RX Family

US159-DA14531EVZ BLE Control Module Using Firmware Integration Technology

Introduction

This application note describes the usage of the US159-DA14531EVZ BLE control module, which conforms to the Firmware Integration Technology (FIT) standard.

In the following pages, the US159-DA14531EVZ BLE control module software is referred to collectively as “the DA14531 BLE FIT module” or “the FIT module.”

The FIT module supports the following BLE module:

- DA14531MOD (US159-DA14531EVZ)

In the following pages, the DA14531MOD is referred to as “the BLE module”.

Target Devices

- RX65N Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family

Related Documents

- Firmware Integration Technology User’s Manual (R01AN1833)
- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
- RX Smart Configurator User’s Guide: e² studio (R20AN0451)
- RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
- RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)

Contents

1. Overview

1.1. DA14531 FIT Module

The FIT module is designed to be added to user projects as an API. For instruction on adding the FIT module, refer to 2.11 Adding the FIT Module to Your Project.

1.2. Overview of the DA14531 BLE FIT Module

The DA14531 is an ultra-low power SoC integrating a 2.4 GHz transceiver and an Arm® Cortex-M0+ microcontroller with a RAM of 48 kB and a One-Time Programmable (OTP) memory of 32 kB. It can be used as a standalone application processor or as a data pump in hosted systems.

The Bluetooth® LE firmware includes the L2CAP service layer protocols, Security Manager (SM), Attribute Protocol (ATT), the Generic Attribute Profile (GATT), and the Generic Access Profile (GAP). All profiles published by the Bluetooth® SIG as well as custom profiles are supported.

1.2.1. Connection with DA14531 BLE

Examples of connection to the DA14531 BLE are shown below.

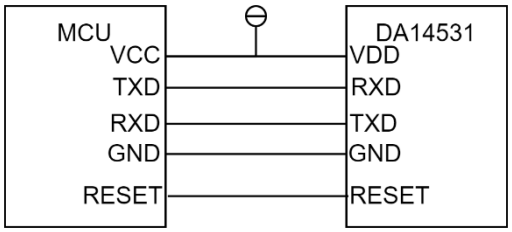


Figure 1-1 Example connection to the DA14531 module.

1.2.2. Software configuration

Figure 1-2 shows the software configuration.

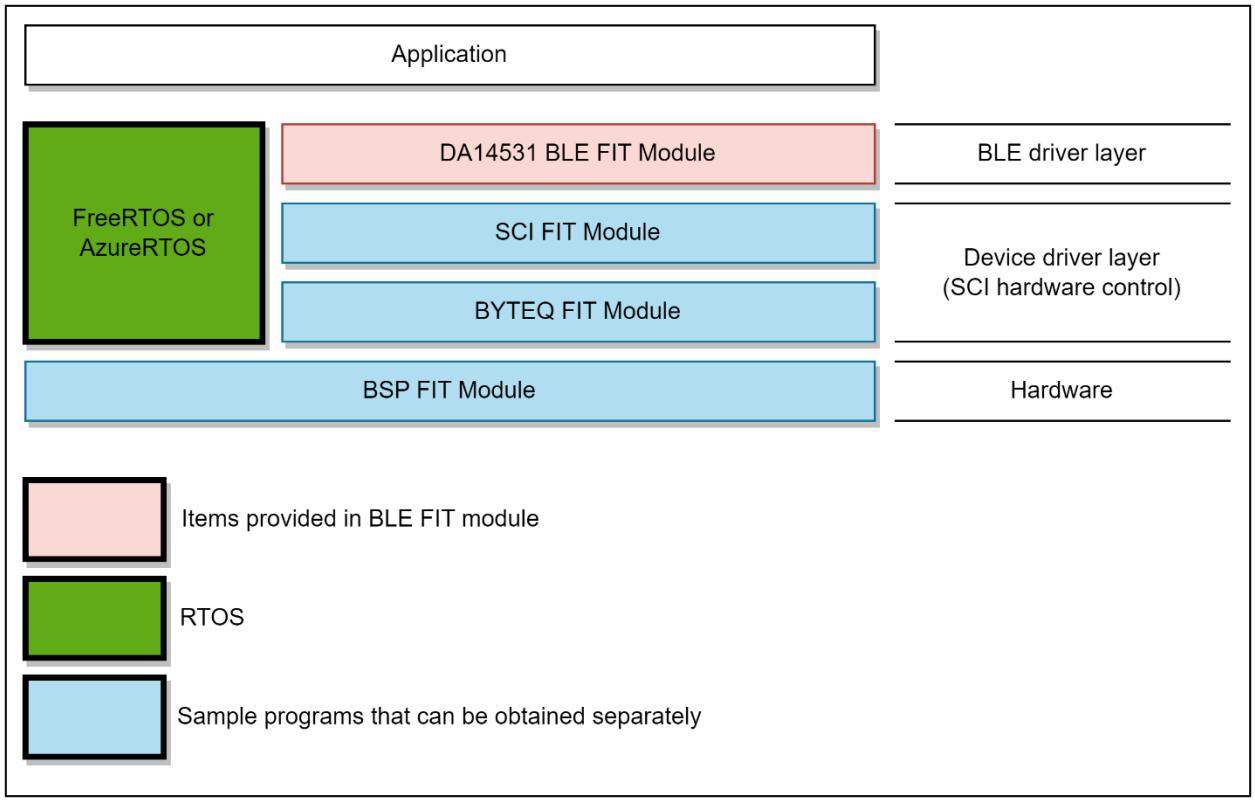


Figure 1-2 Software configuration diagram.

1. DA14531 BLE FIT module
   The FIT module. This software is used to control the BLE module.
2. SCI FIT module
   Implements communication between the BLE module and the MCU. A sample program is available. Refer to “Related Documents” on page 1 and obtain the software.
3. Peripheral function modules
   This software implements timer control and buffer management. Sample programs are available. Refer to “Related Documents” on page 1 and obtain the software.
4. RTOS
   When using the FIT module, you can choose to use FreeRTOS or AzureRTOS or Bare Metal by BSP_CFG_RTOS_USED.

1.3. Features

The Bluetooth Low Energy Abstraction module with GTL supports the following features:

- Common functionality
  - Open/Close the BLE protocol stack.
- The following GAP Role support
  - Peripheral: The device that accepts a connection request from Central and establishes a connection.
- GAP functionality
  - Initialize the Host stack.
  - Setting address.
  - Start/Stop Advertising.
  - Connect/Disconnect a link.
- GATT Common functionality
  - Get MTU Size.
- GATT Server functionality
  - Initialization of GATT Server.
  - Loading of Profile definition.
  - Notification of characteristics modification.
  - Read/Write of GATT Profile from host.

1.4. API Overview

Table 1-1 lists the API functions included in the FIT module. The required memory sizes are lists in 2.8 Code Size.

Table 1-1 API Functions

Function	Function Description
BLE Common Interface	
R_BLE_Open()	Open the BLE protocol stack.
R_BLE_Close()	Close the BLE protocol stack.
R_BLE_Execute()	Execute the BLE task.
R_BLE_IsTaskFree()	Check if the BLE task queue is free or not.
R_BLE_GetVersion()	Get the BLE FIT module version.
BLE GAP Interface	
R_BLE_GAP_Init()	Initialize the Host Stack.
R_BLE_GAP_Terminate()	Terminate the Host Stack.
R_BLE_GAP_UpdConn()	Update the connection parameters.
R_BLE_GAP_SetDataLen()	Update the packet size and the packet transmit time.
R_BLE_GAP_Disconnect()	Disconnect the link.
R_BLE_GAP_GetVerInfo()	Get the version number of the Controller and the host stack.
R_BLE_GAP_ReadRssi()	Get RSSI.
R_BLE_GAP_ReadChMap()	Get the Channel Map.
R_BLE_GAP_SetAdvParam()	Set advertising parameters.
R_BLE_GAP_SetAdvSresData()	Set advertising data/scan response data/periodic advertising data.
R_BLE_GAP_StartAdv()	Start advertising.
R_BLE_GAP_StopAdv()	Stop advertising.
R_BLE_GAP_GetRemainAdvBufSize()	Get buffer size for advertising data/scan response data/periodic advertising data in the Controller.
R_BLE_GAP_GetRemDevInfo()	Get the information about remote device.
BLE GATT Common Interface	
R_BLE_GATT_GetMtu()	Gets the current MTU used in GATT communication.
BLE GATT Server Interface	
R_BLE_GATTS_SetDbInst()	Sets GATT Database to host stack.
R_BLE_GATTS_RegisterCb()	Registers a callback for GATT Server event.
R_BLE_GATTS_DeregisterCb()	Deregisters the callback function for GATT Server event.
R_BLE_GATTS_Notification()	Sends a notification of an attribute's value.
R_BLE_GATTS_Indication()	Sends an indication of an attribute's value.
R_BLE_GATTS_GetAttr()	Gets an attribute value from the GATT Database.
R_BLE_GATTS_SetAttr()	Sets an attribute value to the GATT Database.

BLE GATT Client Interface	
R_BLE_GATTC_RegisterCb()	Registers a callback function for GATT Client event.
R_BLE_GATTC_DeregisterCb()	Deregisters the callback function for GATT Client event.
R_BLE_GATTC_ReqExMtu()	Sends a MTU Exchange Request PDU to a GATT Server in order to change the current MTU.
R_BLE_GATTC_DiscAllPrimServ()	Discovers all Primary Services in a GATT Server.
R_BLE_GATTC_DiscPrimServ()	Discovers Primary Service specified by p_uuid in a GATT Server.
R_BLE_GATTC_DiscIncServ()	Discovers Included Services within the specified attribute handle range in a GATT Server.
R_BLE_GATTC_DiscAllChar()	Discovers Characteristic within the specified attribute handle range in a GATT Server.
R_BLE_GATTC_DiscCharByUuid()	Discovers Characteristic specified by uuid within the specified attribute handle range in a GATT Server.
R_BLE_GATTC_DiscAllCharDesc()	Discovers Characteristic Descriptor within the specified attribute handle range in a GATT Server.
R_BLE_GATTC_ReadChar()	Reads a Characteristic/Characteristic Descriptor in a GATT Server.
R_BLE_GATTC_ReadCharUsingUuid()	Reads a Characteristic in a GATT Server using a specified UUID.
R_BLE_GATTC_ReadLongChar()	Reads a Long Characteristic in a GATT Server.
R_BLE_GATTC_ReadMultiChar()	Reads multiple Characteristics in a GATT Server.
R_BLE_GATTC_WriteCharWithoutRsp()	Writes a Characteristic in a GATT Server without response.
R_BLE_GATTC_SignedWriteChar()	Writes Signed Data to a Characteristic in a GATT Server without response.
R_BLE_GATTC_WriteChar()	Writes a Characteristic in a GATT Server.
R_BLE_GATTC_WriteLongChar()	Writes a Long Characteristic in a GATT Server.
R_BLE_GATTC_ReliableWrites()	Performs the Reliable Writes procedure described in GATT Specification.
R_BLE_GATTC_ExecWrite()	Executes a write to Characteristic.
BLE L2CAP Interface	
R_BLE_ L2CAP_RegisterCfPsm()	Registers PSM that uses L2CAP CBFC Channel and a callback for L2CAP event.
R_BLE_ L2CAP_DeregisterCfPsm()	Stops the use of the L2CAP CBFC Channel specified by the psm parameter and deregisters the callback function for L2CAP event.
R_BLE_ L2CAP_ReqCfConn()	Sends a connection request for L2CAP CBFC Channel.
R_BLE_ L2CAP_DisconnetCf()	Sends a disconnection request for L2CAP CBFC Channel.
R_BLE_ L2CAP_SendCfCredit()	Sends credit to a remote device.
R_BLE_ L2CAP_SendCfData()	Sends the data to a remote device via L2CAP CBFC Channel.

BLE Vendor Specific (VS) Interface	
R_BLE_VS_Init()	Initializes Vendor Specific API and registers a callback function for Vendor Specific Event.
R_BLE_VS_GetBdAddr()	Sets public/random address of local device to the area specified by the parameter.
R_BLE_VS_SetBdAddr()	Gets currently configured public/random address.
R_BLE_VS_GetRand()	Generates 4-16 bytes of random number used in creating keys.
Abstraction API for Renesas QE for BLE	
RM_BLE_ABS_Open()	Host stack is initialized with this function.
RM_BLE_ABS_Close()	Close the BLE channel.
RM_BLE_ABS_StartLegacyAdvertising()	Start Legacy Advertising after setting advertising parameters, advertising data and scan response data.

1.5. Status Transitions

Figure 1- 3 shows the status transitions of the FIT module up to communication status.

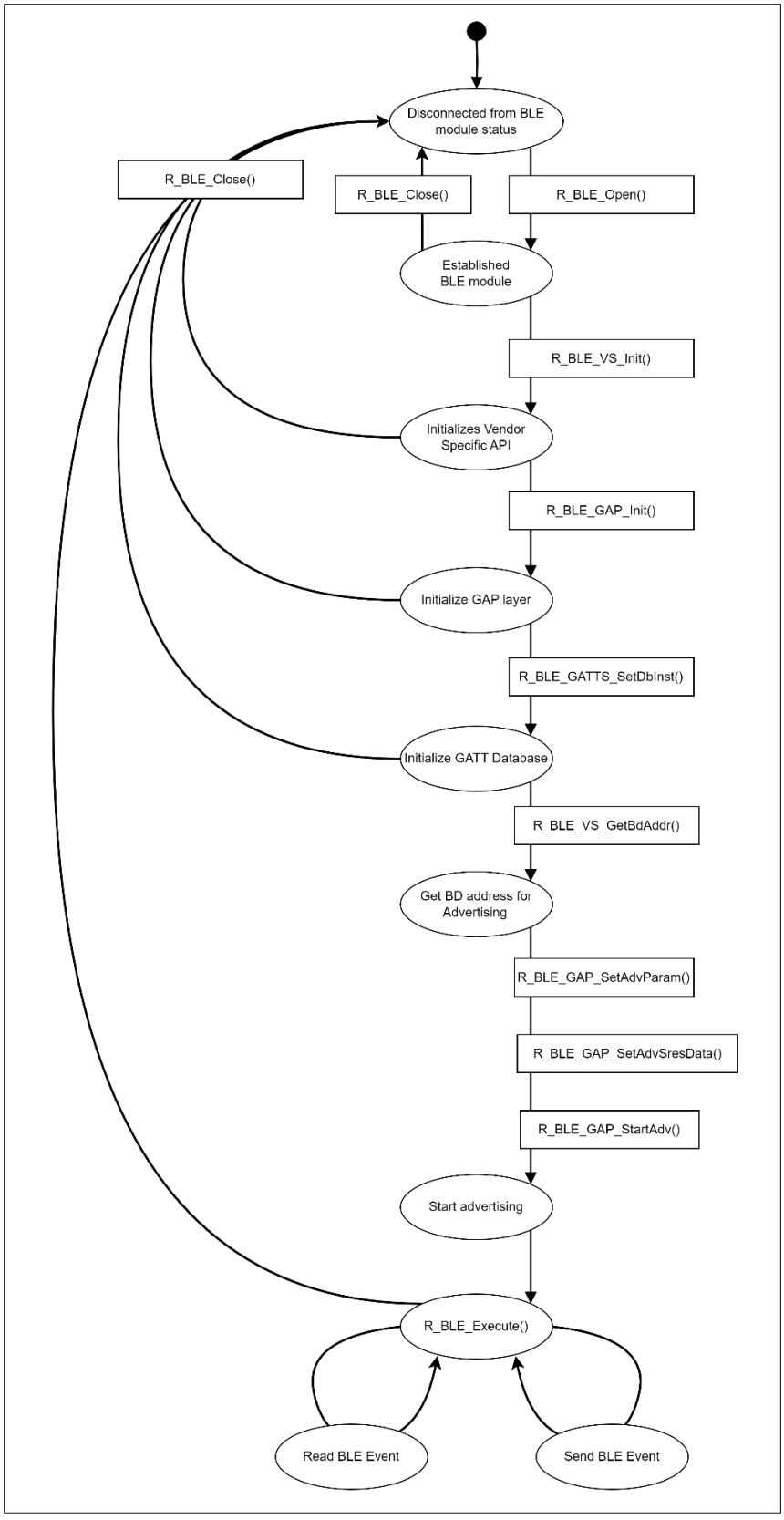


Figure 1- 3 Status transitions

1.6. Usage Notes

When using a public BD address the address pre-programmed into the DA14531 will be used and can't be overridden.

A random address can be set by calling the R_BLE_VS_SetBdAddr function before the R_BLE_GAP_Init function is called.

Ensure the BSP heap size is set to at least 2K bytes.

When using FreeRTOS ensure the heap 4 size is set to a minimum of 2K bytes.

This middleware module is compatible with GTL binary version 6.0.18 and later. You must ensure that the DA14531 Module (or PMOD) you are using contains this version (or later) firmware or that you use the boot from host feature and have the host MCU load the binary into the DA14531.

Instructions detailing how to upgrade the firmware in a DA14531 Module can be found here:

US159-DA14531EVZ Firmware Upgrade
The GTL binary file can be downloaded using the tool described in the above instructions, or by using the following link:

https://www.renesas.com/us/en/document/swo/fsp-gtl-binary-us159-da14531evz-pmod-programming?r=1564826

2. API Information

The FIT module has been confirmed to operate under the following conditions.

2.1. Hardware Requirements

The MCU used must support the following functions:

- Serial communication
- I/O ports

2.2. Software Requirements

The driver is dependent upon the following FIT module:

r_bsp
r_sci_rx
r_byteq_rx
FreeRTOS
AzureRTOS

2.3. Support Toolchain

The FIT module has been confirmed to work with the toolchain listed in 6.2 Confirmed Operation Environment.

2.4. Interrupt Vector

None

2.5. Header Files

All API calls and their supporting interface definitions are in r_ble_da14531_if.h.

2.6. Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.7. Compile Settings

The configuration option settings of the FIT module are contained in r_ble_da14531_config.h.
The names of the options and their setting values are listed in the table below.

Table 2-1 Configuration Options (r_ble_da14531_config.h)

Configuration Options in r_ble_da14531_config.h	
BLE_CFG_PARAM_CHECKING_ENABLE Note: The default is System Default	Parameter checking.
BLE_CFG_TRANSPORT_INTERFACE_UART Note: The default is 1	Use UART Transport Layer Interface
BLE_CFG_SCI_CHANNEL Note: The default is 6	SCI channel for DA14531 GTL command communication.
BLE_CFG_SCI_INTERRUPT_LEVEL Note: The default is 3	Interrupt Level for BLE_CFG_SCI_CHANNEL.
BLE_CFG_RESET_PORT Note: The default is 5	General-purpose port PDR register connected to the DA14531 reset port.
BLE_CFG_RESET_PIN Note: The default is 5	General-purpose port PODR register connected to the DA14531 reset pin.
BLE_CFG_SCK_PORT Note: The default is 0	General-purpose port PDR register connected to the DA14531 SCK port.
BLE_CFG_SCK_PIN Note: The default is 2	General-purpose port PODR register connected to the DA14531 SCK pin.
BLE_CFG_RESET_POLARITY Note: The default is 0	Reset Polarity.
BLE_CFG_HOST_BOOT_MODE Note: The default is 0.	Boot SDK download from host MCU. When using this feature via 1-Wire UART, please refer to 6.1 Limitations
BLE_CFG_ABS_NUMBER_BONDING Note: The default is 1	Configure ABS Number Bonding
BLE_CFG_ABS_TIMER_NUMBER_OF_SLOT Note: The default is 10	Configure ABS Timer number of slot
BLE_CFG_ABS_GATT_MTU_SIZE Note: The default is 247	Configure ABS GATT MTU size
BLE_CFG_ABS_RF_CONNECTION_MAXIMUM Note: the default is 1	Configure ABS RF connection maximum
BLE_CFG_RF_CONN_MAX Note: The default is 1	Configure RF connection maximum

Table 2-2 Configuration Options (r_sci_rx_config.h)

Configuration Options in r_ sci_rx_config.h	
#define SCI_CFG_CHx_INCLUDED Notes: 1. CHx = CH0 to CH12 2. The default values are as follows: CH0 CH2 to CH12: 0, CH1: 1	Each channel has resources such as transmit and receive buffers, counters, interrupts, other programs, and RAM. Setting this option to 1 assigns related resources to the specified channel.
#define SCI_CFG_CHx_TX_BUFSIZ Notes: 1. CHx = CH0 to CH12 2. The default value is 80 for all channels.	Specifies the transmit buffer size of an individual channel. The buffer size of the channel specified by BLE_CFG_SCI_CHANNEL should be set to 4096.
#define SCI_CFG_CHx_RX_BUFSIZ Notes: 1. CHx = CH0 to CH12 2. The default value is 80 for all channels.	Specifies the receive buffer size of an individual channel. The buffer size of the channel specified by BLE_CFG_SCI_CHANNEL should be set to 4096.
#define SCI_CFG_TEI_INCLUDED Note: The default is 0.	Enables the transmit end interrupt for serial transmissions. This option should be set to 1.

Table 2-3 Configuration Options (r_bsp_config.h)

Configuration Options in r_ bsp_config.h	
#define BSP_CFG_RTOS_USED Note: The default is 0.	Specifies the type of real-time OS. When using this FIT module, set the following. Baremetal: 0, FreeRTOS:1, AzureRTOS: 5

2.8. Code Size

Typical code sizes associated with this module are listed below.
The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7 Compile Settings. The table lists reference values when the C compiler's compile options are set to their default values, as described in 2.3 Support Toolchain. The compile option default values are optimization level: 2, optimization type: for size, and data endianness: little-endian. The code size varies depending on the C compiler version and compile options.

Device	RTOS	Category	Memory usage
			Renesas Compiler
RX65N	FreeRTOS	ROM	56478 bytes
		RAM	6174 bytes
	AzureRTOS	ROM	44583 bytes
		RAM	6174 bytes
	Baremetal	ROM	48372 bytes
		RAM	5922 bytes

*** Note:** ROM usage included 23KB (23956 bytes) of DA1453x Boot image

2.9. Return values

The error codes returned by API functions are listed below. The enumerated types of return values and API function declarations are contained in r_ble_api.h.

```
typedef uint16_t ble_status_t;

enum RBLE_STATUS_enum
{
    BLE_SUCCESS = 0x0000,

    /* common error code */
    BLE_ERR_INVALID_PTR         = 0x0001,
    BLE_ERR_INVALID_DATA        = 0x0002,
    BLE_ERR_INVALID_ARG         = 0x0003,
    BLE_ERR_INVALID_FUNC        = 0x0004,
    BLE_ERR_INVALID_CHAN        = 0x0005,
    BLE_ERR_INVALID_MODE        = 0x0006,
    BLE_ERR_UNSUPPORTED         = 0x0007,
    BLE_ERR_INVALID_STATE       = 0x0008,
    BLE_ERR_INVALID_OPERATION   = 0x0009,
    BLE_ERR_ALREADY_IN_PROGRESS = 0x000A,
    BLE_ERR_CONTEXT_FULL        = 0x000B,
    BLE_ERR_MEM_ALLOC_FAILED    = 0x000C,
    BLE_ERR_NOT_FOUND           = 0x000D,
    BLE_ERR_INVALID_HDL         = 0x000E,
    BLE_ERR_DISCONNECTED        = 0x000F,
    BLE_ERR_LIMIT_EXCEEDED      = 0x0010,
    BLE_ERR_RSP_TIMEOUT         = 0x0011,
    BLE_ERR_NOT_YET_READY       = 0x0012,
    BLE_ERR_UNSPECIFIED         = 0x0013,
    BLE_ERR_ALREADY_INITIALIZED = 0x0014,

    /* HCI Spec Error */
    BLE_ERR_HC_UNKNOWN_HCI_CMD              = 0x1001,
    BLE_ERR_HC_NO_CONN                      = 0x1002,
    BLE_ERR_HC_HW_FAIL                      = 0x1003,
    BLE_ERR_HC_PAGE_TO                      = 0x1004,
    BLE_ERR_HC_AUTH_FAIL                    = 0x1005,
    BLE_ERR_HC_KEY_MISSING                  = 0x1006,
    BLE_ERR_HC_MEM_FULL                     = 0x1007,
    BLE_ERR_HC_CONN_TO                      = 0x1008,
```

```
    BLE_ERR_HC_MAX_NUM_OF_CONN              = 0x1009,
    BLE_ERR_HC_MAX_NUM_OF_SCO_CONN          = 0x100A,
    BLE_ERR_HC_ACL_CONN_ALREADY_EXISTS      = 0x100B,
    BLE_ERR_HC_CMD_DISALLOWED               = 0x100C,
    BLE_ERR_HC_HOST_REJ_LIMITED_RESRC       = 0x100D,
    BLE_ERR_HC_HOST_REJ_SEC_REASONS         = 0x100E,
    BLE_ERR_HC_HOST_REJ_PERSONAL_DEV        = 0x100F,
    BLE_ERR_HC_HOST_TO                      = 0x1010,
    BLE_ERR_HC_UNSPRT_FEAT_OR_PARAM         = 0x1011,
    BLE_ERR_HC_INVALID_HCI_CMD_PARAM        = 0x1012,
    BLE_ERR_HC_OTHER_END_TERM_USER          = 0x1013,
    BLE_ERR_HC_OTHER_END_TERM_LOW_RESRC     = 0x1014,
    BLE_ERR_HC_OTHER_END_TERM_PW_OFF        = 0x1015,
    BLE_ERR_HC_CONN_TERM_BY_LOCAL_HOST      = 0x1016,
    BLE_ERR_HC_REPEATED_ATTEMPTS            = 0x1017,
    BLE_ERR_HC_PAIRING_NOT_ALLOWED          = 0x1018,
    BLE_ERR_HC_UNKNOWN_LMP_PDU              = 0x1019,
    BLE_ERR_HC_UNSPRT_REM_FEAT              = 0x101A,
    BLE_ERR_HC_SCO_OFFSET_REJ               = 0x101B,
    BLE_ERR_HC_SCO_INTERVAL_REJ             = 0x101C,
    BLE_ERR_HC_SCO_AIR_MODE_REJ             = 0x101D,
    BLE_ERR_HC_INVALID_LMP_PARAM            = 0x101E,
    BLE_ERR_HC_UNSPECIFIED_ERR              = 0x101F,
    BLE_ERR_HC_UNSPRT_LMP_PARAM_VAL         = 0x1020,
    BLE_ERR_HC_ROLE_CHANGE_NOT_ALLOWED      = 0x1021,
    BLE_ERR_HC_LMP_RSP_TO                   = 0x1022,
    BLE_ERR_HC_LMP_ERR_TX_COLLISION         = 0x1023,
    BLE_ERR_HC_LMP_PDU_NOT_ALLOWED          = 0x1024,
    BLE_ERR_HC_ENC_MODE_NOT_ACCEPTABLE      = 0x1025,
    BLE_ERR_HC_UNIT_KEY_USED                = 0x1026,
    BLE_ERR_HC_QOS_IS_NOT_SPRT              = 0x1027,
    BLE_ERR_HC_INSTANT_PASSED               = 0x1028,
    BLE_ERR_HC_PAIRING_UNIT_KEY_NOT_SPRT    = 0x1029,
    BLE_ERR_HC_DIFF_TRANSACTION_COLLISION   = 0x102A,
    BLE_ERR_HC_QOS_UNACCEPTABLE_PARAM       = 0x102C,
    BLE_ERR_HC_QOS_REJ                      = 0x102D,
    BLE_ERR_HC_CH_CLASSIFICATION_NOT_SPRT   = 0x102E,
    BLE_ERR_HC_INSUFFICIENT_SEC             = 0x102F,
    BLE_ERR_HC_PARAM_OUT_OF_MANDATORY_RANGE = 0x1030,
    BLE_ERR_HC_ROLE_SWITCH_PENDING          = 0x1032,
    BLE_ERR_HC_RESERVED_SLOT_VIOLATION      = 0x1034,
    BLE_ERR_HC_ROLE_SWITCH_FAIL             = 0x1035,
    BLE_ERR_HC_EXT_INQUIRY_RSP_TOO_LARGE    = 0x1036,
    BLE_ERR_HC_SSP_NOT_SPRT_BY_HOST         = 0x1037,
    BLE_ERR_HC_HOST_BUSY_PAIRING            = 0x1038,
    BLE_ERR_HC_CONN_REJ_NO_SUIT_CH_FOUND    = 0x1039,
    BLE_ERR_HC_CTRL_BUSY                    = 0x103A,
    BLE_ERR_HC_UNACCEPTEBALE_CONN_INTERVAL  = 0x103B,
    BLE_ERR_HC_ADV_TO                       = 0x103C,
    BLE_ERR_HC_CONN_TREM_DUE_TO_MIC_FAIL    = 0x103D,
    BLE_ERR_HC_CONN_FAIL_TO_BE_EST          = 0x103E,
    BLE_ERR_HC_MAC_CONN_FAIL                = 0x103F,
    BLE_ERR_HC_COARSE_CLK_ADJUST_REJ        = 0x1040,
    BLE_ERR_HC_TYPE0_SUBMAP_NOT_DEFINED     = 0x1041,
    BLE_ERR_HC_UNKNOWN_ADV_ID               = 0x1042,
    BLE_ERR_HC_LIMIT_REACHED                = 0x1043,
    BLE_ERR_HC_OP_CANCELLED_BY_HOST         = 0x1044,

    /* SMP Spec Error */
    BLE_ERR_SMP_LE_PASSKEY_ENTRY_FAIL        = 0x2001,
    BLE_ERR_SMP_LE_OOB_DATA_NOT_AVAILABLE    = 0x2002,
```

```
    BLE_ERR_SMP_LE_AUTH_REQ_NOT_MET          = 0x2003,
    BLE_ERR_SMP_LE_CONFIRM_VAL_NOT_MATCH     = 0x2004,
    BLE_ERR_SMP_LE_PAIRING_NOT_SPRT          = 0x2005,
    BLE_ERR_SMP_LE_INSUFFICIENT_ENC_KEY_SIZE = 0x2006,
    BLE_ERR_SMP_LE_CMD_NOT_SPRT              = 0x2007,
    BLE_ERR_SMP_LE_UNSPECIFIED_REASON        = 0x2008,
    BLE_ERR_SMP_LE_REPEATED_ATTEMPTS         = 0x2009,
    BLE_ERR_SMP_LE_INVALID_PARAM             = 0x200A,
    BLE_ERR_SMP_LE_DHKEY_CHECK_FAIL          = 0x200B,
    BLE_ERR_SMP_LE_NUM_COMP_FAIL             = 0x200C,
    BLE_ERR_SMP_LE_BREDR_PAIRING_IN_PROGRESS = 0x200D,
    BLE_ERR_SMP_LE_CT_KEY_GEN_NOT_ALLOWED    = 0x200E,
    BLE_ERR_SMP_LE_DISCONNECTED              = 0x200F,
    BLE_ERR_SMP_LE_TO                        = 0x2011,
    BLE_ERR_SMP_LE_LOC_KEY_MISSING           = 0x2014,

    /* GATT Spec Error */
    BLE_ERR_GATT_INVALID_HANDLE              = 0x3001,
    BLE_ERR_GATT_READ_NOT_PERMITTED          = 0x3002,
    BLE_ERR_GATT_WRITE_NOT_PERMITTED         = 0x3003,
    BLE_ERR_GATT_INVALID_PDU                 = 0x3004,
    BLE_ERR_GATT_INSUFFICIENT_AUTHENTICATION = 0x3005,
    BLE_ERR_GATT_REQUEST_NOT_SUPPORTED       = 0x3006,
    BLE_ERR_GATT_INVALID_OFFSET              = 0x3007,
    BLE_ERR_GATT_INSUFFICIENT_AUTHORIZATION  = 0x3008,
    BLE_ERR_GATT_PREPARE_WRITE_QUEUE_FULL    = 0x3009,
    BLE_ERR_GATT_ATTRIBUTE_NOT_FOUND         = 0x300A,
    BLE_ERR_GATT_ATTRIBUTE_NOT_LONG          = 0x300B,
    BLE_ERR_GATT_INSUFFICIENT_ENC_KEY_SIZE   = 0x300C,
    BLE_ERR_GATT_INVALID_ATTRIBUTE_LEN       = 0x300D,
    BLE_ERR_GATT_UNLIKELY_ERROR              = 0x300E,
    BLE_ERR_GATT_INSUFFICIENT_ENCRYPTION     = 0x300F,
    BLE_ERR_GATT_UNSUPPORTED_GROUP_TYPE      = 0x3010,
    BLE_ERR_GATT_INSUFFICIENT_RESOURCES      = 0x3011,

    /* defined in CSS */
    BLE_ERR_GATT_WRITE_REQ_REJECTED       = 0x30FC,
    BLE_ERR_GATT_CCCD_IMPROPERLY_CFG      = 0x30FD,
    BLE_ERR_GATT_PROC_ALREADY_IN_PROGRESS = 0x30FE,
    BLE_ERR_GATT_OUT_OF_RANGE             = 0x30FF,

    /* L2CAP Spec Error */
    BLE_ERR_L2CAP_PSM_NOT_SUPPORTED         = 0x4002,
    BLE_ERR_L2CAP_NO_RESOURCE               = 0x4004,
    BLE_ERR_L2CAP_INSUF_AUTHEN              = 0x4005,
    BLE_ERR_L2CAP_INSUF_AUTHOR              = 0x4006,
    BLE_ERR_L2CAP_INSUF_ENC_KEY_SIZE        = 0x4007,
    BLE_ERR_L2CAP_REFUSE_INSUF_ENC          = 0x4008,
    BLE_ERR_L2CAP_REFUSE_INVALID_SCID       = 0x4009,
    BLE_ERR_L2CAP_REFUSE_SCID_ALREADY_ALLOC = 0x400A,
    BLE_ERR_L2CAP_REFUSE_UNACCEPTABLE_PARAM = 0x400B,
};
```

2.10. Parameter

```
/* Application callback event types */
#define R_BLE_GTL_CB_EVT_TYPE_MASK              0xF000U
#define R_BLE_GTL_CB_EVT_TYPE_GAP               0x1000U
#define R_BLE_GTL_CB_EVT_TYPE_GATTS             0x3000U
#define R_BLE_GTL_CB_EVT_TYPE_GATTC             0x4000U
#define R_BLE_GTL_CB_EVT_TYPE_L2CAP             0x5000U
#define R_BLE_GTL_CB_EVT_TYPE_VS                0x8000U

/* GTL Task ID's */
#define R_BLE_GTL_TASK_ID_GATTM                 0x000B
#define R_BLE_GTL_TASK_ID_GATTC                 0x000C
#define R_BLE_GTL_TASK_ID_GAPM                  0x000D
#define R_BLE_GTL_TASK_ID_GAPC                  0x000E
#define R_BLE_GTL_TASK_ID_GTL                   0x0010

/* GTL GATTM Command ID's */
#define R_BLE_GTL_GATTM_ADD_SVC_REQ             0x0B00
#define R_BLE_GTL_GATTM_ADD_SVC_RSP             0x0B01
#define R_BLE_GTL_GATTM_ATT_GET_VALUE_REQ       0x0B0A
#define R_BLE_GTL_GATTM_ATT_GET_VALUE_RSP       0x0B0B
#define R_BLE_GTL_GATTM_ATT_SET_VALUE_REQ       0x0B0C
#define R_BLE_GTL_GATTM_ATT_SET_VALUE_RSP       0x0B0D

/* GTL GATTC Command ID's */
#define R_BLE_GTL_GATTC_CMP_EVT                 0x0C00
#define R_BLE_GTL_GATTC_EXC_MTU_CMD             0x0C01
#define R_BLE_GTL_GATTC_MTU_CHANGED_IND         0x0C02
#define R_BLE_GTL_GATTC_DISC_CMD                0x0C03
#define R_BLE_GTL_GATTC_DISC_SVC_IND            0x0C04
#define R_BLE_GTL_GATTC_DISC_CHAR_IND           0x0C06
#define R_BLE_GTL_GATTC_DISC_CHAR_DESC_IND      0x0C07
#define R_BLE_GTL_GATTC_READ_CMD                0x0C08
#define R_BLE_GTL_GATTC_READ_IND                0x0C09
#define R_BLE_GTL_GATTC_SEND_EVT_CMD            0x0C10
#define R_BLE_GTL_GATTC_WRITE_CMD               0x0C0A
#define R_BLE_GTL_GATTC_WRITE_EXECUTE_CMD       0x0C0B
#define R_BLE_GTL_GATTC_READ_REQ_IND            0x0C13
#define R_BLE_GTL_GATTC_READ_CFM                0x0C14
#define R_BLE_GTL_GATTC_WRITE_REQ_IND           0x0C15
#define R_BLE_GTL_GATTC_WRITE_CFM               0x0C16

/* GTL GAPM Command ID's */
#define R_BLE_GTL_GAPM_CMP_EVT                  0x0D00
#define R_BLE_GTL_GAPM_DEVICE_READY_IND         0x0D01
#define R_BLE_GTL_GAPM_RESET_CMD                0x0D02
#define R_BLE_GTL_GAPM_CANCEL_CMD               0x0D03
#define R_BLE_GTL_GAPM_SET_DEV_CONFIG_CMD       0x0D04
#define R_BLE_GTL_GAPM_GET_DEV_INFO_CMD         0x0D06
#define R_BLE_GTL_GAPM_DEV_VERSION_IND          0x0D07
#define R_BLE_GTL_GAPM_DEV_BDADDR_IND           0x0D08
#define R_BLE_GTL_GAPM_GEN_RAND_ADDR_CMD        0x0D16
#define R_BLE_GTL_GAPM_GEN_RAND_NB_CMD          0x0D19
#define R_BLE_GTL_GAPM_GEN_RAND_NB_IND          0x0D1A
#define R_BLE_GTL_GAPM_UNKNOWN_TASK_IND         0x0D1D
#define R_BLE_GTL_GAPM_START_ADVERTISE_CMD      0x0D0D

/* GTL GAPC Command ID's */
#define R_BLE_GTL_GAPC_CMP_EVT                  0x0E00
```

```
#define R_BLE_GTL_GAPC_CONNECTION_REQ_IND       0x0E01
#define R_BLE_GTL_GAPC_CONNECTION_CFM           0x0E02
#define R_BLE_GTL_GAPC_DISCONNECT_IND           0x0E03
#define R_BLE_GTL_GAPC_DISCONNECT_CMD           0x0E04
#define R_BLE_GTL_GAPC_GET_INFO_CMD             0x0E05
#define R_BLE_GTL_GAPC_PEER_VERSION_IND         0x0E07
#define R_BLE_GTL_GAPC_PEER_FEATURES_IND        0x0E08
#define R_BLE_GTL_GAPC_CON_RSSI_IND             0x0E09
#define R_BLE_GTL_GAPC_GET_DEV_INFO_REQ_IND     0x0E0A
#define R_BLE_GTL_GAPC_GET_DEV_INFO_CFM         0x0E0B
#define R_BLE_GTL_GAPC_PARAM_UPDATE_CMD         0x0E0E
#define R_BLE_GTL_GAPC_PARAM_UPDATE_REQ_IND     0x0E0F
#define R_BLE_GTL_GAPC_PARAM_UPDATE_CFM         0x0E10
#define R_BLE_GTL_GAPC_PARAM_UPDATED_IND        0x0E11
#define R_BLE_GTL_GAPC_CON_CHANNEL_MAP_IND      0x0E1D
#define R_BLE_GTL_GAPC_LECB_CONNECT_CMD         0x0E20
#define R_BLE_GTL_GAPC_LECB_ADD_CMD             0x0E24
#define R_BLE_GTL_GAPC_LECB_SEND_CMD            0x0E25
#define R_BLE_GTL_GAPC_LECB_DISCONNECT_CMD      0x0E26
#define R_BLE_GTL_GAPC_SET_LE_PKT_SIZE_CMD      0x0E2B
#define R_BLE_GTL_GAPC_LE_PKT_SIZE_IND          0x0E2C

/* Attribute permissions defined in QE profile */
#define R_BLE_GTL_QE_ATT_PERM_READ              0x01
#define R_BLE_GTL_QE_ATT_PERM_WRITE             0x02
#define R_BLE_GTL_QE_ATT_PERM_NOTIFY            0x10
#define R_BLE_GTL_QE_ATT_PERM_INDICATE          0x20

/* Attribute permissions defined in GTL message(s) */
#define R_BLE_GTL_ATT_PERM_READ_ENABLE          0x00000001UL
#define R_BLE_GTL_ATT_PERM_WRITE_ENABLE         0x00000008UL
#define R_BLE_GTL_ATT_PERM_INDICATE_ENABLE      0x00000040UL
#define R_BLE_GTL_ATT_PERM_NOIFY_ENABLE         0x00000200UL
#define R_BLE_GTL_ATT_PERM_WRITE_REQ_ACCEPTED   0x00020000UL
#define R_BLE_GTL_ATT_PERM_UUID_LEN_128         0x00080000UL

#define R_BLE_GTL_SVC_GAP_UUID                  0x1800
#define R_BLE_GTL_SVC_GATT_UUID                 0x1801
#define R_BLE_GTL_ATT_PRIMARY_SVC_DECL          0x2800
#define R_BLE_GTL_ATT_SECONDARY_SVC_DECL        0x2801
#define R_BLE_GTL_CHAR_DECLARATION              0x2803
#define R_BLE_GTL_CHAR_USER_DESC                0x2901
#define R_BLE_GTL_CHAR_DEVICE_NAME              0x2A00
#define R_BLE_GTL_CHAR_APPEARANCE               0x2A01

/* The first two bits of a non-public (random) address must be binary ones */
#define R_BLE_GTL_PUBLIC_BD_ADDR_MASK           0xC0

#define R_BLE_GTL_MS_PER_SECOND                 1000UL
#define R_BLE_GTL_ADV_TIMER_TICKS_PER_SECOND    100UL

/* Service permissions defined in GTL messages(s), can be or'd together */
#define R_BLE_GTL_SVC_PERM_ENABLE               0x04
#define R_BLE_GTL_SVC_PERM_UUID_LEN_128         0x40
#define R_BLE_GTL_SVC_PERM_PRIMARY              0x80

/* "RBLE" in ASCII. Used to determine if the control block is open. */
#define R_BLE_GTL_OPEN                          0x52424C45U

/* UART boot protocol message types */
#define R_BLE_GTL_BOOT_STX                      0x02
```

```
#define R_BLE_GTL_BOOT_SOH                      0x01
#define R_BLE_GTL_BOOT_ACK                      0x06
#define R_BLE_GTL_BOOT_NACK                     0x15

typedef enum e_r_ble_gtl_gapm_operation
{
    R_BLE_GTL_GAPM_OP_NONE = 0x00,
    R_BLE_GTL_GAPM_OP_RESET,
    R_BLE_GTL_GAPM_OP_CANCEL,
    R_BLE_GTL_GAPM_OP_SET_DEV_CONFIG,
    R_BLE_GTL_GAPM_OP_SET_CHANNEL_MAP,
    R_BLE_GTL_GAPM_OP_GET_DEV_VERSION,
    R_BLE_GTL_GAPM_OP_GET_DEV_BDADDR,
    R_BLE_GTL_GAPM_OP_GET_DEV_ADV_TX_POWER,
    R_BLE_GTL_GAPM_OP_GET_WLIST_SIZE,
    R_BLE_GTL_GAPM_OP_ADD_DEV_IN_WLIST,
    R_BLE_GTL_GAPM_OP_RMV_DEV_FRM_WLIST,
    R_BLE_GTL_GAPM_OP_CLEAR_WLIST,
    R_BLE_GTL_GAPM_OP_ADV_NON_CONN,
    R_BLE_GTL_GAPM_OP_ADV_UNDIRECT,
    R_BLE_GTL_GAPM_OP_ADV_DIRECT,
    R_BLE_GTL_GAPM_OP_ADV_DIRECT_LDC,
    R_BLE_GTL_GAPM_OP_UPDATE_ADVERTISE_DATA,
    R_BLE_GTL_GAPM_OP_SCAN_ACTIVE,
    R_BLE_GTL_GAPM_OP_SCAN_PASSIVE,
    R_BLE_GTL_GAPM_OP_CONNECTION_DIRECT,
    R_BLE_GTL_GAPM_OP_CONNECTION_AUTO,
    R_BLE_GTL_GAPM_OP_CONNECTION_SELECTIVE,
    R_BLE_GTL_GAPM_OP_CONNECTION_NAME_REQUEST,
    R_BLE_GTL_GAPM_OP_RESOLV_ADDR,
    R_BLE_GTL_GAPM_OP_GEN_RAND_ADDR,
    R_BLE_GTL_GAPM_OP_USE_ENC_BLOCK,
    R_BLE_GTL_GAPM_OP_GEN_RAND_NB,
    R_BLE_GTL_GAPM_OP_PROFILE_TASK_ADD,
    R_BLE_GTL_GAPM_OP_DBG_GET_MEM_INFO,
    R_BLE_GTL_GAPM_OP_PLF_RESET,
    R_BLE_GTL_GAPM_OP_SET_SUGGESTED_DFLT_LE_DATA_LEN,
    R_BLE_GTL_GAPM_OP_GET_SUGGESTED_DFLT_LE_DATA_LEN,
    R_BLE_GTL_GAPM_OP_GET_MAX_LE_DATA_LEN,
    R_BLE_GTL_GAPM_OP_GET_RAL_SIZE,
    R_BLE_GTL_GAPM_OP_GET_RAL_LOC_ADDR,
    R_BLE_GTL_GAPM_OP_GET_RAL_PEER_ADDR,
    R_BLE_GTL_GAPM_OP_ADD_DEV_IN_RAL,
    R_BLE_GTL_GAPM_OP_RMV_DEV_FRM_RAL,
    R_BLE_GTL_GAPM_OP_CLEAR_RAL,
    R_BLE_GTL_GAPM_OP_USE_P256_BLOCK,
    R_BLE_GTL_GAPM_OP_NETWORK_MODE_RAL,
    R_BLE_GTL_GAPM_OP_DEVICE_MODE_RAL,
    R_BLE_GTL_GAPM_OP_KEY_RENEW,
    R_BLE_GTL_GAPM_OP_GEN_P256_KEY = R_BLE_GTL_GAPM_OP_KEY_RENEW,
    R_BLE_GTL_GAPM_OP_LAST
} r_ble_gtl_gapm_operation_t;

typedef enum e_r_ble_gtl_gapc_operation
{
    R_BLE_GTL_GAPC_OP_NONE = 0x00,
    R_BLE_GTL_GAPC_OP_DISCONNECT,
    R_BLE_GTL_GAPC_OP_GET_PEER_NAME,
    R_BLE_GTL_GAPC_OP_GET_PEER_VERSION,
    R_BLE_GTL_GAPC_OP_GET_PEER_FEATURES,
    R_BLE_GTL_GAPC_OP_GET_PEER_APPEARANCE,
```

```
    R_BLE_GTL_GAPC_OP_GET_PEER_SLV_PREF_PARAMS,
    R_BLE_GTL_GAPC_OP_GET_CON_RSSI,
    R_BLE_GTL_GAPC_OP_GET_CON_CHANNEL_MAP,
    R_BLE_GTL_GAPC_OP_UPDATE_PARAMS,
    R_BLE_GTL_GAPC_OP_BOND,
    R_BLE_GTL_GAPC_OP_ENCRYPT,
    R_BLE_GTL_GAPC_OP_SECURITY_REQ,
    R_BLE_GTL_GAPC_OP_LE_CB_CREATE,
    R_BLE_GTL_GAPC_OP_LE_CB_DESTROY,
    R_BLE_GTL_GAPC_OP_LE_CB_CONNECTION,
    R_BLE_GTL_GAPC_OP_LE_CB_DISCONNECTION,
    R_BLE_GTL_GAPC_OP_LE_CB_ADDITION,
    R_BLE_GTL_GAPC_OP_GET_LE_PING_TO,
    R_BLE_GTL_GAPC_OP_SET_LE_PING_TO,
    R_BLE_GTL_GAPC_OP_SET_LE_PKT_SIZE,
    R_BLE_GTL_GAPC_OP_GET_PEER_CENTRAL_RPA,
    R_BLE_GTL_GAPC_OP_GET_PEER_RPA_ONLY,
    R_BLE_GTL_GAPC_OP_LE_CB_SEND,
} r_ble_gtl_gapc_operation_t;

typedef enum e_r_ble_gtl_gattc_operation
{
    R_BLE_GTL_GATTC_OP_NONE = 0x00,
    R_BLE_GTL_GATTC_OP_MTU_EXCH,
    R_BLE_GTL_GATTC_OP_DISC_ALL_SVC,
    R_BLE_GTL_GATTC_OP_DISC_BY_UUID_SVC,
    R_BLE_GTL_GATTC_OP_DISC_INCLUDED_SVC,
    R_BLE_GTL_GATTC_OP_DISC_ALL_CHAR,
    R_BLE_GTL_GATTC_OP_DISC_BY_UUID_CHAR,
    R_BLE_GTL_GATTC_OP_DISC_DESC_CHAR,
    R_BLE_GTL_GATTC_OP_READ,
    R_BLE_GTL_GATTC_OP_READ_LONG,
    R_BLE_GTL_GATTC_OP_READ_BY_UUID,
    R_BLE_GTL_GATTC_OP_READ_MULTIPLE,
    R_BLE_GTL_GATTC_OP_WRITE,
    R_BLE_GTL_GATTC_OP_WRITE_NO_RESPONSE,
    R_BLE_GTL_GATTC_OP_WRITE_SIGNED,
    R_BLE_GTL_GATTC_OP_EXEC_WRITE,
    R_BLE_GTL_GATTC_OP_REGISTER,
    R_BLE_GTL_GATTC_OP_UNREGISTER,
    R_BLE_GTL_GATTC_OP_NOTIFY,
    R_BLE_GTL_GATTC_OP_INDICATE,
} r_ble_gtl_gattc_operation_t;
```

2.11. Adding the FIT Module to Your Project

The FIT module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) for RX devices that are not supported by the Smart Configurator.

1) Adding the FIT module to your project using the Smart Configurator in e2 studio. By using the Smart Configurator in e2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e2 studio (R20AN0451)” for details.

2) Adding the FIT module to your project using the FIT Configurator in e2 studio. By using the FIT Configurator in e2 studio, the FIT module is automatically added to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

3. API Functions

3.1. R_BLE_Open()

Open the BLE protocol stack.

Format

```
ble_status_t R_BLE_Open (
      void
)
```

Parameters

None

Return values

BLE_SUCCESS	Success

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function should be called once before using the BLE protocol stack.

Reentrant

No

Example

```
R_BLE_Open();
```

Special Notes:

None.

3.2. R_BLE_Close()

Close the BLE protocol stack.

Format

```
ble_status_t R_BLE_Close (
      void
)
```

Parameters

None

Return values

BLE_SUCCESS	Success

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function should be called once to close the BLE protocol stack.

Reentrant

No

Example

```
R_BLE_Close();
```

Special Notes:

None

3.3. R_BLE_Execute()

Execute the BLE task.

Format

```
ble_status_t R_BLE_Execute (
      void
)
```

Parameters

None

Return values

BLE_SUCCESS	Success

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This handles all the task queued in the BLE protocol stack internal task queue and return. This function should be called repeatedly in the main loop.

Reentrant

No

Example

```
R_BLE_Open();
while (1)
{
    R_BLE_Execute();
}
```

Special Notes:

None

3.4. R_BLE_IsTaskFree()

Check if the BLE task queue is free or not.

Format

```
uint32_t R_BLE_IsTaskFree(
      void
)
```

Parameters

None

Return values

0x0	BLE task queue is not free.
0x1	BLE task queue is free.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function returns the BLE task queue free status.

When this function returns 0x0, call R_BLE_Execute() to execute the BLE task.

Example

```
R_BLE_Open();
while (1)
{
    R_BLE_Execute();
    if(0 != R_BLE_IsTaskFree())
    {
        xEventGroupWaitBits();
    }
}
```

Special Notes:

None

3.5. R_BLE_GetVersion()

Get the BLE FIT module version.

Format

```
uint32_t R_BLE_GetVersion(
      void
)
```

Parameters

None

Return values

Version number

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function returns the BLE FIT module version.

The major version(BLE_VERSION_MAJOR) is contained in the two most significant bytes, and the minor version(BLE_VERSION_MINOR) occupies the remaining two bytes.

Example

```
uint32_t version;

version = R_BLE_GetVersion();
```

Special Notes:

None

3.6. R_BLE_GAP_Init()

Initialize the Host Stack.

Format

```
ble_status_t R_BLE_GAP_Init (
      ble_gap_app_cb_t gap_cb
)
```

Parameters

gap_cb	A callback function registered with this function.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	gap_cb is specified as NULL.
BLE_ERR_INVALID_STATE(0x0008)	The reason for this error is as follows: - Host Stack was already initialized. - The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

Host stack is initialized with this function. Before using All the R_BLE APIs, it's necessary to call this function. A callback function is registered with this function. In order to receive the GAP event, it's necessary to register a callback function.

The result of this API call is notified in BLE_GAP_EVENT_STACK_ON event.

Reentrant

No

Example

None

Special Notes:

None

3.7. R_BLE_GAP_Terminate()

Terminate the Host Stack.

Format

```
ble_status_t R_BLE_GAP_Terminate(
      void
)
```

Parameters

None

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	Host stack hasn't been initialized.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The host stack is terminated with this function.

In order to reset all the Bluetooth functions, it's necessary to call this function.

The result of this API call is notified in BLE_GAP_EVENT_STACK_OFF event.

Reentrant

No

Example

None

Special Notes:

None

3.8. R_BLE_GAP_UpdConn()

Update the connection parameters.

Format

```
ble_status_t R_BLE_GAP_UpdConn(
      uint16_t                      conn_hdl,
      uint8_t                       mode,
      uint16_t                      accept,
      st_ble_gap_conn_param_t *     p_conn_updt_param
)
```

Parameters

conn_hdl — Connection handle identifying the link to be updated.

mode — Connection parameter update request or response.

macro	description
BLE_GAP_CONN_UPD_MODE_REQ (0x01)	Request for updating the connection parameters.
BLE_GAP_CONN_UPD_MODE_RSP (0x02)	Reply a connection parameter update request.

accept — When mode is BLE_GAP_CONN_UPD_MODE_RSP, accept or reject the connection parameters update request. If mode is BLE_GAP_CONN_UPD_MODE_REQ, accept is ignored.

macro	description
BLE_GAP_CONN_UPD_ACCEPT (0x0000)	Accept the update request.
BLE_GAP_CONN_UPD_REJECT (0x0001)	Reject the update request.

p_conn_updt_param — Connection parameters to be updated. When mode is BLE_GAP_CONN_UPD_MODE_RSP and accept is BLE_GAP_CONN_UPD_REJECT, p_conn_updt_param is ignored.

Return values

BLE_SUCCESS(0x0000) — Success

BLE_ERR_INVALID_PTR(0x0001) — When accept is BLE_GAP_CONN_UPD_ACCEPT, p_conn_updt_param is specified as NULL.

BLE_ERR_INVALID_ARG(0x0003) — The following is out of range.

- mode
- accept
- conn_intv_min field in p_conn_updt_param
- conn_intv_max field in p_conn_updt_param
- conn_latency in p_conn_updt_param
- sup_to in p_conn_updt_param
- conn_hdl

BLE_ERR_INVALID_STATE(0x0008) — Not connected with the remote device.

BLE_ERR_CONTEXT_FULL(0x000B) — Sending a L2CAP command, an error occurred.

BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function updates the connection parameters or replies to a request for updating connection parameters notified by BLE_GAP_EVENT_CONN_PARAM_UPD_REQ event. When the connection parameters have been updated, BLE_GAP_EVENT_CONN_PARAM_UPD_COMP event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.9. R_BLE_GAP_SetDataLen()

Update the packet size and the packet transmit time.

Format

```
ble_status_t R_BLE_GAP_SetDataLen(
      uint16_t    conn_hdl,
      uint16_t    tx_octets,
      uint16_t    tx_time
)
```

Parameters

conn_hdl	Connection handle identifying the link whose the transmission packet size or the transmission time to be changed.
tx_octets	Maximum transmission packet size. Valid range is 0x001B - 0x00FB.
tx_time	Maximum transmission time(us). Valid range is 0x0148 - 0x4290.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function requests for changing the maximum transmission packet size and the maximum packet transmission time. When Controller has received the request from host stack, BLE_GAP_EVENT_SET_DATA_LEN_COMP event is notified to the application layer. When the transmission packet size or the transmission time has been changed, BLE_GAP_EVENT_DATA_LEN_CHG event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.10. R_BLE_GAP_Disconnect()

Disconnect the link.

Format

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

Parameters

conn_hdl	Connection handle identifying the link to be disconnected.
reason	The reason for disconnection. Usually, set 0x13 which indicates that a user disconnects the link. If setting other than 0x13, refer the error code described in Core Specification Vol.2 Part D ,"2 Error Code Descriptions".

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	conn_hdl is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function disconnects a link. When the link has disconnected, BLE_GAP_EVENT_DISCONN_IND event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.11. R_BLE_GAP_GetVerInfo()

Get the version number of the Controller and the host stack.

Format

```
ble_status_t R_BLE_GAP_GetVerInfo   (
      void
)
```

Parameters

None

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function retrieves the version information of local device. The result of this API call is notified in BLE_GAP_EVENT_LOC_VER_INFO event.

Reentrant

No

Example

None

Special Notes:

None

3.12. R_BLE_GAP_ReadRssi()

Get RSSI.

Format

```
ble_status_t R_BLE_GAP_ReadRssi     (
      uint16_t    conn_hdl
)
```

Parameters

conn_hdl Connection handle identifying the link whose RSSI to be retrieved.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	conn_hdl is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function retrieves RSSI. The result of this API call is notified in BLE_GAP_EVENT_RSSI_RD_COMP event.

Reentrant

No

Example

None

Special Notes:

None

3.13. R_BLE_GAP_ReadChMap()

Get the Channel Map.

Format

```
ble_status_t R_BLE_GAP_ReadChMap    (
      uint16_t    conn_hdl
)
```

Parameters

conn_hdl Connection handle identifying the link whose channel map to be retrieved.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	conn_hdl is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function retrieves the channel map. The result of this API call is notified in BLE_GAP_EVENT_CH_MAP_RD_COMP event.

Reentrant

No

Example

None

Special Notes:

None

3.14. R_BLE_GAP_SetAdvParam()

Set advertising parameters.

Format

```
ble_status_t R_BLE_GAP_SetAdvParam (
      st_ble_gap_adv_param_t * p_adv_param
)
```

Parameters

p_adv_param Advertising parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_adv_param is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The below p_adv_param field value is out of range. • adv_handle • adv_intv_min/adv_intv_max • adv_ch_map • o_addr_type • p_addr_type • adv_phy • sec_adv_phy • scan_req_ntf_flag
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function sets advertising parameters. It's possible to do advertising where the advertising parameters are different every each advertising set. The number of advertising set in the Controller is defined as BLE_MAX_NO_OF_ADV_SETS_SUPPORTED. Each advertising set is identified with advertising handle (0x00-0x03). Create an advertising set with this function before start advertising, setting periodic advertising parameters, start periodic advertising, setting advertising data/scan response data/periodic advertising data. The result of this API call is notified in BLE_GAP_EVENT_ADV_PARAM_SET_COMP event.

Reentrant

No

Example

None

Special Notes:

None

3.15. R_BLE_GAP_SetAdvSresData()

Set advertising data/scan response data/periodic advertising data.

Format

```
ble_status_t R_BLE_GAP_SetAdvSresData     (
      st_ble_gap_adv_data_t *       p_adv_srsp_data
)
```

Parameters

p_adv_srsp_data Advertising data/scan response data/periodic advertising data.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The reason for this error is as follows: • p_adv_srsp_data is specified as NULL. • data_length field in p_adv_srsp_data parameter is not 0 and p_data field is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The following field in p_adv_srsp_data parameter is out of range. • adv_hdl • data_type • data_length • zero_length_flag
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function sets advertising data/scan response data/periodic advertising data to the advertising set. It is necessary to create an advertising set by R_BLE_GAP_SetAdvParam(), before calling this function. Set advertising data/scan response data/periodic advertising data, after allocating the memory for the data. The following shall be applied regarding the adv_prop_type field and the data_type field in st_ble_gap_adv_param_t parameter specified in R_BLE_GAP_SetAdvParam().

Reentrant

No

Example

None

Special Notes:

None

3.16. R_BLE_GAP_StartAdv()

Start advertising.

Format

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

Parameters

adv_hdl	The advertising handle pointing to the advertising set which starts advertising. The valid range is 0x00 - 0x03.
duration	The duration for which the advertising set identified by adv_hdl is enabled. Time = duration * 10ms. When the duration expires, BLE_GAP_EVENT_ADV_OFF event notifies that advertising is stopped. The valid range is 0x0000 - 0xFFFF. The duration parameter is ignored when the value is set to 0x0000.
max_extd_adv_evts	The maximum number of advertising events that be sent during advertising. When all the advertising events(max_extd_adv_evts) have been sent, BLE_GAP_EVENT_ADV_OFF event notifies that advertising is stopped. The max_extd_adv_evts parameter is ignored when the value is set to 0x00.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	adv_hdl is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function starts advertising. Create the advertising set specified with adv_hdl by R_BLE_GAP_SetAdvParam(), before calling this function. The result of this API call is notified in BLE_GAP_EVENT_ADV_ON event.

Reentrant

No

Example

None

Special Notes:

None

3.17. R_BLE_GAP_StopAdv()

Stop advertising.

Format

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

Parameters

adv_hdl The advertising handle pointing to the advertising set which stops advertising. The valid range is 0x00 - 0x03.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	adv_hdl is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function stops advertising. The result of this API call is notified in BLE_GAP_EVENT_ADV_OFF event.

Reentrant

No

Example

None

Special Notes:

None

3.18. R_BLE_GAP_GetRemainAdvBufSize()

Get buffer size for advertising data/scan response data/periodic advertising data in the Controller.

Format

```
ble_status_t R_BLE_GAP_GetRemainAdvBufSize      (
      uint16_t *  p_remain_adv_data_size,
      uint16_t *  p_remain_perd_adv_data_size
)
```

Parameters

p_remain_adv_data_size	The free buffer size of Controller to which advertising data/scan response data can be currently set.
p_remain_perd_adv_data_size	The free buffer size of Controller to which periodic advertising data can be currently set.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_remain_adv_data_size or p_remain_perd_adv_data_size is specified as NULL.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function gets the total size of advertising data/scan response data/periodic advertising data which can be currently set to Controller(all of the advertising sets). The application layer gets the data sizes via the parameters. By this API function call, no events occur.

Reentrant

No

Example

None

Special Notes:

None

3.19. R_BLE_GAP_GetRemDevInfo()

Get the information about remote device.

Format

```
ble_status_t R_BLE_GAP_GetRemDevInfo      (
      uint16_t    conn_hdl
)
```

Parameters

conn_hdl Connection handle identifying the remote device whose information to be retrieved.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

This function retrieves information about the remote device. The information includes BD_ADDR, the version number and LE features. The result of this API call is notified in BLE_GAP_EVENT_GET_REM_DEV_INFO event.

Reentrant

No

Example

None

Special Notes:

None

3.20. R_BLE_GATTS_SetDbInst()

This function sets GATT Database to host stack.

Format

```
ble_status_t R_BLE_GATTS_SetDbInst  (
      st_ble_gatts_db_cfg_t *       p_db_inst
)
```

Parameters

p_db_inst GATT Database to be set.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The reason for this error is as follows. • The db_inst parameter is specified as NULL. • The array in the db_inst is specified as NULL.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.21. R_BLE_GATT_GetMtu()

This function gets the current MTU used in GATT communication.

Format

```
ble_status_t R_BLE_GATT_GetMtu      (
      uint16_t    conn_hdl,
      uint16_t *  p_mtu
)
```

Parameters

conn_hdl	Connection handle identifying the GATT Server or the GATT Client.
p_mtu	The Current MTU. Before MTU exchange, this parameter is 23 bytes. After MTU exchange, this parameter is the negotiated MTU.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The mtu parameter is NULL.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server or the GATT Client specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

Both GATT server and GATT Client can use this function.

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.22. R_BLE_GATTS_RegisterCb()

This function registers a callback for GATT Server event.

Format

```
ble_status_t R_BLE_GATTS_RegisterCb (
      ble_gatts_app_cb_t      cb,
      uint8_t                 priority
)
```

Parameters

cb	Callback function for GATT Server event.
priority	The priority of the callback function. Valid range is 1 <= priority <= BLE_GATTS_MAX_CB. A lower priority number means a higher priority level.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The cb parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The priority parameter is out of range.
BLE_ERR_CONTEXT_FULL(0x000B)	Host stack has already registered the maximum number of callbacks.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The number of the callback that may be registered by this function is the value specified by R_BLE_GATTS_Init().

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.23. R_BLE_GATTS_DeregisterCb()

This function deregisters the callback function for GATT Server event.

Format

```
ble_status_t R_BLE_GATTS_DeregisterCb     (
      ble_gatts_app_cb_t      cb
)
```

Parameters

cb Callback function for GATT Server event.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The cb parameter is specified as NULL.
BLE_ERR_NOT_FOUND(0x000D)	The callback has not been registered.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.24. R_BLE_GATTS_Notification()

This function sends a notification of an attribute's value.

Format

```
ble_status_t R_BLE_GATTS_Notification     (
      uint16_t                            conn_hdl,
      st_ble_gatt_hdl_value_pair_t *      p_ntf_data
)
```

Parameters

conn_hdl	Connection handle identifying the remote device to be sent the notification.
p_ntf_data	The attribute value to send.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_ntf_data parameter or the value field in the value field in the p_ntf_data parameter is NULL.
BLE_ERR_INVALID_ARG(0x0003)	The value_len field in the value field in the p_ntf_data parameter is 0 or the attr_hdl field in the p_ntf_data parameters is 0.
BLE_ERR_INVALID_OPERATION(0x0009)	This function was called while processing other request.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The maximum length of the attribute value that can be sent with notification is MTU-3.

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.25. R_BLE_GATTS_Indication()

This function sends an indication of an attribute's value.

Format

```
ble_status_t R_BLE_GATTS_Indication (
      uint16_t                          conn_hdl,
      st_ble_gatt_hdl_value_pair_t *    p_ind_data
)
```

Parameters

conn_hdl	Connection handle identifying the remote device to be sent the indication.
p_ind_data	The attribute value to send.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_ind_data parameter or the value field in the value field in the p_ind_data parameter is NULL.
BLE_ERR_INVALID_ARG(0x0003)	The value_len field in the value field in the p_ind_data parameter is 0 or the attr_hdl field in the p_ind_data parameters is 0.
BLE_ERR_INVALID_OPERATION(0x0009)	This function was called while processing other request.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The maximum length of the attribute value that can be sent with indication is MTU-3.

The result of this API call is returned by a return value.

The remote device that receives a indication sends a confirmation.

BLE_GATTS_EVENT_HDL_VAL_CNF event notifies the application layer that the confirmation has been received.

Reentrant

No

Example

None

3.26. R_BLE_GATTS_GetAttr()

This function gets an attribute value from the GATT Database.

Format

```
ble_status_t R_BLE_GATTS_GetAttr    (
      uint16_t                conn_hdl,
      uint16_t                attr_hdl,
      st_ble_gatt_value_t *   p_value
)
```

Parameters

conn_hdl If the attribute value that has information about the remote device is retrieved, specify the remote device with the conn_hdl parameter. When information about the remote device is not required, set the conn_hdl parameter to BLE_GAP_INVALID_CONN_HDL.

attr_hdl The attribute handle of the attribute value to be retrieved.

p_value The attribute value to be retrieved.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_value parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The attr_hdl parameter is 0 or larger than the last attribute handle of GATT Database.
BLE_ERR_INVALID_STATE(0x0008)	The attribute is not in a state to be read.
BLE_ERR_INVALID_OPERATION(0x0009)	The attribute cannot be read.
BLE_ERR_NOT_FOUND(0x000D)	The attribute specified by the attr_hdl parameter is not belonging to any services or characteristics.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by the conn_hdl parameter was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.27. R_BLE_GATTS_SetAttr()

This function sets an attribute value to the GATT Database event.

Format

```
ble_status_t R_BLE_GATTS_SetAttr    (
      uint16_t                conn_hdl,
      uint16_t                attr_hdl,
      st_ble_gatt_value_t *   p_value
)
```

Parameters

conn_hdl	If the attribute value that has information about the remote device is retrieved, specify the remote device with the conn_hdl parameter. When information about the remote device is not required, set the conn_hdl parameter to BLE_GAP_INVALID_CONN_HDL.
attr_hdl	The attribute handle of the attribute value to be set.
p_value	The attribute value to be set.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_value parameter is specified as NULL.
BLE_ERR_INVALID_DATA(0x0002)	The write size is larger than the length of the attribute value.
BLE_ERR_INVALID_ARG(0x0003)	The attr_hdl parameter is 0 or larger than the last attribute handle of GATT Database.
BLE_ERR_INVALID_STATE(0x0008)	The attribute is not in a state to be written.
BLE_ERR_INVALID_OPERATION(0x0009)	The attribute cannot be written.
BLE_ERR_NOT_FOUND(0x000D)	The attribute specified by the attr_hdl parameter is not belonging to any services or characteristics.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by the conn_hdl parameter was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.28. R_BLE_GATTC_RegisterCb()

This function registers a callback function for GATT Client event.

Format

```
ble_status_t R_BLE_GATTC_RegisterCb (
      ble_gattc_app_cb_t      cb,
      uint8_t                 priority
)
```

Parameters

cb	Callback function for GATT Client event.
priority	The priority of the callback function. Valid range is 1 <= priority <= BLE_GATTC_MAX_CB. A lower priority number means a higher priority level.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The cb parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The priority parameter is out of range.
BLE_ERR_CONTEXT_FULL(0x000B)	Host stack has already registered the maximum number of callbacks.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.29. R_BLE_GATTC_DeregisterCb()

This function deregisters the callback function for GATT Client event.

Format

```
ble_status_t R_BLE_GATTC_DeregisterCb     (
      ble_gattc_app_cb_t      cb
)
```

Parameters

cb The callback function to be deregistered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The cb parameter is specified as NULL.
BLE_ERR_NOT_FOUND(0x000D)	The callback has not been registered.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.30. R_BLE_GATTC_ReqExMtu()

This function sends a MTU Exchange Request PDU to a GATT Server in order to change the current MTU.

Format

```
ble_status_t R_BLE_GATTC_ReqExMtu   (
      uint16_t    conn_hdl,
      uint16_t    mtu
)
```

Parameters

conn_hdl	Connection handle identifying the GATT Server to be sent.
mtu	The maximum size(in bytes) of the GATT PDU that GATT Client can receive. Valid range is 23 <= mtu <= 247.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The mtu parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

MTU Exchange Response is notified by BLE_GATTC_EVENT_EX_MTU_RSP event.

The new MTU is the minimum value of the mtu parameter specified by this function and the mtu field in BLE_GATTC_EVENT_EX_MTU_RSP event. Default MTU size is 23 bytes.

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.31. R_BLE_GATTC_DiscAllPrimServ()

This function discovers all Primary Services in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_DiscAllPrimServ  (
      uint16_t    conn_hdl
)
```

Parameters

conn_hdl Connection handle identifying the GATT Server to be discovered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_OPERATION(0x0009)	This function was called while processing other requests.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

When 16-bit UUID Primary Service has been discovered,
BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND event is notified to the application layer.

When 128-bit UUID Primary Service has been discovered,
BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND event is notified to the application layer.

When the Primary Service discovery has been completed,
BLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.32. R_BLE_GATTC_DiscPrimServ()

This function discovers Primary Service specified by p_uuid in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_DiscPrimServ     (
      uint16_t    conn_hdl,
      uint8_t *   p_uuid,
      uint8_t     uuid_type
)
```

Parameters

conn_hdl Connection handle identifying the GATT Server to be discovered.

p_uuid UUID of Primary Service to be discovered.

uuid_type UUID type(16-bit or 128-bit).

macro	description
BLE_GATT_16_BIT_UUID_FORMAT(0x01)	16-bit UUID
BLE_GATT_128_BIT_UUID_FORMAT(0x02)	128-bit UUID

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_uuid parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The uuid_type parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

When Primary Service whose uuid is the same as the specified uuid has been discovered,
BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND event or
BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND event is notified to the application layer.

When the Primary Service discovery has been completed,
BLE_GATTC_EVENT_PRIM_SERV_DISC_COMP event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.33. R_BLE_GATTC_DiscIncServ()

This function discovers Included Services within the specified attribute handle range in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_DiscIncServ      (
      uint16_t    conn_hdl,
      st_ble_gatt_hdl_range_t *     p_range
)
```

Parameters

conn_hdl	Connection handle identifying the GATT Server to be discovered.
p_range	Retrieval range of Included Service.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_range parameter is specified as NULL.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

When Included Service that includes 16-bit UUID Service has been discovered, BLE_GATTC_EVENT_INC_SERV_16_DISC_IND event is notified to the application layer.

When Included Service that includes 128-bit UUID Service has been discovered, BLE_GATTC_EVENT_INC_SERV_128_DISC_IND event is notified to the application layer.

When the Included Service discovery has been completed, BLE_GATTC_EVENT_INC_SERV_DISC_COMP event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.34. R_BLE_GATTC_DiscAllChar()

This function discovers Characteristic within the specified attribute handle range in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_DiscAllChar      (
      uint16_t                      conn_hdl,
      st_ble_gatt_hdl_range_t *     p_range
)
```

Parameters

conn_hdl	Connection handle identifying the GATT Server to be discovered.
p_range	Retrieval range of Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_range parameter is specified as NULL.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

When 16-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_16_DISC_IND event is notified to the application layer.

When 128-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_128_DISC_IND event is notified to the application layer.

When the Characteristic discovery has been completed, BLE_GATTC_EVENT_ALL_CHAR_DISC_COMP event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.35. R_BLE_GATTC_DiscCharByUuid()

This function discovers Characteristic specified by uuid within the specified attribute handle range in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_DiscCharByUuid   (
        uint16_t                          conn_hdl,
        uint8_t *                         p_uuid,
        uint8_t                           uuid_type,
        st_ble_gatt_hdl_range_t *         p_range
)
```

Parameters

conn_hdl Connection handle identifying the GATT Server to be discovered.

p_uuid UUID of Characteristic to be discovered.

uuid_type UUID type of Characteristic to be discovered.

macro	description
BLE_GATT_16_BIT_UUID_FORMAT(0x01)	The p_uuid parameter is 16-bit UUID.
BLE_GATT_128_BIT_UUID_FORMAT(0x02)	The p_uuid parameter is 128-bit UUID.

p_range Retrieval range of Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_uuid parameter or the p_range parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The uuid_type parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

When 16-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_16_DISC_IND event is notified to the application layer.

When 128-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_128_DISC_IND event is notified to the application layer.

When the Characteristic discovery has been completed, BLE_GATTC_EVENT_CHAR_DISC_COMP event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.36. R_BLE_GATTC_DiscAllCharDesc()

This function discovers Characteristic Descriptor within the specified attribute handle range in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_DiscAllChar      (
      uint16_t                      conn_hdl,
      st_ble_gatt_hdl_range_t *     p_range
)
```

Parameters

conn_hdl	Connection handle identifying the GATT Server to be discovered.
p_range	Retrieval range of Characteristic Descriptor.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_range parameter is specified as NULL.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

When 16-bit UUID Characteristic Descriptor has been discovered,
BLE_GATTC_EVENT_CHAR_DESC_16_DISC_IND event is notified to the application layer.

When 128-bit UUID Characteristic Descriptor has been discovered,
BLE_GATTC_EVENT_CHAR_DESC_128_DISC_IND event is notified to the application layer.

When the Characteristic Descriptor discovery has been completed,
BLE_GATTC_EVENT_ALL_CHAR_DESC_DISC_COMP event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.37. R_BLE_GATTC_ReadChar()

This function reads a Characteristic/Characteristic Descriptor in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_ReadChar   (
      uint16_t    conn_hdl,
      uint16_t    value_hdl
)
```

Parameters

conn_hdl	Connection handle identifying the GATT Server to be read.
value_hdl	Value handle of the Characteristic/Characteristic Descriptor to be read.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	0 is specified in the value_hdl parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of the read is notified in BLE_GATTC_EVENT_CHAR_READ_RSP event.

Reentrant

No

Example

None

Special Notes:

None

3.38. R_BLE_GATTC_ReadCharUsingUuid()

This function reads a Characteristic in a GATT Server using a specified UUID.

Format

```
ble_status_t R_BLE_GATTC_ReadCharUsingUuid      (
      uint16_t                      conn_hdl,
      uint8_t *                     p_uuid,
      uint8_t                       uuid_type,
      st_ble_gatt_hdl_range_t *     p_range
)
```

Parameters

conn_hdl Connection handle that identifies Characteristic to be read to GATT Server.

p_uuid UUID of the Characteristic to be read.

uuid_type UUID type of the Characteristic to be read.

macro	description
BLE_GATT_16_BIT_UUID_FORMAT(0x01)	The p_uuid parameter is 16-bit UUID.
BLE_GATT_128_BIT_UUID_FORMAT(0x02)	The p_uuid parameter is 128-bit UUID.

p_range Retrieval range of Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_uuid parameter or the p_range parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The uuid_type parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of the read is notified in BLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP event.

Reentrant

No

Example

None

Special Notes:

None

3.39. R_BLE_GATTC_ReadLongChar()

This function reads a Long Characteristic in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_ReadLongChar     (
      uint16_t    conn_hdl,
      uint16_t    value_hdl,
      uint16_t    offset
)
```

Parameters

conn_hdl	Connection handle identifying the GATT Server to be read.
value_hdl	Value handle of the Long Characteristic to be read.
offset	Offset that indicates the location to be read. Normally, set 0 to this parameter.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	0 is specified in the value_hdl parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The contents of the Long Characteristic that has been read is notified every MTU-1 bytes to the application layer by BLE_GATTC_EVENT_CHAR_READ_RSP event.

When all of the contents has been received in GATT Client,
BLE_GATTC_EVENT_LONG_CHAR_READ_COMP event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.40. R_BLE_GATTC_ReadMultiChar()

This function reads multiple Characteristics in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_ReadMultiChar    (
      uint16_t                                conn_hdl,
      st_ble_gattc_rd_multi_req_param_t *     p_list
)
```

Parameters

conn_hdl Connection handle that identifies Characteristic to be read to GATT Server.

p_list List of Value Handles that point the Characteristics to be read.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_list parameter or the p_hdl_list field in the p_list parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	0 is specified in the value_hdl parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The contents of the multiple Characteristics that has been read is notified to the application layer by BLE_GATTC_EVENT_MULTI_CHAR_READ_RSP event.

Reentrant

No

Example

None

Special Notes:

None

3.41. R_BLE_GATTC_WriteCharWithoutRsp()

This function writes a Characteristic in a GATT Server without response.

Format

```
ble_status_t R_BLE_GATTC_WriteCharWithoutRsp    (
      uint16_t                          conn_hdl,
      st_ble_gatt_hdl_value_pair_t *    p_write_data
)
```

Parameters

conn_hdl	Connection handle that identifies Characteristic to be read to GATT Server.
p_write_data	Value to be written to the Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • 0 is specified in the value_len field in the p_value field in the p_write_data parameter. • 0 is specified in the attr_hdl field in the p_write_data parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result is returned from the API.

Reentrant

No

Example

None

Special Notes:

None

3.42. R_BLE_GATTC_SignedWriteChar()

This function writes Signed Data to a Characteristic in a GATT Server without response.

Format

```
ble_status_t R_BLE_GATTC_SignedWriteChar  (
      uint16_t                            conn_hdl,
      st_ble_gatt_hdl_value_pair_t *      p_write_data
)
```

Parameters

conn_hdl	Connection handle identifying the GATT Server to be written.
p_write_data	Signed Data to be written to the Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • 0 is specified in the value_len field in the value field in the p_write_data parameter. • 0 is specified in the attr_hdl field in the p_write_data parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.43. R_BLE_GATTC_WriteChar()

This function writes a Characteristic in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_WriteChar  (
      uint16_t                              conn_hdl,
      st_ble_gatt_hdl_value_pair_t *        p_write_data
)
```

Parameters

conn_hdl	Connection handle identifying the GATT Server to be written.
p_write_data	Signed Data to be written to the Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • 0 is specified in the value_len field in the value field in the p_write_data parameter. • 0 is specified in the attr_hdl field in the p_write_data parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of the write is notified in BLE_GATTC_EVENT_CHAR_WRITE_RSP event.

Reentrant

No

Example

None

Special Notes:

None

3.44. R_BLE_GATTC_WriteLongChar()

This function writes a Long Characteristic in a GATT Server.

Format

```
ble_status_t R_BLE_GATTC_WriteLongChar    (
      uint16_t                            conn_hdl,
      st_ble_gatt_hdl_value_pair_t *      p_write_data,
      uint16_t                            offset
)
```

Parameters

conn_hdl	Connection handle identifying the GATT Server to be written.
p_write_data	Value to be written to the Long Characteristic.
offset	Offset that indicates the location to be written. Normally, set 0 to this parameter. If this parameter sets to a value other than 0, adjust the offset parameter and the length of the value to be written not to exceed the length of the Long Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • The value_len field in the value field in the p_write_data parameter is 0. • The sum of the value_len field in the value field in the p_write_data parameter and the offset parameter larger than 512. • The attr_hdl field in the p_write_data parameter is 0.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of a write that has been done every segmentation is notified to the application layer in BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP event.

The maximum writable size to a Long Characteristic with this function is 512 bytes.

When all of the contents has been written to the Long Characteristic, BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.45. R_BLE_GATTC_ReliableWrites()

This function performs the Reliable Writes procedure described in GATT Specification.

Format

```
ble_status_t R_BLE_GATTC_ReliableWrites   (
      uint16_t                                          conn_hdl,
      st_ble_gattc_reliable_writes_char_pair_t *        p_char_pair,
      uint8_t                                           pair_num,
      uint8_t                                           auto_flag
)
```

Parameters

conn_hdl Connection handle identifying the GATT Server to be written.

p_char_pair Pair of Characteristic Value and Characteristic Value Handle identifying the Characteristic to be written by Reliable Writes.

pair_num The number of the pairs specified by the p_char_pair parameter.

Valid range is 0 < pair_num <= BLE_GATTC_RELIABLE_WRITES_MAX_CHAR_PAIR.

auto_flag The flag that indicates whether auto execution or not.

macro	description
BLE_GATTC_EXEC_AUTO(0x01)	Auto execution.
BLE_GATTC_EXEC_NOT_AUTO (0x02)	Not auto execution.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The reason for this error is as follows: • The p_char_pair parameter is specified as NULL. • The p_value field in the value field in the write_data field in the p_char_pair parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • The pair_num parameter or the auto_flag parameter is out of range. • The value_len field in the value field in the write_data field in the p_char_pair parameter is 0.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function or to store the temporary write data.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

When the data written to the Characteristic has been transmitted, BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP event is notified to the application layer.

If the data included in the event is different from the data that GATT Client has sent, host stack automatically cancels the Reliable Writes.

After all of the contents has been sent to the GATT Server, if the auto_flag parameter has been set to BLE_GATTC_EXEC_AUTO, the GATT Server automatically writes the data to the Characteristic.

If the auto_flag parameter has been set to BLE_GATTC_EXEC_NOT_AUTO, BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP event notifies the application layer in GATT Client that all of the contents has been sent to the GATT Server. Then GATT Client requests for writing the data to the Characteristic to the GATT Server with R_BLE_GATTC_ExecWrite().

When the write has been done, BLE_GATTC_EVENT_RELIABLE_WRITES_COMP event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.46. R_BLE_GATTC_ExecWrite()

This function is used to execute a write to Characteristic.

Format

```
ble_status_t R_BLE_GATTC_ExecWrite  (
      uint16_t    conn_hdl,
      uint8_t     exe_flag
)
```

Parameters

conn_hdl Connection handle identifying the target GATT Server.

exe_flag The flag that indicates whether execution or cancellation.

macro	description
BLE_GATTC_EXECUTE_WRITE_CANCEL_FLAG(0x00)	Execute the write.
BLE_GATTC_EXECUTE_WRITE_EXEC_FLAG(0x01)	Cancel the write.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The exe_flag parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	The reason for this error is as follows: • GATT Client has not requested for Reliable Writes by R_BLE_GATTC_ReliableWrites(). • Although auto execution has been specified by R_BLE_GATTC_ReliableWrites(), this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

When all of the contents has been sent to the GATT Server, BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP event notifies the application layer.

After this event has been received, execute the write by this function.

The result of the write is notified by BLE_GATTC_EVENT_RELIABLE_WRITES_COMP event.

Reentrant

No

Example

None

Special Notes:

None

3.47. R_BLE_ L2CAP_RegisterCfPsm()

This function registers PSM that uses L2CAP CBFC Channel and a callback for L2CAP event.

Format

```
ble_status_t R_BLE_L2CAP_RegisterCfPsm    (
      ble_l2cap_cf_app_cb_t   cb,
      uint16_t                psm,
      uint16_t                lwm
)
```

Parameters

cb Callback function for L2CAP event.

psm Identifier indicating the protocol/profile that uses L2CAP CBFC Channel.

type	range	description
Fixed, SIG assigned	0x0001 - 0x007F	PSM defined by SIG. For more information on PSM, refer Bluetooth SIG Assigned Number. (https://www.bluetooth.com/specifications/assigned-numbers).
Dynamic	0x0080 - 0x00FF	Statically allocated PSM by custom protocol or dynamically allocated PSM by GATT Service.

lwm Low Water Mark that indicates the LE-Frame numbers that the local device can receive.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The cb parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The psm parameter is out of range.
BLE_ERR_CONTEXT_FULL(0x000B)	More than BLE_L2CAP_MAX_CBFC_PSM+1 PSMs, callbacks has been registered.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

Only one callback is available per PSM. Configure in each PSM the Low Water Mark of the LE-Frames that the local device can receive.

When the number of the credit reaches the Low Water Mark,
BLE_L2CAP_EVENT_CF_LOW_RX_CRD_IND event is notified to the application layer.

The number of PSM is defined as BLE_L2CAP_MAX_CBFC_PSM.

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.48. R_BLE_ L2CAP_DeregisterCfPsm()

This function stops the use of the L2CAP CBFC Channel specified by the psm parameter and deregisters the callback function for L2CAP event.

Format

```
ble_status_t R_BLE_L2CAP_DeregisterCfPsm  (
      uint16_t    psm
)
```

Parameters

psm PSM that is to be stopped to use the L2CAP CBFC Channel.
Set the PSM registered by R_BLE_VS_Init().

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_NOT_FOUND(0x000D)	The callback function allocated by the psm parameter is not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.49. R_BLE_ L2CAP_ReqCfConn()

This function sends a connection request for L2CAP CBFC Channel.

Format

```
ble_status_t R_BLE_L2CAP_ReqCfConn  (
      uint16_t                          conn_hdl,
      st_ble_l2cap_conn_req_param_t *   p_conn_req_param
)
```

Parameters

conn_hdl	Connection handle identifying the remote device that the connection request is sent to.
p_conn_req_param	Connection request parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_conn_req_param parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The mtu parameter or the mps parameter is out of range.
BLE_ERR_INVALID_STATE(0x0008)	CF Channel connection has not been established.
BLE_ERR_CONTEXT_FULL(0x000B)	New CF Channel can not be registered or other L2CAP Command is processing.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.
BLE_ERR_NOT_YET_READY(0x0012)	The psm parameter is not registered.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The connection response is notified by BLE_L2CAP_EVENT_CF_CONN_CNF event.

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.50. R_BLE_ L2CAP_DisconnetCf()

This function sends a disconnection request for L2CAP CBFC Channel.

Format

```
ble_status_t R_BLE_L2CAP_DisconnectCf     (
      uint16_t    lcid
)
```

Parameters

lcid	CID identifying the L2CAP CBFC Channel that has been disconnected. The valid range is 0x40 - (0x40 + BLE_L2CAP_MAX_CBFC_PSM - 1).

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_OPERATION(0x0009)	CF Channel connection has not been established.
BLE_ERR_CONTEXT_FULL(0x000B)	This function was called while processing other L2CAP command.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for L2CAP Command.
BLE_ERR_NOT_FOUND(0x000D)	CID specified the lcid parameter is not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

When L2CAP CBFC Channel has been disconnected, BLE_L2CAP_EVENT_CF_DISCONN_CNF event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.51. R_BLE_ L2CAP_SendCfCredit()

This function sends credit to a remote device.

Format

```
ble_status_t R_BLE_L2CAP_SendCfCredit     (
      uint16_t    lcid,
      uint16_t    credit
)
```

Parameters

lcid	CID identifying the L2CAP CBFC Channel on local device that sends credit.
credit	Credit to be sent to the remote device.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The credit parameter is set to 0.
BLE_ERR_CONTEXT_FULL(0x000B)	This function was called while processing other L2CAP command.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for L2CAP Command.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

In L2CAP CBFC communication, if credit is 0, the remote device stops data transmission.

Therefore when processing the received data has been completed and local device affords to receive data, the remote device is notified of the number of LE-Frame that local device can receive by this function and local device can continue to receive data from the remote device.

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.52. R_BLE_ L2CAP_SendCfData()

This function sends the data to a remote device via L2CAP CBFC Channel.

Format

```
ble_status_t R_BLE_L2CAP_SendCfData (
      uint16_t    conn_hdl,
      uint16_t    lcid,
      uint16_t    data_len,
      uint8_t *   p_sdu
)
```

Parameters

conn_hdl	Connection handle identifying the remote device to be sent the data.
lcid	CID identifying the L2CAP CBFC Channel on local device used in the data transmission.
data_len	Length of the data.
p_sdu	Service Data Unit. Input the data length specified by the data_len parameter to the first 2 bytes (Little Endian).

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_data parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The length parameter is out of range.
BLE_ERR_INVALID_STATE(0x0008)	CF Channel connection has not been established or the data whose length exceeds the MTU has been sent.
BLE_ERR_ALREADY_IN_PROGRESS(0x000A)	Data transmission has been already started.
BLE_ERR_CONTEXT_FULL(0x000B)	L2CAP task queue is full.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for L2CAP Command.
BLE_ERR_NOT_FOUND(0x000D)	CID specified the lcid parameter is not found.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by the conn_hdl parameter is not found.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

When the data transmission to Controller has been completed, BLE_L2CAP_EVENT_CF_TX_DATA_CNF event is notified to the application layer.

Reentrant

No

Example

None

Special Notes:

None

3.53. R_BLE_VS_Init()

This function initializes Vendor Specific API and registers a callback function for Vendor Specific Event.

Format

```
ble_status_t R_BLE_VS_Init    (
      ble_vs_app_cb_t   vs_cb
)
```

Parameters

vs_cb Callback function to be registered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The vs_cb parameter is specified as NULL.
BLE_ERR_CONTEXT_FULL(0x000B)	Callback function has already been registered.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of this API call is returned by a return value.

Reentrant

No

Example

None

Special Notes:

None

3.54. R_BLE_VS_GetBdAddr()

This function gets currently configured public/random address.

Format

```
ble_status_t R_BLE_VS_GetBdAddr     (
      uint8_t     area,
      uint8_t     addr_type
)
```

Parameters

area The area that the address is to be retrieved.

Select one of the following.

macro	description
BLE_VS_ADDR_AREA_REG(0x00)	Retrieve the address in register.
BLE_VS_ADDR_AREA_DATA_FLASH(0x01)	Retrieve the address in DataFlash area.

addr_type The address type that is type of the address to be retrieved.

macro	description
BLE_GAP_ADDR_PUBLIC(0x00)	Public address.
BLE_GAP_ADDR_RAND(0x01)	Random address.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The area parameter specifies the place where this function retrieves public/random address.

The result of this API call is notified in BLE_VS_EVENT_GET_ADDR_COMP event.

Reentrant

No

Example

None

Special Notes:

None

3.55. R_BLE_VS_SetBdAddr()

This function sets public/random address of local device to the area specified by the parameter.

Format

```
ble_status_t R_BLE_VS_SetBdAddr     (
      uint8_t                 area,
      st_ble_dev_addr_t *     p_addr
)
```

Parameters

area The area that the address is to be written in.

Select one of the following.

macro	description
BLE_VS_ADDR_AREA_REG(0x00)	Address writing to non-volatile area is not performed. Only the address in register is written.
BLE_VS_ADDR_AREA_DATA_FLASH(0x01)	Address wiring to DataFlash area is performed.

p_addr The address to be set to the area. Set BLE_GAP_ADDR_PUBLIC(0x00) or BLE_GAP_ADDR_RAND(0x01) to the type field in the p_addr parameter.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_addr parameter is specified as NULL.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

If the address is written in non-volatile area, the address is used as default address on the next MCU reset.

For more information on the random address, refer to Core Specification Vol 6, PartB, "1.3.2 Random Device Address".

The result of this API call is notified in BLE_VS_EVENT_SET_ADDR_COMP event.

Reentrant

No

Example

None

Special Notes:

None

3.56. R_BLE_VS_GetRand()

This function generates 4-16 bytes of random number used in creating keys.

Format

```
ble_status_t R_BLE_VS_GetRand (
      uint8_t     rand_size
)
```

Parameters

rand_size Length of the random number (byte).
The valid range is 4<=rand_size<=16.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

Properties

Prototype declarations are contained in r_ble_api.h.

Description

The result of this API call is notified in BLE_VS_EVENT_GET_RAND event.

Reentrant

No

Example

None

Special Notes:

None

4. Abstraction API for Renesas QE for BLE

4.1 RM_BLE_ABS_Open()

Host stack is initialized with this function.

Format

```
fsp_err_t RM_BLE_ABS_Open     (
      ble_abs_ctrl_t * const p_ctrl,
      ble_abs_cfg_t * p_cfg
)
```

Parameters

p_ctrl	Pointer to control structure.
p_cfg	Pointer to the configuration structure for this instance.

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Null pointer presented.
FSP_ERR_ALREADY_OPEN	Requested channel is already open in a different configuration.
FSP_ERR_INVALID_ARGUMENT	Invalid input parameter.
FSP_ERR_INVALID_MODE	Invalid mode during open call.

Properties

Prototype declarations are contained in rm_ble_abs.h.

Description

Before using All the R_BLE APIs, it's necessary to call this function. A callback functions are registered with this function. In order to receive the GAP, GATT, Vendor specific event, it's necessary to register a callback function. The result of this API call is notified in BLE_GAP_EVENT_STACK_ON event. Implements ble_abs_api_t::open.

Reentrant

No

Example

```
/* Open the module. */
err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
```

Special Notes:

None

4.2 RM_BLE_ABS_Close()

Close the BLE channel.

Format

```
fsp_err_t RM_BLE_ABS_Close    (
      ble_abs_ctrl_t * const p_ctrl
)
```

Parameters

p_ctrl Pointer to control structure.

Return values

FSP_SUCCESS	Channel closed successfully.
FSP_ERR_ASSERTION	Null pointer presented.
FSP_ERR_NOT_OPEN	Control block not open.

Properties

Prototype declarations are contained in rm_ble_abs.h.

Description

Implements ble_abs_api_t::close.

Reentrant

No

Example

```
/* Close BLE driver */
err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
```

Special Notes:

None

4.3 RM_BLE_ABS_StartLegacyAdvertising()

Start Legacy Advertising after setting advertising parameters, advertising data and scan response data.

Format

```
fsp_err_t RM_BLE_ABS_StartLegacyAdvertising     (
ble_abs_ctrl_t * const                                  p_ctrl,
ble_abs_legacy_advertising_parameter_t const * const  p_advertising_parameter
)
```

Parameters

p_ctrl	Pointer to control structure.
p_advertising_parameter	Pointer to Advertising parameters for Legacy Advertising.

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_STATE	Host stack hasn't been initialized.
FSP_ERR_INVALID_POINTER	p_advertising_parameter is specified as NULL.
FSP_ERR_INVALID_ARGUMENT	The advertising parameter is out of range.

Properties

Prototype declarations are contained in rm_ble_abs.h.

Description

Legacy advertising uses the advertising set whose advertising handle is 0. The advertising type is connectable and scannable (ADV_IND). The address type of local device is Public Identity Address or RPA (If the resolving list contains no matching entry, use the public address.). Scan request event (BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startLegacyAdvertising.

Reentrant

No

Example

```
  /* Start advertising. */
  err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
```

Special Notes:

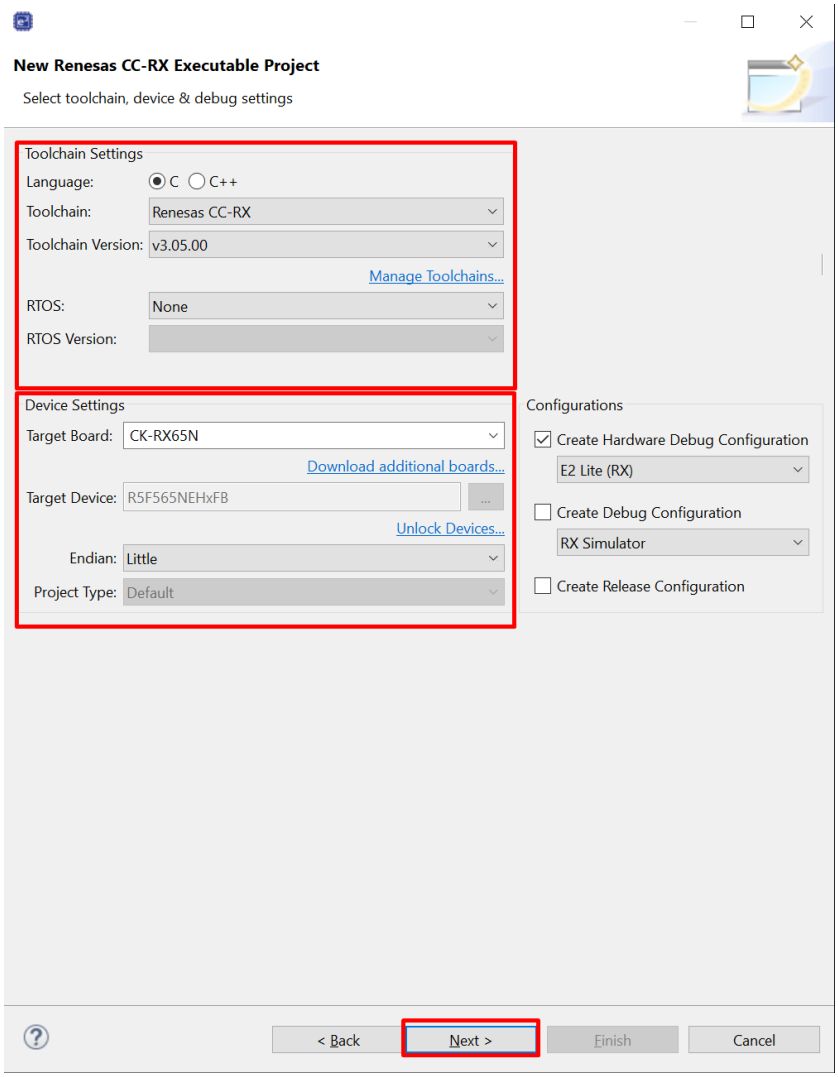
None

5. Sample Code Generation Using QE for BLE

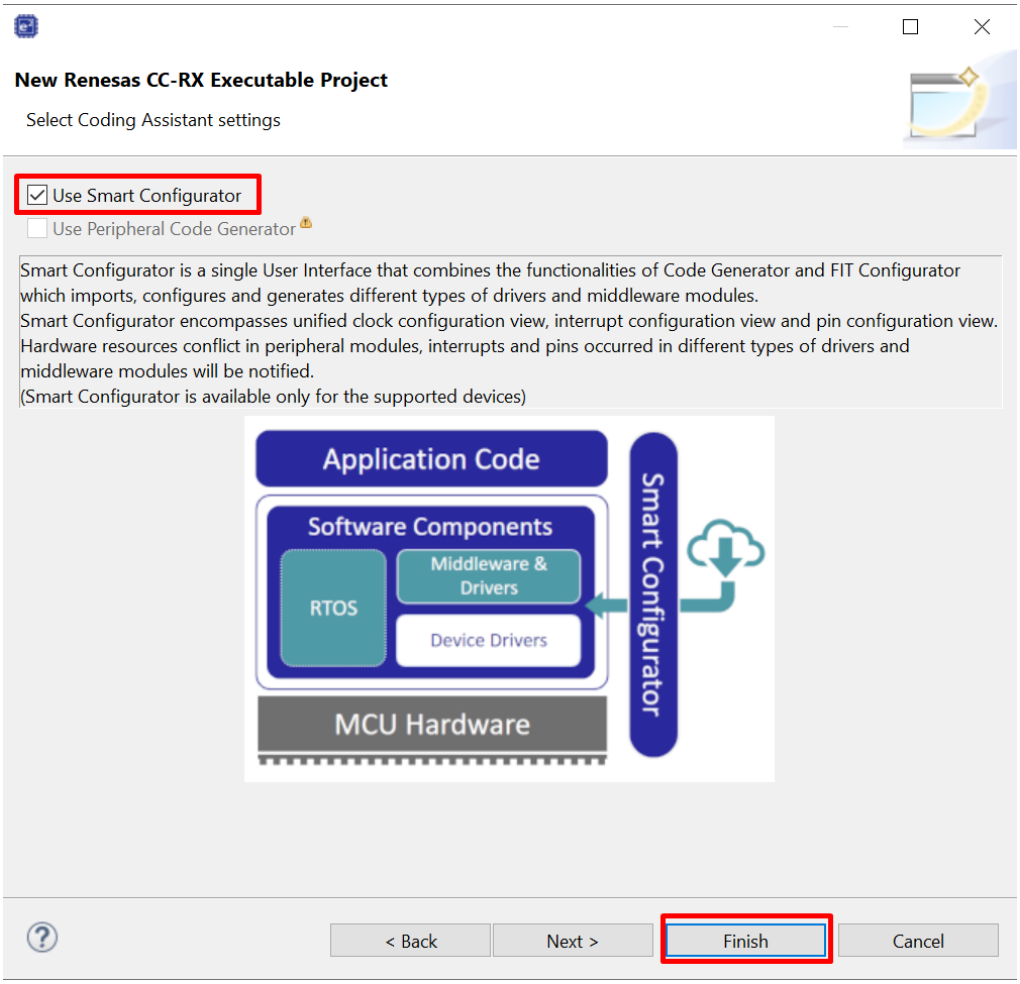
This section describes how to generate sample code using QE for BLE. The settings in this section are an example when using CK-RX65N as a Target Board.

(1) Create a new e^2 studio project with the following settings:

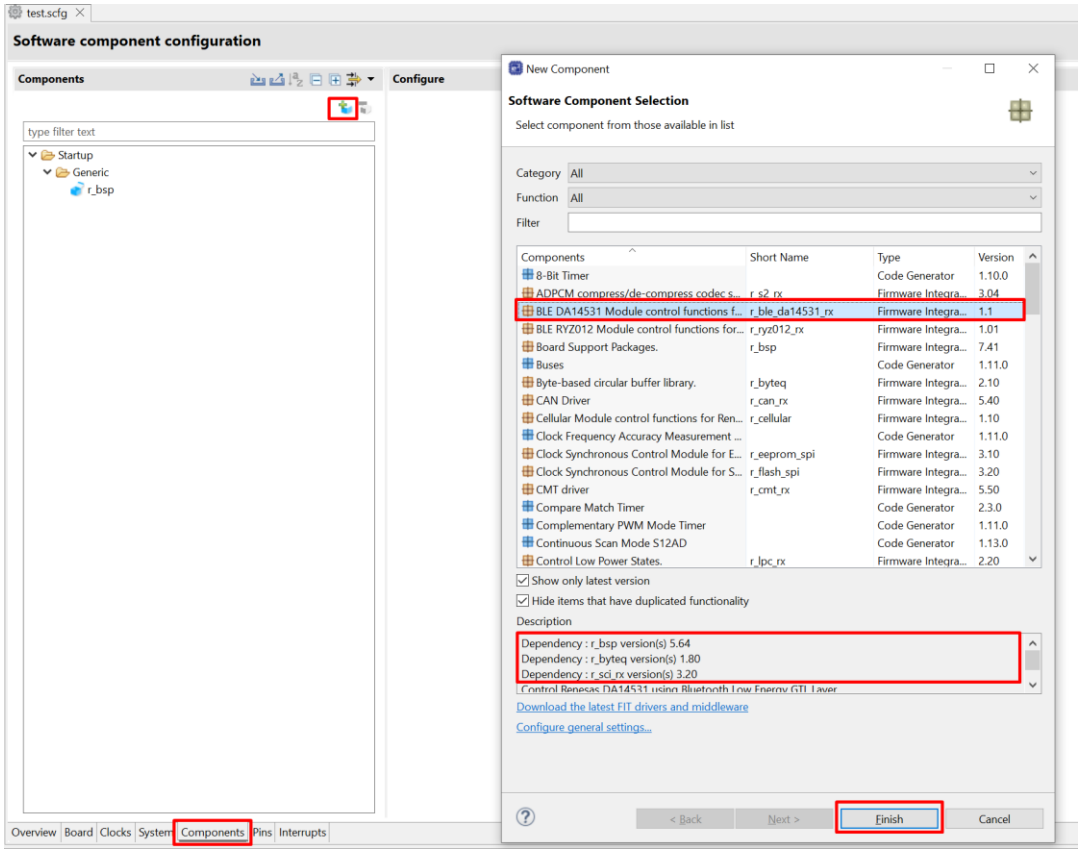
- Renesas CC-RX C/C++ Executable Project
- Project name: (Arbitrary)
- RTOS: None (for Baremetal), FreeRTOS (kernel only) or Azure RTOS
- Target Board: CK-RX65N



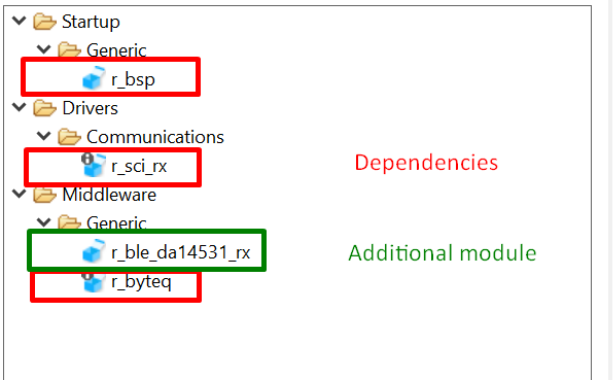
(2) Check **Use Smart Configurator** and click the **Finish** button.



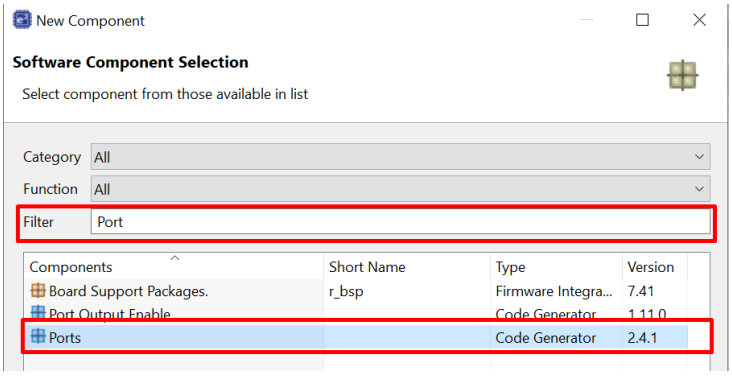
(3) Click **Add component** button on **Components** tab in the Smart Configurator perspective to add the FIT module to the list of components in **New Component** window. Select the FIT module added and click **Finish** button on the window.



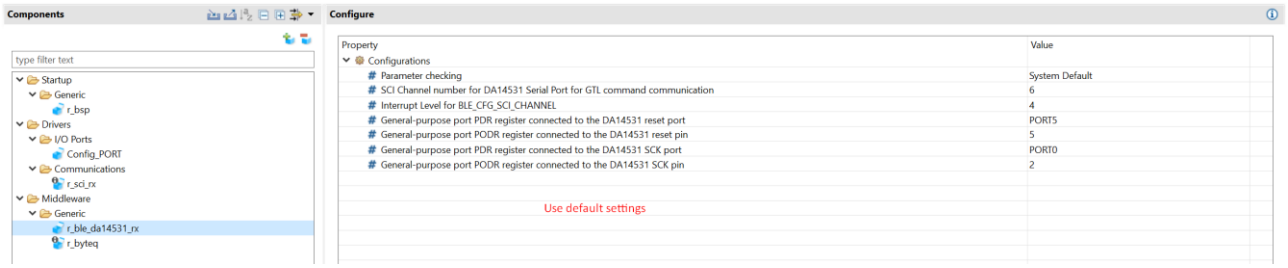
Make sure the FIT module is added.



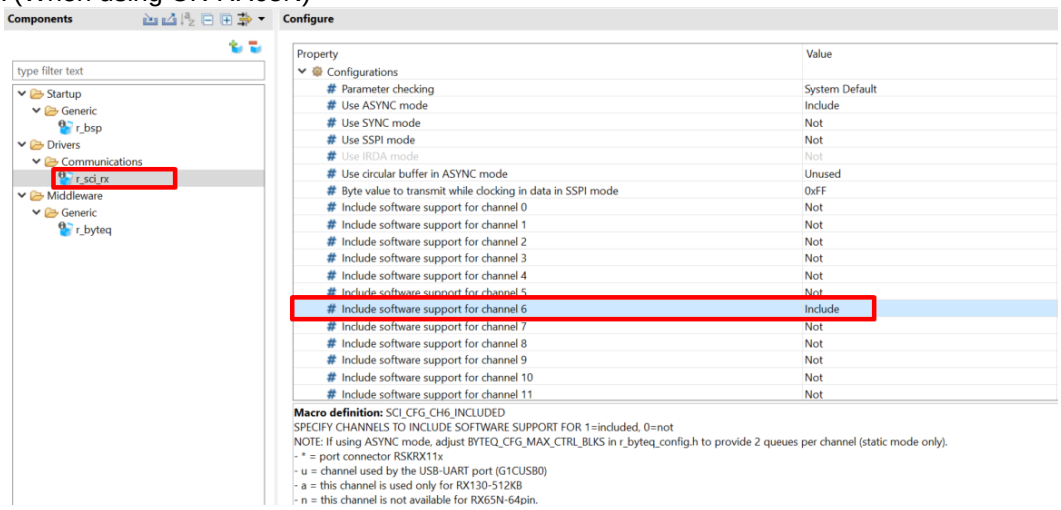
Add other FIT modules to serve the sample app

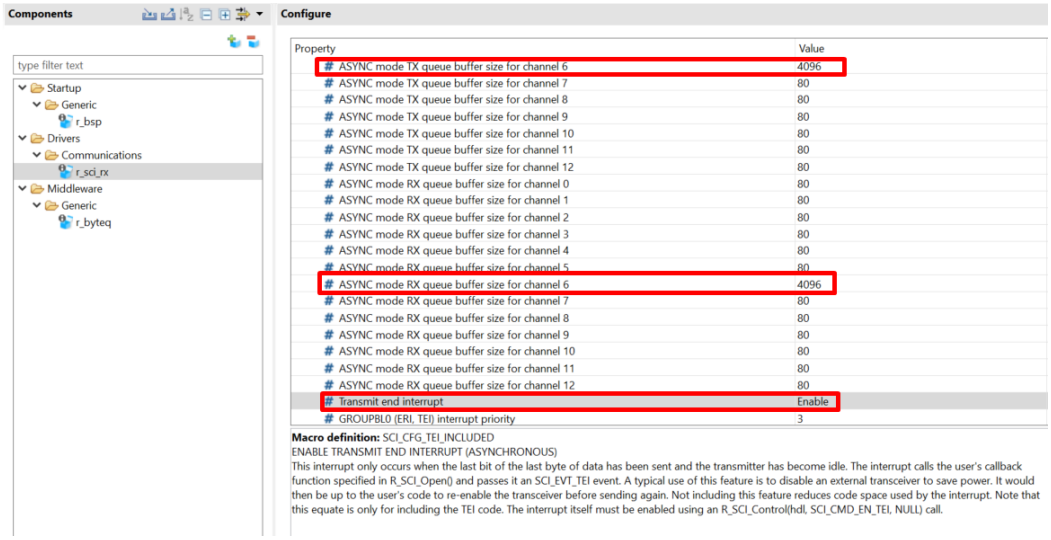


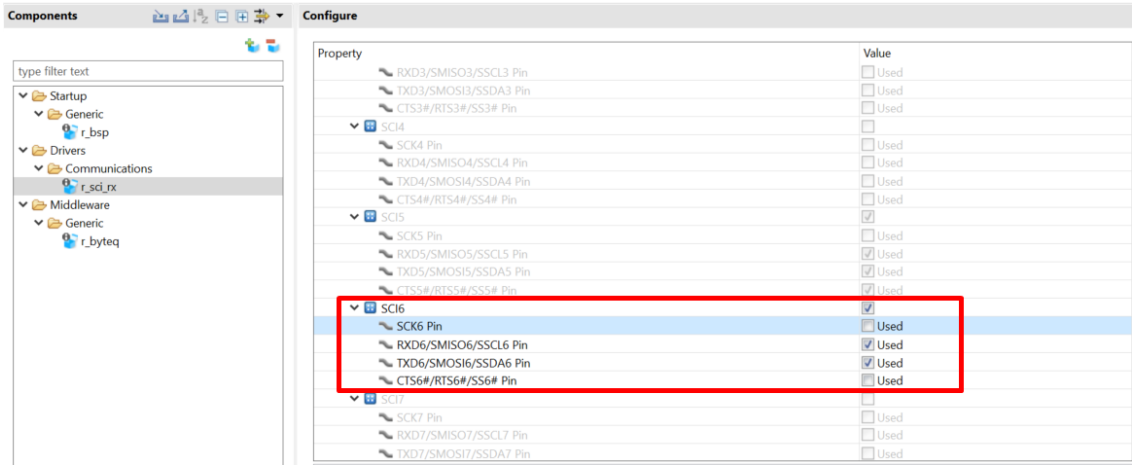
(4) Set configures for PMOD1.
r_ble_da14531_rx



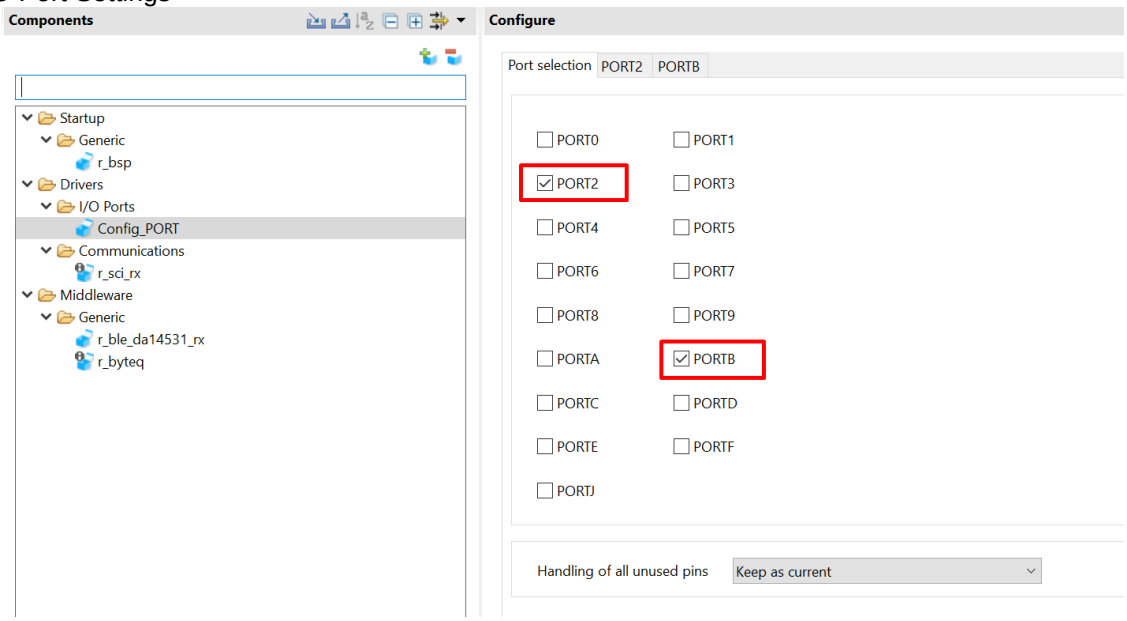
r_sci_rx (When using CK-RX65N)

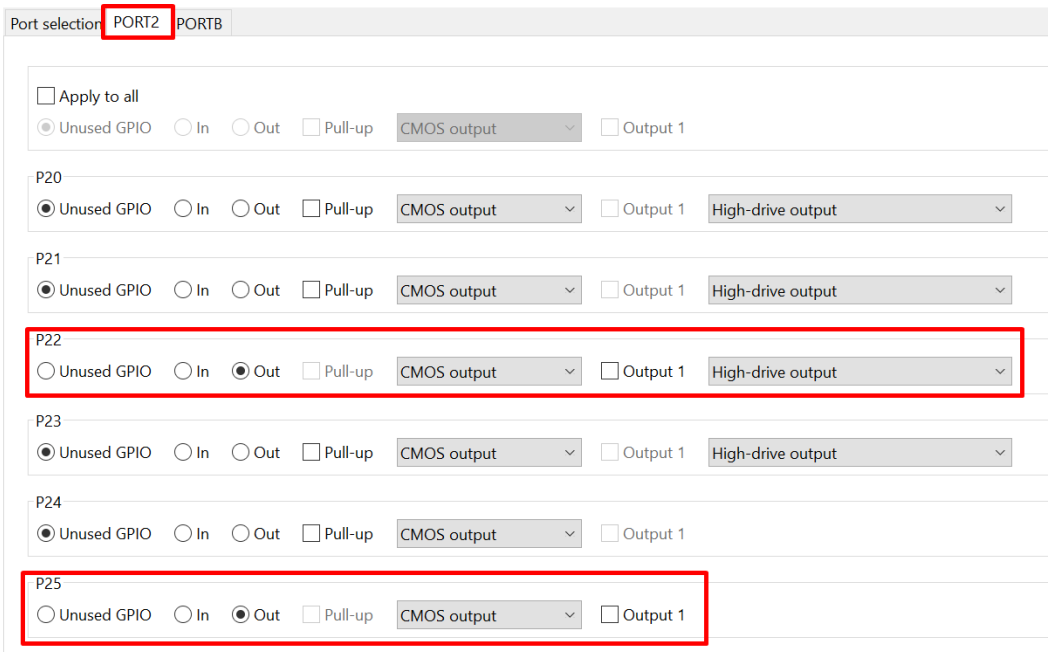


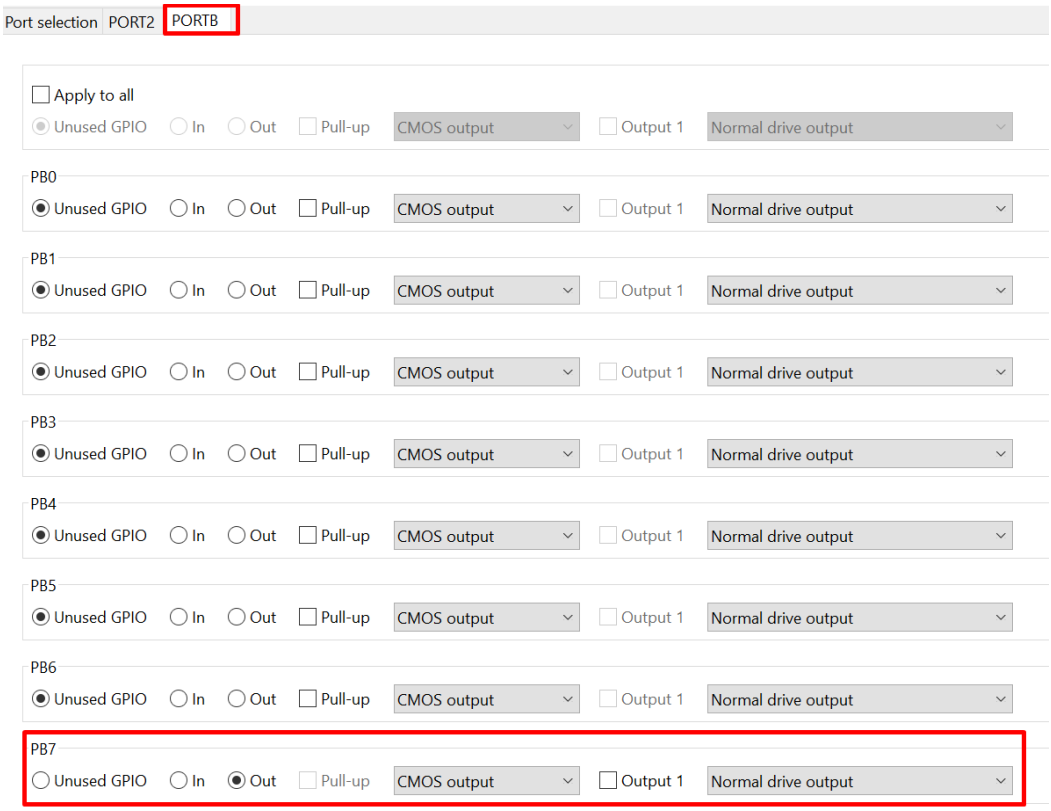




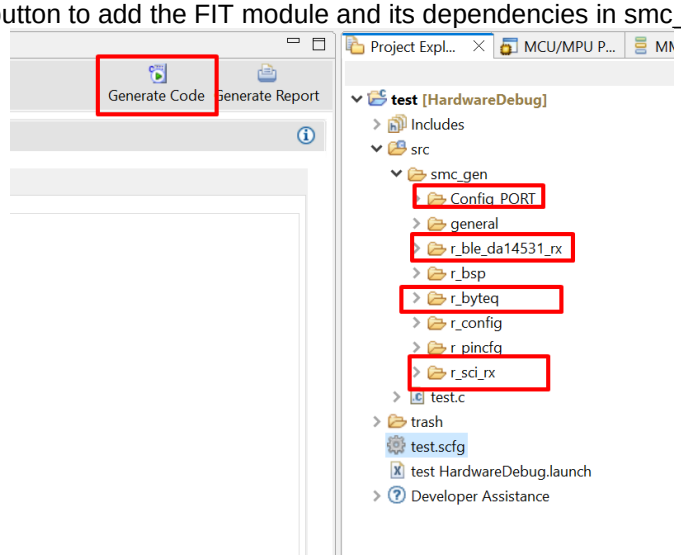
GPIO Port Settings





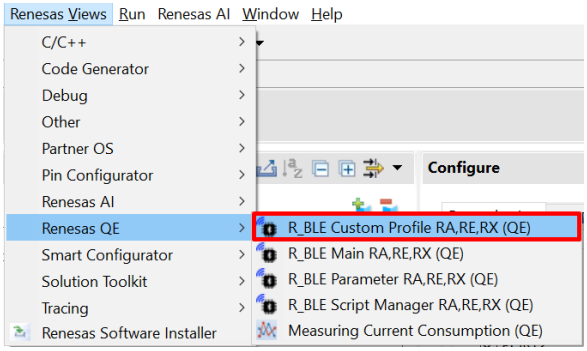

Port selection
PORT2
PORTB
Apply to all
Unused GPIO
In
Out
Pull-up
CMOS output
Output 1
Normal drive output
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7

(5) Click **Generate Code** button to add the FIT module and its dependencies in smc_gen folder.

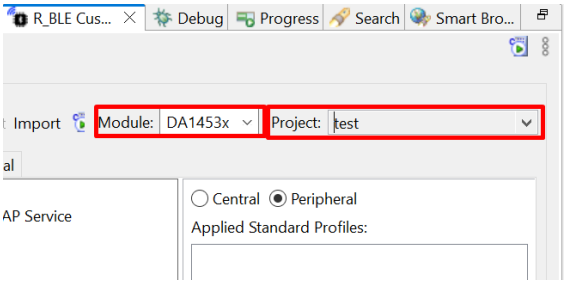


(6) Open QE for BLE window to generate sample code.

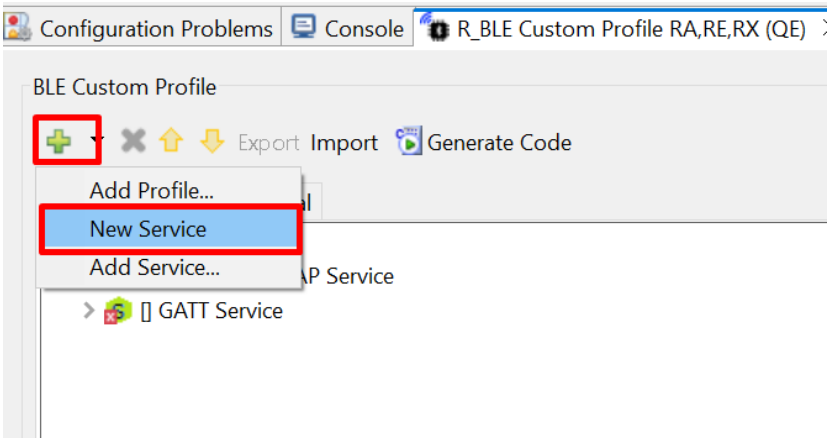
1) Open R_BLE_Custom Profile Tab

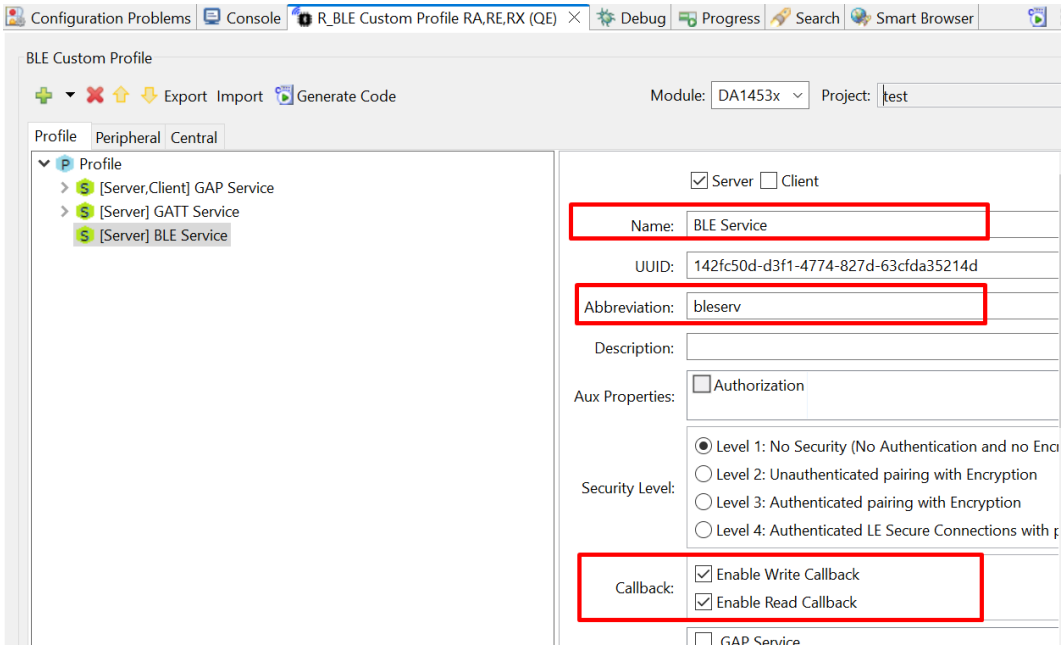


2) Select main project and module “DA1453X”

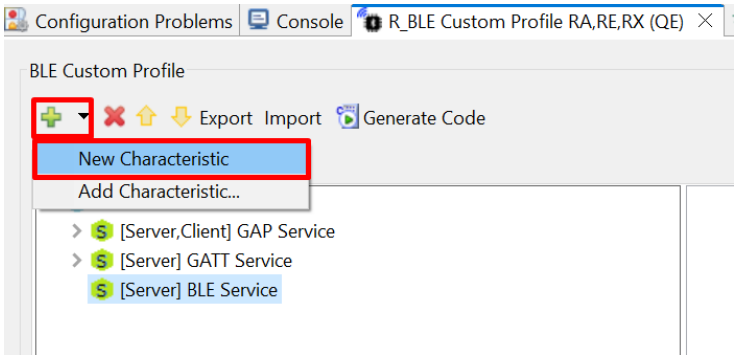


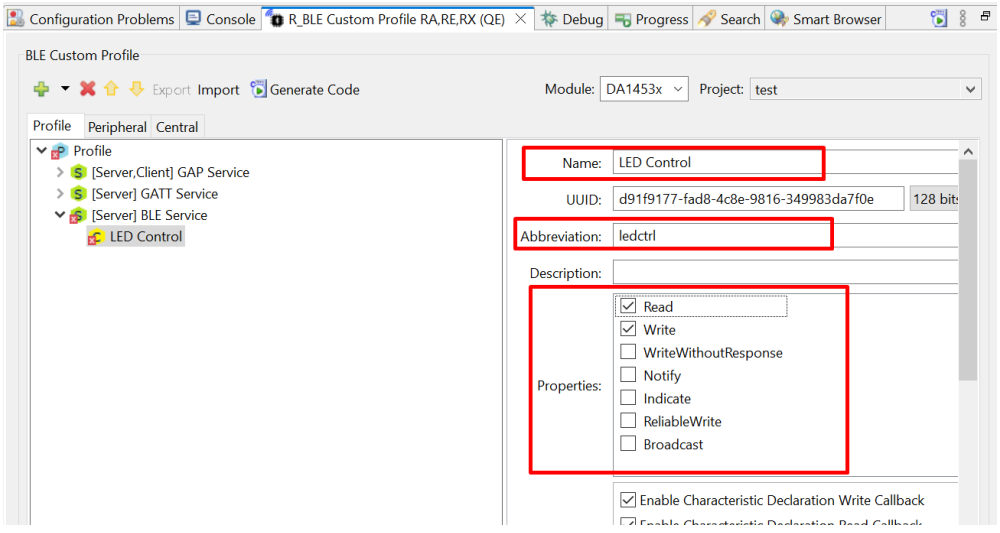
3) Create new service

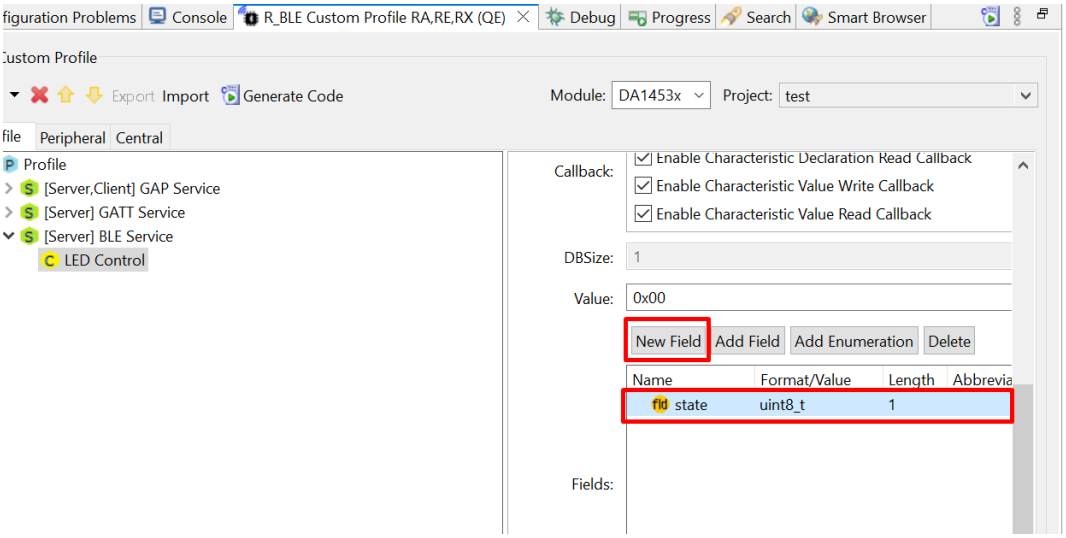




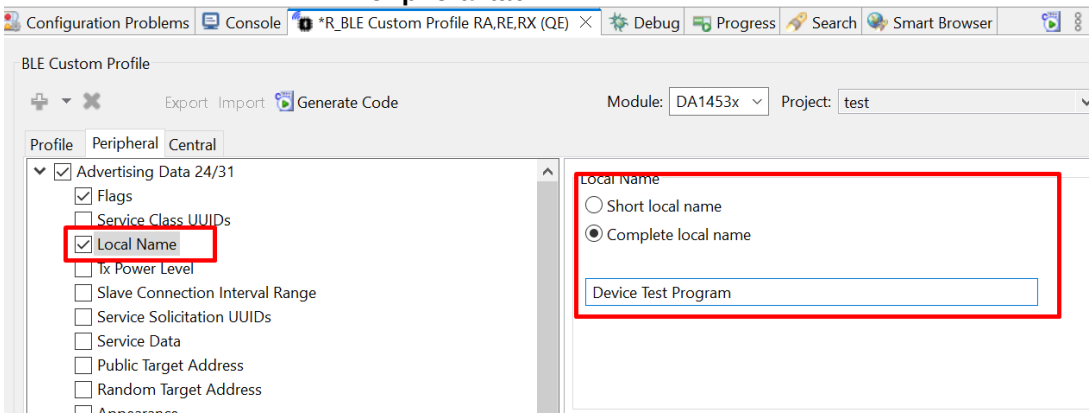
4) Create new Characteristic

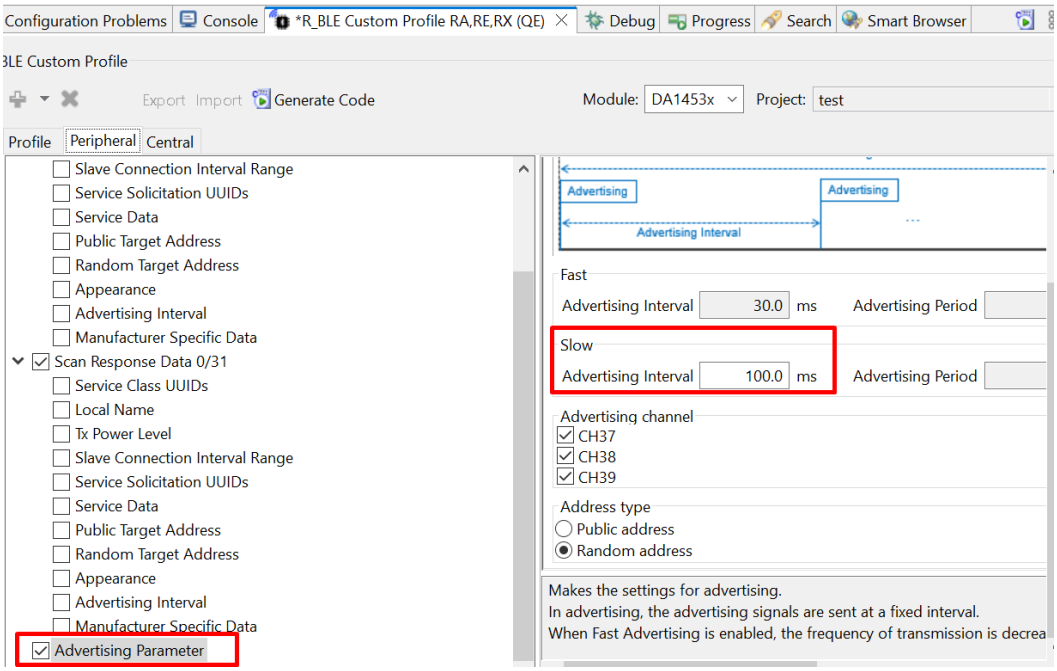




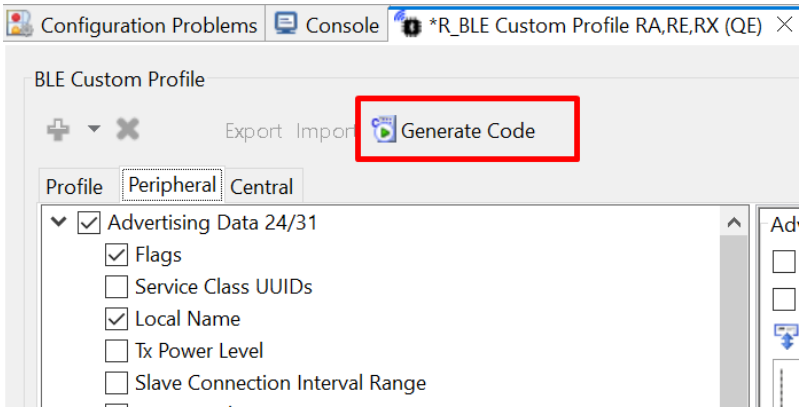


5) Check “Local Name” checkbox in **Peripheral** tab.





6) Click **Generate Code** button to make QE for BLE generate sample code.



(7) In e² studio project explorer, open the file src\[Project name].c including the main function and add the yellow highlighted code, resulting in the code shown below:

```
#include "r_smc_entry.h"

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

void main_task(void)
{
    app_main();

    while(1) {
        R_BSP_SoftwareDelay(1000, BSP_DELAY_MILLISECS);
    }
}
```

(8) In e² studio project explorer, open the file qe_gen\ble\app_main.c including the app_main function, the bleservs_cb function and add the yellow highlighted code, resulting in the code shown below:
Add macro definitions:

```
/* Start user code for macro definitions. Do not edit comment generated here */
#define GPIO_PORT(x, y)           ((PORT##x.PODR.BIT.B##y))
/* End user code. Do not edit comment generated here */
```

The app_main function:

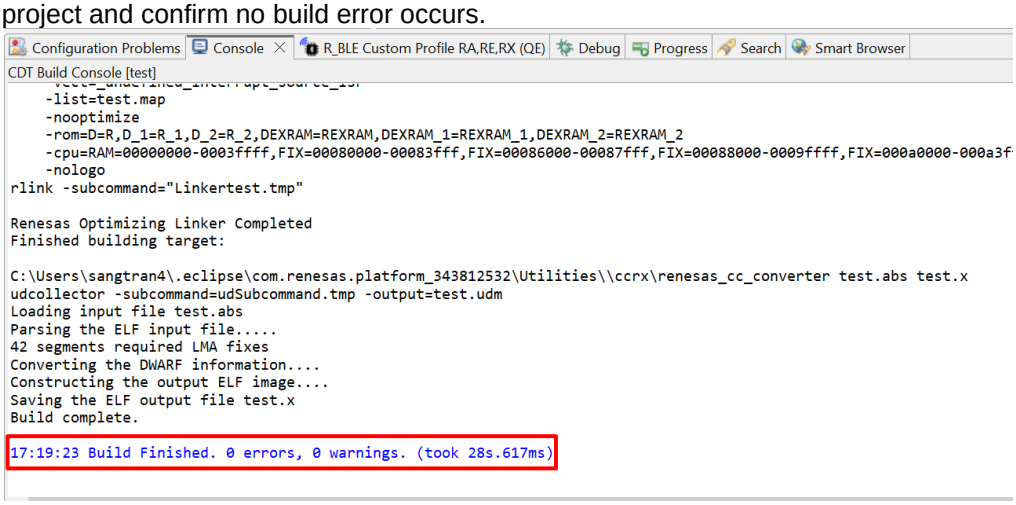
```
void app_main(void)
{
#if (BSP_CFG_RTOS == 2 || BSP_CFG_RTOS_USED == 1)
    /* Create Event Group */
    g_ble_event_group_handle = xEventGroupCreate();
    assert(g_ble_event_group_handle);
#endif

    ble_status_t status;
    fsp_err_t err;

    /* Initialize BLE and profiles */
    if (BLE_SUCCESS == ble_init())
    {
        GPIO_PORT(B, 7) = 1;
        GPIO_PORT(2, 2) = 1;
        GPIO_PORT(2, 5) = 1;
    }
    else
    {
```

```
        GPIO_PORT(2, 5) = 0;
    }

    /* main loop */
    while (1)
    {
        /* Process BLE Event */
        R_BLE_Execute();

/* When this BLE application works on the FreeRTOS */
#if (BSP_CFG_RTOS == 2 || BSP_CFG_RTOS_USED == 1)
        if(0 != R_BLE_IsTaskFree())
        {
            xEventGroupWaitBits(g_ble_event_group_handle,
                                (EventBits_t)BLE_EVENT_PATTERN,
                                pdTRUE,
                                pdFALSE,
                                portMAX_DELAY);
        }
#endif
 }

    /* Terminate BLE */
    RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
}
```

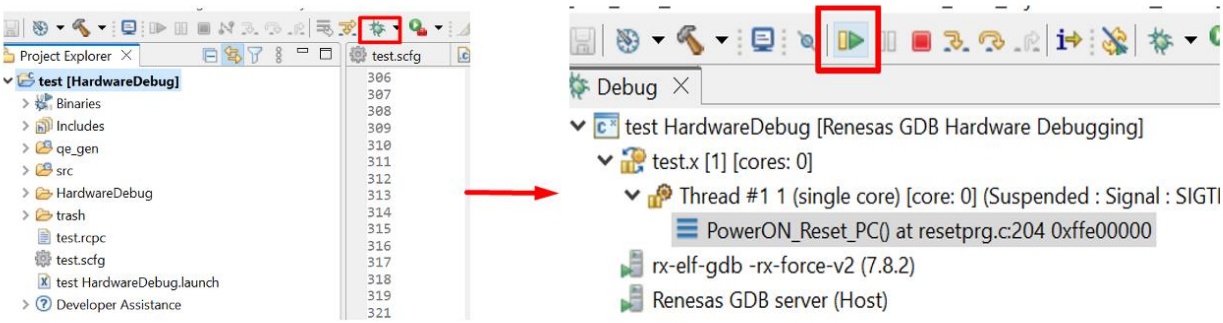
The bleservs_cb function:

```
static void bleservs_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t
*p_data)
{
    uint8_t state;

    switch(type)
    {
        case BLE_BLESERVS_EVENT_LEDCTRL_WRITE_REQ:
            if (BLE_SUCCESS == result)
            {
                state = *(uint8_t *)p_data->p_param;
                if(state == 0x00)
                {
                    GPIO_PORT(2, 2) = 1;
                }
                else
                {
                    GPIO_PORT(2, 2) = 0;
                }
            }
            break;
        case BLE_BLESERVS_EVENT_LEDCTRL_READ_REQ:
            if (BLE_SUCCESS == result)
            {
                state = GPIO_PORT(2, 2);
                R_BLE_BLESERVS_SetLedctrl(&state);
            }
            break;
        default:
            break;
    }
}
```

(9) Build the project and confirm no build error occurs.

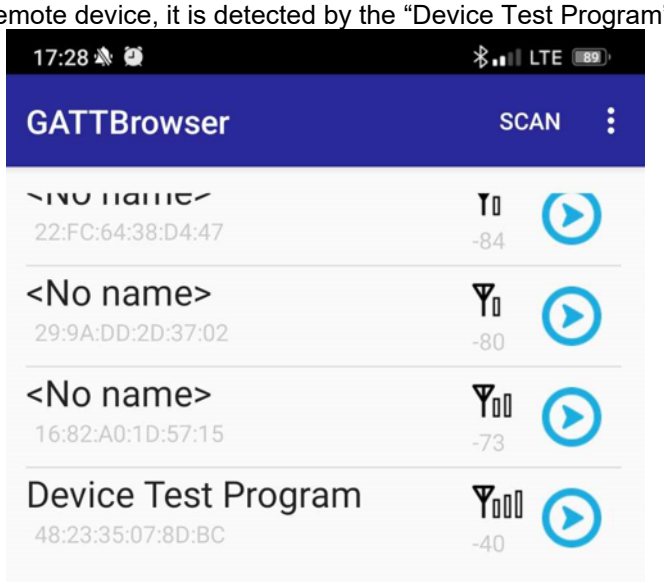


(10) Click the **Launch in Debug Mode** button to write the application to the target board and execute it.

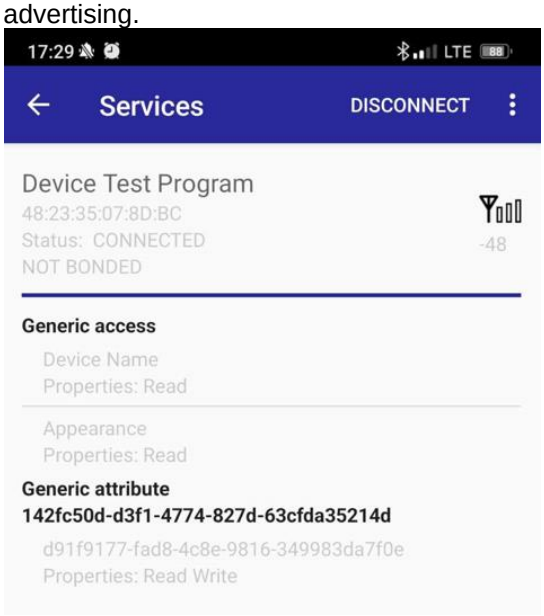


(11) Connect to the application from Renesas GATT Browser.
The GATT Server demo works as below.

- After starting, it starts advertising and waits for a command.
- By scanning from a remote device, it is detected by the “Device Test Program” device name.



- When connected, it stops advertising.

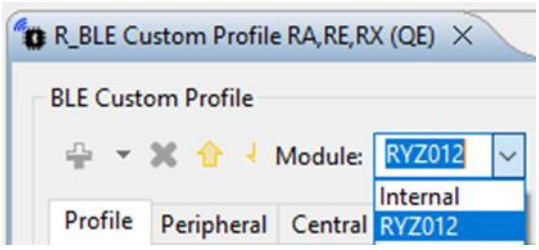


- By writing a number to the LED Control characteristic, the LED turns on by writing the number (0x01~0xFF) to the characteristic. The LED turns off by writing zero to the characteristic.
- When disconnected, it restarts advertising.

6. Appendix

6.1. Limitations

1) The QE tool for BLE (v1.6.0) does not support DA14531 yet, however next version (v1.7.0) will do. Till then, users can select RYZ012 as a work-around.



2) For Boot SDK download from host MCU, developers should be aware of the following limitations when using the BLE_ABS:

Following a power on reset, the R_BLE_VS_GetRand function always returns the same number. Subsequent calls to this function produce random numbers.

Service and characteristic write callback functions, created when using the QE Tool are not supported.

The boot from host feature currently only supports 1-wire UART operation. This means that the UART RX and TX pins on the host RX MCU must be tied together using a 1K ohm resistor in order to boot the DA14531 - this resistor can remain in place after the boot operation has been completed.

6.2. Confirmed Operation Environment

This section describes confirmed operation environment for the FIT module.

Table 6.1 Confirmed Operation Environment (Ver. 1.00)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2023.01
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.05.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian order	Big endian / little endian
Revision of the module	Rev.1.00
Board used	Renesas CK-RX65N Cloud Kit (Product no.: RTK5CK65N0S04000BE)

Table 6.2 Confirmed Operation Environment (Ver. 1.20)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2023.07
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.05.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian order	Big endian / little endian
Revision of the module	Rev.1.20
Board used	Renesas CK-RX65N Cloud Kit (Product no.: RTK5CK65N0S04000BE)

6.3. Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got an error: Could not open-source file "platform.h".

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following document:

For e2 studio, Application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)".

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note "Board Support Package Module Using Firmware Integration Technology (R01AN1685)".

(2) Q: I have added the FIT module to the project and built it. Then I got an error of wrong setting configuration.

A: The setting in the file “r_ble_da14531_config.h” may be wrong. Check the file "r_ble_da14531_config.h". If there is a wrong setting, set the correct value for that. Refer to 2.7 Compile Settings for details.

(3) Q: The pin setting is supposed to be done, but it doesn’t look like that.

A: The pin setting may not be performed correctly. When using this FIT module, the pin setting must be performed. Refer to 2.7 Compile Settings for details.

7. Reference Documents

User's Manual: Hardware
(The latest versions can be downloaded from the Renesas Electronics website.)

Technical Update/Technical News
(The latest information can be downloaded from the Renesas Electronics website.)

User's Manual: Development Tools
RX Family CC-RX Compiler User's Manual (R20UT3248)
(The latest versions can be downloaded from the Renesas Electronics website.)

Revision History

Rev.	Date	Revision History	
		Page	Summary
1.00	Jun. 30, 2023	-	First edition issued
1.10	Sep. 18, 2023	6	Add support AzureRTOS
		7-9	Update Table 1.1 API functions
		11	Update Table 2.1 and Table 2.3
		16	Update data of some parameters
		19-93	Update description of API functions
		94-105	Add Sample Code Generation using QE for BLE
		106	Update Revision of Table 5.1
1.20	Feb. 23, 2024	-	Update document format
		5	Update Figure 1-1 to update the connection with BLE DA14531 module
		6	Update description of RTOS in Software Configuration Section
		7	Add 1.3 Features
		8, 27	Add R_BLE_GetVersion()
		11	Add 1.5 Status Transitions
		12	Add 1.6 Usage Notes
		14	Update Table 2.1
		16	Update Table Memory Usage in 2.8 Code Size
		20-21	Add new parameters about UART boot protocol message types
		96-108	Update 5. Sample Code Generation Using QE BLE
		109	Update 6.1 Limitations
		109	Add Table 6.2

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Contact information

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

Trademarks

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