EN	0 - 1 (0)	Enable or disable the automatically reading A/D convertor (ADC).

Macro	Setting range (default)	Description
BLE_CFG_MCU_PSM_OPE	0 - 1 (1)	Select MCU Power Supply Mode during Operating mode (OPE).
BLE_CFG_MCU_VBB_SSTBY	0 - 1 (1)	Enable or disable Back Bias Voltage (VBB) control of MCU power control mode during Software standby mode (SSTBY).

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: SC display name	Configuration Options: Macro	Setting range (default)	Library	Additional Size (bytes)
Maximum number of connections	BLE_CFG_RF_CONN_MAX	1 - 7 (2)	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 (252)	All features	Described in Table 2.3
Maximum advertising set number [*1]	BLE_CFG_RF_ADV_SET_MAX	1 - 4 (1)	All features	308
Maximum periodic sync set number [*2]	BLE_CFG_RF_SYNC_SET_MAX	1 - 2 (1)	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.
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: BLE Protocol Stack does not check address format.
	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.

RE01B provides the function to store the static BD address such as Public device address and Static address in the user area of the internal ROM. Code flash (CF) can be used as the user area. They are set as follows in default by Configuration option.

Table 2.5 BD address configurations

Configuration option	Initial value
BLE_CFG_DEV_DATA_CF_BLOCK	383 (CF block 383 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)

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 "2.3.1 How to use random address of device specific data".

The adopted BD address is determined as below at application startup according to "6.2.4 BD address adoption flow" in "Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)".

Table 2.6 BD address adoption method

Priority	BD address adoption method	Initial value	Description
1	CF is used. (BLE_CFG_DEV_DATA_CF_BLOCK is set 0 to 383.) 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 RE01B memory protection function, it can be guarded against being rewritten by third parties.
2	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.
3	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 BLE 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 to CF, refer to "5.6.3 Writing to user area (ROM)" in "Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)".
As for details on RE01B memory protection function, refer to "48.5.1.4 Startup Area Select" in "RE01B Group Product with 1.5-Mbyte Flash Memory User’s Manual: Hardware (R01UH0903)".

2.3.1 How to use random address of device specific data

The following is a sample code for advertising with a random address determined by device-specific data. 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 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 the managed RAM of BLE 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 curr rnd
$ BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8
 addr:D9:7C:E6:81:83:35 rnd on current register
```

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 in r_core_cfg.h. Note: Make non-used clocks disable or set minimum clock frequency.
	ICLK/PCLKA : x1 (32MHz)	
	PCLKB : x1 (32MHz)	
r_ble_cfg.h component set	DC-DC on the RF: Enable (BLE_CFG_RF_DDC_EN=1)	Note: Refer to "RE01B Group Hardware Design Guide (R01AN5471)".
	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.
	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, BLE 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.

- NORMAL+SLEEP mode, NORMAL+SSTBY mode, and VBB+SSTBY mode (*1) are available as Low power consumption state.

  *1: The transition to VBB + SSTBY requires BLE_CFG_MCU_VBB_SSTBY to be set to 1.
- 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 “12. Power-Saving Functions” in “RE01B Group Product with 1.5-Mbyte Flash Memory User’s Manual: Hardware (R01UH0903)”.
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

- VBB+SSTBY mode
  In *check_vbb_software_standby()* function, add processing to check if there is no problem even if the component enters to VBB+SSTBY mode. Add processing to the location of "/* add check for other components */" comment in Code 2-2.

```
static bool check_vbb_software_standby(void)
{
    /* inhibit flag is true */
    if (g_inhibit_software_standby)
    {
        return false;
    }
    /* When the RF_SLEEP function is enabled and RF is active,
     * the transition to VBB mode is restricted because RF interrupts will be occured. */
    if( (0 != r_ble_rf_power_save_mode()) &&
        (BLE_RF_STATE_POW_SAVE != R_BLE_GetRfPowerState()) )
    {
        return false;
    }

    /* add check for other components */

    return true;
}
```

Code 2-2 Location to check for transition to VBB+SSTBY mode

- NORMAL+SSTBY mode
  In *check_normal_software_standby()* function, add processing to check if there is no problem even if entering to NORMAL+SSTBY mode. Add processing to the location of "/* add check for other components */" in Code 2-3.

```
static bool check_normal_software_standby(void)
{
    /* inhibit flag is true */
    if (g_inhibit_software_standby)
    {
        return false;
    }

    /* add check for other components */

    return true;
}
```

Code 2-3 Location to check for transition to NORMAL+SSTBY 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)
{
    switch(mode)
    {
        case LPC_LP_NORMAL_SLEEP:
        {
        (OMISSION)

            /* add implementation for transiting the MCU sleep mode. */

        } break;
        case LPC_LP_NORMAL_SSTBY:
        case LPC_LP_VBB_SSTBY:
        {
#if (BLE_CFG_CMD_LINE_EN)
            R_BLE_CLI_Terminate();
#endif /* (BLE_CFG_CMD_LINE_EN) */

            /* add implementation for transiting the MCU software standby mode. */

        } break;
        (OMISSION)
}
```

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)
{
    switch(mode)
    {
        case LPC_LP_NORMAL_SLEEP:
        {
        (OMISSION)

            /* add implementation for transiting the active state. */

        } break;
        case LPC_LP_NORMAL_SSTBY:
        case LPC_LP_VBB_SSTBY:
        {
            {
                extern ble_status_t R_BLE_SetRfWakeup(void);
                (void)R_BLE_SetRfWakeup();
            }
#if (BLE_CFG_CMD_LINE_EN)
            R_BLE_CLI_Init();
#endif /* (BLE_CFG_CMD_LINE_EN) */

            /* add implementation for transiting the active state. */

        } break;
        (OMISSION)
}
```

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

3. How to implement user code

As for basic implementation method of user code, refer to “5 How to implement user code” in “Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)”.

As for how to implement advertising of RF event shown in "1.5 BLE Protocol Stack Operation Overview", refer to “5 Advertising”. As for scanning, refer to “6 Scan”. As for initiating, refer to “7 Connection”. As for connection, refer to “8 Communication”.

In order to perform application development synchronized with RF event, it is necessary to use the Event notification function and the RF communication timing notification function. The following shows how to use their function.

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

Table 3.1 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 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 BLE_CFG_CMD_LINE_EN.

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

```
[Device\BLE\platform\r_ble_pf_functions.c]
(OMISSION)

#define pf R_BLE_CLI_Printf
void rf_ntf_recv_event(void)
{
        pf("RF event has come!!\n");
}

(OMISSION)

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)

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)

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 3-1 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 "r_ble_api.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 3-2 Sample log display of RF communication timing notification (app_main.c)

4. app_lib

4.1 Abstraction API

The Abstraction API is intended to make it easier to use the functions often used in the BLE protocol stack. The Abstraction API internally uses GAP, GATT server, GATT client, and Vendor Specific API to realize each function. Table 4.1 shows the APIs called by the Abstraction APIs and the events notified as a result. Refer to the API document (r_ble_api_spec.chm) for detailed specifications of each Abstraction API.

Table 4.1 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	BLE 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

Abstraction API	Description	API to use	Events
			BLE_GAP_EVENT_ADV_OFF
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

4.2 Software Timer

Refer to “3.5 Software timer” in “Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)”.

4.3 Profile common

This function provides the common interfaces (Service Discovery, GATT Client process, and GATT Server process) in the BLE Profile. The interfaces are call by the code generated by the QE for BLE. Refer to “10 Profile and service” and “Bluetooth Low Energy Profile Developer’s Guide(R01AN5638)” for the details of the profile common and the profile development.

4.4 Logger

Refer to “3.7 Logger” in “Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)”.

4.5 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, 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

By default, the commands in Table 4.3 are supported. For more information about the commands, refer to "Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)".

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.
	scan_ch_map	Set/Get the scan channel map.
sys	stby	Set software standby mode.
ble	reset	Reset the BLE Protocol Stack.
	close	Terminate the BLE Protocol Stack.

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

4.5.1 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 "r_ble_api.h"
```

Code 4-1 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 command line.
R_BLE_CLI_RegisterCmds	Register the commands.
R_BLE_CMD_SetResetCb	Register a callback that restarts the BLE 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_init();
}

/** some code is omitted  **/

/* Initialize BLE Protocol Stack */
static ble_status_t ble_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 4-2 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 4-3 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 4-4 Sample of adding the command line to the main loop

4.5.2 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 4-5 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 4-6 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 4-7 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 4-8 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 4-9 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 4-10 Sample of initialization and command registration

4.6 LED and Switch control

Applications can use the LED and Switch control function to easily control the LEDs and Switch on the board. When using Command Line Interface features, set the BLE_CFG_BOARD_LED_SW_EN configuration option to “1”. LED and Push-switch on the board can be controlled by setting the configuration options shown in Table 4.7 and Table 4.8 according to the board environment.

Table 4.7 LED and Push-switch Configuration Options

Configuration options (r_ble_board.c)	
Macro	Setting range (default)
SW2_IRQ	0,1,5,6,7 (1)
LED2_PORT	000, 001, 002, 003, 012, 500, 505, 506, 102, 107, 112, 113, 606, 607, 608, 609, 300, 301, 305, 700, 701, 704, 200, 201, 207, 411, 412, 413 (506)

Table 4.8 IRQ Configuration Options

Configuration options (r_system_cfg.c)		
Macro	Setting range	Description
SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0	SYSTEM_IRQ_EVENT_NUMBER_NOT_USED or SYSTEM_IRQ_EVENT_NUMBER0 (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)	Set whether to generate an IRQ0 interrupt by inputting P411 or P506.
SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ1	SYSTEM_IRQ_EVENT_NUMBER_NOT_USED or SYSTEM_IRQ_EVENT_NUMBER1 (SYSTEM_IRQ_EVENT_NUMBER1)	Set whether to generate an IRQ1 interrupt by inputting P207 or P505.
SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ5	SYSTEM_IRQ_EVENT_NUMBER_NOT_USED or SYSTEM_IRQ_EVENT_NUMBER5 (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)	Set whether to generate an IRQ5 interrupt by inputting P113.
SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ6	SYSTEM_IRQ_EVENT_NUMBER_NOT_USED or SYSTEM_IRQ_EVENT_NUMBER6 (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)	Set whether to generate an IRQ6 interrupt by inputting P112.
SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ7	SYSTEM_IRQ_EVENT_NUMBER_NOT_USED or SYSTEM_IRQ_EVENT_NUMBER7 (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)	Set whether to generate an IRQ7 interrupt by inputting P107.

Modify the pin settings shown in Table 4.9 according to needs.

Table 4.9 Pin setting functions

Pin setting functions (pin.c)	
Function	Description
R_ICU_Pinset_CH0	Set either P411 or P506 as the input pin. By default, P411 is set.
R_ICU_Pinset_CH1	Set either P207 or P505 as the input pin. By default, P505 is set.
R_ICU_Pinset_CH5	Set P113 as the input pin.
R_ICU_Pinset_CH6	Set P112 as the input pin.
R_ICU_Pinset_CH7	Set P107 as the input pin.

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 4-11 Inclusion of LED and Switch control header file

4.6.1 LED and Switch 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 4-12 LED and Switch control initialization

4.6.2 ON or OFF 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.

4.6.3 Callback for pressing Switch

Call R_BLE_BOARD_RegisterSwitchCb to register a function to process after pressing Switch.
An example of LED2 turned on/off by sw_cb when pressing SW2 is shown below.

```
static void sw_cb(void)
{
    R_BLE_BOARD_ToggleLEDState(BLE_BOARD_LED2);
}

/** 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 4-13 Sample of callback allocated for Switch press

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 5.4 are performed by an Abstraction advertising API call. Regarding to the way of using the API, refer to 5.8.

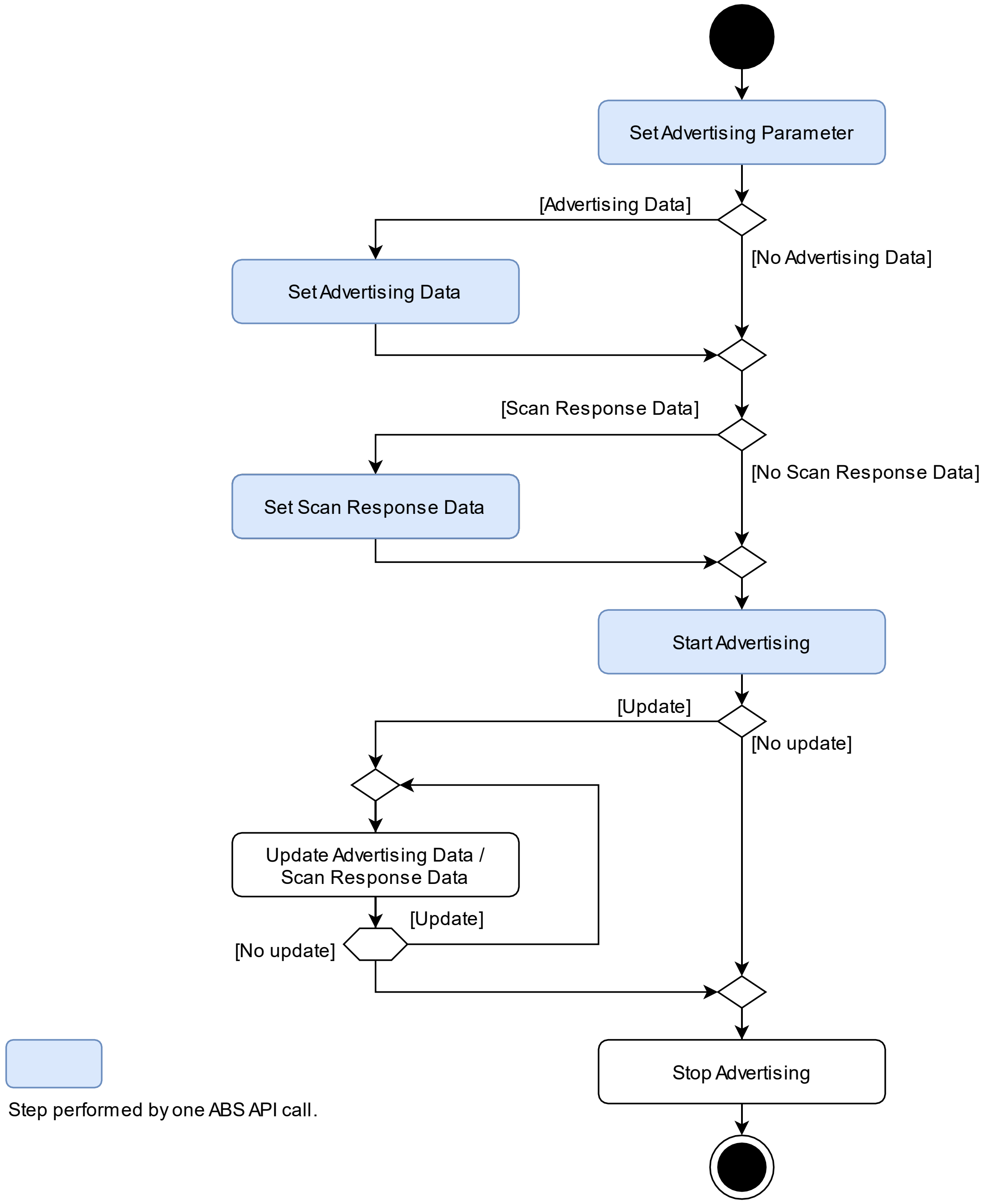

Figure 5.1 Advertising Procedure

5.2 Advertising Parameter

It is necessary to set the advertising parameters by R_BLE_GAP_SetAdvParam to starting 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 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 by HCI on the receiver, 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 BLE Protocol Stack library type. All features library supports legacy and extended advertising. Balance and Compact libraries support only 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.

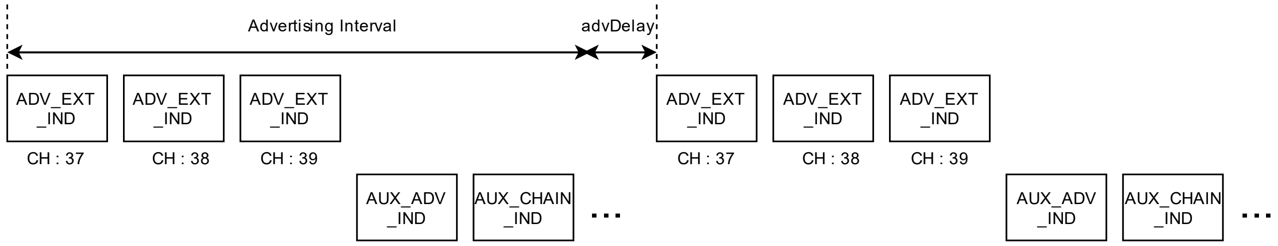

Figure 5.2 Extended Advertising PDU

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 by HCI on the receiver, combine them on the receiver if necessary.

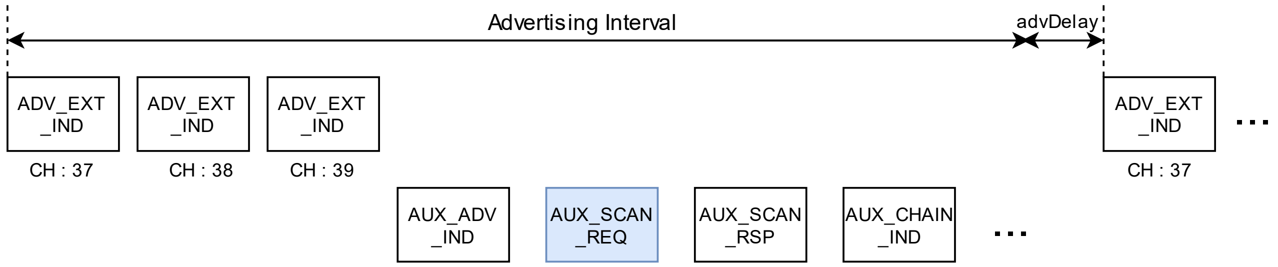

Figure 5.3 Scannable Advertising PDU

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.

5.2.2 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.
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)	Process scan and connection requests from all devices.
BLE_GAP_ADV_ALLOW_ SCAN_WLST_CONN_ANY(0x01)	Process connection requests from all devices and scan requests from only devices that are in the White List.
BLE_GAP_ADV_ALLOW_ SCAN_ANY_CONN_WLST(0x02)	Process scan requests from all devices and connection requests from only devices that are in the White List.
BLE_GAP_ADV_ALLOW_ SCAN_WLST_CONN_WLST(0x03)	Process scan and connection requests from only devices in the White List.

5.2.3 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 and resolve 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 regularly.

Table 5.4 The parameters used for the privacy feature

<table>
<tr><th>Field</th><th>Value</th><th>Description</th></tr>
<tr><td rowspan="2">o_addr_type</td><td>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)</td><td>Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is public address.</td></tr>
<tr><td>BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)</td><td>Specify the value if the Identity Address registered by R_BLE_GAP_SetLocIdInfo is public address.</td></tr>
<tr><td>p_addr_type</td><td rowspan="2">Specify the remote device identity address registered by R_BLE_GAP_ConfRslvList().</td><td rowspan="2">—</td></tr>
<tr><td>p_addr</td></tr>
</table>

5.2.4 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.

5.3 Advertising Data / Scan Response Data

For details about setting Advertising Data / Scan Response Data, refer to "5.7 Advertising Data / Scan Response Data / Periodic Advertising Data".
For details updating Advertising Data / Scan Response Data setting, refer to "5.7.2 Advertising Data Update".

5.4 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).

5.5 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 with a single HCI command, the advertising needs to be stopped before update.

5.6 Periodic Advertising

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.

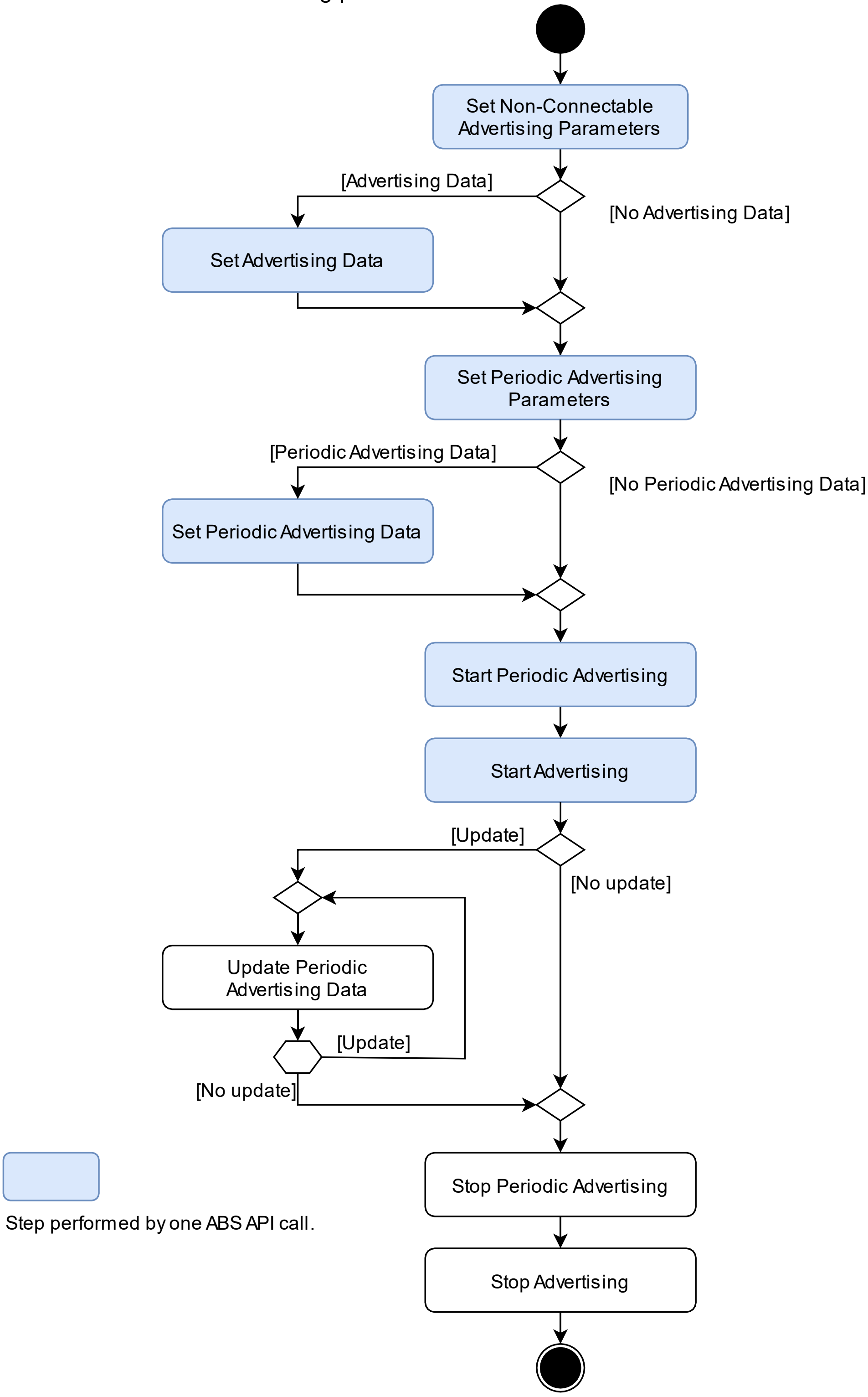

Figure 5.4 Periodic Advertising procedure

5.6.1 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

5.6.2 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 by HCI on the receiver, combine them on the receiver if necessary.

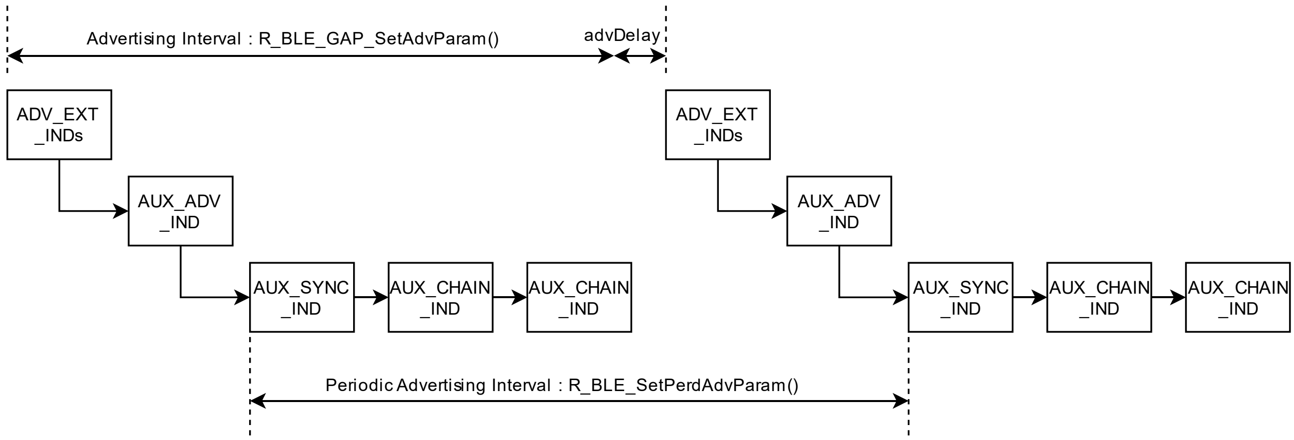

Figure 5.5 Periodic Advertising PDUs

5.6.3 Periodic Advertising Data

For details about setting Periodic Advertising Data, refer to “5.7 Advertising Data / Scan Response Data / Periodic Advertising Data”.
For details updating Periodic Advertising Data, refer to “5.7.3 Periodic Advertising Data Update”.

5.6.4 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

5.6.5 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 with a single HCI command, the Periodic Advertising needs to be stopped before update.

5.7 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.

5.7.1 Format

Figure 5.6 shows the data format.

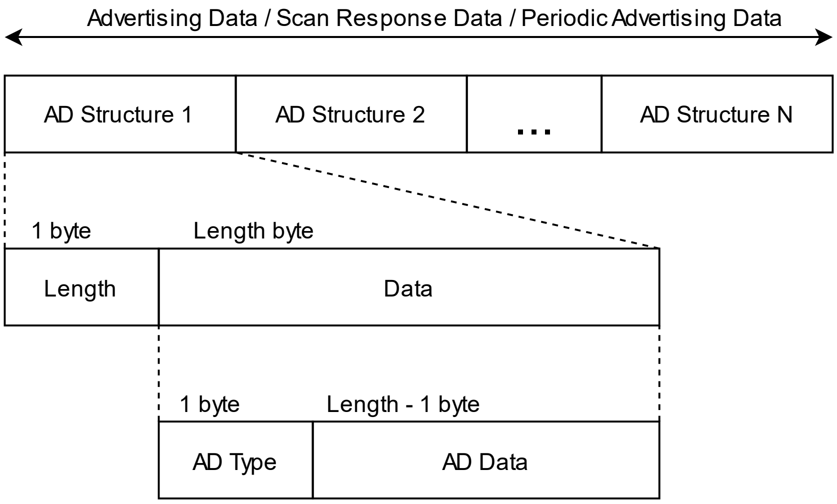

Figure 5.6 Advertising Data / Scan Response Data / Periodic Advertising Data format

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 shows the head of the device name to the middle.</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

5.7.2 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 "5.5" and update the data by R_BLE_GAP_SetAdvSresData.

5.7.3 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 "5.6.5" and update the data by R_BLE_GAP_SetAdvSresData.

5.7.4 Buffer Size

The size of the buffer for Advertising Data / Scan Response Data in the BLE 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 BLE 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.

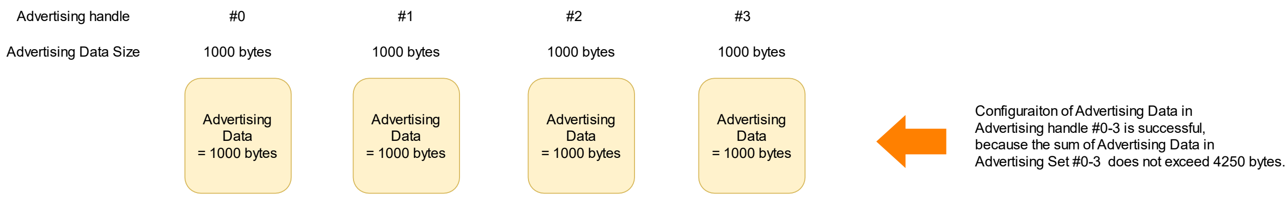

Figure 5.7 Successful sample of setting Advertising Data

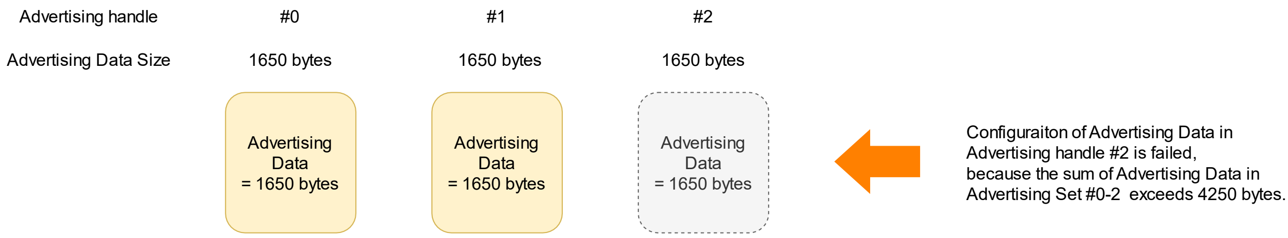

Figure 5.8 Failed sample of setting Advertising Data

5.8 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	1	245
			AUX_ADV_IND		
		Connectable Directed	ADV_EXT_IND		239
			AUX_ADV_IND		
R_BLE_ABS_StartNonConnAdv	Legacy	Non-Connectable and Non-Scannable Undirected	ADV_NONCONN_IND	2	31
	Extended		ADV_EXT_IND		BLE_CFG_RF_ADV_DATA_MAX
			AUX_ADV_IND		
			AUX_CHAIN_IND		
		Non-Connectable and Non-Scannable Directed	ADV_EXT_IND		BLE_CFG_RF_ADV_DATA_MAX
			AUX_ADV_IND		
			AUX_CHAIN_IND		
R_BLE_ABS_StartPerdAdv	Extended	Periodic	ADV_EXT_IND	3	BLE_CFG_RF_ADV_DATA_MAX
			AUX_ADV_IND		
			AUX_SYNC_IND		
			AUX_CHAIN_IND		

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

5.8.1 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.
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 .

5.8.2 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 and resolve 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 regularly.

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.
p_addr	Specify the remote device identity address registered by R_BLE_GAP_ConfRslvList().	—

5.9 Connection with Smart Phone

Call R_BLE_ABS_StartLegacyAdv to send connectable Legacy Advertising packets to connect with 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 Phone can detect the device to connect.

5.10 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://developers.google.com/beacons/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 BLE 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_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 scan interval, scan window, duration and period field specify the interval and period of scan. Figure 6.1 shows those parameters relationship.

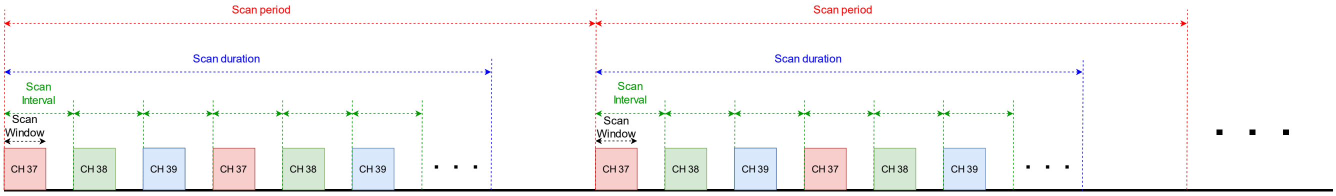

Figure 6.1 The relationship of scan interval, window, duration, period

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.

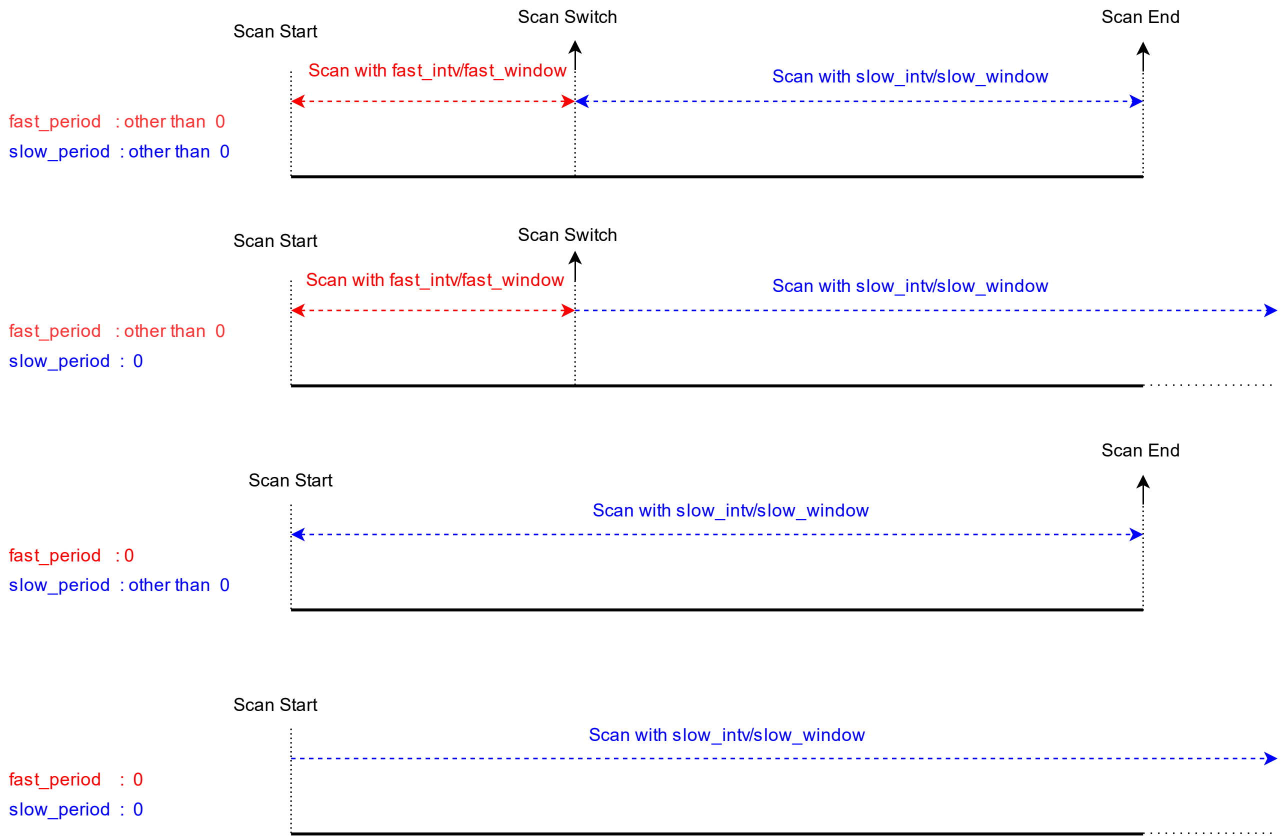

Figure 6.2 The relationship between the fast scan and slow scan

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 and resolve local device RPA”, prepare for the privacy feature in advance. If the local device use RPA, call R_BLE_GAP_StartScan. Table 6.7 shows the fields in the st_ble_gap_scan_param_t structure(Table 6.1) to enable the privacy feature. Because the peer device address type in the scan request by R_BLE_ABS_StartScan is fixed to public address, this API does not support the privacy feature.

Table 6.7 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.

6.3 Received information by scan

After calling the Start Scan API, the BLE Protocol Stack notifies receiving an advertising packet from another device by 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 by the receiver HCI is 229 byte or less, 230 byte or more Advertising Data will be notified separately. Combine them on the receiver if necessary.

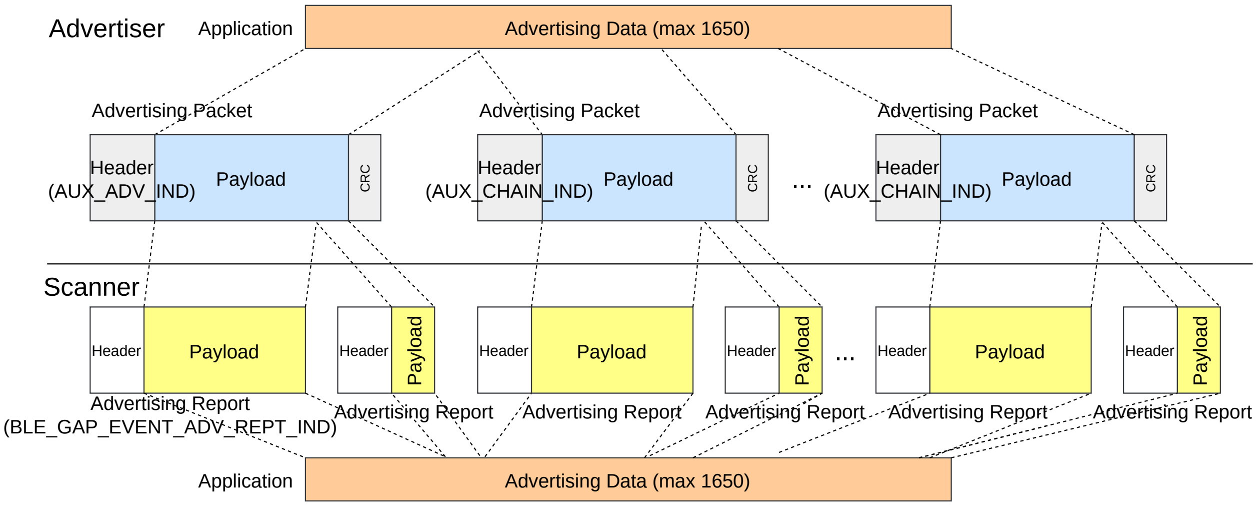

Figure 6.3 Dividing and combining Advertising Data

Received advertising packet is stored in a st_ble_gap_adv_rept_evt_t structure variable. Table 6.8 shows st_ble_gap_adv_rept_evt_t structure.

Table 6.8 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. Note: Advertising Data of 230 bytes or more will be notified separately.
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. Note: Periodic Advertising Data of 248 bytes or more will be notified separately.
} param;		

Depending on the BLE Protocol Stack, the field of advertising varies. Table 6.9 and Table 6.10 show the advertising field.

Table 6.9 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 6.10 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. Note: When combining the divided Advertising Data, refer to the more data bit to combine.
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 (r_ble_api_spec.chm).

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.
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 packet 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 6.11 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 6.11 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.

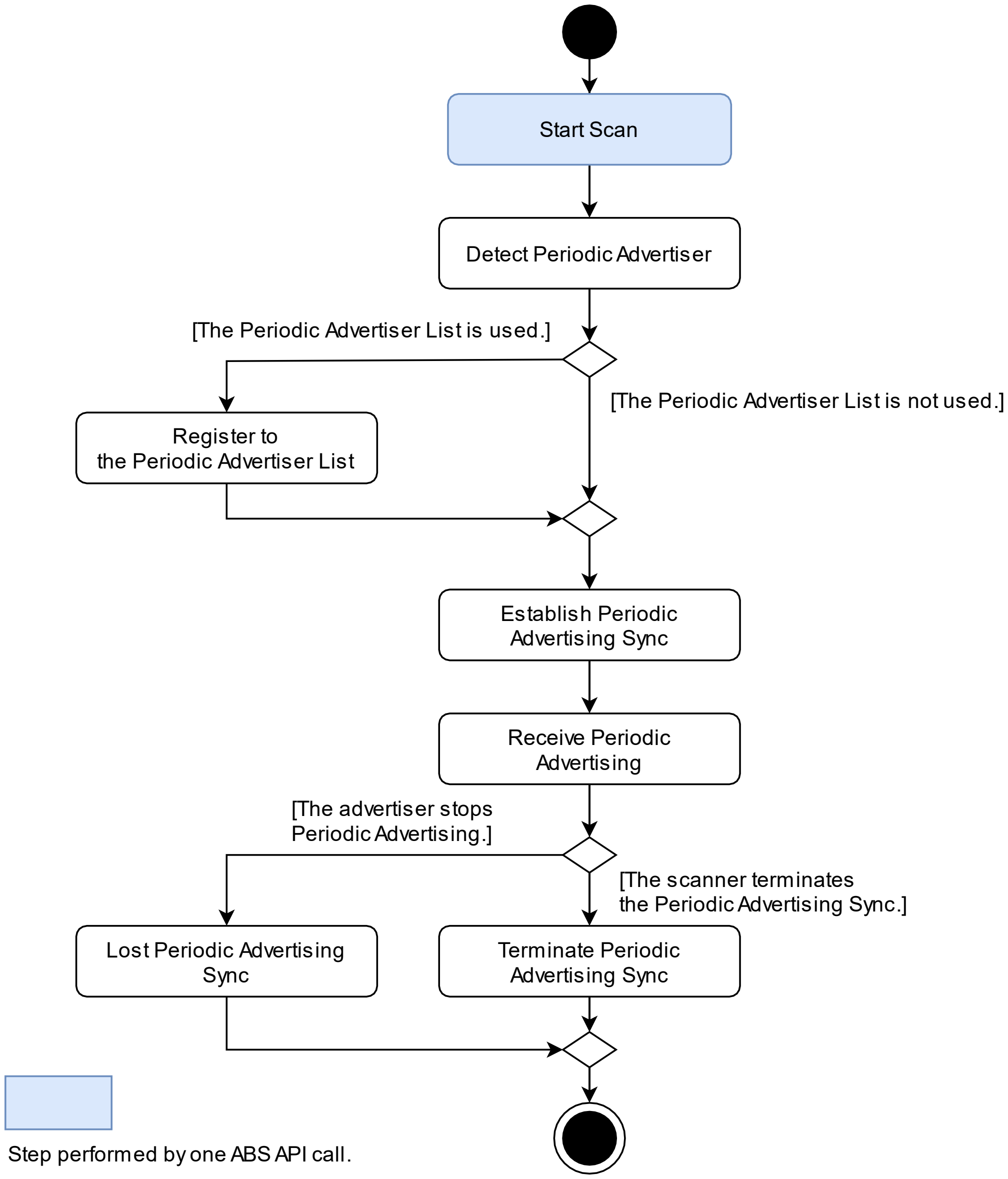

Figure 6.4 Periodic Advertising Sync procedure

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 6.10) included in a received advertising packet is not 0. Specify the advertiser with the addr_type, p_addr, adv_sid field in Table 6.10 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 6.8) variable. Table 6.12 shows the st_ble_gap_perd_adv_rept_t structure in case of Periodic Advertising.

Table 6.12 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 Note: When combining the divided Periodic Advertising Data, refer to data_status to combine.
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 by the BLE_GAP_EVENT_SYNC_LOST event is notified.

6.5.7 Terminate Periodic Advertising Sync

If the scanner terminates the Periodic Advertising Sync, call BLE_GAP_TerminateSync. When the Periodic Advertising Sync has been terminated, the BLE_GAP_EVENT_SYNC_TERM event is notified.

7. Connection

7.1 Requesting Connection

Central device sends a connection request by the below APIs.
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

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.
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 and resolve local device RPA”, prepare for the privacy feature in advance. If the local device use RPA, call R_BLE_GAP_CreateConn. Table 7.1 shows the fields in the st_ble_gap_create_conn_param_t structure to enable the privacy feature. Because the peer device address type in the connection request by R_BLE_ABS_CreateConn is fixed to public address, this API does not support the privacy feature.

Table 7.1 The parameters used for the privacy feature

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.	—

7.2 Cancelling Connection Request

A connection request cannot be sent until the connection is established by previous connection request or 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 BLE 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.

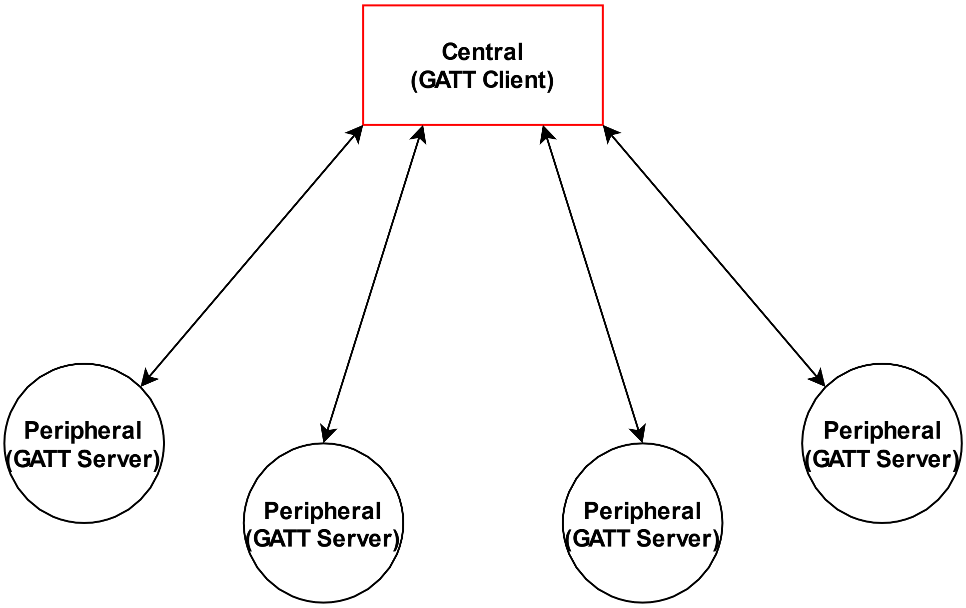

Figure 7.1 Connection with multiple peripheral devices

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 BLE protocol stack. Repeat this procedure to connect multiple peripheral devices.

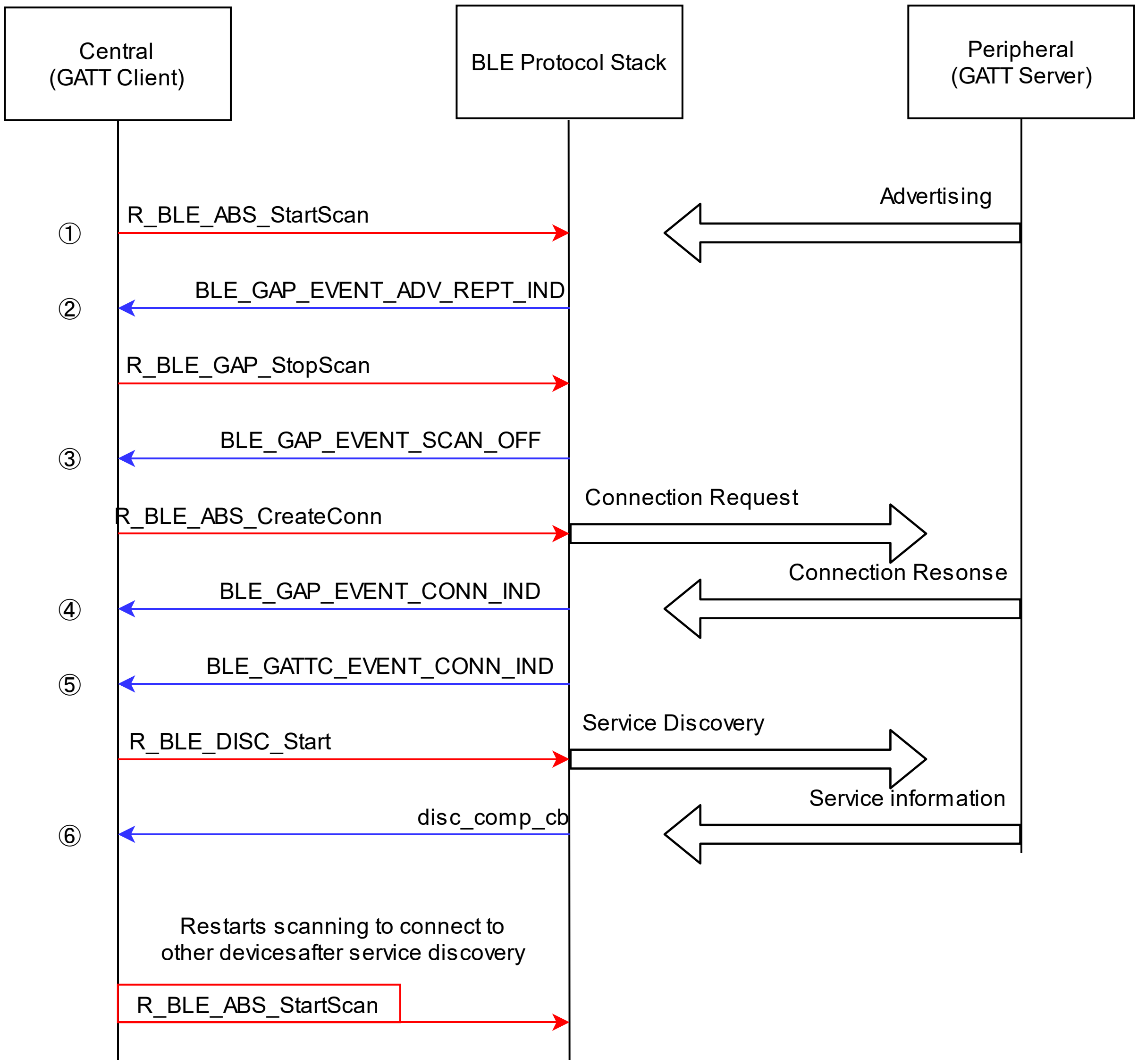

Figure 7.2 Sequence chart when connecting to a peripheral device (The circled numbers in the chart correspond to the numbers in Code 7-3 below.)

```
/* 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.

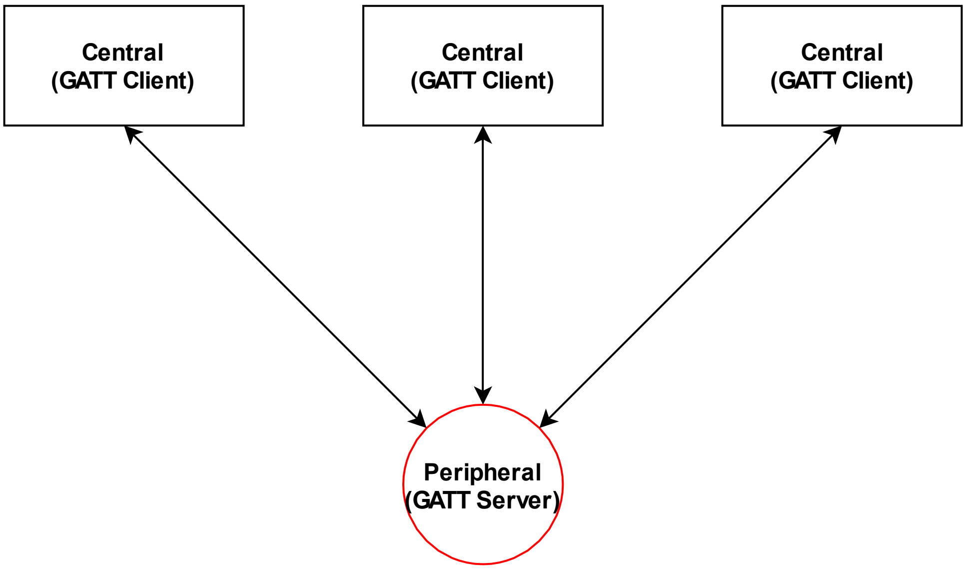

Figure 7.3 Connection with multiple central devices

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 BLE protocol stack. Repeat this procedure to accept connections from multiple central devices.

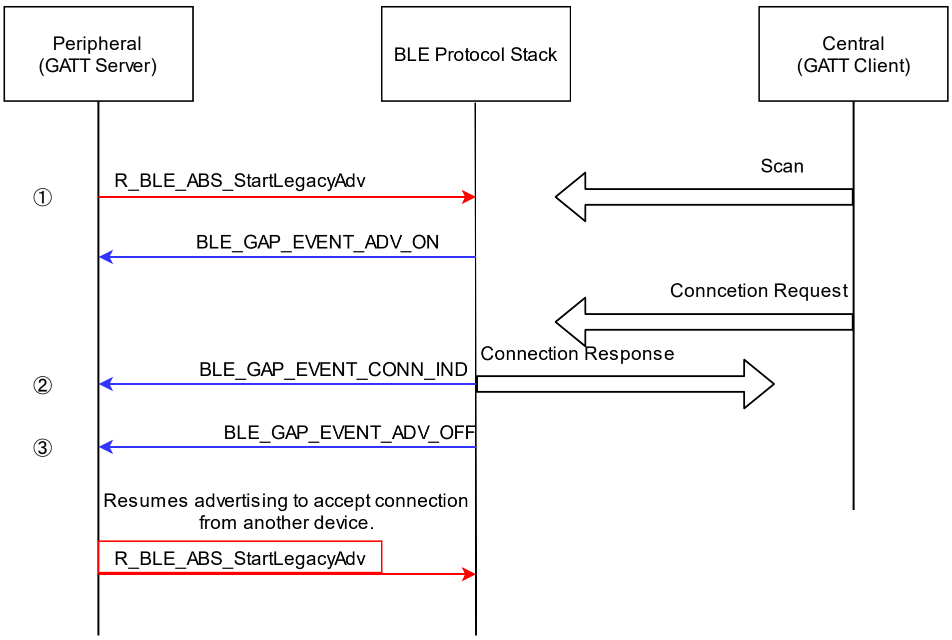

Figure 7.4 Sequence chart when connecting to a central device (The circled numbers in the chart correspond to the numbers in Code 7-6 below.)

```
/* 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 master (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 slave latency and supervision timeout time. For updating 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 BLE Protocol Stack are changed, and different values are held for up to 7 clients. A database value is returned for each client accessed.

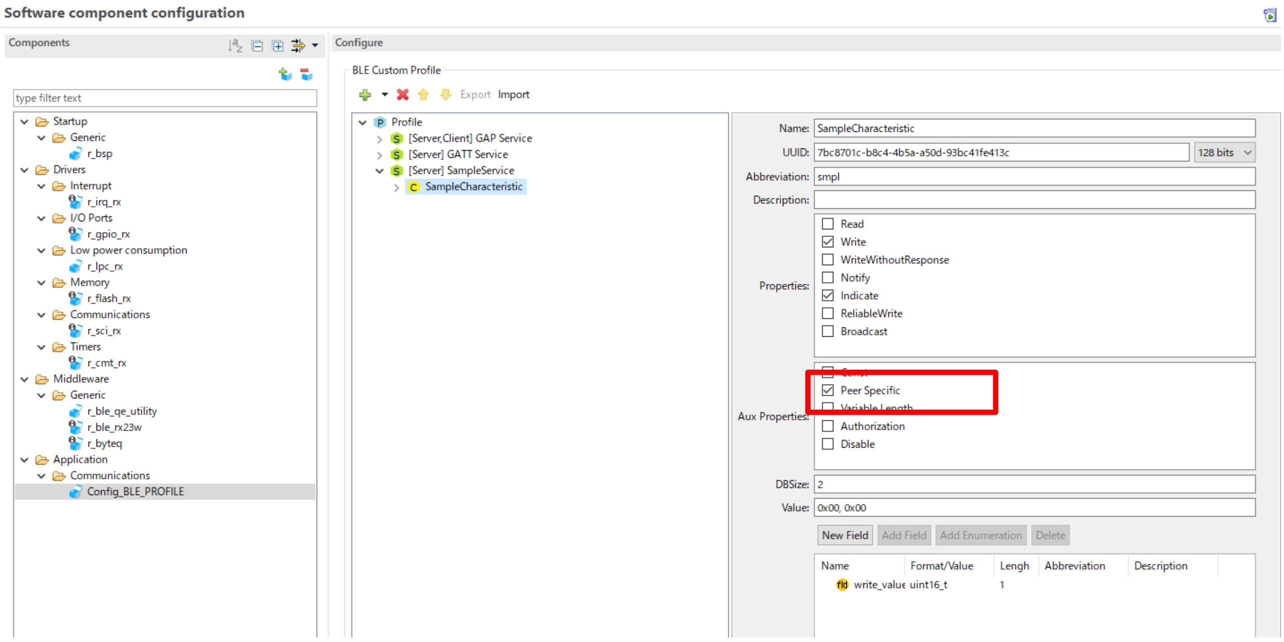

Figure 7.5 Setting to retain the value of characteristic for each device

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.

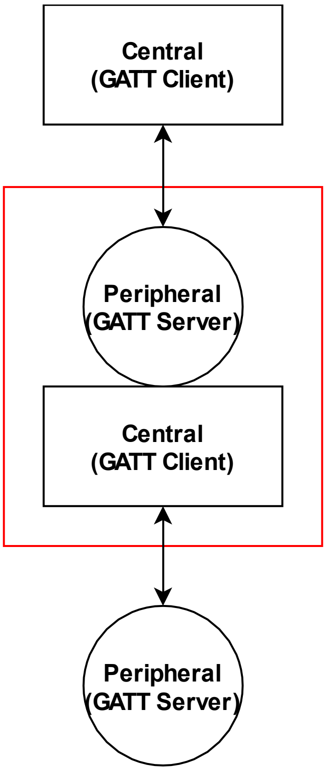

Figure 7.6 Multi roll connection example

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,
                                        .min_ce_length = 0xFFFF,
                                        .max_ce_length = 0xFFFF,
                                };

                                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,
                    .min_ce_length = 0xFFFF,
                    .max_ce_length = 0xFFFF,
                };

                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.

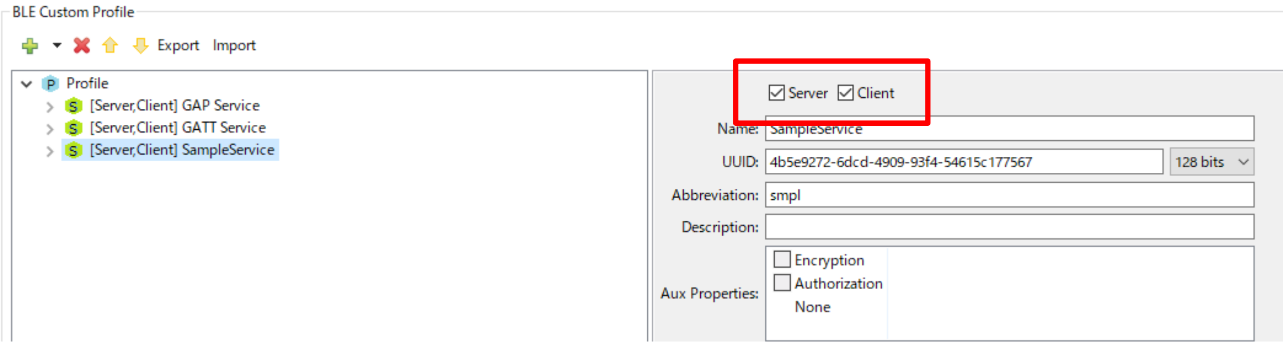

Figure 7.7 Select GATT Role on Service Screen

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.

If the local device disconnects the link by R_BLE_GAP_Disconnect, the reason field in the st_ble_gap_disconn_evt_t structure notified in the BLE_GAP_EVENT_DISCONN_IND event is 0x16 (Connection Terminated by Local Host).

If the remote device disconnects the link, the reason field in the st_ble_gap_disconn_evt_t structure notified in the BLE_GAP_EVENT_DISCONN_IND event is specified as the reason why the remote device disconnects.

8. Communication

In Bluetooth Low Energy, you can adjust the communication speed and power consumption to suit your application by changing the communication parameters. This chapter describes how to set communication parameters using the BLE Protocol Stack. The optional feature may not be supported by the remote device.

Table 8.1 Bluetooth version and supported features and parameters

Communication Parameter	Feature name	Bluetooth Version	Description
PHY	LE 2M PHY LE Coded PHY LE 1M PHY	5.0 (optional) 5.0 (optional) 4.0	Double the symbol rate Forward error correction code added -
Maximums transmit packet length	LE Data Length Extension	4.2 (optional)	Maximum number of transmitted bytes 27 → 251 bytes
Connection parameters	-	4.0	-
MTU	-	4.0	-

The following explains how to use the API to change the communication parameters. Refer to the API document (r_ble_api_spec.chm) included in the "Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)" 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. In the figure, the local device is the central. Local device can change it from either role.

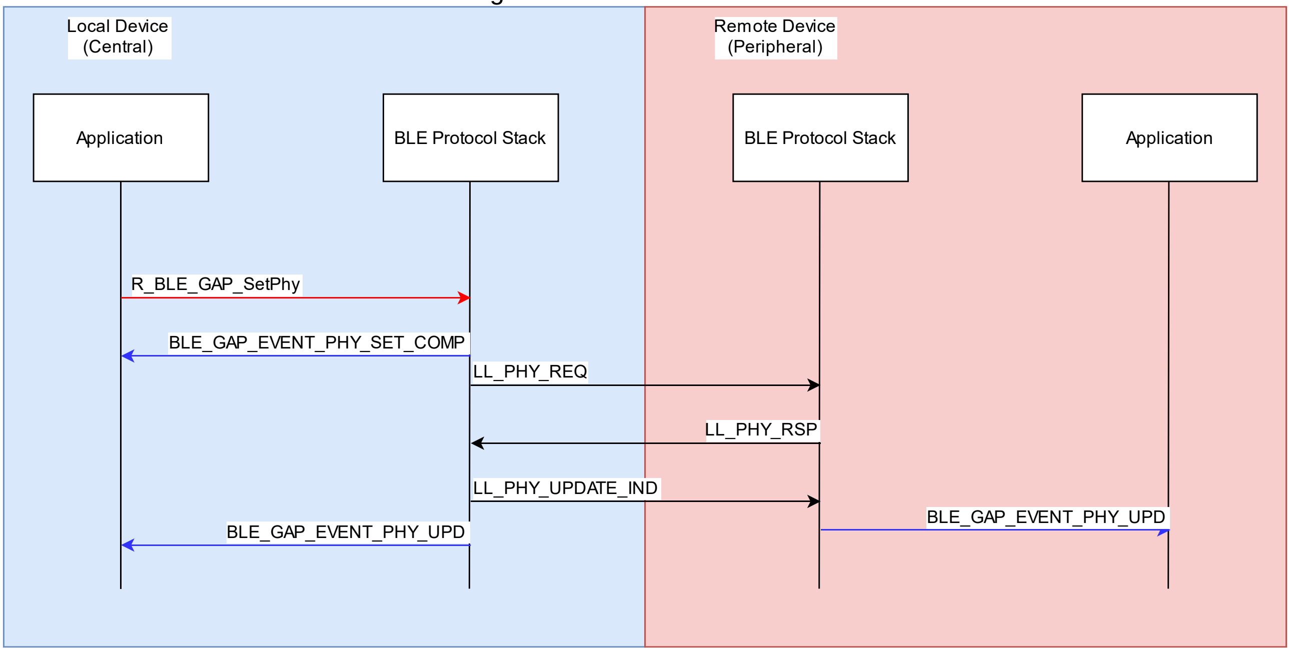

Figure 8.1 Sequence chart when changing PHY

The sample code when changing the PHY to LE Coded PHY (S=8) is shown below. Multiple PHYs can be specified by bit sum.

```
st_ble_gap_set_phy_param_t set_phy = {
     .tx_phys = BLE_GAP_SET_PHYS_HOST_PREF_CD | BLE_GAP_SET_PHYS_HOST_PREF_1M,
     .rx_phys = BLE_GAP_SET_PHYS_HOST_PREF_CD | BLE_GAP_SET_PHYS_HOST_PREF_1M,
     .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 BLE 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 BLE 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 BLE 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.

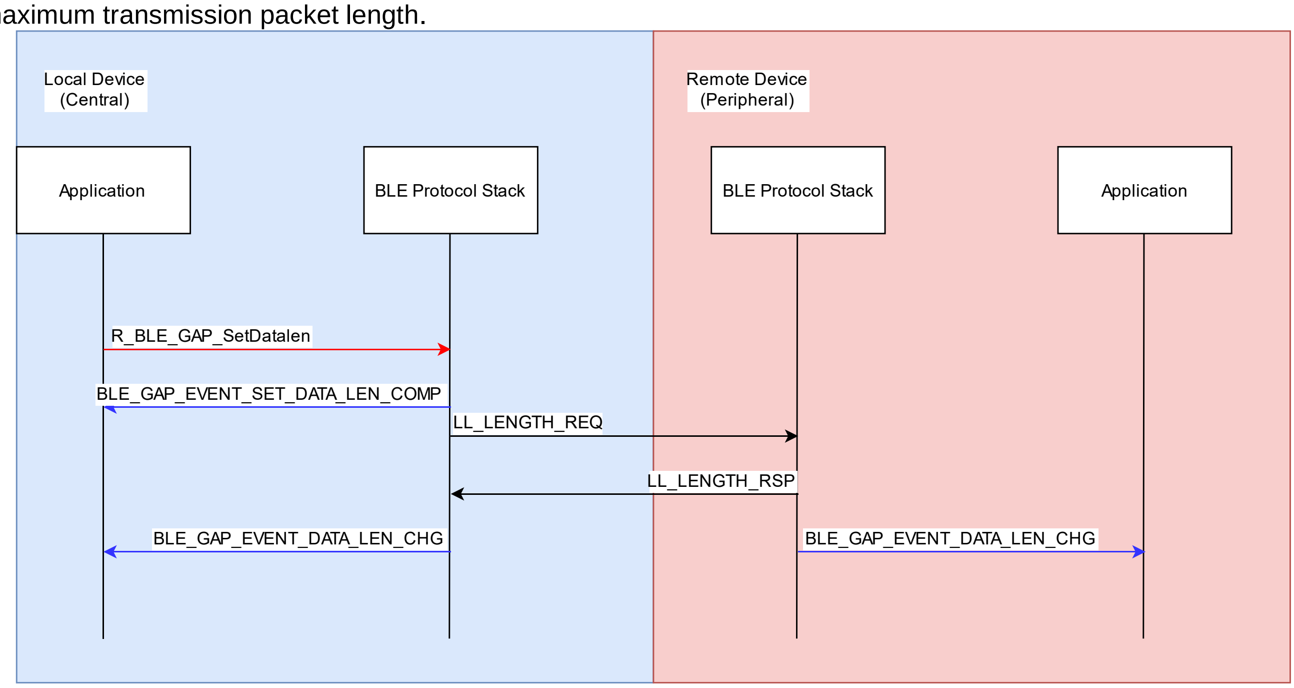

Figure 8.2 Sequence chart when changing the maximum transmission packet length

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.

- Slave Latency

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

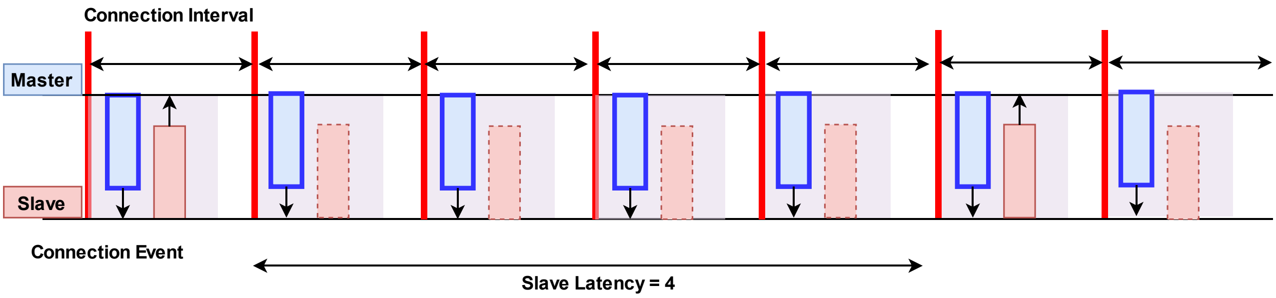

Figure 8.3 Schematic diagram of slave latency and connection event

- 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.

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

- 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.

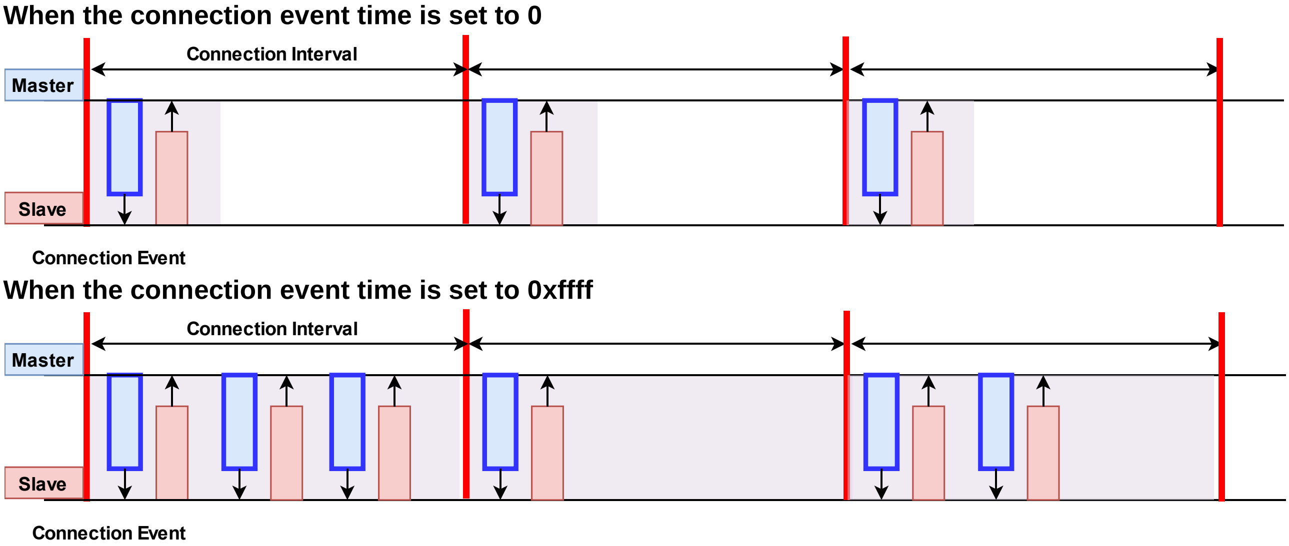

Figure 8.4 Schematic diagram of connection event time and packet exchange

The master determines and changes the connection parameters, but slaves 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 API document (r_ble_api_spec.chm) included in "Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)" for details of PDUs exchanged in Link Layer.

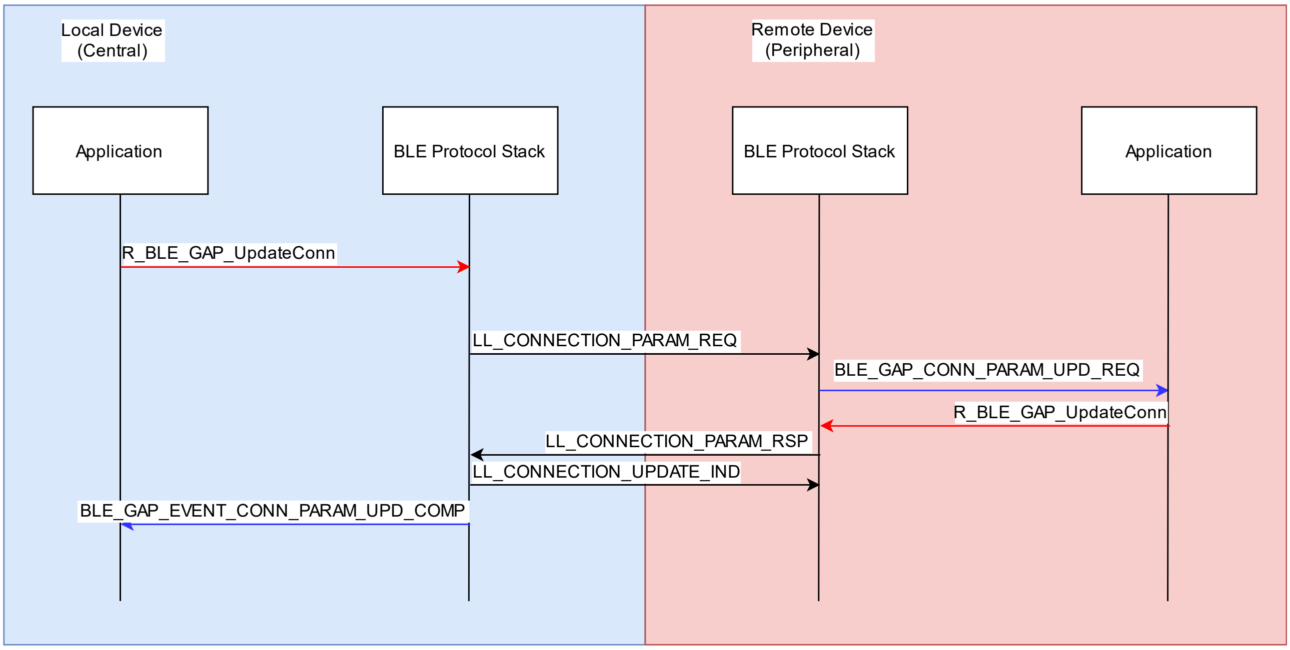

Figure 8.5 Sequence chart when updating connection parameters

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,  //Slave 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,
              .min_ce_length = 0xFFFF,
              .max_ce_length = 0xFFFF,
            };

            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/jp/accessories/Accessory-Design-Guidelines-JP.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 request 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,       /* Slave 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.

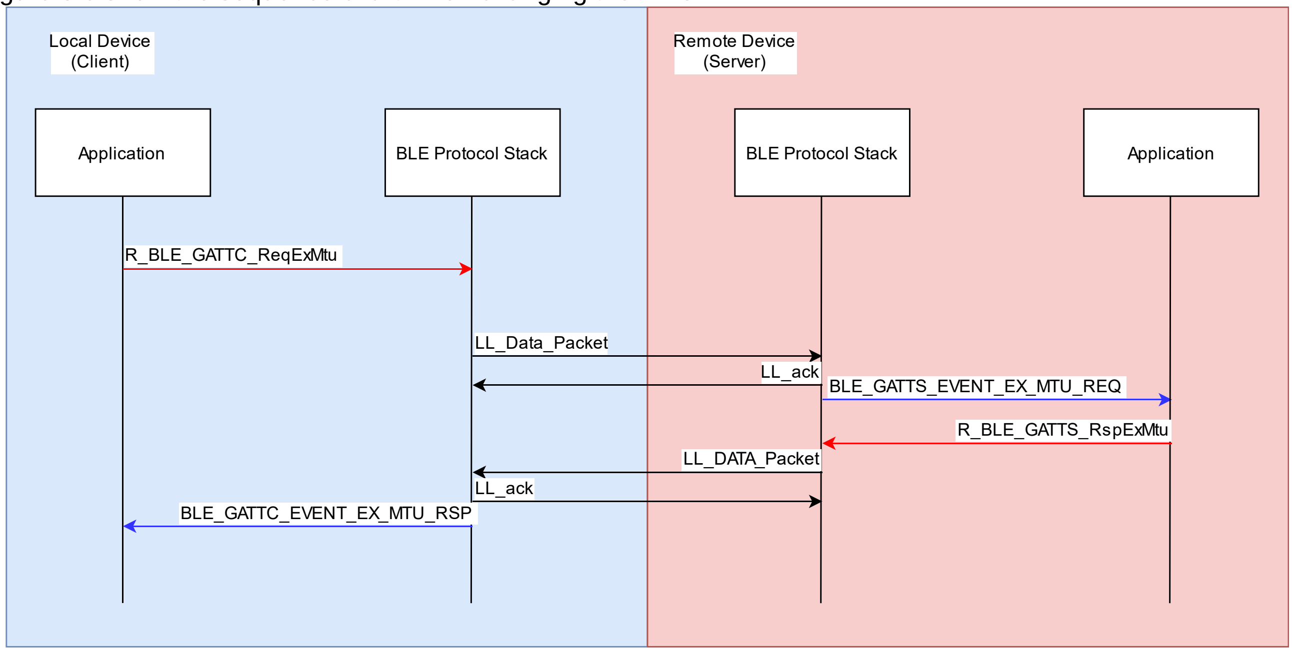

Figure 8.6 Sequence chart when changing MTU

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 BLE Protocol Stack has a flow control function to send large application data in a short time. To realize the flow control function, the BLE 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.

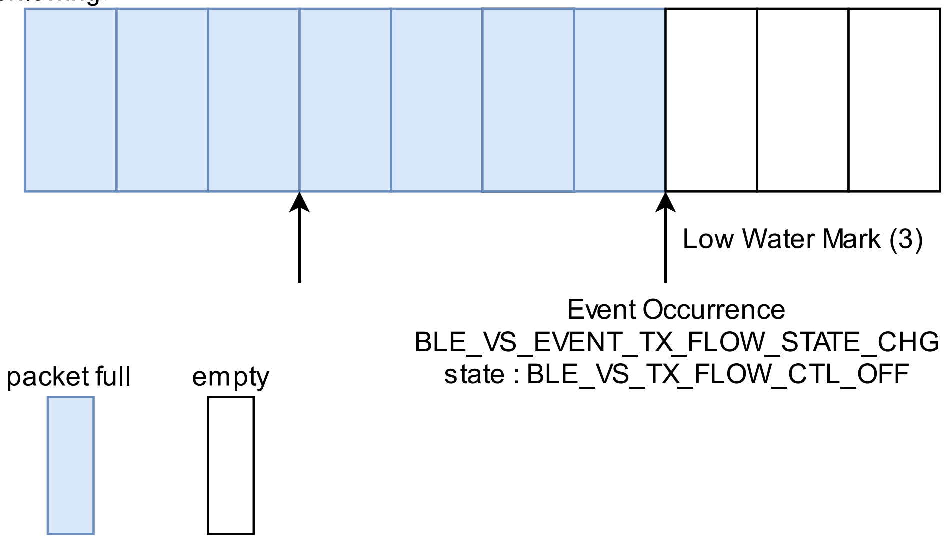

Figure 8.7 Number of empty buffers and events

When the BLE 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.

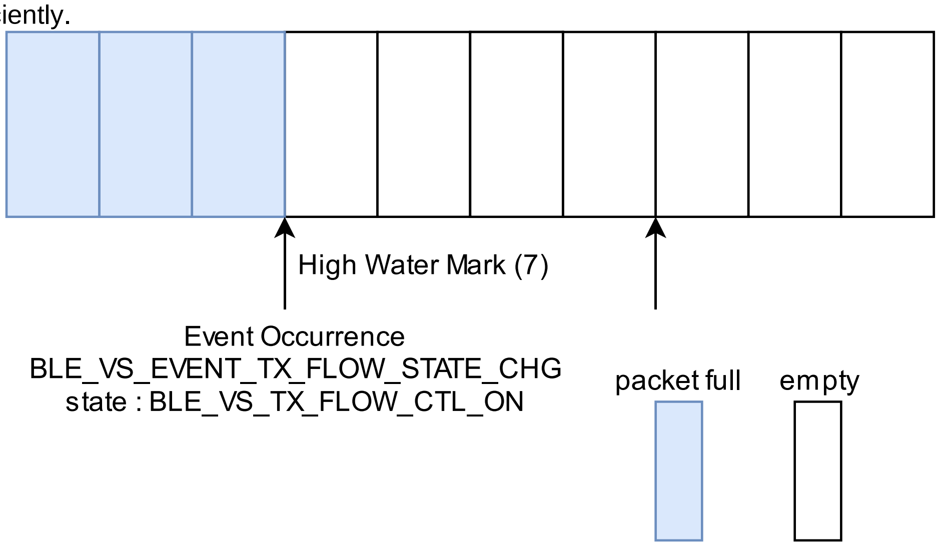

Figure 8.8 Number of empty buffers and events

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 (GAP settings optimization) and to call the send function continuously using the flow control function (Continuous transmission requests).

8.6.1 Continuous transmission requests

Continuous transmission requests uses the Tx flow control feature of the Vendor Specific API provided by the BLE protocol stack. This function executes callback function to transmit when the transmit buffer has space. For the Tx flow control feature of the BLE protocol stack, refer API document (r_ble_api_spec.chm). The application realizes highspeed communication by receiving this callback function and continuously transmitting.

8.6.2 GAP settings optimization

The GAP settings optimization changes the setting of GAP parameter to the optimum value to realize highspeed communication.

Table 8.3 GAP settings for high-speed communication

parameter	value
Connection Interval	50 (msec)
PHY	2M PHY
Max packet length	251 (byte)

8.6.3 Bluetooth Low Energy and Throughput

In this chapter, we explain briefly the relationship between the Bluetooth low energy communication mechanism and throughput. For details on communication standards, please refer to the Bluetooth specifications. Bluetooth Low Energy has three major layers. Controller and Host are connected by Host Controller Interface (HCI). Application and Host are connected by API (R_BLE_API in BLE Module) (Figure 8.9)

In Bluetooth Low Energy, the Link Layer of the controller controls the actual communication path and transmission / reception interval. The operation of this Link Layer is important to achieve highspeed communication. The behavior of the Link Layer is set by the GAP of the Host Layer.

On the other hand, when sending meaningful data for application, the GATT of the host layer is used. In GATT, the profile determines the application data transmission procedure and the data structure to be transmitted and received. The design of this profile is also important for achieving highspeed communication.

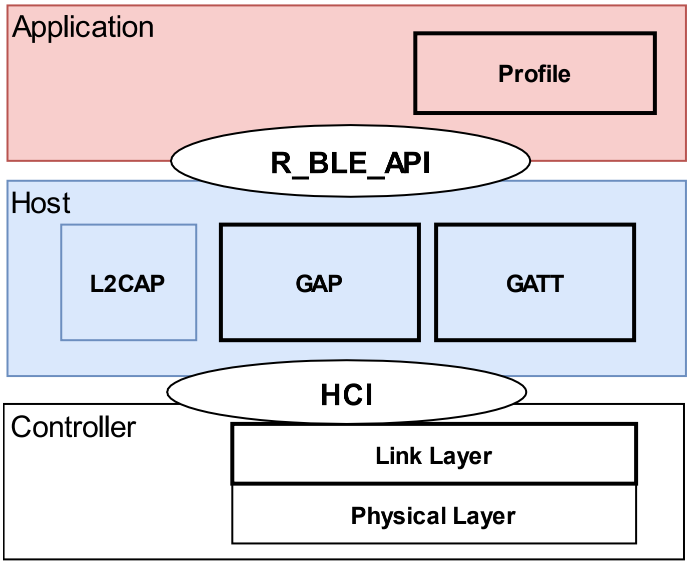

Figure 8.9 Three major layer in Bluetooth Low Energy

8.6.4 Generic Access Profile (GAP)

Generic Access Profile (GAP) defines the procedure for detecting connectable devices and establishing connections. GAP sets the operation of Link Layer and realizes these procedures.

8.6.4.1 Device detection and connection establishment

In Bluetooth Low Energy, a connection is established by one device transmitting (advertising) its own device information and the other device performing device detection (scanning) and connection request (initiating). The device that performs scanning and connection request is the central device, and the device that advertises is the peripheral device. Central determines parameters related to connection maintenance such as frequency map and communication interval (connection interval) after connection is established. The GAP will manage the following information.

- Connection Interval
- PHY
- Maximum Packet Length
- Information for Pairing
- etc.

8.6.4.2 Communication after establishing connection

In Bluetooth Low Energy, after the connection is established, the device exchanges radio frames with a connection event that occurs at each connection interval. In the Link Layer, the central is the master and the peripherals are the slaves. Radio frames are transmitted by the master in time with pre-shared connection events. (Figure 8.10)

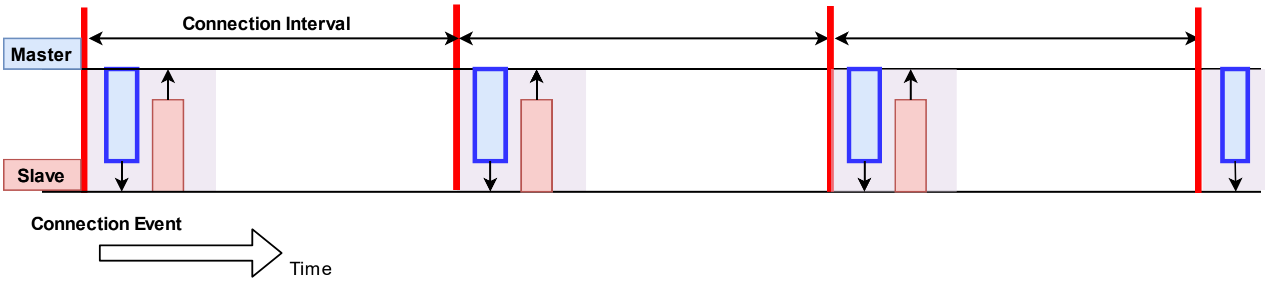

Figure 8.10 Exchange of connection event and radio frame

The connection is maintained by exchanging radio frames at the connection event. If there is additional data to send to either device, the More Data Bit in the radio frame will be set and the connection event will be extended. The connection event ends when the More Data Bit of each other is no longer set or when an error occurs in the received packet. Once the connection event ends, radio frames are not exchanged until the next connection event (Figure 8.11). In order to realize highspeed communication, it is important to communicate using this More Data.

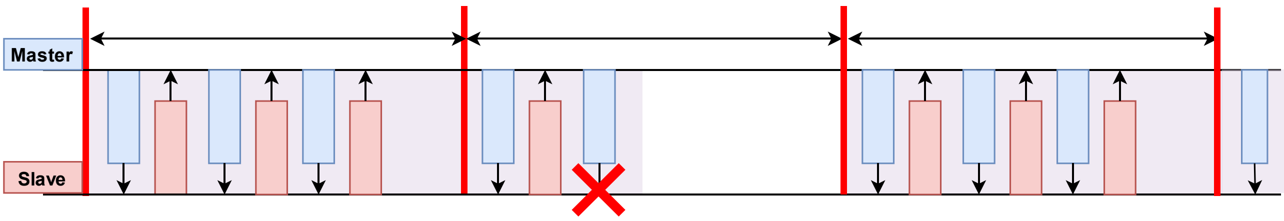

Figure 8.11 Communication by More Data

8.6.4.3 Setting the connection interval

Figure 8.12 shows a schematic diagram of Link Layer operation when the connection interval is changed. Even if the connection interval is changed, if communication by More Data is stable, there will be no significant change in throughput. Note that If the connection interval is shortened extremely, the overhead of waiting time for each interval will hinder throughput.
Figure 8.13 shows the relationship between the connection interval and throughput assuming that the communication environment is good and frame exchange is always successful. The settings of GAP and GATT are PHY: 2M PHY, maximum packet length is 251 bytes, MTU is 247 bytes, and 244 bytes of application data are always notified. If the connection interval is 7.5msec, it will be about 1040kbps.
The throughput per connection interval is calculated from the waiting time $T_{overhead}$ immediately before the connection event, the minimum transfer time T_{frame} for the radio frame to make a round trip, and the application data length (L_{data}). If the packet length is 251 bytes, T_{frame} will be about 1.408 msec.

$$Throughput\ (kbps) = floor\left(\frac{connection\ interval - T_{overhead}}{T_{frame}}\right) * 8 * L_{\mathrm{data}} * \frac{1}{connection\ interval}$$

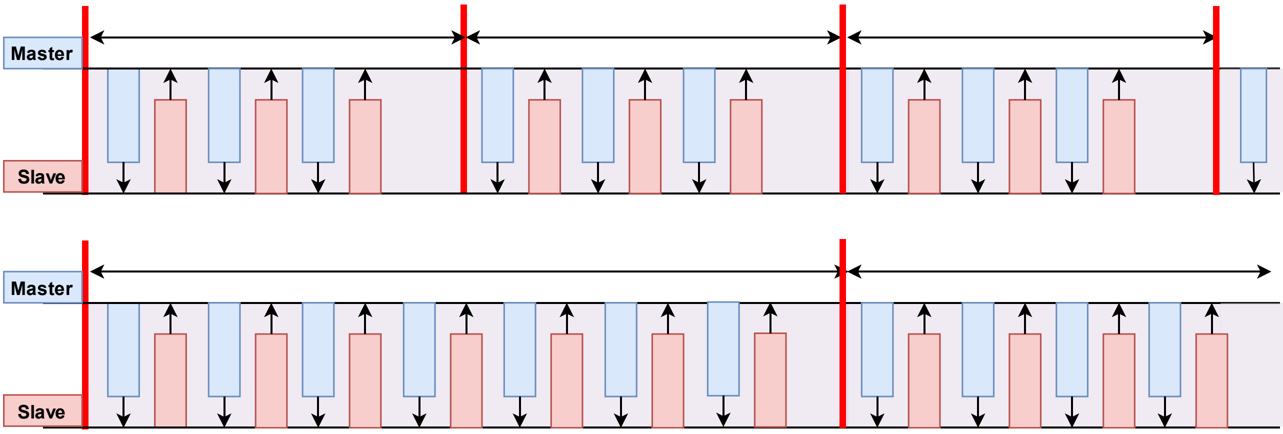

Figure 8.12 Change in connection interval and number of radio frames

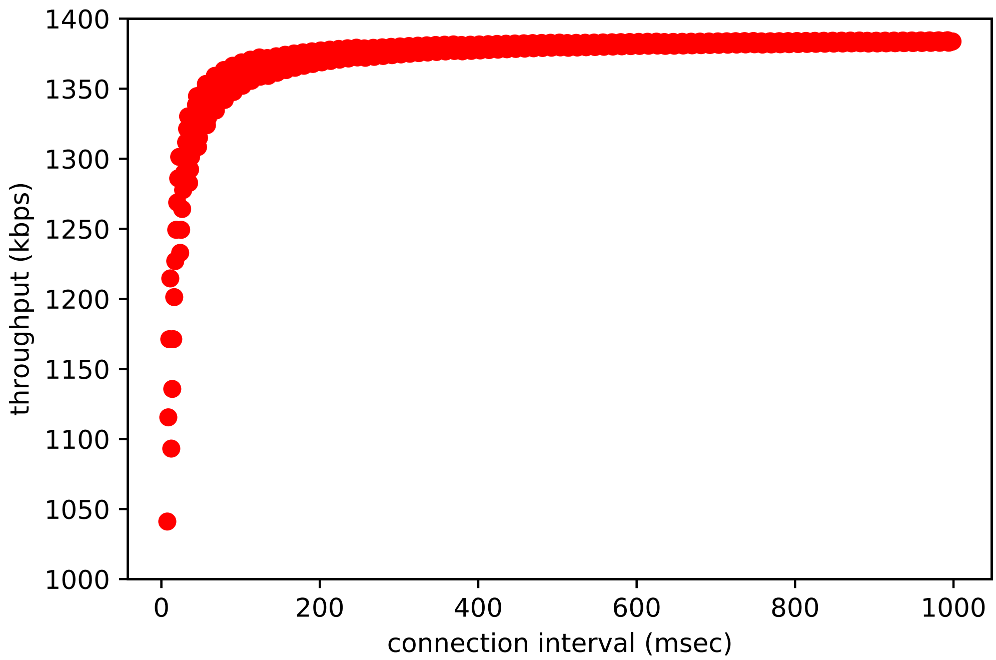

Figure 8.13 The relationship connection interval and throughput

If the communication environment is good, the throughput will not be affected even if the connection interval becomes long, but if the communication by using the More Data bit is interrupted due to a communication error, the difference in the connection interval will have a large effect (Figure 8.14). When a communication error occurs, each device waits until the next connection event, so if the connection interval increases, the throughput decreases.

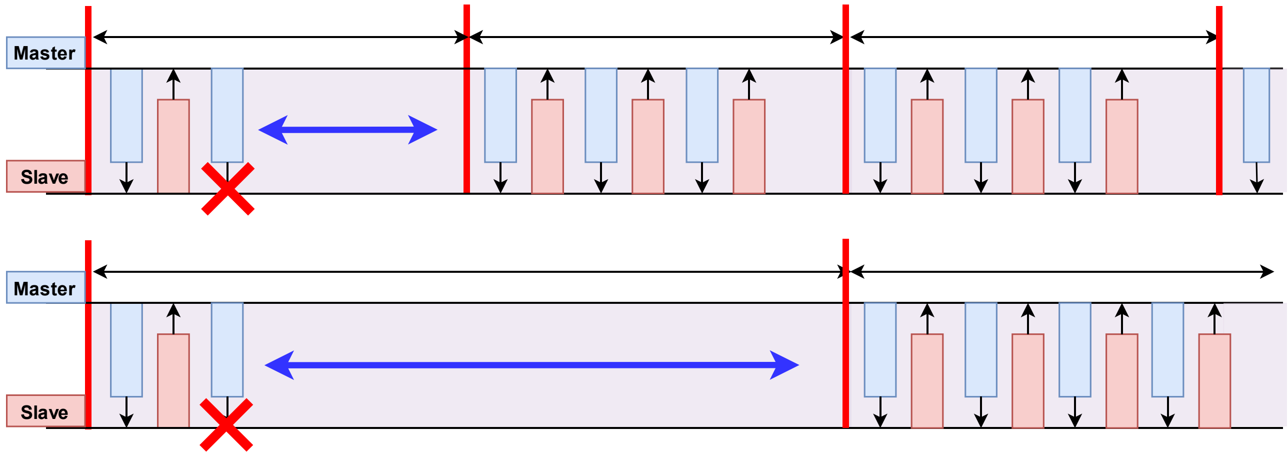

Figure 8.14 Connection interval and throughput when communication error occurs

Figure 8.15 shows the relationship between connection interval, probability of frame exchange failure, and throughput. GAP settings are PHY: 2M PHY, maximum packet length 251 bytes, MTU 247 bytes, and the value when 244 bytes of application data are always notified. The expected value of throughput per connection interval is plotted.

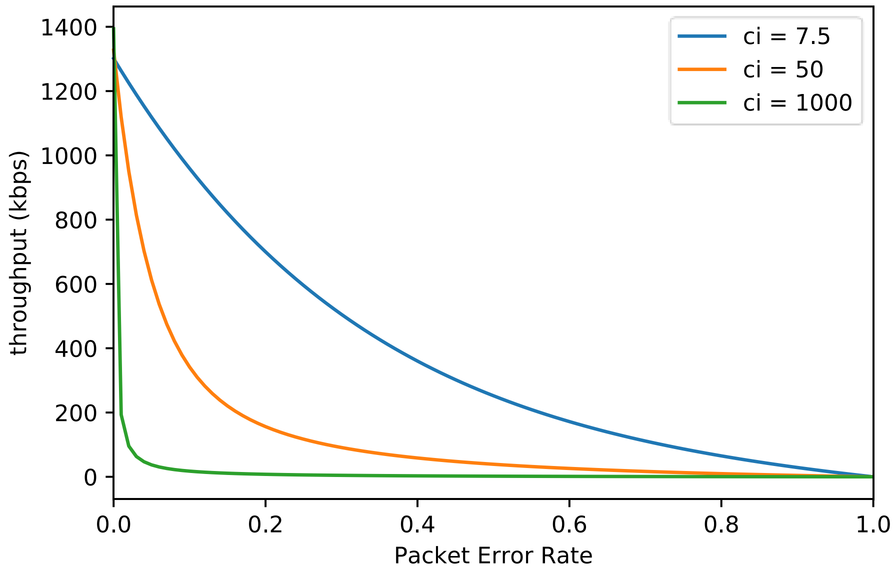

Figure 8.15 Relationship between frame exchange failure probability and throughput

To change the connection interval, execute the gap conn_cfg update command. To change using R_BLE_API, use R_BLE_GAP_UpdConn. For details about the API, refer to "API document (r_ble_api_spec.chm)".

8.6.4.4 Setting the PHY

Figure 8.16 shows a schematic diagram of Link Layer operation when the physical layer (PHY) settings are changed. When the physical layer PHY is changed, the air frame occupation time changes. If the data length is the same, the air occupation time will be about half in 1M PHY, as it is in 2M PHY. If the occupied time of one frame in the air is short, the number of packets transmitted / received per unit time increases, and the throughput improves.

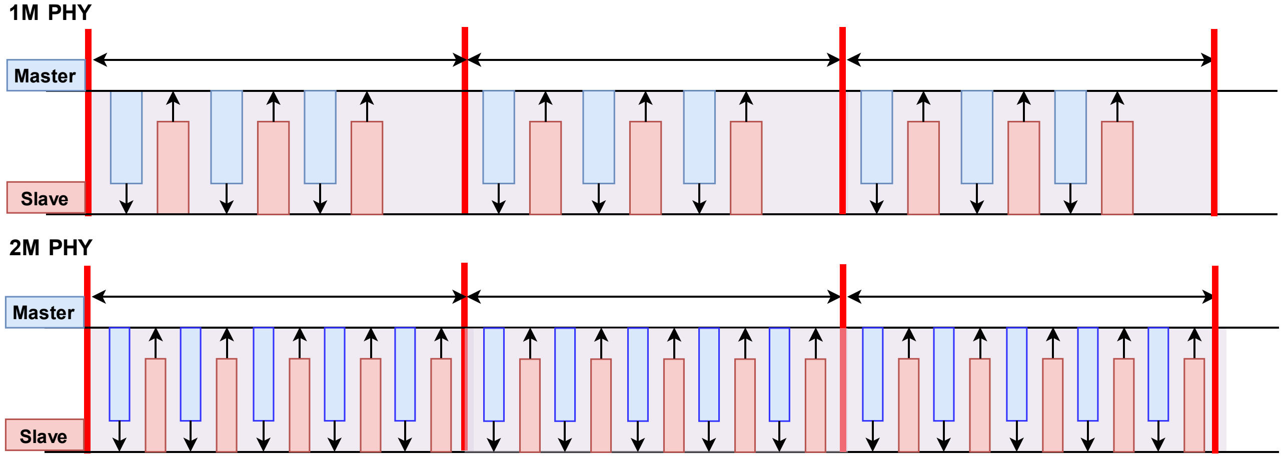

Figure 8.16 Schematic diagram when using 2M PHY

To change the PHY, execute the gap conn_cfg phy command. To change using R_BLE_API, use R_BLE_GAP_SetPhy. For details about the API, refer to "API document (r_ble_api_spec.chm)".

8.6.4.5 Setting the Maximum packet length

Figure 8.17 shows a schematic diagram of Link Layer operation when the maximum packet length is set to a high value and data with a large packet length is transmitted. The application information can be efficiently transmitted by minimizing the header information of the radio frame and the transmission / reception interval.

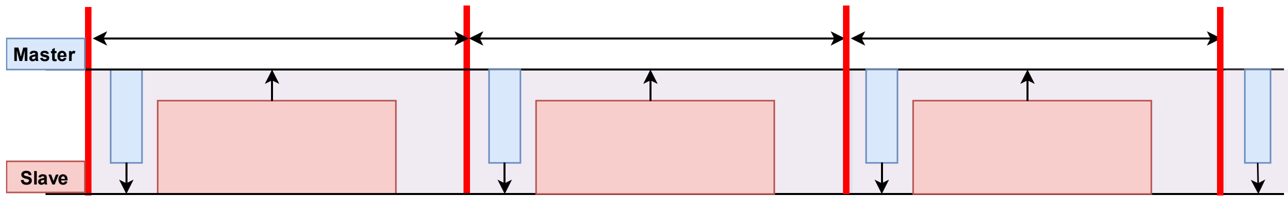

Figure 8.17 Schematic diagram of Link Layer when changing packet length

To change the Maximum packet length, execute the gap conn_cfg data_len command. To change from R_BLE_API, use R_BLE_GAP_SetDatalen. For details about the API, refer to "API document (r_ble_api_spec.chm)".

8.6.4.6 Setting the encryption of communication

Figure 8.18 shows a schematic diagram of Link Layer operation when communication is encrypted. Through encryption, the data for checking packet integrity (4 bytes) is carried in the radio frame, which may reduce the throughput.

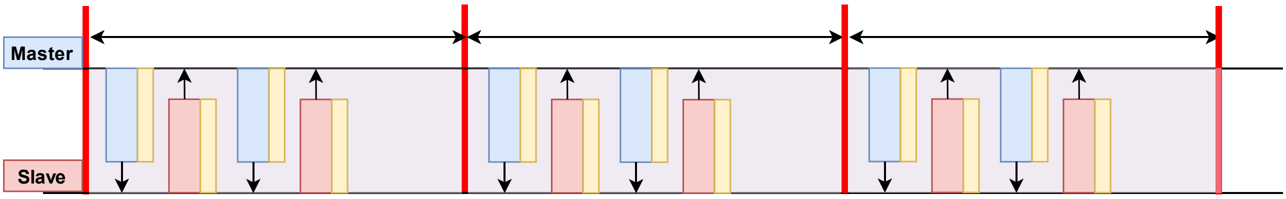

Figure 8.18 Schematic diagram of Link Layer in encrypted communication

To encrypt communication, execute the gap auth start command. When performing encryption from R_BLE_API, use R_BLE_GAP_StartEnc or R_BLE_GAP_StartPairing. You can also use R_BLE_ABS_StartAuth of Abstraction API of app_lib. For details on these APIs, refer to the “API document (r_ble_api_spec.chm)”.

8.6.5 Generic Attribute Profile (GATT)

In Bluetooth Low Energy, the information for detecting and connecting (advertising and scanning, initiating) a communication device and for continuous communication after the connection is made is managed by the GAP.
On one hand, the communication procedure of application data (sensor data etc.) is determined by Generic Attribute Profile (GATT). GATT implements a client-server architecture over the communication path established by GAP. The client reads / writes data from / to the GATT database held by the server using a predetermined procedure. At this time, the server returns a response to the client. On the other hand, it is also possible for the server to notify to the client (Figure 8.19). All applications that perform Bluetooth low energy data communication perform data communication according to GATT.

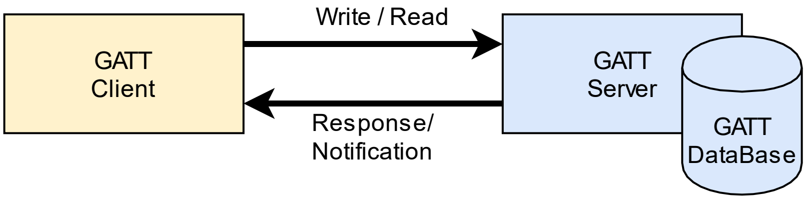

Figure 8.19 GATT architecture

In GATT, communication is focused on the feature of the application. The feature of an application is called a "service", and the data required for that feature is called a "characteristic." A "profile" is a set of features (services) required to realize an application and defines the communication specifications of the application. When performing GATT communication, it is necessary to share in advance information about the feature (service) as an application and the data (characteristics) necessary to realize that feature. The server store information about the service it has and the characteristic that the service has in a database.
Figure 3.12 shows the relationship between profile, service, and characteristic when using a thermometer as an example. The features of the thermometer application are temperature measurement and device information. Features and data are kept on the GATT database as services and characteristics.

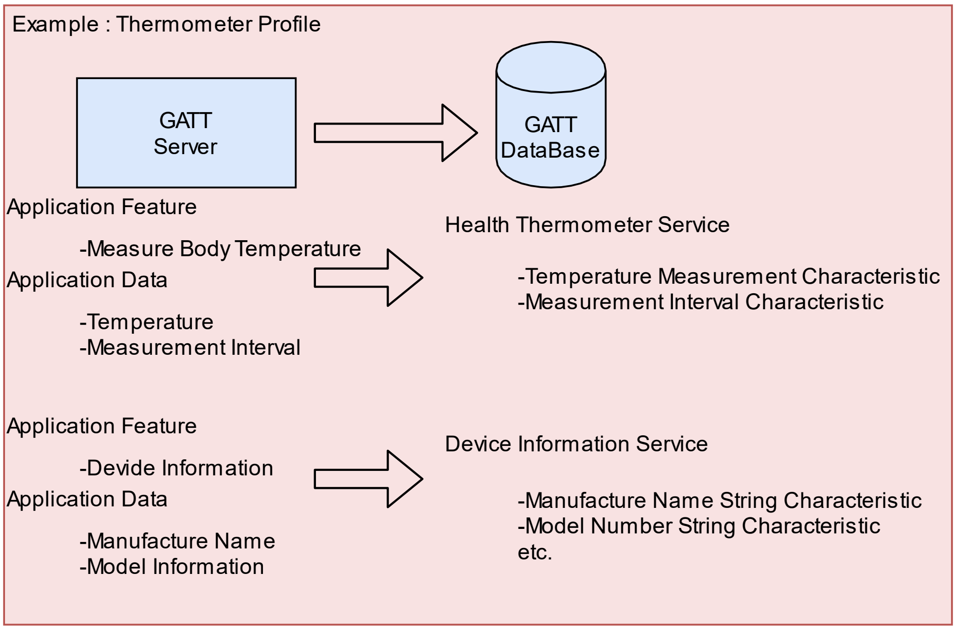

Figure 8.20 Application Data and GATT database

Simply by establishing a connection, the client does not have service information for the server. The client queries the server for a particular service using a procedure called service discovery (Figure 8.21). This procedure gives the client information about the services the client wants on the server and handle information about the data in the database. The client uses this to handle information to read and write to the database.

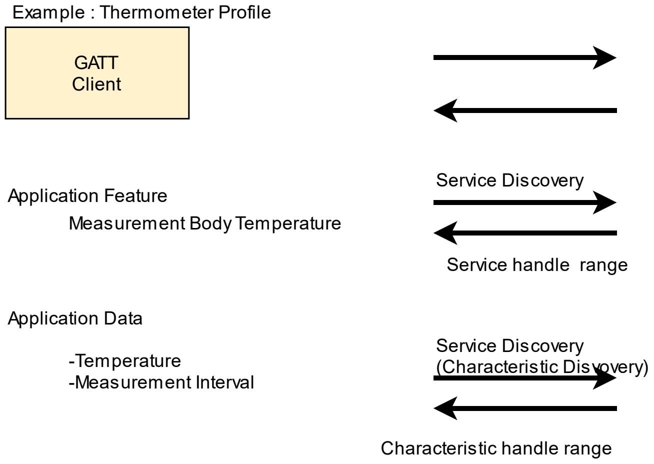

Figure 8.21 Service discovery operation

In the characteristic, the data and its structure are decided, but in addition, the procedure of exchanging data between the client and the server is also decided. If there are additional options for the data and communication procedure determined by the characteristic, they are described in the characteristic descriptor.
Transmission and reception of characteristic data is performed according to the procedure determined by the characteristic. Table 8.4 summarizes the typical procedures. These procedures are characterized by the direction of data transmission and whether or not to wait for a response from the other. A procedure that requires a response cannot perform the same procedure before receiving a response from the other. Figure 8.22 shows a schematic diagram of the Read operation in which the client reads the server data. Data communication by GATT is performed based on the handle information.

Table 8.4 Typical communication procedure of GATT communication

Procedure name	operation	Direction to transmit	Response require
Read	Read	From client to server	Yes
Write	Write	From client to server	Yes
Write Without Response	Write	From client to server	No
Indication	Notify	From server to client	Yes
Notification	Notify	From server to client	No

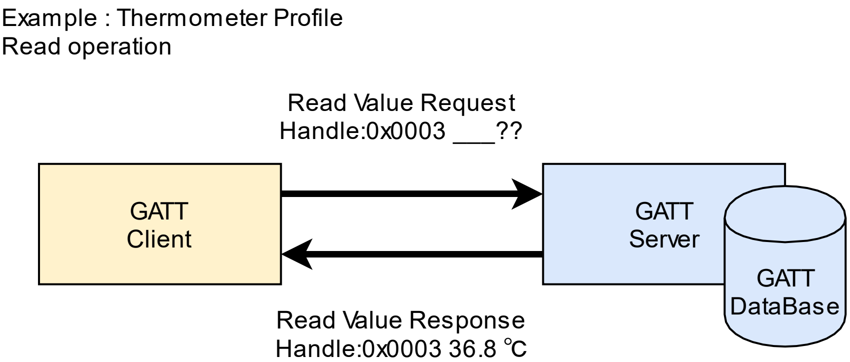

Figure 8.22 Read operation

8.6.5.1 No response operation (Notification / Write Without Response)

With Notification or Write Without Response operation, the next packet can be transmitted without waiting for the response from the opposition. Therefore, More Data communication can be performed by continuously sending transmission requests. Figure 8.23 shows a schematic diagram of the Notification operation in the Link Layer.

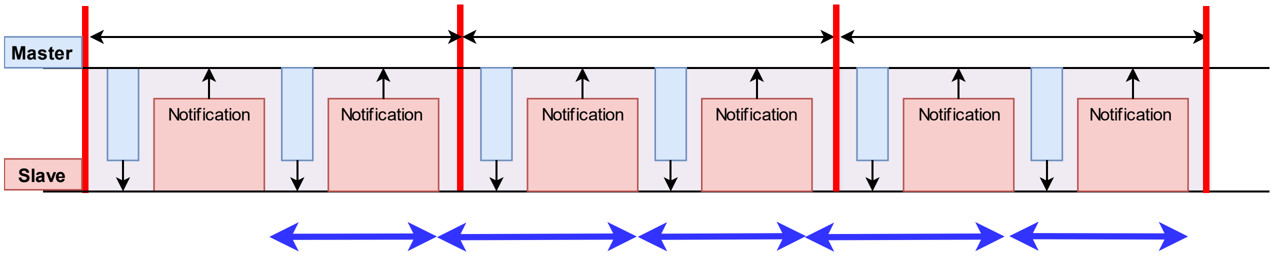

Figure 8.23 Schematic diagram of Link Layer during Notification operation

In the figure above, the slave acts as a server and performs a Notification. Slave and master are roles in the Link Layer and have no relationship with GATT role.

8.6.5.2 Response operation (Indication / Write)

In Indication and Write operations, after transmitting it is necessary to wait for the response from the other device before transmitting the next data. Therefore, a request to transmit the next data cannot be sent in one connection event, and more data communication cannot be performed. Figure 8.24 shows a schematic diagram of Link Layer operation for Indication. It takes twice as long as the connection interval to transmit one data packet.

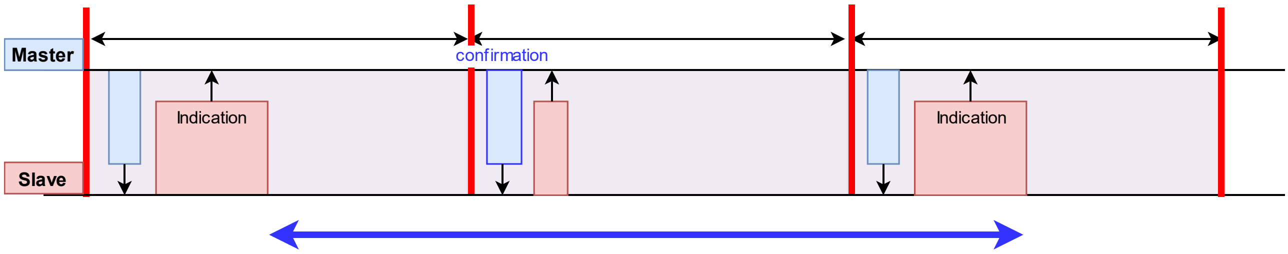

Figure 8.24 Schematic diagram of Link Layer during Indication operation

In the figure above, the slave acts as a server and performs Indication. Slave and master are roles in the Link Layer and have no relationship with GATT role.

8.6.6 Data type

Use the st_ble_seq_data_t structure defined by Profile Common Library. This structure has members that mean the start address of array data and length of array data. The encode / decode functions for this structure are implemented in the Profile Common Library. Therefore, you can transmit and receive array data without implementing the encode / decode functions.

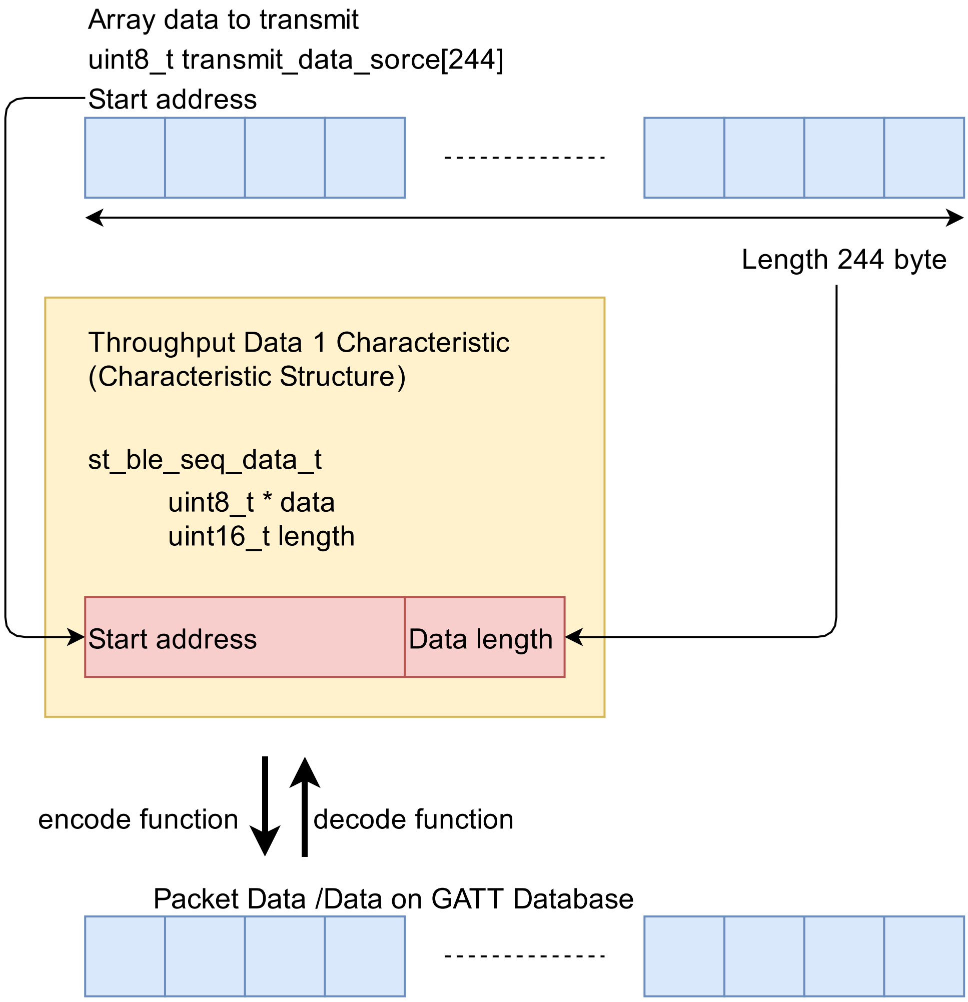

Figure 8.25 Transmitting array data using st_ble_seq_data_t structure

9. Security

This section describes the security functions provided by the Bluetooth Low Energy.

9.1 Pairing

Pairing has to be done to use the Bluetooth security function. In the case such as the following, pairing is necessary.

- The remote device sets security requirement for the access to the GATT service.
- The local device resolves the remote device address.

Pairing exchanges the keys with a remote device. The keys to be exchanged are followings.

- LTK (Long Term Key)
  Encryption uses LTK.

- IRK (Identity Resolving Key)
  Privacy function uses IRK.

- CSRK (Connection Signature Resolving Key)
  Signed data transmission 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 BLE Protocol Stack performs LE Secure Connections. If a remote device does not support LE Secure Connections, the BLE Protocol Stack performs LE Legacy Pairing.
The pairing procedure in an application shows Figure 9.1. The following sections describe the details of pairing steps.

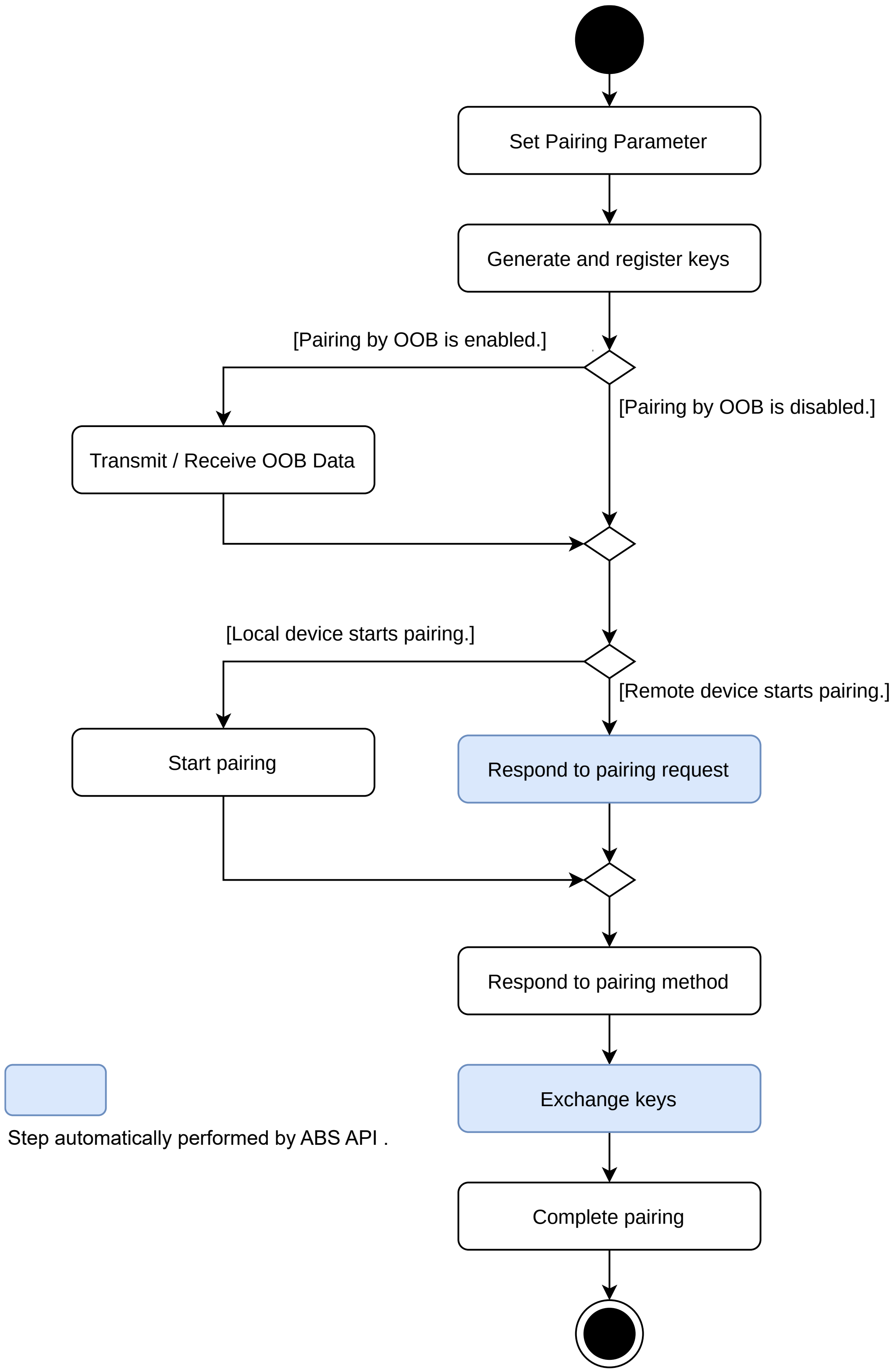

Figure 9.1 Pairing procedure in application

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_DISPLAY_YESNO(0x01) BLE_GAP_IOCAP_KEYBOARD_ONLY(0x02) BLE_GAP_IOCAP_NOINPUT_NOOUTPUT(0x03) BLE_GAP_IOCAP_KEYBOARD_DISPLAY(0x04)	BLE_GAP_IOCAP_NOINPUT_NOOUTPUT(0x03)
2. MITM Protection Request		mitm	mitm	BLE_GAP_SEC_MITM_BEST_EFFORT(0x00) BLE_GAP_SEC_MITM_STRICT(0x01)	BLE_GAP_SEC_MITM_BEST_EFFORT(0x00)
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	7～16	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) BLE_GAP_KEY_DIST_SIGNKEY(0x04)	0
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_SPRT(0x01)	BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT(0x00)
7. LE Secure Connections Request		sec_conn_only	sec_conn_only	BLE_GAP_SC_BEST_EFFORT(0x00) BLE_GAP_SC_STRICT(0x01)	BLE_GAP_SC_BEST_EFFORT(0x00)

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)
Yes	BLE_GAP_SEC_MITM_STRICT(0x01)

Completing pairing with the pairing method except Just Works according to “9.1.6 Pairing method” enables the MITM protection.

3. Bonding

Table 9.6 shows the bonding parameter settings which indicate whether or not the device does bonding. 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 recommends that the encryption key size is 16 bytes because the short encryption key size causes to reject access to the remote device.

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	BLE_GAP_KEY_DIST_ENCKEY(0x01)
IRK	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 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)

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	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.

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 vscb(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 communicate with the signed data, it does not need to generate and register the local device CSRK. It does not need to generate and register the local device LTK on the application before start pairing.

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 by 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.

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 Central and 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 with the request event by R_BLE_GAP_ReplyPairing.

An example of responding 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 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 in 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 handle the pairing, because the BLE 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 handle the 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

<table>
<tr><th rowspan="2">Peripheral</th><th colspan="5">Central</th></tr>
<tr><th>DisplayOnly</th><th>DisplayYesNo</th><th>KeyboardOnly</th><th>NoInputNoOutput</th><th>KeyboardDisplay</th></tr>
<tr><td>DisplayOnly</td><td>Just Works</td><td>Just Works</td><td>Passkey Entry</td><td>Just Works</td><td>Passkey Entry</td></tr>
<tr><td rowspan="2">DisplayYesNo</td><td rowspan="2">Just Works</td><td>Just Works
(LE legacy pairing)</td><td rowspan="2">Passkey Entry</td><td rowspan="2">Just Works</td><td>Passkey Entry
(LE legacy pairing)</td></tr>
<tr><td>Numeric Comparison
(LE Secure Connections)</td><td>Numeric Comparison
(LE Secure Connections)</td></tr>
<tr><td>KeyboardOnly</td><td>Passkey Entry</td><td>Passkey Entry</td><td>Passkey Entry</td><td>Just Works</td><td>Passkey Entry</td></tr>
<tr><td>NoInputNoOutput</td><td>Just Works</td><td>Just Works</td><td>Just Works</td><td>Just Works</td><td>Just Works</td></tr>
<tr><td rowspan="2">KeyboardDisplay</td><td rowspan="2">Passkey Entry</td><td>Passkey Entry
(LE legacy pairing)</td><td rowspan="2">Passkey Entry</td><td rowspan="2">Just Works</td><td>Passkey Entry
(LE legacy pairing)</td></tr>
<tr><td>Numeric Comparison
(LE Secure Connections)</td><td>Numeric Comparison
(LE Secure Connections)</td></tr>
</table>

The pairing events and the API used for the response differ 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.

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 local 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 pairing has been completed, the BLE_GAP_EVENT_PAIRING_COMP event is notified. If the pairing is successful, the event result is BLE_SUCCESS(0x00). Any other value indicates a pairing failure.

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 BLE Protocol Stack.

Note: RE01B does not have a data flash area, so writing bonding information to the non-volatile area is currently not supported. Bonding information is store only in RAM and is erased when the power is turned off.

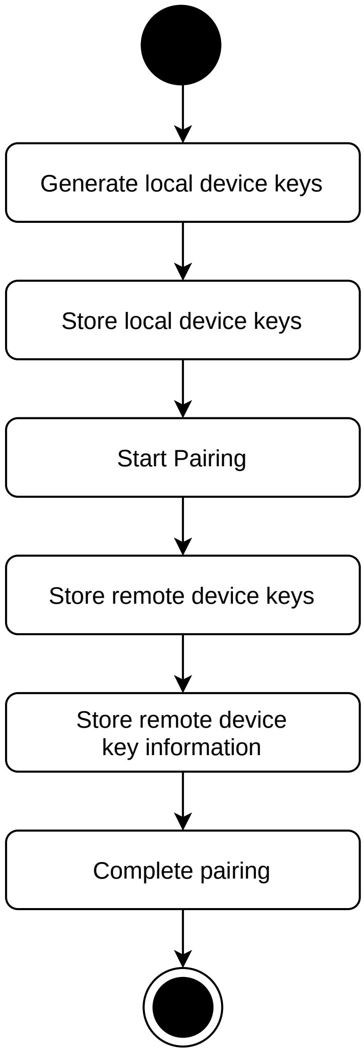

Figure 9.2 Boding procedure

9.2.1 Store local device keys

If the local device uses the privacy feature, the IRK and the identity address 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.

9.2.2 Store remote device keys

Store remote device keys and key information received by the following events.

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. */
        }
    }
    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. */
    }
    break;
```

Code 9-5 Sample of storing received keys

9.3 Encryption

Bluetooth LE enables secure communication by encrypting data packets.
The encryption in reconnection after pairing uses the key exchanged by pairing.

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.
R_BLE_ABS_StartAuth
R_BLE_GAP_StartEnc

Depending on the remote device implementation, the remote device does not respond 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(master)

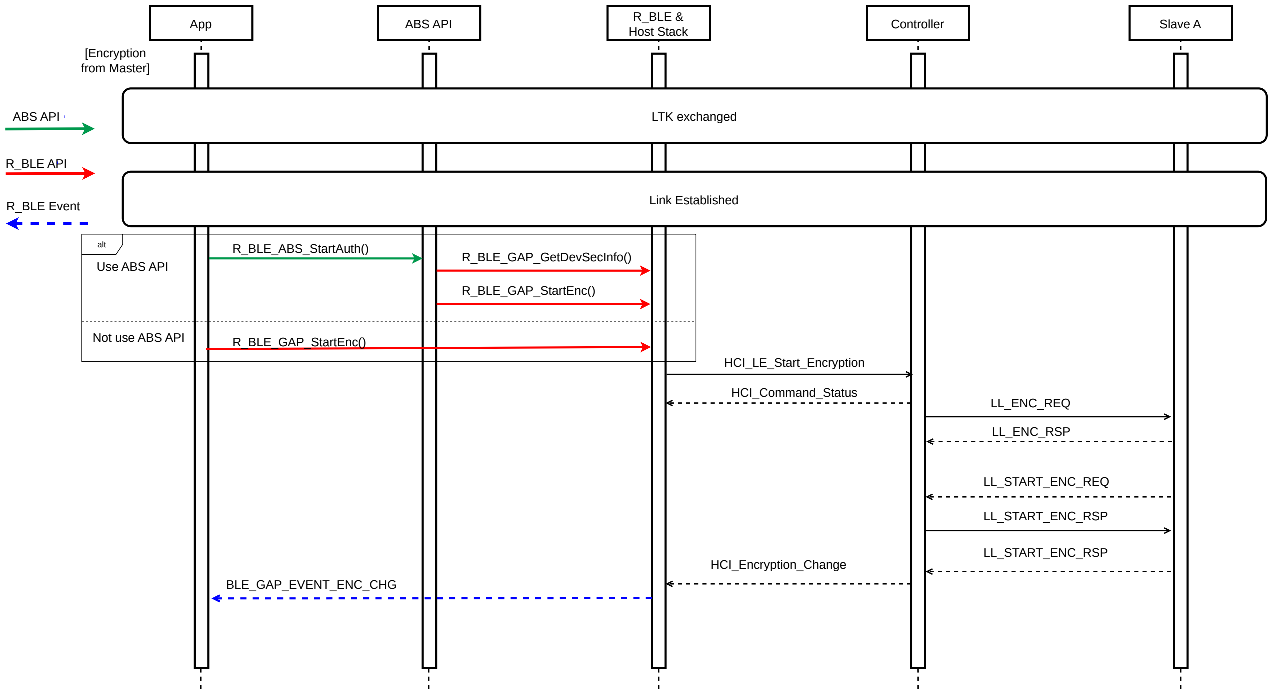

Figure 9.3 Sequence of encryption request from local device(master)

(2) Encryption request from local device(slave)

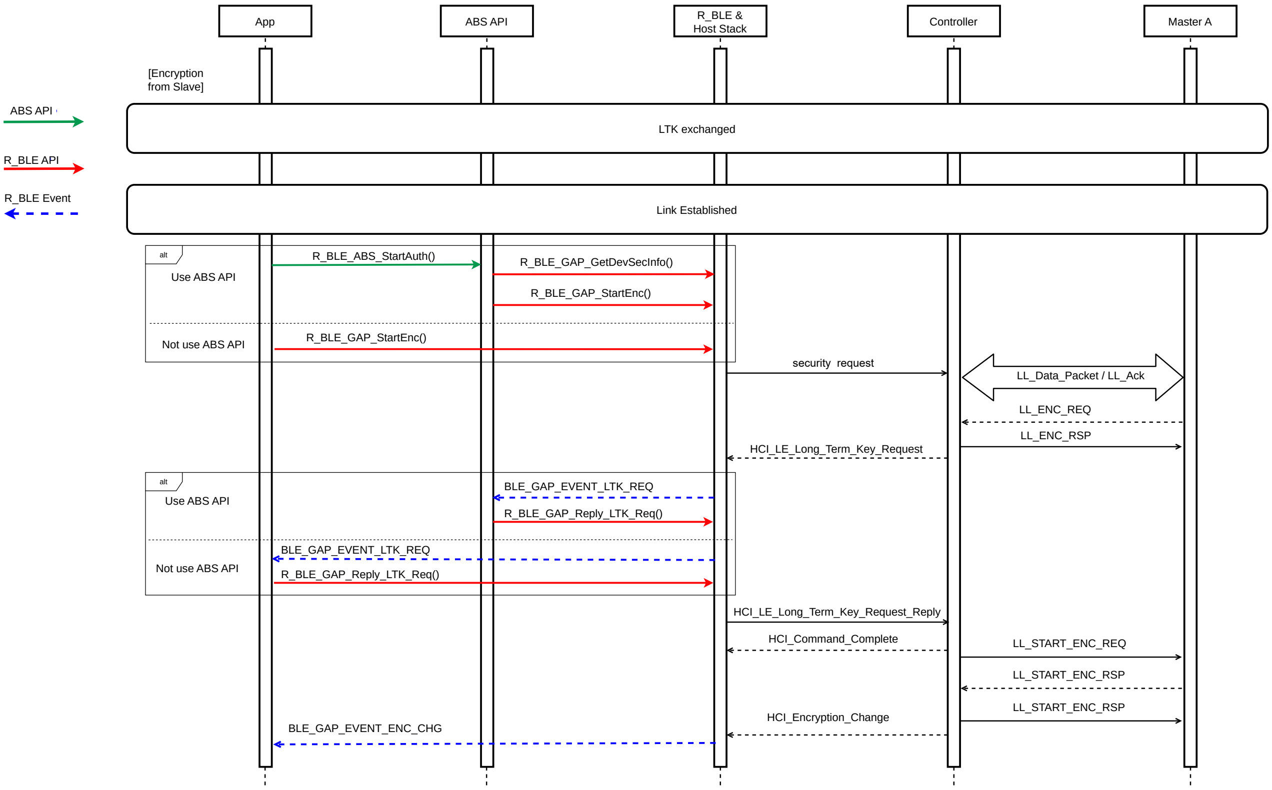

Figure 9.4 Sequence of encryption request from local device(slave)

9.3.2 Respond to an encryption request

When receiving an encryption request from a remote device, BLE_GAP_EVENT_LTK_REQ event is notified. Call R_BLE_GAP_ReplyLtkReq with the parameter notified the event to respond to the encryption request. If the encryption is complete successfully, BLE_GAP_EVENT_LTK_RSP_COMP event is notified. If it fails, remove the remote device LTK and do pairing again.

An example of an encryption request event and respond 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 in the event

If using Abstraction API, it automatically responds with a remote device.

The local device requires to respond to the encryption request when it reconnects to a paired smart phone. The sequence of response to an encryption request is shown below.

(1) Response to an encryption request from remote device(master)

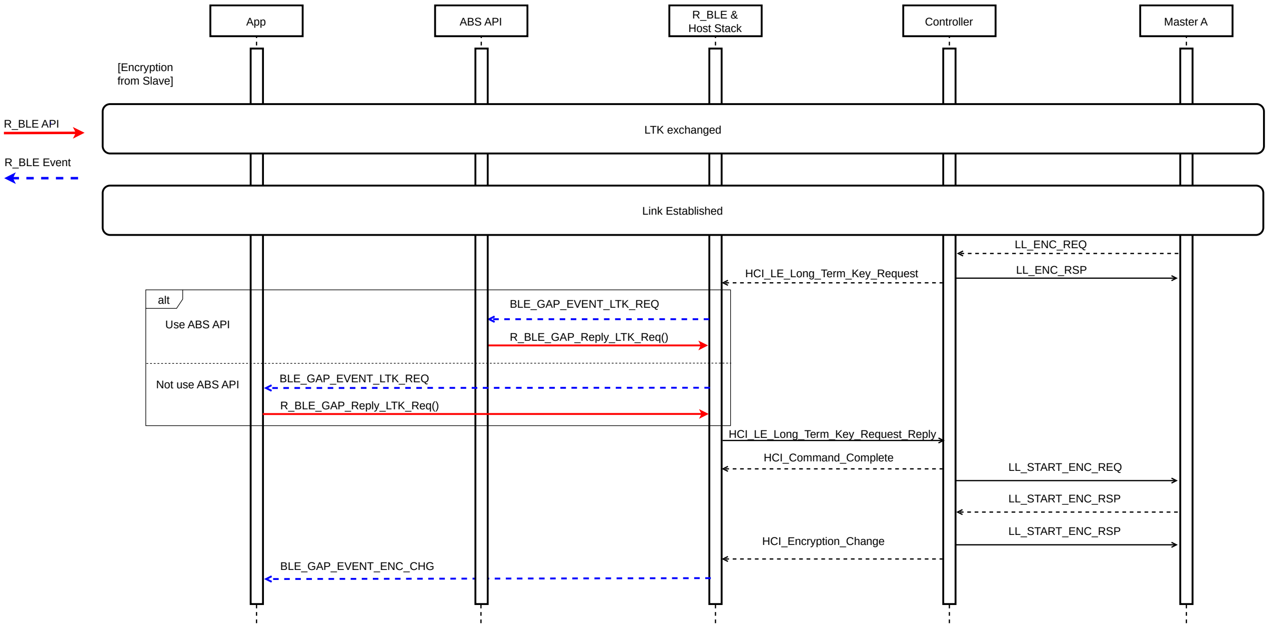

Figure 9.5 Sequence of response to an encryption request from remote device(master)

(2) Response to an encryption request from remote device(slave)

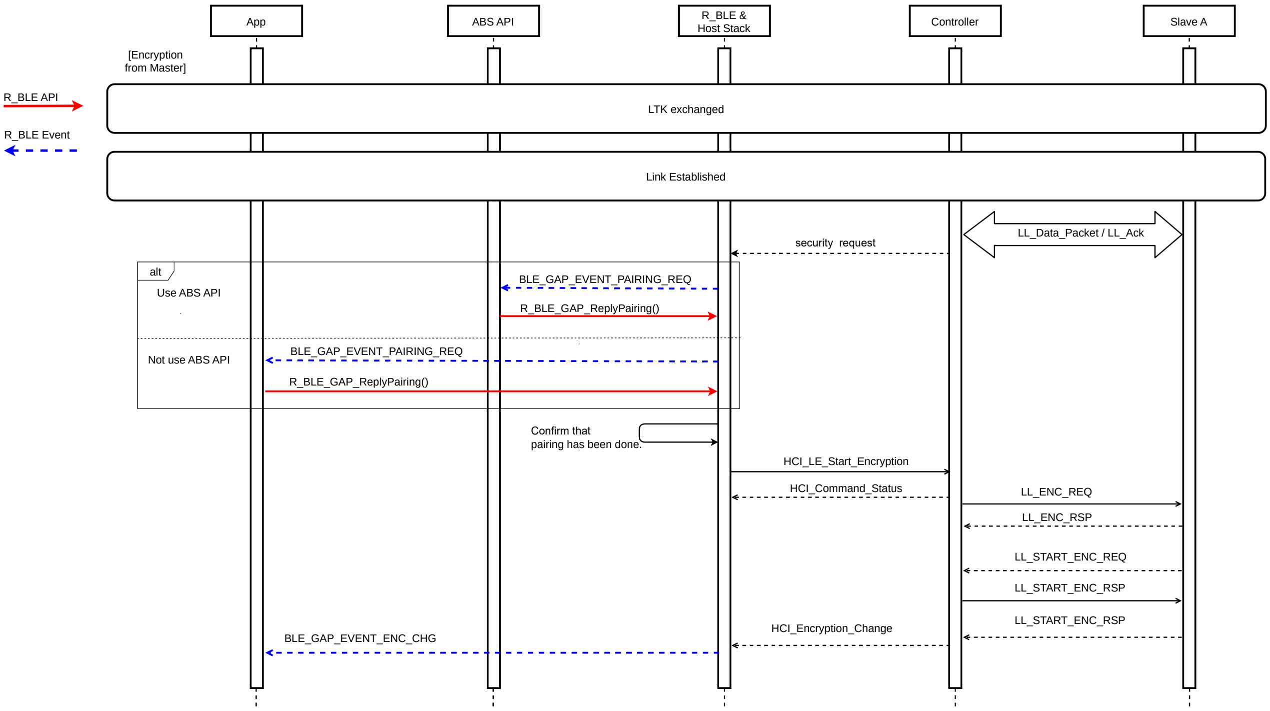

Figure 9.6 Sequence of response to an encryption request from remote device(slave)

9.3.3 Completion of encryption

If the encryption has been completed successfully, BLE_GAP_EVENT_ENC_CHG event is notified. If the encryption has been failed because the remote device lost the LTK, BLE_GAP_EVENT_PAIRING_COMP event with result: BLE_ERR_SMP_LE_LOC_KEY_MISSING(0x2014) is notified. If the event is received, delete the local device LTK and do pairing again and encrypt.

9.4 Privacy

The privacy feature allows local device to change the address not to be identified from other devices. There are two privacy mode: Network Privacy Mode and Device Privacy Mode. In Network Privacy Mode, both local device and remote device use RPA. In Device Privacy Mode, only local device uses RPA. Default is Network Privacy Mode.

The following describes how to use the privacy feature.

9.4.1 Generate and resolve 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 BLE 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. If only the local device is uses RPA or it is in unpaired state, register a dummy remote device identity address and IRK 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 1-4 procedure is shown below.

```
/** 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);

            /* Dummy remote address & remote IRK */
            st_ble_gap_rslv_list_key_set_t peer_irk;

            memset(peer_irk.remote_irk, 0xAA, BLE_GAP_IRK_SIZE);
            peer_irk.local_irk_type = BLE_GAP_RL_LOC_KEY_REGISTERED;
            memset(gs_rem_bd_addr.addr, 0x55, 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, 0x55, 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 or Connection 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.

[Connection]
When create a connection by R_BLE_GAP_CreateConn, configure RPA as own address type.

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.

2. Pairing
   Receive the remote device IRK and identity address by 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, register a dummy IRK. 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 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 1-3 procedure, the BLE 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.

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 multiple 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 “RE01B Group Bluetooth Low Energy Profile Developer’s Guide (R01AN5638)”.

This chapter introduces the profiles and services provided by Renesas and explains APIs for each GATT procedure including examples of how to use them.

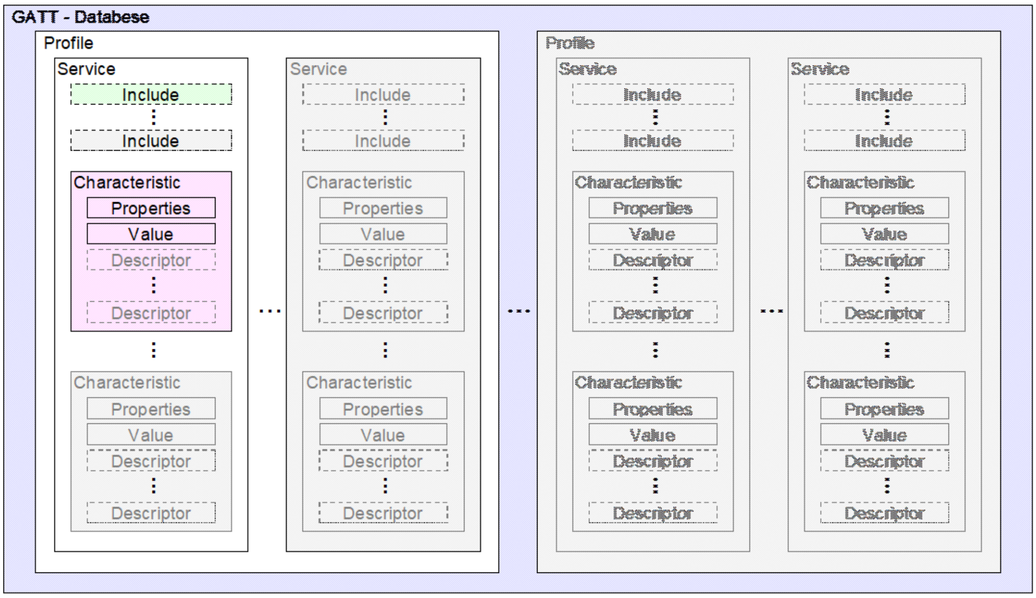

Figure 10.1 Data structure of GATT database

10.1 Standard profile and Standard Service

Standard profiles and services can be used in user applications using QE for BLE. RE01B 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 RE01B

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
Body Composition Service **BCS**	Body Composition Feature	Read
	Body Composition Measurement	Indicate
Continuous Glucose Monitoring Service **CGMS**	CGM Measurement	Notify
	CGM Feature	Read
	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
Device Information	Manufacturer Name String	Read

Service	Characteristic	GATT Procedure
Service **DIS**	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
	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, 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
	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
	Record Access Control Point	Write, Indicate
Reconnection Configuration Service **RCS**	RC Feature	Read
	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
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 following description, we will use 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 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", BLE 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 BLE 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.

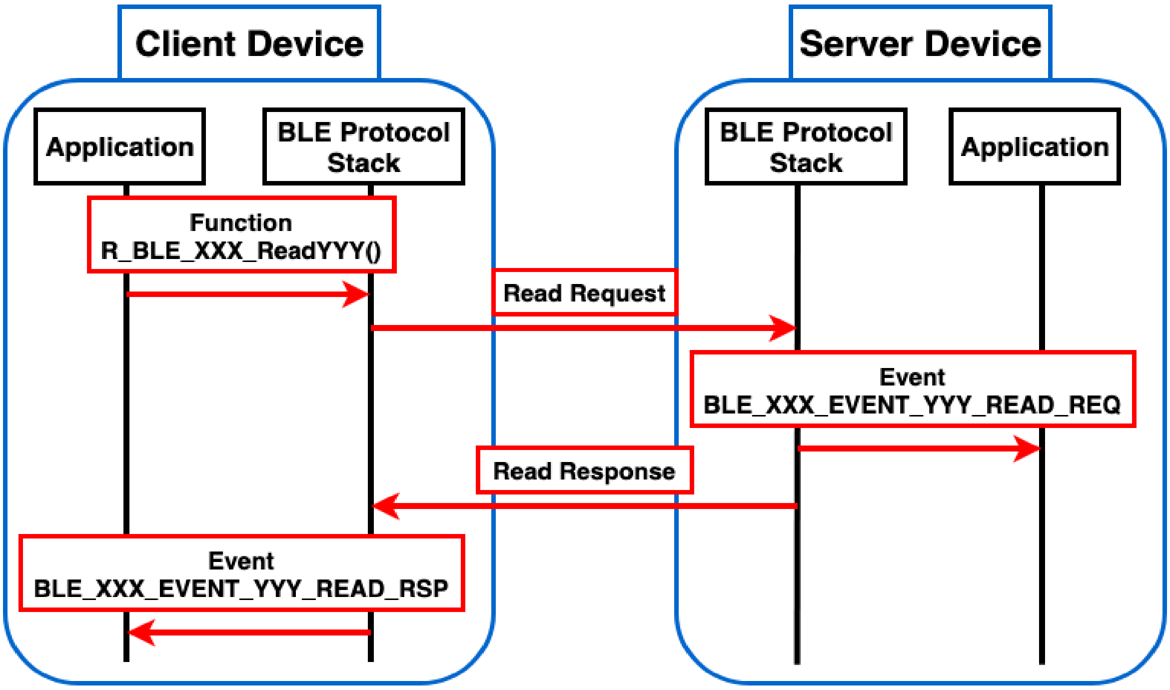

Figure 10.2 Flow of Read operation

10.2.2 Write operation

Write operation is 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 BLE 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, BLE 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.

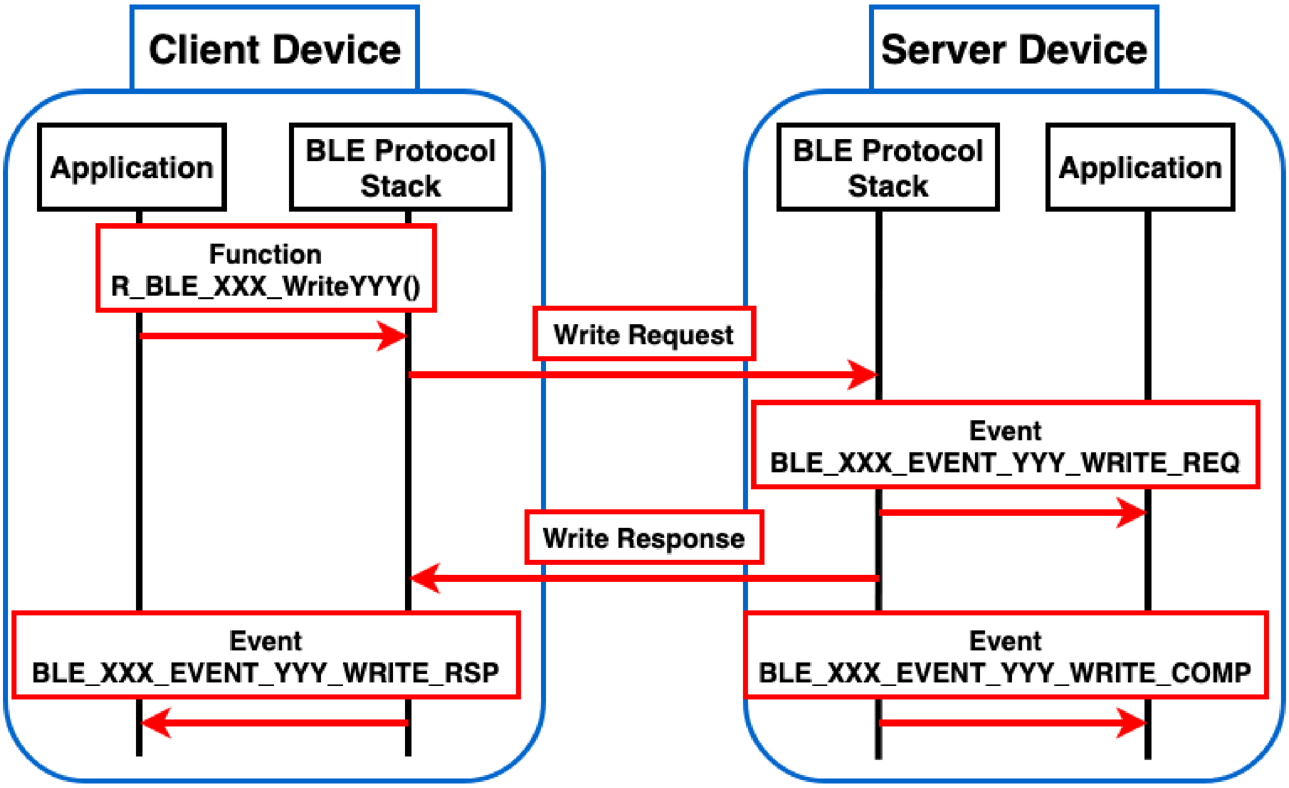

Figure 10.3 Flow of Write operation

10.2.3 WriteWithoutResponse operation

WriteWithoutResponse operation is 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 BLE 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.

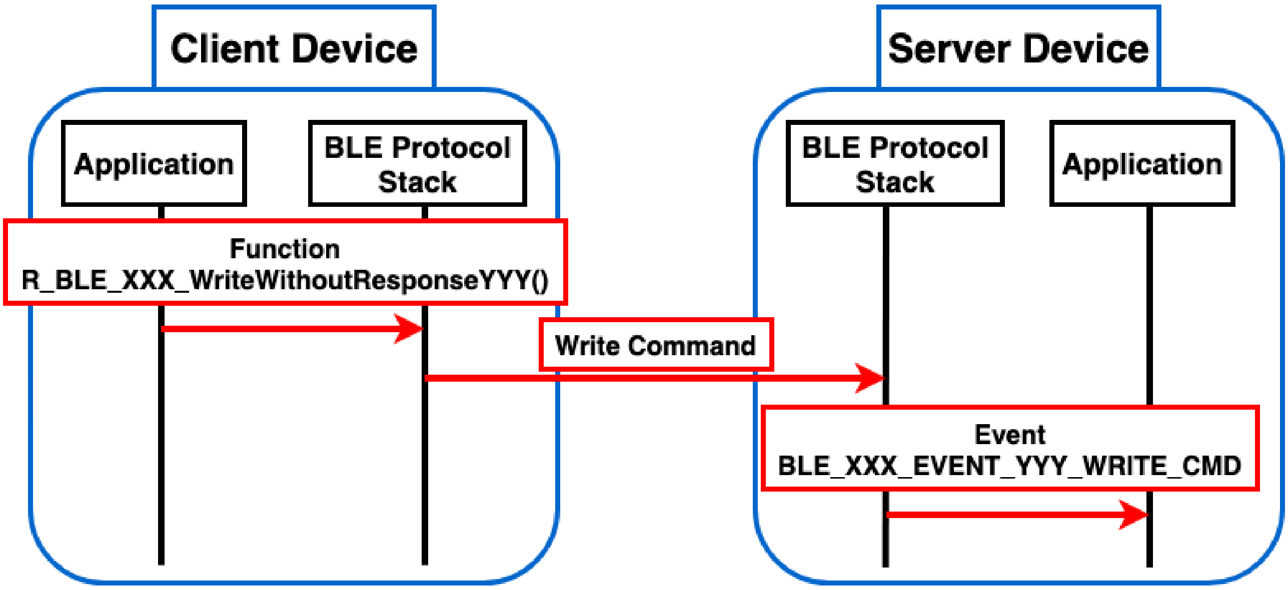

Figure 10.4 Flow of WriteWithoutResponse operation

10.2.4 Notification operation

Notification operation is procedure to send data from the GATT server to the GATT client. For Notification operation, the CCCD must have been added as 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 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 BLE Protocol Stack.

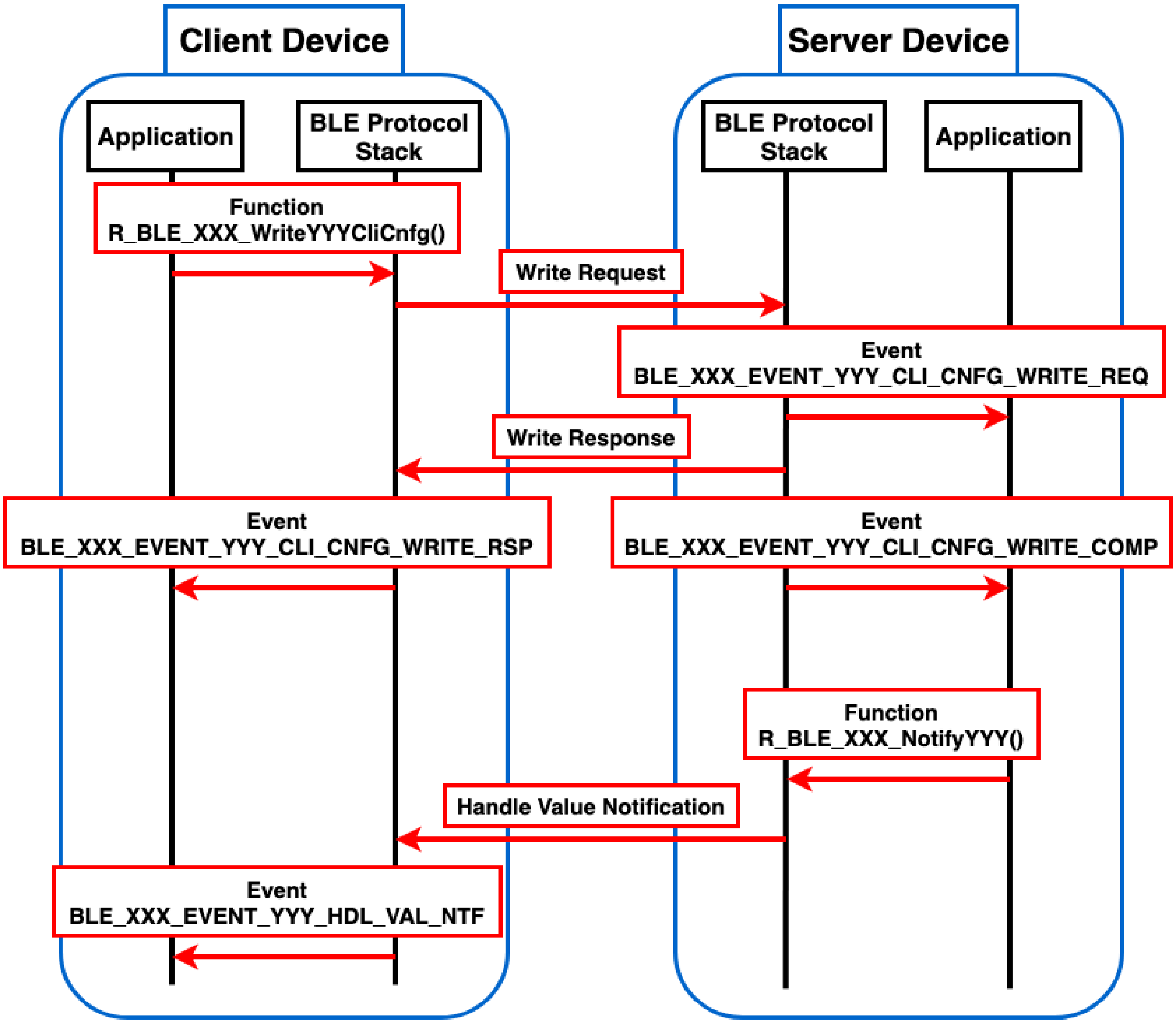

Figure 10.5 Flow of Notification operation

10.2.5 Indication operation

Indication operation is procedure to send data from GATT server to GATT client. For Indication operation, the CCCD must have been added as 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 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 BLE Protocol Stack. After the event "BLE_XXX_EVENT_YYY_HDL_VAL_IND", BLE Protocol Stack sends "Handle Value Confirmation", so you do not need to add any process to respond in application.

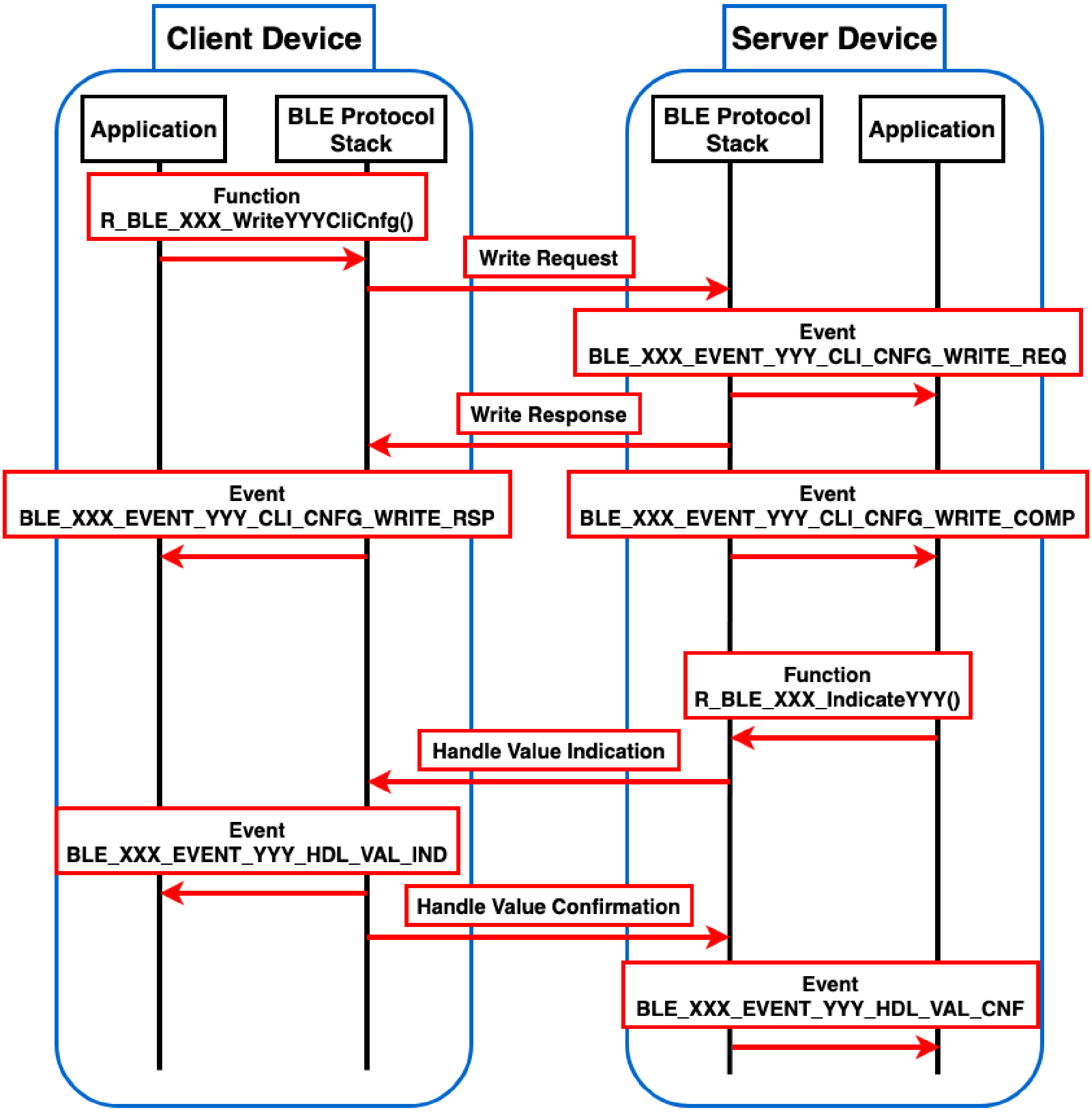

Figure 10.6 Flow of Indication operation

10.2.6 ReliableWrite operation

The ReliableWrite operation is procedure to send data from GATT client to GATT server, ensure that the correct values are written, and then reflect it in the GATT database. There are two steps for ReliableWrite 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 to highly reliable data communication.
APIs of ReliableWrite operation is not included in the API of service generated from QE for BLE, so it must be implemented using APIs from BLE Protocol Stack. In addition, Characteristic Extended Properties Descriptor must have been added as a descriptor for ReliableWrite 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 ReliableWrite 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.

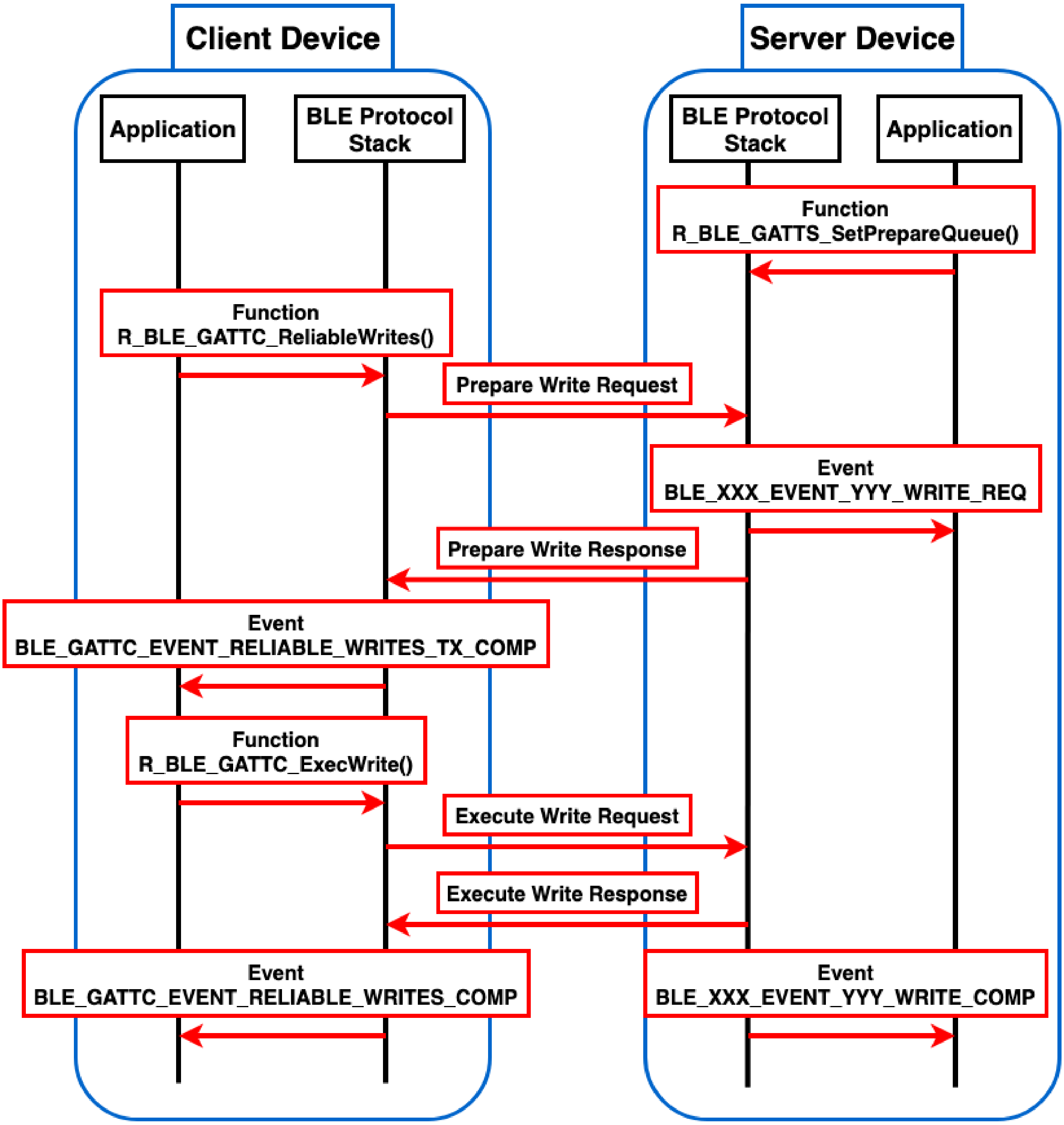

Figure 10.7 Flow of ReliableWrite operation

10.2.7 Broadcast Operation

Broadcast operation is 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 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 BLE Protocol Stack. In addition, Server Characteristic Configuration Properties Descriptor must have been 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), 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 (5.7), 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.

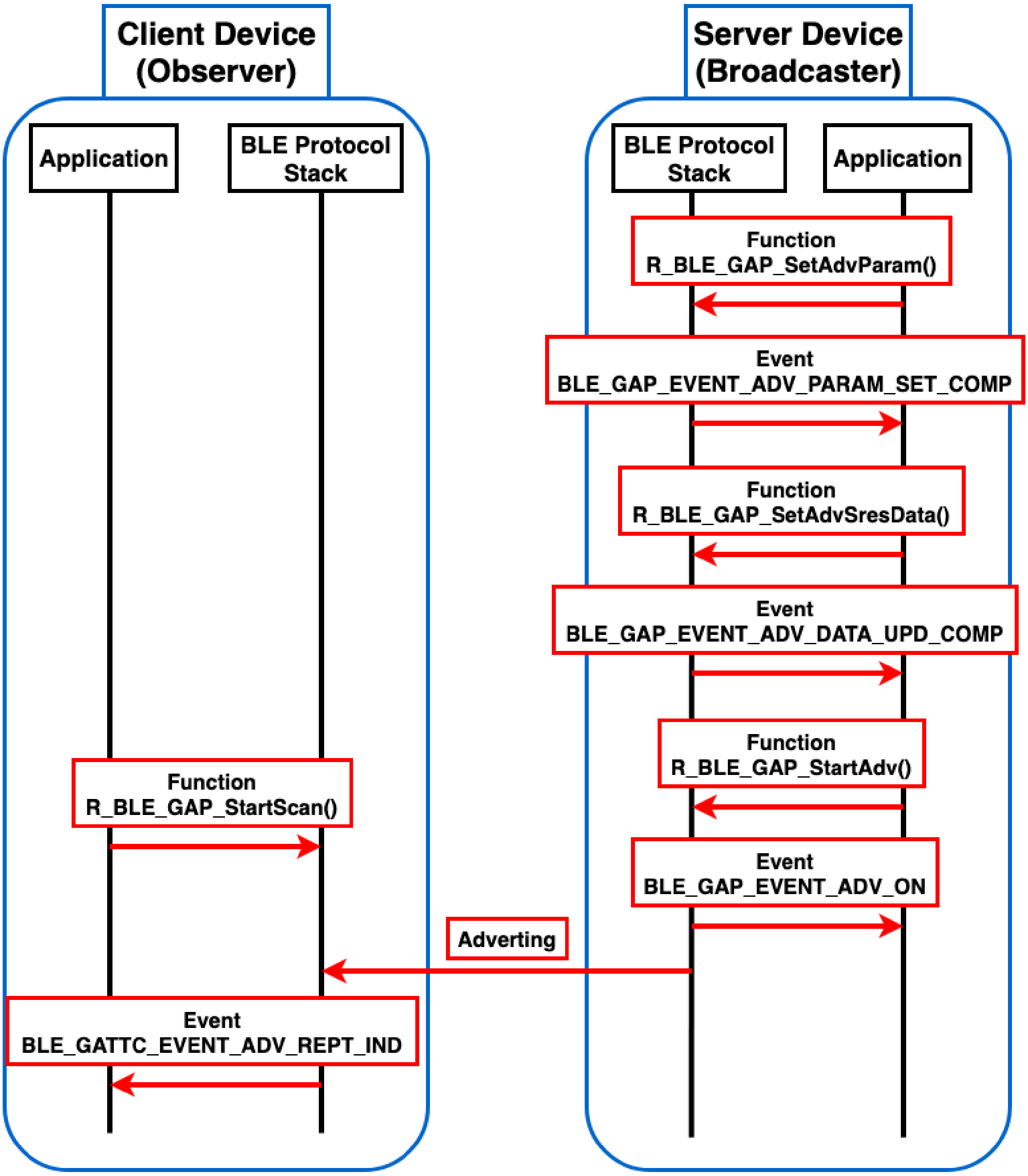

Figure 10.8 Flow of Broadcast operation

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)
```

Add library for using switch

```
/* some code is omitted */

static void sw_cb(void)
{
    R_BLE_LSC_ReadBlinkRate(g_conn_hdl);
}
```

Start Read operation in callback of switch input

```
/* 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;
    }
}
```

Start Write operation depending on received value

```
/* 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 */
}
```

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"

/* 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;
    }
}

/* 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;
}

/* some code is omitted */

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

    R_BLE_BOARD_Init();

/* some code is omitted */

}
```

Add library for using LED

Blink LED depending on received value

Write CCCD after discovery is completed

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 "3.7 Logger" in "Bluetooth Low Energy Sample code (using CMSIS Driver Package) (R01AN5606)".

11.1 Using Logger function

If changing BLE_DEFAULT_LOG_LEVEL before including r_ble_logger.h, the log level can be changed. If it 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 (r_ble_appapp.c).

```
[app_main.c]
#define BLE_DEFAULT_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 );

(OMISSION)

    switch (type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            BLE_LOG_ERR("R_BLE_ABS_StartLegacyAdv");
            BLE_LOG_WRN("interval=%d", (uint32_t)(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)

```
[r_ble_appapp.c]
#include "r_ble_api.h"
#define BLE_DEFAULT_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 (r_ble_appapp.c)

One line is displayed by one logger call, therefore line breaks are not required.

```
app_main: [ERR] (gap_cb:259) R_BLE_ABS_StartLegacyAdv
app_main: [WRN] (gap_cb:260) interval=160
app_main: [DBG] (gap_cb:262) data[02]
app_main: [DBG] (gap_cb:262) data[01]
app_main: [DBG] (gap_cb:262) data[06]
app_main: [DBG] (gap_cb:262) data[05]
app_main: [DBG] (gap_cb:262) data[08]
app_main: [DBG] (gap_cb:262) data[52]
app_main: [DBG] (gap_cb:262) data[42]
app_main: [DBG] (gap_cb:262) data[4C]
app_main: [DBG] (gap_cb:262) data[45]
app_main: [DBG] (gap_cb:262) data[11]
app_main: [DBG] (gap_cb:262) data[06]
app_main: [DBG] (gap_cb:262) data[E0]
app_main: [DBG] (gap_cb:262) data[FC]
app_main: [DBG] (gap_cb:262) data[8E]
app_main: [DBG] (gap_cb:262) data[8E]
app_main: [DBG] (gap_cb:262) data[96]
app_main: [DBG] (gap_cb:262) data[B4]
app_main: [DBG] (gap_cb:262) data[01]
app_main: [DBG] (gap_cb:262) data[AB]
app_main: [DBG] (gap_cb:262) data[67]
app_main: [DBG] (gap_cb:262) data[42]
app_main: [DBG] (gap_cb:262) data[05]
app_main: [DBG] (gap_cb:262) data[5F]
app_main: [DBG] (gap_cb:262) data[26]
app_main: [DBG] (gap_cb:262) data[19]
app_main: [DBG] (gap_cb:262) data[83]
app_main: [DBG] (gap_cb:262) data[58]
appapp: [YYY] (appapp:18) data[09]
appapp: [YYY] (appapp:18) data[09]
appapp: [YYY] (appapp:18) data[52]
appapp: [YYY] (appapp:18) data[42]
appapp: [YYY] (appapp:18) data[4C]
appapp: [YYY] (appapp:18) data[45]
appapp: [YYY] (appapp:18) data[2D]
appapp: [YYY] (appapp:18) data[44]
appapp: [YYY] (appapp:18) data[45]
appapp: [YYY] (appapp:18) data[56]
app_main: [XXX] (gap_cb:265) advlen=27, sreslen=10
receive BLE_GAP_EVENT_ADV_ON result : 0x0000, adv_hdl : 0x0000
```

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 "r_ble_api.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_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 How to implement user code".
Newly create r_ble_cmd_rf.h in “src” folder.

```
[src\r_ble_cmd_rf.h]
#include "r_ble_api.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_api.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 = 0;
extern uint32_t pl_get_elapsed_time_ms2(bool expired);

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 uint32_t log_cnt = 0;
static void show_rf_logs( void )
{
    show_rf_log( gs_rf_log[log_cnt].elapsed_time, gs_rf_log[log_cnt].event_type,
gs_rf_log[log_cnt].event_data, gs_rf_log[log_cnt].start_end );
    log_cnt++;
    if( log_cnt >= BLE_RF_LOG_NUM_MAX )
    {
        log_cnt = 0;
    }
    else
    {
        R_BLE_SetEvent( show_rf_logs );
    }
}

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");
        R_BLE_SetEvent( show_rf_logs );
    }
    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)

Newly create r_ble_timer_sotb2.c in “src” folder. This code uses the timer of AGT0.

```
#include "r_ble_api.h"
#include "r_ble_cmd_rf.h"

#if (BLE_CFG_HCI_MODE_EN == 0) && (BSP_CFG_RTOS_USED == 0)

#include "r_core_cfg.h"

static uint32_t gs_timer_clock_hz;
static uint32_t gs_us_per_tick;
static uint32_t gs_current_timeout_tick;
static uint32_t gs_current_timeout_ms;
static uint32_t gs_elapsed_timeout_ms;

#define USE_AGT_CH              BLE_CFG_SOFT_TIMER_AGT_CH
#define AGT_AGTI_PRIORITY       (3)                 ///< AGTn_AGTI priority value(set to 0 to 3, 0 is 
highest priority.)

#if (USE_AGT_CH == 1)
#define AGTn                    AGT0
#define AGT_MSTPDn              MSTPD3
#define AGTn_AGTI_EVENT_NUM     SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI
#define AGT_IESR_VAL            (0x00000013)
#define AGT_SYSTEM_LOCK         SYSTEM_LOCK_AGT0
#else /* (USE_AGT_CH == 1) */
#define AGTn                    AGT1
#define AGT_MSTPDn              MSTPD2
#define AGTn_AGTI_EVENT_NUM     SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI
#define AGT_IESR_VAL            (0x00000006)
#define AGT_SYSTEM_LOCK         SYSTEM_LOCK_AGT1
#endif /* (USE_AGT_CH == 1) */


#define TIMER_ENTER_CRITICAL()   GLOBAL_INT_DISABLE()
#define TIMER_EXIT_CRITICAL()    GLOBAL_INT_RESTORE()

void pl_start_timer2(uint32_t timeout_ms);

static void timer_cb2(void)
{
    /* interrupt disable */
    TIMER_ENTER_CRITICAL();

    /* Disable AGTn_AGTI interrupt */
    R_NVIC_DisableIRQ(AGTn_AGTI_EVENT_NUM);

    /* interrupt restore */
    TIMER_EXIT_CRITICAL();

    pl_start_timer2(63000);
}

void pl_init_timer2(void)
{
    /* Lock AGTn resource */
    if( 0 != R_SYS_ResourceLock(AGT_SYSTEM_LOCK) )
    {
        __WFI();
        return;
    }

    /* interrupt disable */
    TIMER_ENTER_CRITICAL();

    /* Clear AGTn module stop  */
    MSTP->MSTPCRD_b.AGT_MSTPDn = 0U;

    /* Initialize AGTn */
    AGTn->AGTCR            = 0x00U;
```

```
    AGTn->AGTMR1_b.TCK    = 4U; /* TCK[2:0]  -> 1 0 0: AGTLCLK(LOCO:32.768kHz)>>AGTMR2.CKS */

    AGTn->AGTMR2_b.CKS    = 5U; /* CKS[2:0]  -> 1 0 1: 1/32 -> 32.768kHz/32=1.024kHz=1cycle 977usec */
    AGTn->AGTMR2_b.LPM    = 0U; /* Normal mode */

    gs_timer_clock_hz = 32768 >> AGTn->AGTMR2_b.CKS;
    gs_us_per_tick    = (uint32_t)(1000000 / gs_timer_clock_hz );

    /* When AGTLCLK is selected as the count source, AGT_MSTPDn needs to be 1. */
    MSTP->MSTPCRD_b.AGT_MSTPDn = 1U;

    /* Initialize AGTn(AGTn_AGTI) interrupt handler */
    /* Register AGTn_AGTI interrupt handler */
    if ((-1) == R_SYS_IrqEventLinkSet(AGTn_AGTI_EVENT_NUM, AGT_IESR_VAL, timer_cb2))
    {
        __WFI();
        return;
    }

    /* AGT0_AGTI priority setting */
    R_NVIC_SetPriority(AGTn_AGTI_EVENT_NUM, AGT_AGTI_PRIORITY);
    if (R_NVIC_GetPriority (AGTn_AGTI_EVENT_NUM) != AGT_AGTI_PRIORITY)
    {
        __WFI();
        return;
    }

    /* interrupt restore */
    TIMER_EXIT_CRITICAL();
}

void pl_start_timer2(uint32_t timeout_ms)
{
    /* interrupt disable */
    TIMER_ENTER_CRITICAL();

    /* Disable AGTn_AGTI interrupt */
    R_NVIC_DisableIRQ(AGTn_AGTI_EVENT_NUM);

    gs_current_timeout_ms = MIN(timeout_ms, 63000);
    gs_elapsed_timeout_ms = 0;

    /* Clear AGTn module stop  */
    MSTP->MSTPCRD_b.AGT_MSTPDn = 0U;

    /* Set AGTn counter */
    gs_current_timeout_tick = (gs_current_timeout_ms * gs_timer_clock_hz) / 1000;
    AGTn->AGT = (uint16_t)(gs_current_timeout_tick);

    /* Start AGTn count */
    AGTn->AGTCR_b.TEDGF  = 0U;
    AGTn->AGTCR_b.TUNDF  = 0U;
    AGTn->AGTCR_b.TCMAF  = 0U;
    AGTn->AGTCR_b.TCMBF  = 0U;
    AGTn->AGTCR_b.TSTART = 1U;

    /* When AGTLCLK is selected as the count source, AGT_MSTPDn needs to be 1. */
    MSTP->MSTPCRD_b.AGT_MSTPDn = 1U;

    /* Enable AGTn_AGTI interrupt */
    R_SYS_IrqStatusClear(AGTn_AGTI_EVENT_NUM);
    R_NVIC_ClearPendingIRQ(AGTn_AGTI_EVENT_NUM);
    R_NVIC_EnableIRQ(AGTn_AGTI_EVENT_NUM);

    /* interrupt restore */
    TIMER_EXIT_CRITICAL();
}

uint32_t pl_get_elapsed_time_ms2(bool expired)
{
    uint32_t elapsed_time_from_prev_update_ms;
    uint32_t total_elapsed_timeout_ms;

    uint16_t cmstr;
```

```
    uint16_t cmcnt;
    uint16_t cmudf;

    /* interrupt disable */
    TIMER_ENTER_CRITICAL();

    /* Clear AGTn module stop  */
    MSTP->MSTPCRD_b.AGT_MSTPDn = 0U;
    cmstr = AGTn->AGTCR_b.TSTART;
    cmudf = AGTn->AGTCR_b.TUNDF;
    cmcnt = (uint16_t)(gs_current_timeout_tick - AGTn->AGT);
    /* When AGTLCLK is selected as the count source, AGT_MSTPDn needs to be 1. */
    MSTP->MSTPCRD_b.AGT_MSTPDn = 1U;

    if (expired)
    {
        elapsed_time_from_prev_update_ms = gs_current_timeout_ms - gs_elapsed_timeout_ms;
        gs_elapsed_timeout_ms = gs_current_timeout_ms;
    }
    else if (cmstr)
    {
        if( cmudf )
        {
            cmcnt = gs_current_timeout_tick;
        }
        total_elapsed_timeout_ms = ((uint32_t)cmcnt * gs_us_per_tick) / 1000;
        elapsed_time_from_prev_update_ms = total_elapsed_timeout_ms - gs_elapsed_timeout_ms;
        gs_elapsed_timeout_ms = total_elapsed_timeout_ms;
    }
    else
    {
        elapsed_time_from_prev_update_ms = 0;
    }
    /* interrupt restore */
    TIMER_EXIT_CRITICAL();

    return elapsed_time_from_prev_update_ms;
}

#endif /* (BLE_CFG_HCI_MODE_EN == 0) && (BSP_CFG_RTOS_USED == 0) */
```

The following code saves the RF communication timing notification as logs.

```
[Device\BLE\platform\r_ble_pf_functions.c]
extern uint32_t gs_timer_elapsed_time;
#include "r_ble_cmd_rf.h"

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;
    }
}
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;
    }
}

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 starts the timer of AGT0 and registers rf command.

```
[app_main.c]
#include "r_ble_api.h"

#include "r_ble_cmd_rf.h"
/* CommandLine parameters */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
    &g_rf_cmd,
};
extern void pl_init_timer2(void);
extern void pl_start_timer2(uint32_t timeout_ms);

(OMISSION)

void app_main(void)
{
    (OMISSION)
    /* Create timer */
    pl_init_timer2();
    pl_start_timer2(63000);
    /* Configure CommandLine */
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, ARRAY_SIZE(gsp_cmds));
    R_BLE_CMD_SetResetCb(ble_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.

```
[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 Slave. 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,
                .min_ce_length = 0xFFFF,
                .max_ce_length = 0xFFFF,
            };

            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

Refer to “RE01B Group Product with 1.5-Mbyte Flash Memory User’s Manual: Hardware (R01UH0903)”.

11.6.2 Generating MOT file

When checking [Project] → [Properties] → [C/C++ Build] → [Settings] → [Tool Settings] → [Cross ARM GNU Create Flash Image] → [General] → [Output file format] to "Motorola S-record", MOT file is generated.

11.6.3 Outputting detail to MAP file

When checking [Project] → [Properties] → [C/C++ Build] → [Settings] → [Tool Settings] → [Cross ARM C Linker] → [Misscellaneous] → [Cross reference (-Xlinker --cref)] to ON, the details of MAP file are outputted.

11.6.4 Optimization

When setting [Project] → [Properties] → [C/C++ Build] → [Settings] → [Tool Settings] → [Optimization] → [Optimization Level] to "None (-O0)", the memory contents can be confirmed during debugging.

11.6.5 Using %f with printf

When checking [Project] → [Properties] → [C/C++ Build] → [Settings] → [Tool Settings] → [Cross ARM C Linker] → [Miscellaneous] → [Use float with nano printf (-u _printf_float)] to ON, %f can be used with printf.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar.25.2021	–	First edition issued.

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

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

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

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

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

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

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

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

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

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

Notice

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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