User's Manual

Bluetooth® Low Energy Protocol Stack

API Reference Manual: Basics

Renesas MCU

Target Device

RL78/G1D

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.18 Aug. 2016

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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended 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; and industrial robots etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(2012.4)

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

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

1. Handling of Unused Pins

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

   ¾ The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

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

   ¾ The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   ¾ The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   ¾ When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   ¾ The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

How to Use This Manual

1. Purpose and Target Readers

This manual describes the API (Application Program Interface) of the basic features of the Bluetooth Low Energy protocol stack (BLE software), which is used to develop Bluetooth applications that incorporate the Renesas Bluetooth low energy microcontroller RL78/G1D. It is intended for users designing application systems incorporating this software. A basic knowledge of microcontrollers and Bluetooth low energy is necessary in order to use this manual.

Related documents

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Document Name	Document No.
Bluetooth Low Energy Protocol Stack	
User's Manual	R01UW0095E
API Reference Manual: Basics	This manual
API Reference Manual: FMP	R01UW0089E
API Reference Manual: PXP	R01UW0090E
API Reference Manual: HTP	R01UW0091E
API Reference Manual: BLP	R01UW0092E
API Reference Manual: HOGP	R01UW0093E
API Reference Manual: ScPP	R01UW0094E
API Reference Manual: HRP	R01UW0097E
API Reference Manual: CSCP	R01UW0098E
API Reference Manual: CPP	R01UW0099E
API Reference Manual: GLP	R01UW0103E
API Reference Manual: TIP	R01UW0106E
API Reference Manual: RSCP	R01UW0107E
API Reference Manual: ANP	R01UW0108E
API Reference Manual: PASP	R01UW0109E
API Reference Manual: LNP	R01UW0113E
Sample Program Application Note	R01AN1375E
rBLE command specifications	R01AN1376E

List of Abbreviations and Acronyms

Abbreviation	Full Form	Remark
ANP	Alert Notification Profile	
ANS	Alert Notification Service	
API	Application Programming Interface	
ATT	Attribute Protocol	
BAS	Battery Service	
BB	Base Band	
BD_ADDR	Bluetooth Device Address	
BLE	Bluetooth low energy	
BLP	Blood Pressure Profile	
BLS	Blood Pressure Service	
CPP	Cycling Power Profile	
CPS	Cycling Power Service	
CSCP	Cycling Speed and Cadence Profile	
CSCS	Cycling Speed and Cadence Service	
CSRK	Connection Signature Resolving Key	
CTS	Current Time Service	
DIS	Device Information Service	
EDIV	Encrypted Diversifier	
FMP	Find Me Profile	
GAP	Generic Access Profile	
GATT	Generic Attribute Profile	
GLP	Glucose Profile	
GLS	Glucose Service	
HCI	Host Controller Interface	
HID	Human Interface Device	
HIDS	HID Service	
HOGP	HID over GATT Profile	
HRP	Heart Rate Profile	
HRS	Heart Rate Service	
HTP	Health Thermometer Profile	
HTS	Health Thermometer Service	
IAS	Immediate Alert Service	
IRK	Identity Resolving Key	
L2CAP	Logical Link Control and Adaptation Protocol	
LE	Low Energy	
LL	Link Layer	

Abbreviation	Full Form	Remark
LLS	Link Loss Service	
LNP	Location and Navigation Profile	
LNS	Location and Navigation Service	
LTK	Long Term Key	
MCU	Micro Controller Unit	
MITM	Man-in-the-middle	
MTU	Maximum Transmission Unit	
OOB	Out of Band	
OS	Operating System	
PASP	Phone Alert Status Profile	
PASS	Phone Alert Status Service	
PXP	Proximity Profile	
RF	Radio Frequency	
RSCP	Running Speed and Cadence Profile	
RSCS	Running Speed and Cadence Service	
RSSI	Received Signal Strength Indication	
ScPP	Scan Parameters Profile	
ScPS	Scan Parameters Service	
SM	Security Manager	
SMP	Security Manager Protocol	
STK	Short Term Key	
TK	Temporary Key	
TPS	Tx Power Service	
UART	Universal Asynchronous Receiver Transmitter	
UUID	Universal Unique Identifier	

Abbreviation	Full Form	Remark
APP	Application	
CSI	Clocked Serial Interface	
IIC	Inter-Integrated Circuit	
RSCIP	Renesas Serial Communication Interface Protocol	
VS	Vendor Specific	

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

Contents

1. Overview 1

2. General 2

2.1 BLE Software and its APIs 2

2.2 rBLE API 3

2.2.1 Language 3

2.2.2 rBLE API procedure 3

2.2.3 Classification of rBLE API profiles 4

2.2.4 Handling of parameters of rBLE API functions 5

2.2.5 Registration of callback functions for event notification 5

2.2.6 Basic operation of callback functions for event notification 7

2.3 BLE Software State Transitions 9

2.3.1 rBLE_Core state transitions 9

2.3.2 rBLE_HOST state transitions 10

2.4 BLE Software Initialization Procedure 11

3. Common Definitions 11

3.1 Standard Typedef 11

3.2 Generic Definitions 11

3.3 GATT Definitions 16

4. Initialization 19

4.1 Definitions 19

4.2 Functions 20

4.2.1 RBLE_Init 20

4.3 Events 21

4.3.1 RBLE_INIT_EVENT_MODE_CHANGE 21

5. Generic Access Profile 22

5.1 Definitions 22

5.2 Functions 38

5.2.1 RBLE_GAP_Reset 39

5.2.2 RBLE_GAP_Set_Name 39

5.2.3 RBLE_GAP_Observation_Enable 40

5.2.4 RBLE_GAP_Observation_Disable 40

5.2.5 RBLE_GAP_Broadcast_Enable 41
5.2.6 RBLE_GAP_Broadcast_Disable 43
5.2.7 RBLE_GAP_Set_Bonding_Mode 43
5.2.8 RBLE_GAP_Set_Security_Request 43
5.2.9 RBLE_GAP_Get_Device_Info 44
5.2.10 RBLE_GAP_Get_White_List_Size 44
5.2.11 RBLE_GAP_Add_To_White_List 44
5.2.12 RBLE_GAP_Del_From_White_List 45
5.2.13 RBLE_GAP_Get_Remote_Device_Name 46
5.2.14 RBLE_GAP_Get_Remote_Device_Info 47
5.2.15 RBLE_GAP_Device_Search 47
5.2.16 RBLE_GAP_Set_Random_Address 48
5.2.17 RBLE_GAP_Set_Privacy_Feature 48
5.2.18 RBLE_GAP_Create_Connection 49
5.2.19 RBLE_GAP_Connection_Cancel 50
5.2.20 RBLE_GAP_Disconnect 50
5.2.21 RBLE_GAP_Start_Bonding 51
5.2.22 RBLE_GAP_Bonding_Info_Ind 52
5.2.23 RBLE_GAP_Bonding_Response 53
5.2.24 RBLE_GAP_Change_Connection_Param 54
5.2.25 RBLE_GAP_Channel_Map_Req 55
5.2.26 RBLE_GAP_Read_RSSI 55
5.2.27 RBLE_GAP_Authorized_Ind 55
5.3 Events 56
5.3.1 RBLE_GAP_EVENT_RESET_RESULT 57
5.3.2 RBLE_GAP_EVENT_SET_NAME_COMP 57
5.3.3 RBLE_GAP_EVENT_OBSERVATION_ENABLE_COMP 57
5.3.4 RBLE_GAP_EVENT_OBSERVATION_DISABLE_COMP 57
5.3.5 RBLE_GAP_EVENT_BROADCAST_ENABLE_COMP 57
5.3.6 RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP 58
5.3.7 RBLE_GAP_EVENT_SET_BONDING_MODE_COMP 58
5.3.8 RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP 58
5.3.9 RBLE_GAP_EVENT_GET_DEVICE_INFO_COMP 58
5.3.10 RBLE_GAP_EVENT_GET_WHITE_LIST_SIZE_COMP 59
5.3.11 RBLE_GAP_EVENT_ADD_TO_WHITE_LIST_COMP 59
5.3.12 RBLE_GAP_EVENT_DEL_FROM_WHITE_LIST_COMP 59
5.3.13 RBLE_GAP_EVENT_GET_REMOTE_DEVICE_NAME_COMP 59
5.3.14 RBLE_GAP_EVENT_GET_REMOTE_DEVICE_INFO_COMP 60

5.3.15 RBLE_GAP_EVENT_DEVICE_SEARCH_COMP 60
5.3.16 RBLE_GAP_EVENT_DEVICE_SEARCH_RESULT_IND 61
5.3.17 RBLE_GAP_EVENT_RPA_RESOLVED 61
5.3.18 RBLE_GAP_EVENT_SET_RANDOM_ADDRESS_COMP 61
5.3.19 RBLE_GAP_EVENT_SET_PRIVACY_FEATURE_COMP 62
5.3.20 RBLE_GAP_EVENT_CONNECTION_COMP 62
5.3.21 RBLE_GAP_EVENT_CONNECTION_CANCEL_COMP 62
5.3.22 RBLE_GAP_EVENT_DISCONNECT_COMP 63
5.3.23 RBLE_GAP_EVENT_ADVERTISING_REPORT_IND 63
5.3.24 RBLE_GAP_EVENT_BONDING_COMP 64
5.3.25 RBLE_GAP_EVENT_BONDING_REQ_IND 65
5.3.26 RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_REQ_IND 66
5.3.27 RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_COMP 66
5.3.28 RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_RESPONSE 66
5.3.29 RBLE_GAP_EVENT_CHANNEL_MAP_REQ_COMP 66
5.3.30 RBLE_GAP_EVENT_READ_RSSI_COMP 67
5.3.31 RBLE_GAP_EVENT_WR_CHAR_IND 67
5.3.32 RBLE_GAP_EVENT_COMMAND_DISALLOWED_IND 67

6. Security Manager 68

6.1 Definitions 68
6.2 Functions 72
6.2.1 RBLE_SM_Set_Key 73
6.2.2 RBLE_SM_Start_Enc 73
6.2.3 RBLE_SM_Tk_Req_Resp 74
6.2.4 RBLE_SM_Ltk_Req_Resp 74
6.2.5 RBLE_SM_Irk_Req_Resp 75
6.2.6 RBLE_SM_Csrk_Req_Resp 75
6.2.7 RBLE_SM_Chk_Bd_Addr_Req_Resp 76
6.3 Events 77
6.3.1 RBLE_SM_EVENT_SET_CNF 78
6.3.2 RBLE_SM_ENC_START_IND 78
6.3.3 RBLE_SM_TK_REQ_IND 78
6.3.4 RBLE_SM_LTK_REQ_IND 79
6.3.5 RBLE_SM_LTK_REQ_FOR_ENC_IND 79
6.3.6 RBLE_SM_IRK_REQ_IND 80
6.3.7 RBLE_SM_CSRK_REQ_IND 80
6.3.8 RBLE_SM_KEY_IND 80

6.3.9 RBLE_SM_CHK_BD_ADDR_REQ 80
6.3.10 RBLE_SM_TIMEOUT_EVT 81
6.3.11 RBLE_SM_EVENT_COMMAND_ERROR_IND 81

7. Generic Attribute Profile 82

7.1 Definitions 82
7.2 Functions 96
7.2.1 RBLE_GATT_Enable 97
7.2.2 RBLE_GATT_Discovery_Service_Request 98
7.2.3 RBLE_GATT_Discovery_Char_Request 99
7.2.4 RBLE_GATT_Discovery_Char_Descriptor_Request 100
7.2.5 RBLE_GATT_Read_Char_Request 101
7.2.6 RBLE_GATT_Write_Char_Request 103
7.2.7 RBLE_GATT_Write_Reliable_Request 104
7.2.8 RBLE_GATT_Execute_Write_Char_Request 104
7.2.9 RBLE_GATT_Notify_Request 105
7.2.10 RBLE_GATT_Indicate_Request 105
7.2.11 RBLE_GATT_Write_Response 105
7.2.12 RBLE_GATT_Set_Permission 106
7.2.13 RBLE_GATT_Set_Data 106
7.3 Events 107
7.3.1 RBLE_GATT_EVENT_DISC_SVC_ALL_CMP 108
7.3.2 RBLE_GATT_EVENT_DISC_SVC_ALL_128_CMP 108
7.3.3 RBLE_GATT_EVENT_DISC_SVC_BY_UUID_CMP 109
7.3.4 RBLE_GATT_EVENT_DISC_SVC_INCL_CMP 109
7.3.5 RBLE_GATT_EVENT_DISC_CHAR_ALL_CMP 110
7.3.6 RBLE_GATT_EVENT_DISC_CHAR_ALL_128_CMP 110
7.3.7 RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_CMP 111
7.3.8 RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_128_CMP 111
7.3.9 RBLE_GATT_EVENT_DISC_CHAR_DESC_CMP 112
7.3.10 RBLE_GATT_EVENT_DISC_CHAR_DESC_128_CMP 112
7.3.11 RBLE_GATT_EVENT_READ_CHAR_RESP 112
7.3.12 RBLE_GATT_EVENT_READ_CHAR_LONG_RESP 113
7.3.13 RBLE_GATT_EVENT_READ_CHAR_MULT_RESP 113
7.3.14 RBLE_GATT_EVENT_READ_CHAR_LONG_DESC_RESP 114
7.3.15 RBLE_GATT_EVENT_WRITE_CHAR_RESP 114
7.3.16 RBLE_GATT_EVENT_WRITE_CHAR_RELIABLE_RESP 114
7.3.17 RBLE_GATT_EVENT_CANCEL_WRITE_CHAR_RESP 114

7.3.18 RBLE_GATT_EVENT_HANDLE_VALUE_NOTIF 115
7.3.19 RBLE_GATT_EVENT_HANDLE_VALUE_IND 115
7.3.20 RBLE_GATT_EVENT_HANDLE_VALUE_CFM 115
7.3.21 RBLE_GATT_EVENT_DISCOVERY_CMP 115
7.3.22 RBLE_GATT_EVENT_COMPLETE 116
7.3.23 RBLE_GATT_EVENT_WRITE_CMD_IND 116
7.3.24 RBLE_GATT_EVENT_RESP_TIMEOUT 116
7.3.25 RBLE_GATT_EVENT_SET_PERM_CMP 116
7.3.26 RBLE_GATT_EVENT_SET_DATA_CMP 117
7.3.27 RBLE_GATT_EVENT_NOTIFY_COMP 117
7.3.28 RBLE_GATT_EVENT_COMMAND_DISALLOWED_IND 117

8. Vendor Specific 118

8.1 Definitions 118
8.2 Functions 125
8.2.1 RBLE_VS_Enable 126
8.2.2 RBLE_VS_Test_Rx_Start 126
8.2.3 RBLE_VS_Test_Tx_Start 127
8.2.4 RBLE_VS_Test_End 127
8.2.5 RBLE_VS_Set_Test_Parameter 128
8.2.6 RBLE_VS_Read_Test_RSSI 128
8.2.7 RBLE_VS_Write_Bd_Address 129
8.2.8 RBLE_VS_Set_Tx_Power 130
8.2.9 RBLE_VS_GPIO_Dir 131
8.2.10 RBLE_VS_GPIO_Access 131
8.2.11 RBLE_VS_Flash_Management 132
8.2.12 RBLE_VS_Flash_Access 132
8.2.13 RBLE_VS_Flash_Operation 133
8.2.14 RBLE_VS_Flash_Get_Space 133
8.2.15 RBLE_VS_Flash_Get_EEL_Ver 133
8.2.16 RBLE_VS_Adapt_Enable 134
8.2.17 RBLE_VS_RF_Control 134
8.2.18 RBLE_VS_Set_Params 135
8.3 Events 136
8.3.1 RBLE_VS_EVENT_TEST_RX_START_COMP 137
8.3.2 RBLE_VS_EVENT_TEST_TX_START_COMP 137
8.3.3 RBLE_VS_EVENT_TEST_END_COMP 137
8.3.4 RBLE_VS_EVENT_WR_BD_ADDR_COMP 137

8.3.5 RBLE_VS_EVENT_SET_TEST_PARAM_COMP 137
8.3.6 RBLE_VS_EVENT_READ_TEST_RSSI_COMP 138
8.3.7 RBLE_VS_EVENT_GPIO_DIR_COMP 138
8.3.8 RBLE_VS_EVENT_GPIO_ACCESS_COMP 138
8.3.9 RBLE_VS_EVENT_FLASH_MANAGEMENT_COMP 138
8.3.10 RBLE_VS_EVENT_FLASH_ACCESS_COMP 139
8.3.11 RBLE_VS_EVENT_FLASH_OPERATION_COMP 139
8.3.12 RBLE_VS_EVENT_FLASH_GET_SPACE_COMP 139
8.3.13 RBLE_VS_EVENT_FLASH_GET_EEL_VER_COMP 139
8.3.14 RBLE_VS_EVENT_ADAPT_ENABLE_COMP 140
8.3.15 RBLE_VS_EVENT_ADAPT_STATE_IND 140
8.3.16 RBLE_VS_EVENT_COMMAND_DISALLOWED_IND 140
8.3.17 RBLE_VS_EVENT_SET_TX_POWER_COMP 140
8.3.18 RBLE_VS_EVENT_SET_PARAMS_COMP 140
8.3.19 RBLE_VS_EVENT_RF_CONTROL_COMP 141

9. RWKE 142

9.1 Type Declaration 142
9.2 Kernel Event Management 142
9.2.1 ke_evt_get 143
9.2.2 ke_evt_set 143
9.2.3 ke_evt_clear 143
9.3 Message Communication Management 144
9.3.1 ke_msg_alloc 145
9.3.2 ke_msg_free 145
9.3.3 ke_msg_send 145
9.3.4 ke_msg_send_basic 146
9.3.5 ke_msg_forward 146
9.3.6 ke_msg2param 146
9.3.7 ke_param2msg 146
9.4 Task State Management 147
9.4.1 ke_state_get 147
9.4.2 ke_state_set 147
9.5 Timer Management 148
9.5.1 ke_time 148
9.5.2 ke_timer_set 148
9.5.3 ke_timer_clear 149
9.6 Memory Management 149

9.6.1 ke_malloc 149
9.6.2 ke_free 149
9.7 Exclusive Control 150
9.8 Initialization and Event Loop Execution 150
9.9 RWKE APIs Usable in Interrupt Processing 151

10. Notes 152

Appendix A Message Sequence Chart 153

Appendix B How to Read Definition Tables 172

Appendix C Referenced Documents 174

Appendix D Terminology 175

1. Overview

This manual describes the API (Application Program Interface) of the basic features of the Bluetooth Low Energy protocol stack (BLE software), which is used to develop Bluetooth applications that incorporate Renesas Bluetooth low energy microcontroller RL78/G1D.

For details about the organization and features of BLE software, see the Bluetooth Low Energy Protocol Stack User's Manual.

2. General

2.1 BLE Software and its APIs

BLE software refers to a set of software that includes BLE stacks compliant with the Bluetooth Low Energy protocol (Bluetooth v4.2).

Figure 2-1 shows the BLE software configuration.

BLE software runs in a configuration in which the application is mounted on the RL78/G1D (hereafter referred to as the Embedded configuration) and in a configuration in which the application is mounted on another MCU (hereafter referred to as the Modem configuration). BLE software provides APIs which can use the same application in both configurations.

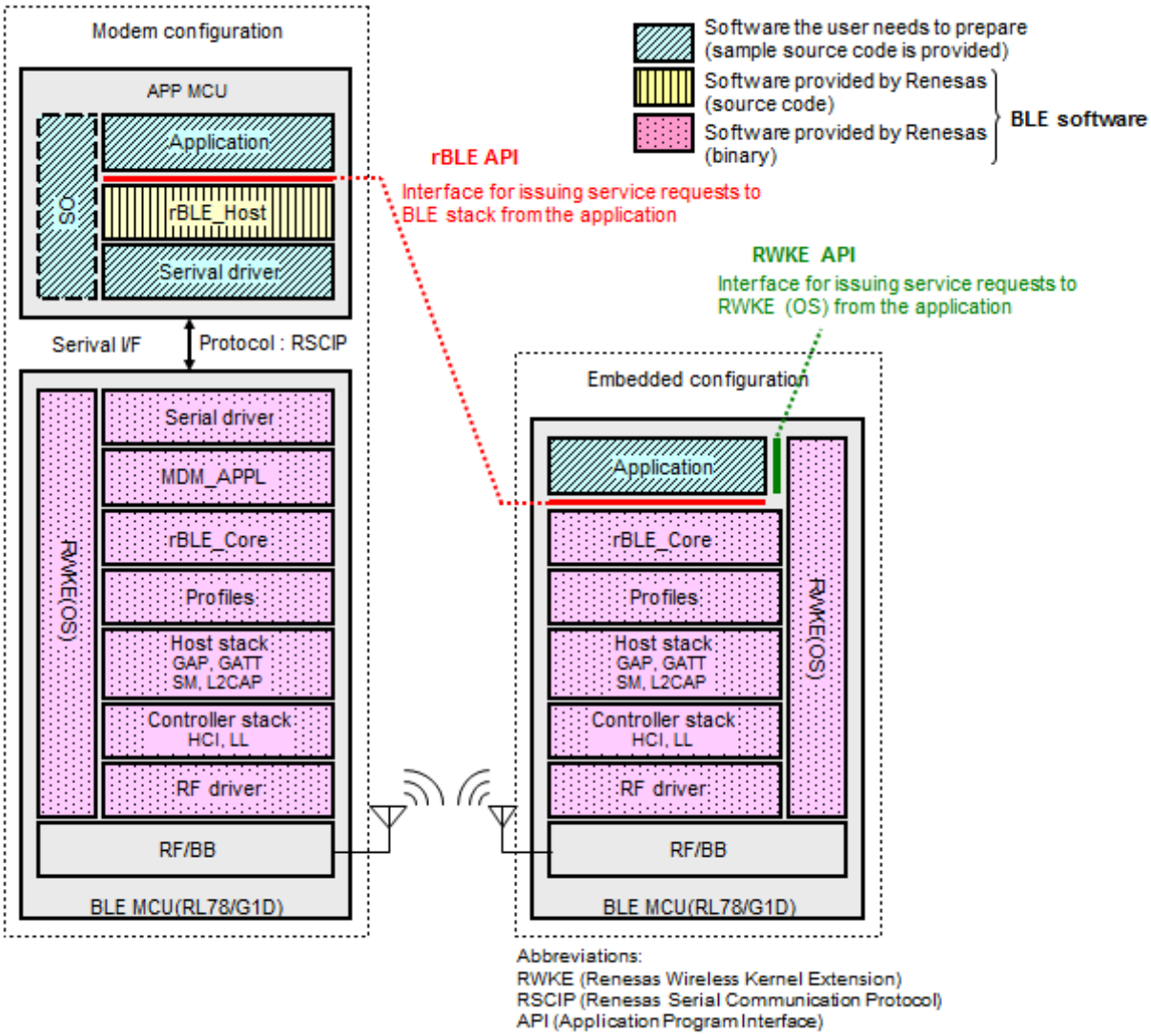

Figure 2-1 BLE Software Configuration

BLE software in the Modem configuration runs on two chips, the APP MCU and the BLE MCU (RL78/G1D). BLE software is configured of an rBLE_Host block that runs on the APP MCU (block in the figure), and software that runs on the BLE MCU (blocks in the figure).

The software to be prepared by the user (blocks in the figure) consists of the APP MCU's application block, UART driver block, and OS block. However, if there is no OS in the APP MCU, software for the OS block does not have to be prepared because the rBLE_Host block does not use resources of the OS.

The application that runs on the APP MCU executes communication between the BLE MCU and BLE services via rBLE_Host. The APP MCU and BLE MCU are physically connected via UART, and communication is executed using RSCIP (Renesas Serial Communication Interface Protocol) under the control of rBLE_Host.

BLE software in the Embedded configuration runs on only a single chip, the BLE MCU (RL78/G1D). The software to be prepared by the user is only the application block and it should be implemented on the BLE MCU.

The APIs of the BLE software described in this document correspond to rBLE APIs and RWKE APIs shown in Figure 2-1. The rBLE APIs are the APIs for issuing service requests to BLE stacks from the application. The RWKE APIs are the APIs for issuing service requests to the RWKE (Renesas Wireless Kernel Extension) which is a simple operating system designed for running BLE software.

2.2 rBLE API

2.2.1 Language

The rBLE APIs use the C language.

2.2.2 rBLE API procedure

This section describes the procedures for the rBLE APIs.

As shown in Figure 2-2 and Figure 2-3, API function calls are used for command requests from the application to rBLE_HOST or rBLE_Core. Further, event notifications from rBLE_HOST or rBLE_Core to the application are called by the callback function for event notification.

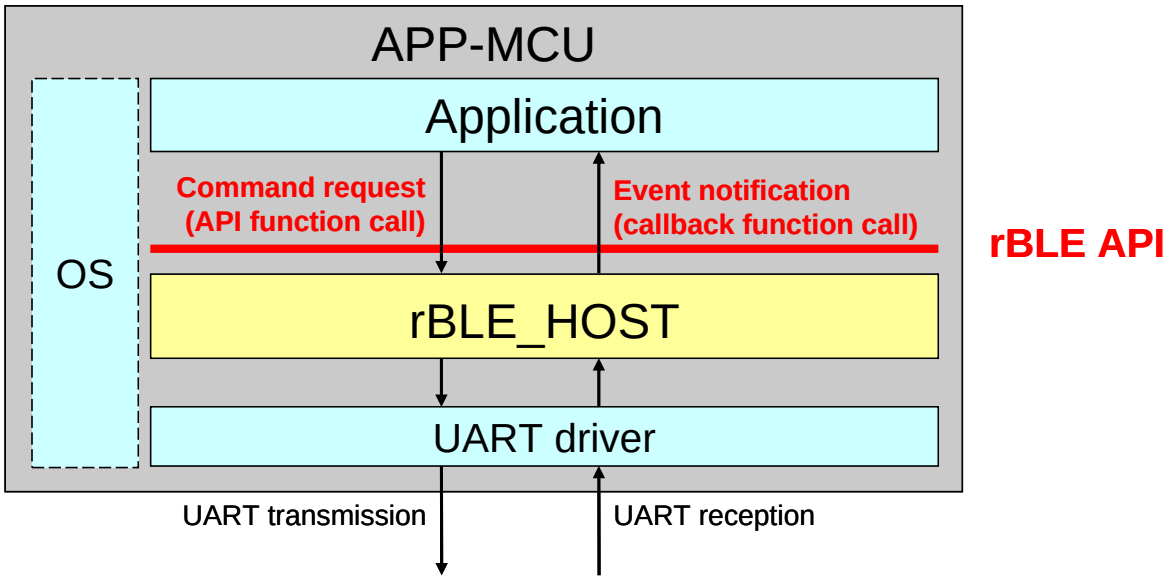

Figure 2-2 rBLE API Procedure (for Modem Configuration)

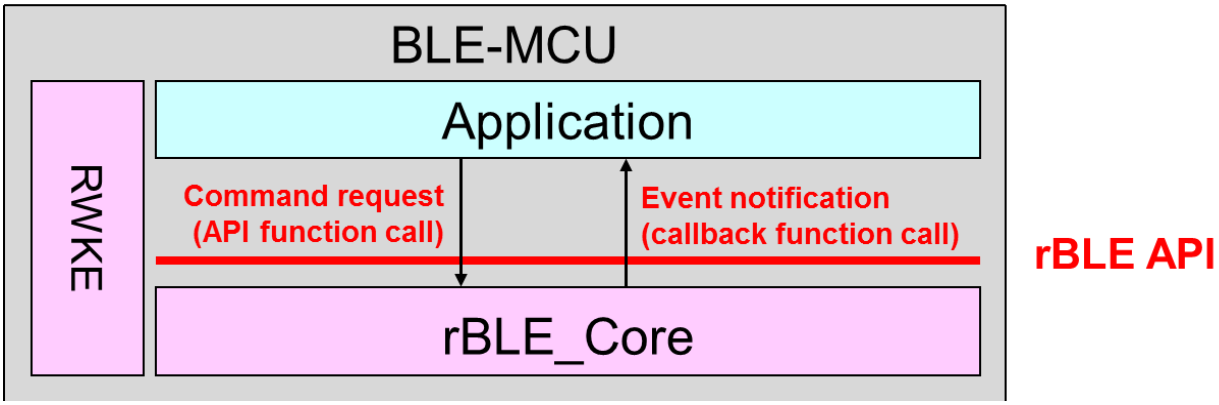

Figure 2-3 rBLE API Procedure (for Embedded Configuration)

The processing results of commands issued from the application are reported as API function return values or events that occur asynchronously as API function calls.

If the return value of the API function that issued a call during a command request is an error, that command request is not processed. Typical causes of API function errors are incorrect parameter values and states in which processing is not possible.

2.2.3 Classification of rBLE API profiles

This section describes the classification of rBLE API profiles.

The rBLE API profiles are classified into the basic profiles of Initialization, Generic Access Profile (GAP), Security Manager (SM), Generic Attribute Profile (GATT), Vendor Specific (VS), and GATT-based Profile. Table 2-1 lists these profiles.

Table 2-1 API Profile Classification

Profile	Abbreviation	Overview
Initialization	INIT	Initializes the BLE software.
Generic Access Profile	GAP	Executes access procedures according to the link management and security requirements for processes such as device discovery and peer device connection and disconnection.
Security Manager	SM	Executes pairing between two devices, communication encryption and data signing to ensure security. Also executes information exchange between devices as needed for the above.
Generic Attribute Profile	GATT	Allows the acquisition of the handles of characteristic values exposed from the client to the server. (Some limitations apply to the features that can be used.)
Vendor Specific	VS	Provides extended functionality such as for Direct Test Mode or Renesas's original Direct Test Mode.
GATT-based Profile	-	See the *Bluetooth Low Energy Protocol Stack User's Manual*.

The detailed explanations of the rBLE APIs in Chapter 4 and later follow the classification shown in Table 2-1.

The abbreviations of the profile names are also frequently used in the C language descriptions. For the meanings of these

profile classification abbreviations, please refer to the list of abbreviations in the table above.

2.2.4 Handling of parameters of rBLE API functions

The API arguments listed in this document include a large number of pointer arguments, but all the memory areas of the addresses indicated by pointers are handled as input-only areas. These memory areas are never overwritten from within API functions.

2.2.5 Registration of callback functions for event notification

This section explains mainly the handling of callback functions for rBLE API event notification.

The callback functions for event notification are to be provided by the customer. This is in order to allow processing of the response operation program for event notification within the callback function for event notification.

Therefore, the callback functions for event notification need to be registered. Moreover, callback functions for event notification must comply with the specifications registered for each of the profiles described in *2.2.3*. Table 2-2 lists the functions for registering the callback function for event notification for each profile.

Table 2-2 List of Callback Registration Functions for Each Profile

Profile	Abbreviation	Callback Registration Function	Remark
Initialization	INIT	RBLE_Init	
Generic Access Profile	GAP	RBLE_GAP_Reset	The same function is used for GAP and SM because the callback function for these two profiles is registered at the same time.
Security Manager	SM		
Generic Attribute Profile	GATT	RBLE_GATT_Enable	
Vendor Specific	VS	RBLE_VS_Enable	
GATT-based Profile	-	RBLE_xxx_###_Enable *	See Enable function of each profile's *Bluetooth Low Energy Protocol Stack API Reference Manual.*

* xxx is short name of profile and ### is name of role.

Because GAP and SM are registered at the same time, callback function registration for these two profiles is performed together by using the RBLE_GAP_Reset function.

Moreover, a callback function must be registered for each role specified in the specifications of each GATT-based Profile. This is because in carrying out communication, only one of the communicating devices plays a role and thus basically only one of the roles is used, so the callback functions are registered separately for each role to save resources.

The Figure 2-4 following shows an example of the program used to register a callback function for event notification.

```
/* Callback function for reporting GAP event */
void GAP_CallBack( RBLE_GAP_EVENT *event )
{
	switch( event->type ) {
		case RBLE_GAP_EVENT_RESET_RESULT:
			/* Event processing */
			break;
		default:
			break;
	}
}

/* Callback function for reporting SM event */
void SM_CallBack( RBLE_SM_EVENT *event )
{
	switch( event->type ) {
		case RBLE_SM_EVENT_SET_CNF:
			/* Event processing */
			break;
		default:
			break;
	}
}

/* GAP reset processing */
void GAP_Reset_Function( void )
{
	RBLE_GAP_Reset( &GAP_CallBack, &SM_CallBack );
}
```

Figure 2-4 Example of Program for Registering a Callback Function for Event Notification

2.2.6 Basic operation of callback functions for event notification

This section explains the basic operation of the callback functions for event notification.
The callback function for event notification specifies an event type for each event that occurs from rBLE_HOST (rBLE_Core for Embedded configuration) and passes data to the application in the data format defined for that event type. The data structure of the event type used for the Target role of FMP is shown in Figure 2-5.

```
/* Data structure for FMP Target role event type */
typedef struct RBLE_FMPT_EVENT_t
{
     RBLE_FMP_EVENT_TYPE                type;                /* Event type */
     uint8_t                            reserved;
     union Event_Fmt_Parameter_u {
          /* RBLE_EVT_FMP_Target_Enable_Comp */
          struct RBLE_FMP_Target_Enable_t{
               RBLE_STATUS              status;
               uint8_t                  reserved;
               uint16_t                 conhdl;
          }target_enable;

          /* RBLE_EVT_FMP_Target_Disable_Comp */
          struct RBLE_FMP_Target_Disable_t{
               RBLE_STATUS              status;
               uint8_t                  reserved;
               uint16_t                 conhdl;
          }target_disable;

          /* RBLE_EVT_FMP_Target_Alert_Ind */
          struct RBLE_FMP_Target_Alert_Ind_t{
               uint16_t                 conhdl;
               uint8_t                  alert_lvl;
               uint8_t                  reserved;
          }target_alert_ind;

          /* RBLE_EVT_FMP_CMD_DISALLOWED_IND */
          struct RBLE_FMP_Target_Command_Disallowed_Ind_t{
               RBLE_STATUS              status;
               uint8_t                  reserved;
               uint16_t                 opcode;
          }cmd_disallowed_ind;
     }param;
}RBLE_FMPT_EVENT;
```

Figure 2-5 Data Structure for FMP Target Role Event Type

The structure shown in Figure 2-5 defines four type members for event type notification and the data format for each event type by using unions (target_enable, target_disable, target_alert_ind, and cmd_err_ind).

The processing executed by the callback function that performed event notification is shown in Figure 2-1.

```
void FMPT_CallBack( RBLE_FMPT_EVENT *event )
{
      switch( event->type){
          case RBLE_FMP_EVENT_TARGET_ENABLE_COMP:
              /* Event processing */
              break;
          case RBLE_FMP_EVENT_TARGET_DISABLE_COMP:
              /* Event processing */
              break;
          case RBLE_FMP_EVENT_TARGET_ALERT_IND:
              /* Event processing */
              break;
          case RBLE_FMP_EVENT_TARGET_COMMAND_ERROR_IND:
              /* Event processing */
              break;
          default:
              break;
     }
}
```

Figure 2-6 Callback Function for Reporting FMP Target Role Event

The callback function shown in Figure 2-6 is programmed so as to allow processing in response to the four events that can occur for the FMP Target role. First, the event type is determined by event->type, and the processing is branched by the switch statement. The application should then be implemented by incorporating the response processing for each event.

2.3 BLE Software State Transitions

This section presents the BLE software state definitions and explains the state transitions.

First, the BLE software has three states, the rBLE_Core state is referred to as "rBLE core mode" and the rBLE_HOST state is referred to as "rBLE mode". These modes are reported by using the mode change notification callback RBLE_INIT_CB.

2.3.1 rBLE_Core state transitions

Table 2-3 shows the state definitions of the rBLE core mode.

Table 2-3 rBLE Core Mode Definitions

rBLE Core Mode	Description
RBLE_MODE_RESET	Means that rBLE_Core has not yet been initialized. The RBLE_MODE_INITIALIZE state is entered by the application calling the RBLE_Init() function.
RBLE_MODE_INITIALIZE	Means that rBLE_Core is being initialized. Upon completion of initialization, the RBLE_MODE_ACTIVE state is entered.
RBLE_MODE_ACTIVE	Means that rBLE_Core is in the active state (operation enabled). Once this state has been entered, no other state is entered unless initialization is performed.

Next, a diagram of the state transitions of the rBLE core mode is shown below.

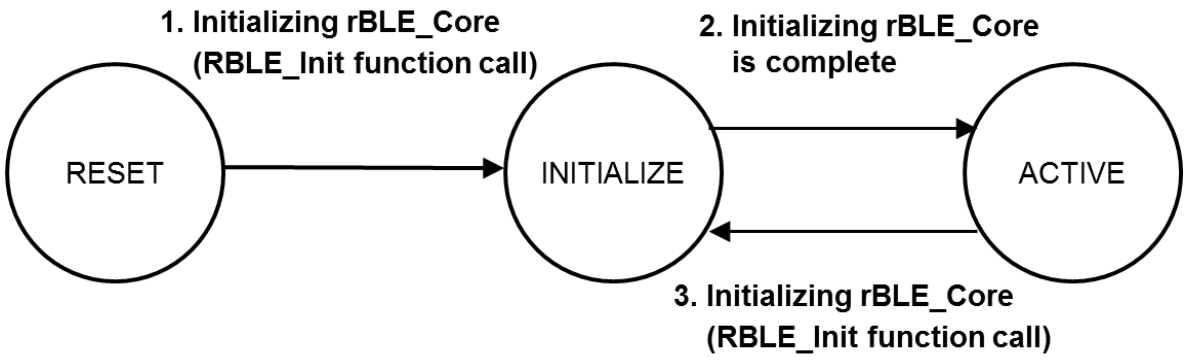

Figure 2-1 State Transitions of rBLE_Core

State transition 1 is the initialization timing of rBLE_Core. This is the timing at which the application (MDM APPL for the Modem configuration) calls the RBLE_Init function when initializing rBLE_Core.

State transition 2 is the initialization completion timing of rBLE_Core. This is the timing at which initialization of rBLE_Core is completed after the RBLE_Init function is called from the application.

State transition 3 is the timing at which rBLE_Core is reset. This is the timing at which the application calls the RBLE_Init function from the active state when resetting rBLE_Core.

2.3.2 rBLE_HOST state transitions

Table 2-4 shows the state definitions of the rBLE mode.

Table 2-4 rBLE Mode Definitions

rBLE Mode	Description
RBLE_MODE_INITIALIZE	Means that rBLE_HOST is being initialized. The RBLE_MODE_INITIALIZE state is entered by the application calling the RBLE_Init() function. Upon completion of initialization, the RBLE_MODE_ACTIVE state is entered.
RBLE_MODE_ACTIVE	Means that rBLE_HOST is in the active state (operation enabled). When the RSCIP connection is reset, the RBLE_MODE_RESET state is entered.
RBLE_MODE_RESET	Means that reset processing is in progress as the result of the RSCIP connection having been reset. Upon completion of the reset processing, the RBLE_MODE_ACTIVE state is entered.

Next, a diagram of the state transitions of the rBLE mode is shown below.

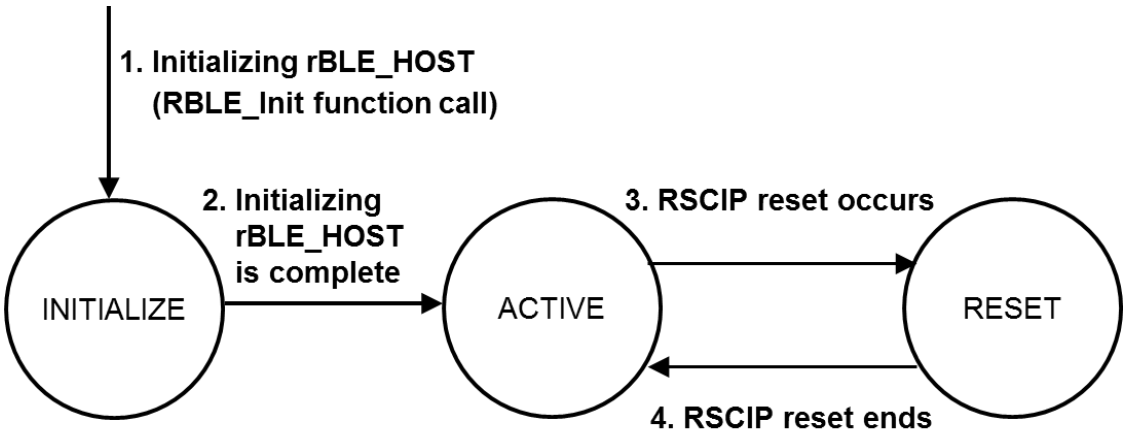

Figure 2-2 State Transitions of rBLE_HOST

State transition 1 is the initialization timing of rBLE_HOST. This is the timing at which the application calls the RBLE_Init function when initializing rBLE_HOST.

State transition 2 is the initialization completion timing of rBLE_HOST. This is the timing at which initialization of rBLE_HOST is completed after the RBLE_Init function is called from the application.

State transition 3 is the timing at which the RSCIP connection is reset. This state transition occurs upon occurrence of a reset upon detection of an abnormal state, or upon occurrence of a communication reset caused by a communication anomaly, either on the APP MCU side or the BLE MCU side.

State transition 4 is the timing at which RSCIP connection reset is completed. This state transition occurs upon reset of the RSCIP connection, and upon completion of this reset processing, the operation enabled state is entered once again.

2.4 BLE Software Initialization Procedure

This section describes the initialization procedure of the BLE software.

The initialization procedure for FMP Target role application is shown in the sequence chart below as an example.

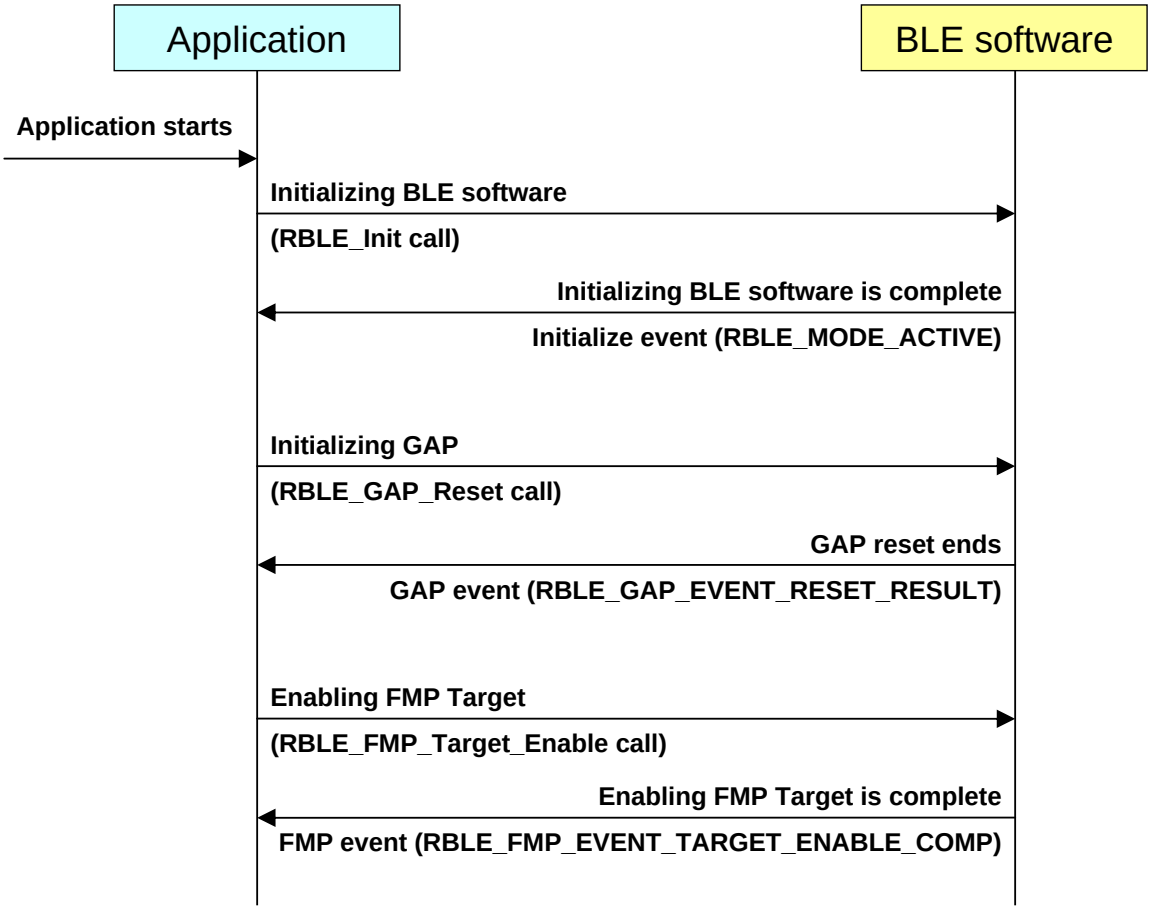

Figure 2-3 Example BLE Software Initialization Procedure

The BLE software is initialized by calling the RBLE_Init function belonging to the Initialize function. Completion of initialization is reported by the rBLE mode change notification callback, and the state that is reported is RBLE_MODE_ACTIVE.

Next, to enable the GAP and SM, the RBLE_GAP_Reset function is called. In response to this call, the RBLE_GAP_EVENT_RESET_RESULT event, which reports the completion of the GAP reset, is sent.

Finally, the profile determined by the application product to be implemented must be enabled (in the figure, this is RBLE_FMP_Target_Enable because the role is FMP Target). The profile becomes usable when the event notifying completion of profile function enabling (in the figure, RBLE_FMP_EVENT_TARGET_ENABLE_COMP) is received.

3. Common Definitions

This section describes the definitions common to the APIs of rBLE.

3.1 Standard Typedef

- Declaration of data type

```
typedef    unsigned char        uint8_t;        Unsigned 8-bit integer
typedef    unsigned short       uint16_t;       Unsigned 16-bit integer
typedef    unsigned long        uint32_t;       Unsigned 32-bit integer
typedef    signed char          int8_t;         Signed 8-bit integer
typedef    signed short         int16_t;        Signed 16-bit integer
typedef    signed long          int32_t;        Signed 32-bit integer
typedef    unsigned char        bool;           Boolean data type
typedef    signed int           int_t;          Signed int
typedef    unsigned int         uint_t;         Unsigned int
typedef    char                 char_t;         String
```

3.2 Generic Definitions

- Constant definitions

```
#define RBLE_BD_ADDR_LEN              0x06     Bluetooth device address
                                               length
#define RBLE_BD_NAME_SIZE             0x41     Bluetooth device name length
#define RBLE_ADV_DATA_LEN             0x1F     Number of Advertising data
                                               bytes
#define RBLE_SCAN_RSP_DATA_LEN        0x1F     Number of Scan Response data
                                               bytes
#define RBLE_KEY_LEN                  0x10     Key length
#define RBLE_LE_FEATS_LEN             0x08     Feature length
#define RBLE_LE_CHNL_MAP_LEN          0x05     Channel map length
#define RBLE_ATTM_MAX_VALUE           0x18     Maximum attribute value
                                               length
#define RBLE_RAND_NB_LEN              0x08     Random number length
#define RBLE_MASTER                   0x00     Master role
#define RBLE_SLAVE                    0x01     Slave role
```

- Declaration of data type for rBLE status

```
typedef uint8_t  RBLE_STATUS;
```

- Declaration of enumerated type for rBLE status

```
enum RBLE_STATUS_enum {
    RBLE_OK                              = 0x00,   Normal operation
    RBLE_UNKNOWN_HCI_COMMAND             = 0x01,   Unknown command received
    RBLE_UNKNOWN_CONNECTION_ID           = 0x02,   Unknown connection ID specified
    RBLE_HARDWARE_FAILURE                = 0x03,   Hardware error occurred
    RBLE_PAGE_TIMEOUT                    = 0x04,   Page timeout occurred
    RBLE_AUTH_FAILURE                    = 0x05,   Authentication failed
    RBLE_PIN_MISSING                     = 0x06,   PIN code is missing
    RBLE_MEMORY_CAPA_EXCEED              = 0x07,   Memory capacity exceeded
    RBLE_CON_TIMEOUT                     = 0x08,   Connection timeout occurred
    RBLE_CON_LIMIT_EXCEED                = 0x09,   Number of connected devices has
                                                   reached the limit
    RBLE_COMMAND_DISALLOWED              = 0x0C,   The command is not permitted.
    RBLE_CONN_REJ_LIMITED_RESOURCES      = 0x0D,   Connection rejected due to
                                                   resource restriction
    RBLE_CONN_REJ_SECURITY_REASONS       = 0x0E,   Connection rejected due to
                                                   security reasons
    RBLE_CONN_REJ_UNACCEPTABLE_BDADDR    = 0x0F,   Connection rejected due to
                                                   unacceptable BD address
    RBLE_CONN_ACCEPT_TIMEOUT_EXCEED      = 0x10,   Connection acceptance timeout
                                                   occurred
    RBLE_UNSUPPORTED                     = 0x11,   Unsupported
    RBLE_INVALID_HCI_PARAM               = 0x12,   Invalid parameter specified
    RBLE_REMOTE_USER_TERM_CON            = 0x13,   Disconnected by remote user
    RBLE_REMOTE_DEV_TERM_LOW_RESOURCES   = 0x14,   Disconnected due to insufficient
                                                   resources
    RBLE_REMOTE_DEV_POWER_OFF            = 0x15,   Remote device power is off
    RBLE_CON_TERM_BY_LOCAL_HOST          = 0x16,   Disconnected by local host
    RBLE_REPEATED_ATTEMPTS               = 0x17,   Number of retries for pairing
                                                   authentication has reached the limit
    RBLE_PAIRING_NOT_ALLOWED             = 0x18,   Pairing is not permitted.
    RBLE_UNSUPPORTED_REMOTE_FEATURE      = 0x1A,   Unsupported remote device
    RBLE_UNSPECIFIED_ERROR               = 0x1F,   Unspecified error
    RBLE_LMP_RSP_TIMEOUT                 = 0x22,   LMP/LL response timed out
    RBLE_ENC_MODE_NOT_ACCEPT             = 0x25,   Requested encryption mode is not
                                                   acceptable
    RBLE_LINK_KEY_CANT_CHANGE            = 0x26,   Link key cannot be changed
    RBLE_INSTANT_PASSED                  = 0x28,   Execution time has elapsed
    RBLE_PAIRING_WITH_UNIT_KEY_NOT_SUP   = 0x29,   Pairing using a UNIT key is not
                                                   supported
    RBLE_DIFF_TRANSACTION_COLLISION      = 0x2A,   Multiple transactions collided
    RBLE_CHANNEL_CLASS_NOT_SUP           = 0x2E,   Channel assessment mode is not
                                                   supported
    RBLE_INSUFFICIENT_SECURITY           = 0x2F,   Insufficient security error
    RBLE_PARAM_OUT_OF_MAND_RANGE         = 0x30,   Parameter is out of mandatory range
    RBLE_SP_NOT_SUPPORTED_HOST           = 0x37,   The host does not support SSP
    RBLE_HOST_BUSY_PAIRING               = 0x38,   Pairing busy because host is paired
                                                   with another device
```

```
RBLE_CONTROLLER_BUSY                    = 0x3A,    Unexecutable because other
                                                   processing is in progress
RBLE_UNACCEPTABLE_CONN_INT              = 0x3B,    Specified connection parameter is
                                                   unacceptable
RBLE_DIRECT_ADV_TO                      = 0x3C,    Directed Advertising timed out
RBLE_TERMINATED_MIC_FAILURE             = 0x3D,    Disconnected due to incomplete
                                                   received packet message
RBLE_CONN_FAILED_TO_BE_ES               = 0x3E,    Connection establishment failed

RBLE_GAP_INVALID_PARAM_ERR              = 0x40,    GAP invalid parameter error
RBLE_GAP_AUTO_EST_ERR,                             Automatic GAP connection error
RBLE_GAP_SELECT_EST_ERR,                           GAP selective connection error
RBLE_GAP_SET_RECON_ADDR_ERR,                       GAP reconnection address setup
                                                   error
RBLE_GAP_SET_PRIVACY_FEAT_ERR,                     GAP privacy feature setup error

RBLE_GATT_INVALID_PARAM_ERR             = 0x50,    GATT invalid parameter error
RBLE_GATT_INDICATE_NOT_ALLOWED,                    GATT indication disallowed
RBLE_GATT_NOTIFY_NOT_ALLOWED,                      GATT notification disallowed
RBLE_GATT_INVALID_TYPE_IN_SVC_SEARCH,              GATT invalid service search type error
RBLE_GATT_ATTRIBUTE_CLIENT_MISSING,                GATT ATT Client missing
RBLE_GATT_ATTRIBUTE_SERVER_MISSING,                GATT ATT Server missing
RBLE_GATT_RELIABLE_WRITE_ERR,                      GATT reliable write error
RBLE_GATT_BUFF_OVER_ERR,                           GATT buffer over error

RBLE_ATT_INVALID_PARAM_ERR              = 0x60,    Invalid ATT parameter error

RBLE_SM_INVALID_PARAM_ERR               = 0x70,    Invalid SM parameter error
RBLE_SM_PAIR_ERR_PASSKEY_ENTRY_FAILED,             Invalid passkey entry
RBLE_SM_PAIR_ERR_OOB_NOT_AVAILABLE,                OOB data is not available
RBLE_SM_PAIR_ERR_AUTH_REQUIREMENTS,                Authentication requirements are
                                                   not met
RBLE_SM_PAIR_ERR_CFM_VAL_FAILED,                   Confirm value mismatch
RBLE_SM_PAIR_ERR_PAIRING_NOT_SUPPORTED,            Pairing not supported
RBLE_SM_PAIR_ERR_ENCRYPTION_KEY_SIZE,              Invalid encryption key size
RBLE_SM_PAIR_ERR_CMD_NOT_SUPPORTED,                Unsupported SMP command is
                                                   received
RBLE_SM_PAIR_ERR_UNSPECIFIED_REASON,               Pairing failed due to an unknown
                                                   error
RBLE_SM_PAIR_ERR_REPEATED_ATTEMPTS,                Number of pairing attempts reached
                                                   the upper limit in a short time
RBLE_SM_PAIR_ERR_INVALID_PARAMS,                   Invalid parameter

RBLE_L2C_INVALID_PARAM_ERR              = 0x80,    Invalid L2CAP parameter error

RBLE_ERR,                               = 0xF0,    Error
RBLE_TRANS_ERR                          = 0xF1,    Communication error
RBLE_STATUS_ERROR                       = 0xF2,    Status error
```

```
    RBLE_PARAM_ERR                        = 0xF3,    Parameter error
    RBLE_BUSY                             = 0xF4,    Busy error occurred
    RBLE_SHORTAGE_OF_RESOURCE             = 0xF5,    Insufficient resources
    RBLE_EXIT                             = 0xF6,    Exit
    RBLE_VERSION_FAIL                     = 0xF7,    library combination error
    RBLE_TEST_VERSION                     = 0xF8     BLE software is test version
};
```

Note: The profile-specific statuses are described in each profile edition of the API Reference Manual.

- Declaration of enumerated type for ATT error code

```
enum RBLE_ATT_ERR_CODE_enum {
    RBLE_ATT_ERR_NO_ERROR                 = 0x00,    Success
    RBLE_ATT_ERR_INVALID_HANDLE,                     Invalid handle
    RBLE_ATT_ERR_READ_NOT_PERMITTED,                 Reading is not permitted.
    RBLE_ATT_ERR_WRITE_NOT_PERMITTED,                Writing is not permitted.
    RBLE_ATT_ERR_INVALID_PDU,                        Invalid PDU
    RBLE_ATT_ERR_INSUFF_AUTHEN,                      Authentication required for the
                                                     request
    RBLE_ATT_ERR_REQUEST_NOT_SUPPORTED,              Unsupported request
    RBLE_ATT_ERR_INVALID_OFFSET,                     Invalid offset
    RBLE_ATT_ERR_INSUFF_AUTHOR,                      Authorization required for the
                                                     request
    RBLE_ATT_ERR_PREPARE_QUEUE_FULL,                 The queue is full
    RBLE_ATT_ERR_ATTRIBUTE_NOT_FOUND,                The attribute could not be found
    RBLE_ATT_ERR_ATTRIBUTE_NOT_LONG,                 The attribute is not long enough
    RBLE_ATT_ERR_INSUFF_ENC_KEY_SIZE,                Insufficient encryption key size
    RBLE_ATT_ERR_INVALID_ATTRIBUTE_VAL_LEN,          Invalid attribute value size
    RBLE_ATT_ERR_UNLIKELY_ERR,                       Unexpected error
    RBLE_ATT_ERR_INSUFF_ENC,                         Encryption required for the request
    RBLE_ATT_UNSUPP_GRP_TYPE,                        The specified group type is not
                                                     supported
    RBLE_ATT_INSUFF_RESOURCE,                        Insufficient resources
    RBLE_ATT_ERR_APP_ERROR                = 0x80,    Application error
    RBLE_ATT_ERR_IMPROPERLY_CONFIGURED= 0xFD,        Configuration Descriptor
                                                     Improperly Configured
    RBLE_ATT_ERR_ALREADY_IN_PROGRESS      = 0xFE,    Procedure Already in Progress
    RBLE_ATT_ERR_OUT_OF_RANGE             = 0xFF,    Out of Range
};
```

- Declaration of Bluetooth device name structure

```
typedef struct RBLE_BD_NAME_t {
    uint8_t              namelen;                    Device name length
    uint8_t              name[RBLE_BD_NAME_SIZE];    Bluetooth device name
} RBLE_BD_NAME;
```

- Declaration of Bluetooth device address structure

```
typedef struct RBLE_BD_ADDR_t {
```

```
    uint8_t                addr[RBLE_BD_ADDR_LEN];         Bluetooth device address
} RBLE_BD_ADDR;
```

- Declaration of Bluetooth channel map structure

```
typedef struct RBLE_LE_CHNL_MAP_t{
    uint8_t  map[RBLE_LE_CHNL_MAP_LEN];      Channel map (5 bytes = 40 ch/8 bits)
                                             Set each bit to 0 (do not use) or 1 (use)
} RBLE_LE_CHNL_MAP;
```

3.3 GATT Definitions

- GATT attribute type UUID definitions

```
#define RBLE_DECL_PRIMARY_SERVICE        0x2800u     Primary Service Declaration
#define RBLE_DECL_SECONDARY_SERVICE      0x2801u     Secondary Service Declaration
#define RBLE_DECL_INCLUDE                0x2802u     Include Declaration
#define RBLE_DECL_CHARACTERISTIC         0x2803u     Characteristic Declaration
```

- Characteristic descriptor UUID definitions

```
#define RBLE_DESC_CHAR_EXT_PROPERTIES       0x2900u   Characteristic Extended
                                                      Properties
#define RBLE_DESC_CHAR_USER_DESCRIPTION     0x2901u   Characteristic User Description
#define RBLE_DESC_CLIENT_CHAR_CONF          0x2902u   Client Characteristic
                                                      Configuration
#define RBLE_DESC_SERVER_CHAR_CONF          0x2903u   Server Characteristic
                                                      Configuration
#define RBLE_DESC_CHAR_PRESENTATION_FMT     0x2904u   Characteristic Presentation
                                                      Format
#define RBLE_DESC_CHAR_AGGREGATE_FMT        0x2905u   Characteristic Aggregate Format
#define RBLE_DESC_VALID_RANGE               0x2906u   Valid Range
#define RBLE_DESC_EXT_REPORT_REFERENCE      0x2907u   External Report Reference
#define RBLE_DESC_REPORT_REFERENCE          0x2908u   Report Reference
```

- Characteristic UUID definitions

```
#define RBLE_CHAR_GAP_DEVICE_NAME            0x2A00u    Device Name
#define RBLE_CHAR_GAP_APPEARANCE             0x2A01u    Appearance
#define RBLE_CHAR_GAP_PH_PRIV_FLAG           0x2A02u    Peripheral Privacy Flag
#define RBLE_CHAR_GAP_RECONN_ADDRESS         0x2A03u    Reconnection Address
#define RBLE_CHAR_GAP_PH_PREF_CONN_PARAM     0x2A04u    Peripheral Preferred
                                                        Connection Parameters
#define RBLE_CHAR_GATT_SERVICE_CHANGED       0x2A05u    Service Changed
#define RBLE_CHAR_ALERT_LEVEL                0x2A06u    Alert Level
#define RBLE_CHAR_TX_POWER_LEVEL             0x2A07u    Tx Power Level
#define RBLE_CHAR_DATE_TIME                  0x2A08u    Date Time
#define RBLE_CHAR_DAY_OF_WEEK                0x2A09u    Day of Week
#define RBLE_CHAR_DAY_DATE_TIME              0x2A0Au    Day Date Time
#define RBLE_CHAR_EXACT_TIME_256             0x2A0Cu    Exact Time 256
#define RBLE_CHAR_DST_OFFSET                 0x2A0Du    DST Offset
#define RBLE_CHAR_TIME_ZONE                  0x2A0Eu    Time Zone
#define RBLE_CHAR_LOCAL_TIME_INFO            0x2A0Fu    Local Time Information
#define RBLE_CHAR_TIME_WITH_DST              0x2A11u    Time with DST
#define RBLE_CHAR_TIME_ACCURACY              0x2A12u    Time Accuracy
#define RBLE_CHAR_TIME_SOURCE                0x2A13u    Time Source
#define RBLE_CHAR_REF_TIME_INFO              0x2A14u    Reference Time Information
#define RBLE_CHAR_TIME_UPDATE_CTRL_POINT     0x2A16u    Time Update Control Point
#define RBLE_CHAR_TIME_UPDATE_STATE          0x2A17u    Time Update State
```

```
#define RBLE_CHAR_GLUCOSE_MEASUREMENT         0x2A18u    Glucose Measurement
#define RBLE_CHAR_BATTERY_LEVEL               0x2A19u    Battery Level
#define RBLE_CHAR_TEMPERATURE_MEAS            0x2A1Cu    Temperature Measurement
#define RBLE_CHAR_TEMPERATURE_TYPE            0x2A1Du    Temperature Type
#define RBLE_CHAR_INTERMEDIATE_TEMP           0x2A1Eu    Intermediate Temperature
#define RBLE_CHAR_MEAS_INTERVAL               0x2A21u    Measurement Interval
#define RBLE_CHAR_BOOT_KB_INPUT_REPORT        0x2A22u    Boot Keyboard Input Report
#define RBLE_CHAR_SYSTEM_ID                   0x2A23u    System ID
#define RBLE_CHAR_MODEL_NUMBER_STRING         0x2A24u    Model Number String
#define RBLE_CHAR_SERIAL_NUMBER_STRING        0x2A25u    Serial Number String
#define RBLE_CHAR_FW_REVISION_STRING          0x2A26u    Firmware Revision String
#define RBLE_CHAR_HW_REVISION_STRING          0x2A27u    Hardware Revision String
#define RBLE_CHAR_SW_REVISION_STRING          0x2A28u    Software Revision String
#define RBLE_CHAR_MANUF_NAME_STRING           0x2A29u    Manufacturer Name String
#define RBLE_CHAR_IEEE_CERTIF                 0x2A2Au    IEEE 11073-20601 Regulatory
                                                         Certification Data List
#define RBLE_CHAR_CURRENT_TIME                0x2A2Bu    Current Time
#define RBLE_CHAR_SCAN_REFRESH                0x2A31u    Scan Refresh
#define RBLE_CHAR_BOOT_KB_OUTPUT_REPORT       0x2A32u    Boot Keyboard Output Report
#define RBLE_CHAR_BOOT_MOUSE_INPUT_REPORT     0x2A33u    Boot Mouse Input Report
#define RBLE_CHAR_GLUCOSE_MEAS_CONTEXT        0x2A34u    Glucose Measurement Context
#define RBLE_CHAR_BLOOD_PRESSURE_MEAS         0x2A35u    Blood Pressure Measurement
#define RBLE_CHAR_INTERMEDIATE_BLOOD_PRESS    0x2A36u    Intermediate Cuff Pressure
#define RBLE_CHAR_HEART_RATE_MEAS             0x2A37u    Heart Rate Measurement
#define RBLE_CHAR_BODY_SENSOR_LOCATION        0x2A38u    Body Sensor Location
#define RBLE_CHAR_HEART_RATE_CTRL_POINT       0x2A39u    Heart Rate Control Point
#define RBLE_CHAR_ALERT_STATUS                0x2A3Fu    Alert Status
#define RBLE_CHAR_RINGER_CTRL_POINT           0x2A40u    Ringer Control Point
#define RBLE_CHAR_RINGER_SETTING              0x2A41u    Ringer Setting
#define RBLE_CHAR_AL_CATEGORY_ID_BIT_MASK     0x2A42u    Alert Category ID Bit Mask
#define RBLE_CHAR_AL_CATEGORY_ID              0x2A43u    Alert Category ID
#define RBLE_CHAR_AL_NOTIF_CTRL_POINT         0x2A44u    Alert Notification Control
                                                         Point
#define RBLE_CHAR_UNREAD_ALERT_STATUS         0x2A45u    Unread Alert Status
#define RBLE_CHAR_NEW_ALERT                   0x2A46u    New Alert
#define RBLE_CHAR_SUPP_NEW_AL_CATEGORY        0x2A47u    Supported New Alert Category
#define RBLE_CHAR_SUPP_UNREAD_AL_CATEGORY     0x2A48u    Supported Unread Alert
                                                         Category
#define RBLE_CHAR_BLOOD_PRESSURE_FEAT         0x2A49u    Blood Pressure Feature
#define RBLE_CHAR_HID_INFO                    0x2A4Au    HID Information
#define RBLE_CHAR_REPORT_MAP                  0x2A4Bu    Report Map
#define RBLE_CHAR_HID_CTRL_POINT              0x2A4Cu    HID Control Point
#define RBLE_CHAR_REPORT                      0x2A4Du    Report
#define RBLE_CHAR_PROTOCOL_MODE               0x2A4Eu    Protocol Mode
#define RBLE_CHAR_SCAN_INTERVAL_WINDOW        0x2A4Fu    Scan Interval Window
#define RBLE_CHAR_PNP_ID                      0x2A50u    PnP ID
#define RBLE_CHAR_GLUCOSE_FEATURE             0x2A51u    Glucose Feature
```

```
#define RBLE_CHAR_RECORD_ACCESS_CTRL_POINT    0x2A52u    Record Access Control Point
#define RBLE_CHAR_SC_CNTL_POINT               0x2A53u    RSC Measurement
#define RBLE_CHAR_CSC_MEAS                    0x2A54u    RSC Feature
#define RBLE_CHAR_SC_CNTL_POINT               0x2A55u    SC Control Point
#define RBLE_CHAR_CSC_MEAS                    0x2A5Bu    CSC Measurement
#define RBLE_CHAR_CSC_FEATURE                 0x2A5Cu    CSC Feature
#define RBLE_CHAR_SENSOR_LOCATION             0x2A5Du    Sensor Location
#define RBLE_CHAR_CYCLING_POWER_MEAS          0x2A63u    Cycling Power Measurements
#define RBLE_CHAR_CYCLING_POWER_VECTOR        0x2A64u    Cycling Power Vector
#define RBLE_CHAR_CYCLING_POWER_FEATURE       0x2A65u    Cycling Power Feature
#define RBLE_CHAR_CYCLING_POWER_CNTL_POINT    0x2A66u    Cycling Power Control Point
#define RBLE_CHAR_LOCATION_SPEED              0x2A67u    Location and Speed
#define RBLE_CHAR_NAVIGATION                  0x2A68u    Navigation
#define RBLE_CHAR_POSITION_QUALITY            0x2A69u    Position Quality
#define RBLE_CHAR_LN_FEATURE                  0x2A6Au    LN Feature
#define RBLE_CHAR_LN_CNTL_POINT               0x2A6Bu    LN Control Point
```

- Service UUID definitions

```
#define RBLE_SVC_GENERIC_ACCESS               0x1800u    Generic Access
#define RBLE_SVC_GENERIC_ATTRIBUTE            0x1801u    Generic Attribute
#define RBLE_SVC_IMMEDIATE_ALERT              0x1802u    Immediate Alert
#define RBLE_SVC_LINK_LOSS                    0x1803u    Link Loss
#define RBLE_SVC_TX_POWER                     0x1804u    Tx Power
#define RBLE_SVC_CURRENT_TIME                 0x1805u    Current Time Service
#define RBLE_SVC_REFERENCE_TIME_UPDATE        0x1806u    Reference Time Update Service
#define RBLE_SVC_NEXT_DST_CHANGE              0x1807u    Next DST Change Service
#define RBLE_SVC_GLUCOSE                      0x1808u    Glucose
#define RBLE_SVC_HEALTH_THERMOMETER           0x1809u    Health Thermometer
#define RBLE_SVC_DEVICE_INFORMATION           0x180Au    Device Information
#define RBLE_SVC_HEART_RATE                   0x180Du    Heart Rate
#define RBLE_SVC_PHONE_ALERT_STATUS           0x180Eu    Phone Alert Status Service
#define RBLE_SVC_BATTERY_SERVICE              0x180Fu    Battery Service
#define RBLE_SVC_BLOOD_PRESSURE               0x1810u    Blood Pressure
#define RBLE_SVC_ALERT_NOTIFICATION           0x1811u    Alert Notification Service
#define RBLE_SVC_HUMAN_INTERFACE_DEVICE       0x1812u    Human Interface Device
#define RBLE_SVC_SCAN_PARAMETERS              0x1813u    Scan Parameters
#define RBLE_SVC_RUNNING_SPEED                0x1814u    Running Speed and Cadence
#define RBLE_SVC_CYCLING_SPEED                0x1816u    Cycling Speed and Cadence
#define RBLE_SVC_CYCLING_POWER                0x1818u    Cycling Power
#define RBLE_SVC_LOCATION_NAVIGATION          0x1819u    Location and Navigation
```

4. Initialization

This section describes the APIs for rBLE initialization.

4.1 Definitions

This section describes the definitions used by the APIs for rBLE initialization.

- Declaration of data type for callback function that reports an rBLE mode change

```
typedef void ( *RBLE_INIT_CB )( RBLE_MODE mode )
```

- Declaration of enumerated type for rBLE mode

```
enum RBLE_MODE_enum {
   RBLE_MODE_INITIALIZE          = 0,              rBLE is being initialized
   RBLE_MODE_ACTIVE,                               rBLE is active (operation
                                                   enabled)
   RBLE_MODE_RESET,                                rBLE is being reset
   RBLE_MODE_ERROR                                 Error occurred in rBLE
                                                   initialization processing
};
```

- Declaration of data type for rBLE mode

```
typedef uint8_t  RBLE_MODE;
```

4.2 Functions

The following table shows the API functions defined for initialization of rBLE and the following sections describe the API functions in detail.

Table 4-1 API Functions Used by the rBLE Initialization

RBLE_Init	Initializes rBLE.

4.2.1 RBLE_Init

RBLE_STATUS RBLE_Init (RBLE_INIT_CB call_back)		
This function initializes the BLE software. This function must be called before using any of the rBLE profiles. The reporting of a change to the active (operation enabled) state by the rBLE mode change notification callback RBLE_INIT_CB indicates successful completion of initialization.		
Parameters:		
	call_back	Specifies the callback function that reports the rBLE software mode change.
Return:		
	RBLE_OK	Success
	RBLE_ERR	Error occurred in initialization processing
	RBLE_PARAM_ERR	Invalid parameter

4.3 Events

The following table shows the events defined for initialization of rBLE and the following sections describe the events in detail.

Table 4-2 Events Defined for rBLE Initialization

RBLE_INIT_EVENT_MODE_CHANGE	Reports the rBLE mode change.

4.3.1 RBLE_INIT_EVENT_MODE_CHANGE

void (*RBLE_INIT_CB)(RBLE_MODE mode)			
This is a callback function that reports the rBLE mode change.			
Parameters:			
	mode	RBLE_MODE_INITIALIZE	Means that the BLE software is being initialized. The RBLE_MODE_INITIALIZE state is entered by the application calling the RBLE_Init() function. Upon completion of initialization, the RBLE_MODE_ACTIVE state is entered.
		RBLE_MODE_ACTIVE	Means that the BLE software is in the active state (operation enabled).
		RBLE_MODE_RESET	Means that the RSCIP connection has been reset, and the corresponding reset processing is in progress. Upon completion of the reset processing, the BLE software enters the RBLE_MODE_ACTIVE state.
		RBLE_MODE_ERROR	Indicates that an error occurred during rBLE initialization processing.
Return:			
	none		

5. Generic Access Profile

This section describes the APIs for general processing such as discovery, connection, and bonding of Bluetooth devices.

5.1 Definitions

This section describes the definitions used by the APIs for general processing such as discovery, connection, and bonding of Bluetooth devices.

- Declaration of enumerated type for GAP event types

```
enum RBLE_GAP_EVENT_TYPE_enum {
    RBLE_GAP_EVENT_RESET_RESULT = 1,            Reset completion event
                                                (Parameter: reset_result)
    RBLE_GAP_EVENT_SET_NAME_COMP,               Device name setup completion event
                                                (Parameter: status)
    RBLE_GAP_EVENT_OBSERVATION_ENABLE_COMP,     Observation enable event
                                                (Parameter: status)
    RBLE_GAP_EVENT_OBSERVATION_DISABLE_COMP,    Observation disable event
                                                (Parameter: status)
    RBLE_GAP_EVENT_BROADCAST_ENABLE_COMP,       Broadcast enable event
                                                (Parameter: status)
    RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP,      Broadcast disable event
                                                (Parameter: status)
    RBLE_GAP_EVENT_SET_BONDING_MODE_COMP, B     onding mode setup event
                                                (Parameter: status)
    RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP,   Security mode setup event
                                                (Parameter: set_sec_req)
    RBLE_GAP_EVENT_GET_DEVICE_INFO_COMP,        Device information acquisition
                                                completion event
                                                (Parameter: get_dev_ver)
    RBLE_GAP_EVENT_GET_WHITE_LIST_SIZE_COMP,    Local device White List size read
                                                completion event
                                                (Parameter: get_wlst_size)
    RBLE_GAP_EVENT_ADD_TO_WHITE_LIST_COMP,      White List device add completion
                                                event
                                                (Parameter: status)
    RBLE_GAP_EVENT_DEL_FROM_WHITE_LIST_COMP,    White List device delete completion
                                                event
                                                (Parameter: status)
    RBLE_GAP_EVENT_GET_REMOTE_DEVICE_NAME_COMP, Remote device name acquisition
                                                completion event
                                                (Parameter: get_remote_dev_name)
```

RBLE_GAP_EVENT_GET_REMOTE_DEVICE_INFO_COMP,	Remote device information acquisition completion event (Parameter: get_remote_dev_info)
RBLE_GAP_EVENT_DEVICE_SEARCH_COMP,	Device search command completion event (Parameter: status)
RBLE_GAP_EVENT_DEVICE_SEARCH_RESULT_IND,	Device search result notification event (Parameter: dev_search_result)
RBLE_GAP_EVENT_RPA_RESOLVED,	Resolvable Private Address resolution completion event (Parameter: rpa_resolved)
RBLE_GAP_EVENT_SET_RANDOM_ADDRESS_COMP,	Random address setup command completion event (Parameter: set_rand_adr)
RBLE_GAP_EVENT_SET_PRIVACY_FEATURE_COMP,	Privacy feature setup completion event (Parameter: status)
RBLE_GAP_EVENT_CONNECTION_COMP,	LE link connection completion event (Parameter: conn_comp)
RBLE_GAP_EVENT_CONNECTION_CANCEL_COMP,	LE link connection cancel completion event (Parameter: status)
RBLE_GAP_EVENT_DISCONNECT_COMP,	LE link disconnection completion event (Parameter: disconnect)
RBLE_GAP_EVENT_ADVERTISING_REPORT_IND,	Advertising report and data report notification event (Parameter: adv_report)
RBLE_GAP_EVENT_BONDING_COMP,	Bonding completion event (Parameter: bonding_comp)
RBLE_GAP_EVENT_BONDING_REQ_IND,	Peer device bonding request notification event (Parameter: bonding_req)
RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_REQ_IND,	Connection parameter change request notification event (Parameter: chg_connect_param_req)

```
    RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_COMP,
                                                Connection parameter change
                                                completion event
                                                (Parameter: chg_connect_param)
    RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_RESPONSE,
                                                Connection parameter change request
                                                response notification event
                                                (Parameter: chg_connect_param_resp)
    RBLE_GAP_EVENT_CHANNEL_MAP_REQ_COMP,        Channel map setup/acquisition
                                                completion event
                                                (Parameter: channel_map_req_cmp)
    RBLE_GAP_EVENT_READ_RSSI_COMP,              RSSI acquisition completion event
                                                (Parameter: read_rssi)
    RBLE_GAP_EVENT_WR_CHAR_IND,                 GAP characteristics write indication
                                                event
                                                (Parameter: wr_char)
    RBLE_GAP_EVENT_COMMAND_DISALLOWED_IND       GAP command disallowed notification
                                                event
                                                (Parameter: cmd_disallowed_ind)
};
```

- Declaration of data type for GAP event types

```
typedef uint8_t  RBLE_GAP_EVENT_TYPE;
```

- Declaration of data type for GAP event callback function

```
typedef void ( *RBLE_GAP_EVENT_HANDLER )( RBLE_GAP_EVENT *event );
```

- Declaration of enumerated type for GAP Observation and connection establishment procedure

```
enum RBLE_GAP_OBSERV_MODE_enum {
    RBLE_GAP_OBSERVER               = 0x0800,         Observation procedure
    RBLE_GAP_AUTO_CONNECT           = 0x1000,         Auto connection procedure
    RBLE_GAP_SELECT_CONNECT         = 0x2000          Selective connection procedure
};
```

- Declaration of enumerated type for GAP discovery modes

```
enum RBLE_GAP_DISCOVERABLE_MODE_enum {
    RBLE_GAP_NON_DISCOVERABLE       = 0x0001,         Non-discoverable mode
    RBLE_GAP_GEN_DISCOVERABLE       = 0x0002,         General discoverable mode
    RBLE_GAP_LIM_DISCOVERABLE       = 0x0004          Limited discoverable mode
};
```

- Declaration of enumerated type for GAP bondable modes

```
enum RBLE_GAP_BONDABLE_MODE_enum {
    RBLE_GAP_NON_BONDABLE           = 0x0100,         Non-bondable mode
    RBLE_GAP_BONDABLE               = 0x0200          Bondable mode
};
```

- Declaration of enumerated type for GAP broadcast mode

```
enum RBLE_GAP_BROADCAST_MODE_enum {
    RBLE_GAP_BROADCASTER            = 0x0400          Broadcast mode
};
```

- Declaration of enumerated type for GAP connectable modes

```
enum RBLE_GAP_CONNECTABLE_MODE_enum {
    RBLE_GAP_NON_CONNECTABLE        = 0x0010,         Non-connectable mode
    RBLE_GAP_UND_CONNECTABLE        = 0x0020,         Undirected connectable mode
    RBLE_GAP_DIR_CONNECTABLE        = 0x0040          Directed connectable mode
};
```

- Declaration of enumerated type for GAP security modes

```
enum RBLE_GAP_SECURITY_MODE_enum {
    RBLE_GAP_NO_SEC                 = 0x00,   Security mode 1 level 1 (No security
                                              (No authentication and no encryption))
    RBLE_GAP_SEC1_NOAUTH_PAIR_ENC,            Security mode 1 level 2 (Unauthenticated
                                              pairing with encryption)
    RBLE_GAP_SEC1_AUTH_PAIR_ENC,              Security mode 1 level 3 (Authenticated
                                              pairing with encryption)
    RBLE_GAP_SEC2_NOAUTH_DATA_SGN,            Security mode 2 level 1 (Unauthenticated
                                              pairing with data signing)
    RBLE_GAP_SEC2_AUTH_DATA_SGN               Security mode 2 level 2 (Authenticated
                                              pairing with data signing)
};
```

- Declaration of enumerated type for GAP Advertising types

```
enum RBLE_GAP_ADV_TYPE_enum {
    RBLE_GAP_ADV_CONN_UNDIR         = 0x00,    Connectable Undirected advertising
                                               (Can respond to CONNECT_REQ or SCAN_REQ)
    RBLE_GAP_ADV_CONN_DIR_HIGH_DUTY,           Connectable high duty cycle
                                               directed advertising
                                               (Only connectable with specified device)
    RBLE_GAP_ADV_DISC_UNDIR,                   Discoverable undirected advertising
                                               (Can respond to SCAN_REQ)
    RBLE_GAP_ADV_NONCONN_UNDIR,                Non-connectable undirected advertising
                                               (Only information sent from Advertiser)
    RBLE_GAP_ADV_CONN_DIR_LOW_DUTY,            Connectable low duty cycle
                                               directed advertising
                                               (Only connectable with specified device)
};
```

- Declaration of enumerated type for GAP initiator filter policy

```
enum RBLE_GAP_INIT_FILTER_enum {
    RBLE_GAP_INIT_FILT_IGNORE_WLST        = 0x00,         Ignore the White List.
    RBLE_GAP_INIT_FILT_USE_WLST                           Use the White List.
};
```

- Declaration of enumerated type for GAP Advertising channel

```
enum RBLE_GAP_ADV_CH_enum {
    RBLE_ADV_CHANNEL_37                 = 0x01,         Use channel 37.
    RBLE_ADV_CHANNEL_38                 = 0x02,         Use channel 38.
    RBLE_ADV_CHANNEL_39                 = 0x04,         Use channel 39.
    RBLE_ADV_ALL_CHANNELS               = 0x07          Use all channels
                                                        (37, 38, and 39).
};
```

- Declaration of enumerated type for GAP Advertising filter policy

```
enum RBLE_GAP_ADV_FILTER_enum {
    RBLE_ADV_ALLOW_SCAN_ANY_CON_ANY     = 0x00,     Allow SCAN_REQ from any.
                                                    Allow CONNECT_REQ from any.
    RBLE_ADV_ALLOW_SCAN_WLST_CON_ANY,               Allow SCAN_REQ from White List
                                                    only.
                                                    Allow CONNECT_REQ from any.
    RBLE_ADV_ALLOW_SCAN_ANY_CON_WLST,               Allow SCAN_REQ from any.
                                                    Allow CONNECT_REQ from White List
                                                    only.
    RBLE_ADV_ALLOW_SCAN_WLST_CON_WLST               Allow SCAN_REQ from White List
                                                    only.
                                                    Allow CONNECT_REQ from White List
                                                    only.
};
```

- Declaration of enumerated type for GAP address types

```
enum RBLE_GAP_ADDR_TYPE_enum {
    RBLE_ADDR_PUBLIC                    = 0x00,     Public type
    RBLE_ADDR_RAND                                  Random type
};
```

- Declaration of enumerated type for GAP Scan types

```
enum RBLE_GAP_SCAN_TYPE_enum {
    RBLE_SCAN_PASSIVE                   = 0x00,     Passive Scanning. (No SCAN_REQ
                                                    packets shall be sent.)
    RBLE_SCAN_ACTIVE                                Active scanning. (SCAN_REQ
                                                    packets may be sent.)
};
```

- Declaration of enumerated type for GAP scanning filter policy

```
enum RBLE_GAP_SCAN_FILTER_enum {
    RBLE_SCAN_ALLOW_ADV_ALL             = 0x00,     Accept all advertisement packets.
    RBLE_SCAN_ALLOW_ADV_WLST                        Accept advertisement packets in
                                                    White List only.
};
```

- Declaration of enumerated type for GAP scanning duplicate filter policy

```
enum RBLE_GAP_SCAN_DUPLIC_enum {
    RBLE_SCAN_FILT_DUPLIC_DIS             = 0x00,    Disable duplicated filtering of
                                                     received data.
    RBLE_SCAN_FILT_DUPLIC_EN                         Enable duplicated filtering of
                                                     received data.
};
```

- Declaration of enumerated type for GAP privacy setting

```
enum RBLE_GAP_PRIV_SETTING_enum {
    RBLE_DEVICE_PRIV_DISABLE              = 0x00,    Disable the privacy feature.
    RBLE_CENTRAL_PRIV_ENABLE,                        Enable the privacy feature for
                                                     Centrals.
    RBLE_PH_PRIV_ENABLE,                             Enable the privacy feature for
                                                     Peripherals.
    RBLE_BCST_PRIV_ENABLE,                           Enable the privacy feature for
                                                     Broadcasters.
    RBLE_OBSERV_PRIV_ENABLE,                         Enable the privacy feature for
                                                     Observers.
    RBLE_OBSERV_PRIV_RESOLVE                         Address resolution performed by
                                                     Observer.
};
```

- Declaration of enumerated type for GAP key distribution flag

```
enum RBLE_GAP_KEY_DIST_enum {
    RBLE_KEY_DIST_NONE           = 0x00,      Distribute no key.
    RBLE_KEY_DIST_ENCKEY         = 0x01,      Distribute an encryption key.
    RBLE_KEY_DIST_IDKEY          = 0x02,      Distribute an IRK (Identity Resolving
                                              Key).
    RBLE_KEY_DIST_SIGNKEY        = 0x04       Distribute a CSRK (Connection Signature
                                              Resolving Key).
};
```

- Declaration of enumerated type for GAP OOB data flag

```
enum RBLE_GAP_OOB_PRESENT_enum {
    RBLE_OOB_AUTH_DATA_NOT_PRESENT = 0x00,           OOB data not present
    RBLE_OOB_AUTH_DATA_FROM_REMOTE_DEV_PRESENT       OOB data from a remote device
                                                     present
};
```

- Declaration of enumerated type for GAP IO capabilities

```
enum RBLE_GAP_IO_CAP_enum {
    RBLE_IO_CAP_DISPLAY_ONLY          = 0x00,      Input: No, output: Display
    RBLE_IO_CAP_DISPLAY_YES_NO,                    Input: Yes/No, output: Display
    RBLE_IO_CAP_KB_ONLY,                           Input: Keyboard, output: No
    RBLE_IO_CAP_NO_INPUT_NO_OUTPUT,                Input: No, output: No
    RBLE_IO_CAP_KB_DISPLAY                         Input: Keyboard, output: Display
```

```
};
```

- Declaration of enumerated type for authentication requirements

```
enum RBLE_AUTH_REQ_enum {
    RBLE_AUTH_REQ_NO_MITM_NO_BOND     = 0x00,       MITM protection not required.
                                                    No bonding.
    RBLE_AUTH_REQ_NO_MITM_BOND        = 0x01,       MITM protection not required.
                                                    Bonding.
    RBLE_AUTH_REQ_MITM_NO_BOND        = 0x04,       MITM protection required.
                                                    No bonding.
    RBLE_AUTH_REQ_MITM_BOND           = 0x05        MITM protection required.
                                                    Bonding.
};
```

- Declaration of enumerated type for GAP device discovery

```
enum RBLE_GAP_DISCOVERY_TYPE_enum {
    RBLE_GAP_GEN_DISCOVERY_TYPE       = 0x00,       General discovery. (Discover
                                                    devices in general or limited
                                                    discoverable mode.)
    RBLE_GAP_LIM_DISCOVERY_TYPE,                    Limited discovery. (Discover
                                                    devices in limited discoverable
                                                    mode.)
    RBLE_GAP_CANCEL_DISCOVERY                       Terminate device discovery.
};
```

- Declaration of enumerated type for GAP bonding information

```
enum RBLE_GAP_BOND_INFO_enum {
    RBLE_GAP_BOND_ADDED,                            Bonding information added.
    RBLE_GAP_BOND_REMOVED                           Bonding information removed.
};
```

- Declaration of enumerated type for GAP characteristic codes

```
enum RBLE_GAP_WR_CHAR_CODE_enum {
    RBLE_GAP_WR_CHAR_NAME,                          Device name characteristeic.
    RBLE_GAP_WR_CHAR_APPEARANCE                     Appearance characteristeic.
};
```

- Declaration of enumerated type for clock accuracy

```
enum RBLE_SAC_CLOCK_ACCURACY_enum {
    RBLE_SCA_500PPM,                                Clock accuracy: 500 ppm
    RBLE_SCA_250PPM,                                Clock accuracy: 250 ppm
    RBLE_SCA_150PPM,                                Clock accuracy: 150 ppm
    RBLE_SCA_100PPM,                                Clock accuracy: 100 ppm
    RBLE_SCA_75PPM,                                 Clock accuracy: 75 ppm
    RBLE_SCA_50PPM,                                 Clock accuracy: 50 ppm
    RBLE_SCA_30PPM,                                 Clock accuracy: 30 ppm
    RBLE_SCA_20PPM                                  Clock accuracy: 20 ppm
};
```

- Advertising parameter structure

```
typedef struct RBLE_SET_ADV_PARAM_t {
    uint16_t        adv_intv_min;                   Minimum advertising interval
    uint16_t        adv_intv_max;                   Maximum advertising interval
    uint8_t         adv_type;                       Advertising type
    uint8_t         own_addr_type;                  Local device address type
    uint8_t         direct_addr_type;               Direct address type
    RBLE_BD_ADDR    direct_addr;                    Direct connection Bluetooth
                                                    address
    uint8_t         adv_chnl_map;                   Advertising channel map
    uint8_t         adv_filt_policy;                Advertising filter policy
    uint8_t         reserved;                       Reserved
} RBLE_SET_ADV_PARAM;
```

- Advertising data structure

```
typedef struct RBLE_ADV_DATA_t {
    uint8_t       data[RBLE_ADV_DATA_LEN];          Advertising data
} RBLE_ADV_DATA;
```

- Advertising data setup structure

```
typedef struct RBLE_SET_ADV_DATA_t {
    uint8_t           adv_data_len;                 Advertising data length
    RBLE_ADV_DATA     adv_data;                     Advertising data
} RBLE_SET_ADV_DATA;
```

- Scan Response data structure

```
typedef struct RBLE_SCAN_RSP_DATA_t {
    uint8_t        data[RBLE_SCAN_RSP_DATA_LEN];    Scan Response data
} RBLE_SCAN_RSP_DATA;
```

- Scan Response data setup structure

```
typedef struct RBLE_SET_SCAN_RSP_DATA_t {
    uint8_t                 scan_rsp_data_len;          Scan Response data length
    RBLE_SCAN_RSP_DATA      data;                       Scan Response data
} RBLE_SET_SCAN_RSP_DATA;
```

- Advertising information structure

```
typedef struct RBLE_ADV_INFO_t {
    RBLE_SET_ADV_PARAM          adv_param;              Advertising parameter
    RBLE_SET_ADV_DATA           adv_data;               Advertising data
    RBLE_SET_SCAN_RSP_DATA scan_rsp_data;               Scan Response data
} RBLE_ADV_INFO;
```

- Scan parameter structure

```
typedef struct RBLE_SET_SCAN_PARAMETER_t {
    uint8_t         scan_type;                          Scan type
    uint8_t         reserved;                           Reserved
    uint16_t        scan_intv;                          Scan interval
    uint16_t        scan_window;                        Scan window
    uint8_t         own_addr_type;                      Local device address type
    uint8_t         scan_filt_policy;                   Scanning filter policy
} RBLE_SET_SCAN_PARAMETER;
```

- Scan information structure

```
typedef struct RBLE_SCANNING_INFO_t {
    RBLE_SET_SCAN_PARAMETER     set_scan;               Scan parameter
    uint8_t                     filter_dup;             Duplicate filter policy
    uint8_t                     reserved;               Reserved
} RBLE_SCANNING_INFO;
```

- White List add/remove parameter structure

```
typedef struct RBLE_DEV_ADDR_INFO_t {
    uint8_t                 dev_addr_type;              Device address type
    RBLE_BD_ADDR            dev_addr;                   Device address
} RBLE_DEV_ADDR_INFO;
```

- Connection parameter structure

```
typedef struct RBLE_CREATE_CONNECT_PARAM_t {
    uint16_t        scan_intv;              Scan interval
    uint16_t        scan_window;            Scan window
    uint8_t         init_filt_policy;       Initiator filter policy
    uint8_t         peer_addr_type;         Peer device address type
    RBLE_BD_ADDR    peer_addr;              Peer device address
    uint8_t         own_addr_type;          Local device address type
    uint8_t         reserved;               Reserved
    uint16_t        con_intv_min;           Minimum connection interval
    uint16_t        con_intv_max;           Maximum connection interval
    uint16_t        con_latency;            Connection latency
    uint16_t        superv_to;              Supervision timeout
    uint16_t        ce_len_min;             Minimum connection event
                                            length
    uint16_t        ce_len_max;             Maximum connection event
                                            length
} RBLE_CREATE_CONNECT_PARAM;
```

- Connection completion parameter structure

```
typedef struct RBLE_CONNECT_INFO_t {
    uint8_t         status;                 Connection establishment
                                            result
    uint8_t         role;                   Role
    uint16_t        conhdl;                 Connection handle
    uint8_t         peer_addr_type;         Peer device address type
    RBLE_BD_ADDR    peer_addr;              Peer device address
    uint8_t         idx;                    Connection index
    uint16_t        con_interval;           Connection interval
    uint16_t        con_latency;            Connection latency
    uint16_t        sup_to;                 Supervision timeout
    uint8_t         clk_accuracy;           Master clock accuracy
    uint8_t         reserved3;              Reserved
}RBLE_CONNECT_INFO;
```

- Scan enable/disable setup structure

```
typedef struct RBLE_SET_SCAN_EN_t {
    uint8_t         scan_en;                Enable/disable scanning
    uint8_t         filter_duplic_en;   Enable/disable duplicated
                                            filtering
} RBLE_SET_SCAN_EN;
```

- Bonding parameter structure

```
typedef struct RBLE_BOND_PARAM_t {
    RBLE_BD_ADDR        addr;                       Device address
    uint8_t             oob;                        OOB information
    uint8_t             iocap;                      I/O capabilities
    uint8_t             auth;                       Authentication requirement
    uint8_t             key_size;                   Encryption key size
    uint8_t             ikey_dist;                  Initiator key distribution
                                                    flag
    uint8_t             rkey_dist;                  Responder key distribution
                                                    flag
} RBLE_BOND_PARAM;
```

- Bonding response parameter structure

```
typedef struct RBLE_BOND_RESP_PARAM_t {
    uint16_t            conhdl;                     Connection handle
    uint8_t             accept;                     Accept/reject flag
    uint8_t             io_cap;                     I/O capabilities
    uint8_t             oob;                        OOB information
    uint8_t             auth_req;                   Authentication requirement
    uint8_t             max_key_size;               Maximum key size
    uint8_t             ikeys;                      Initiator key distribution
                                                    flag
    uint8_t             rkeys;                      Responder key distribution
                                                    flag
    uint8_t             reserved;                   Reserved
} RBLE_BOND_RESP_PARAM;
```

- Connection update parameter structure

```
typedef struct RBLE_CONN_PARAM_t {
    uint16_t            intv_min;                   Minimum connection interval
    uint16_t            intv_max;                   Maximum connection interval
    uint16_t            latency;                    Connection latency
    uint16_t            time_out;                   Supervision timeout
} RBLE_CONN_PARAM;
```

- Device version information structure

```
typedef struct RBLE_DEVICE_VER_INFO_t {
    uint8_t             hci_ver;                    HCI version
    uint8_t             lmp_ver;                    LMP version
    uint8_t             host_ver;                   Host version
    uint8_t             reserved;                   Reserved
    uint16_t            hci_subver;                 HCI subversion
    uint16_t            lmp_subver;                 LMP subversion
    uint16_t            host_subver;                Host subversion
    uint16_t            company_id;                 Company ID
} RBLE_DEVICE_VER_INFO;
```

- LE features structure

```
typedef struct RBLE_FEATURES_t {
    uint8_t            feats[RBLE_LE_FEATS_LEN];          LE Features
} RBLE_FEATURES;
```

- Advertising report structure

```
typedef struct RBLE_ADV_REPORT_t {
    uint8_t            evt_type;                          Advertising event type
    uint8_t            adv_addr_type;                     Advertising address type
    RBLE_BD_ADDR       adv_addr;                          Advertising device address
    uint8_t            data_len;                          Advertising data length
    uint8_t            data[RBLE_ADV_DATA_LEN];           Advertising data
    uint8_t            rssi;                              RSSI value
} RBLE_ADV_REPORT;
```

- Advertising report event structure

```
typedef struct RBLE_ADV_REPORT_EVT_t {
    RBLE_ADV_REPORT        adv_rep;                       Advertising report
} RBLE_ADV_REPORT_EVT;
```

- GAP event parameter structure

```
typedef struct RBLE_GAP_EVENT_t {
    RBLE_GAP_EVENT_TYPE      type;                        GAP event type
    uint8_t                  reserved;                    Reserved
    union Event_Parameter_u {
        Generic event
        RBLE_STATUS        status;                        Status

        Reset completion event
        struct RBLE_GAP_Reset_Result_t {
            RBLE_STATUS         status;                   Reset result
            uint8_t             rBLE_major_ver;           rBLE major version
            uint8_t             rBLE_minor_ver;           rBLE minor version
        } reset_result;

        Security mode setup event
        struct RBLE_GAP_Set_Security_Request_t{
            RBLE_STATUS         status;                   Status
            uint8_t             sec;                      Security mode
        } set_sec_req;
```

Device information acquisition completion event

```
struct RBLE_GAP_Get_Device_Info_t {
    RBLE_STATUS             status;             Status
    RBLE_BD_ADDR            addr;               Device address
    uint8_t                 reserved;           Reserved
    RBLE_DEVICE_VER_INFO    ver_info;           Version information
} get_dev_ver;
```

Local device White List size read completion event

```
struct RBLE_GAP_Get_Wlst_size_t {
    RBLE_STATUS         status;                 Status
    uint8_t             wlist_size;             White List size
} get_wlst_size;
```

Remote device name acquisition completion event

```
struct RBLE_GAP_Get_Remote_Device_Name_t {
    RBLE_STATUS         status;                 Status
    RBLE_BD_NAME        bd_name;                Device name
    uint8_t             reserved;  Reserved
} get_remote_dev_name;
```

Remote device information acquisition completion event

```
struct RBLE_GAP_GET_Remote_Device_Info_t {
    RBLE_STATUS         status;                 Status
    uint8_t             reserved;  Reserved
    uint16_t            conhdl;                 Connection handle
    uint16_t            vers;                   LMP version
    uint16_t            compid;                 Company ID
    uint16_t            subvers;                LMP subversion
    RBLE_FEATURES       feats_used;             LE Features
} get_remote_dev_info;
```

Device search result notification event

```
struct RBLE_GAP_Device_Search_Result_t {
    RBLE_ADV_REPORT     adv_resp;               Advertising report
} dev_search_result;
```

Resolvable Private Address resolution comletion event

```
struct RBLE_GAP_RPA_Resolved_Evt_t {
    RBLE_BD_ADDR        res_addr;               Resolved device address
    uint8_t             res_addr_type;          Resolved address type
    RBLE_BD_ADDR        addr;                   Previous device address
    uint8_t             addr_type;              Previous address type
} rpa_resolved;
```

Random address setup completion event

```
struct RBLE_GAP_Set_Random_Address_t {
    RBLE_STATUS         status;                     Status
    RBLE_BD_ADDR        addr;                       Device address
} set_rand_adr;
```

LE link connection completion event

```
struct RBLE_GAP_Connection_t {
    RBLE_CONNECT_INFO    connect_info;              Connection completion
                                                    parameter
} conn_comp;
```

LE link disconnection completion event

```
struct RBLE_GAP_Disconnect_t {
    uint8_t            reason;                      Reason for disconnection
    RBLE_STATUS        status;                      Status
    uint16_t           conhdl;                      Connection handle
} disconnect;
```

Advertising report notification event

```
struct RBLE_GAP_Advertising_Report_t {
    RBLE_ADV_REPORT_EVT            evt;             Advertising event
    uint8_t                        reserved;        Reserved
} adv_report;
```

Bonding completion event

```
struct RBLE_GAP_Bonding_Comp_t {
    uint16_t           conhdl;                      Connection handle
    uint8_t            idx;                         Connection index
    RBLE_STATUS        status;                      Status
    uint8_t            key_size;                    Key size
    uint8_t            sec_prop;                    Security property
} bonding_comp;
```

Bonding request notification event

```
struct RBLE_GAP_Bonding_Req_t {
    RBLE_BD_ADDR       bd_addr;                     Device address
    uint8_t            index;                       Connection index
    uint8_t            auth_req;                    Authentication requirement
    uint8_t            io_cap;                      I/O Capability
    uint8_t            oob_data_flg;                OOB data flag
    uint8_t            max_enc_size;                Maximum key size
    uint8_t            ikey_dist;                   Initiator key distribution
                                                    flag
    uint8_t            rkey_dist;                   Responder key distribution
                                                    flag
} bonding_req;
```

Connection parameter change request notification event

```
struct RBLE_GAP_Change_Connection_Param_Req_Ind_t {
    uint16_t          conhdl;                     Connection handle
    RBLE_CONN_PARAM   conn_param;                 Connection parameter
} chg_connect_param_req;
```

Connection parameter change completion event

```
struct RBLE_GAP_Change_Connection_Param_t {
    RBLE_STATUS       status;                     Status
    uint8_t           reserved;                   Reserved
    uint16_t          con_interval;               Connection interval
    uint16_t          con_latency;                Connection latency
    uint16_t          sup_to;                     Supervision timeout
} chg_connect_param;
```

Connection parameter change request response notification event

```
struct RBLE_GAP_Change_Connection_Param_Response_t {
    RBLE_STATUS       status;                     Status
    uint8_t           reserved;                   Reserved
    uint16_t          result;                     Change result
    uint16_t          conhdl;                     Connection handle
} chg_connect_param_resp;
```

RSSI acquisition completion event

```
struct RBLE_GAP_Read_RSSI_Cmp_Evt_t{
    uint16_t          conhdl;                     Connection handle
    RBLE_STATUS       status;                     Status
    uint8_t           rssi;                       RSSI value
} read_rssi;
```

GAP characteristic write indication event

```
struct RBLE_GAP_Wr_Char_Ind_Evt_t{
    uint16_t          conhdl;                     Connection handle
    uint16_t          type;                       Write characteristic code
    union {
        RBLE_BD_NAME  name;                       Device name characteristic
        uint16_t      appearance;                 Appearance characteristic
    } param;
} wr_char;
```

```
        GAP command error notification event
        struct RBLE_GAP_Command_Error_Ind_t {
            RBLE_STATUS        status;                     Status
            uint8_t            reserved;                   Reserved
            uint16_t           opcode;                     Opcode
        } cmd_disallowed_ind;
    } param;
} RBLE_GAP_EVENT;
```

5.2 Functions

Table 5-1 shows the API functions defined for the GAP of rBLE and the following sections describe the API functions in detail.

Table 5-1 API Functions Used by the GAP

RBLE_GAP_Reset	Resets the GAP.
RBLE_GAP_Set_Name	Sets the local device name.
RBLE_GAP_Observation_Enable	Enables observation.
RBLE_GAP_Observation_Disable	Disables observation.
RBLE_GAP_Broadcast_Enable	Enables broadcasting.
RBLE_GAP_Broadcast_Disable	Disables broadcasting.
RBLE_GAP_Set_Bonding_Mode	Sets up bonding mode.
RBLE_GAP_Set_Security_Request	Sets up security mode.
RBLE_GAP_Get_Device_Info	Acquires local device information.
RBLE_GAP_Get_White_List_Size	Acquires the local device White List size.
RBLE_GAP_Add_To_White_List	Adds a device to the White List.
RBLE_GAP_Del_From_White_List	Deletes a device from the White List.
RBLE_GAP_Get_Remote_Device_Name	Acquires the remote device name.
RBLE_GAP_Get_Remote_Device_Info	Acquires remote device information.
RBLE_GAP_Device_Search	Searches for a remote device.
RBLE_GAP_Set_Random_Address	Sets up a random address to the link layer.
RBLE_GAP_Set_Privacy_Feature	Sets up the GAP privacy feature.
RBLE_GAP_Create_Connection	Starts connection to an LE link.
RBLE_GAP_Connection_Cancel	Cancels connection to an LE link.
RBLE_GAP_Disconnect	Disconnects an LE link.
RBLE_GAP_Start_Bonding	Starts bonding.
RBLE_GAP_Bonding_Info_Ind	Indicates bonding information.
RBLE_GAP_Bonding_Response	Responds to a bonding request.
RBLE_GAP_Change_Connection_Param	Changes the link parameter.
RBLE_GAP_Channel_Map_Req	Sets or acquires a channel map.
RBLE_GAP_Read_RSSI	Reads RSSI.
RBLE_GAP_Authorized_Ind	Indicates authorization.

5.2.1 RBLE_GAP_Reset

RBLE_STATUS RBLE_GAP_Reset (RBLE_GAP_EVENT_HANDLER gap_call_back, RBLE_SM_EVENT_HANDLER sm_call_back)		
This function resets the GAP. This function must be called before using any of the Bluetooth profiles. The result is reported by using the GAP reset completion event RBLE_GAP_EVENT_RESET_RESULT. * If a Bluetooth profile is used (another function is called) before this function is called, no events will be reported. The operation of the profile is also not guaranteed.		
Parameters:		
gap_call_back	Specify the callback function that reports the GAP event.	
sm_call_back	Specify the callback function that reports the SM event.	
Return:		
RBLE_OK	Success	
RBLE_PARAM_ERR	Invalid parameter	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.2 RBLE_GAP_Set_Name

RBLE_STATUS RBLE_GAP_Set_Name(RBLE_BD_NAME *dev_name)		
This function sets the name of the local device. A character string of up to 64 bytes can be specified for the device name. The result is reported by using the device name setup completion event RBLE_GAP_EVENT_SET_NAME_COMP. * During the power on, the device name set by this function is retained until the next reset of the GAP (RBLE_GAP_Reset).		
Parameters:		
**dev_name*	*namelen*	Device name data length
	name	Device name data
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.3 RBLE_GAP_Observation_Enable

RBLE_STATUS RBLE_GAP_Observation_Enable(uint16_t mode, RBLE_SCANNING_INFO *set_scan)				
This function enables the observation procedure or connection procedure. The result is reported by using the observation enable event RBLE_GAP_EVENT_OBSERVATION_ENABLE_COMP.				
Parameters:				
	mode	RBLE_GAP_OBSERVER	Executes the observation procedure.	
		RBLE_GAP_AUTO_CONNECT	Executes the auto connection procedure.	
		RBLE_GAP_SELECT_CONNECT	Executes the selective connection procedure.	
	**set_scan*	*scan_type*	RBLE_SCAN_PASSIVE	Executes passive scanning. (No SCAN_REQ packets shall be sent.)
			RBLE_SCAN_ACTIVE	Executes active scanning. (SCAN_REQ packets may be sent.)
		scan_intv	Scan interval N = 0x0004 to 0x4000 (Time = N x 0.625 ms (2.5 ms to 10.24 sec.))	
		scan_window	Scan window size N = 0x0004 to 0x4000 (Time = N x 0.625 ms (2.5 ms to 10.24 sec.)) * Scan interval > Scan window size	
		own_addr_type	RBLE_ADDR_PUBLIC	Public BD address
			RBLE_ADDR_RAND	Random BD address
		scan_filt_policy	RBLE_SCAN_ALLOW_ADV_ALL	Accept all advertisement packets.
			RBLE_SCAN_ALLOW_ADV_WLST	Accept advertisement packets in White List only.
		filter_dup	RBLE_SCAN_FILT_DUPLIC_DIS	Disables duplicated filtering of received data.
			RBLE_SCAN_FILT_DUPLIC_EN	Enables duplicated filtering of received data.
Return:				
	RBLE_OK	Success		
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.		

5.2.4 RBLE_GAP_Observation_Disable

RBLE_STATUS RBLE_GAP_Observation_Disable(void)		
This function disables the mode enabled by using the RBLE_GAP_Observation_Enable function. The result is reported by using the observation disable event RBLE_GAP_EVENT_OBSERVATION_DISABLE_COMP.		
Parameters:		
	none	
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.5 RBLE_GAP_Broadcast_Enable

RBLE_STATUS RBLE_GAP_Broadcast_Enable(uint16_t disc_mode, uint16_t conn_mode, RBLE_ADV_INFO *adv_info)

This function sets up the Discoverable mode and Connectable mode.

When operating as a Broadcaster (such as beacon), set the parameters as follows:

disc_mode = RBLE_GAP_BROADCASTER

con_mode = 0

And, specify the RBLE_GAP_ADV_DISC_UNDIR or RBLE_GAP_ADV_NONCONN_UNDIR to *adv_type*.

The result is reported by using the broadcast enable event RBLE_GAP_EVENT_BROADCAST_ENABLE_COMP.

* If the *disc_mode* has set to RBLE_GAP_LIM_DISCOVERABLE, do not call RBLE_GAP_Broadcast_Disable() function.

Parameters:

disc_mode	RBLE_GAP_NON_DISCOVERABLE		Not discoverable by any device performing either the general discovery procedure or the limited discovery procedure.
	RBLE_GAP_GEN_DISCOVERABLE		Discoverable by devices performing the general discovery procedure.
	RBLE_GAP_LIM_DISCOVERABLE		Discoverable for a limited period of time by other devices performing the limited or general device discovery procedure.
	RBLE_GAP_BROADCASTER		Data is broadcast by an Advertising event
conn_mode	0		Operates as a Broadcaster.
	RBLE_GAP_NON_CONNECTABLE		Connection not allowed.
	RBLE_GAP_UND_CONNECTABLE		Connectable
	RBLE_GAP_DIR_CONNECTABLE		Only connectable with a known device
**adv_info*	*adv_intv_min*	Minimum advertising interval N = 0x0020 to 0x4000 (Time = N x 0.625 ms (20 ms to 10.24 sec.)) * If the *adv_type* is set to RBLE_GAP_ADV_DISC_UNDIR or RBLE_GAP_ADV_NONCONN_UNDIR, *adv_intv_min* shall not be set to less than 0x00A0 (100 ms).	
	adv_intv_max	Maximum advertising interval N = 0x0020 to 0x4000 (Time = N x 0.625 ms (20 ms to 10.24 sec.)) * If the *adv_type* is set to RBLE_GAP_ADV_DISC_UNDIR or RBLE_GAP_ADV_NONCONN_UNDIR, *adv_intv_max* shall not be set to less than 0x00A0 (100 ms).	
	adv_type	RBLE_GAP_ADV_CONN_UNDIR	Can respond to CONNECT_REQ or SCAN_REQ.
		RBLE_GAP_ADV_CONN_DIR_HIGH_DUTY	Only connectable with specified device.
		RBLE_GAP_ADV_DISC_UNDIR	Can respond to SCAN_REQ.
		RBLE_GAP_ADV_NONCONN_UNDIR	Only information sent from Advertiser
		RBLE_GAP_ADV_CONN_DIR_LOW_DUTY	Only connectable with specified device.
	own_addr_type	Local device address type Public address: RBLE_ADDR_PUBLIC Random address: RBLE_ADDR_RAND	

		direct_addr_type	Direct connection address type Public address: RBLE_ADDR_PUBLIC Random address: RBLE_ADDR_RAND	
		direct_addr	Direct connection address	
		adv_chnl_map	RBLE_ADV_CHANNEL_37	Use channel 37.
			RBLE_ADV_CHANNEL_38	Use channel 38.
			RBLE_ADV_CHANNEL_39	Use channel 39.
			RBLE_ADV_ALL_CHANNELS	Use all channels (37, 38, and 39).
		adv_filt_policy	RBLE_ADV_ALLOW_SCAN_ANY_CON_ANY	Allow SCAN_REQ from any. Allow CONNECT_REQ from any.
			RBLE_ADV_ALLOW_SCAN_WLST_CON_ANY	Allow SCAN_REQ from White List only. Allow CONNECT_REQ from any.
			RBLE_ADV_ALLOW_SCAN_ANY_CON_WLST	Allow SCAN_REQ from any. Allow CONNECT_REQ from White List only.
			RBLE_ADV_ALLOW_SCAN_WLST_CON_WLST	Allow SCAN_REQ from White List only. Allow CONNECT_REQ from White List only.
		adv_data_len	Advertising data length	
		adv_data	Advertising data Note: For details about the Advertising data format, see the *Bluetooth Low Energy Protocol Stack User's Manual*.	
		scan_rsp_data_len	Scan Response data length	
		data	Scan Response data Note: For details about the Scan Response data format, see the *Bluetooth Low Energy Protocol Stack User's Manual*.	
Return:				
	RBLE_OK		Success	
	RBLE_STATUS_ERROR		Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.6 RBLE_GAP_Broadcast_Disable

RBLE_STATUS RBLE_GAP_Broadcast_Disable(void)		
This function disables the mode enabled by using the RBLE_GAP_Broadcast_Enable function. The result is reported by using the broadcast disable event RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP. * When the RBLE_GAP_LIM_DISCOVERABLE has been set by RBLE_GAP_Broadcast_Enable() function, do not call this function.		
Parameters:		
none		
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.7 RBLE_GAP_Set_Bonding_Mode

RBLE_STATUS RBLE_GAP_Set_Bonding_Mode(uint16_t mode)		
This function sets up the bonding mode. The result is reported by using the bonding mode setup event RBLE_GAP_EVENT_SET_BONDING_MODE_COMP.		
Parameters:		
mode	RBLE_GAP_NON_BONDABLE	Non-bondable mode
	RBLE_GAP_BONDABLE	Bondable mode
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.8 RBLE_GAP_Set_Security_Request

RBLE_STATUS RBLE_GAP_Set_Security_Request(uint8_t sec)		
This function sets up the security mode. The result is reported by using the security mode setup event RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP.		
Parameters:		
sec	RBLE_GAP_NO_SEC	No security
	RBLE_GAP_SEC1_NOAUTH_PAIR_ENC	Unauthenticated pairing with encryption
	RBLE_GAP_SEC1_AUTH_PAIR_ENC	Authenticated pairing with encryption
	RBLE_GAP_SEC2_NOAUTH_DATA_SGN	Unauthenticated pairing with data signing
	RBLE_GAP_SEC2_AUTH_DATA_SGN	Authenticated pairing with data signing
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.9 RBLE_GAP_Get_Device_Info

RBLE_STATUS RBLE_GAP_Get_Device_Info(void)	
This function acquires local device information (device address, BLE stack version). The result is reported by using the device information acquisition completion event RBLE_GAP_EVENT_GET_DEVICE_INFO_COMP.	
Parameters:	
none	
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.10 RBLE_GAP_Get_White_List_Size

RBLE_STATUS RBLE_GAP_Get_White_List_Size(void)	
This function reads the size of the White List of the local device. The result is reported by using the local device White List size read completion event RBLE_GAP_EVENT_GET_WHITE_LIST_SIZE_COMP.	
Parameters:	
none	
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.11 RBLE_GAP_Add_To_White_List

RBLE_STATUS RBLE_GAP_Add_To_White_List(RBLE_DEV_ADDR_INFO *dev_info)			
This function adds specified known devices such as bonded devices to the White List. The result is reported by using the White List device addition completion event RBLE_GAP_EVENT_ADD_TO_WHITE_LIST_COMP.			
Parameters:			
**dev_info*	*dev_addr_type*	RBLE_ADDR_PUBLIC	Public BD address
		RBLE_ADDR_RAND	Random BD address
	dev_addr	BD address of the device added to the White List	
Return:			
RBLE_OK		Success	
RBLE_STATUS_ERROR		Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.12 RBLE_GAP_Del_From_White_List

RBLE_STATUS RBLE_GAP_Del_From_White_List(bool all_dev, RBLE_DEV ADDR_INFO *dev_info)				
This function removes the specified devices from the White List. The result is reported by using the White List device removal completion event RBLE_GAP_EVENT_DEL_FROM_WHITE_LIST_COMP.				
Parameters:				
	all_dev	Flag indicating removal of all devices from the White List (TRUE: All removed, FALSE: Only specified device removed) * If all_dev is TRUE, the following parameters are invalid:		
	**dev_info*	*dev_addr_type*	RBLE_ADDR_PUBLIC	Public BD address
			RBLE_ADDR_RAND	Random BD address
		dev_addr	BD address of the device removed from the White List	
Return:				
	RBLE_OK	Success		
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.		

5.2.13 RBLE_GAP_Get_Remote_Device_Name

RBLE_STATUS RBLE_GAP_Get_Remote_Device_Name(RBLE_CREATE_CONNECT_PARAM *connect_param)
This function acquires the name of the specified remote device. The result is reported by using the remote device name acquisition completion event RBLE_GAP_EVENT_GET_REMOTE_DEVICE_NAME_COMP. * If already connected, set the BD address of the connected device to *peer_addr* of the following parameters.

Parameters:

*connect_param	*scan_intv*	Scan interval N = 0x0004 to 0x4000 (Time = N x 0.625 ms (2.5 ms to 10.24 sec.))	
	scan_window	Scan window size N = 0x0004 to 0x4000 (Time = N x 0.625 ms (2.5 ms to 10.24 sec.)) * Scan interval > Scan window size	
	init_filt_policy	RBLE_GAP_INIT_FILT_IGNORE_WLST	Connect to the device specified by *peer_addr_type, peer_addr* without using the White List.
		RBLE_GAP_INIT_FILT_USE_WLST	Use the White List to connect to the device registered in the White List. (*peer_addr_type, peer_addr* is ignored.)
	peer_addr_type	Peer device address type Public address: RBLE_ADDR_PUBLIC Random address: RBLE_ADDR_RAND * This parameter is only available when *init_filt_policy* is RBLE_GAP_INIT_FILT_IGNORE_WLST.	
	peer_addr	Peer device address * This parameter is only available when *init_filt_policy* is RBLE_GAP_INIT_FILT_IGNORE_WLST.	
	own_addr_type	Local device address type Public address: RBLE_ADDR_PUBLIC Random address: RBLE_ADDR_RAND	
	con_intv_min	Minimum connection interval N = 0x0006 to 0x0C80 (Time = N x 1.25 ms (7.5 ms to 4.0 sec.))	
	con_intv_max	Maximum connection interval N = 0x0006 to 0x0C80 (Time = N x 1.25 ms (7.5 ms to 4.0 sec.))	
	con_latency	Connection slave latency (0x0000 to 0x01F3)	
	superv_to	Supervision timeout N = 0x000A to 0x0C80 (Time = N x 10 ms (100 ms to 32 sec.))	
	ce_len_min	Minimum connection event length (0x0000 to 0xFFFF) *This parameter is reserved for the future , and it's unused currently in the BLE software.	
	ce_len_max	Maximum connection event length (0x0000 to 0xFFFF) *This parameter is reserved for the future , and it's unused currently in the BLE software.	

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.14 RBLE_GAP_Get_Remote_Device_Info

RBLE_STATUS RBLE_GAP_Get_Remote_Device_Info(unit16_t conhdl)	
This function acquires information about the specified device such as the BLE stack version and the LE support features. The result is reported by using the remote device information acquisition completion event RBLE_GAP_EVENT_GET_REMOTE_DEVICE_INFO_COMP.	
Parameters:	
conhdl	Connection handle
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.15 RBLE_GAP_Device_Search

RBLE_STATUS RBLE_GAP_Device_Search(uint8_t discovery_type, uint8_t addr_type)

This function performs scanning to search for peripheral devices. Searching for devices is performed for 7.68 seconds and then automatically stops.

Performs scanning with the following parameters.

Scan Type	Active scan
Scan Interval	11.25msec
Scan Window	11.25msec
Duplicate filtering	Enable

The result is reported by using the device search command completion event RBLE_GAP_EVENT_DEVICE_SEARCH_COMP.

Each time a device is discovered, the result is reported by using the device search result notification event RBLE_GAP_EVENT_DEVICE_SEARCH_RESULT_IND.

Parameters:

discovery_type	RBLE_GAP_GEN_DISCOVERY_TYPE	General discovery. (Discover devices in general or limited discoverable mode.)
	RBLE_GAP_LIM_DISCOVERY_TYPE	Limited discovery (Discover devices in limited discoverable mode.)
	RBLE_GAP_CANCEL_DISCOVERY	Cancel discovery
addr_type	RBLE_ADDR_PUBLIC	Public BD address
	RBLE_ADDR_RAND	Random BD address

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.16 RBLE_GAP_Set_Random_Address

RBLE_STATUS RBLE_GAP_Set_Random_Address(RBLE_BD_ADDR *bd_addr)		
This function sets the local device address as the specified random address. The result is reported by using the random address setup completion event RBLE_GAP_EVENT_SET_RANDOM_ADDRESS_COMP. * During the power on, the random address set by this function is retained until the next reset of the GAP (RBLE_GAP_Reset).		
Parameters:		
**bd_addr*	Random address to be set	
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.17 RBLE_GAP_Set_Privacy_Feature

RBLE_STATUS RBLE_GAP_Set_Privacy_Feature(uint8_t priv_flag, uint8_t set_to_ll)		
This function sets the privacy feature for the local device. When enable the privacy feature for each role, Resolvable Private Address is generated. For this reason, before this function is executed, the IRK must be set by using the RBLE_SM_Set_Key function. When enable only address resolution procedures, specify RBLE_OBSERV_PRIV_RESOLVE to priv_flag. The result is reported by using the privacy feature setup completion event RBLE_GAP_EVENT_SET_PRIVACY_FEATURE_COMP.		
Parameters:		
priv_flag	RBLE_DEVICE_PRIV_DISABLE	Disables the privacy feature.
	RBLE_CENTRAL_PRIV_ENABLE	Enables the privacy feature for Centrals.
	RBLE_PH_PRIV_ENABLE	Enables the privacy feature for Peripherals.
	RBLE_BCST_PRIV_ENABLE	Enables the privacy feature for Broadcasters.
	RBLE_OBSERV_PRIV_ENABLE	Enables the privacy feature for Observers.
	RBLE_OBSERV_PRIV_RESOLVE	Enables Resolvable Private Address resolution procedure.
set_to_ll	Flag indicating whether a generated random address has been specified as a Link Layer address or not. (TRUE: Specified as Link Layer address, FALSE: Not specified as Link Layer address)	
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.18 RBLE_GAP_Create_Connection

RBLE_STATUS RBLE_GAP_Create_Connection(RBLE_CREATE_CONNECT_PARAM *connect_param)			
This function establishes a link with the specified remote device. The result is reported by using the LE link establishment event RBLE_GAP_EVENT_CONNECTION_COMP.			
Parameters:			
	*connect_param	*scan_intv*	Scan interval N = 0x0004 to 0x4000 (Time = N x 0.625 ms (2.5 ms to 10.24 sec.))
		scan_window	Scan window size N = 0x0004 to 0x4000 (Time = N x 0.625 ms (2.5 ms to 10.24 sec.)) * Scan interval > Scan window size
		init_filt_policy	RBLE_GAP_INIT_FILT_IGNORE_WLST: Connect to the device specified by *peer_addr_type, peer_addr* without using the White List.
			RBLE_GAP_INIT_FILT_USE_WLST: Use the White List to connect to the device registered in the White List. (*peer_addr_type, peer_addr* is ignored.)
		peer_addr_type	Peer device address type Public address: RBLE_ADDR_PUBLIC Random address: RBLE_ADDR_RAND * This parameter is only available when *init_filt_policy* is RBLE_GAP_INIT_FILT_IGNORE_WLST.
		peer_addr	Peer device address * This parameter is only available when *init_filt_policy* is RBLE_GAP_INIT_FILT_IGNORE_WLST.
		own_addr_type	Local device address type Public address: RBLE_ADDR_PUBLIC Random address: RBLE_ADDR_RAND
		con_intv_min	Minimum connection interval N = 0x0006 to 0x0C80 (Time = N x 1.25 ms (7.5 ms to 4.0 sec.))
		con_intv_max	Maximum connection interval N = 0x0006 to 0x0C80 (Time = N x 1.25 ms (7.5 ms to 4.0 sec.))
		con_latency	Connection slave latency (0x0000 to 0x01F3)
		superv_to	Supervision timeout N = 0x000A to 0x0C80 (Time = N x 10 ms (100 ms to 32 sec.)) *The Supervision timeout shall be larger than (1 + *con_latency*) * *con_intv_max* * 2, where con_intv_max is given in milliseconds.
		ce_len_min	Minimum connection event length (0x0000 to 0xFFFF) *This parameter is reserved for the future , and it's unused currently in the BLE software.
		ce_len_max	Maximum connection event length (0x0000 to 0xFFFF) *This parameter is reserved for the future , and it's unused currently in the BLE software.
Return:			
	RBLE_OK	Success	
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.19 RBLE_GAP_Connection_Cancel

RBLE_STATUS RBLE_GAP_Connection_Cancel(void)		
This function cancels the request for establishing a link with a remote device. The result is reported by using the LE link connection cancel completion event RBLE_GAP_EVENT_CONNECTION_CANCEL_COMP.		
Parameters:		
	none	
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.20 RBLE_GAP_Disconnect

RBLE_STATUS RBLE_GAP_Disconnect(uint16_t conhdl)		
This function disconnects the link with the specified remote device. The result is reported by using the LE link disconnection completion event RBLE_GAP_EVENT_DISCONNECT_COMP.		
Parameters:		
	conhdl	Connection handle
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.21 RBLE_GAP_Start_Bonding

RBLE_STATUS RBLE_GAP_Start_Bonding(RBLE_BOND_PARAM *bond_param)			
This function starts bonding with the specified remote device. The result is reported by using the bonding completion event RBLE_GAP_EVENT_BONDING_COMP.			
Parameters:			
*bond_param	*addr*	BD address of the remote device with which to create a bond	
	oob	RBLE_OOB_AUTH_DATA_NOT_PRESENT	OOB data not present
		RBLE_OOB_AUTH_DATA_FROM_REMOTE_DEV_PRESENT	OOB data from a remote device present
	iocap	RBLE_IO_CAP_DISPLAY_ONLY	Input: No Output: Display
		RBLE_IO_CAP_DISPLAY_YES_NO	Input: Yes/No Output: Display
		RBLE_IO_CAP_KB_ONLY	Input: Keyboard Output: No
		RBLE_IO_CAP_NO_INPUT_NO_OUTPUT	Input: No Output: No
		RBLE_IO_CAP_KB_DISPLAY	Input: Keyboard Output: Display
	auth	RBLE_AUTH_REQ_NO_MITM_NO_BOND	Protection against MITM not implemented. No bonding performed.
		RBLE_AUTH_REQ_NO_MITM_BOND	Protection against MITM not implemented. Bonding performed.
		RBLE_AUTH_REQ_MITM_NO_BOND	Protection against MITM implemented. No bonding performed.
		RBLE_AUTH_REQ_MITM_BOND	Protection against MITM implemented. Bonding performed.
	key_size	Maximum encryption key size	
	ikey_dist	Type of key distributed by the initiator (select by using OR) RBLE_KEY_DIST_NONE: No key distributed. RBLE_KEY_DIST_ENCKEY: LTK distributed. RBLE_KEY_DIST_IDKEY: IRK distributed. RBLE_KEY_DIST_SIGNKEY: CSRK distributed.	
	rkey_dist	Type of key distributed by the responder (select by using OR) RBLE_KEY_DIST_NONE: No key distributed. RBLE_KEY_DIST_ENCKEY: LTK distributed. RBLE_KEY_DIST_IDKEY: IRK distributed. RBLE_KEY_DIST_SIGNKEY: CSRK distributed.	
Return:			
RBLE_OK	Success		
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.		

5.2.22 RBLE_GAP_Bonding_Info_Ind

RBLE_STATUS RBLE_GAP_Bonding_Info_Ind(uint8_t bond_op, RBLE_BD_ADDR *addr)		
This function indicates the bonding information for the GAP layer.		
Parameters:		
bond_op	RBLE_GAP_BOND_ADDED	Bonding information added.
	RBLE_GAP_BOND_REMOVED	Bonding information removed.
**addr*	BD address of the remote device to be added or removed	
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

5.2.23 RBLE_GAP_Bonding_Response

RBLE_STATUS RBLE_GAP_Bonding_Response(RBLE_BOND_RESP_PARAM *res_bond_param)			
This function responds to a bonding request from the specified remote device. The result is reported by using the bonding completion event RBLE_GAP_EVENT_BONDING_COMP.			
Parameters:			
*res_bond_param	*conhdl*	Connection handle	
	accept	Bonding request response flag RBLE_OK: Acceptable RBLE_CONN_REJ_UNACCEPTABLE_BDADDR: Unacceptable	
	iocap	RBLE_IO_CAP_DISPLAY_ONLY	Input: No Output: Display
		RBLE_IO_CAP_DISPLAY_YES_NO	Input: Yes/No Output: Display
		RBLE_IO_CAP_KB_ONLY	Input: Keyboard Output: No
		RBLE_IO_CAP_NO_INPUT_NO_OUTPUT	Input: No Output: No
		RBLE_IO_CAP_KB_DISPLAY	Input: Keyboard Output: Display
	oob	RBLE_OOB_AUTH_DATA_NOT_PRESENT	OOB data not present
		RBLE_OOB_AUTH_DATA_FROM_REMOTE_DEV_PRESENT	OOB data from a remote device present
	auth_req	RBLE_AUTH_REQ_NO_MITM_NO_BOND	Protection against MITM not implemented. No bonding performed.
		RBLE_AUTH_REQ_NO_MITM_BOND	Protection against MITM not implemented. Bonding performed.
		RBLE_AUTH_REQ_MITM_NO_BOND	Protection against MITM implemented. No bonding performed.
		RBLE_AUTH_REQ_MITM_BOND	Protection against MITM implemented. Bonding performed.
	max_key_size	Maximum encryption key size	
	ikeys	Type of key distributed by the initiator (select by using OR) RBLE_KEY_DIST_NONE: No key distributed. RBLE_KEY_DIST_ENCKEY: LTK distributed. RBLE_KEY_DIST_IDKEY: IRK distributed. RBLE_KEY_DIST_SIGNKEY: CSRK distributed.	
	rkeys	Type of key distributed by the responder (select by using OR) RBLE_KEY_DIST_NONE: No key distributed. RBLE_KEY_DIST_ENCKEY: LTK distributed. RBLE_KEY_DIST_IDKEY: IRK distributed. RBLE_KEY_DIST_SIGNKEY: CSRK distributed.	

Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.24 RBLE_GAP_Change_Connection_Param

RBLE_STATUS RBLE_GAP_Change_Connection_Param(uint16_t conhdl, uint16_t result, RBLE_CONN_PARAM *conn_param, uint8_t role)

This function changes a connection parameter for an established link. This function is used for the cases below and the result is reported by different events according to the purpose.

1. The master uses this function to change a connection parameter. The result is reported by using the connection parameter change completion event RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_COMP.
2. The slave uses this function to request the master to change a connection parameter. The result is reported by using the connection parameter change request response notification event RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_RESPONSE.
3. The master uses this function to respond to the request from the slave to change a connection parameter. The result is reported by using the connection parameter change completion event RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_COMP.

Parameters:

conhdl	Connection handle	
result	Response to the request to change a connection parameter (0x0000: Acceptable, 0x0001: Not acceptable) * This parameter is only available for case 3 above.	
**conn_param*	intv_min	Minimum connection interval N = 0x0006 to 0x0C80 (Time = N x 1.25 ms (7.5 ms to 4.0 sec.))
	intv_max	Maximum connection interval N = 0x0006 to 0x0C80 (Time = N x 1.25 ms (7.5 ms to 4.0 sec.))
	latency	Connection slave latency (0x0000 to 0x01F3)
	time_out	Supervision timeout N = 0x000A to 0x0C80 (Time = N x 10 ms (100 ms to 32 sec.))
role	Role of the local device (RBLE_MASTER: Master, RBLE_SLAVE: Slave)	

Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.25 RBLE_GAP_Channel_Map_Req

RBLE_STATUS RBLE_GAP_Channel_Map_Req(bool update_map, uint16_t conhdl, RBLE_LE_CHNL_MAP *chmap)	
This function sets up or acquires the data channel map. The result is reported by using the channel map setup/acquisition completion event RBLE_GAP_EVENT_CHANNEL_MAP_REQ_COMP. * Channel map setting is only available for the Master role.	
Parameters:	
update_map	Update map flag (TRUE: Set up the channel map, FALSE: Acquire the channel map)
conhdl	Connection handle * This parameter is only available for acquiring the channel map.
**chmap*	37-bit value that indicates classification for data channels 0 to 36 (0: Bad, 1: Unknown) * This parameter is only available for setting up the channel map.
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.26 RBLE_GAP_Read_RSSI

RBLE_STATUS RBLE_GAP_Read_RSSI(uint16_t conhdl)	
This function acquires the RSSI received from the specified remote device. The result is reported by using the RSSI acquisition completion event RBLE_GAP_EVENT_READ_RSSI_COMP.	
Parameters:	
conhdl	Connection handle
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.2.27 RBLE_GAP_Authorized_Ind

RBLE_STATUS RBLE_GAP_Authorized_Ind (uint16_t conhdl)	
This function indicates that the specified remote device has been authorized by user. If require authorized to connect with the specified remote device, please confirm to the user at the time of connection is completed, and call this function.	
Parameters:	
conhdl	Connection handle
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

5.3 Events

Table 5-2 shows the events defined for the GAP of rBLE and the following sections describe the events in detail.

Table 5-2 Events Defined for the GAP

RBLE_GAP_EVENT_RESET_RESULT	Reset completion event
RBLE_GAP_EVENT_SET_NAME_COMP	Device name setup completion event
RBLE_GAP_EVENT_OBSERVATION_ENABLE_COMP	Observation enable event
RBLE_GAP_EVENT_OBSERVATION_DISABLE_COMP	Observation disable event
RBLE_GAP_EVENT_BROADCAST_ENABLE_COMP	Broadcast enable event
RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP	Broadcast disable event
RBLE_GAP_EVENT_SET_BONDING_MODE_COMP	Bonding mode setup event
RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP	Security mode setup event
RBLE_GAP_EVENT_GET_DEVICE_INFO_COMP	Device information acquisition completion event
RBLE_GAP_EVENT_GET_WHITE_LIST_SIZE_COMP	Local device White List size read completion event
RBLE_GAP_EVENT_ADD_TO_WHITE_LIST_COMP	White List device add completion event
RBLE_GAP_EVENT_DEL_FROM_WHITE_LIST_COMP	White List device delete completion event
RBLE_GAP_EVENT_GET_REMOTE_DEVICE_INFO_COMP	Remote device information acquisition completion event
RBLE_GAP_EVENT_GET_REMOTE_DEVICE_NAME_COMP	Remote device name acquisition completion event
RBLE_GAP_EVENT_DEVICE_SEARCH_COMP	Device search command completion event
RBLE_GAP_EVENT_DEVICE_SEARCH_RESULT_IND	Device search result notification event
RBLE_GAP_EVENT_RPA_RESOLVED	Resolvable Private Address resolution completion event
RBLE_GAP_EVENT_SET_RANDOM_ADDRESS_COMP	Random address setup command completion event
RBLE_GAP_EVENT_SET_PRIVACY_FEATURE_COMP	Privacy feature setup completion event
RBLE_GAP_EVENT_CONNECTION_COMP	LE link connection event
RBLE_GAP_EVENT_CONNECTION_CANCEL_COMP	LE link connection cancel completion event
RBLE_GAP_EVENT_DISCONNECT_COMP	LE link disconnection completion event
RBLE_GAP_EVENT_ADVERTISING_REPORT_IND	Advertising report and data report notification event
RBLE_GAP_EVENT_BONDING_COMP	Bonding completion event
RBLE_GAP_EVENT_BONDING_REQ_IND	Peer device bonding request notification event
RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_REQ_IND	Connection parameter change request notification event
RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_COMP	Connection parameter change completion event
RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_RESPONSE	Connection parameter change request response notification event
RBLE_GAP_EVENT_CHANNEL_MAP_REQ_COMP	Channel map setup/acquisition completion event
RBLE_GAP_EVENT_READ_RSSI_COMP	RSSI acquisition completion event
RBLE_GAP_EVENT_WR_CHAR_IND	GAP characteristics write indication event
RBLE_GAP_EVENT_COMMAND_DISALLOWED_IND	GAP Command disallowed notification event

5.3.1 RBLE_GAP_EVENT_RESET_RESULT

RBLE_GAP_EVENT_RESET_RESULT	
This event reports the result of executing a GAP reset (RBLE_GAP_Reset).	
Parameters:	
status	Result of executing a GAP reset (See *3.2*, *Declaration of enumerated type for rBLE status*.)
rBLE_major_ver	rBLE major version
rBLE_minor_ver	rBLE minor version

5.3.2 RBLE_GAP_EVENT_SET_NAME_COMP

RBLE_GAP_EVENT_SET_NAME_COMP	
This event reports the result of setting the local device name (RBLE_GAP_Set_Name).	
Parameters:	
status	Result of setting the local device name (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.3 RBLE_GAP_EVENT_OBSERVATION_ENABLE_COMP

RBLE_GAP_EVENT_OBSERVATION_ENABLE_COMP	
This event reports the result of enabling observation (RBLE_GAP_Observation_Enable).	
Parameters:	
status	Result of enabling observation (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.4 RBLE_GAP_EVENT_OBSERVATION_DISABLE_COMP

RBLE_GAP_EVENT_OBSERVATION_DISABLE_COMP	
This event reports the result of disabling observation (RBLE_GAP_Observation_Disable).	
Parameters:	
status	Result of disabling observation (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.5 RBLE_GAP_EVENT_BROADCAST_ENABLE_COMP

RBLE_GAP_EVENT_BROADCAST_ENABLE_COMP	
This event reports the result of enabling a broadcast (RBLE_GAP_Broadcast_Enable).	
Parameters:	
status	Result of enabling broadcast (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.6 RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP

RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP		
This event reports the result of disabling a broadcast (RBLE_GAP_Broadcast_Disable).		
Parameters:		
	status	Result of disabling broadcast (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.7 RBLE_GAP_EVENT_SET_BONDING_MODE_COMP

RBLE_GAP_EVENT_SET_BONDING_MODE_COMP		
This event reports the result of setting up the bonding mode (RBLE_GAP_Set_Bonding_Mode).		
Parameters:		
	status	Result of setting up bonding mode (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.8 RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP

RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP		
This event reports the result of setting up the security mode (RBLE_GAP_Set_Security_Request).		
Parameters:		
	status	Result of setting up security mode (See *3.2*, *Declaration of enumerated type for rBLE status*.)
	sec	Current security mode

5.3.9 RBLE_GAP_EVENT_GET_DEVICE_INFO_COMP

RBLE_GAP_EVENT_GET_DEVICE_INFO_COMP			
This event reports completion of acquiring local device information.			
Parameters:			
	status	Result of acquiring local device information (See *3.2*, *Declaration of enumerated type for rBLE status*.)	
	addr	BD address of the local device	
	ver_info	*hci_ver*	HCI version
		lmp_ver	LMP version
		host_ver	Host version
		hci_subver	HCI subversion
		lmp_subver	LMP subversion
		host_subver	Host subversion
		company_id	Company ID Bluetooth SIG assigned numbers. See https://www.bluetooth.org/Technical/AssignedNumbers/home.htm.

5.3.10 RBLE_GAP_EVENT_GET_WHITE_LIST_SIZE_COMP

RBLE_GAP_EVENT_GET_WHITE_LIST_SIZE_COMP	
This event reports the result of reading the local device White List size (RBLE_GAP_Get_White_List_Size).	
Parameters:	
status	Result of reading local device White List size (See *3.2*, *Declaration of enumerated type for rBLE status*.)
wlist_size	Local device White List size * This parameter becomes invalid if an error occurs in White List size read processing.

5.3.11 RBLE_GAP_EVENT_ADD_TO_WHITE_LIST_COMP

RBLE_GAP_EVENT_ADD_TO_WHITE_LIST_COMP	
This event reports the result of adding the specified device to the White List (RBLE_GAP_Add_To_White_List).	
Parameters:	
status	Result of adding specified device to the White List (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.12 RBLE_GAP_EVENT_DEL_FROM_WHITE_LIST_COMP

RBLE_GAP_EVENT_DEL_FROM_WHITE_LIST_COMP	
This event reports the result of deleting the specified device from the White List (RBLE_GAP_Del_From_White_List).	
Parameters:	
status	Result of deleting specified device from the White List (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.13 RBLE_GAP_EVENT_GET_REMOTE_DEVICE_NAME_COMP

RBLE_GAP_EVENT_GET_REMOTE_DEVICE_NAME_COMP	
This event reports the result of acquiring the remote device name (RBLE_GAP_Get_Remote_Device_Name).	
Parameters:	
status	Result of acquiring remote device name (See *3.2*, *Declaration of enumerated type for rBLE status*.)
bd_name	Remote device name * This parameter becomes invalid if an error occurs in remote device name acquisition processing.

5.3.14 RBLE_GAP_EVENT_GET_REMOTE_DEVICE_INFO_COMP

RBLE_GAP_EVENT_GET_REMOTE_DEVICE_INFO_COMP	
This event reports the result of acquiring remote device information (RBLE_GAP_Get_Remote_Device_Info).	
Parameters:	
status	Result of acquiring remote device information (See *3.2*, *Declaration of enumerated type for rBLE status*.)
conhdl	Connection handle
vers	LMP version * This parameter becomes invalid if an error occurs in remote device information acquisition processing.
compid	Company ID * This parameter becomes invalid if an error occurs in remote device information acquisition processing.
subvers	LMP subversion * This parameter becomes invalid if an error occurs in remote device information acquisition processing.
feats_used	LE features supported by the remote device Bit 0: LE encryption (1: Supported, 0: Not supported) Other bits are reserved for future use. * This parameter becomes invalid if an error occurs in remote device information acquisition processing.

5.3.15 RBLE_GAP_EVENT_DEVICE_SEARCH_COMP

RBLE_GAP_EVENT_DEVICE_SEARCH_COMP	
This event reports completion of searching for peripheral devices (RBLE_GAP_Device_Search).	
Parameters:	
status	Result of searching for peripheral devices (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.16 RBLE_GAP_EVENT_DEVICE_SEARCH_RESULT_IND

RBLE_GAP_EVENT_DEVICE_SEARCH_RESULT_IND			
This event indicates the result of searching for peripheral devices.			
Parameters:			
	adv_resp	*evt_type*	Advertising event type 0x00: Connectable undirected advertising 0x01: Connectable directed advertising 0x02: Scannable undirected advertising 0x03: Non connectable undirected advertising 0x04: Scan Response
		adv_addr_type	Advertiser address type Public address: RBLE_ADDR_PUBLIC Random address: RBLE_ADDR_RAND
		adv_addr	BD address of advertiser
		data_len	Advertising data length
		data[RBLE_ADV_DATA_LEN]	Advertising or scan response data Note: For details about the advertising and scan response data formats, see *Bluetooth Low Energy Protocol Stack User's Manual*.
		rssi	RSSI when advertising data is received

5.3.17 RBLE_GAP_EVENT_RPA_RESOLVED

RBLE_GAP_EVENT_RPA_RESOLVED		
This event indicates the result of Resolvable Private Address resolution.		
Parameters:		
	res_addr	Resolved BD address
	res_addr_type	Resolved address type • Public address: RBLE_ADDR_PUBLIC • Random address: RBLE_ADDR_RAND
	addr	Previous BD address
	addr_type	Previous address type • Public address: RBLE_ADDR_PUBLIC • Random address: RBLE_ADDR_RAND

5.3.18 RBLE_GAP_EVENT_SET_RANDOM_ADDRESS_COMP

RBLE_GAP_EVENT_SET_RANDOM_ADDRESS_COMP		
This event reports the result of setting the random address (RBLE_GAP_Set_Random_Address).		
Parameters:		
	status	Result of setting random address (See *3.2*, *Declaration of enumerated type for rBLE status*.)
	addr	Random address set

5.3.19 RBLE_GAP_EVENT_SET_PRIVACY_FEATURE_COMP

RBLE_GAP_EVENT_SET_PRIVACY_FEATURE_COMP	
This event reports the result of setting the privacy feature for the local device (RBLE_GAP_Set_Privacy_Feature).	
Parameters:	
status	Result of setting privacy feature for local device (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.20 RBLE_GAP_EVENT_CONNECTION_COMP

RBLE_GAP_EVENT_CONNECTION_COMP		
This event reports the result of connecting an LE link.		
Parameters:		
connect_info	*status*	Result of connecting LE link (See *3.2*, *Declaration of enumerated type for rBLE status*.) * The following parameters become invalid if an error occurs in connection processing.
	role	Role of the local device (RBLE_MASTER: Master, RBLE_SLAVE: Slave)
	conhdl	Connection handle
	peer_addr_type	Peer device address type Public address: RBLE_ADDR_PUBLIC Random address: RBLE_ADDR_RAND
	peer_addr	BD address of peer device
	idx	Connection index
	con_interval	Connection interval
	con_latency	Slave latency
	sup_to	Supervision timeout
	clk_accuracy	Master clock accuracy (See *5.1*, *Declaration of enumerated type for clock accuracy*.)

5.3.21 RBLE_GAP_EVENT_CONNECTION_CANCEL_COMP

RBLE_GAP_EVENT_CONNECTION_CANCEL_COMP	
This event reports the result of canceling an LE link connection. (RBLE_GAP_Connection_Cancel).	
Parameters:	
status	Result of canceling LE link connection (See *3.2*, *Declaration of enumerated type for rBLE status*.)

5.3.22 RBLE_GAP_EVENT_DISCONNECT_COMP

RBLE_GAP_EVENT_DISCONNECT_COMP	
This event reports the result of disconnecting an LE link.	
Parameters:	
reason	Reason for disconnection (See *3.2*, *Declaration of enumerated type for rBLE status*.)
status	Result of disconnection (See *3.2*, *Declaration of enumerated type for rBLE status*.)
conhdl	Connection handle

5.3.23 RBLE_GAP_EVENT_ADVERTISING_REPORT_IND

RBLE_GAP_EVENT_ADVERTISING_REPORT_IND			
This event indicates an advertising report.			
Parameters:			
evt	*adv_rep*	*evt_type*	Advertising event type 0x00: Connectable undirected advertising 0x01: Connectable directed advertising 0x02: Scannable undirected advertising 0x03: Non connectable undirected advertising 0x04: Scan Response
		adv_addr_type	Advertiser address type Public address: RBLE_ADDR_PUBLIC Random address: RBLE_ADDR_RAND
		adv_addr	BD address of advertiser
		data_len	Advertising data length
		data[RBLE_ADV_DATA_LEN]	Advertising or scan response data Note: For details about the Advertising and Scan Response data formats, see the *Bluetooth Low Energy Protocol Stack User's Manual*.
		rssi	RSSI when advertising data is received

5.3.24 RBLE_GAP_EVENT_BONDING_COMP

RBLE_GAP_EVENT_BONDING_COMP	
This event reports the result of bonding.	
Parameters:	
conhdl	Connection handle
idx	Connection index
status	Result of bonding (See *3.2*, *Declaration of enumerated type for rBLE status*.)
key_size	Encryption key size
sec_prop	Key security property RBLE_SMP_KSEC_NONE: No security RBLE_SMP_KSEC_UNAUTH_NO_MITM: MITM protection not implemented. RBLE_SMP_KSEC_AUTH_MITM: MITM protection implemented.

5.3.25 RBLE_GAP_EVENT_BONDING_REQ_IND

RBLE_GAP_EVENT_BONDING_REQ_IND				
This event indicates a bonding request from a remote device. To respond to this request, use the remote device bonding request response function RBLE_GAP_Bonding_Response.				
Parameters:				
	bonding_req	*addr*	BD address of the remote device for which to request bonding	
		index	Connection index	
		auth_req	RBLE_AUTH_REQ_NO_MITM_NO_BOND	Protection against MITM not implemented. No bonding performed.
			RBLE_AUTH_REQ_NO_MITM_BOND	Protection against MITM not implemented. Bonding performed.
			RBLE_AUTH_REQ_MITM_NO_BOND	Protection against MITM implemented. No bonding performed.
			RBLE_AUTH_REQ_MITM_BOND	Protection against MITM implemented. Bonding performed.
		io_cap	RBLE_IO_CAP_DISPLAY_ONLY	Input: No Output: Display
			RBLE_IO_CAP_DISPLAY_YES_NO	Input: Yes/No Output: Display
			RBLE_IO_CAP_KB_ONLY	Input: Keyboard Output: No
			RBLE_IO_CAP_NO_INPUT_NO_OUTPUT	Input: No Output: No
			RBLE_IO_CAP_KB_DISPLAY	Input: Keyboard Output: Display
		oob_data_flg	RBLE_OOB_AUTH_DATA_NOT_PRESENT	OOB data not present
			RBLE_OOB_AUTH_DATA_FROM_REMOTE_DEV_PRESENT	OOB data from a remote device present
		max_enc_size	Maximum encryption key size	
		ikey_dist	Type of key distributed by the initiator (select by using OR) RBLE_KEY_DIST_NONE: No key distributed. RBLE_KEY_DIST_ENCKEY: LTK distributed. RBLE_KEY_DIST_IDKEY: IRK distributed. RBLE_KEY_DIST_SIGNKEY: CSRK distributed.	
		rkey_dist	Type of key distributed by the responder (select by using OR) RBLE_KEY_DIST_NONE: No key distributed. RBLE_KEY_DIST_ENCKEY: LTK distributed. RBLE_KEY_DIST_IDKEY: IRK distributed. RBLE_KEY_DIST_SIGNKEY: CSRK distributed.	

5.3.26 RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_REQ_IND

RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_REQ_IND			
This event indicates a request for changing a connection parameter from a remote peripheral device. To respond to this request, use the connection parameter change function RBLE_GAP_Change_Connection_Param.			
Parameters:			
	conhdl	Connection handle	
	conn_param	*intv_min*	Minimum connection interval
		intv_max	Maximum connection interval
		latency	Connection slave latency
		time_out	Supervision timeout

5.3.27 RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_COMP

RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_COMP		
This event reports the result of changing a connection parameter.		
Parameters:		
	status	Result of changing connection parameter (See *3.2*, *Declaration of enumerated type for rBLE status*.)
	con_interval	Connection interval
	con_latency	Connection slave latency
	sup_to	Supervision timeout

5.3.28 RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_RESPONSE

RBLE_GAP_EVENT_CHANGE_CONNECTION_PARAM_RESPONSE		
This event reports the response from the master to the request to change a connection parameter.		
Parameters:		
	status	Result of request for changing the connection parameter (See *3.2*, *Declaration of enumerated type for rBLE status*.)
	result	Result of request to change a connection parameter 0x0000: Changing the connection parameter accepted 0x0001: Changing the connection parameter rejected
	conhdl	Connection handle

5.3.29 RBLE_GAP_EVENT_CHANNEL_MAP_REQ_COMP

RBLE_GAP_EVENT_CHANNEL_MAP_REQ_COMP		
This function reports the result of setting or acquiring the data channel map.		
Parameters:		
	conhdl	Connection handle
	status	Result of setting or acquiring data channel map (See *3.2*, *Declaration of enumerated type for rBLE status*.)
	chmap	37-bit value that indicates classification for data channels 0 to 36 (0: unused, 1: used) * This parameter is only available for acquiring the channel map.

5.3.30 RBLE_GAP_EVENT_READ_RSSI_COMP

RBLE_GAP_EVENT_READ_RSSI_COMP	
This event reports the result of acquiring the RSSI from the specified remote device.	
Parameters:	
conhdl	Connection handle
status	Result of acquiring RSSI (See *3.2*, *Declaration of enumerated type for rBLE status*.)
rssi	RSSI value from the specified remote device * This parameter becomes invalid if an error occurs in RSSI acquisition processing.

5.3.31 RBLE_GAP_EVENT_WR_CHAR_IND

RBLE_GAP_EVENT_WR_CHAR_IND		
This event notifies the reception of the write GAP characteristic value from a remote device..		
Parameters:		
conhdl	Connection handle	
type	GAP characteristic code of written by a remote device. - RBLE_GAP_WR_CHAR_NAME The parameter *param* is stored in the format *name*. - RBLE_GAP_WR_CHAR_APPEARANCE The parameter *param* is stored in the format *appearance*.	
param	*name*	Device name characteristic
	appearance	Appearance characteristic

5.3.32 RBLE_GAP_EVENT_COMMAND_DISALLOWED_IND

RBLE_GAP_EVENT_COMMAND_DISALLOWED_IND	
This event indicates that a GAP command was disallowed.	
Parameters:	
status	Result of command execution (See *3.2*, *Declaration of enumerated type for rBLE status*.)
opcode	Opcode of the disallowed command

6. Security Manager

This section describes the APIs related to security features such as pairing, encryption, and data signing.

6.1 Definitions

This section describes the definitions used by the APIs related to security features such as pairing, encryption, and data signing.

- Declaration of enumerated type for SM event types

```
enum RBLE_SM_EVENT_TYPE_enum {
    RBLE_SM_EVENT_SET_CNF = 1,                  Key setup completion event
                                                (Parameter: set_conf)
    RBLE_SM_ENC_START_IND,                      Encryption start notification event
                                                (Parameter: sec_start)
    RBLE_SM_TK_REQ_IND,                         TK request notification event
                                                (Parameter: tk_req)
    RBLE_SM_LTK_REQ_IND,                        LTK (for key distribution) request
                                                notification event
                                                (Parameter: ltk_req)
    RBLE_SM_LTK_REQ_FOR_ENC_IND,                LTK (for encryption) request
                                                notification event
                                                (Parameter: ltk_req_for_enc)
    RBLE_SM_IRK_REQ_IND,                        IRK request notification event
                                                (Parameter: irk_req)
    RBLE_SM_CSRK_REQ_IND,                       CSRK request notification event
                                                (Parameter: csrk_req)
    RBLE_SM_KEY_IND,                            Key notification event
                                                (Parameter: key_ind)
    RBLE_SM_CHK_BD_ADDR_REQ,                    BD address check request event
                                                (Parameter: chk_bdaddr)
    RBLE_SM_TIMEOUT_EVT,                        SM processing timeout notification
                                                event
                                                (Parameter: timeout_evt)
    RBLE_SM_EVENT_COMMAND_DISALLOWED_IND        SM command disallowed notification event
                                                (Parameter: cmd_disallowed_ind)
};
```

- Declaration of data type for SM event types

```
typedef uint8_t RBLE_SM_EVENT_TYPE;
```

- Declaration of data type for SM event callback function

```
typedef void ( *RBLE_SM_EVENT_HANDLER )( RBLE_SM_EVENT *event );
```

- Declaration of enumerated type for key distribution flag

```
enum RBLE_SMP_KEY_DIST_FLAG_enum {
    RBLE_SMP_KDIST_NONE            = 0x00,          Distribute no key.
    RBLE_SMP_KDIST_ENCKEY          = 0x01,          Distribute LTK.
    RBLE_SMP_KDIST_IDKEY           = 0x02,          Distribute IRK.
    RBLE_SMP_KDIST_SIGNKEY         = 0x04           Distribute CSRK.
};
```

- Declaration of enumerated type for security property of distributed key

```
enum RBLE_SMP_KSEC_enum {
    RBLE_SMP_KSEC_NONE                = 0x00,       No security
    RBLE_SMP_KSEC_UNAUTH_NO_MITM,                   Unauthenticated, no MITM
                                                    protection
    RBLE_SMP_KSEC_AUTH_MITM                         Authenticated, MITM
                                                    protection
};
```

- Declaration of enumerated type for BD address check request response

```
enum RBLE_SMP_CHK_BD_REQ_RSP_enum {
    RBLE_SMP_SEC_NONE                 = 0x00,       No security
    RBLE_SMP_UNAUTHENTICATED          = 0x01,       Unauthenticated pairing
                                                    performed
    RBLE_SMP_AUTHENTICATED            = 0x02,       Authenticated pairing
                                                    performed
    RBLE_SMP_AUTHORIZED               = 0x04,       Authorized
    RBLE_SMP_BONDED                   = 0x08        Bonded
};
```

- Declaration of security key structure

```
typedef struct RBLE_KEY_VALUE_t{
    uint8_t    key[RBLE_KEY_LEN];                   Key
}RBLE_KEY_VALUE;
```

- Declaration of random number structure

```
typedef struct RBLE_RAND_NB_t{
    uint8_t    nb[RBLE_RAND_NB_LEN];                Random number (Rand)
}RBLE_RAND_NB;
```

- SM event parameter structure

```
typedef struct RBLE_SM_EVENT_t {
    RBLE_SM_EVENT_TYPE     type;                    SM event type
    uint8_t                reserved;                Reserved
    union Event_Parameter_u {
```

Key setup completion event

```
struct RBLE_EVT_SM_Set_Cnf_t{
    RBLE_STATUS    status;                          Status
    uint8_t        key_code;                        Key type
}set_conf;
```

Encryption start notification event

```
struct RBLE_EVT_SM_Sec_Start_t{
    uint8_t        idx;                             Connection index
    RBLE_STATUS    status;                          Status
    uint8_t        key_size;                        Key size
    uint8_t        sec_prop;                        Security property
    uint8_t        bonded;                          Bonding status flag
    uint8_t        reserved;
}sec_start;
```

TK request notification event

```
struct RBLE_EVT_SM_Tk_Req_t{
    uint8_t        idx;                             Connection index
    uint8_t        oob_en;                          OOB enable flag
    uint8_t        disp_en;                         TK display flag
}tk_req;
```

LTK (for key distribution) request notification event

```
struct RBLE_EVT_SM_Ltk_Req_For_Enc_t{
    uint8_t        idx;                             Connection index
    uint8_t        auth_req;                        Authentication requirement
}ltk_req;
```

LTK (for encryption) request notification event

```
struct RBLE_EVT_SM_Ltk_Req_t{
    uint8_t        idx;                             Connection index
    uint8_t        auth_req;                        Authentication requirement
    uint16_t           ediv;                        EDIV
    RBLE_RAND_NB       nb;                          Rand
}ltk_req_for_enc;
```

IRK request notification event

```
struct RBLE_EVT_SM_Irk_Req_t{
    uint8_t        idx;                             Connection index
}irk_req;
```

CSRK request notification event

```
struct RBLE_EVT_SM_Csrk_Req_t{
    uint8_t        idx;                          Connection index
    RBLE_BD_ADDR   addr;                         Device address
    uint8_t        reserved;                     Reserved
    uint32_t       signcnt;                      Sign counter value
}csrk_req;
```

Key notification event

```
struct RBLE_EVT_SM_Key_t{
    uint8_t            idx;                      Connection index
    uint8_t            key_code;                 Key type
    uint16_t           ediv;                     EDIV
    RBLE_RAND_NB       nb;                       Rand
    RBLE_KEY_VALUE     ltk;                      Key value
}key_ind;
```

BD address check request event

```
struct RBLE_EVT_SM_Chk_Bd_Addr_Req_t{
    uint8_t        idx;                          Connection index
    uint8_t        type;                         Address type
    RBLE_BD_ADDR   addr;                         Device address
}chk_bdaddr;
```

SM processing timeout notification event

```
struct RBLE_EVT_SM_Timeout_Evt_t{
    uint8_t        idx;                          Connection index
}timeout_evt;
```

SM command disallowed notification event

```
struct RBLE_EVT_SM_Command_Disallowed_Ind_t{
    RBLE_STATUS    status;                       Status
    uint8_t        reserved;                     Reserved
    uint16_t       opcode;                       Opcode
}cmd_disallowed_ind;
} param;
} RBLE_SM_EVENT;
```

6.2 Functions

Table 6-1 shows the API functions defined for the SM of rBLE and the following sections describe the API functions in detail.

Table 6-1 API Functions Used by the SM

RBLE_SM_Set_Key	Sets the key.
RBLE_SM_Start_Enc	Starts encryption.
RBLE_SM_Tk_Req_Resp	Responds to a TK request.
RBLE_SM_Ltk_Req_Resp	Responds to an LTK request.
RBLE_SM_Irk_Req_Resp	Responds to an IRK request.
RBLE_SM_Csrk_Req_Resp	Responds to a CSRK request.
RBLE_SM_Chk_Bd_Addr_Req_Resp	Responds to a BD address check request.

6.2.1 RBLE_SM_Set_Key

RBLE_STATUS RBLE_SM_Set_Key(uint8_t Key_code, RBLE_KEY_VALUE *Key_Value)		
This function sets a key held by an application to the SM. The result is reported by using the key setup completion event RBLE_SM_EVENT_SET_CNF.		
Parameters:		
Key_code	RBLE_SMP_KDIST_IDKEY	Sets an IRK (Identity Resolving Key).
	RBLE_SMP_KDIST_SIGNKEY	Sets a CSRK (Connection Signature Resolving Key).
**Key_Value*	Pointer to the location in which the key to be set is stored	
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

6.2.2 RBLE_SM_Start_Enc

RBLE_STATUS RBLE_SM_Start_Enc(uint8_t idx, uint8_t auth_req, uint16_t ediv, RBLE_RAND_NB *rand_nb, RBLE_KEY_VALUE *ltk)		
This function starts encryption of the link with a remote device in accordance with the specified parameters. The result is indicated by using the encryption start notification event RBLE_SM_ENC_START_IND.		
Parameters:		
idx	Connection index	
auth_req	RBLE_AUTH_REQ_NO_MITM_NO_BOND	Protection against MITM not implemented. No bonding performed.
	RBLE_AUTH_REQ_NO_MITM_BOND	Protection against MITM not implemented. Bonding performed.
	RBLE_AUTH_REQ_MITM_NO_BOND	Protection against MITM implemented. No bonding performed.
	RBLE_AUTH_REQ_MITM_BOND	Protection against MITM implemented. Bonding performed.
ediv	EDIV	
**rand_nb*	Pointer to the location in which Rand is stored	
**ltk*	Pointer to the location in which the LTK is stored	
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

6.2.3 RBLE_SM_Tk_Req_Resp

RBLE_STATUS RBLE_SM_Tk_Req_Resp(uint8_t idx, uint8_t status, RBLE_KEY_VALUE *tk)		
This function responds to a TK request.		
Parameters:		
	idx	Connection index
	status	Responds to a TK request. RBLE_OK: TK Other than above: No TK * If status is not RBLE_OK, the following parameter is invalid.
	**tk*	Pointer to the location in which the TK is stored
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

6.2.4 RBLE_SM_Ltk_Req_Resp

RBLE_STATUS RBLE_SM_Ltk_Req_Resp(uint8_t idx, uint8_t status, uint8_t sec_prop, uint16_t ediv, RBLE_RAND_NB *nb, RBLE_KEY_VALUE *ltk)		
This function responds to an LTK request.		
Parameters:		
	idx	Connection index
	status	Responds to an LTK request. RBLE_OK: LTK Other than above: No LTK * If status is not RBLE_OK, the following parameter is invalid.
	sec_prop	LTK security property RBLE_SMP_KSEC_NONE: No security requirements RBLE_SMP_KSEC_UNAUTH_NO_MITM: Unauthenticated, no MITM protection RBLE_SMP_KSEC_AUTH_MITM: Authenticated, MITM protection
	ediv	EDIV
	**nb*	Pointer to the location in which Rand is stored
	**ltk*	Pointer to the location in which the LTK is stored
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

6.2.5 RBLE_SM_Irk_Req_Resp

RBLE_STATUS RBLE_SM_Irk_Req_Resp(uint8_t idx, uint8_t status, RBLE_BD_ADDR *orig_addr, RBLE_KEY_VALUE *irk , uint8_t lk_sec_status)	
This function responds to the request for an IRK used to resolve the address.	
Parameters:	
idx	Connection index
status	Responds to an IRK request. RBLE_OK: IRK Other than above: No IRK * If status is not RBLE_OK, the following parameters are invalid.
**orig_addr*	Original BD address of the remote device
**irk*	IRK of the remote device
lk_sec_status	Security status of the remote device. (select by using OR) RBLE_SMP_SEC_NONE: No security RBLE_SMP_UNAUTHENTICATED: Unauthenticated pairing performed RBLE_SMP_AUTHENTICATED: Authenticated pairing performed RBLE_SMP_AUTHORIZED: Authorized RBLE_SMP_BONDED: Bonded
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

6.2.6 RBLE_SM_Csrk_Req_Resp

RBLE_STATUS RBLE_SM_Csrk_Req_Resp(uint8_t idx, uint8_t status, RBLE_KEY_VALUE *csrk, uint8_t lk_sec_status)	
This function responds to a CSRK request.	
Parameters:	
idx	Connection index
status	Responds to a CSRK request. RBLE_OK: CSRK Other than above: No CSRK * If status is not RBLE_OK, the following parameter is invalid.
**csrk*	Pointer to the location in which the CSRK is stored
lk_sec_status	Security status of the remote device. (select by using OR) RBLE_SMP_SEC_NONE: No security RBLE_SMP_UNAUTHENTICATED: Unauthenticated pairing performed RBLE_SMP_AUTHENTICATED: Authenticated pairing performed RBLE_SMP_AUTHORIZED: Authorized RBLE_SMP_BONDED: Bonded
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

6.2.7 RBLE_SM_Chk_Bd_Addr_Req_Resp

RBLE_STATUS RBLE_SM_Chk_Bd_Addr_Req_Resp (uint8_t idx, uint8_t type, uint8_t found_flag, uint8_t lk_sec_status, RBLE_BD_ADDR *addr)	
This function responds to a request for checking a BD address.	
Parameters:	
idx	Connection index
type	Remote device address type Public address: RBLE_ADDR_PUBLIC Random address: RBLE_ADDR_RAND
found_flag	Flag indicating BD address information flag (TRUE: Has information, FALSE: Does not have information)
lk_sec_status	Security status of the remote device. (select by using OR) RBLE_SMP_SEC_NONE: No security RBLE_SMP_UNAUTHENTICATED: Unauthenticated pairing performed RBLE_SMP_AUTHENTICATED: Authenticated pairing performed RBLE_SMP_AUTHORIZED: Authorized RBLE_SMP_BONDED: Bonded
**addr*	BD address of the remote device
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

6.3 Events

Table 6-2 shows the events defined for the SM of rBLE and the following sections describe the events in detail.

Table 6-2 Events Defined for the SM

RBLE_SM_EVENT_SET_CNF	Key setup completion event
RBLE_SM_ENC_START_IND	Encryption start notification event
RBLE_SM_TK_REQ_IND	TK request notification event
RBLE_SM_LTK_REQ_IND	LTK (for key distribution) request notification event
RBLE_SM_LTK_REQ_FOR_ENC_IND	LTK (for encryption) request notification event
RBLE_SM_IRK_REQ_IND	IRK request notification event
RBLE_SM_CSRK_REQ_IND	CSRK request notification event
RBLE_SM_KEY_IND	Key notification event
RBLE_SM_CHK_BD_ADDR_REQ	BD address check request event
RBLE_SM_TIMEOUT_EVT	SM processing timeout notification event
RBLE_SM_EVENT_COMMAND_DISALLOWED_IND	SM command disallowed notification event

6.3.1 RBLE_SM_EVENT_SET_CNF

RBLE_SM_EVENT_SET_CNF	
This event reports the result of setting up the specified key (RBLE_SM_Set_Key).	
Parameters:	
status	Result of setting up specified key (See *3.2*, *Declaration of enumerated type for rBLE status*.)
key_code	Key being set RBLE_SMP_KDIST_IDKEY: IRK RBLE_SMP_KDIST_SIGNKEY: CSRK

6.3.2 RBLE_SM_ENC_START_IND

RBLE_SM_ENC_START_IND	
This event indicates the result of starting encryption of a link.	
Parameters:	
idx	Connection index
status	Result of starting encryption of link (See *3.2*, *Declaration of enumerated type for rBLE status*.)
key_size	Encryption key size
sec_prop	Key security property RBLE_SMP_KSEC_NONE: No security requirements RBLE_SMP_KSEC_UNAUTH_NO_MITM: Unauthenticated, no MITM protection RBLE_SMP_KSEC_AUTH_MITM: Authenticated, MITM protection
bonded	Bonding status 0: Unbonded 1: Bonded

6.3.3 RBLE_SM_TK_REQ_IND

RBLE_SM_TK_REQ_IND	
This event indicates a TK request. To respond to this event, use the TK request respond function RBLE_SM_Tk_Req_Resp.	
Parameters:	
idx	Connection index
oob_en	Flag for executing paring OOB (TRUE: Execute OOB pairing, FALSE: Execute pairing using a mechanism other than OOB)
disp_en	Flag indicating whether to display a TK (TRUE: Display, FALSE: Do not display)

6.3.4 RBLE_SM_LTK_REQ_IND

RBLE_SM_LTK_REQ_IND			
This event indicates an LTK request for key distribution phase. To respond to this event, use the LTK request respond function RBLE_SM_Ltk_Req_Resp.			
Parameters:			
	idx	Connection index	
	auth_req	RBLE_AUTH_REQ_NO_MITM_NO_BOND	Protection against MITM not implemented. No bonding performed.
		RBLE_AUTH_REQ_NO_MITM_BOND	Protection against MITM not implemented. Bonding performed.
		RBLE_AUTH_REQ_MITM_NO_BOND	Protection against MITM implemented. No bonding performed.
		RBLE_AUTH_REQ_MITM_BOND	Protection against MITM implemented. Bonding performed.

6.3.5 RBLE_SM_LTK_REQ_FOR_ENC_IND

RBLE_SM_LTK_REQ_FOR_ENC_IND			
This event indicates an LTK request for encryption setup. To respond to this event, use the LTK request respond function RBLE_SM_Ltk_Req_Resp.			
Parameters:			
	idx	Connection index	
	auth_req	Authentication Requirements * This parameter is only valid when local device is Master role.	
		RBLE_AUTH_REQ_NO_MITM_NO_BOND	Protection against MITM not implemented. No bonding performed.
		RBLE_AUTH_REQ_NO_MITM_BOND	Protection against MITM not implemented. Bonding performed.
		RBLE_AUTH_REQ_MITM_NO_BOND	Protection against MITM implemented. No bonding performed.
		RBLE_AUTH_REQ_MITM_BOND	Protection against MITM implemented. Bonding performed.
	ediv	EDIV * This parameter is only valid when local device is Slave role.	
	nb	Rand * This parameter is only valid when local device is Slave role.	

6.3.6 RBLE_SM_IRK_REQ_IND

RBLE_SM_IRK_REQ_IND	
This event indicates a request for the IRK of a remote device. To respond to this event, use the IRK request respond function RBLE_SM_Irk_Req_Resp.	
Parameters:	
idx	Connection index

6.3.7 RBLE_SM_CSRK_REQ_IND

RBLE_SM_CSRK_REQ_IND	
This event indicates a CSRK request. To respond to this event, use the CSRK request respond function RBLE_SM_Csrk_Req_Resp.	
Parameters:	
idx	Connection index
addr	BD address of the remote device
signcnt	Counter of signs included in the signature of received data

6.3.8 RBLE_SM_KEY_IND

RBLE_SM_KEY_IND	
This event indicates the distributed key.	
Parameters:	
idx	Connection index
key_code	Distributed key RBLE_SMP_KDIST_ENCKEY: LTK RBLE_SMP_KDIST_IDKEY: IRK RBLE_SMP_KDIST_SIGNKEY: CSRK
ediv	EDIV * This parameter is only valid when key_code is RBLE_SMP_KDIST_ENCKEY.
nb	Rand * This parameter is only valid when key_code is RBLE_SMP_KDIST_ENCKEY.
ltk	Value of key indicated by key_code

6.3.9 RBLE_SM_CHK_BD_ADDR_REQ

RBLE_SM_CHK_BD_ADDR_REQ	
This function reports a request for checking a BD address. To respond to this event, use the BD address check function RBLE_SM_Chk_Bd_Addr_Req_Resp.	
Parameters:	
idx	Connection index
type	Address type
addr	BD address to be checked

6.3.10 RBLE_SM_TIMEOUT_EVT

RBLE_SM_TIMEOUT_EVT	
This event reports that SM processing timed out.	
Parameters:	
idx	Connection index

6.3.11 RBLE_SM_EVENT_COMMAND_ERROR_IND

RBLE_SM_EVENT_COMMAND_DISALLOWED_IND	
This event indicates that an SM command was disallowed.	
Parameters:	
status	Result of command execution (See *3.2*, *Declaration of enumerated type for rBLE status*.)
opcode	Opcode of the disallowed command

7. Generic Attribute Profile

This section describes the APIs of the General Attribute (GATT) profile. Refer to the Bluetooth Low Energy Protocol Stack User's Manual about the database structure used by the local GATT server.

7.1 Definitions

This section describes the definitions used by the APIs of the GATT profile.

- GATT constant definitions

```
#define RBLE_GATT_MAX_VALUE                 0x18            Maximum size of
                                                            characteristic value
#define RBLE_GATT_MAX_HDL_LIST              0x08            Maximum number of handle
                                                            lists
#define RBLE_GATT_MAX_LONG_VALUE            0x48            Maximum size of long
                                                            characteristic value
#define RBLE_GATT_MAX_NB_HDLS               0x04            Maximum number of handle
                                                            pair
#define RBLE_GATT_16BIT_UUID_OCTET          0x02            16-bit UUID octet
#define RBLE_GATT_3BIT_UUID_OCTET           0x04            32-bit UUID octet
#define RBLE_GATT_128BIT_UUID_OCTET         0x10            128-bit UUID octet
#define RBLE_GATT_MAX_RELIABLE_WRITE_CONTENTS  0x10         Maximum size of reliable
                                                            data write
#define RBLE_GATT_MAX_RELIABLE_WRITE_NUM       0x04         Maximum number of reliable
                                                            data write
```

- Expected response data size on GATT read multiple definitions

```
#define RBLE_GATT_LEN_UNDEF                 0xFF            Variable length
```

- GATT attribute permission definition

```
#define RBLE_GATT_PERM_NONE         0x0000        No permission
#define RBLE_GATT_PERM_RD           0x0001        Readable
#define RBLE_GATT_PERM_RD_UNAUTH    0x0002        Unauthenticated pairing required
                                                  to read
#define RBLE_GATT_PERM_RD_AUTH      0x0004        Authenticated pairing required
                                                  to read
#define RBLE_GATT_PERM_RD_AUTZ      0x0008        Authorization requited to read
#define RBLE_GATT_PERM_WR           0x0010        Writable
#define RBLE_GATT_PERM_WR_UNAUTH    0x0020        Unauthenticated pairing required
                                                  to write
#define RBLE_GATT_PERM_WR_AUTH      0x0040        Authenticated pairing required
                                                  to write
#define RBLE_GATT_PERM_WR_AUTZ      0x0080        Authorization required to write
#define RBLE_GATT_PERM_NI           0x0100        Able to be notified / indicated
#define RBLE_GATT_PERM_NI_UNAUTH    0x0200        Unauthenticated pairing required
                                                  for notification / indication
#define RBLE_GATT_PERM_NI_AUTH      0x0400        Authenticated pairing required
```

```
                                                for notification / indication
#define RBLE_GATT_PERM_NI_AUTZ      0x0800      Authorization required for
                                                notification / indication
#define RBLE_GATT_PERM_EKS          0x1000      Encryption by key of suficient
                                                length Required
#define RBLE_GATT_PERM_HIDE         0x2000      Unexposed (hidden)
#define RBLE_GATT_PERM_ENC          0x4000      Encryption required
#define RBLE_GATT_PERM_NOTIFY_COMP_EN 0x8000    Enable notification completion
                                                event indication.
```

- Declaration of enumerated type for GATT event types

```
enum RBLE_GATT_EVENT_TYPE_enum {
    RBLE_GATT_EVENT_DISC_SVC_ALL_CMP = 1,     All 16bit UUID services discovery
                                              completion event
                                              (Parameters : disc_svc_all_cmp)
    RBLE_GATT_EVENT_DISC_SVC_ALL_128_CMP,     All 128bit UUID services discovery
                                              completion event
                                              (Parameters : disc_svc_all_128_cmp)
    RBLE_GATT_EVENT_DISC_SVC_BY_UUID_CMP,     Service discovery completion event
                                              by UUID
                                              (Parameter: disc_char_by_uuid_cmp)
    RBLE_GATT_EVENT_DISC_SVC_INCL_CMP,        Include service discovery completion
                                              event
                                              (Parameters : disc_svc_incl_cmp)
    RBLE_GATT_EVENT_DISC_CHAR_ALL_CMP,        All 16bit UUID characteristics discovery
                                              event
                                              (Parameters : disc_char_all_cmp)
    RBLE_GATT_EVENT_DISC_CHAR_ALL_128_CMP,    All 128bit UUID characteristics discovery
                                              event
                                              (Parameters : disc_char_all_128_cmp)
    RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_CMP,    16bit UUID characteristic discovery
                                              completion event
                                              (Parameters : disc_char_by_uuid_cmp)
    RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_128_CMP,
                                              128bit UUID characteristic discovery
                                              completion event
                                              (Parameters : disc_char_by_uuid_128_cmp)
    RBLE_GATT_EVENT_DISC_CHAR_DESC_CMP,       16bit characteristic descriptor
                                              discovery completion event
                                              (Parameters : disc_char_desc_cmp)
    RBLE_GATT_EVENT_DISC_CHAR_DESC_128_CMP,   128bit characteristic descriptor
                                              discovery completion event
                                              (Parameters : disc_char_desc_128_cmp)
    RBLE_GATT_EVENT_READ_CHAR_RESP,           Read characteristic and characteristic
                                              descriptor response event
                                              (Parameters : read_char_resp)
```

```
    RBLE_GATT_EVENT_READ_CHAR_LONG_RESP,      Read long characteristic response event
                                              (Parameters : read_char_long_resp)
    RBLE_GATT_EVENT_READ_CHAR_MULT_RESP,      Read multiple characteristics response event
                                              (Parameters : read_char_mult_resp)
    RBLE_GATT_EVENT_READ_CHAR_LONG_DESC_RESP, Read long characteristic descriptor response
                                              event
                                              (Parameters : read_char_long_desc_resp)
    RBLE_GATT_EVENT_WRITE_CHAR_RESP,          Write characteristic response event
                                              (Parameters : write_char_resp)
    RBLE_GATT_EVENT_WRITE_CHAR_RELIABLE_RESP, Write reliable characteristic response event
                                              (Parameters : write_reliable_resp)
    RBLE_GATT_EVENT_CANCEL_WRITE_CHAR_RESP,   Cancel write response event
                                              (Parameters : cancel_write_resp)
    RBLE_GATT_EVENT_HANDLE_VALUE_NOTIF,       Characteristic value notification event
                                              (Parameters : handle_value_notif)
    RBLE_GATT_EVENT_HANDLE_VALUE_IND,         Characteristic value indication event
                                              (Parameters : handle_value_ind)
    RBLE_GATT_EVENT_HANDLE_VALUE_CFM,         Characteristic value indication confirmation
                                              Event
                                              (Parameters : handle_value_cfm)
    RBLE_GATT_EVENT_DISCOVERY_CMP,            Discovery completion event
                                              (Parameters : discovery_cmp)
    RBLE_GATT_EVENT_COMPLETE,                 GATT processing completion event
                                              (Parameters : complete)
    RBLE_GATT_EVENT_WRITE_CMD_IND,            Write indication event
                                              (Parameters : write_cmd_ind)
    RBLE_GATT_EVENT_RESP_TIMEOUT,             GATT response timeout event
                                              (Parameters : none)
    RBLE_GATT_EVENT_SET_PERM_CMP,             Set permission completion event
                                              (Parameters : set_perm_cmp)
    RBLE_GATT_EVENT_SET_DATA_CMP,             Set data completion event
                                              (Parameters : set_data_cmp)
    RBLE_GATT_EVENT_NOTIFY_COMP,              Notification completion
                                              (Parameters : notify_cmp)
    RBLE_GATT_EVENT_COMMAND_DISALLOWED_IND,   GATT Command disallowed notification
                                              event
                                              (Parameter: cmd_disallowed_ind)
};
```

- Declaration of data type for GATT event types

```
typedef uint8_t  RBLE_GATT_EVENT_TYPE;
```

- Declaration of data type for GATT event callback function

```
typedef void ( *RBLE_GATT_EVENT_HANDLER )( RBLE_GATT_EVENT *event );
```

- Declaration of enumerated type for GATT request types

```
enum RBLE_GATT_REQ_TYPE_enum {
    RBLE_GATT_DISC_ALL_SVC= 0x00,                    Discover all services.
    RBLE_GATT_DISC_BY_UUID_SVC,                      Discover services based on
                                                     UUID.
    RBLE_GATT_DISC_INCLUDED_SVC,                     Discover Included services.
    RBLE_GATT_DISC_ALL_CHAR,                         Discover all characteristics.
    RBLE_GATT_DISC_BY_UUID_CHAR,                     Discover characteristics based
                                                     on UUID.
    RBLE_GATT_DISC_DESC_CHAR,                        Discover characteristic
                                                     descriptors.
    RBLE_GATT_READ_CHAR,                             Read a characteristic value.
    RBLE_GATT_READ_BY_UUID_CHAR,                     Read a characteristic value
                                                     based on UUID.
    RBLE_GATT_READ_LONG_CHAR,                        Read a long characteristic
                                                     value.
    RBLE_GATT_READ_MULT_LONG_CHAR,                   Read multiple long
                                                     characteristic values.
    RBLE_GATT_READ_DESC,                             Read a characteristic
                                                     descriptor.
    RBLE_GATT_READ_LONG_DESC,                        Read a long characteristic
                                                     descriptor.
    RBLE_GATT_WRITE_NO_RESPONSE,                     Write a characteristic value
                                                     with no response.
    RBLE_GATT_WRITE_SIGNED,                          Write a signed characteristic
                                                     value.
    RBLE_GATT_WRITE_CHAR,                            Write a characteristic value.
    RBLE_GATT_WRITE_LONG_CHAR,                       Write a long characteristic
                                                     value.
    RBLE_GATT_WRITE_RELIABLE_CHAR,                   Write a reliable characteristic
                                                     value.
    RBLE_GATT_WRITE_DESC,                            Write a characteristic
                                                     descriptor.
    RBLE_GATT_WRITE_LONG_DESC,                       Write a long characteristic
                                                     descriptor.
    RBLE_GATT_WRITE_CANCEL_CHAR                      Cancel writing a characteristic
                                                     value.
};
```

- Declaration of enumerated type for GATT characteristic property

```
enum RBLE_GATT_CHAR_PROP_enum {
    RBLE_GATT_CHAR_PROP_BCAST            = 0x01,        Broadcast by server
    RBLE_GATT_CHAR_PROP_RD               = 0x02,        Readable
    RBLE_GATT_CHAR_PROP_WR_NO_RESP       = 0x04,        Writable (without response)
    RBLE_GATT_CHAR_PROP_WR               = 0x08,        Writable
    RBLE_GATT_CHAR_PROP_NTF              = 0x10,        Notified by server
    RBLE_GATT_CHAR_PROP_IND              = 0x20,        Indicated by server
    RBLE_GATT_CHAR_PROP_AUTH             = 0x40,        Signed writable
    RBLE_GATT_CHAR_PROP_EXT_PROP         = 0x80         Extended property
```

```
};
```

- Declaration of structure type for discovery request

```
typedef struct RBLE_GATT_DESIRED_TYPE_t {
    uint16_t   value_size;                                  Request data size
    uint8_t    value[RBLE_GATT_128BIT_UUID_OCTET];          Request data
} RBLE_GATT_DESIRED_TYPE;
```

- Declaration of structure type for discovery request by UUID

```
typedef struct RBLE_GATT_UUID_TYPE_t {
    uint8_t    value_size;                                  UUID size
    uint8_t    expect_resp_size;                            Expected data size on read
                                                            multiple
    uint8_t    value[RBLE_GATT_128BIT_UUID_OCTET];          UUID
} RBLE_GATT_UUID_TYPE;
```

- Declaration of structure type for reliable write data request

```
typedef struct RBLE_GATT_RELIABLE_WRITE_t {
    uint16_t   elmt_hdl;                                    Characteristic handle
    uint16_t   size;                                        Data size
    uint8_t    value[RBLE_GATT_MAX_RELIABLE_WRITE_CONTENTS];
                                                            Write data
} RBLE_GATT_RELIABLE_WRITE;
```

- Declaration of structure type for service discovery request

```
typedef struct RBLE_GATT_DISC_SVC_REQ_t {
    uint8_t    req_type;                                    Request type
    uint8_t    reserved;                                    Reserved
    uint16_t   conhdl;                                      Connection handle
    uint16_t   start_hdl;                                   Discovery start handle
    uint16_t   end_hdl;                                     Discovery end handle
    RBLE_GATT_DESIRED_TYPE desired_svc;                     Discovery request type
} RBLE_GATT_DISC_SVC_REQ;
```

- Declaration of structure type for characteristic discovery request

```
typedef struct RBLE_GATT_DISC_CHAR_REQ_t {
    uint8_t    req_type;                                    Request type
    uint8_t    reserved;                                    Reserved
    uint16_t   conhdl;                                      Connection handle
    uint16_t   start_hdl;                                   Discovery start handle
    uint16_t   end_hdl;                                     Discovery end handle
    RBLE_GATT_DESIRED_TYPE desired_char;                    Discovery request type
} RBLE_GATT_DISC_CHAR_REQ;
```

- Declaration of structure type for descriptor discovery request

```
typedef struct RBLE_GATT_DISC_CHAR_DESC_REQ_t {
    uint16_t   conhdl;                                  Connection handle
    uint16_t   start_hdl;                               Discovery start handle
    uint16_t   end_hdl;                                 Discovery end handle
} RBLE_GATT_DISC_CHAR_DESC_REQ;
```

- Declaration of structure type for characteristic read request

```
typedef struct RBLE_GATT_READ_CHAR_REQ_t {
    uint8_t    req_type;                                Request type
    uint8_t    reserved;                                Reserved
    uint16_t   offset;                                  Read offset
    uint16_t   conhdl;                                  Connection handle
    uint16_t   start_hdl;                               Start handle
    uint16_t   end_hdl;                                 End handle
    uint16_t   nb_uuid;                                 number of UUIDs
    RBLE_GATT_UUID_TYPE   uuid[RBLE_GATT_MAX_NB_HDLS];  UUID data
} RBLE_GATT_READ_CHAR_REQ;
```

- Declaration of structure type for characteristic write request

```
typedef struct RBLE_GATT_WRITE_CHAR_REQ_t {
    uint16_t   conhdl;                                  Connection handle
    uint16_t   charhdl;                                 Characteristic handle
    uint16_t   wr_offset;                               Write offset
    uint16_t   val_len;                                 Write data size
    uint8_t    req_type;                                Request type
    uint8_t    auto_execute;                            Automatic execute write flag
                                        (for write long characteristic value)
    uint8_t    value[RBLE_GATT_MAX_LONG_VALUE];         Write data
} RBLE_GATT_WRITE_CHAR_REQ;
```

- Declaration of structure type for reliable write request

```
typedef struct RBLE_GATT_WRITE_RELIABLE_REQ_t {
    uint8_t    nb_writes;                               Number of write data
    uint8_t    auto_execute;                            Automatic execute write flag
    uint16_t   conhdl;                                  Connection handle
    RBLE_GATT_RELIABLE_WRITE     value[RBLE_GATT_MAX_RELIABLE_WRITE_NUM];
                                                        Write data
} RBLE_GATT_WRITE_RELIABLE_REQ;
```

- Declaration of structure type for execute write request

```
typedef struct RBLE_GATT_EXE_WR_CHAR_REQ_t {
    uint8_t    exe_wr_ena;                              Execute write / cancle flag
    uint8_t    reserved;                                Reserved
    uint16_t   conhdl;                                  Connection handle
} RBLE_GATT_EXE_WR_CHAR_REQ;
```

- Declaration of structure type for notification request

```
typedef struct RBLE_GATT_NOTIFY_REQ_t {
    uint16_t   conhdl;                                  Connection handle
    uint16_t   charhdl;                                 Characteristic handle
} RBLE_GATT_NOTIFY_REQ;
```

- Declaration of structure type for indication request

```
typedef struct RBLE_GATT_INDICATE_REQ_t {
    uint16_t   conhdl;                                  Connection handle
    uint16_t   charhdl;                                 Characteristic handle
} RBLE_GATT_INDICATE_REQ;
```

- Declaration of structure type for write response

```
typedef struct RBLE_GATT_WRITE_RESP_t {
    uint16_t   conhdl;                                  Connection handle
    uint16_t   att_hdl;                                 Attribute handle
    uint8_t    att_code;                                Response code
    uint8_t    reserved;                                Reserved
} RBLE_GATT_WRITE_RESP;
```

- Declaration of structure type for setting permission

```
typedef struct RBLE_GATT_SET_PERM_t {
    uint16_t   start_hdl;                               Start handle
    uint16_t   end_hdl;                                 End handle
    uint16_t   perm;                                    Permission
} RBLE_GATT_SET_PERM;
```

- Declaration of structure type for setting data

```
typedef struct RBLE_GATT_SET_DATA_t {
    uint16_t   val_hdl;                                 Attribute handle
    uint16_t   val_len;                                 Setting data size
    uint8_t    value[RBLE_GATT_MAX_LONG_VALUE];         Setting data
} RBLE_GATT_SET_DATA;
```

- Declaration of structure type for 16bit UUID service list

```
typedef struct RBLE_GATT_SVC_LIST_t {
    uint16_t   start_hdl;                               Start handle
    uint16_t   end_hdl;                                 End handle
    uint16_t   attr_hdl;                                Service UUID
} RBLE_GATT_SVC_LIST;
```

- Declaration of structure type for 128bit UUID service list

```
typedef struct RBLE_GATT_SVC_128_LIST_t {
    uint16_t   start_hdl;                                Start handle
    uint16_t   end_hdl;                                  End handle
    uint8_t    attr_hdl[RBLE_GATT_128BIT_UUID_OCTET];    Service UUID
} RBLE_GATT_SVC_128_LIST;
```

- Declaration of structure type for service range list

```
typedef struct RBLE_GATT_SVC_RANGE_LIST_t {
    uint16_t   start_hdl;                                Start handle
    uint16_t   end_hdl;                                  End handle
} RBLE_GATT_SVC_RANGE_LIST;
```

- Declaration of structure type for 16bit include service list

```
typedef struct RBLE_GATT_INCL_LIST_t {
    uint16_t   attr_hdl;                                 Attribute handle
    uint16_t   start_hdl;                                Start handle
    uint16_t   end_hdl;                                  End handle
    uint16_t   uuid;                                     Include service UUID
} RBLE_GATT_INCL_LIST;
```

- Declaration of structure type for 128bit include service list

```
typedef struct RBLE_GATT_INCL_128_LIST_t {
    uint16_t   attr_hdl;                                 Attribute handle
    uint16_t   start_hdl;                                Start handle
    uint16_t   end_hdl;                                  End handle
    uint16_t   uuid[RBLE_GATT_128BIT_UUID_OCTET];        Include service UUID
} RBLE_GATT_INCL_128_LIST;
```

- Declaration of structure type for 16bit characteristics list

```
typedef struct RBLE_GATT_CHAR_LIST_t {
    uint16_t   attr_hdl;                                 Characteristic handle
    uint8_t    prop;                                     Characteristic property
    uint8_t    reserved;                                 Reserved
    uint16_t   pointer_hdl;                              Characteristic value handle
    uint16_t   uuid;                                     Characteristic UUID
} RBLE_GATT_CHAR_LIST;
```

- Declaration of structure type for 128bit characteristics list

```
typedef struct RBLE_GATT_CHAR_128_LIST_t {
    uint16_t   attr_hdl;                                 Characteristic handle
    uint8_t    prop;                                     Characteristic property
    uint8_t    reserved;                                 Reserved
    uint16_t   pointer_hdl;                              Characteristic value handle
    uint8_t    uuid[RBLE_GATT_128BIT_UUID_OCTET];        Characteristic UUID
} RBLE_GATT_CHAR_128_LIST;
```

- Declaration of structure type for 16bit characteristic descriptor list

```
typedef struct RBLE_GATT_CHAR_DESC_LIST_t {
    uint16_t   attr_hdl;                                Characteristic handle
    uint16_t   desc_hdl;                                Descriptor UUID
} RBLE_GATT_CHAR_DESC_LIST;
```

- Declaration of structure type for 128bit characteristic descriptor list

```
typedef struct RBLE_GATT_CHAR_DESC_128_LIST_t {
    uint16_t   attr_hdl;                                Characteristic handle
    uint8_t    uuid[RBLE_GATT_128BIT_UUID_OCTET];       Descriptor UUID
} RBLE_GATT_CHAR_DESC_128_LIST;
```

- Declaration of structure type for read data

```
typedef struct RBLE_GATT_INFO_DATA_t {
    uint8_t    each_len;                                size of each handle and
                                                        value pair
    uint8_t    len;                                     Read data size
    uint8_t    data[RBLE_GATT_MAX_VALUE];               Read data
} RBLE_GATT_INFO_DATA;
```

- Declaration of structure type for multiple read data

```
typedef struct RBLE_GATT_QUERY_RESULT_t {
    uint8_t    len;                                     Read data size
    uint8_t    value[RBLE_GATT_MAX_VALUE];              Read data
} RBLE_GATT_QUERY_RESULT;
```

- GATT event parameter structure

```
typedef struct RBLE_GATT_EVENT_t {
    RBLE_GATT_EVENT_TYPE      type;                     GATT event type
    uint8_t                   reserved;                 Reserved
    union Event_Gatt_Parameter_u {

        All 16bit UUID services discovery completion event
        struct RBLE_GATT_Disc_Svc_All_Comp_t {
            uint16_t      conhdl;                       Connection handle
            uint8_t       att_code;                     Status
            uint8_t       nb_resp;                      Number of obtained lists
            RBLE_GATT_SVC_LIST   list[RBLE_GATT_MAX_HDL_LIST];
                                                        Obtained service list
        } disc_svc_all_cmp;
```

All 128bit UUID services discovery completion event

```
struct RBLE_GATT_Disc_Svc_All_128_Comp_t {
    uint16_t      conhdl;                       Connection handle
    uint8_t       att_code;                     Status
    uint8_t       nb_resp;                      Number of obtained lists
    RBLE_GATT_SVC_128_LIST list;                Obtained service list
} disc_svc_all_128_cmp;
```

Service discovery completion event by UUID

```
struct RBLE_GATT_Disc_Svc_By_Uuid_Comp_t {
    uint16_t      conhdl;                       Connection handle
    uint8_t       att_code;                     Status
    uint8_t       nb_resp;                      Number of obtained lists
    RBLE_GATT_SVC_RANGE_LIST list[RBLE_GATT_MAX_HDL_LIST];
                                                Obtained service list range
} disc_svc_by_uuid_cmp;
```

Include service discovery completion event

```
struct RBLE_GATT_Disc_Svc_Incl_Comp_t {
    uint16_t      conhdl;                       Connection handle
    uint8_t       nb_entry;                     Number of obtained services
    uint8_t       entry_len;                    Size of obtained service UUIDs
    union incl_list_u {
        RBLE_GATT_INCL_128_LIST incl;           128bit include services
        RBLE_GATT_INCL_LIST list[RBLE_GATT_MAX_HDL_LIST];
                                                16bit include services
    } incl_list;
} disc_svc_incl_cmp;
```

All 16bit UUID characteristics discovery completion event

```
struct RBLE_GATT_Disc_Char_All_Comp_t {
    uint16_t      conhdl;                       Connection handle
    uint8_t       att_code;                     Status
    uint8_t       nb_entry;                     Number of obtained lists
    RBLE_GATT_CHAR_LIST list[RBLE_GATT_MAX_HDL_LIST];
                                                Obtained characteristic lists
} disc_char_all_cmp;
```

All 128bit UUID characteristics discovery completion event

```
struct RBLE_GATT_Disc_Char_All_128_Comp_t {
    uint16_t      conhdl;                       Connection handle
    uint8_t       att_code;                     Status
    uint8_t       nb_entry;                     Number of obtained lists
    RBLE_GATT_CHAR_128_LIST list;               Obtained characteristic lists
} disc_char_all_128_cmp;
```

16bit UUID characteristic discovery completion event

```
struct RBLE_GATT_Disc_Char_By_Uuid_Comp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    uint8_t       nb_entry;                      Number of obtained lists
    RBLE_GATT_CHAR_LIST list[RBLE_GATT_MAX_HDL_LIST];
                                                 Obtained characteristic lists
} disc_char_by_uuid_cmp;
```

128bit UUID characteristic discovery completion event

```
struct RBLE_GATT_Disc_Char_By_Uuid_128_Comp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    uint8_t       nb_entry;                      Number of obtained lists
    RBLE_GATT_CHAR_128_LIST list;                Obtained characteristic lists
} disc_char_by_uuid_128_cmp;
```

16bit characteristic descriptor discovery completion event

```
struct RBLE_GATT_Disc_Char_Desc_Comp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       nb_entry;                      Number of obtained lists
    uint8_t       reserved;                      Reserved
    RBLE_GATT_CHAR_DESC_LIST list[RBLE_GATT_MAX_HDL_LIST];
                                                 Obtained characteristic
                                                 descriptor lists
} disc_char_desc_cmp;
```

128bit characteristic descriptor discovery completion event

```
struct RBLE_GATT_Disc_Char_Desc_128_Comp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       nb_entry;                      Number of obtained lists
    uint8_t       reserved;                      Reserved
    RBLE_GATT_CHAR_DESC_128_LIST list_128;       Obtained characteristic
                                                 descriptor lists
} disc_char_desc_128_cmp;
```

Read characteristic and characteristic descriptor response event

```
struct RBLE_GATT_Read_Char_Resp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    RBLE_GATT_INFO_DATA data;                    Read data
} read_char_resp;
```

Read long characteristic response event

```
struct RBLE_GATT_Read_Char_Long_Resp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    uint8_t       val_len;                       Read data size
    uint16_t      attr_hdl;                      Characteristic handle
    uint8_t       value[RBLE_GATT_MAX_VALUE];    Read data
} read_char_long_resp;
```

Read multiple characteristic response event

```
struct RBLE_GATT_Read_Char_Mult_Resp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    uint8_t       val_len;                       Read data size
    RBLE_GATT_QUERY_RESULT data[RBLE_GATT_MAX_NB_HDLS];
                                                 Read data
} read_char_mult_resp;
```

Read long characteristic descriptor response event

```
struct RBLE_GATT_Read_Char_Long_Desc_Resp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    uint8_t       val_len;                       Read data size
    uint8_t       value[RBLE_GATT_MAX_VALUE];    Read data
    uint16_t      attr_hdl;                      Chacateristic descriptor
                                                 handle
} read_char_long_desc_resp;
```

Write characteristic response event

```
struct RBLE_GATT_Write_Char_Resp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    uint8_t       reserved;                      Reserved
} write_char_resp;
```

Reliable write characteristic response event

```
struct RBLE_GATT_Write_Reliable_Resp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    uint8_t       reserved;                      Reserved
} write_reliable_resp;
```

Cancel write response event

```
struct RBLE_GATT_Cancel_Write_Char_Resp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    uint8_t       reserved;                      Reserved
} cancel_write_resp;
```

Characteristic value notification event

```
struct RBLE_GATT_Handle_Value_Notif_t {
    uint16_t      conhdl;                        Connection handle
    uint16_t      charhdl;                       Characteristic handle
    uint8_t       size;                          Notification data size
    uint8_t       value[RBLE_GATT_MAX_VALUE];    Notification data
    uint8_t       reserved;                      Reserved
} handle_value_notif;
```

Characteristic value indication event

```
struct RBLE_GATT_Handle_Value_Ind_t {
    uint16_t      conhdl;                        Connection handle
    uint16_t      charhdl;                       Characteristic handle
    uint8_t       size;                          Indication data size
    uint8_t       value[RBLE_GATT_MAX_VALUE];    Indication data size
    uint8_t       reserved;                      Reserved
} handle_value_ind;
```

Characteristic value indication confirmation event

```
struct RBLE_GATT_Handle_Value_Cfm_t {
    RBLE_STATUS   status;                        Characteristic value
                                                 indication result
} handle_value_cfm;
```

Discovery completion event

```
struct RBLE_GATT_Discovery_Comp_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    uint8_t       reserved;                      Reserved
} discovery_cmp;
```

GATT completion event

```
struct RBLE_GATT_Complete_t {
    uint16_t      conhdl;                        Connection handle
    uint8_t       att_code;                      Status
    uint8_t       reserved;                      Reserved
} complete;
```

Write indication event

```
struct RBLE_GATT_Write_Cmd_Ind_t {
    uint16_t      conhdl;                        Connection handle
    uint16_t      elmt;                          Characteristic handle
    uint16_t      size;                          Write data size
    uint8_t       offset;                        Write data position
    bool          resp;                          Confirmation request flag
    uint8_t       value[RBLE_GATT_MAX_VALUE];    Write data
} write_cmd_ind;
```

Set permission completion event

```
struct RBLE_GATT_Set_Perm_Complete_t {
    RBLE_STATUS   status;                        Permission setting result
} set_perm_cmp;
```

Set data completion event

```
struct RBLE_GATT_Set_Data_Complete_t {
    RBLE_STATUS   status;                        Data setting result
} set_data_cmp;
```

Notification completion event

```
struct RBLE_GATT_Notify_Comp_t {
    uint16_t      conhdl;                        Connection handle
    uint16_t      charhdl;                       Characteristic handle
    RBLE_STATUS   status;                        Notification result
    uint8_t       reserved;                      Reserved
} notify_cmp;
```

GATT command disallowed notification event

```
struct RBLE_EVT_GATT_Command_Disallowed_Ind_t{
    RBLE_STATUS   status;                        Status
    uint8_t       reserved;                      Reserved
    uint16_t      opcode;                        Opcode
}cmd_disallowed_ind;
    } param;
} RBLE_GATT_EVENT;
```

7.2 Functions

Table 7-1 shows the API functions defined for GATT of rBLE and the following sections describe the API functions in detail.

Table 7-1 API Functions Used by GATT

RBLE_GATT_Enable	Enables GATT.
RBLE_GATT_Discovery_Service_Request	Discovers service.
RBLE_GATT_Discovery_Char_Request	Discovers characteristic.
RBLE_GATT_Discovery_Char_Descriptor_Request	Discovers characteristic descriptor.
RBLE_GATT_Read_Char_Request	Reads characteristic value.
RBLE_GATT_Write_Char_Request	Writes characteristic value.
RBLE_GATT_Write_Reliable_Request	Reliable writes characteristic value.
RBLE_GATT_Execute_Write_Char_Request	Requests execution of write characteristic.
RBLE_GATT_Notify_Request	Requests notification of characteristic value.
RBLE_GATT_Indicate_Request	Requests inidication of characteristic value.
RBLE_GATT_Write_Response	Responds to a write characteristic value request.
RBLE_GATT_Set_Permission	Sets permission of the local database.
RBLE_GATT_Set_Data	Sets data of the local database.

7.2.1 RBLE_GATT_Enable

RBLE_STATUS RBLE_GATT_Enable(RBLE_GATT_EVENT_HANDLER callback)	
This function enables GATT function. This function should be called before using GATT rBLE APIs.	
Parameters:	
callback	Specify the callback function that reports the GATT event.
Return:	
RBLE_OK	Success
RBLE_PARAM_ERR	Invalid parameter
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.2.2 RBLE_GATT_Discovery_Service_Request

RBLE_STATUS RBLE_GATT_Discovery_Service_Request(RBLE_GATT_DISC_SVC_REQ *disc_svc)

This function performs the service discovery on the remote GATT server. Depending on the request type, one of either Discover All Primary Services, Discover Primary Service by Service UUID, or Find Include Services can be executed.

In case of Discover All Primary Services, when the service is discovered, either the all 16bit UUID service discovery completion event RBLE_GATT_EVENT_DISC_SVC_ALL_CMP or the all 128bit UUID service discovery completion event RBLE_GATT_EVENT_DISC_SVC_ALL_128_CMP is notified depending on the UUID of the discovered service. Completion of the service discovery will be notified by the discovery completion event RBLE_GATT_EVENT_DISCOVERY_CMP.

In case of Discover Primary Service by Service UUID, when the corresponding service is discovered, the service discovery by UUID completion event RBLE_GATT_EVENT_DISC_SVC_BY_UUID_CMP is notified. Completion of the service discovery will be notified by the GATT processing completion event RBLE_GATT_EVENT_COMPLETE.

In case of Find Include Services, when the corresponding service is discovered, the include service discovery completion event RBLE_GATT_EVENT_DISC_SVC_INCL_CMP is notified. Completion of the discovery will be notified by the discovery completion event RBLE_GATT_EVENT_DISCOVERY_CMP.

Parameters:

**disc_svc*	*req_type*	RBLE_GATT_DISC_ALL_SVC	Discover All Primary Services
		RBLE_GATT_DISC_BY_UUID_SVC	Discover Primary Service by Service UUID
		RBLE_GATT_DISC_INCLUDED_SVC	Find Include Services
	conhdl	Connection handle	
	start_hdl	Discovery start handle (valid if discovering include services)	
	end_hdl	Discovery end handle (valid if discovering include services)	
	desired_svc	To discover all primary services:	
		value_size	Specify RBLE_GATT_16BIT_UUID_OCTET
		value[RBLE_GATT_128BIT_UUID_OCTET]	16bit service UUID to suspend the discovery of services (the least significant byte first, left justified)
		To discover primary services by service UUID	
		value_size	Octet size of discover target service UUID
		value[RBLE_GATT_128BIT_UUID_OCTET]	Discover target service UUID (the least significant byte first, left justified)

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.2.3 RBLE_GATT_Discovery_Char_Request

RBLE_STATUS RBLE_GATT_Discovery_Char_Request(RBLE_GATT_DISC_CHAR_REQ *disc_char)
This function performs the characteristic discovery on the remote GATT server. Depending on the request type, either Discover All Characteristics of a Service or Discover Characteristics by UUID can be executed. In case of Discover All Characteristics of a Service, when the characteristics are discovered, either the all 16bit UUID characteristic discovery completion event RBLE_GATT_EVENT_DISC_CHAR_ALL_CMP or the all 128bit UUID characteristic discovery completion event RBLE_GATT_EVENT_DISC_CHAR_ALL_128_CMP is notified depending on the UUID of the discovered characteristic. In case of Discover Characteristics by UUID, when the corresponding characteristic is discovered, either the 16bit UUID characteristic discovery completion event RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_CMP or the 128bit UUID characteristic discovery completion event RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_128_CMP is notified. Completion of each discovery is notified by the GATT processing completion event RBLE_GATT_EVENT_COMPLETE.

Parameters:

*disc_char	*req_type*	RBLE_GATT_DISC_ALL_CHAR	Discover All Characteristics of a Service
		RBLE_GATT_DISC_BY_UUID_CHAR	Discover Characteristics by UUID
	conhdl	Connection handle	
	start_hdl	Discovery start handle	
	end_hdl	Discovery end handle	
	desired_char	*value_size*	Octet size of discover target characteristic UUID (valid if discovering characteristics by UUID)
		value[RBLE_GATT_128BIT_UUID_OCTET]	Discover target characteristic UUID (valid if discovering characteristics by UUID) (the least significant byte first, left justified)

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.2.4 RBLE_GATT_Discovery_Char_Descriptor_Request

```
RBLE_STATUS RBLE_GATT_Discovery_Char_Descriptor_Request(
    RBLE_GATT_DISC_CHAR_DESC_REQ *disc_char_desc )
```

This function performs the characteristic descriptor discovery on the remote GATT server.

When the characteristic descriptor within the specified range by handles is discovered, either the 16bit UUID characteristic descriptor discovery completion event RBLE_GATT_EVENT_DISC_CHAR_DESC_CMP or the 128bit UUID characteristic descriptor discovery completion event RBLE_GATT_EVENT_DISC_CHAR_DESC_128_CMP is notified depending on the UUID of the discovered characteristic descriptor.

Completion of each discovery is notified by the GATT processing completion event RBLE_GATT_EVENT_COMPLETE.

Parameters:

**disc_char_desc*	*conhdl*	Connection handle
	start_hdl	Discovery start handle
	end_hdl	Discovery start handle

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.2.5 RBLE_GATT_Read_Char_Request

RBLE_STATUS RBLE_GATT_Read_Char_Request(RBLE_GATT_READ_CHAR_REQ *rd_char)

This function is used to read either characteristic value or characteristic descriptor on the remote GATT server. Depending on the request type, one of the following read operations can be executed.

- Read characteristic value by specified handle
- Read characteristic value by specified UUID
- Read long characteristic value by specified handle
- Read multiple characteristic values
- Read characteristic descriptor by specified handle
- Read long characteristic descriptor by specified handle

In case of reading characteristic value by specified handle or reading characteristic descriptor by specified handle, when the read operation is completed, the read characteristic or characteristic descriptor response event RBLE_GATT_EVENT_READ_CHAR_RESP is notified.

In case of reading characteristic value by specified UUID, when the corresponding characteristic value is read, the read characteristic or characteristic descriptor response event RBLE_GATT_EVENT_READ_CHAR_RESP is notified.

In case of reading long characteristic value by specified handle, when the read operation is completed, the read long characteristic response event RBLE_GATT_EVENT_READ_CHAR_LONG_RESP is notified.

In case of reading multiple characteristic values, when the read operation is completed, the read multiple characteristics response event RBLE_GATT_EVENT_READ_CHAR_MULT_RESP is notified.

In case of reading long characteristic descriptor by specified handle, when the read operation is completed, the read long characteristic descriptor response event RBLE_GATT_EVENT_READ_CHAR_LONG_DESC_RESP is notified.

Parameters:

*rd_char	*req_type*	RBLE_GATT_READ_CHAR	Read characteristic value by specified handle
		RBLE_GATT_READ_BY_UUID_CHAR	Read characteristic value by specified UUID
		RBLE_GATT_READ_LONG_CHAR	Read long characteristic value by specified handle
		RBLE_GATT_READ_MULT_LONG_CHAR	Read multiple characteristic values
		RBLE_GATT_READ_DESC	Read characteristic descriptor by specified handle
		RBLE_GATT_READ_LONG_DESC	Read long characteristic descriptor by specified handle
	offset	Read offset (valid if reading long characteristic value by specified handle or reading a long characteristic descriptor by specified handle)	
	conhdl	Connection handle	
	start_hdl	Read start handle (valid if reading characteristic value by specified UUID. otherwise, specify 0.)	
	end_hdl	Read end handle (valid if reading characteristic value by specified UUID. otherwise, specify 0.)	
	nb_uuid	Number of handles to be read (the members *nb_uuid* is valid if reading multiple characteristic values)	

RBLE_STATUS RBLE_GATT_Read_Char_Request(RBLE_GATT_READ_CHAR_REQ *rd_char)				
		uuid[RBLE_GATT_MAX_NB_HDLS]	*value_size*	Size of UUID specified the members *uuid[0].value[ ]* - 16bit UUID RBLE_GATT_16BIT_UUID_OCTET - 128bit UUID RBLE_GATT_128BIT_UUID_OCTET (valid if reading characteristic value by specified UUID.)
			expect_resp_size	Expected read data size. (octets) If it contains a variable length data, RBLE_GATT_LEN_UNDEF can be specified only for the nb_uuid -th element. * Sum of each expected read data size should not exceed 22 octets (ATT_MTU - 1).
			value[RBLE_GATT_128BIT_UUID_OCTET]	- Read characteristic value by specified UUID Characteristic UUID of read target. - Other than above Known attribute handle of read target. (the least significant byte first, left justified)
Return:				
	RBLE_OK		Success	
	RBLE_STATUS_ERROR		Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

7.2.6 RBLE_GATT_Write_Char_Request

RBLE_STATUS RBLE_GATT_Write_Char_Request(RBLE_GATT_WRITE_CHAR_REQ *wr_char)

This function is used to write either characteristic value or characteristic descriptor on the remote GATT server. Depending on the request type, one of the following write operations can be executed.

- Write characteristic value or characteristic descriptor without response
- Signed write characteristic value
- Write characteristic value or characteristic descriptor with response
- Write long characteristic value or long characteristic descriptor

In case of writing characteristic value or characteristic descriptor without response, no completion event is notified.

In case of signed write characteristic value, it is necessary in advance to set the CSRK to the local device by using the RBLE_SM_Set_Key and deliver it to the remote GATT server device by performing the pairing. No completion event is notified.

In case of writing characteristic value or characteristic descriptor with response, when the write operation is completed, the write characteristic response event RBLE_GATT_EVENT_WRITE_CHAR_RESP is notified.

In case of writing long characteristic value or long characteristic descriptor with setting true to automatic execute write flags, when the write operation is completed, the GATT processing completion event RBLE_GATT_EVENT_COMPLETE is notified.

Parameters:

*wr_char	*conhdl*	Connection handle	
	charhdl	Characteristic value handle	
	wr_offset	Write offset (valid if writing long characteristic value or long characteristic descriptor)	
	val_len	Write data size	
	req_type	RBLE_GATT_WRITE_NO_RESPONSE	Write characteristic value or characteristic descriptor without response
		RBLE_GATT_WRITE_SIGNED	Signed write characteristic value
		RBLE_GATT_WRITE_CHAR	Write characteristic value with response
		RBLE_GATT_WRITE_LONG_CHAR	Write long characteristic value
		RBLE_GATT_WRITE_DESC	Write long characteristic descriptor with response
		RBLE_GATT_WRITE_LONG_DESC	Write long characteristic descriptor
	auto_execute	Automatic execute write flag (TRUE: execute write is performed automatically, FALSE: user performs the execute write) (valid if Write long characteristic value or long characteristic descriptor)	
	value[RBLE_GATT_MAX_LONG_VALUE]	Write data	

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.2.7 RBLE_GATT_Write_Reliable_Request

RBLE_STATUS RBLE_GATT_Write_Reliable_Request(RBLE_GATT_WRITE_RELIABLE_REQ *rel_write)				
This function is used to perform reliable write characteristic value on the remote GATT server. In case of reliable writing characteristic value with setting true to automatic execute write flags, when the all write operations are completed, the write reliable characteristic response event RBLE_GATT_EVENT_WRITE_CHAR_RELIABLE_RESP is notified.				
Parameters:				
	rel_write	*nb_writes*	Number of write data	
		auto_execute	Automatic execute write flag (TRUE: execute write is performed automatically, FALSE: user performs the execute write) (valid if Write long characteristic value or long characteristic descriptor)	
		conhdl	Connection handle	
		value[RBLE_GATT_MAX_RELIABLE_WRITE_NUM]	*elmt_hdl*	Characteristic value handle
			size	Write data size
			value[RBLE_GATT_MAX_RELIABLE_WRITE_CONTENTS]	Write data
Return:				
	RBLE_OK		Success	
	RBLE_STATUS_ERROR		Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

7.2.8 RBLE_GATT_Execute_Write_Char_Request

RBLE_STATUS RBLE_GATT_Execute_Write_Char_Request(RBLE_GATT_EXE_WR_CHAR_REQ *exe_wr_char)			
This function is used either to perform or to cancel all prepared writes of characteristic value on the remote GATT server. This function is usefull if the automatic execute write is not performed when the RBLE_GATT_Write_Char_Request (request type is either RBLE_GATT_WRITE_LONG_CHAR or RBLE_GATT_WRITE_LONG_DESC) or the RBLE_GATT_Execute_Write_Char_Request is called. In case of executing write long characteristic value or long characteristic descriptor, when all preprared writes are performed, the GATT processing completion event RBLE_GATT_EVENT_COMPLETE is notified. In case of executing reliable write characteristic value, when all preprared writes are performed, the write reliable characteristic response event RBLE_GATT_EVENT_WRITE_CHAR_RELIABLE_RESP is notified. In case of canceling write, when cancel operation is completed, the cancel write response event BLE_GATT_EVENT_CANCEL_WRITE_CHAR_RESP is notified.			
Parameters:			
	**exe_wr_char*	*exe_wr_ena*	Execute write flag (TRUE: perform all prepared writes, FALSE: cancel all prepared writes)
		conhdl	Connection handle
Return:			
	RBLE_OK		Success
	RBLE_STATUS_ERROR		Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.2.9 RBLE_GATT_Notify_Request

RBLE_STATUS RBLE_GATT_Notify_Request(RBLE_GATT_NOTIFY_REQ *notify)			
This function is used to notify characteristic value from the local GATT server to the remote GATT client. This function reads the characteristic value to be notified from the local GATT database. So, update the data in the local GATT database before calling this function.			
Parameters:			
	**notify*	*conhdl*	Connection handle
		charhdl	Characteristic value handle
Return:			
	RBLE_OK		Success
	RBLE_STATUS_ERROR		Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.2.10 RBLE_GATT_Indicate_Request

RBLE_STATUS RBLE_GATT_Indicate_Request(RBLE_GATT_INDICATE_REQ *indicate)			
This function is used to indicate characteristic value from the local GATT server to the remote GATT client. This function reads the characteristic value to be indicated from the local GATT database. So, update the data in the local GATT database before calling this function. The result is notified by the characteristic value indication confirmation event RBLE_GATT_EVENT_HANDLE_VALUE_CFM.			
Parameters:			
	**indicate*	*conhdl*	Connection handle
		charhdl	Characteristic value handle
Return:			
	RBLE_OK		Success
	RBLE_STATUS_ERROR		Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.2.11 RBLE_GATT_Write_Response

RBLE_STATUS RBLE_GATT_Write_Response(RBLE_GATT_WRITE_RESP *wr_resp)			
This function is used to respond to the write characteristic value request from the remote GATT client. * The BLE protocol stack doesn't update the local GATT database. So, update the data in the local GATT database before responding to the write request.			
Parameters:			
	**wr_resp*	*conhdl*	Connection handle
		att_hdl	Characteristic value handle
		att_code	Response to the write request (Refer to "Declaration of enumerated type for ATT error code" in 3.2)
Return:			
	RBLE_OK		Success
	RBLE_STATUS_ERROR		Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.2.12 RBLE_GATT_Set_Permission

RBLE_STATUS RBLE_GATT_Set_Permission(RBLE_GATT_SET_PERM *set_perm)			
This function is used to set the permission to the specified range of local GATT database by handles. The result is notified by the set permission completion event RBLE_GATT_EVENT_SET_PERM_CMP.			
Parameters:			
	**set_perm*	*start_hdl*	Start handle to set permission
		end_hdl	End handle to set permission
		perm	Permission (Refer to “Declaration of enumerated type for GATT attribute permission” in 7.1)
Return:			
	RBLE_OK		Success
	RBLE_STATUS_ERROR		Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.2.13 RBLE_GATT_Set_Data

RBLE_STATUS RBLE_GATT_Set_Data(RBLE_GATT_SET_DATA *set_data)			
This function is used to update data in the local GATT database by specifying handle. The result is notified by the set data completion event RBLE_GATT_EVENT_SET_DATA_CMP.			
Parameters:			
	**set_data*	*val_hdl*	Attribute handle
		val_len	Data size
		value[RBLE_GATT_MAX_LONG_VALUE]	Data
Return:			
	RBLE_OK		Success
	RBLE_STATUS_ERROR		Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

7.3 Events

Table 7-2 shows the events defined for GATT of rBLE and the following sections describe the events in detail.

Table 7-2 Events Defined for GATT

RBLE_GATT_EVENT_DISC_SVC_ALL_CMP	All 16bit UUID service discovery completion event
RBLE_GATT_EVENT_DISC_SVC_ALL_128_CMP	All 128bit UUID service discovery completion event
RBLE_GATT_EVENT_DISC_SVC_BY_UUID_CMP	Service discovery by UUID completion event
RBLE_GATT_EVENT_DISC_SVC_INCL_CMP	Include service discovery completion event
RBLE_GATT_EVENT_DISC_CHAR_ALL_CMP	All 16bit UUID characteristic discovery completion event
RBLE_GATT_EVENT_DISC_CHAR_ALL_128_CMP	All 128bit UUID characteristic discovery completion event
RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_CMP	16bit UUID characteristic discovery completion event
RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_128_CMP	128bit UUID characteristic discovery completion event
RBLE_GATT_EVENT_DISC_CHAR_DESC_CMP	16bit characteristic descriptor discovery completion event
RBLE_GATT_EVENT_DISC_CHAR_DESC_128_CMP	128bit characteristic descriptor discovery completion event
RBLE_GATT_EVENT_READ_CHAR_RESP	Read characteristic and characteristic descriptor response event
RBLE_GATT_EVENT_READ_CHAR_LONG_RESP	Read long characteristic response event
RBLE_GATT_EVENT_READ_CHAR_MULT_RESP	Read multiple characteristics response event
RBLE_GATT_EVENT_READ_CHAR_LONG_DESC_RESP	Read long characteristic descriptor response event
RBLE_GATT_EVENT_WRITE_CHAR_RESP	Write characteristic response event
RBLE_GATT_EVENT_WRITE_CHAR_RELIABLE_RESP	Write reliable characteristic response event
RBLE_GATT_EVENT_CANCEL_WRITE_CHAR_RESP	Cancel write response event
RBLE_GATT_EVENT_HANDLE_VALUE_NOTIF	Characteristic value notification event
RBLE_GATT_EVENT_HANDLE_VALUE_IND	Characteristic value indication event
RBLE_GATT_EVENT_HANDLE_VALUE_CFM	Characteristic value indication confirmation event
RBLE_GATT_EVENT_DISCOVERY_CMP	Discovery completion event
RBLE_GATT_EVENT_COMPLETE	GATT processing completion event
RBLE_GATT_EVENT_WRITE_CMD_IND	Write indication event
RBLE_GATT_EVENT_RESP_TIMEOUT	GATT response timeout event
RBLE_GATT_EVENT_SET_PERM_CMP	Set permission completion event
RBLE_GATT_EVENT_SET_DATA_CMP	Set data completion event
RBLE_GATT_EVENT_NOTIFY_COMP	Notification completion event
RBLE_GATT_EVENT_COMMAND_DISALLOWED_IND	GATT command disallowed indication event

7.3.1 RBLE_GATT_EVENT_DISC_SVC_ALL_CMP

RBLE_GATT_EVENT_DISC_SVC_ALL_CMP				
This event notifies the result of 16bit UUID primary service discovery on the remote GATT server. This event will be notified more than once if all of the data can not be notified at once due to limitations of the MTU.				
Parameters:				
	disc_svc_all_cmp	*conhdl*	Connection handle	
		att_code	Result of service discovery * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)	
		nb_resp	Number of discovery result The first nb_resp elements of the following parameters are valid.	
		list[RBLE_GATT_MAX_HDL_LIST]	*start_hdl*	Start of service handle
			end_hdl	End of service handle
			attr_hdl	16bit service UUID

7.3.2 RBLE_GATT_EVENT_DISC_SVC_ALL_128_CMP

RBLE_GATT_EVENT_DISC_SVC_ALL_128_CMP				
This event notifies the result of 128bit UUID primary service discovery on the remote GATT server. This event will be notified whenever 128bit UUID primary service is discovered.				
Parameters:				
	disc_svc_all_128_cmp	*conhdl*	Connection handle	
		att_code	Result of service discovery * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)	
		nb_resp	Number of discovery result. The first nb_resp elements of the following parameters are valid.	
		list	*start_hdl*	Start of service handle
			end_hdl	End of service handle
			attr_hdl[RBLE_GATT_128BIT_UUID_OCTET]	128bit service UUID

7.3.3 RBLE_GATT_EVENT_DISC_SVC_BY_UUID_CMP

RBLE_GATT_EVENT_DISC_SVC_BY_UUID_CMP				
This event notifies the result of primary service discovery by the specified UUID on the remote GATT server. This event will be notified more than once if all of the data can not be notified at once due to limitations of the MTU.				
Parameters:				
	disc_svc_by_uuid_cmp	*conhdl*	Connection handle	
		att_code	Result of service discovery * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)	
		nb_resp	Number of discovery result The first nb_resp elements of the following parameters are valid.	
		list[RBLE_GATT_MAX_HDL_LIST]	*start_hdl*	Start of service handle
			end_hdl	End of service handle

7.3.4 RBLE_GATT_EVENT_DISC_SVC_INCL_CMP

RBLE_GATT_EVENT_DISC_SVC_INCL_CMP					
This event notifies the result of include service discovery on the remote GATT server. This event will be notified more than once if all of the data can not be notified at once due to limitations of the MTU.					
Parameters:					
	disc_svc_incl_cmp	*conhdl*	Connection handle		
		nb_entry	Number of discovery result		
		entry_len	UUID size: If this parameter is equal to RBLE_GATT_128BIT_UUID_OCTET, the parameter incl_list is stored in the format incl. If this parameter is equal to RBLE_GATT_16BIT_UUID_OCTET, the parameter incl_list is stored in the format list[]. The first nb_entry elements of the following parameters are valid.		
		incl_list	*incl*	*attr_hdl*	Attribute handle
				start_hdl	Start of service handle
				end_hdl	End of service handle
				uuid[RBLE_GATT_128BIT_UUID_OCTET]	Service UUID
			list[RBLE_GATT_MAX_HDL_LIST]	*attr_hdl*	Attribute handle
				start_hdl	Start of service handle
				end_hdl	End of service handle
				uuid	Service UUID

7.3.5 RBLE_GATT_EVENT_DISC_CHAR_ALL_CMP

RBLE_GATT_EVENT_DISC_CHAR_ALL_CMP				
This event notifies the result of 16bit UUID characteristic discovery on the remote GATT server. This event will be notified more than once if all of the data can not be notified at once due to limitations of the MTU.				
Parameters:				
	disc_char_all_cmp	*conhdl*	Connection handle	
		att_code	Result of service discovery * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)	
		nb_entry	Number of discovery result * The first nb_entry elements of the following parameters are valid.	
		list[RBLE_GATT_MAX_HDL_LIST]	*attr_hdl*	Characteristic handle
			prop	Characteristic value property
			pointer_hdl	Characteristic value handle
			uuid	Characteristic value UUID

7.3.6 RBLE_GATT_EVENT_DISC_CHAR_ALL_128_CMP

RBLE_GATT_EVENT_DISC_CHAR_ALL_128_CMP				
This event notifies the result of 128bit UUID characteristic discovery on the remote GATT server. This event will be notified whenever 128bit UUID characteristic is discovered.				
Parameters:				
	disc_char_all_128_cmp	*conhdl*	Connection handle	
		att_code	Result of characteristic discovery * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)	
		nb_entry	Number of discovery result	
		list	*attr_hdl*	Characteristic handle
			prop	Characteristic value property
			pointer_hdl	Characteristic value handle
			uuid[RBLE_GATT_128BIT_UUID_OCTET]	Characteristic value UUID

7.3.7 RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_CMP

RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_CMP			
This event notifies the result of 16bit UUID characteristic descriptor discovery by the specified 16bit UUID on the remote GATT server. This event will be notified more than once if all of the data can not be notified at once due to limitations of the MTU.			
Parameters:			
disc_char_by_uuid_cmp	*conhdl*	Connection handle	
	att_code	Result of characteristic discovery * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)	
	nb_entry	Number of discovery result * The first nb_entry elements of the following parameters are valid.	
	list[RBLE_GATT_MAX_HDL_LIST]	*attr_hdl*	Characteristic handle of the specified UUID
		prop	Characteristic value property
		pointer_hdl	Characteristic value handle
		uuid	Characteristic value UUID

7.3.8 RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_128_CMP

RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_128_CMP			
This event notifies the result of 128bit UUID characteristic descriptor discovery by the specified 128bit UUID on the remote GATT server. This event will be notified whenever the specified 128bit UUID characteristic descriptor is discovered.			
Parameters:			
disc_char_by_uuid_128_cmp	*conhdl*	Connection handle	
	att_code	Result of characteristic discovery * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)	
	nb_entry	Number of discovery result.	
	list	*attr_hdl*	Characteristic handle
		prop	Characteristic value property
		pointer_hdl	Characteristic value handle
		uuid[RBLE_GATT_128BIT_UUID_OCTET]	Characteristic value UUID

7.3.9 RBLE_GATT_EVENT_DISC_CHAR_DESC_CMP

RBLE_GATT_EVENT_DISC_CHAR_DESC_CMP				
This event notifies the result of 16bit UUID characteristic descriptor discovery on the remote GATT server. This event will be notified more than once if all of the data can not be notified at once due to limitations of the MTU.				
Parameters:				
	disc_char_desc_cmp	*conhdl*	Connection handle	
		nb_entry	Number of discovery result. * The first nb_entry elements of the following parameters are valid.	
		list[RBLE_GATT_MAX_HDL_LIST]	*attr_hdl*	Characteristic descriptor handle
			desc_hdl	Characteristic descriptor UUID

7.3.10 RBLE_GATT_EVENT_DISC_CHAR_DESC_128_CMP

RBLE_GATT_EVENT_DISC_CHAR_DESC_128_CMP				
This event notifies the result of 128bit UUID characteristic descriptor discovery on the remote GATT server. This event will be notified whenever 128bit UUID characteristic descriptor is discovered.				
Parameters:				
	disc_char_desc_128_cmp	*conhdl*	Connection handle	
		nb_entry	Number of discovery result	
		list_128	*attr_hdl*	Characteristic handle
			uuid[RBLE_GATT_128BIT_UUID_OCTET]	Characteristic value UUID

7.3.11 RBLE_GATT_EVENT_READ_CHAR_RESP

RBLE_GATT_EVENT_READ_CHAR_RESP				
This event notifies the reception of the response (to the read characteristic value or characteristic descriptor request) from the remote GATT server.				
Parameters:				
	read_char_resp	*conhdl*	Connection handle	
		att_code	Result of read characteristic value or characteristic descriptor * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)	
		data	*each_len*	Size of each handle and value pair. * This parameter is valid if reading characteristic value or characteristic descriptor by UUID.
			len	Read data size
			data[RBLE_GATT_MAX_VALUE]	Read data

7.3.12 RBLE_GATT_EVENT_READ_CHAR_LONG_RESP

RBLE_GATT_EVENT_READ_CHAR_LONG_RESP			
This event notifies the reception of the response (to the read long characteristic request) from the remote GATT server. This event will be notified more than once, until all of the characteristic are read.			
Parameters:			
	read_char_long_resp	*conhdl*	Connection handle
		att_code	Result of read long characteristic * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)
		val_len	Read data size in octets. If this size is less than 22 octets (ATT_MTU – 1 octets), all of the characteristic value are read completely.
		attr_hdl	Characteristic handle
		value[RBLE_GATT_MAX_VALUE]	Read data

7.3.13 RBLE_GATT_EVENT_READ_CHAR_MULT_RESP

RBLE_GATT_EVENT_READ_CHAR_MULT_RESP				
This event notifies the reception of response (to read multiple characteristics request) from the remote GATT server.				
Parameters:				
	read_char_mult_resp	*conhdl*	Connection handle	
		att_code	Result of read multiple characteristics * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)	
		val_len	Total read data size	
		data[RBLE_GATT_MAX_NB_HDLS]	*len*	Read data size
			value[RBLE_GATT_MAX_VALUE]	Reda data

7.3.14 RBLE_GATT_EVENT_READ_CHAR_LONG_DESC_RESP

RBLE_GATT_EVENT_READ_CHAR_LONG_DESC_RESP			
This event notifies the reception of the response (to the read long characteristic descriptor request) from the remote GATT server. This event will be notified more than once, until all of the characteristic descriptors are read.			
Parameters:			
	read_char_long_desc_resp	*conhdl*	Connection handle
		att_code	Result of read long characteristic descriptor * The rest of parameters are valid if this value is 0x00. (Refer to “Declaration of enumerated type for ATT error code” in 3.2)
		val_len	Read data size in octets. If this size is less than 22 octets (ATT_MTU – 1 octets), all of the characteristic descriptors are read completely.
		value[RBLE_GATT_MAX_VALUE]	Read data
		attr_hdl	Characteristic descriptor handle

7.3.15 RBLE_GATT_EVENT_WRITE_CHAR_RESP

RBLE_GATT_EVENT_WRITE_CHAR_RESP			
This event notifies the result of command execution (write characteristic) from the remote GATT server.			
Parameters:			
	write_char_resp	*conhdl*	Connection handle
		att_code	Result of write characteristic (Refer to “Declaration of enumerated type for ATT error code” in 3.2)

7.3.16 RBLE_GATT_EVENT_WRITE_CHAR_RELIABLE_RESP

RBLE_GATT_EVENT_WRITE_CHAR_RELIABLE_RESP			
This event notifies the result of command execution (reliable write characteristic) from the remote GATT server.			
Parameters:			
	write_reliable_resp	*conhdl*	Connection handle
		att_code	Result of reliable write characteristic (Refer to “Declaration of enumerated type for ATT error code” in 3.2)

7.3.17 RBLE_GATT_EVENT_CANCEL_WRITE_CHAR_RESP

RBLE_GATT_EVENT_CANCEL_WRITE_CHAR_RESP			
This event notifies the result of command execution (cancel prepared writes) from the remote GATT server.			
Parameters:			
	cancel_write_resp	*conhdl*	Connection handle
		att_code	Result of cancel prepared writes (Refer to “Declaration of enumerated type for ATT error code” in 3.2)

7.3.18 RBLE_GATT_EVENT_HANDLE_VALUE_NOTIF

RBLE_GATT_EVENT_HANDLE_VALUE_NOTIF			
This event notifies the reception of the characteristic value notification from the remote GAT server.			
Parameters:			
	handle_value_notif	*conhdl*	Connection handle
		charhdl	Characteristic value handle
		size	Notification data size
		value[RBLE_GATT_MAX_VALUE]	Notification data

7.3.19 RBLE_GATT_EVENT_HANDLE_VALUE_IND

RBLE_GATT_EVENT_HANDLE_VALUE_IND			
This event notifies the reception of the characteristic value indication from the remote GAT server. * The BLE protocol stack performs automatically the characteristic value indication confirmation to a GATT client.			
Parameters:			
	handle_value_ind	*conhdl*	Connection handle
		charhdl	Characteristic value handle
		size	Indication data size
		value[RBLE_GATT_MAX_VALUE]	Indication data

7.3.20 RBLE_GATT_EVENT_HANDLE_VALUE_CFM

RBLE_GATT_EVENT_HANDLE_VALUE_CFM			
This event notifies the reception of the characteristic value indication confirmation from a remote GAT client.			
Parameters:			
	handle_value_cfm	*status*	Result of command execution (Refer to “Declaration of enumerated type for rBLE status” in 3.2)

7.3.21 RBLE_GATT_EVENT_DISCOVERY_CMP

RBLE_GATT_EVENT_DISCOVERY_CMP			
This event notifies the results of service discovery (discover all services and find include service).			
Parameters:			
	discovery_cmp	*conhdl*	Connection handle
		att_code	Result of service discovery (Refer to “Declaration of enumerated type for rBLE status” in 3.2)

7.3.22 RBLE_GATT_EVENT_COMPLETE

RBLE_GATT_EVENT_COMPLETE			
This event notifies the completion of GATT processing (GATT command execution).			
Parameters:			
	complete	*conhdl*	Connection handle
		att_code	Result of command execution (Refer to “Declaration of enumerated type for rBLE status” in 3.2)

7.3.23 RBLE_GATT_EVENT_WRITE_CMD_IND

RBLE_GATT_EVENT_WRITE_CMD_IND			
This event notifies the reception of the write characteristic value request from a remote GATT client. To respond to the write request, use RBLE_GATT_Write_Response. The necessity of response can be determined by parameter resp. Check the validity of data and if correct, update the corresponding data of the local GATT database by using RBLE_GATT_Set_Data.			
Parameters:			
	write_cmd_ind	*conhdl*	Connection handle
		elmt	Characteristic value handle
		size	Write data size
		offset	Write data offset
		resp	Necessity of response to write request (TRUE: necessary, FALSE: unnecessary)
		value[RBLE_GATT_MAX_VALUE]	Write data

7.3.24 RBLE_GATT_EVENT_RESP_TIMEOUT

RBLE_GATT_EVENT_RESP_TIMEOUT	
This event notifies that the response timeout occurs during the GATT processing on the remote device. * Timeout period is 30 seconds.	
Parameters:	
	none

7.3.25 RBLE_GATT_EVENT_SET_PERM_CMP

RBLE_GATT_EVENT_SET_PERM_CMP			
This event notifies the results of command execution (set permission of local GATT database).			
Parameters:			
	set_perm_cmp	*status*	Result of command execution (Refer to “Declaration of enumerated type for rBLE status” in 3.2)

7.3.26 RBLE_GATT_EVENT_SET_DATA_CMP

RBLE_GATT_EVENT_SET_DATA_CMP		
This event notifies the results of command execution (set data of local GATT database).		
Parameters:		
set_data_cmp	*status*	Result of command execution (Refer to “Declaration of enumerated type for rBLE status” in 3.2)

7.3.27 RBLE_GATT_EVENT_NOTIFY_COMP

RBLE_GATT_EVENT_NOTIFY_COMP		
This event notifies results of the characteristic value notification to a remote GATT client. To enable this event, set the RBLE_GATT_PERM_NOTIFY_COMP_EN permission of the relevant characteristic in GATT database. * This event does not guarantee the sending.		
Parameters:		
notify_cmp	*conhdl*	Connection handle
	charhdl	Characteristic value handle
	status	Result of sending notification (Refer to “Declaration of enumerated type for rBLE status” in 3.2)

7.3.28 RBLE_GATT_EVENT_COMMAND_DISALLOWED_IND

RBLE_GATT_EVENT_COMMAND_DISALLOWED_IND		
This event notifies that a GATT command was disallowed.		
Parameters:		
cmd_disallowed_ind	*status*	Result of command execution (Refer to “Declaration of enumerated type for rBLE status” in 3.2)
	opcode	Opcode of the disallowed command

8. Vendor Specific

This section describes the APIs of the Vendor Specific (VS) profile. By using VS, features such as Direct Test Mode and Renesas-unique extended Direct Test Mode can be used.

8.1 Definitions

This section describes the definitions used by the API of VS.

- GPIO bits definitions

```
#define RBLE_VS_GPIO_BIT_0                0x01            Bit0
#define RBLE_VS_GPIO_BIT_1                0x02            Bit1
#define RBLE_VS_GPIO_BIT_2                0x04            Bit2
#define RBLE_VS_GPIO_BIT_3                0x08            Bit3
```

- GPIO input/output direction definitions

```
#define RBLE_VS_GPIO_INPUT                0               Input
#define RBLE_VS_GPIO_OUTPUT               1               Output
```

- GPIO input/output vlue definitions

```
#define RBLE_VS_GPIO_LOW                  0               Low
#define RBLE_VS_GPIO_HIGH                 1               High
```

- GPIO input/output direction setting macro definitions

```
#define RBLE_VS_GPIO_DIR_SETTING(val, bit, dir)           \
                          val = (uint8_t)(((dir)==RBLE_VS_GPIO_INPUT)\
                          ?((uint8_t)(val)&~(bit))\
                          :((uint8_t)(val)|(bit)))
```

- GPIO output setting macro definitions

```
#define RBLE_VS_GPIO_OUTPUT_SETTING(val, bit, set)        \
                          val = (uint8_t)(((set)==RBLE_VS_GPIO_LOW)\
                          ?((uint8_t)(val)&~(bit))\
                          :((uint8_t)(val)|(bit)))
```

- Declaration of enumerated type for VS event types

```
enum RBLE_VS_EVENT_TYPE_enum {
    RBLE_VS_EVENT_TEST_RX_START_COMP = 0x01,  Reception test start completion event
                                              (Parameter: status)
    RBLE_VS_EVENT_TEST_TX_START_COMP,         Transmission test start completion
                                              event
                                              (Parameter: status)
    RBLE_VS_EVENT_TEST_END_COMP,              Test end event
                                              (Parameter: test_end_cmp)
    RBLE_VS_EVENT_WR_BD_ADDR_COMP,            BD address write completion event
                                              (Parameter: status)
```

```
    RBLE_VS_EVENT_SET_TEST_PARAM_COMP,        Extended parameter setup completion
                                              event in Direct Test mode
                                              (Parameter: status)
    RBLE_VS_EVENT_READ_TEST_RSSI_COMP,        RSSI acquisition completion event in
                                              Direct Test Mode
                                              (Parameter: test_rssi_cmp)
    RBLE_VS_EVENT_GPIO_DIR_COMP,              GPIO input/output direction setting
                                              completion event
                                              (Parameters: gpio_dir_cmp)
    RBLE_VS_EVENT_GPIO_ACCESS_COMP,           GPIO access completion event
                                              (Parameters: gpio_access_cmp)
    RBLE_VS_EVENT_FLASH_MANAGEMENT_COMP,      Data Flash access management command
                                              completion event
                                              (Parameters: management_comp)
    RBLE_VS_EVENT_FLASH_ACCESS_COMP,          Data Flash data access command
                                              completion event
                                              (Parameters: access_comp)
    RBLE_VS_EVENT_FLASH_OPERATION_COMP,       Data Flash block operation completion
                                              event
                                              (Parameters: operation_comp)
    RBLE_VS_EVENT_FLASH_GET_SPACE_COMP,       Data Flash free space acquisition
                                              completion event
                                              (Parameters: get_space)
    RBLE_VS_EVENT_FLASH_GET_EEL_VER_COMP,     Data Flash EEL version acquisition
                                              completion event
                                              (Parameters: get_eel_ver)
    RBLE_VS_EVENT_ADAPT_ENABLE_COMP,          Adaptable function enable completion
                                              event
                                              (Parameters: adapt_enable_cmp)
    RBLE_VS_EVENT_ADAPT_STATE_IND,            Adaptable function state change
                                              notification event
                                              (Parameters: adapt_state_ind)
    RBLE_VS_EVENT_COMMAND_DISALLOWED_IND,     VS command disallowed notification event
                                              (Parameter: cmd_disallowed_ind)
    RBLE_VS_EVENT_SET_TX_POWER_COMP,          Transmit power setup completion event
                                              (Parameter: status)
    RBLE_VS_EVENT_SET_PARAMS_COMP,            Parameter setting completion event
                                              (Parameter: status)
    RBLE_VS_EVENT_RF_CONTROL_COMP             RF power supply control completion event
                                              (Parameters: rf_control_cmp)

};
```

- Declaration of data type for VS event types

```
typedef uint8_t  RBLE_VS_EVENT_TYPE;
```

- Declaration of data type for VS event callback function

```
typedef void ( *RBLE_VS_EVENT_HANDLER )( RBLE_VS_EVENT *event );
```

- Declaration of enumerated type for transmission data pattern

```
enum RBLE_TEST_DATA_PATTERN_enum {
    RBLE_TEST_DATA_PATTERN_PN9            = 0x00,        Pseudo random bit sequence 9
    RBLE_TEST_DATA_PATTERN_11110000       = 0x01,        Bit pattern '11110000'
    RBLE_TEST_DATA_PATTERN_10101010       = 0x02,        Bit pattern '10101010'
    RBLE_TEST_DATA_PATTERN_PN15           = 0x03,        Pseudo random bit sequence 15
    RBLE_TEST_DATA_PATTERN_ALL1           = 0x04,        All bits are 1
    RBLE_TEST_DATA_PATTERN_ALL0           = 0x05,        All bits are 0
    RBLE_TEST_DATA_PATTERN_00001111       = 0x06,        Bit pattern '00001111'
    RBLE_TEST_DATA_PATTERN_01010101       = 0x07         Bit pattern '01010101'
};
```

- Declaration of enumerated type for transmit power level

```
enum RBLE_VS_TXPW_SET_LEVEL_enum {
    RBLE_VS_TXPW_LV1       = 0x01,                       -15dbm
    RBLE_VS_TXPW_LV2       = 0x02,                       -10dbm
    RBLE_VS_TXPW_LV3       = 0x03,                       -7dbm
    RBLE_VS_TXPW_LV4       = 0x04,                       -2dbm
    RBLE_VS_TXPW_LV5       = 0x05,                       Reserved
    RBLE_VS_TXPW_LV6       = 0x06,                       Reserved
    RBLE_VS_TXPW_LV7       = 0x07,                       -1dbm
    RBLE_VS_TXPW_LV8       = 0x08,                       Reserved
    RBLE_VS_TXPW_LV9       = 0x09                        -0dbm
};
```

- Declaration of enumerated type for transmit power setting mode

```
enum RBLE_VS_TXPW_MODE_enum {
    RBLE_VS_TXPW_MODE_NORMAL,                            Adaptable function disabled
    RBLE_VS_TXPW_MODE_ADAPT_NEAR,                        RF low-power mode
    RBLE_VS_TXPW_MODE_ADAPT_MIDDLE,                      RF normal mode
    RBLE_VS_TXPW_MODE_ADAPT_FAR                          RF high-performance mode
};
```

- Declaration of enumerated type for GPIO access mode

```
enum RBLE_VS_GPIO_MD_enum {
     RBLE_VS_GPIO_INPUT_MD,                              Input mode
     RBLE_VS_GPIO_OUTPUT_MD,                             Output mode
};
```

- Declaration of enumerated type for adaptable state

```
enum RBLE_VS_ADAPT_STATE_enum {
     RBLE_VS_ADAPT_MODE_NEAR,                          State in RF low-power mode
     RBLE_VS_ADAPT_MODE_MIDDLE,                        State in RF normal mode
     RBLE_VS_ADAPT_MODE_FAR                            State in RF high-performance mode
};
```

- Declaration of enumerated type for adaptable function commands

```
enum RBLE_VS_ADAPT_CMD_enum {
     RBLE_VS_ADAPT_CMD_DISABLE       = 0x00,         Adaptable function disable
     RBLE_VS_ADAPT_CMD_ENABLE        = 0x01,         Adaptable function enable,
                                                     State indication enable
     RBLE_VS_ADAPT_CMD_ENABLE_WO_IND = 0x81          Adaptable function enable,
                                                     State indication disable
};
```

- Declaration of enumerated type for Data Flash control commands

```
enum RBLE_VS_FLASH_CMD_enum {
    RBLE_VS_FLASH_CMD_START,                         Access start
    RBLE_VS_FLASH_CMD_STOP,                          Access stop
    RBLE_VS_FLASH_CMD_WRITE,                         Write data
    RBLE_VS_FLASH_CMD_READ,                          Read data
    RBLE_VS_FLASH_CMD_CLEANUP,                       Cleanup
    RBLE_VS_FLASH_CMD_FORMAT                         Format
};
```

- Declaration of enumerated type for RF chip power supply control commands

```
enum RBLE_VS_RFCNTL_CMD_enum {
    RBLE_VS_RFCNTL_CMD_POWDOWN,                      RF power supply OFF
    RBLE_VS_RFCNTL_CMD_POWUP_DDCON,                  RF power supply ON (DC-DC enable)
    RBLE_VS_RFCNTL_CMD_POWUP_DDCOFF                  RF power supply ON (DC-DC disable)
};
```

- Declaration of enumerated type for setting parameters

```
enum RBLE_VS_SET_PARAM_enum {
    RBLE_VS_PARAM_DISC_SCAN_TIME = 0x00,             gap_discovery_scan_time
    RBLE_VS_PARAM_DISC_SCAN_INTV,                    gap_dev_search_scan_intv
    RBLE_VS_PARAM_DISC_SCAN_WIND,                    gap_dev_search_scan_window
    RBLE_VS_PARAM_LIM_ADV_TO,                        gap_lim_adv_timeout
    RBLE_VS_PARAM_SCAN_FAST_INTV,                    gap_scan_fast_intv
    RBLE_VS_PARAM_SCAN_FAST_WIND,                    gap_scan_fast_window
    RBLE_VS_PARAM_CONN_INTV_MIN,                     gap_init_conn_min_intv
    RBLE_VS_PARAM_CONN_INTV_MAX,                     gap_init_conn_max_intv
    RBLE_VS_PARAM_CONN_CE_MIN,                       gap_conn_min_ce_length
    RBLE_VS_PARAM_CONN_CE_MAX,                       gap_conn_max_ce_length
    RBLE_VS_PARAM_CONN_SLAVE_LATENCY,                gap_conn_slave_latency
    RBLE_VS_PARAM_CONN_SVTO,                         gap_dev_supervision_timeout
    RBLE_VS_PARAM_RPA_INTV,                          gap_resolvable_private_addr_intv
    RBLE_VS_PARAM_USER_DEFINED_TOP  = 0x80           First of the user-defined parameters
};
```

- Data Flash access parameters structure

```
typedef struct RBLE_VS_FLASH_ACCESS_PARAM_t {
    uint8_t cmd;                                    Execution command
    uint8_t id;                                     Data ID
    uint8_t size;                                   Data size
    uint8_t reserved;                               Reserved
    uint8_t *addr;                                  Pointer to the data buffer
} RBLE_VS_FLASH_ACCESS_PARAM;
```

- VS event parameter structure

```
typedef struct RBLE_VS_EVENT_t {
    RBLE_VS_EVENT_TYPE        type;                     VS event type
    uint8_t                   reserved;                 Reserved
    union Event_Parameter_u {
        Generic event
        RBLE_STATUS           status;                   Status

        Test end event
        struct RBLE_VS_Test_End_Comp_t{
            RBLE_STATUS       status;                   Status
            uint8_t           reserved;                 Reserved
            uint16_t          nb_packet_received;       Number of packets received
        }test_end_cmp;

        RSSI acquisition completion event in Direct Test Mode
        struct RBLE_VS_Read_Test_RSSI_Comp_t {
           RBLE_STATUS        status;                   Status
           uint8_t            rssi;                     RSSI value
        } test_rssi_cmp;

        GPIO input/output direction setting completion event
        struct RBLE_VS_GPIO_Dir_Comp_t {
           RBLE_STATUS        status;                   Status
           uint8_t            mask;                     GPIO mask
        } gpio_dir_cmp;

        GPIO accsess completeion event
        struct RBLE_VS_GPIO_Access_Comp_t {
           RBLE_STATUS        status;                   Satus
           uint8_t            value;                    GPIO input value
        } gpio_access_cmp;
```

Data Flash access management command completion event

```
struct RBLE_VS_Flash_Management_Comp_t {
   RBLE_STATUS         status;                     Status
   uint8_t             cmd;                        Execution command
} management_comp;
```

Data Flash data access completionn event

```
struct RBLE_VS_Flash_Access_Comp_t {
   RBLE_STATUS         status;                     Status
   uint8_t             cmd;                        Execution command
   uint8_t             id;                         Data ID
   uint8_t             size;                       Data size
   uint8_t             *addr;                      Poiter to data buffer
}access_comp;
```

Data Flash block operation completionn event

```
struct RBLE_VS_Flash_Operation_Comp_t {
   RBLE_STATUS         status;                     Status
   uint8_t             cmd;                        Execution command
}operation_comp;
```

Data Flash free space acquisition completionn event

```
struct RBLE_VS_Flash_Get_Space_Comp_t {
   RBLE_STATUS         status;                     Status
   uint8_t             reserved;                   Reserved
   uint16_t            size;                       Free space
}get_space;
```

Data Flash EEL version information acquisition completionn event

```
struct RBLE_VS_Flash_Get_EEL_Ver_Comp_t {
   RBLE_STATUS         status;                     Status
   uint8_t             version[24];                Version information
}get_eel_ver;
```

Adaptable function enable completion event

```
struct RBLE_VS_Adapt_Enable_Comp_t {
   RBLE_STATUS         status;                     Status
   uint8_t             cmd;                        Adaptable function
                                                   enable/disable command
}adapt_enable_cmp;
```

Adaptable mode state change notification event

```
struct RBLE_VS_Adapt_State_Ind_t {
   uint8_t             state;                      Adaptable state
}adapt_state_ind;
```

```
        RF power supply control completion event
        struct RBLE_VS_RF_Control_Comp_t {
           RBLE_STATUS         status;                     Status
        }rf_control_cmp;

        VS command disallowed notification event
        struct RBLE_VS_Command_Disallowed_Ind_t{
            RBLE_STATUS        status;                     Status
            uint8_t            reserved;                   Reserved
            uint16_t           opcode;                     Opcode
        }cmd_disallowed_ind;
    } param;
} RBLE_VS_EVENT;
```

8.2 Functions

The following table shows the API functions defined for VS of rBLE and the following sections describe the API functions in detail.

Table 8-1 API Functions Used by VS

RBLE_VS_Enable	Enables VS.
RBLE_VS_Test_Rx_Start	Starts a reception test.
RBLE_VS_Test_Tx_Start	Starts a transmission test.
RBLE_VS_Test_End	Ends a transmission/reception test.
RBLE_VS_Set_Test_Parameter	Sets the extended parameter in Direct Test Mode.
RBLE_VS_Read_Test_RSSI	Reads RSSI in Direct Test Mode.
RBLE_VS_Write_Bd_Address	Writes a DB address.
RBLE_VS_Set_Tx_Power	Sets a transmission power.
RBLE_VS_GPIO_Dir	Sets the input/output direction of GPIO.
RBLE_VS_GPIO_Access	Accesses to the GPIO.
RBLE_VS_Flash_Management	Executes a Data Flash access management command.
RBLE_VS_Flash_Access	Accesses to Data Flash.
RBLE_VS_Flash_Operation	Executes a Data Flash block operation
RBLE_VS_Flash_Get_Space	Gets a free space of Data Flash.
RBLE_VS_Flash_Get_EEL_Ver	Gets the version of EEL.
RBLE_VS_Adapt_Enable	Enables / Disables adaptable function.
RBLE_VS_RF_Control	Controls the power supply of the RF chip.
RBLE_VS_Set_Params	Sets a parameter.

8.2.1 RBLE_VS_Enable

RBLE_STATUS RBLE_VS_Enable(RBLE_VS_EVENT_HANDLER callback)		
This function enables VS.		
Parameters:		
	callback	Specify the callback function that reports the VS event.
Return:		
	RBLE_OK	Success
	RBLE_PARAM_ERR	Invalid parameter
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.2 RBLE_VS_Test_Rx_Start

RBLE_STATUS RBLE_VS_Test_Rx_Start(uint8_t rx_freq)		
This function starts a reception test. The result is reported by using the reception test start completion event RBLE_VS_EVENT_TEST_RX_START_COMP.		
Parameters:		
	rx_freg	Reception frequency N = (F – 2402) / 2 (Range: 0x00 to 0x27, F: 2402 MHz to 2480 MHz)
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.3 RBLE_VS_Test_Tx_Start

RBLE_STATUS RBLE_VS_Test_Tx_Start(uint8_t tx_freq, uint8_t test_data_len, uint8_t pk_payload_type)		
This function starts a transmission test. The result is reported by using the transmission test start completion event RBLE_VS_EVENT_TEST_TX_START_COMP.		
Parameters:		
tx_freq	Transmission frequency N = (F – 2402) / 2 (Range: 0x00 to 0x27, F: 2402 MHz to 2480 MHz)	
test_data_len	Transmission packet payload length (0x00 to 0x25)	
pk_payload_type	RBLE_TEST_DATA_PATTERN_PN9	Pseudo random bit sequence 9
	RBLE_TEST_DATA_PATTERN_11110000	Bit pattern '11110000'
	RBLE_TEST_DATA_PATTERN_10101010	Bit pattern '10101010'
	RBLE_TEST_DATA_PATTERN_PN15	Pseudo random bit sequence 15
	RBLE_TEST_DATA_PATTERN_ALL1	All bits are 1
	RBLE_TEST_DATA_PATTERN_ALL0	All bits are 0
	RBLE_TEST_DATA_PATTERN_00001111	Bit pattern '00001111'
	RBLE_TEST_DATA_PATTERN_01010101	Bit pattern '01010101'
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

8.2.4 RBLE_VS_Test_End

RBLE_STATUS RBLE_VS_Test_End(void)	
This function ends the reception or transmission test being executed. The result is reported by using the test end event RBLE_VS_EVENT_TEST_END_COMP.	
Parameters:	
none	
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.5 RBLE_VS_Set_Test_Parameter

RBLE_STATUS RBLE_VS_Set_Test_Parameter(uint16_t rx_nb_packet, uint16_t tx_nb_packet, uint8_t infinite_setting)

This function sets parameters for the extended Direct Test Mode features.

The following extended features are available:

- Ends the reception test when the specified number of packets have been received.
  (The test ends when the specified number of packets have been received or the test end function RBLE_VS_Test_End is called.)
- Ends the transmission test when the specified number of packets have been transmitted.
  (The test ends when the specified number of packets have been transmitted or the test end function RBLE_VS_Test_End is called.)
- Performs burst transfer during a transmission or reception test.
  (The test ends when the test end function RBLE_VS_Test_End is called.)
- Performs continuous carrier wave (CW) output during a transmission test.
  (The test ends when the test end function RBLE_VS_Test_End is called.)

The result is reported by using the extended feature parameter setup completion event RBLE_VS_EVENT_SET_TEST_PARAM_COMP.

* This function must be called before executing Direct Test Mode.

Parameters:

rx_nb_packet	The number of packets at which to end a reception test (If 0 is specified, the test does not end automatically.)
tx_nb_packet	The number of packets at which to end a transmission test (If 0 is specified, the test does not end automatically.)
infinite_setting	0: Disable burst transfer, 1: Enable burst transfer, 2: Enable continuous carrier wave (CW) output

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.6 RBLE_VS_Read_Test_RSSI

RBLE_STATUS RBLE_VS_Read_Test_RSSI(void)

This function reads the RSSI value when reception Direct Test Mode is executed.

The result is reported by using the RSSI acquisition completion event in Direct Test mode RBLE_VS_EVENT_READ_TEST_RSSI_COMP.

* The RSSI value can be acquired in the period from when reception Direct Test Mode starts to immediately before a normal packet is received after exiting Direct Test Mode.

Parameters:

none

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.7 RBLE_VS_Write_Bd_Address

RBLE_STATUS RBLE_VS_Write_Bd_Address(RBLE_BD_ADDR *address)		
This function writes the specified public address to Data Flash. The result is reported by using the BD address write completion event RBLE_VS_EVENT_WR_BD_ADDR_COMP. * The BD address will be reflected when the GAP reset processing (RBLE_GAP_Reset) is finished after Bluetooth device restart. * Before calling this function, it is necessary to start the access to the Data Flash by using the RBLE_VS_Flash_Management. In addition, maintain buffer that is specified in the parameter until the BD address writing is completed.		
Parameters:		
	address	Public address to be stored in Data Flash
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.8 RBLE_VS_Set_Tx_Power

RBLE_STATUS RBLE_VS_Set_Tx_Power(uint16_t conhdl, uint8_t power_lvl, uint8_t state)
This function sets the transmit power for the procedure specified by the connection handle. The result is reported by using the transmit power setup completion event RBLE_VS_EVENT_SET_TX_POWER_COMP. * Note In the following cases, changing the transmit power of the connected device might cause unexpected behavior. - Expose the Tx Power Level using Proximity profile. - Contain the Tx Power Level AD type in Advertising data.

Parameters:

conhdl	Connection handle The transmit power during the Advertising, Scanning, or Initiating procedure can be set by specifying 0x10 for this parameter.
power_lvl	Transmit power level RBLE_VS_TXPW_LV1: -15dBm RBLE_VS_TXPW_LV2: -10dBm RBLE_VS_TXPW_LV3: -7dBm RBLE_VS_TXPW_LV4: -2dBm RBLE_VS_TXPW_LV5: Reserved RBLE_VS_TXPW_LV6: Reserved RBLE_VS_TXPW_LV7: -1dBm RBLE_VS_TXPW_LV8: Reserved RBLE_VS_TXPW_LV9: 0dBm
state	Operating state to set the transmit power level. RBLE_VS_TXPW_MODE_NORMAL: Adaptable function disabled RBLE_VS_TXPW_MODE_ADAPT_NEAR: RF low-power mode RBLE_VS_TXPW_MODE_ADAPT_MIDDLE: RF normal mode RBLE_VS_TXPW_MODE_ADAPT_FAR: RF high-performance mode

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.9 RBLE_VS_GPIO_Dir

RBLE_STATUS RBLE_VS_GPIO_Dir (uint8_t dir)		
This function sets the input/output direction of GPIO [3:0] pins on RF chip. When using alternate function, give priority to the following function. GPIO[0]: When using external power amplifier, give priority to TXSELH_RF function. GPIO[1]: When using external power amplifier, give priority to TXSELL_RF function. GPIO[2]: When using high-speed clock, give priority to CLKOUT_RF. GPIO[3]: When using 32kHz external subsystem clock, give priority to CLK32KIN/EXSLK_RF. Initial output value of the GPIO pins that sets output direction is Low(0). The result is reported by using the GPIO input/output direction setting completion event RBLE_VS_EVENT_GPIO_DIR_COMP. * Refer to the Bluetooth Low Energy Protocol Stack User's Manual about the alternate function.		
Parameters:		
	dir	Input/output direction of GPIO pins. (RBLE_VS_GPIO_INPUT: input, RBLE_VS_GPIO_OUTPUT: output) bit3: GPIO3 direction setting bit bit2: GPIO2 direction setting bit bit1: GPIO1 direction setting bit bit0: GPIO0 direction setting bit
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.10 RBLE_VS_GPIO_Access

RBLE_STATUS RBLE_VS_GPIO_Access (uint8_t mode, uint8_t value)		
This function acquires the input value or sets the output value of GPIO pins. Before calling this function, sets the direction of GPIO pins by using the RBLE_VS_GPIO_Dir. The result is reported by using the GPIO access completion event RBLE_VS_EVENT_GPIO_ACCESS_COMP. * When the RF chip has changed in Deep Sleep mode, GPIO[2:0] is reset to the input, and GPIO[3] is reset to alternate function output, the output value cannot be maintained. When the RF chip to wake-up from Deep Sleep mode, recover the output value set in this function.		
Parameters:		
	mode	Mode setting RBLE_VS_GPIO_INPUT_MD: Acquires the input value RBLE_VS_GPIO_OUTPUT_MD: Sets the output value
	value	Output value of GPIO pins (valid only output pins) bit3: GPIO3 output value bit bit2: GPIO2 output value bit bit1: GPIO1 output value bit bit0: GPIO0 output value bit
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.11 RBLE_VS_Flash_Management

RBLE_STATUS RBLE_VS_Flash_Management (uint8_t cmd)		
This function executes Data Flash access management functions. The result is reported by using the Data Flash access management command completion event RBLE_VS_EVENT_FLASH_MANAGEMENT_COMP. * During the period from the start of the access to the Data Flash to stop, SLEEP function is disabled.		
Parameters:		
	cmd	Data Flash access management command RBLE_VS_FLASH_CMD_START: Access start RBLE_VS_FLASH_CMD_STOP: Access stop
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.12 RBLE_VS_Flash_Access

RBLE_STATUS RBLE_VS_Flash_Access (RBLE_VS_FLASH_ACCESS_PARAM *param)		
This function writes the data to Data Flash or reads data from Data Flash. The result is reported by using the Data Flash access command completion event RBLE_VS_EVENT_FLASH_ACCESS_COMP. * Before calling this function, starts the access to Data Flash by using the RBLE_VS_Flash_Management. In addition, maintain buffer that is specified in the parameter until the data writing or reading is completed.		
Parameters:		
	cmd	Data Flash access command RBLE_VS_FLASH_CMD_WRITE: Writes the data RBLE_VS_FLASH_CMD_READ: Reads the data
	id	Data ID (0x01 – 0xFF)
	size	Data size (1 – 255 bytes)
	**addr*	Pointer to writing or reading buffer
Return:		
	RBLE_OK	Success
	RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.13 RBLE_VS_Flash_Operation

RBLE_STATUS RBLE_VS_Flash_Operation (uint8_t cmd)		
This function executes Data Flash block operation command. The result is reported by using the Data Flash block operation command completion event RBLE_VS_EVENT_FLASH_OPERATION_COMP. * Before calling this function, starts the access to Data Flash by using the RBLE_VS_Flash_Management. The BD address in Data Flash is stored at the time of cleanup execution and written to Data Flash once again after cleanup completion.		
Parameters:		
cmd	Data Flash block operation command RBLE_VS_FLASH_CMD_CLEANUP: Cleanup RBLE_VS_FLASH_CMD_FORMAT: Format	
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.	

8.2.14 RBLE_VS_Flash_Get_Space

RBLE_STATUS RBLE_VS_Flash_Get_Space (void)	
This function acquires the total free space of the current valid block and preparation block of Data Flash. The result is reported by using the Data Flash free space acquisition completion event RBLE_VS_EVENT_FLASH_GET_SPACE_COMP.	
Parameters:	
none	
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.15 RBLE_VS_Flash_Get_EEL_Ver

RBLE_STATUS RBLE_VS_Flash_Get_EEL_Ver (void)	
This function acquires the version information of EEPROM Emulation Library (EEL) used for Data Flash access. The result is reported by using the Data Flash EEL version acquisition completion event RBLE_VS_EVENT_FLASH_GET_EEL_VER_COMP.	
Parameters:	
none	
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.16 RBLE_VS_Adapt_Enable

RBLE_STATUS RBLE_VS_Adapt_Enable (uint8_t cmd)	
This function enables or disables the adaptable function. The result is reported by using the adaptable function enable completion event RBLE_VS_EVENT_ADAPT_ENABLE_COMP. * Note In the following cases, shall not enable this feature. - Expose the Tx Power Level using Proximity profile. - Contain the Tx Power Level AD type in Advertising data.	
Parameters:	
cmd	Adaptable function command RBLE_VS_ADAPT_CMD_DISABLE: Adaptable function disable RBLE_VS_ADAPT_CMD_ENABLE: Adaptable function enable, State indication enable RBLE_VS_ADAPT_CMD_ENABLE_WO_IND: Adaptable function enable, State indication disable
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.17 RBLE_VS_RF_Control

RBLE_STATUS RBLE_VS_RF_Control (uint8_t cmd)	
This function controls the power supply of the RF chip. The result is reported by using the RF power supply control completion event RBLE_VS_EVENT_RF_CONTROL_COMP. * Note During power OFF of the RF chip, will not be able to use the following functions. - External clock from RF part. - GPIO [3:0] pins on RF chip. - ke_timer function - Retransmission of RSCIP (at Modem configuration) * Note After setting to ON the RF power supply, please call the RBLE_GAP_Reset function.	
Parameters:	
cmd	RF power supply control command RBLE_VS_RFCNTL_CMD_POWDOWN: RF power supply OFF RBLE_VS_RFCNTL_CMD_POWUP_DDCON: RF power supply ON (DC-DC enable) RBLE_VS_RFCNTL_CMD_POWUP_DDCOFF: RF power supply ON (DC-DC disable)
Return:	
RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.2.18 RBLE_VS_Set_Params

RBLE_STATUS RBLE_VS_Set_Params (uint8_t param_id, uint8_t param_len, uint8_t *param_data)

This function sets the parameters in BLE MCU.

The result is reported by using the parameter setting completion event RBLE_VS_EVENT_SET_PARAMS_COMP.

It is possible to use freely more than 0x80 *param_id*. When more than 80 was set as *param_id*, will be called RBLE_User_Set_Params function of arch_main.c. Please implement any of processing and set a processing result as a return value.

Parameters:

param_id

Setting parameter ID

Setting parameter ID	Variable name
RBLE_VS_PARAM_DISC_SCAN_TIME	gap_discovery_scan_time
RBLE_VS_PARAM_DISC_SCAN_INTV	gap_dev_search_scan_intv
RBLE_VS_PARAM_DISC_SCAN_WIND	gap_dev_search_scan_window
RBLE_VS_PARAM_LIM_ADV_TO	gap_lim_adv_timeout
RBLE_VS_PARAM_SCAN_FAST_INTV	gap_scan_fast_intv
RBLE_VS_PARAM_SCAN_FAST_WIND	gap_scan_fast_window
RBLE_VS_PARAM_CONN_INTV_MIN	gap_init_conn_min_intv
RBLE_VS_PARAM_CONN_INTV_MAX	gap_init_conn_max_intv
RBLE_VS_PARAM_CONN_CE_MIN	gap_conn_min_ce_length
RBLE_VS_PARAM_CONN_CE_MAX	gap_conn_max_ce_length
RBLE_VS_PARAM_CONN_SLAVE_LATENCY	gap_conn_slave_latency
RBLE_VS_PARAM_CONN_SVTO	gap_dev_supervision_timeout
RBLE_VS_PARAM_RPA_INTV	gap_resolvable_private_addr_intv

* RBLE_VS_PARAM_USER_DEFINED_TOP (0x80) or more, it is possible to use freely.

param_len: Length of setting parameter

**param_data*: Pointer to the parameter data(the least significant byte first, left justified)

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Not executable because the rBLE mode is other than RBLE_MODE_ACTIVE.

8.3 Events

The following table shows the events defined for VS of rBLE and the following sections describe the events in detail.

Table 8-2 Events Defined for VS

RBLE_VS_EVENT_TEST_RX_START_COMP	Reception test start completion event
RBLE_VS_EVENT_TEST_TX_START_COMP	Transmission test start completion event
RBLE_VS_EVENT_TEST_END_COMP	Test end event
RBLE_VS_EVENT_WR_BD_ADDR_COMP	BD address write completion event
RBLE_VS_EVENT_SET_TEST_PARAM_COMP	Extended parameter setup completion event in Direct Test mode
RBLE_VS_EVENT_READ_TEST_RSSI_COMP	RSSI acquisition completion event in Direct Test Mode
RBLE_VS_EVENT_GPIO_DIR_COMP	GPIO input/output direction setting completion event
RBLE_VS_EVENT_GPIO_ACCESS_COMP	GPIO access completion event
RBLE_VS_EVENT_FLASH_MANAGEMENT_COMP	Data Flash data access command completion event
RBLE_VS_EVENT_FLASH_ACCESS_COMP	Data Flash data access command completion event
RBLE_VS_EVENT_FLASH_OPERATION_COMP	Data Flash block operation completion event
RBLE_VS_EVENT_FLASH_GET_SPACE_COMP	Data Flash free space acquisition completion event
RBLE_VS_EVENT_FLASH_GET_EEL_VER_COMP	Data Flash EEL version acquisition completion event
RBLE_VS_EVENT_ADAPT_ENABLE_COMP	Adaptable function enable completion event
RBLE_VS_EVENT_ADAPT_STATE_IND	Adaptable mode state change notification event
RBLE_VS_EVENT_COMMAND_DISALLOWED_IND	VS command disallowed notification event
RBLE_VS_EVENT_SET_TX_POWER_COMP	Transmit power setup completion event
RBLE_VS_EVENT_SET_PARAMS_COMP	Parameter setting completion event
RBLE_VS_EVENT_RF_CONTROL_COMP	RF power supply control completion event

8.3.1 RBLE_VS_EVENT_TEST_RX_START_COMP

RBLE_VS_EVENT_TEST_RX_START_COMP	
This event reports completion of starting a reception test.	
Parameters:	
status	Result of starting a reception test (See *3.2*, *Declaration of enumerated type for rBLE status*.)

8.3.2 RBLE_VS_EVENT_TEST_TX_START_COMP

RBLE_VS_EVENT_TEST_TX_START_COMP	
This event reports completion of starting a transmission test.	
Parameters:	
status	Result of starting a transmission test (See *3.2*, *Declaration of enumerated type for rBLE status*.)

8.3.3 RBLE_VS_EVENT_TEST_END_COMP

RBLE_VS_EVENT_TEST_END_COMP	
This event reports completion of the reception or transmission test being executed.	
Parameters:	
status	Result of ending a test (See *3.2*, *Declaration of enumerated type for rBLE status*.)
nb_packet_received	The number of packets received during the reception test * This parameter becomes invalid when a transmission test ends.

8.3.4 RBLE_VS_EVENT_WR_BD_ADDR_COMP

RBLE_VS_EVENT_WR_BD_ADDR_COMP	
This event reports completion of writing a BD address.	
Parameters:	
status	Result of writing the BD address (See *3.2*, *Declaration of enumerated type for rBLE status*.)

8.3.5 RBLE_VS_EVENT_SET_TEST_PARAM_COMP

RBLE_VS_EVENT_SET_TEST_PARAM_COMP	
This event reports completion of setting up the extended parameters for Direct Test Mode.	
Parameters:	
status	Result of setting up extended parameters for Direct Test mode (See *3.2*, *Declaration of enumerated type for rBLE status*.)

8.3.6 RBLE_VS_EVENT_READ_TEST_RSSI_COMP

RBLE_VS_EVENT_READ_TEST_RSSI_COMP		
This event reports completion of acquiring the RSSI value for reception Direct Test Mode.		
Parameters:		
	status	Result of acquiring the RSSI value for reception Direct Test Mode (See *3.2*, *Declaration of enumerated type for rBLE status*.)
	rssi	RSSI value (unit: dBm) * If *status* is not RBLE_OK, this parameter is invalid.

8.3.7 RBLE_VS_EVENT_GPIO_DIR_COMP

RBLE_VS_EVENT_GPIO_DIR_COMP		
This event reports completion of input/output direction setting of GPIO[3:0] pins on RF chip.		
Parameters:		
	status	Result of input/output direction setting of GPIO[3:0] pins (See *3.2*, *Declaration of enumerated type for rBLE status*.)
	mask	GPIO mask bit3: GPIO3 mask bit (1: GPIO, 0: uses alternate function) bit2: GPIO2 mask bit (1: GPIO, 0: uses alternate function) bit1: GPIO1 mask bit (1: GPIO, 0: uses alternate function) bit0: GPIO0 mask bit (1: GPIO, 0: uses alternate function)

8.3.8 RBLE_VS_EVENT_GPIO_ACCESS_COMP

RBLE_VS_EVENT_GPIO_ACCESS_COMP		
This event reports completion of acquiring the input value or setting the output value of GPIO pins.		
Parameters:		
	status	Result of acquiring the input value or setting the output value of GPIO pins (See *3.2*, *Declaration of enumerated type for rBLE status*.)
	value	GPIO input value (RBLE_VS_GPIO_LOW: 0, RBLE_VS_GPIO_HIGH: 1) bit3: GPIO3 input value bit bit2: GPIO2 input value bit bit1: GPIO1 input value bit bit0: GPIO0 input value bit

8.3.9 RBLE_VS_EVENT_FLASH_MANAGEMENT_COMP

RBLE_VS_EVENT_FLASH_MANAGEMENT_COMP		
This event reports completion of executing the Data Flash access management command.		
Parameters:		
	status	Result of executing the Data Flash access management command (See *3.2*, *Declaration of enumerated type for rBLE status*.)
	cmd	Execution command

8.3.10 RBLE_VS_EVENT_FLASH_ACCESS_COMP

RBLE_VS_EVENT_FLASH_ACCESS_COMP	
This event reports completion of executing the Data Flash access command.	
Parameters:	
status	Result of executing the Data Flash access command (See *3.2*, *Declaration of enumerated type for rBLE status*.)
cmd	Execution command
id	Data ID
size	Data size
**addr*	Pointer to data buffer

8.3.11 RBLE_VS_EVENT_FLASH_OPERATION_COMP

RBLE_VS_EVENT_FLASH_OPERATION_COMP	
This event reports completion of executing the Data Flash block operation command.	
Parameters:	
status	Result of executing the Data Flash block operation command (See *3.2*, *Declaration of enumerated type for rBLE status*.)
cmd	Execution command

8.3.12 RBLE_VS_EVENT_FLASH_GET_SPACE_COMP

RBLE_VS_EVENT_FLASH_GET_SPACE_COMP	
This event reports completion of acquiring the free space of Data Flash.	
Parameters:	
status	Result of acquiring the free space of Data Flash (See *3.2*, *Declaration of enumerated type for rBLE status*.)
wsize	Word size of free space (4bytes/word)

8.3.13 RBLE_VS_EVENT_FLASH_GET_EEL_VER_COMP

RBLE_VS_EVENT_FLASH_GET_EEL_VER_COMP	
This event reports completion of acquiring the EEL version information.	
Parameters:	
status	Result of acquiring the EEL version information (See *3.2*, *Declaration of enumerated type for rBLE status*.)
version[24]	Version information

8.3.14 RBLE_VS_EVENT_ADAPT_ENABLE_COMP

RBLE_VS_EVENT_ADAPT_ENABLE_COMP	
This event reports the result of enabling or disabling the adaptable function.	
Parameters:	
status	Result of enabling or disabling the adaptable function (See *3.2*, *Declaration of enumerated type for rBLE status*.)
cmd	Adaptable function enable / disable command

8.3.15 RBLE_VS_EVENT_ADAPT_STATE_IND

RBLE_VS_EVENT_ADAPT_ENABLE_COMP	
This event indicates change of adaptable state.	
Parameters:	
state	State of adaptable function

8.3.16 RBLE_VS_EVENT_COMMAND_DISALLOWED_IND

RBLE_VS_EVENT_COMMAND_DISALLOWED_IND	
This event indicates that a VS command was disallowed.	
Parameters:	
status	Result of command execution (See *3.2*, *Declaration of enumerated type for rBLE status*.)
opcode	Opcode of the disallowed command

8.3.17 RBLE_VS_EVENT_SET_TX_POWER_COMP

RBLE_VS_EVENT_SET_TX_POWER_COMP	
This event reports completion of setting up a transmit power.	
Parameters:	
status	Result of setting up a transmit power (See *3.2*, *Declaration of enumerated type for rBLE status*.)

8.3.18 RBLE_VS_EVENT_SET_PARAMS_COMP

RBLE_VS_EVENT_SET_PARAMS_COMP	
This event reports completion of setting up a parameter.	
Parameters:	
status	Result of setting up a parameter (See *3.2*, *Declaration of enumerated type for rBLE status*.)

8.3.19 RBLE_VS_EVENT_RF_CONTROL_COMP

RBLE_VS_EVENT_RF_CONTROL_COMP		
This event reports completion of control the power supply of the RF chip.		
Parameters:		
	status	Result of setting up a parameter (See *3.2*, *Declaration of enumerated type for rBLE status*.)

9. RWKE

This section describes the APIs of the RWKE (Renesas Wireless Kernel Extension).

The RWKE which is basic software designed for operating BLE protocol stacks serves as a simplified operating system based on pseudo multitasking (non-preemptive multitasking).

9.1 Type Declaration

```
typedef  uint32_t  evt_field_t ;                                  Kernel event field
typedef  void  ( * evt_ptr_t ) ( void ) ;                         Kernel event handler
typedef  uint16_t  ke_state_t ;                                   Task state
typedef  uint16_t  ke_task_id_t ;                                 Task identifier
typedef  uint16_t  ke_msg_id_t ;                                  Message identifier
typedef  int  ( * ke_msg_func_t ) ( const ke_msg_id_t msg,        Message handler
    const void *param,
    const ke_task_id_t dst,
    const ke_task_id_t src );
typedef  uint16_t  ke_time_t ;                                    Relative time (10-ms
                                                                  units)
```

9.2 Kernel Event Management

The RWKE provides the kernel event management functionality as a means to execute delay processing of interrupts.

The RWKE has a loop that is executed at all times (kernel event loop) and it confirms whether a kernel event was generated at every loop. When a kernel event is generated, the RWKE calls the corresponding kernel event handler and processes the kernel event, then it returns to the kernel event loop. When multiple kernel events occur simultaneously, the kernel event with a higher priority is processed first.

Kernel events are identified uniquely in the system based on the kernel event numbers from 0 to 31. The priority of kernel event number 0 is the highest and the priority of 31 is the lowest. In kernel event management APIs, instead of the kernel event number, the kernel event field of the evt_field_t type is used. There is the following correspondence relation between the kernel event number "evt" and the kernel event field "evt_field".

$$evt\_field = (uint32\_t)\ 1 << (31 - evt)$$

When multiple kernel event fields are specified, the logical sum of individual kernel event fields is calculated for each bit.

Table 9-1 Kernel Event Management

RWKE API Name	Functional Overview
ke_evt_get	Acquires the set state of kernel events.
ke_evt_set	Sets a kernel event.
ke_evt_clear	Clears a kernel event.

The kernel event handler is an evt_ptr_t type function. When the kernel event handler is called, processing for the kernel event is performed, ke_evt_clear is called, and the kernel event is cleared. Note that the kernel event handler will continue to be called until the kernel event is cleared.

Kernel events are not countable. In other words, when a kernel event is set but the corresponding kernel event handler has not been called yet, even if the same kernel event is set again, the corresponding kernel event handler will be called only

once.

9.2.1 ke_evt_get

evt_field_t ke_evt_get (void)	
Acquires the set state of kernel events.	
Parameters:	
none	
Return:	
The set state of kernel events is returned. The MSB of the evt_field_t return value sequentially corresponds with kernel event numbers 0, 1, 2, …, and 31, one bit each. A bit set to 1 indicates that the kernel event is set and a bit cleared to 0 indicates that the kernel event is cleared.	

9.2.2 ke_evt_set

void ke_evt_set (evt_field_t evt)	
Sets the kernel event specified with evt.	
Parameters:	
evt_field_t evt	The kernel event to be set. When multiple events are specified, specify the logical sum for each bit.
Return:	
None	

9.2.3 ke_evt_clear

void ke_evt_clear (evt_field_t evt)	
Clears the kernel event specified with evt.	
Parameters:	
evt_field_t evt	The kernel event to be cleared. When multiple events are specified, specify the logical sum for each bit.
Return:	
None	

9.3 Message Communication Management

The RWKE provides the message communication management functionality as a means to perform synchronization and communication between tasks or between a kernel event handler and a task.

When a task sends a message, that message is temporarily placed in the kernel message queue of the RWKE. After that, the message is retrieved from the kernel message queue by the message scheduler which is a kernel event handler and passed to the message handler of the receiving task. (The message handler of the receiving task is called with the pointer to the message used as an argument.)

A message is configured with the message body and the message header which contains information, such as the task identifier of the transmitting task, the task identifier of the receiving task, the message type, and the message length. The message type is the message category which has been uniquely defined by the transmitting task and receiving task.

A message is managed as the following structure shown in Table 9-1. The part with the (2) yellow background color is the message header and the part with (3) the blue background color is the message body. The actual size of the message body is param_len bytes. The part with the (1) red background color is the area used for RWKE management.

Table 9-1 Message Structure

```
struct ke_msg
{
    struct co_list_hdr hdr;     ///< List header for chaining
#if (BLE_SPLIT || BLE_FULLEMB)
    uint8_t  hci_type;          ///< Type of HCI data(used by the HCI only)
    int8_t  hci_off;            ///< Offset of the HCI data in the message
                                /// (used by the HCI only)
    uint16_t  hci_len;          ///< Length of the HCI traffic (used by the HCI only)
#endif
    ke_msg_id_t  id;            ///< Message id.
    ke_task_id_t  dest_id;      ///< Destination kernel identifier.
    ke_task_id_t  src_id;       ///< Source kernel identifier.
    uint16_t  param_len;        ///< Parameter embedded struct length.
    uint32_t  param[1];         ///< Parameter embedded struct.
                                /// Must be word-aligned.
};
```

(1)

(2)

(3)

Table 9-2 Message Communication Management

RWKE API Name	Functional Overview
ke_msg_alloc	Allocates a memory block for a message.
ke_msg_free	Releases a memory block for a message.
ke_msg_send	Sends a message.
ke_msg_send_basic	Sends a blank message (message with only a message header).
ke_msg_forward	Forwards a message.
ke_msg2param	Acquires the address of the message body from the start address of the message header.
ke_param2msg	Acquires the address of the message header from the start address of the message body.

The message handler is a ke_msg_func_t type function. When the message handler is called, the given message is processed, and one of the following values is returned.

KE_MSG_CONSUMED	The given message is processed. The RWKE deletes (releases) the message.
KE_MSG_NO_FREE	The given message is processed. The RWKE does not delete (release) the message.
KE_MSG_SAVED	The given message is not processed. The RWKE passes the message to the message handler again when the task state changes.

In the RWKE, a task is configured with a task descriptor and multiple message handlers. The task descriptor contains the task state and information on associating the message types and message handlers. The message scheduler searches for the task descriptor of the receiving task using the state of the receiving task at that point and the message type in the message as the keys, selects a suitable message handler, and passes the message.

9.3.1 ke_msg_alloc

void * ke_msg_alloc (ke_msg_id_t id, ke_task_id_t dest_id, ke_task_id_t src_id, uint16_t param_len)	
Allocates a memory block for a message.	
Parameters:	
ke_msg_id_t id	Message type to be sent
ke_task_id_t dest_id	Task identifier of the receiving task
ke_task_id_t src_id	Task identifier of the transmitting task
uint16_t param_len	Size of the area to be allocated for the message body *Set the allocatable size of ke_malloc.
Return:	
Start address of the message body in the memory block allocated for the message	

9.3.2 ke_msg_free

void ke_msg_free (const struct ke_msg *msg)	
Releases the memory block for a message which was allocated by ke_msg_alloc.	
Parameters:	
*const struct ke_msg *msg*	Start address of the memory block for a message which is to be released
Return:	
None	

9.3.3 ke_msg_send

void ke_msg_send (const void *param_ptr)	
Sends a message that includes the message body and is specified by param_ptr.	
Parameters:	
*const void *param_ptr*	Start address of the message body which is to be sent
Return:	
None	

9.3.4 ke_msg_send_basic

void ke_msg_send_basic (ke_msg_id_t id, ke_task_id_t dest_id, ke_task_id_t src_id)	
Sends a blank message (message with only a message header).	
Parameters:	
ke_msg_id_t id	Message type to be sent
ke_task_id_t dest_id	Task identifier of the receiving task
ke_task_id_t src_id	Task identifier of the transmitting task
Return:	
None	

9.3.5 ke_msg_forward

void ke_msg_forward (const void *param_ptr, ke_task_id_t dest_id, ke_task_id src_id)	
Forwards a message that includes the message body and is specified by param_ptr.	
Parameters:	
*const void *param_ptr*	Start address of the message body of the message to be transferred
ke_task_id_t dest_id	Task identifier of the transfer destination
ke_task_id_t src_id	Task identifier of the transfer source
Return:	
None	

9.3.6 ke_msg2param

void * ke_msg2param (const struct ke_msg *msg)	
Calculates the start address of the message body from the start address of the message that is specified by param_ptr.	
Parameters:	
*const struct ke_msg *msg*	Start address of the message
Return:	
Start address of the message body	

9.3.7 ke_param2msg

struct ke_msg * ke_param2msg (const void *param_ptr)	
Calculates the start address of the message from the start address of the message body that is specified by param_ptr.	
Parameters:	
*const void *param_ptr*	Start address of the message body
Return:	
Start address of the message	

9.4 Task State Management

Tasks are identified uniquely in the system based on the task types from 0 to 63. Each task can have a task index from 0 to 63 and a task can be changed into a multiple instance task. The instances of a task are identified uniquely in the system based on the task identifier of the ke_task_id_t type.

There is the following correspondence relation between the task type "type", task index "idx", and task identifier "task_id".

task_id = (idx << 8) | type

Normally, a task should be used with the task index set to 0.

Each instance of a task manages a single variable named “state” having the ke_state_t type. The value of the state has a different meaning for each instance of a task. Immediately after system initialization, the state value is normally 0.

The RWKE provides the task state management functionality as a means for the task to manage the state.

Table 9-3 Task State Management

RWKE API Name	Functional Overview
ke_state_get	References the task state.
ke_state_set	Sets (changes) the task state.

9.4.1 ke_state_get

ke_state_t ke_state_get (const ke_task_id_t task)	
Acquires the state of the task specified by "task".	
Parameters:	
const ke_task_id_t task	Task identifier of the task whose state is acquired
Return:	
Returns the task state.	

9.4.2 ke_state_set

void ke_state_set (const ke_task_id_t task, const ke_state_t state)	
Sets the state of the task specified by "task" in "state". * “task” that can be specified is only the “user task”. If specified the other than "user task", the operation is not guaranteed.	
Parameters:	
const ke_task_id_t task	Task identifier of the task whose state is set
const ke_state_t state	Value of the state that is set
Return:	
None	

9.5 Timer Management

The RWKE provides the timer management functionality as a means to execute time-dependent processing.

The timer management functionality provided by the RWKE sends a blank message to the specified task at the specified time. The actual processing is performed by the message handler of the task which has received the blank message.

When a task specifies the timer, a timer request block is created and placed in the kernel timer queue of the RWKE. After that, when the specified time is reached, the timer scheduler which is a kernel event handler retrieves the timer request block from the kernel timer queue, and a blank message is sent to the specified task.

Table 9-4 Timer Management

RWKE API Name	Functional Overview
ke_time	Acquires the current timer value.
ke_timer_set	Sets the timer.
ke_timer_clear	Cancels the set timer.

9.5.1 ke_time

ke_time_t ke_time (void)	
Acquires the current timer value.	
Parameters:	
None	
Return:	
Current timer value (10-ms units)	

9.5.2 ke_timer_set

void ke_timer_set (ke_msg_id_t timerid, ke_task_id_t task, ke_time_t delay)	
Sets the timer. After the period specified by "delay" has passed, a blank message of the timerid message type is sent to the task specified by "task".	
Parameters:	
ke_msg_id_t timerid	Message type of the message which is sent after the specified time has passed
ke_task_id task	Task to receive the message which is sent after the specified time has passed
ke_time_t delay	Time (10-ms units) *Set time from 1 to 29999.
Return:	
None	

9.5.3 ke_timer_clear

void ke_timer_clear (ke_msg_id_t timerid, ke_task_id_t task)	
Cancels the set timer.	
Parameters:	
ke_msg_id_t timerid	Message type of the set timer
ke_task_id_t task	Task to receive the set timer
Return:	
None	

9.6 Memory Management

The RWKE provides the memory management functionality as a means to dynamically manage memory.

The heap area is a single continuous area allocated in RAM. For the heap area, the start address is indicated by ke_mem_heap and (last address + 1) is indicated by ke_mem_heap_end.

```
extern uint8_t  ke_mem_heap[];
extern uint8_t  ke_mem_heap_end[];
```

The beginning of the memory block which is allocated from the heap area is aligned with the 2-byte boundary.

Table 9-5 Memory Management

RWKE API Name	Functional Overview
ke_malloc	Allocates a memory block.
ke_free	Releases a memory block.

9.6.1 ke_malloc

void * ke_malloc (size_t size)	
Allocates a memory block of the size specified by "size" from the heap area.	
Parameters:	
size_t size	Size of the memory block to be allocated *Upper limit is the size which is allocated for memory of user application in BLE_HEAP_SIZE.
Return:	
Start address of the allocated memory block	

9.6.2 ke_free

void ke_free (void *mem_ptr)	
Releases the memory block that was allocated by ke_malloc.	
Parameters:	
*void *mem_ptr*	Start address of the memory block to be released
Return:	
None	

9.7 Exclusive Control

The RWKE provides the exclusive control functionality which disables interrupts as a means to perform exclusive control between the main processing (message handler, event handler, etc.) and interrupt processing.

Interrupts can be disabled using the IE bit of PSW of the RL78/G1D. The ISP0 and ISP1 bits of PSW should not be changed. The RWKE does not intervene with execution of the interrupt processing.

Table 9-6 Exclusive Control

RWKE API Name	Functional Overview
GLOBAL_INT_START	Enables interrupts.
GLOBAL_INT_STOP	Disables interrupts.
GLOBAL_INT_DISABLE	Saves the interrupt disabled state (enabled or disabled) and disables interrupts.
GLOBAL_INT_RESTORE	Restores the interrupt disabled state.

Note 1: GLOBAL_INT_START and GLOBAL_INT_STOP are macros.
They can be used only when it is obvious that interrupts are disabled (enabled or disabled).

Note 2: GLOBAL_INT_DISABLE and GLOBAL_INT_RESTORE are macros.
They must be used as a pair in the same function. They can be nested.

9.8 Initialization and Event Loop Execution

The ke_init function needs to be called to initialize the RWKE before using the RWKE functionalities.

After initialization of the application system has finished, perform loop processing to continuously call the ke_evt_schedule function.

The simplified main function for executing the RWKE is as follows:

```
void main(void)
{
        ke_init();                          // Initialization of RWKE
        GLOBAL_INT_START();
        for ( ; ; ) {                       // Execution of RWKE event loop
                ke_evt_schedule();          //
        }                                   //
}
```

Table 9-7 Initialization and Event Loop Execution

Function Name	Functional Overview
ke_init	Initializes the RWKE.
ke_evt_schedule	Executes the kernel event loop processing of the RWKE once.

9.9 RWKE APIs Usable in Interrupt Processing

The APIs of the RWKE which can be used in the interrupt processing (maskable interrupts only) are shown below.

When an RWKE API other than those listed below is called in the interrupt processing, correct operation is not guaranteed.

ke_evt_set ke_evt_clear
ke_msg_alloc ke_msg_send ke_msg_send_basic

10. Notes

Appendix A Message Sequence Chart

A. 1 Initialization of BLE S/W

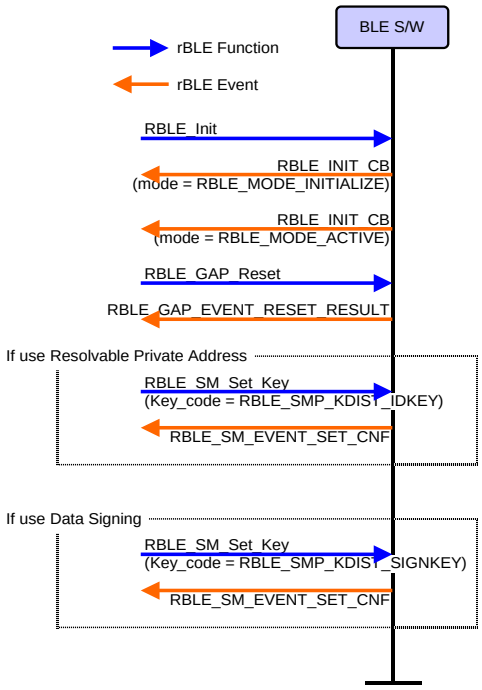

Figure A-1 Initialization of BLE S/W

A. 2 Broadcast Mode & Observation Procedure

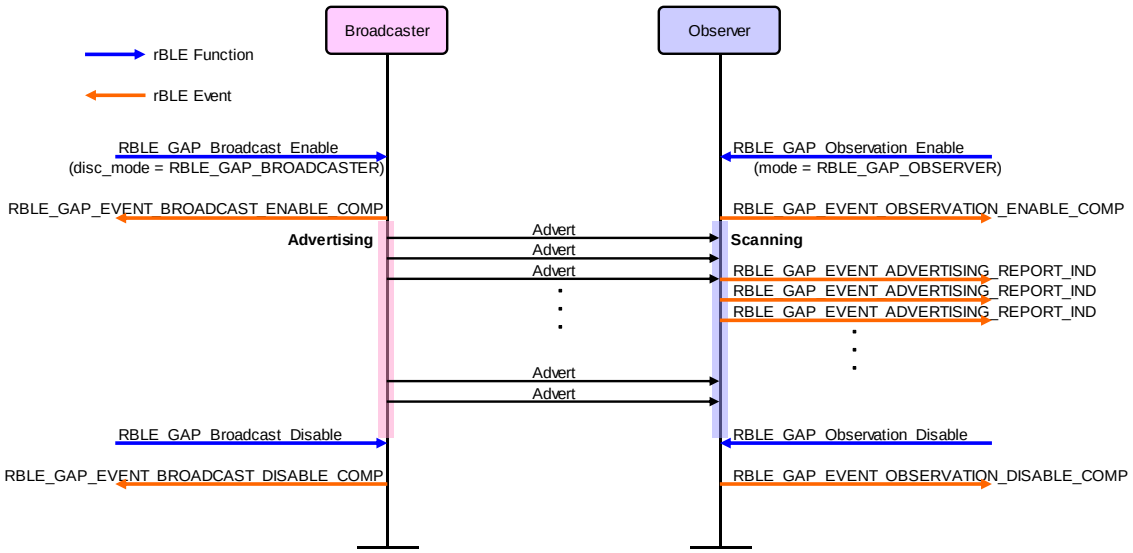

Figure A-2 Broadcast Mode & Observation Procedure

A. 3 General Discoverable Mode

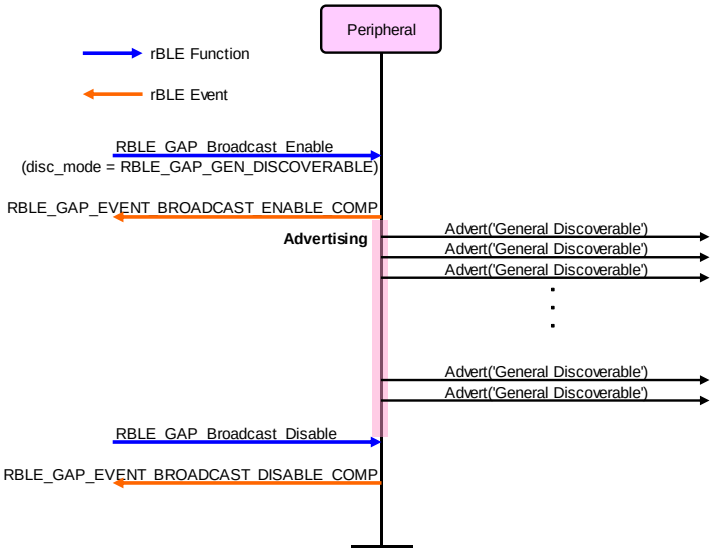

Figure A-3 General Discoverable Mode

A. 4 General Discovery Procedure

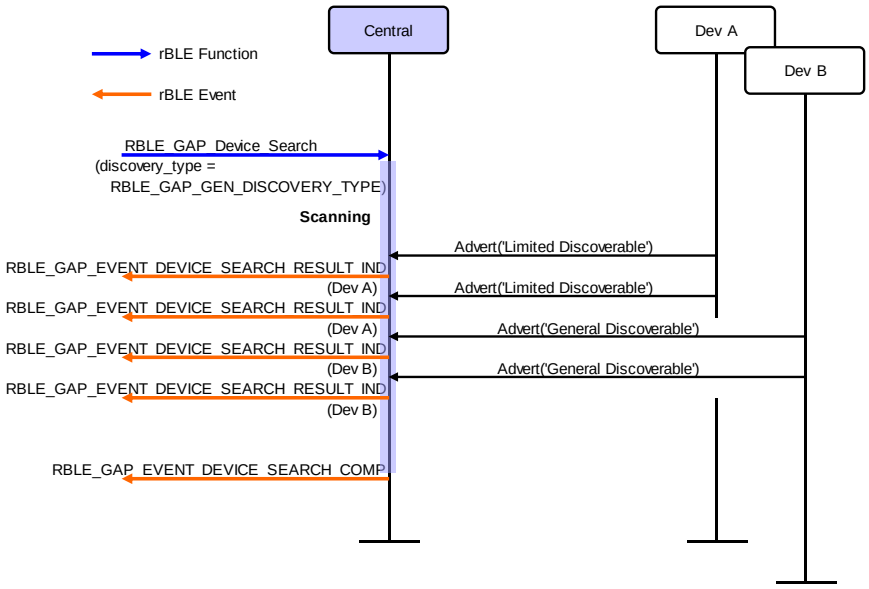

Figure A-4 General Discovery Procedure

A. 5 Limited Discovery Procedure

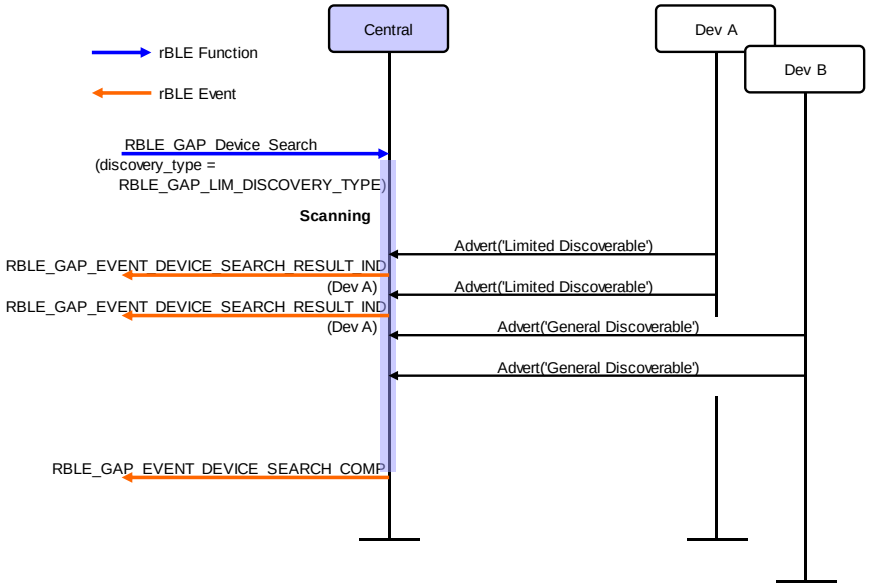

Figure A-5 Limited Discovery Procedure

A. 6 Name Discovery Procedure (Non-connected state)

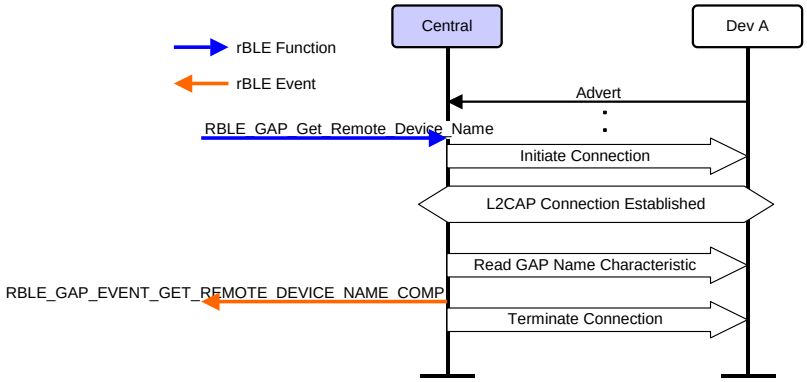

Figure A-6 Name Discovery Procedure (Non-connected state)

A. 7 Name Discovery Procedure (Connected state)

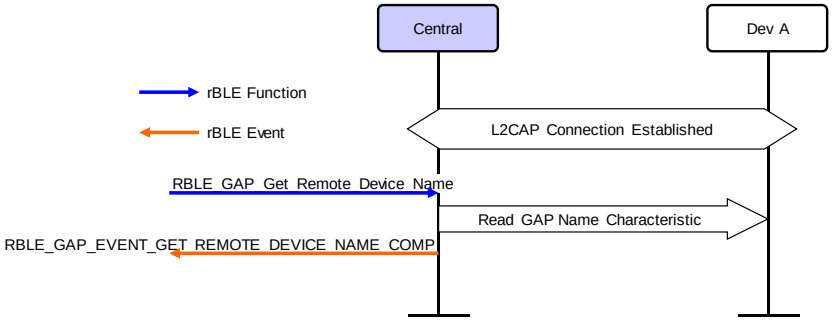

Figure A-7 Name Discovery Procedure (Connected state)

A. 8 General Connection Establishment Procedure

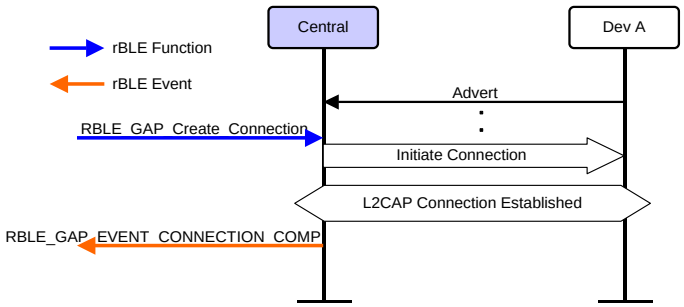

Figure A-8 General Connection Establishment Procedure

A. 9 Terminate Connection Procedure

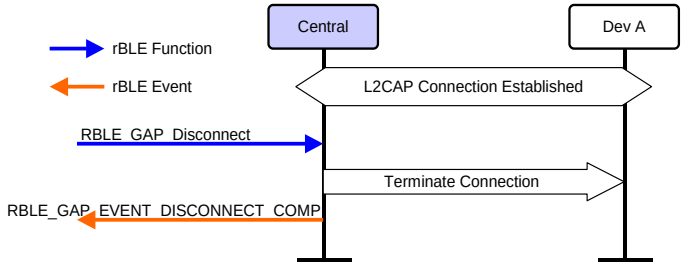

Figure A-9 Terminate Connection Procedure

A. 10 Auto Connection Establishment Procedure

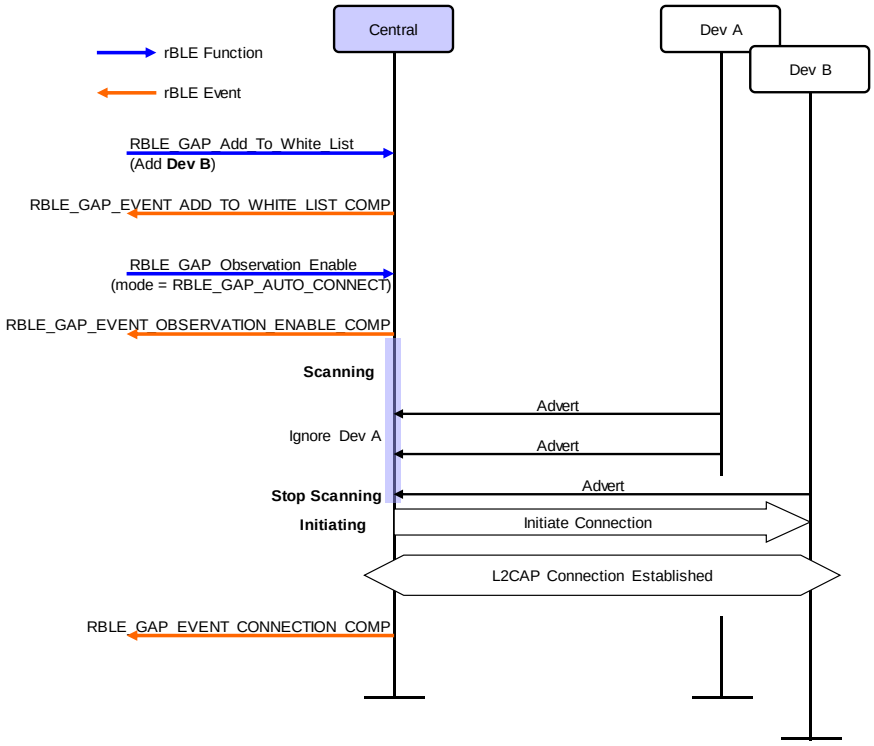

Figure A-10 Auto Connection Establishment Procedure

A. 11 Connection Parameter Update Procedure - Central initiate

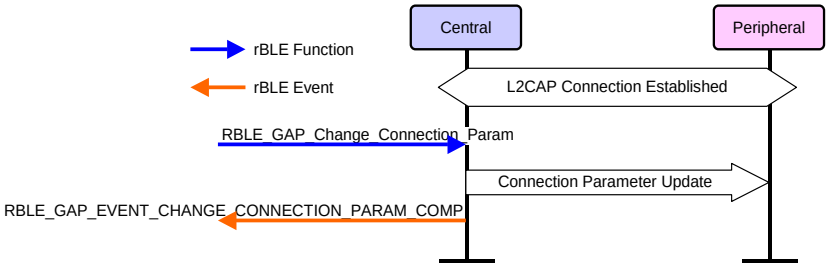

Figure A-11 Connection Parameter Update Procedure - Central initiate

A. 12 Connection Parameter Update Procedure - Peripheral request

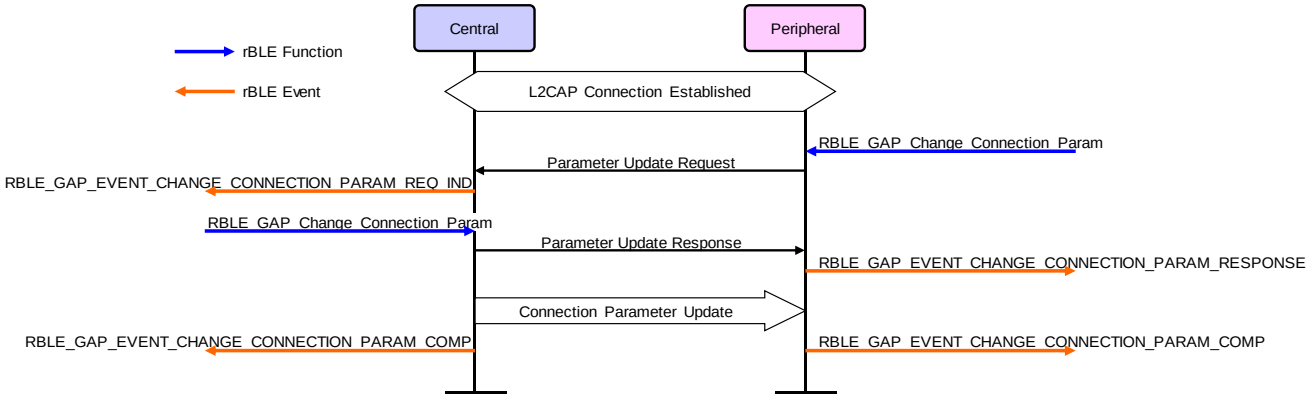

Figure A-12 Connection Parameter Update Procedure - Peripheral request

A. 13 Bonding Procedure - Central Initiate

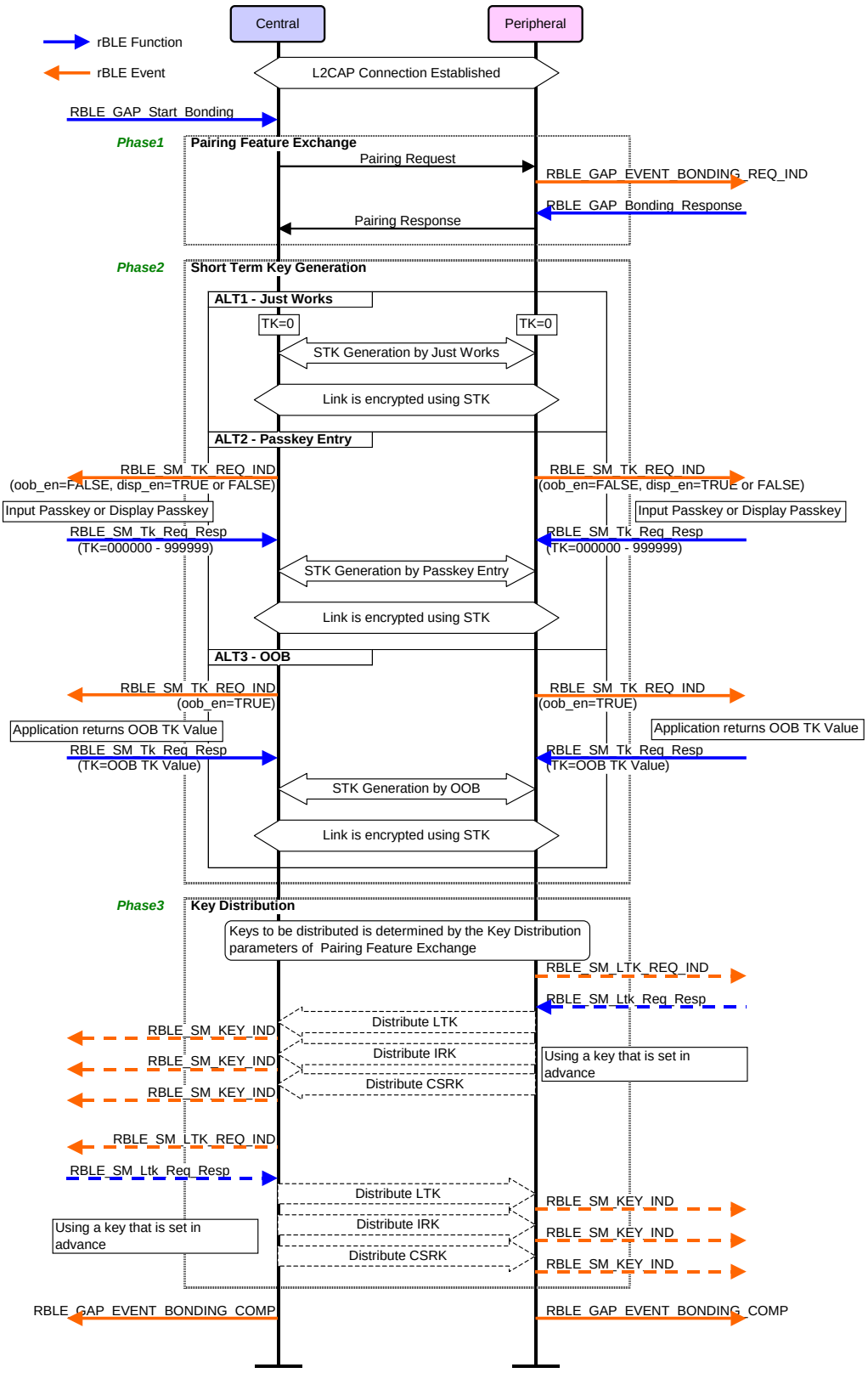

Figure A-13 Bonding Procedure - Central Initiate

A. 14 Bonding Procedure - Peripheral Request

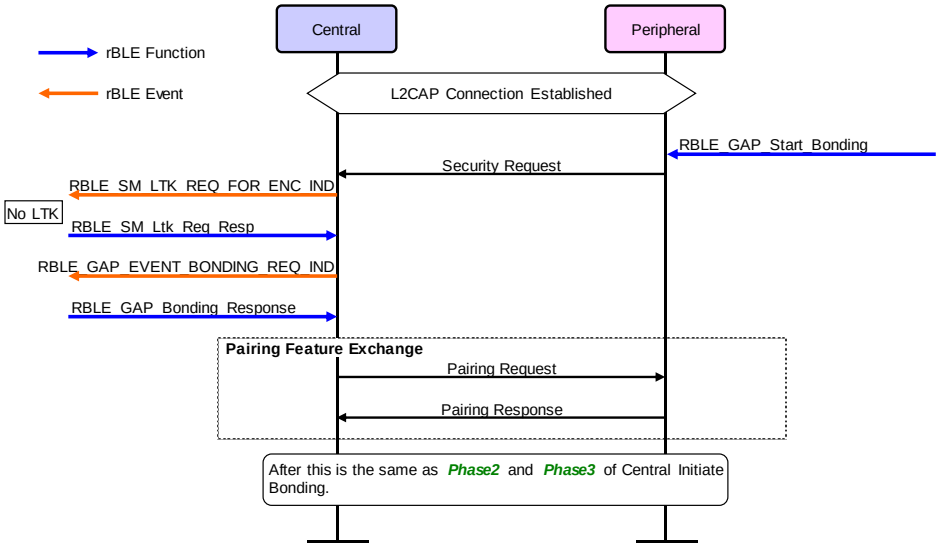

Figure A-14 Bonding Procedure - Peripheral Request

A. 15 Bonding Procedure - Central Initiate, Peripheral Reject

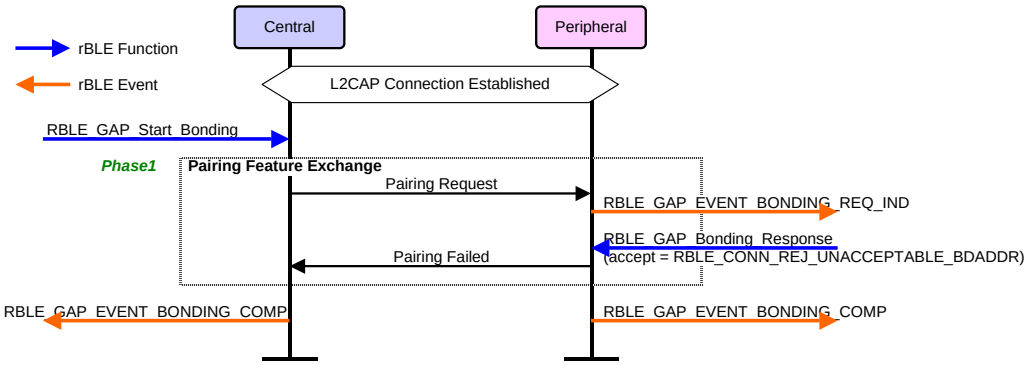

Figure A-15 Bonding Procedure - Central Initiate, Peripheral Reject

A. 16 Bonding Procedure - Peripheral Request, Central Reject

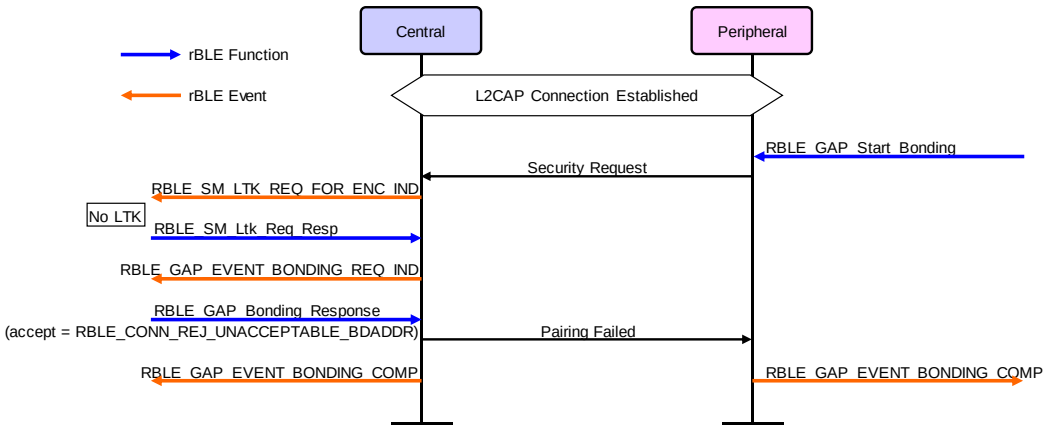

Figure A-16 Bonding Procedure - Peripheral Request, Central Reject

A. 17 Central Initiated Link Layer Encryption

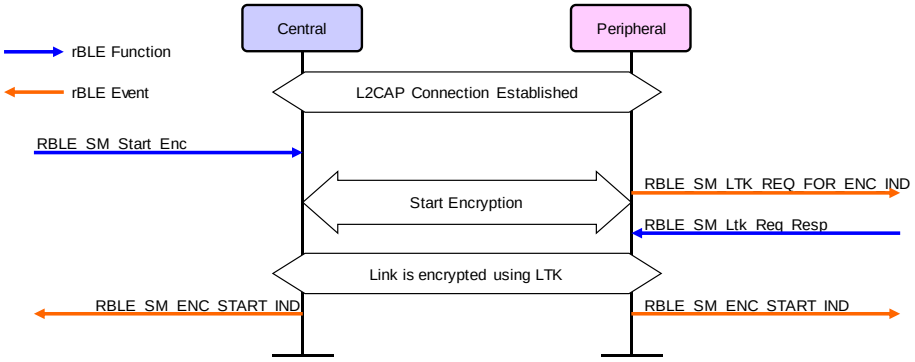

Figure A-17 Central Initiated Link Layer Encryption

A. 18 Peripheral request, Central Initiated Link Layer Encryption

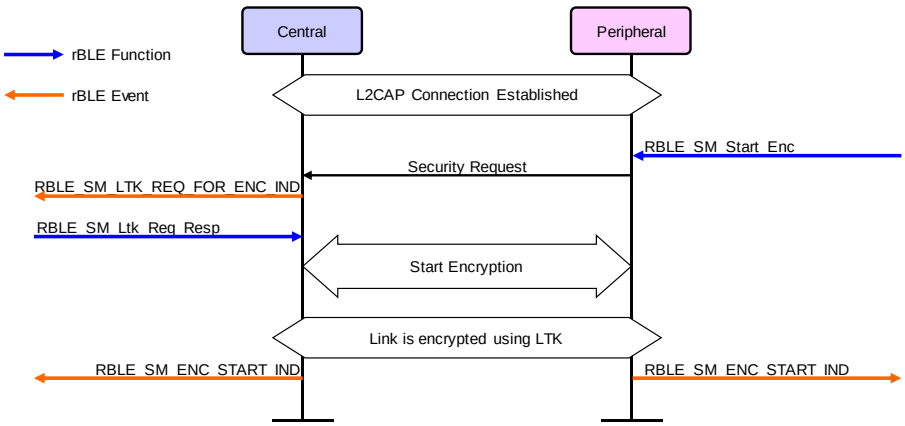

Figure A-18 Peripheral request, Central Initiated Link Layer Encryption

A. 19 GATT Discover All Primary Services

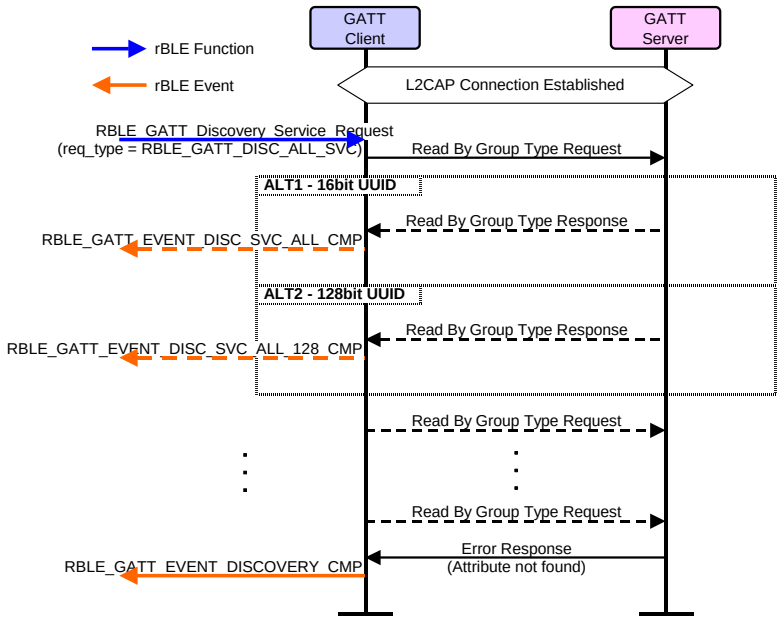

Figure A-19 Discover All Primary Services

A. 20 GATT Discover Primary Services by UUID

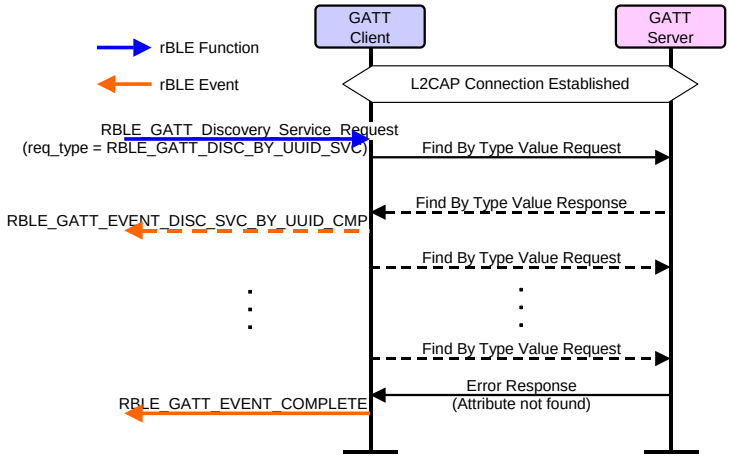

Figure A-20 Discover Primary Services by UUID

A. 21 GATT Discover Included Services

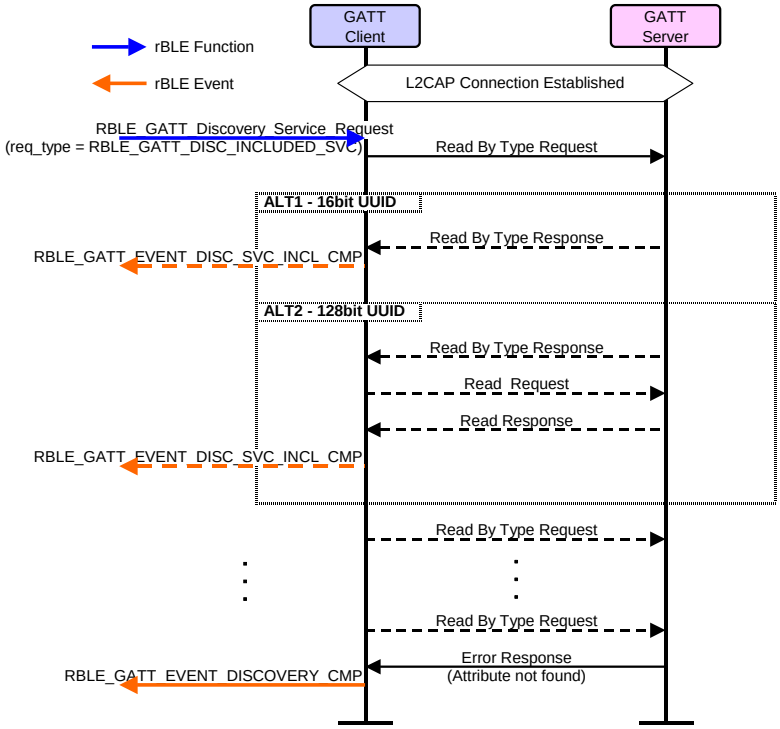

Figure A-21 Discover Included Services

A. 22 GATT Discover All Characteristics

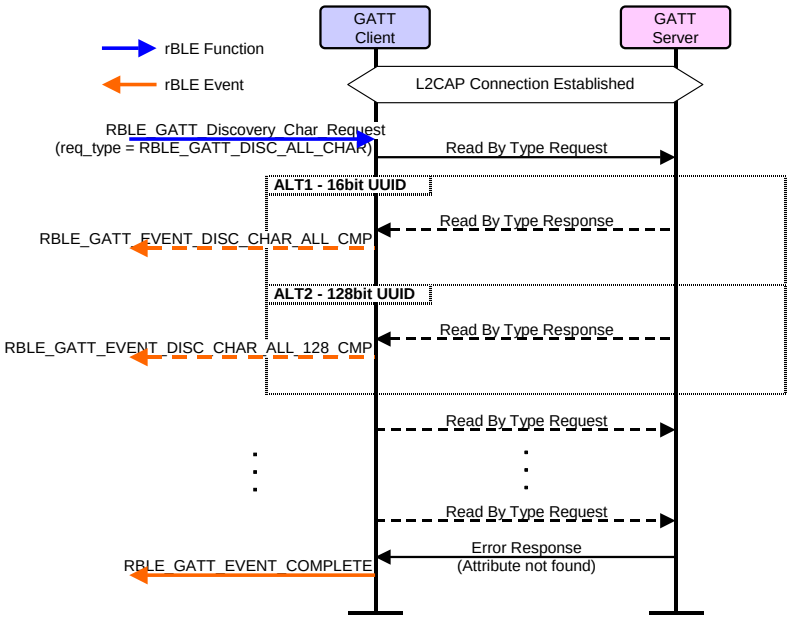

Figure A-22 Discover All Characteristics

A. 23 GATT Discover Characteristics by UUID

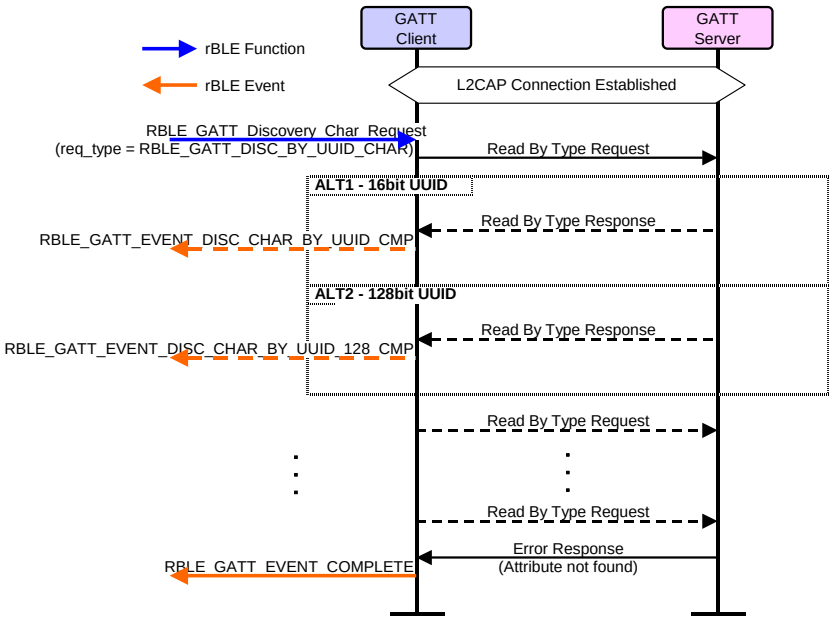

Figure A-23 Discover Characteristics by UUID

A. 24 GATT Read Characteristic Value

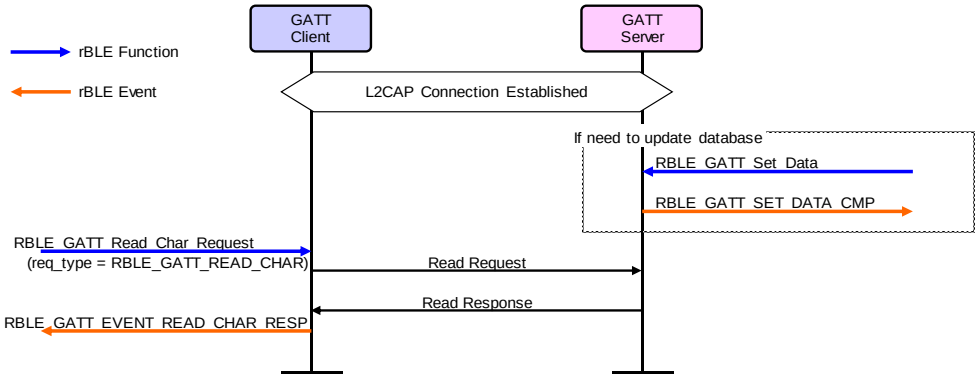

Figure A-24 Read Characteristic Value

A. 25 GATT Read Using Characteristic UUID

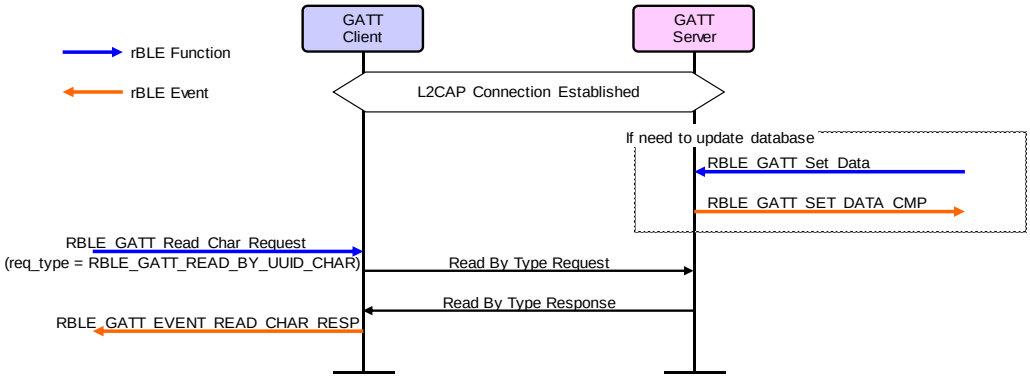

Figure A-25 Read Using Characteristic UUID

A. 26 GATT Read Long Characteristic Values

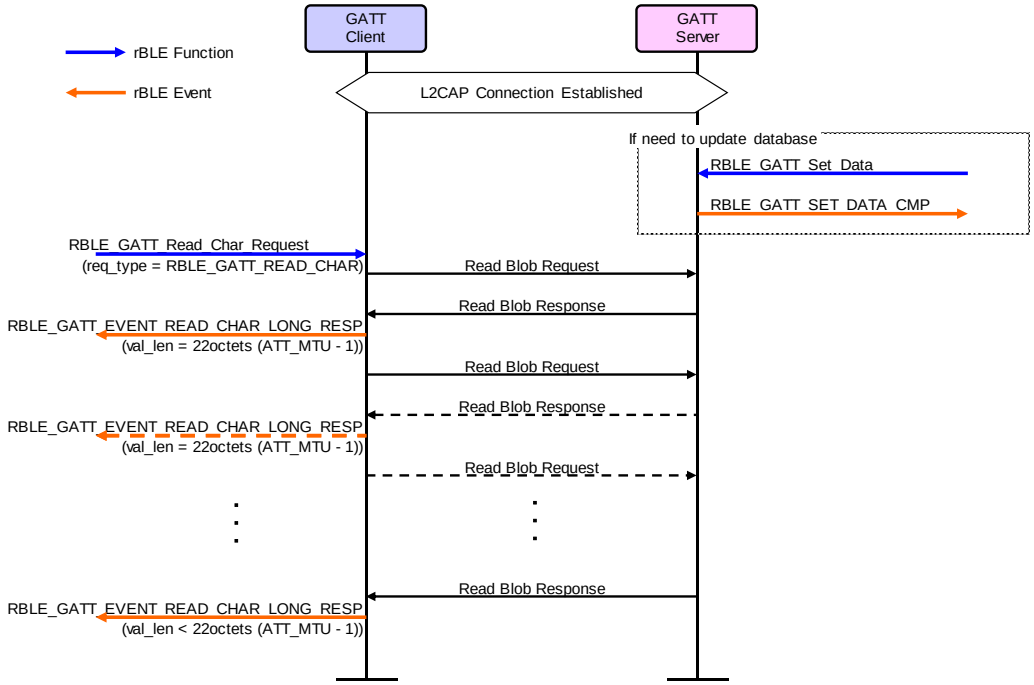

Figure A-26 Read Long Characteristic Values

A. 27 GATT Read Multiple Characteristic Values

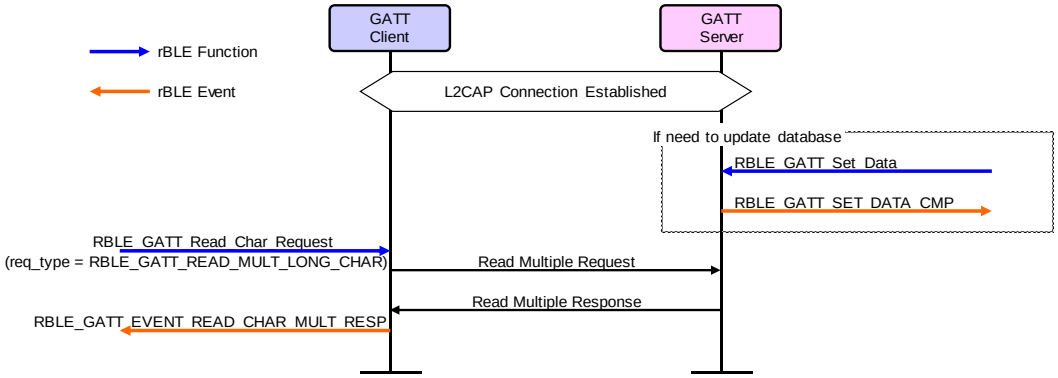

Figure A-27 Read Multiple Characteristic Values

A. 28 GATT Read Characteristic Descriptors

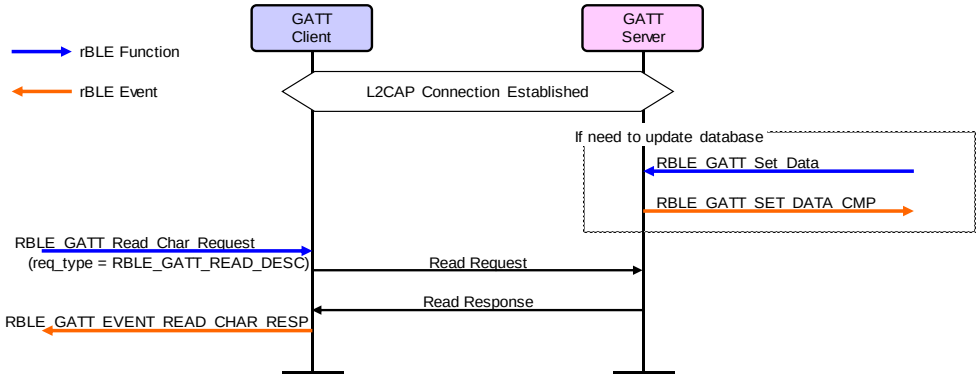

Figure A-28 Read Characteristic Descriptors

A. 29 GATT Read Long Characteristic Descriptors

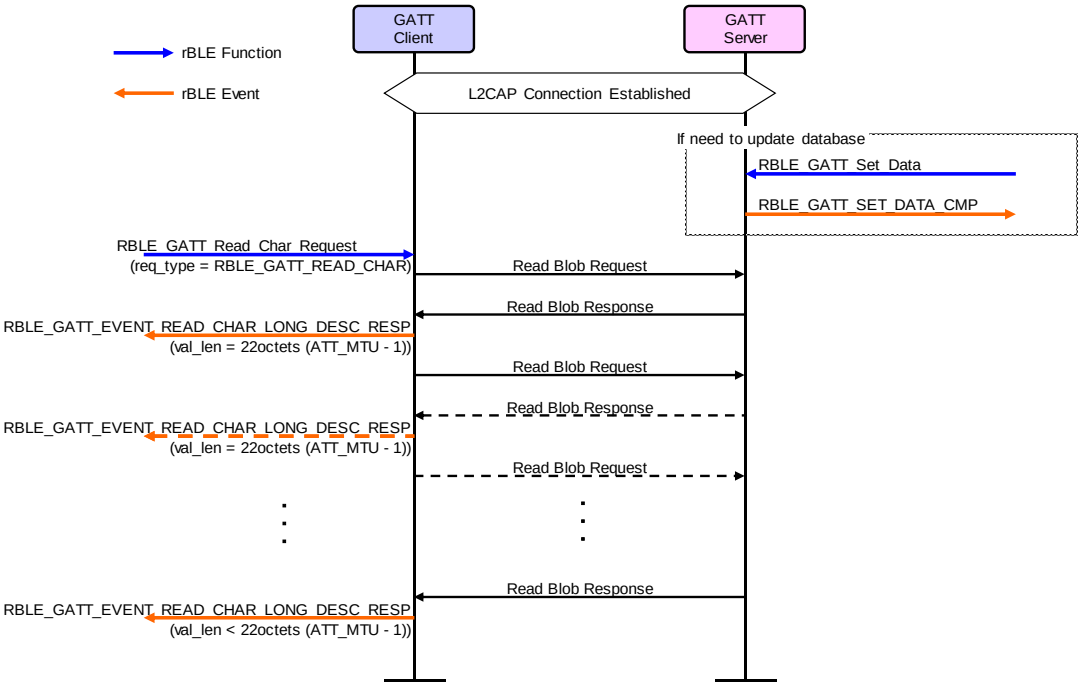

Figure A-29 Read Long Characteristic Descriptors

A. 30 GATT Write Without Response

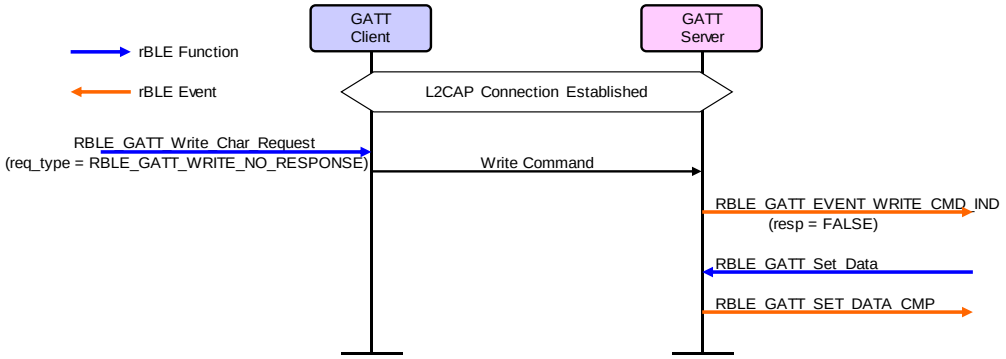

Figure A-30 Write Without Response

A. 31 GATT Signed Write Without Response

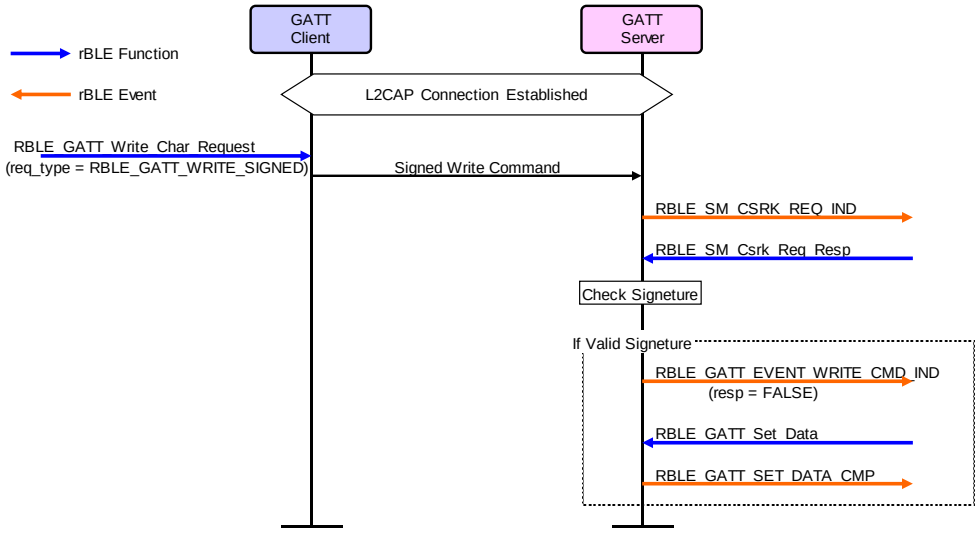

Figure A-31 Signed Write Without Response

A. 32 GATT Write Characteristic Value / Write Characteristic Descriptor

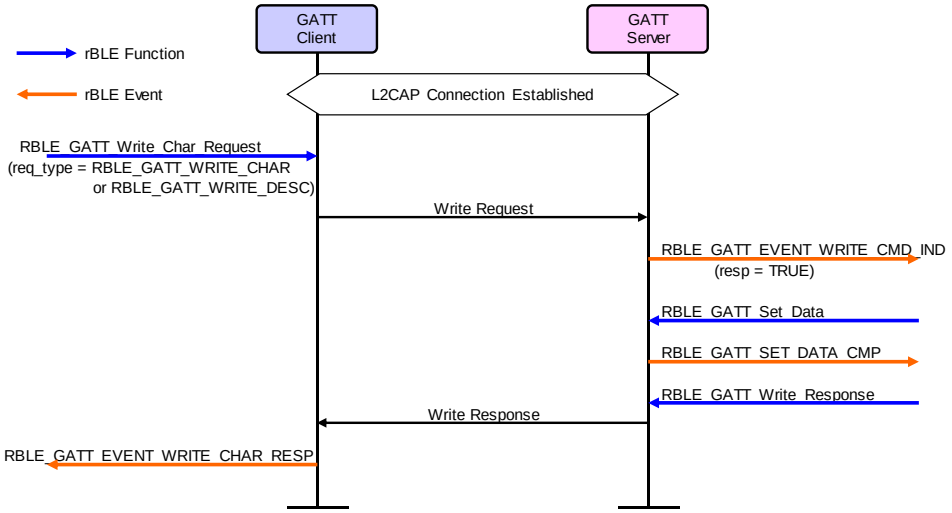

Figure A-32 Write Characteristic Value / Write Characteristic Descriptor

A. 33 GATT Write Long Characteristic Value / Write Long Characteristic Descriptor

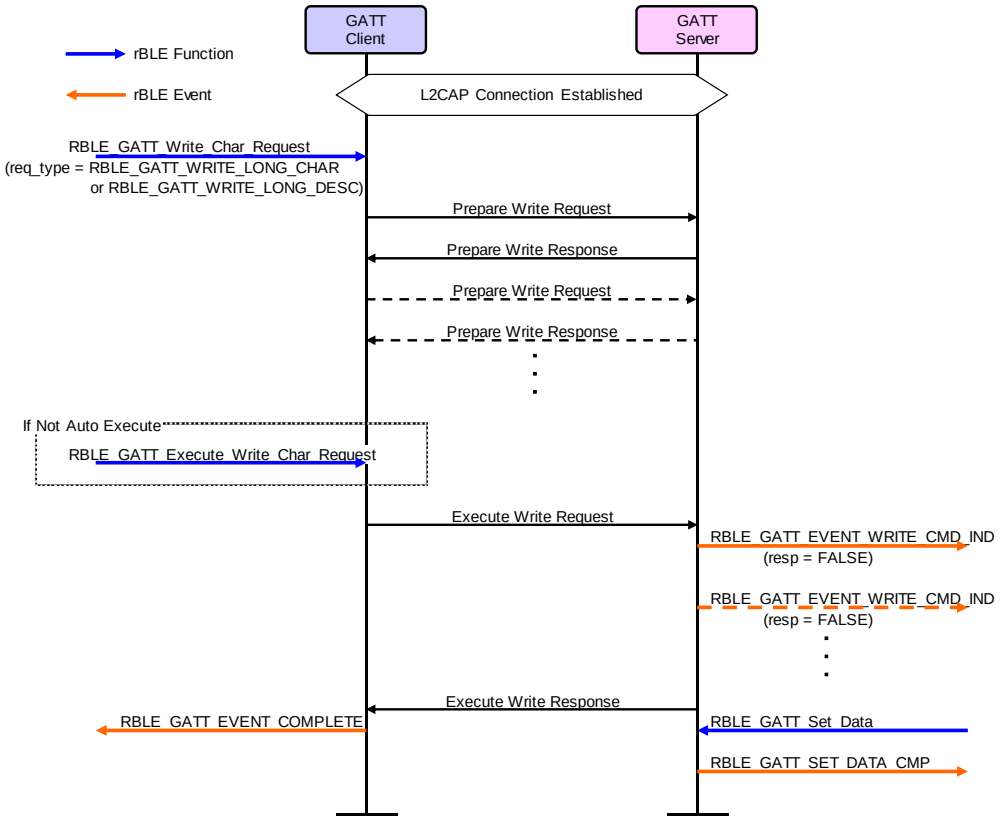

Figure A-33 Write Long Characteristic Value / Write Long Characteristic Descriptor

A. 34 GATT Reliable Writes

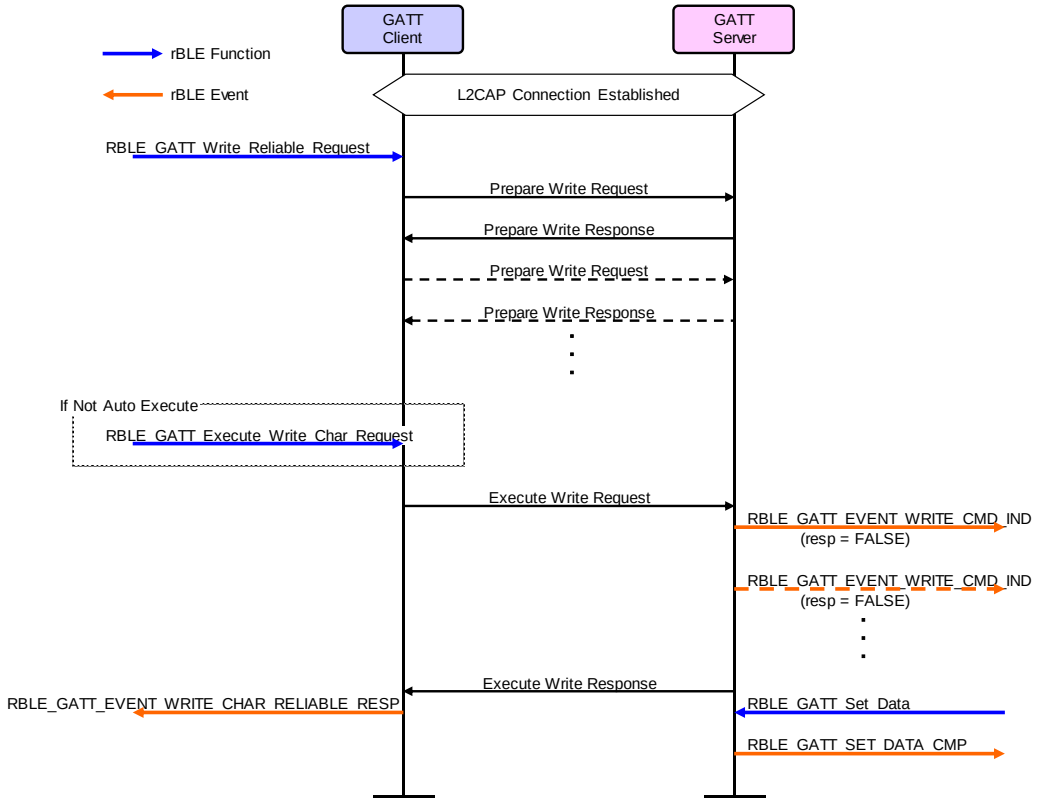

Figure A-34 Reliable Writes

A. 35 GATT Notifications

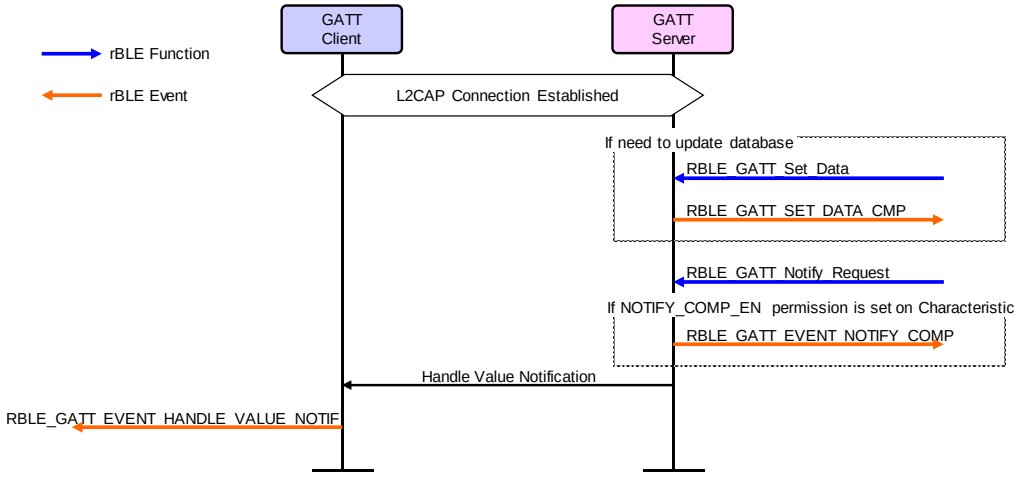

Figure A-35 Notifications

A. 36 GATT Indications

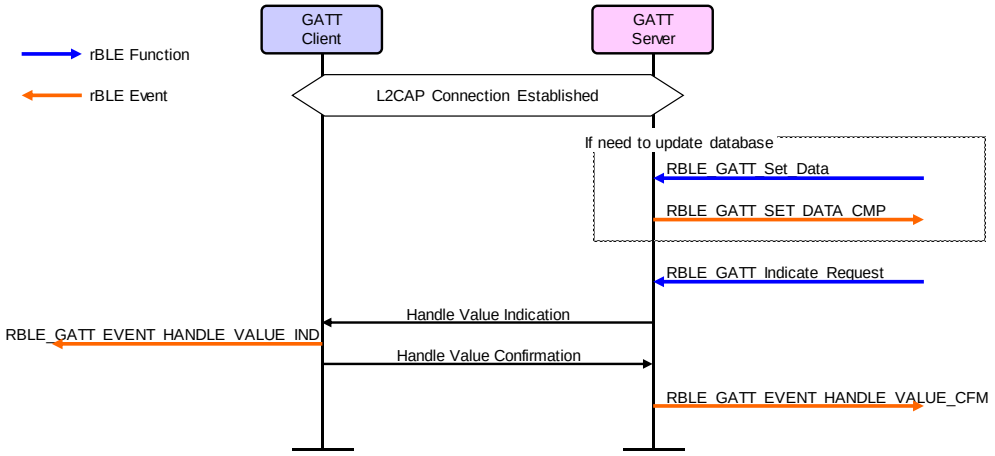

Figure A-36 Indications

A. 37 Receiver Test

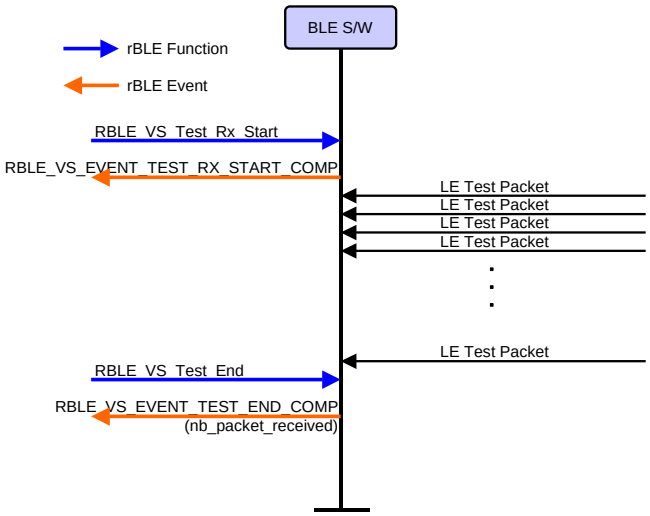

Figure A-37 Receiver Test

A. 38 Transmitter Test

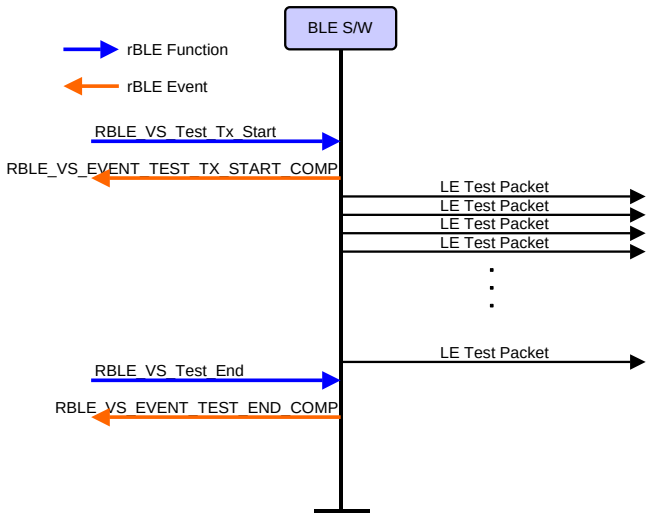

Figure A-38 Transmitter Test

A. 39 Extended Receiver Test

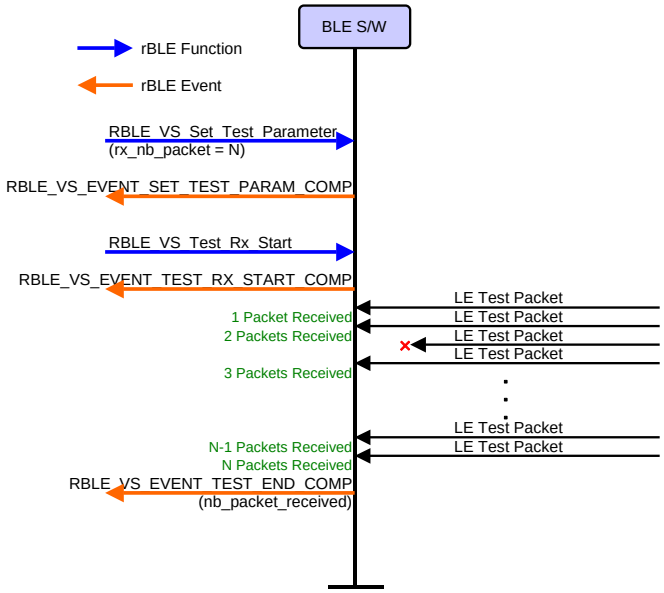

Figure A-39 Extended Receiver Test

A. 40 Read RSSI during Receiver Test

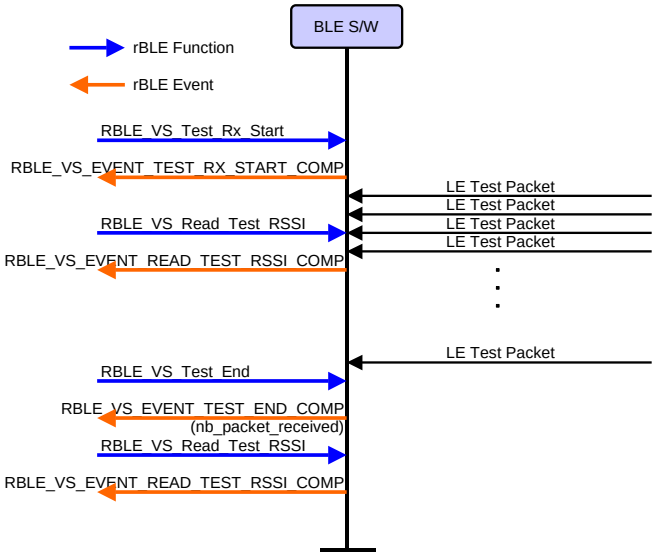

Figure A-40 Read RSSI during Receiver Test

Appendix B How to Read Definition Tables

This section shows how to read the tables that describes the rBLE API functions and events shown in this document.

B.1 How to Read Function Definition Tables

The following contents are included in the function definition tables:

The Parameters area describes the parameters specified for the function.
The italicized character strings on the left are the parameters of the function. The meaning of each parameter is described on the far right following the variables.

The italicized character string(s) next to each parameter indicate the member(s) of the parameter (structure).

The values that can be specified for the parameter might be described between the parameter name and its description.

The function definition is shown at the top of the table in the row with the light green background. This area shows the function prototype.				
The operation of the function and the event reported after executing the function are described in this area.				
Parameters:				
	Parameter 1	Description of parameter 1		
	Parameter 2	*Member 1*	Value 1 that can be specified for member 1	Description of value 1 that can be specified for member 1
			Value 1 that can be specified for member 2	Description of value 1 that can be specified for member 2
		Member 2	Description of member 2	
Return:				
	Value 1 that might be returned	Description of value 1 that might be returned		
	Value 2 that might be returned	Description of value 2 that might be returned		

The Return area describes the values returned for the function.
The leftmost row shows the value that might be returned, and the next row describes the return value.

B.2 How to Read Event Definition Tables

The following contents are included in the event definition tables:

The Parameters area describes the parameters specified for the event.
The italicized character strings on the left show the parameters of the event parameter structure. The meaning of each parameter is described on the far right.

The italicized character string(s) next to each parameter indicate the member(s) of the parameter (structure).

The event definition is shown at the top of the table in the row with the orange background. This area shows the event type.			
The information reported by the event is described in this area.			
Parameters:			
	Parameter 1	Description of parameter 1	
	Parameter 2	*Member 1*	Description of member 1
		Member 2	Description of member 2
		Member 3	Description of member 3
	Parameter 3	Value 1 that can be specified for parameter 3	Description of value 1 that can be specified for parameter 3
		Value 2 that can be specified for parameter 3	Description of value 2 that can be specified for parameter 3

The values that can be specified for the parameter might be shown between the parameter name and its description.

Appendix C Referenced Documents

1. Bluetooth Core Specification v4.2, Bluetooth SIG
2. Find Me Profile Specification v1.0, Bluetooth SIG
3. Immediate Alert Service Specification v1.0, Bluetooth SIG
4. Proximity Profile Specification v1.0, Bluetooth SIG
5. Link Loss Service Specification v1.0, Bluetooth SIG
6. Tx Power Service Specification v1.0, Bluetooth SIG
7. Health Thermometer Profile Specification v1.0, Bluetooth SIG
8. Health Thermometer Service Specification v1.0, Bluetooth SIG
9. Device Information Service Specification v1.1, Bluetooth SIG
10. Blood Pressure Profile Specification v1.0, Bluetooth SIG
11. Blood Pressure Service Specification v1.0, Bluetooth SIG
12. HID over GATT Profile Specification v1.0, Bluetooth SIG
13. HID Service Specification v1.0, Bluetooth SIG
14. Battery Service Specification v1.0, Bluetooth SIG
15. Scan Parameters Profile Specification v1.0, Bluetooth SIG
16. Scan Parameters Service Specification v1.0, Bluetooth SIG
17. Heart Rate Profile Specification v1.0, Bluetooth SIG
18. Heart Rate Service Specification v1.0, Bluetooth SIG
19. Cycling Speed and Cadence Profile Specification v1.0, Bluetooth SIG
20. Cycling Speed and Cadence Service Specification v1.0, Bluetooth SIG
21. Cycling Power Profile Specification v1.0, Bluetooth SIG
22. Cycling Power Service Specification v1.0, Bluetooth SIG
23. Glucose Profile Specification v1.0, Bluetooth SIG
24. Glucose Service Specification v1.0, Bluetooth SIG
25. Time Profile Specification v1.0, Bluetooth SIG
26. Current Time Service Specification v1.0, Bluetooth SIG
27. Next DST Change Service Specification v1.0, Bluetooth SIG
28. Reference Time Update Service Specification v1.0, Bluetooth SIG
29. Alert Notification Service Specification v1.0, Bluetooth SIG
30. Alert Notification Profile Specification v1.0, Bluetooth SIG
31. Location and Navigation Service Specification v1.0, Bluetooth SIG
32. Location and Navigation Profile Specification v1.0, Bluetooth SIG
33. Phone Alert Status Service Specification v1.0, Bluetooth SIG
34. Phone Alert Status Profile Specification v1.0, Bluetooth SIG
35. Bluetooth SIG Assigned Numbers https://www.bluetooth.org/Technical/AssignedNumbers/home.htm
36. Services & Characteristics UUID http://developer.bluetooth.org/gatt/Pages/default.aspx
37. Personal Health Devices Transcoding White Paper v1.2, Bluetooth SIG

Appendix D Terminology

Term	Description
Characteristic	A characteristic is a value used to identify services. The characteristics to be exposed and their formats are defined by each service.
Role	Each device takes the role prescribed by the profile or service in order to implement the specified use case.
Connection Handle	This is the handle determined by the controller stack and is used to identify connection with a remote device. The valid handle range is between 0x0000 and 0x0EFF.
Universally Unique Identifier	This is an identifier for uniquely identifying an item. In the BLE standard, a 16-bit UUID is defined for identifying services and their characteristics.
Bluetooth Device Address	This is a 48-bit address for identifying a Bluetooth device. The BLE standard defines both public and random addresses, and at least one or the other must be supported.
Public Address	This is an address that includes an allocated 24-bit OUI (Organizationally Unique Identifier) registered with the IEEE.
Ramdom Address	This is an address that contains a random number and belongs to one of the following three categories: Static Address Non-Resolvable Private Address Resolvable Private Address
Static Address	This is an address whose 2 most significant bits are both 1, and whose remaining 46 bits form a random number other than all 1's or all 0's. This static address cannot be changed until the power is switched off.
Non-Resolvable Private Address	This is an address whose 2 most significant bits are both 0, and whose remaining 46 bits form a random number other than all 1's or all 0's. Static addresses and public addresses must not be equal. This type of address is used to make tracking by an attacker difficult by changing the address frequently.
Resolvable Private Address	This is an address generated from an IRK and a 24-bit random number. Its 2 most significant bits are 0 and 1, and the remaining higher 22 bits form a random number other than all 1's or all 0's. The lower 24 bits are calculated based on an IRK and the higher random number. This type of address is used to make tracking by an attacker difficult by changing the address frequently. By allocating an IRK to the peer device, the peer device can identify the communicating device by using that IRK.
Broadcaster	This is one of the roles of GAP. It is used to transmit advertising data.
Observer	This is one of the roles of GAP. It is used to receive advertising data.
Central	This is one of the roles of GAP. It is used to establish a physical link. In the link layer, it is called Master.
Peripheral	This is one of the roles of GAP. It is used to accept the establishment of a physical link. In the link layer, it is called Slave.
Advertising	Advertising is used to transmit data on a specific channel for the purpose of establishing a connection or performing data transmission.

Term	Description
Scan	Scans are used to receive advertising data. There are two types of scans: Passive scan, in which data is simply received, and active scan, in which additional information is requested by sending SCAN_REQ.
White List	By registering known devices that are connected or bonded to a White List, it is possible to filter devices that can accept advertising data or connection requests.
Device Name	This is a user-friendly name freely assigned to a Bluetooth device to identify it. In the BLE standard, the device name is exposed to the peer device by the GATT server as a GAP characteristic.
Reconection Address	If a non-resolvable private address is used and the address is changed frequently, not only attackers but also the peer device will have difficulty identifying the device. Therefore, the address to be used at reconnection is reported by setting a new reconnection address as the exposed reconnection address characteristic.
Scan Interval	This is the interval for receiving advertising data.
Scan Window	This is the period of time during which advertising data is received at the scan interval.
Connecton Interval	This is the interval for transmitting and receiving data periodically following connection establishment.
Connecton Event	This is the period of time during which data is transmitted and received at the connection interval.
Slave Latency	This is the period of time during which data is transmitted and received at the connection interval.
Supervision Timeout	This is the timeout interval after which the link is considered to have been lost when no response is received from the peer device.
Passkey Entry	This is a pairing method whereby a six-digit number is input by each device to the other, or a six-digit number is displayed by one of the devices and that number is input to the other device.
Just Works	This is a pairing method that does not require user action.
OOB	This is a pairing method whereby pairing is performed by using data obtained by a communication method other than Bluetooth.
Identity Resolving Key	This is a 128-bit key used to generate and resolve resolvable private addresses.
Connection Signature Resolving Key	This is a 128-bit key used to create data signatures and verify the signature of incoming data.
Long Term Key	This is a 128-bit key used for encryption. The key size to be used is the size agreed on during pairing.
Short Term Key	This is a 128-bit key used for encryption during key exchange. It is generated using TK.
Temporary Key	This is a 128-bit key used required for STK generation. In the case of Just Works, the TK value is 0. In the case of Passkey Entry, it is the 6-digit number that was input, and in the case of OOB, it is the OOB data.

REVISION HISTORY	Bluetooth Low Energy Protocol Stack API Reference Manual: Basics

Rev.	Date	Description	
		Page	Summary
0.80	Sep 19, 2012	---	First Edition issued
1.10	Mar 19, 2013	---	The description about the high-speed access to the service for a second or subsequent time is added.
1.11	Jun 28, 2013	---	Fixed parameter in the stack is clarified. The scope of the function arguments for RWKE is clarified.
1.12	Sep 06, 2013	---	Added members to RBLE_ATT_ERR_CODE_enum.
1.13	Nov 29, 2013	--- --- 16 4,5 174	Added document name to Related documents. Added member to List of Abbreviations and Acronyms. Added definitions to Characteristic UUID definitions and Service UUID definitions Table 2-1, Table 2-2 is changed. Added document name to Appendix C Referenced Documents.
1.14	Sep 19, 2014	16 22 41 46 47 48 52 58 61 62 61 67 75 82	Added definitions to Characteristic UUID definitions and Service UUID definitions Removed the following members from the definition of the GAP event types. - RBLE_GAP_EVENT_KNOWN_ADDRESS_IND - RBLE_GAP_EVENT_KNOWN_DEVICE_SEARCH_RESULT_IND - RBLE_GAP_EVENT_SET_RECONNECT_ADDRESS_COMP - RBLE_GAP_EVENT_SET_PERIPHERAL_PRIVACY_FEATURE_COMP Added the following members to the definition of the GAP event types. - RBLE_GAP_EVENT_RPA_RESOLVED - RBLE_GAP_EVENT_WR_CHAR_IND Removed GAP characteristic UUID definition. Added definition of GAP characteristics codes. Removed the following structures from the GAP event parameter structure. - Known device address notification event parameter sturucture - Known device search result notification event parameter sturucture Added the following structures to the GAP event parameter structure. - Resolvable Private Address resolution completion event parameter sturucture - GAP characteristic write indication event parameter sturucture Removed the description for setting value of advertising interval in each mode. Changed the argument of RBLE_GAP_Get_Remote_Device_Name function. Removed the description of RBLE_GAP_KNOWN_DEV_DISCOVERY_TYPE. Removed the RBLE_GAP_Set_Reconnect_Address function. Removed the addr_visible argument and changed the description of the RBLE_GAP_Set_Privacy_Feature function. Remove the RBLE_GAP_Set_Peripheral_Privacy_Feature function. Removed the nb_bond argument of the RBLE_GAP_Bonding_Info_Ind function. Remove the following events. - RBLE_GAP_EVENT_KNOWN_ADDRESS_IND - RBLE_GAP_EVENT_KNOWN_DEVICE_SEARCH_RESULT_IND - RBLE_GAP_EVENT_SET_RECONNECT_ADDRESS_COMP - RBLE_GAP_EVENT_SET_PERIPHERAL_PRIVACY_FEATURE_COMP Added the following events. - RBLE_GAP_EVENT_RPA_RESOLVED - RBLE_GAP_EVENT_WR_CHAR_IND Added the lk_sec_status argument of the RBLE_SM_Irk_Req_Resp function. Added definition of expected response data size on GATT read multiple. Removed definition of enumerated type for expected response data size on GATT read multiple.

		101	Removed the description of RBLE_GATT_Read_Char_Request function.
1.15	Jan 30, 2015	11	Added the following members to the definition of the RBLE_STATUS types. - RBLE_GATT_INVALID_TYPE_IN_SVC_SEARCH - RBLE_GATT_ATTRIBUTE_CLIENT_MISSING - RBLE_GATT_ATTRIBUTE_SERVER_MISSING - RBLE_GATT_RELIABLE_WRITE_ERR - RBLE_GATT_BUFF_OVER_ERR
		22	Changed the following members to the definition of the GAP Advertising types. - RBLE_GAP_ADV_CONN_DIR →RBLE_GAP_ADV_CONN_DIR_HIGH_DUTY Added the following members to the definition of the GAP Advertising types. - RBLE_GAP_ADV_CONN_DIR_LOW_DUTY Added the following members to the definition of the GAP device discovery types. - RBLE_GAP_CANCEL_DISCOVERY
		55	Added the RBLE_GAP_Authorized_Ind function.
		75	Added the lk_sec_status argument of the RBLE_SM_Csrk_Req_Resp function.
		82	Added definitions of 32bit UUID octet length. Changed the enumerated type for GATT attribute permission to definisions. Added the following structures to the GATT event parameter structure. - Charcteristic notification completion event parameter sturucture Added the connection handle to the following event parameters.
		108	- RBLE_GATT_EVENT_DISC_SVC_ALL_CMP - RBLE_GATT_EVENT_DISC_SVC_ALL_128_CMP - RBLE_GATT_EVENT_DISC_SVC_BY_UUID_CMP - RBLE_GATT_EVENT_DISC_SVC_INCL_CMP - RBLE_GATT_EVENT_DISC_CHAR_ALL_CMP - RBLE_GATT_EVENT_DISC_CHAR_ALL_128_CMP - RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_CMP - RBLE_GATT_EVENT_DISC_CHAR_BY_UUID_128_CMP - RBLE_GATT_EVENT_DISC_CHAR_DESC_CMP - RBLE_GATT_EVENT_DISC_CHAR_DESC_128_CMP - RBLE_GATT_EVENT_READ_CHAR_RESP - RBLE_GATT_EVENT_READ_CHAR_LONG_RESP - RBLE_GATT_EVENT_READ_CHAR_MULT_RESP - RBLE_GATT_EVENT_READ_CHAR_LONG_DESC_RESP - RBLE_GATT_EVENT_WRITE_CHAR_RESP - RBLE_GATT_EVENT_WRITE_CHAR_RELIABLE_RESP - RBLE_GATT_EVENT_CANCEL_WRITE_CHAR_RESP - RBLE_GATT_EVENT_DISCOVERY_CMP - RBLE_GATT_EVENT_COMPLETE - RBLE_GATT_EVENT_WRITE_CMD_IND Added the following events.
		117	- RBLE_GATT_EVENT_NOTIFY_COMP
		118	Added definitions of GPIO and Data Flash
		129	Changed the description of RBLE_VS_Write_Bd_Address function.
		130	Added the state argument of the RBLE_VS_Set_Tx_Power function.
		131	Added the following functions. - RBLE_VS_GPIO_Dir - RBLE_VS_GPIO_Access - RBLE_VS_Flash_Management - RBLE_VS_Flash_Access - RBLE_VS_Flash_Operation - RBLE_VS_Flash_Get_Space - RBLE_VS_Flash_Get_EEL_Ver - RBLE_VS_Set_Params Added the following events.

			- RBLE_VS_EVENT_GPIO_DIR_COMP - RBLE_VS_EVENT_GPIO_ACCESS_COMP - RBLE_VS_EVENT_FLASH_MANAGEMENT_COMP - RBLE_VS_EVENT_FLASH_ACCESS_COMP - RBLE_VS_EVENT_FLASH_OPERATION_COMP - RBLE_VS_EVENT_FLASH_GET_SPACE_COMP - RBLE_VS_EVENT_FLASH_GET_EEL_VER_COMP - RBLE_VS_EVENT_SET_PARAMS_COMP
1.16	Apr, 17, 2015	 118 134 140	Added the following the definitions for VS. - enum RBLE_VS_ADAPT_STATE_enum - enum RBLE_VS_ADAPT_CMD_enum Added the following functions. - RBLE_VS_Adapt_Enable Added the following events. - RBLE_VS_EVENT_ADAPT_ENABLE_COMP - RBLE_VS_EVENT_ADAPT_STATE_IND Removed the following MSCs. - Known Device Discovery Procedure (Remote uses Public Address) - Known Device Discovery Procedure (Remote uses Resolvable Private Address) Changed the MSC of “Connection Parameter Update Procedure - Peripheral request”.
1.17	Oct 30, 2015	41 118 129 134 141	Changed the description of RBLE_GAP_Broadcast_Enable function. Changed the following the definitions for VS. - enum RBLE_VS_ADAPT_CMD_enum Added the following the definitions for VS. - enum RBLE_VS_RFCNTL_CMD_enum Added the following functions. Changed the description of RBLE_VS_Write_Bd_Address function. - RBLE_VS_RF_Control Added the following events. - RBLE_VS_EVENT_RF_CONTROL_COMP Changed the MSC of “Broadcast Mode & Observation Procedure”.
1.18	Aug 31, 2016	22 41 47 49 62 79 99 135 143 145 148 152 159 159 160 160 163 163	Changed the definition of struct RBLE_CONNECT_INFO_t. Fixed the description of the advertising type. Fixed the description of the discovery type. Fixed the description of the supervison timeout. Added the following parameters for connection completion event. - role - idx Added the RBLE_SM_LTK_REQ_FOR_ENC_IND event. Fixed the description of the end handle. Fixed the description of the setting parameter ID. Fixed the misspelled function name. Fixed the description of the task identifiers. Fixed the misspelled variable type. Added the following the sequence chart in Appendix A. - Name Discovery Procedure (Connected state) Changed the following the sequence chart in Appendix A for RBLE_SM_LTK_REQ_FOR_ENC_IND. - Bonding Procedure - Peripheral Request - Bonding Procedure - Peripheral Request, Central Reject - Central Initiated Link Layer Encryption - Peripheral request, Central Initiated Link Layer Encryption Fixed the following the sequence chart in Appendix A for Set Data sequence. - GATT Read Characteristic Value - GATT Read Using Characteristic UUIC

		164	- GATT Read Long Characteristic Values
		164	- GATT Read Multiple Characteristic Values
		165	- GATT Read Characteristic Descriptors
		165	- GATT Read Long Characteristic Descriptors
		166	- GATT Write Writhout Response
		166	- GATT Signed Write Without Response
		167	- GATT Write Characteristic Value / Write Characteristic Descriptor
		167	- GATT Write Long Characteristic Value / Write Long Characteristic Descriptor
		168	- GATT Reliable Writes
		168	- GATT Notifications
		169	- GATT Indications

Bluetooth Low Energy Protocol Stack
API Reference Manual: Basics

Publication Date: Rev.1.18 Aug 31, 2016

Published by: Renesas Electronics Corporation

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

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

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

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

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

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

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

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

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

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

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

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

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

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2016 Renesas Electronics Corporation. All rights reserved.
Colophon 4.0

Bluetooth Low Energy Protocol Stack

Renesas Electronics Corporation

R01UW0088EJ0118