RX23W Group

Bluetooth Low Energy Profile Developer's Guide

Introduction

This document guides you on how to generate and customize BLE(Bluetooth® Low Energy) profiles for developer using the following target device.

Target Device

- RX23W Group

Related Documents

- Bluetooth Core Specification (https://www.bluetooth.com)
- RX23W Group User's Manual: Hardware (R01UH0823)
- Firmware Integration Technology User's Manual (R01AN1833)
- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723)
- Renesas Smart Configurator User Guide: e^2 studio (R20AN0451)
- RX23W Group BLE Module Firmware Integration Technology(R01AN4860)
- RX23W Group BLE QE Utility Module Firmware Integration Technology(R01AN4907)
- QE for BLE[RX] V1.0.0 Release Note(R20UT4644)

Contents

1. Overview

1.1 Structure of profile

In a BLE Communication, Generic Attribute Protocol (GATT) is primarily used. GATT defines client and server roles, and profile communication is performed between client and server. The server has the profile data in GATT database and the client accesses the profile data by BLE communication.

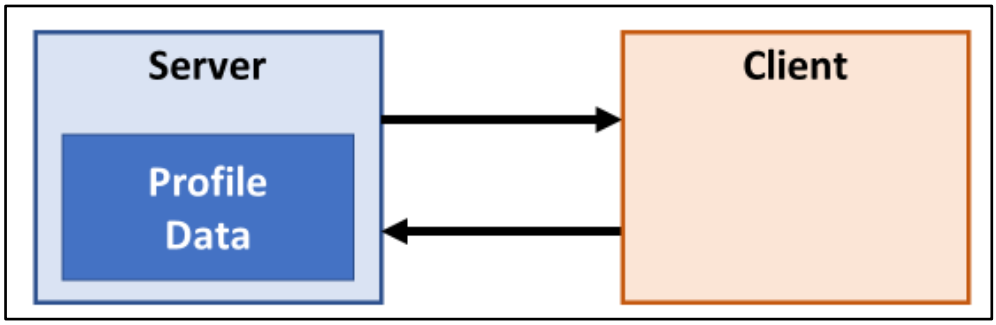

Figure 1.1 Overview of profile communication

Figure 1.2 shows Structure of profile in BLE software.

In this BLE software, user application and profiles run on the BLE Protocol Stack. User application and profile consist of 3 types of program:

- Framework for using BLE features and profiles from user application.
- GATT database that defines the data structure of the services configured in the profile.
- API program for accessing profile data.

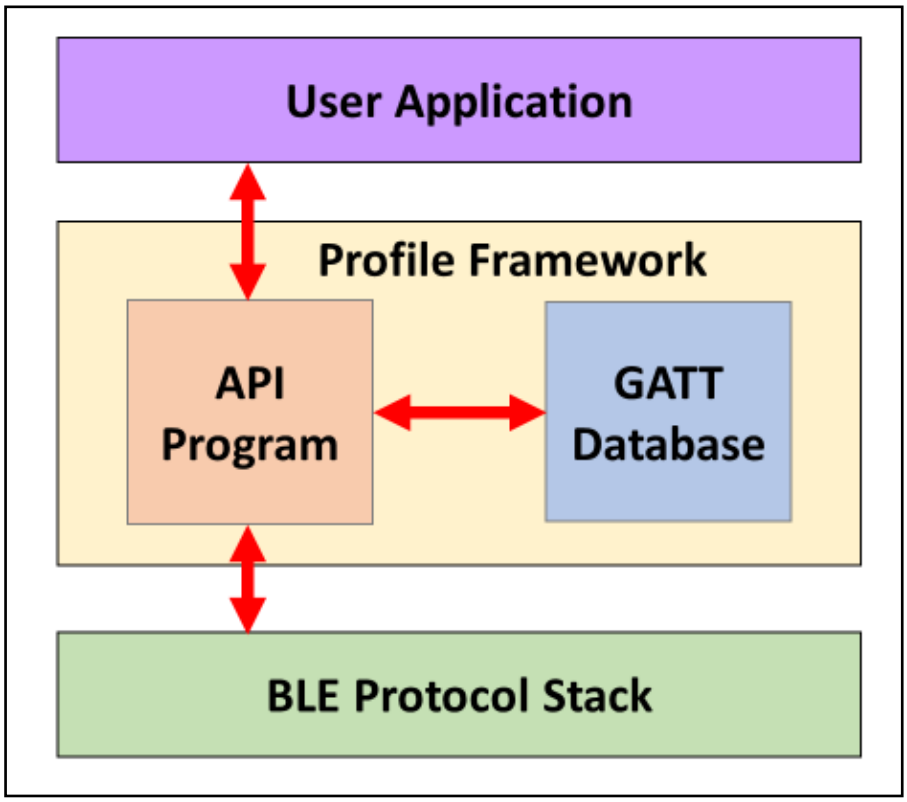

Figure 1.2 Structure of profile

BLE software defines the data structure of the profile as a GATT database and data accessing method as an API program for the service. The user application uses the API program for service to access the profile data to perform BLE profile communication.

Bluetooth SIG Inc. defines specification of several services. In this document, those services are referred as SIG adopted services. On the other hand, if you want to achieve functionality that is not supported by SIG adopted service, you must define your own service. In this document, these are referred as custom service.

1.2 Flow of profile development

Figure 1.3 shows flow of profile development.

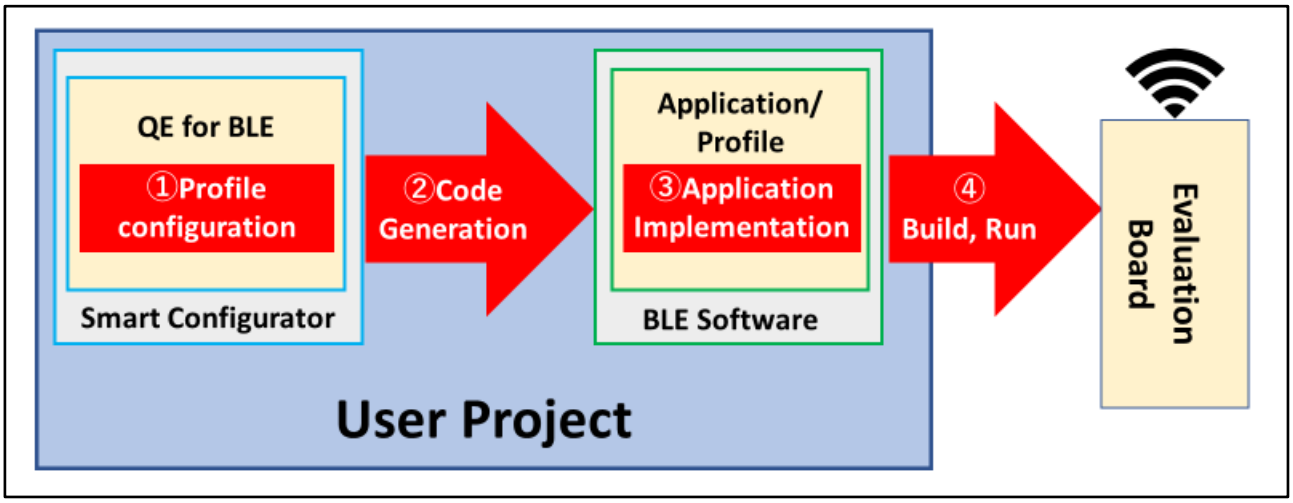

Figure 1.3 Flow of profile development

The steps for profile development are as follows:

1) Configure profile with QE for BLE

Configure profile by GUI provided from QE for BLE. Check [3 Profile Configuration in QE for BLE] for more information.

2)Generate profile from Smart Configurator

Smart Configurator generates API program of profile configured on QE for BLE, GATT database, application framework on user project created over user project. BLE Protocol Stack, which provides basic BLE functionality, can also be generated from Smart Configurator if BLE FIT Module is added to it.

3) Implement application into generated program

You can implement the application on user project using program generated from Smart Configurator. For more information about implementing user application using generated program, refer [4 Implementation of program].

4)Build and Run implemented application

Implemented application can be built and run on evaluation board. For running application and profile, BLE protocol Stack is needed.

2. Development environment

2.1 BLE QE Utility Module

BLE QE Utility Module is used by QE for BLE to generate SIG adopted service programs. Table 2.1 shows the list of SIG adopted service that are supported by BLE QE Utility Module. Specifications of each service is defined by Bluetooth SIG. Check Web page of Bluetooth SIG (https://www.bluetooth.com) for more information.

Table 2.1 SIG adopted service supported by BLE QE Utility Module

Service name	abbreviation	version	service name	abbreviation	version
Alert Notification Service	AN	1.0	Automation IO Service	AIO	1.0
Battery Service	BA	1.0	Blood Pressure Service	BL	1.0
Body Composition Service	BC	1.0	Bond Management Service	BM	1.0
Continuous Glucose Monitoring Service	CGM	1.0.1	Current Time Service	CT	1.1
Cycling Power Service	CP	1.1	Cycling Speed and Cadence Service	CSC	1.0
Device Information Service	DI	1.1	Environmental Sensing Service	ES	1.0
Fitness Machine Service	FTM	1.0	Glucose Service	GL	1.0
Health Thermometer Service	HT	1.0	Heart Rate Service	HR	1.0
Human Interface Device Service	HID	1.0	Immediate Alert Service	IA	1.0
Insulin Delivery Service	ID	1.0	Link Loss Service	LL	1.0.1
Location and Navigation Service	LN	1.0	Next DST Change Service	NDC	1.0
Object Transfer Service	OT	1.0	Phone Alert Status Service	PAS	1.0
Pulse Oximeter Service	PLX	1.0	Reconnection Configuration Service	RC	1.0
Reference Time Update Service	RTU	1.0	Running Speed and Cadence Service	RSC	1.0
Scan Parameters Service	SCP	1.0	Tx Power Service	TP	1.0
User Data Service	UD	1.0	Weight Scale Service	WS	1.0
GATT Service			GAP Service		

Note: Object Transfer Service is not authenticated. Please contact us when considering this service to be used in your product.

2.2 QE for BLE

QE for BLE provides GUI to configure profile and generates program. Table 2.2 lists the program that QE for BLE generates.

Table 2.2 Programs generated by QE for BLE

file name	description
app_main.c	Application/Profile framework Skeleton program that is the basis of application/profile development.
gatt_db.c gatt_db.h	GATT database program Data structure of service which is checked on [server] in QE for BLE is defined.
r_ble_[abbreviation][s or c].c r_ble_[abbreviation][s or c].h	Profile API program API program for accessing profile data categorized by service. Each file name is determined based on the [abbreviation], [server], and [client] set in QE for BLE. [abbreviation][s] is the server program, [abbreviation][c] is the client program. Example) [abbreviation]=[sig], [server] : r_ble_sigs.c, r_ble_sigs.h [abbreviation]=[cus], [client] : r_ble_cusc.c, r_ble_cusc.h

All program generated by QE for BLE is output in [Project Name]/src/smc_gen/Config_BLE_PROFILE.

Skeleton Programs are basis for profile and application development. Implementation of skeleton program is described in [4 Implementation of program].

2.3 Building development environment

2.3.1 Install QE for BLE

QE for BLE can be downloaded from the web page below.

https://www.renesas.com/qe-ble

For more information about installation, refer to the document below.

- QE for BLE[RX] V1.0.0 Release Note(R20UT4644)

2.3.2 Download FIT module

After downloading the FIT modules required for profile development, you can use them without installation by placing them to appropriate folder. You can also download directly to the appropriate folder by downloading them from e^2 studio. This section describes how to download the FIT module using e^2 studio.

For using QE for BLE, FIT module shown below is needed for using QE for BLE.

- RX23W Group BLE QE Utility Module Firmware Integration Technology(R01AN4907)

Also, FIT modules shown below is needed for construction of BLE Protocol Stack which is needed for running BLE communication.

- RX23W Group BLE Module Firmware Integration Technology(R01AN4860)

Click the tab at the bottom of the Smart Configurator on e² studio to open the [Components] tab. On the [Components] tab, click [Add module]. To use the Smart Configurator, you must create a new project. For more information about creating a new project, refer [Renesas Smart Configurator User Guide: e2 studio (R20AN0451)]

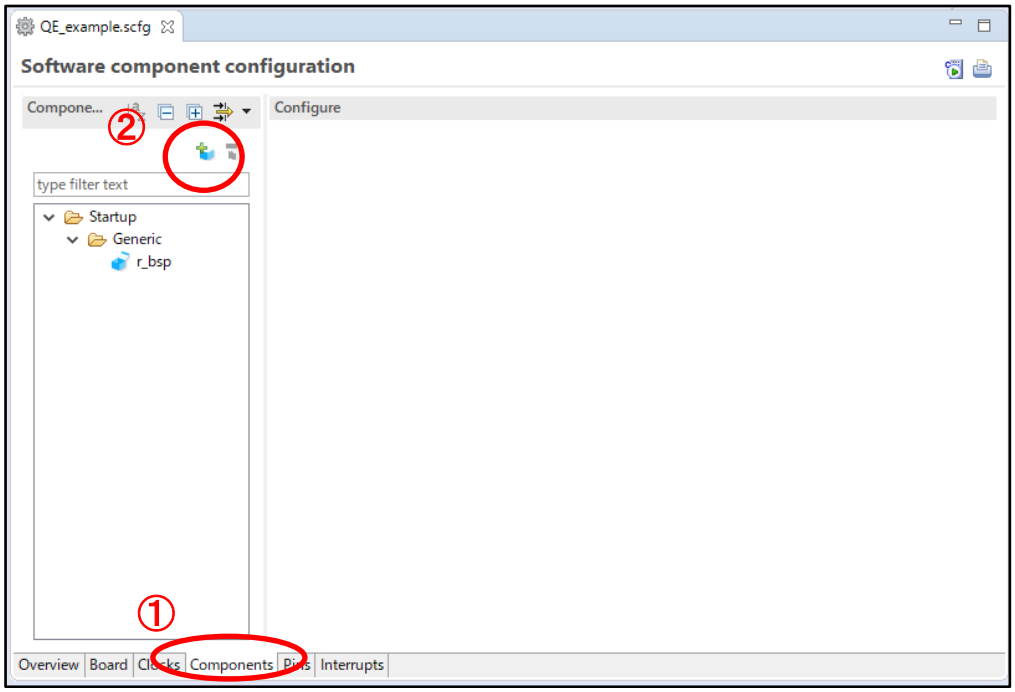

Figure 2.1 Add Module

Before downloading FIT module, folder for downloading FIT module should be specified. Click [Configure general setting] and open Preference window.

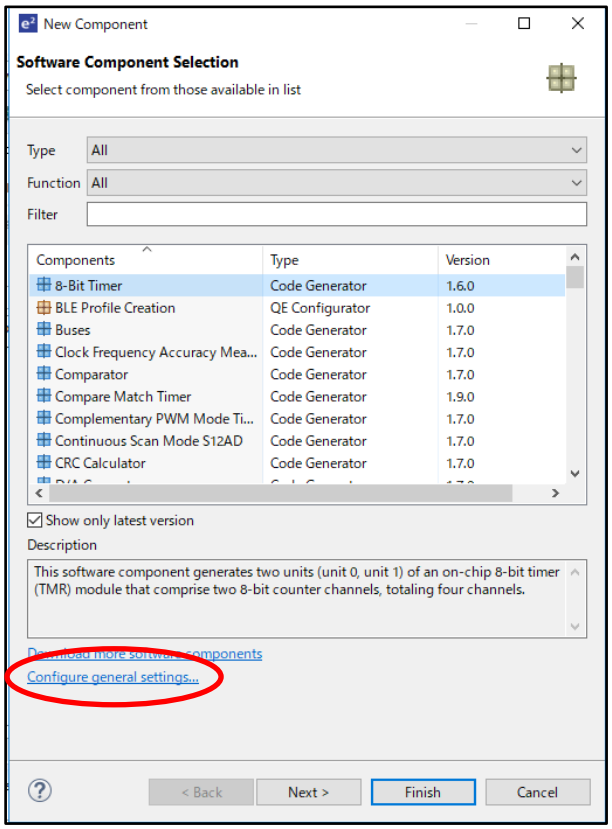

Figure 2.2 Click configure general setting

Click the [Browse] button in Location(RX) to specify folder. Verify that the path to specified folder is displayed. After verifying that the correct folder path is displayed, click [Apply and Close] button to close Preference Window.

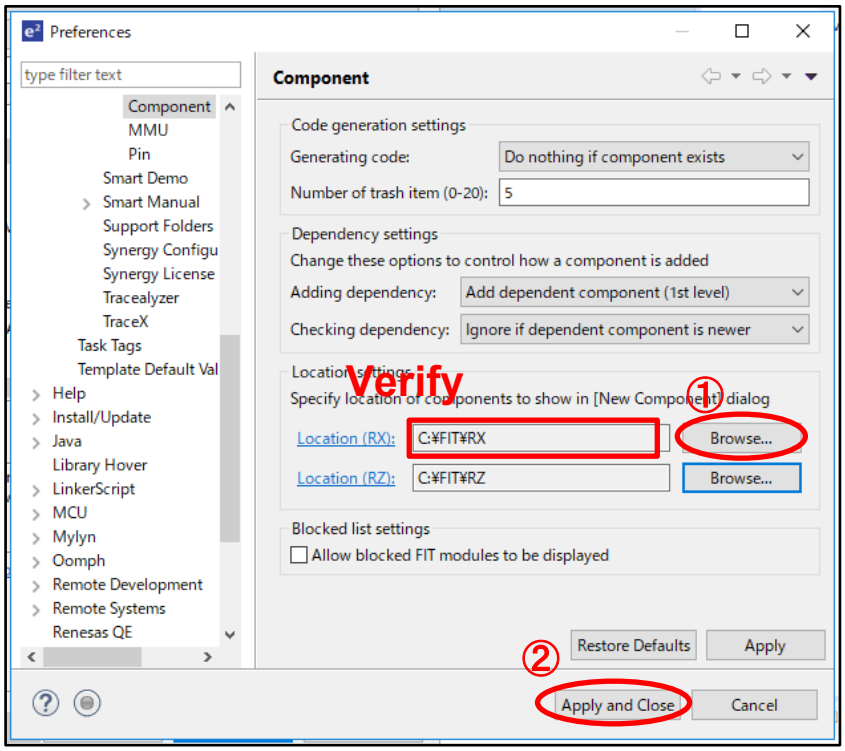

Figure 2.3 Preference window

Click [Download more software components] on New Component window

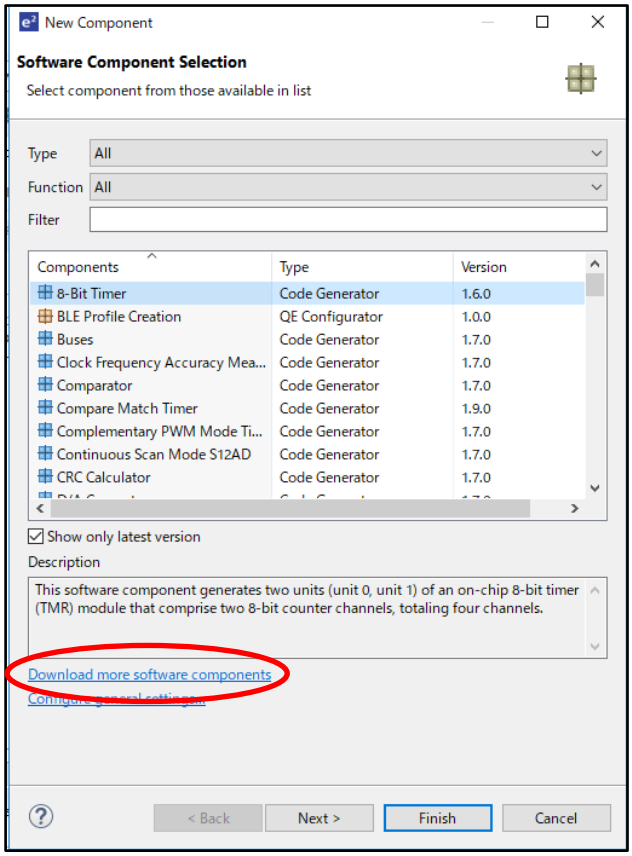

Figure 2.4 Click Download more software components

When My Renesas login screen appears, enter the email address and password registered to My Renesas. After that, click [OK] button.

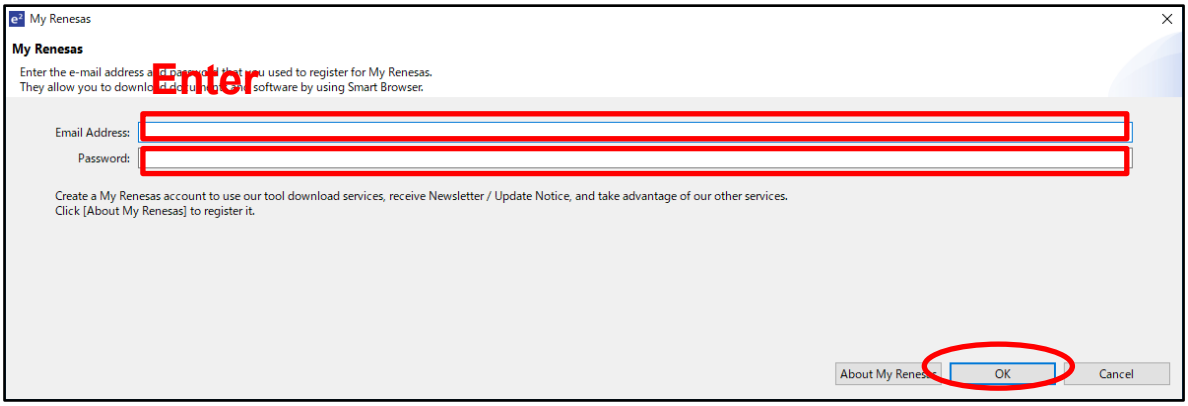

Figure 2.5 My Renesas login window

If region selection screen appears, select your region. Click [OK] button to close the window.

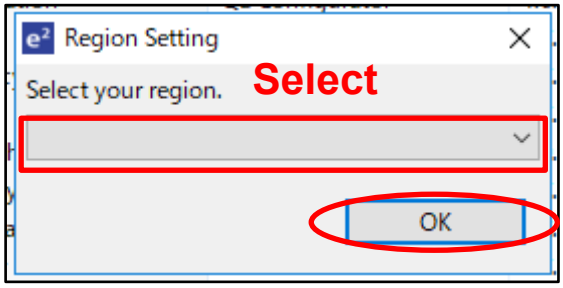

Figure 2.6 Region Setting window

Select the required FIT module on the FIT selection screen. Check the box for all FIT modules you want to download. Next make sure that the path of the Module Folder Path is the path specified on the Preference window. If it is incorrect, click [Browse] button to specify the correct folder path. After you have done all of this, click [Download] button.

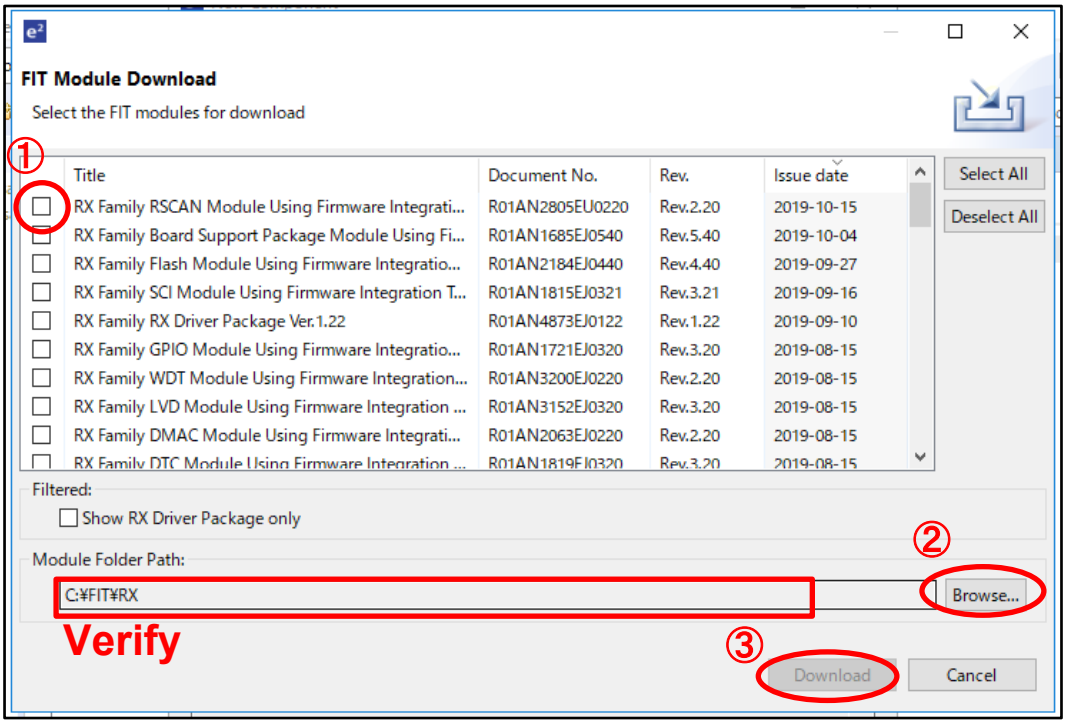

Figure 2.7 FIT Module Download window

3. Profile Configuration in QE for BLE

This section guides how to configure profiles in QE for BLE.

3.1 Overview of profile configuration

To configure your profile, you must follow these steps.

①Create a new project that uses Smart Configurator.
②Add FIT modules to Smart Configurator.
③Add QE for BLE to Smart Configurator.
④Configure Profile using QE for BLE.
⑤Generate code based on configured profile.
Each step of configuration is described here.

3.1.1 Create new project

To create RX23W Project that QE for BLE operate, you need to create project which has Smart Configurator. To create project which has Smart Configurator, refer to application note [Renesas Smart Configurator User Guide: e2 studio (R20AN0451)] and [BLE Fit Module application note(R01AN4860]]. How to create RX23W project and information about Smart Configurator in the e^2 studio are described these application note.

3.1.2 Addition of BLE QE Utility module

BLE QE Utility module provide feature which generate code to QE for BLE.QE for BLE presume BLE QE Utility module when it is installed. Therefore, BLE QE Utility module should be added to component before QE for BLE is added. The following describe procedure for adding BLE QE Utility module to component.

Launch Smart Configurator

Open ${project_name}.scfg file in the project explorer.

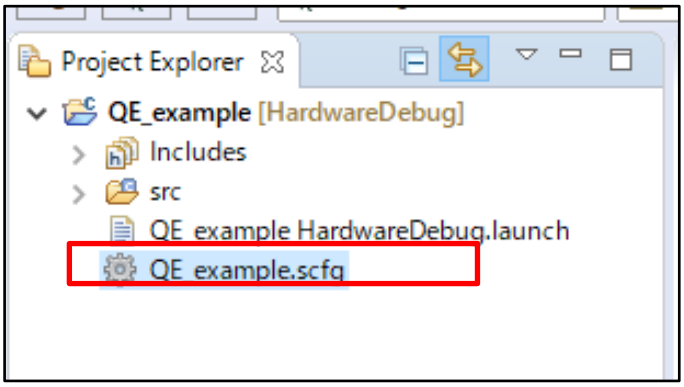

Figure 3.1 Launch Smart Configurator

Add to component

Open [component] tab. Tab is located in the bottom of smart configurator window.

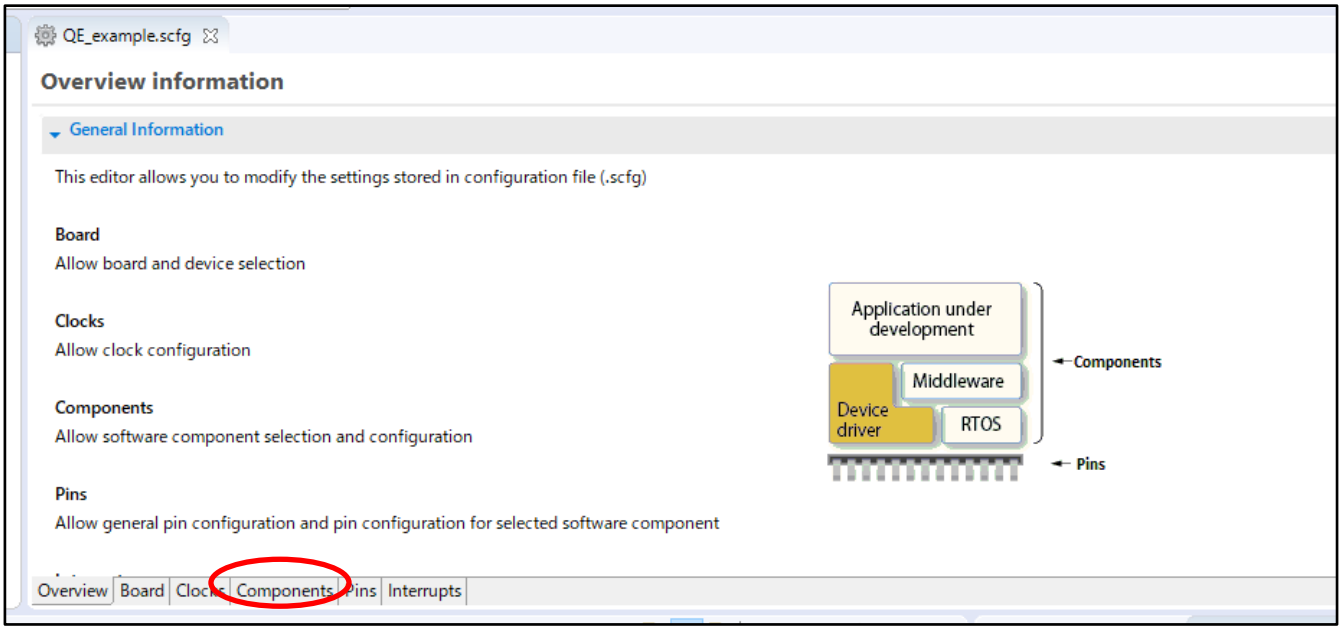

Figure 3.2 The place located [Component] tab

Click green plus button in component window and [Software Component Selection] window open.

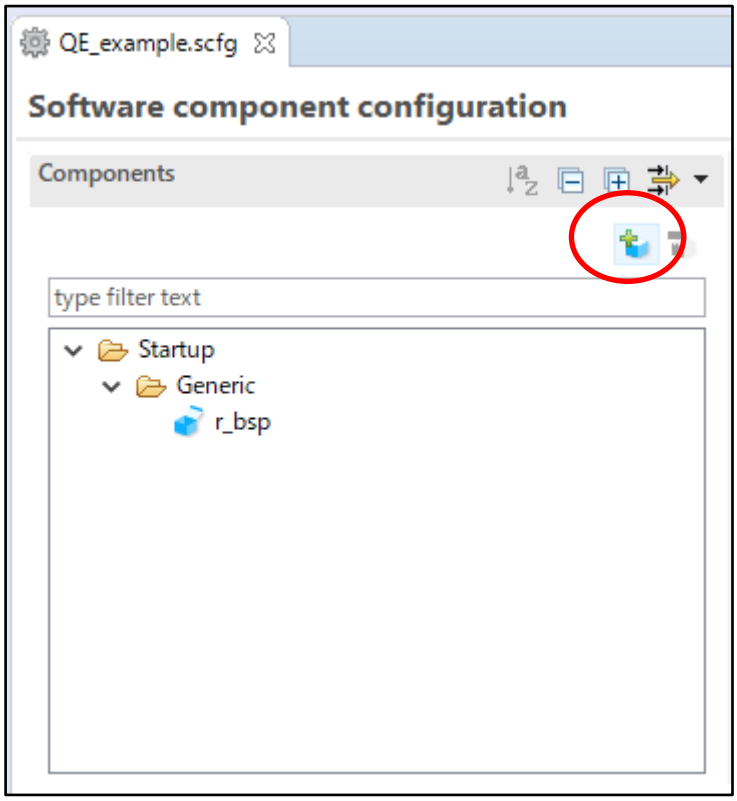

Figure 3.3 Add button for component

BLE QE Utility Module is displayed as [r_ble_qe_utility] at component lists. Select [r_ble_qe_utility] and Click [Finish Button] to add module.

Note: If the [r_ble_qe_utility] is not displayed, you need to download BLE QE Utility module referring to [2.3 Building development environment].

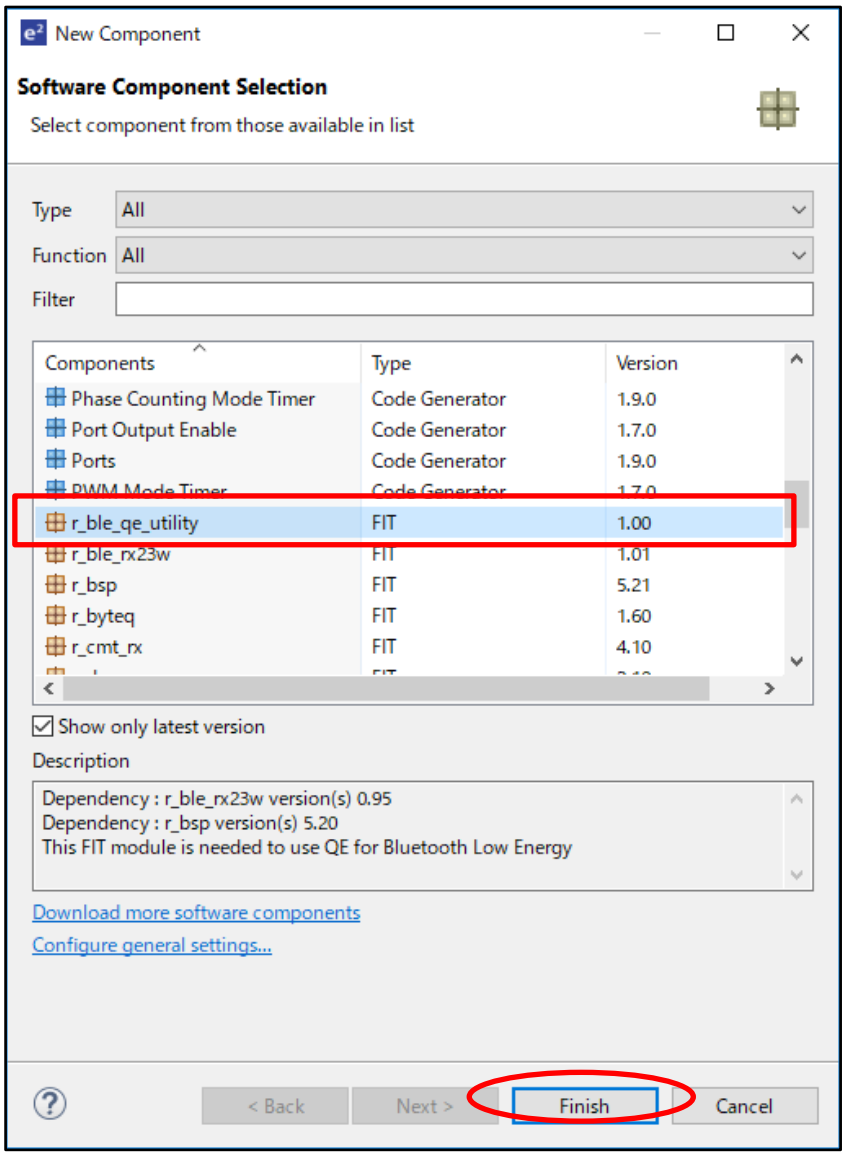

Figure 3.4 Window to add BLE QE Utility

Confirm component

Confirm that [r_ble_qe_utility] is added to Middleware - Generic folder in Component window. BLE QE Utility module don't have the configure items.

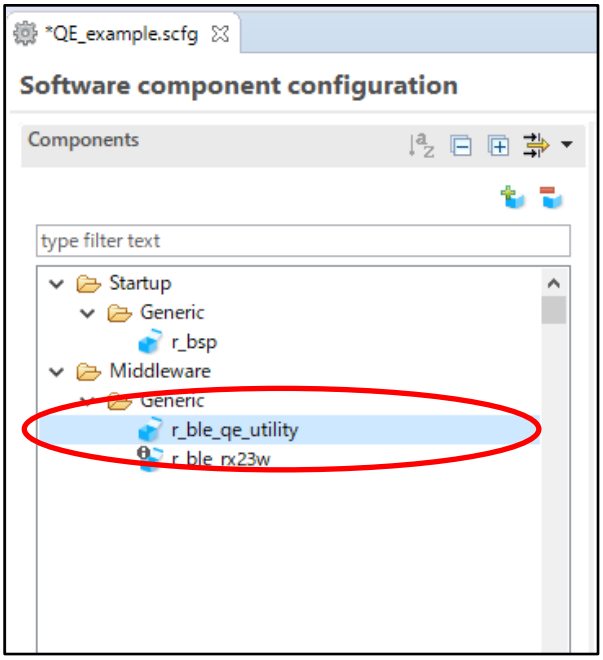

Figure 3.5 Confirm BLE QE Utility module

3.1.3 Addition of QE for BLE

QE for BLE provide GUI for developing profile on the Smart Configurator. BLE QE Utility Module is necessary for QE for BLE to be used. Therefore, QE for BLE should be added to component after BLE QE Utility Module is added.

Add to component

Click green plus button in component window and [Software Component Selection] window open. QE for BLE is displayed as [BLE Profile Creation] at component lists. Select [BLE Profile Creation] and Click [Finish Button] to add module.

Note: If the [BLE Profile Creation] is not displayed, you need to download QE for BLE referring to [2.3 Building development environment].

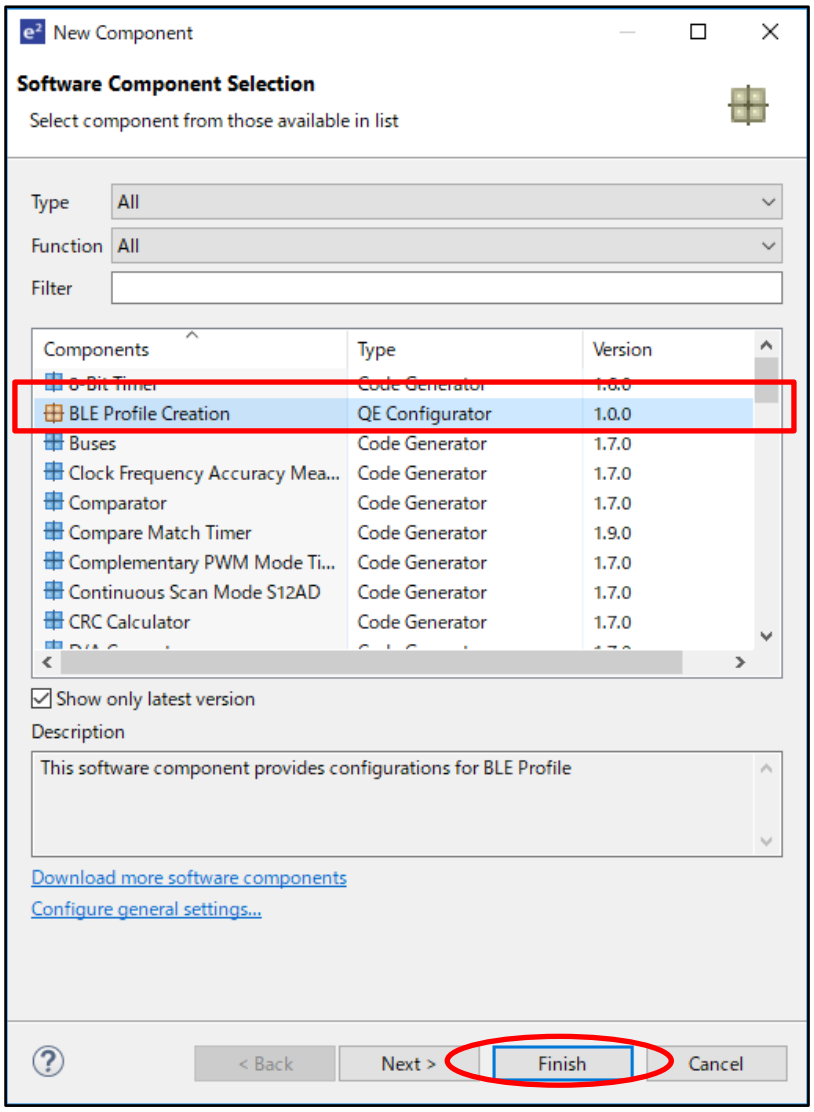

Figure 3.6 Window to add QE for BLE

Confirm component

Confirm that [Config_BLE_PROFILE] is added to Application - Communication folder in Component window. Confirm that BLE custom profile configuration screen is displayed when you select [Config_BLE_PROFILE]. In custom profile configuration screen, confirm that [GAP service] and [GATT service] is added. If these contents are not added, [r_ble_qe_utility] module might not be added to Smart Configurator.

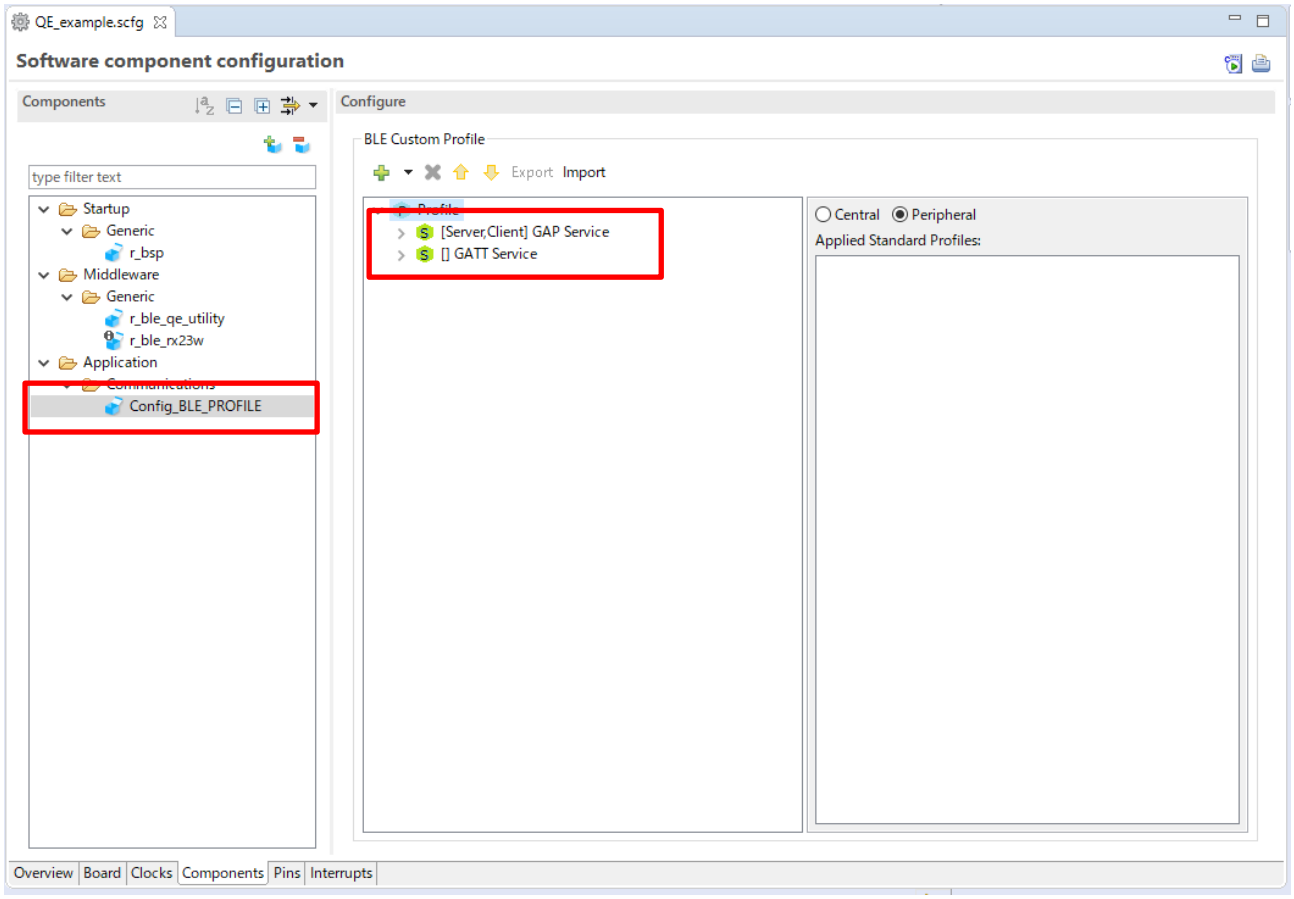

Figure 3.7 Confirm BLE QE Utility module

3.1.4 Addition of BLE FIT module

By adding BLE FIT module to Smart Configurator in the created project, you can develop not only profile but also BLE application by using created project. How to add BLE FIT module to smart configurator in a project is described at Application Note [BLE FIT Module APN(R01AN4860)].

3.1.5 Configuration of profile

You can configure profile using custom profile configuration screen on QE for BLE. For detailed description of the custom profile configuration screen, refer [3.2How to use QE for BLE].

3.1.6 Code generation

After profile configuration using QE for BLE, you can generate program by clicking [Generate Code] button in Smart Configurator.

Figure 3.8 Generate Code button

Generated program can be divided into following 4 types:

- API program generated from SIG standard service.
- API program generated from custom service.
- Application framework
- GATT database

User application will be created using these generated programs. Refer [4.1Implementation of custom service] for modifying API program of custom service, and refer [4.2Implementation of app_main.c] for modifying application framework.

Figure 3.98 shows an example of files generated from QE for BLE.

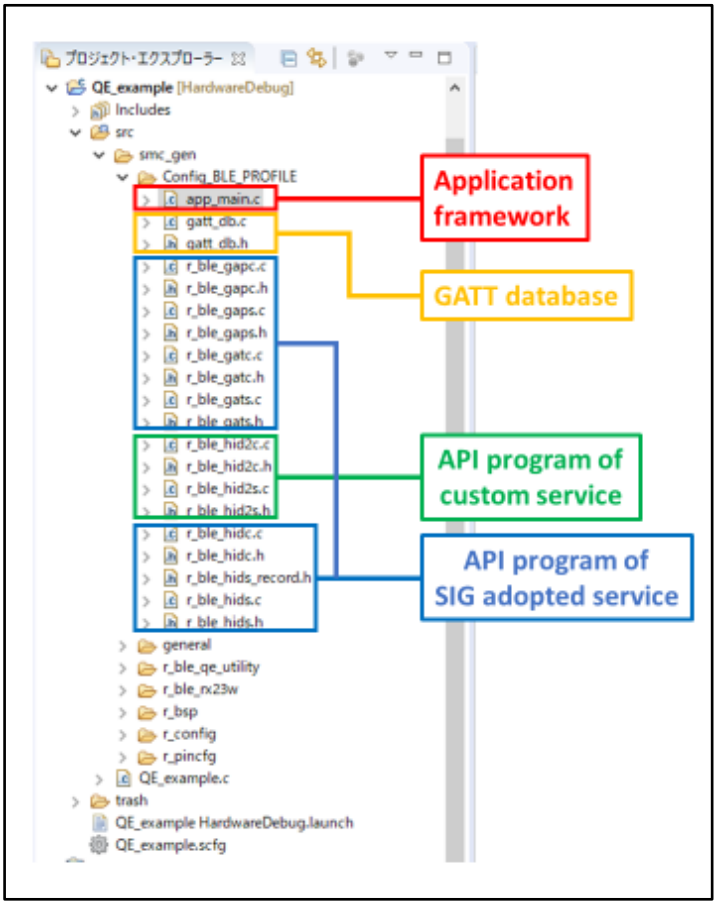

Figure 3.9 Files generated from QE for BLE

3.2 How to use QE for BLE

You can configure profile using QE for BLE. Figure 3.10 shows configuration screen of QE for BLE.

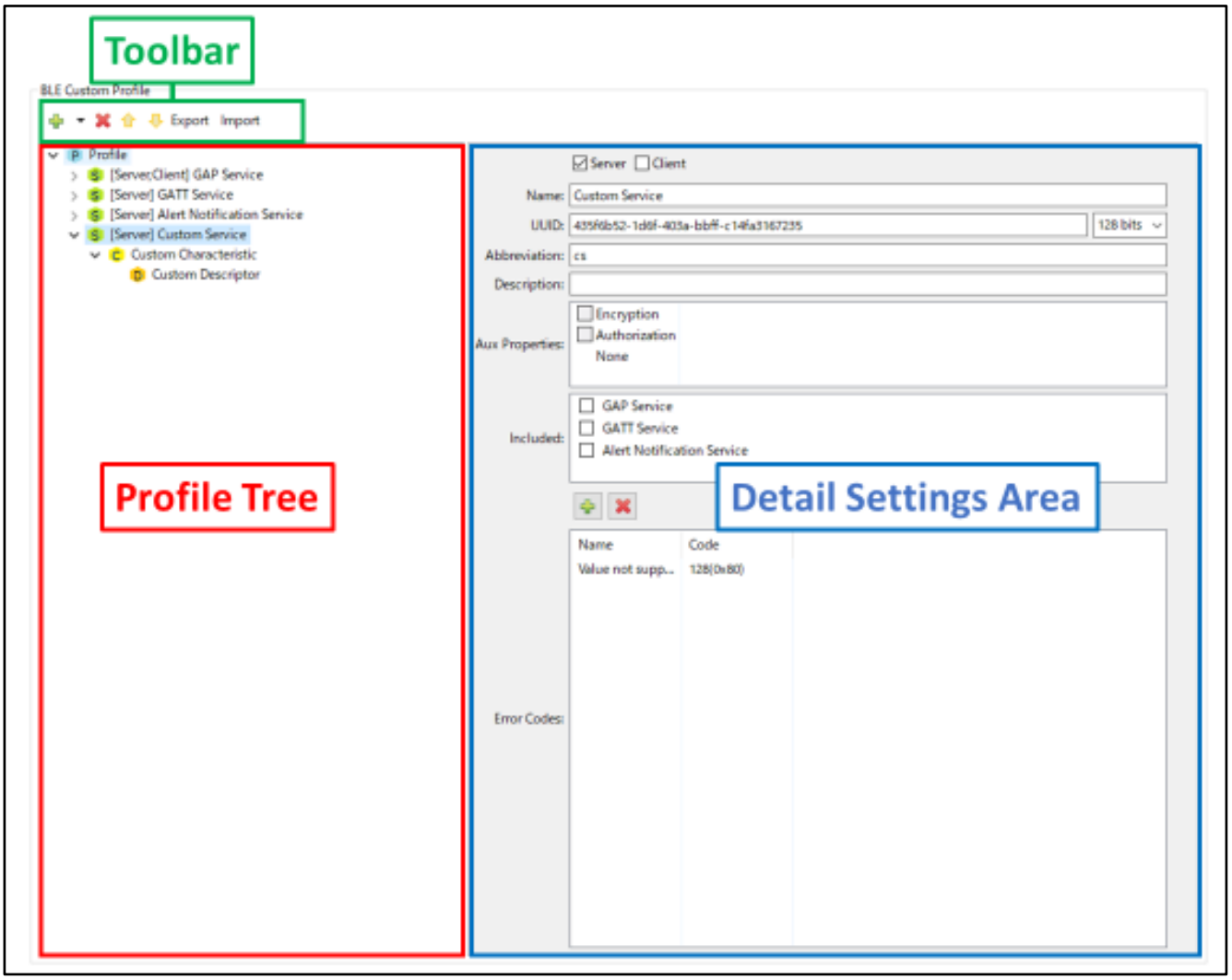

Figure 3.10 QE for BLE configuration screen

You can check the configuration of the profile that you are currently designing from [Profile Tree].

When you select each content of [Profile Tree], the settings are displayed for each type of contents selected in [Detail Setting Area]. You can configure the functionality of contents added to [Profile Tree] by editing the items displayed in [Detail Setting Area].

[Toolbar] is used when you want to add or delete contents from [Profile Tree]. The icons on [Toolbar] and their behavior are as follows:

- [ ] : Adds an contents to [Profile Tree]. The contents added depends on the contents selected in [Profile Tree]
- [ ] : Deletes selected contents in [Profile Tree].
- [ ][ ] : Moves selected contents in [Profile Tree]. Use this to rearrange contents in [Profile Tree].
- [Export] : Outputs the configured service as JSON file.
- [Import] : Loads service defined JSON file and adds it to the profile.

3.2.1 Configuration of profile

When you select profile [ ] in [Profile Tree], profile configuration screen (Figure 3.11) will be shown in [Detail Settings Screen].

You can select GAP role on profile configuration screen. Use the radio button to choose whether to set profile to [Central] or [Peripheral]. Application framework is generated depending on this item. If you select [Peripheral], program which can advertise is generated. If you select [Central], program which can scan and issue connection request is generated.

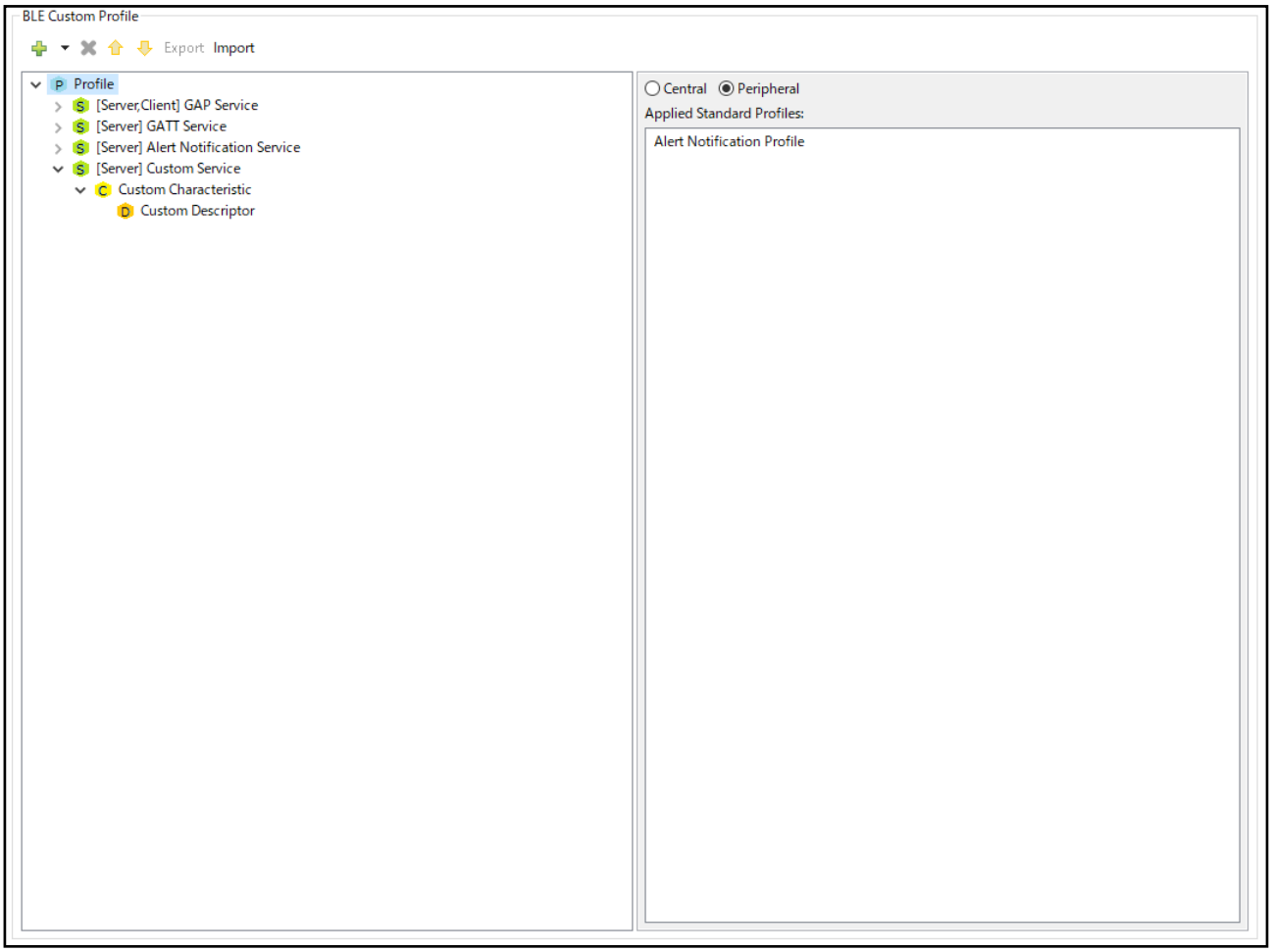

Figure 3.11 Profile configuration screen

You can add services by clicking [ ] button with the profile selected in [Profile Tree] (Figure 3.12).

Select [New service] to add custom service or [Add service] to add a SIG adopted service. If you select [Add profile], you can select the profile that are defined in Bluetooth Specification. When you select the Bluetooth defined profile, all services consisting that profile are added to the configuring profile. Also name of selected profile will be added to [Applied Standard Profile]. In [Applied Standard Service], profile name will also be listed for services added independently.

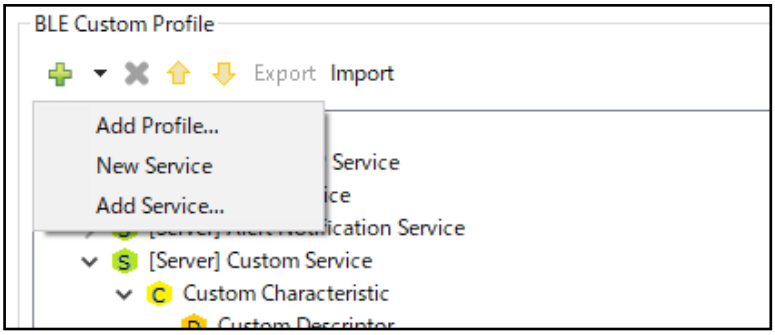

Figure 3.12 Adding service

3.2.2 Configuration of service

When you select service [ ] in [Profile Tree], service configuration screen(Figure 3.13) will be shown in [Detail Settings Screen]. Table 3.1 describes each item on the configuration screen.

Note: The GAP service and GATT service are mandatory services. Do not delete these services.

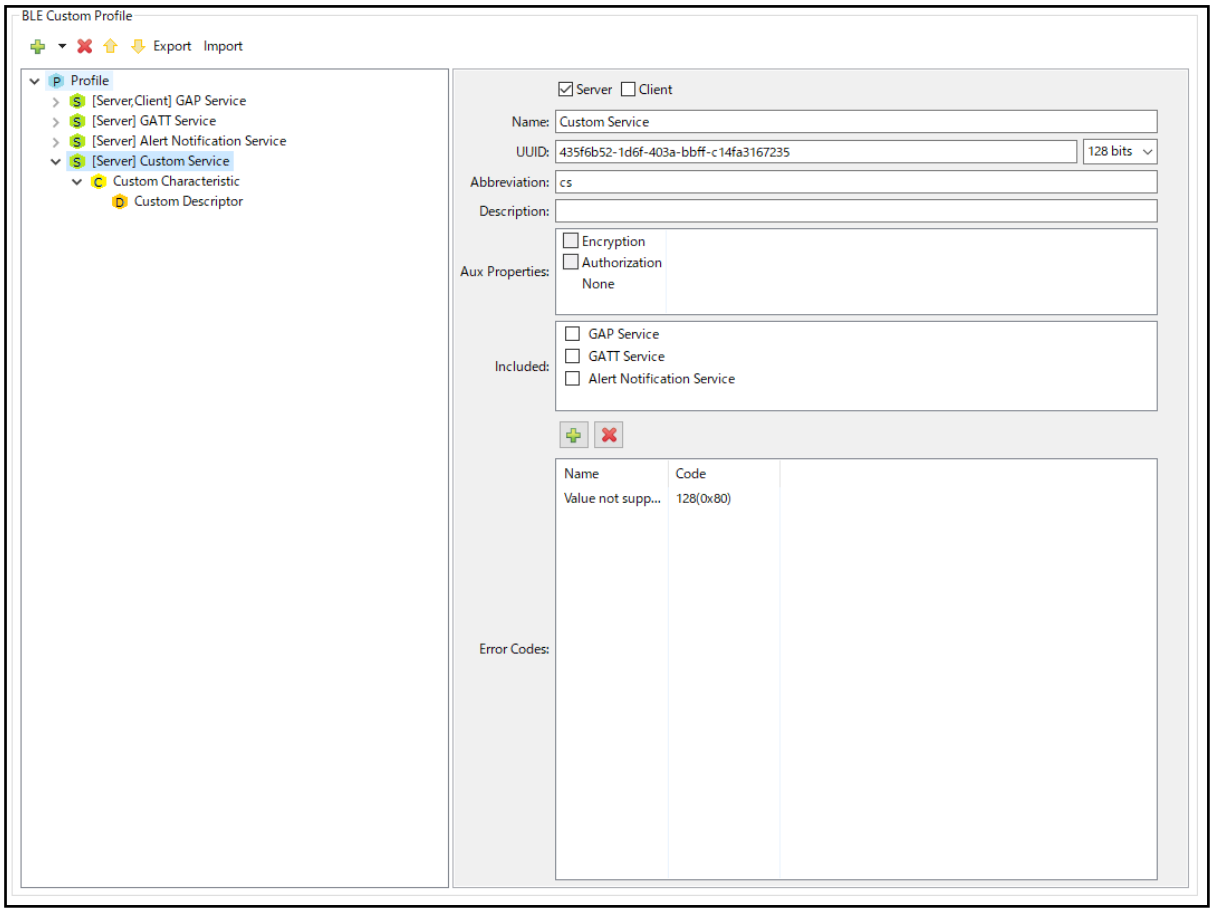

Figure 3.13 Service configuration screen

Table 3.1 Service configuration

Item	Description	
Server	Set check on this item to generate service program as server. It also adds characteristic and descriptors to GATT database.	
Client	Set check on this item to generate service program as client.	
Name	Name of service. Example) Custom service	
UUID	UUID of service. Select 128bit if service is custom service. Initial value is entered randomly. Please modify if needed. Example) 16bit : 0xe237 128bit : 96FE7990-2C76-89AB-DC49-AB7F123DEF9C	
Abbreviation	Abbreviation of service. This value is used in file name, function name and variable name. Beware not to conflict with other services. Example) cs	
Description	Description of service. Explain usage if needed. This description will be used as comments in generated program. Example) This service used for sending sensor data.	
Aux properties	AUX properties of service. Items below can be configured.	
	Encryption	Enable encryption.
	Authorization	Enable authorization. Use function R_BLE_GAP_AuthorizeDev( ) to authorize.
	Select appropriate security requirement stage from the following item	
	None	There are no requirements about security for accessing service.
	Unauthentication	Requires pairing for accessing service.
	Authentication	Requires pairing with MITM for accessing service.
	Secure Connection	Requires pairing on secure connection for accessing service.
Included	Sets Included service. Select the service to be included from the list.	
Error Codes	Adds error code of service. Error code added can be used by function R_BLE_GATTS_SendErrRsp( ).	
	Name	Name of error code. Example) Value not Supported
	Code	Value of error code. Select from value list.

Figure 3.14 shows the service configuration screen for SIG adopted service. In this state, only [server], [client], and [Included] items can be configured. You can edit all items by clicking the [Customize] button. Please use it in case creating a custom service based on SIG adopted service.

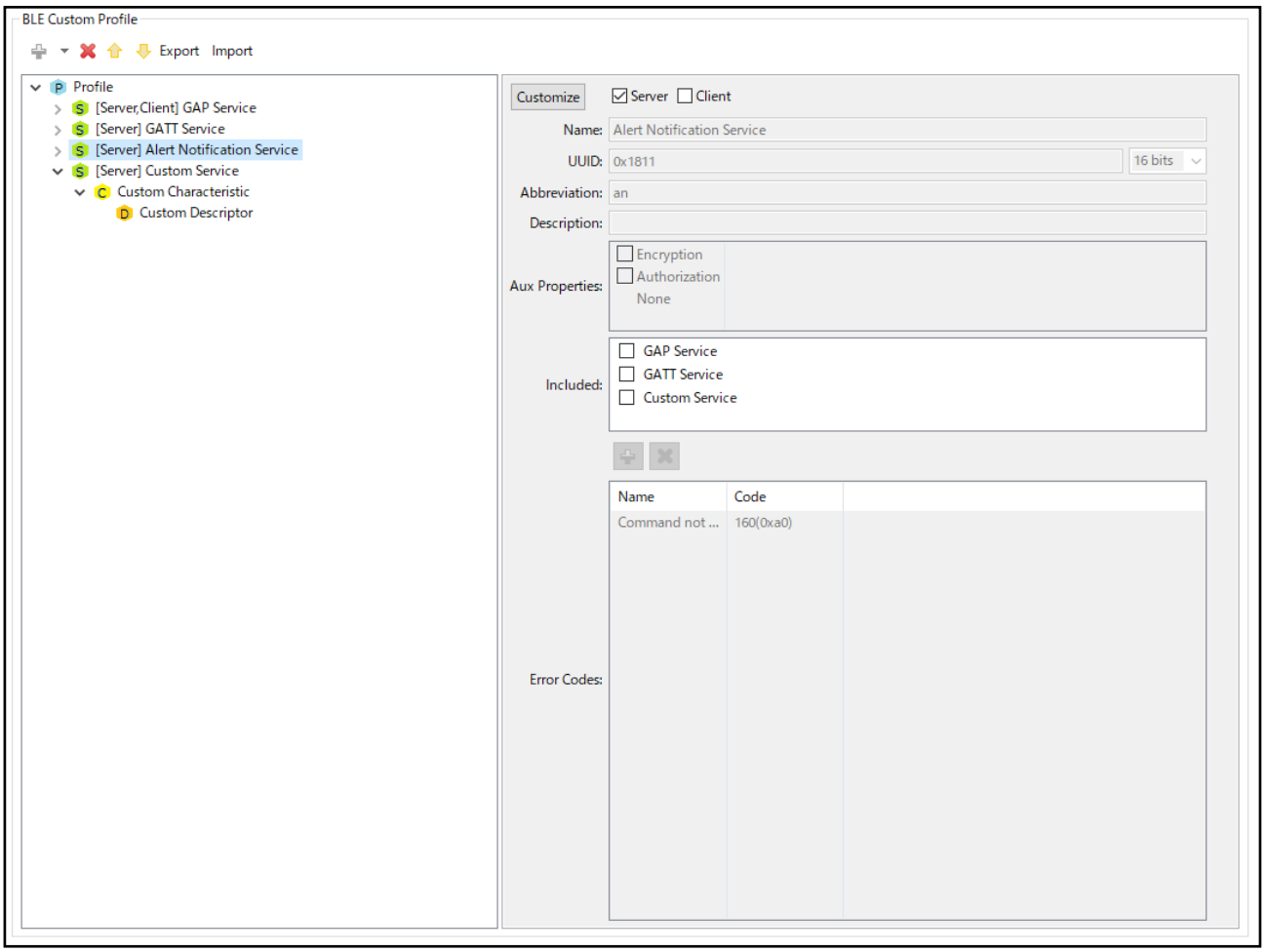

Figure 3.14 SIG adopted service configuration screen

You can add a characteristic by clicking [ ] button with the service selected. Select [New Characteristic] to add a custom characteristic or [Add Characteristic] to add SIG adopted characteristic.

3.2.3 Configuration of characteristic

When you select characteristic [C] in [Profile Tree], characteristic configuration screen (Figure 3.15) will be shown in [Detail Settings Screen]. Table 3.2 and Table 3.3 describes each item on the configuration screen.

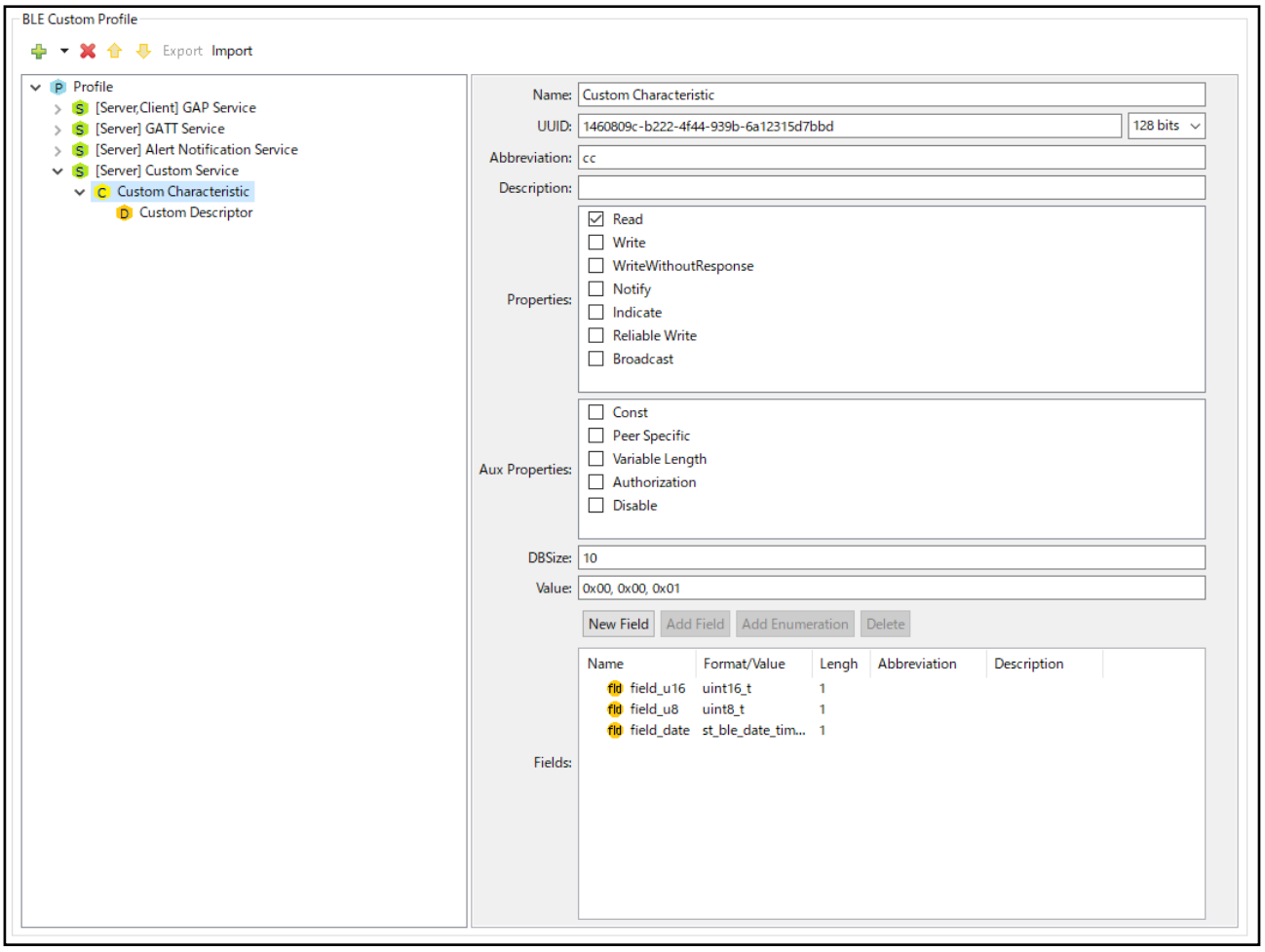

Figure 3.15 Characteristic configuration screen

You can add a descriptor by clicking [+] button with the characteristic selected. Select [New Descriptor] to add a custom descriptor or [Add Descriptor] to add SIG adopted descriptor.

Table 3.2 Characteristic configuration

Item	Description	
Name	Name of characteristic. Example) Custom Characteristic	
UUID	UUID of characteristic. Select 128bit if service is custom characteristic. Initial value is entered randomly. Please modify if needed. Example) 16bit : 0xe237 128bit : 96FE7990-2C76-89AB-DC49-AB7F123DEF9C	
Abbreviation	Abbreviation of characteristic. This value is used in function name and variable name. Beware not to conflict with other characteristics. Example) cc	
Description	Description of Characteristic. Explain usage if needed. This description will be used as comment of generated program. Example) This Characteristic is used for sending sensor data	
Properties	Properties of characteristic. Items below can be configured.	
	Read	Enable Read operation.
	Write	Enable Write operation.
	WriteWithoutResponse	Enable WriteWithoutResponse operation.
	Notify	Enable Notify operation.
	Indicate	Enable Indicate operation.
	ReliableWrite	Enable ReliableWrite operation.
	Broadcast	Enable Broadcast operation.
Aux Properties	AUX properties of characteristic. Items below can be configured.	
	Const	Value will not be able to change.
	Peer Specific	Value will be kept individually for each connection.
	Variable Length	Value length will be variable.
	Authorization	Enable authorization. Use function R_BLE_GAP_AuthorizeDev( ) to authorize.
	Disable	Disable attribute.
DBSize	Size of characteristic. Unit of value is byte. Example) 5	
Value	Initial value of characteristic. If you want to enter a number, enter it separated by 8bit digit. If you want to enter string, you can easily enter it by enclosing it in “”. Example) For numbers: 0x12, 0x34, 56,78 For string: “example”	
Field	Set value field used in application. Please refer Table 3.3 for configuration.	

Table 3.3 Characteristic Field configuration

New Field	Add new field. Items below can be configured		
	Name	Name of field. Example) field_name	
	Format/Value	Format of field. Value can be selected from below.	
		bool	Boolean type
		char	char type
		uint8_t	unsigned 8bit data type
		uint16_t	unsigned 16bit data type
		uint32_t	unsigned 32bit data type
		int8_t	signed 8bit data type
		int16_t	signed 16bit data type
		int32_t	signed 32bit data type
		st_ble_ieee_11073_float_t	IEEE-11073 32bit FLOAT type
		st_ble_ieee_11073_sfloat_t	IEEE-11073 16bit SFLOAT type
		st_ble_date_time_t	Structure for setting date and time information.
		st_ble_dev_addr_t	Structure for setting BLE address data.
		st_ble_seq_data_t	Structure for array. Select this when only one field is set, and length is set more than 2.
		struct	Structure type. Select this when selecting [Add Field].
	Length	Data length of field.	
	Abbreviation	Abbreviation of field.	
	Description	Description of field. Explain usage if needed	
Add Field	Adds a new Field inside the selected Field. Please use it if you configure data that has hierarchy. The Format/Value of the selected Field is set to [struct]. Added Field can be configured same items explained in [New Field].		
Add Enumeration	Defines enumeration usable for selected field. Items below can be configured.		
	Name	Name of enumeration. Example) enable	
	Format/Value	Value code of enumeration. Example) 0x01	
	Description	Description of enumeration.	
Delete	Delete selected field.		

3.2.4 Configuration of descriptor

When you select descriptor [D] in [Profile Tree], descriptor configuration screen (Figure 3.16) will be shown in [Detail Settings Screen]. Table 3.4 and Table 3.5 describe each item on the configuration screen.

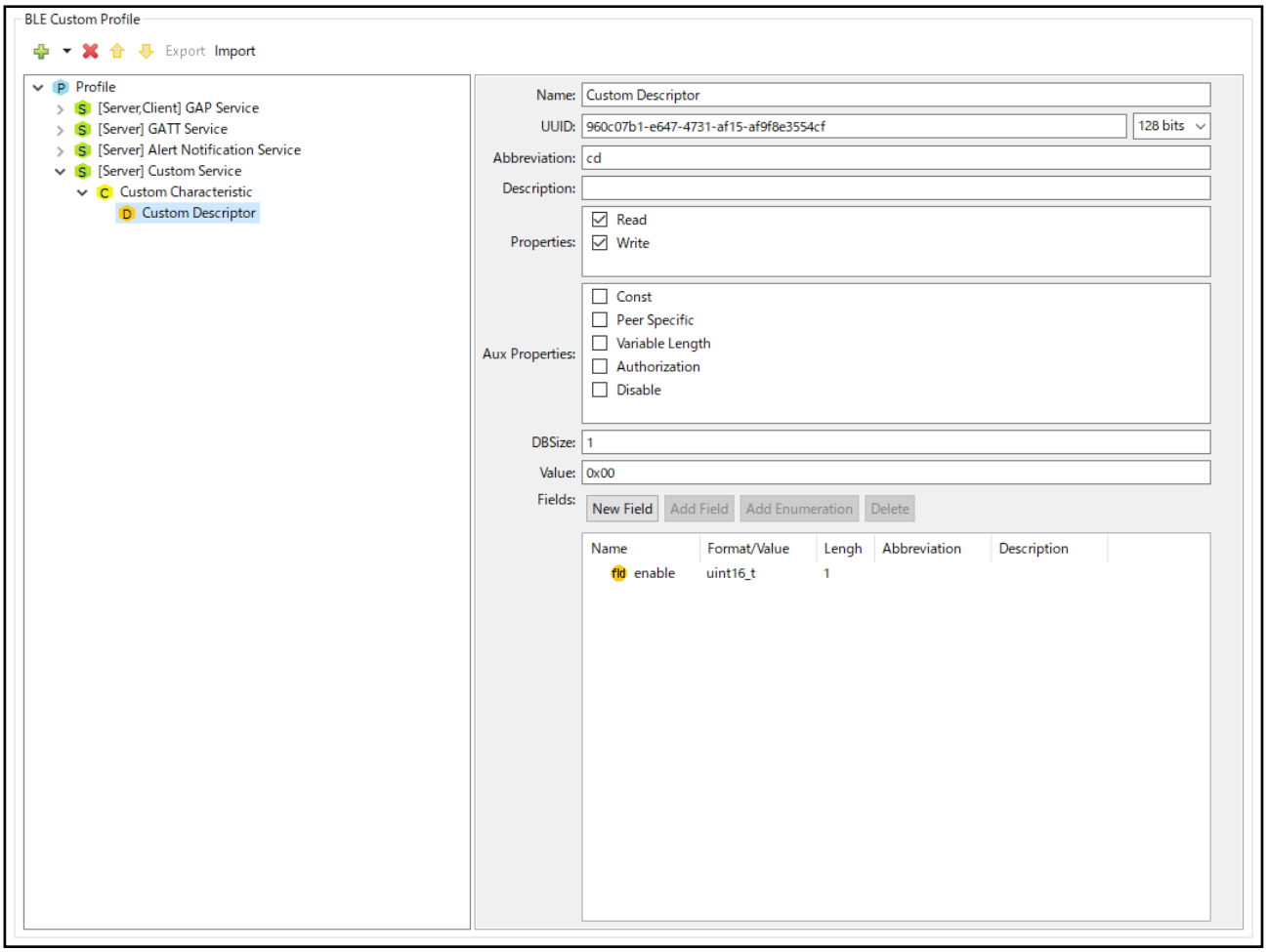

Figure 3.16 Descriptor configuration screen

Table 3.4 Descriptor configuration

Item	Description	
Name	Name of descriptor. Example) Custom Descriptor	
UUID	UUID of descriptor. Select 128bit if service is custom descriptor. Initial value is entered randomly. Please modify if needed. Example) 16bit : 0xe237 128bit : 96FE7990-2C76-89AB-DC49-AB7F123DEF9C	
Abbreviation	Abbreviation of descriptor. This value is used in function name and variable name. Beware not to conflict with other descriptors. Example) cd	
Description	Description of descriptor. Explain usage if needed. This description will be used as comment of generated program. Example) This descriptor is used for sending sensor data	
Properties	Properties of descriptor. Items below can be configured	
	Read	Enable Read operation.
	Write	Enable Write operation.
Aux Properties	AUX properties of descriptor. Items below can be configured.	
	Const	Value will not be able to change.
	Peer Specific	Value will be kept individually for each connection.
	Variable Length	Value length will be variable.
	Authorization	Enable authorization. Use function R_BLE_GAP_AuthorizeDev( ) to authorize.
	Disable	Disable attribute.
DBSize	Size of descriptor. Unit of value is byte. Example) 5	
Value	Initial value of descriptor. If you want to enter a number, enter it separated by 8bit digit. If you want to enter string, you can easily enter it by enclosing it in “”. Example) For numbers: 0x12, 0x34, 56,78 For string: “example”	
Field	Set value field used in application. Please refer Table 3.5 for configuration.	

Table 3.5 Descriptor Field configuration

New Field	Add new field. Items below can be configured.		
	Name	Name of field. Example) field_name	
	Format/Value	Format of field. Value can be selected from below.	
		bool	Boolean type
		char	char type
		uint8_t	unsigned 8bit data type
		uint16_t	unsigned 16bit data type
		uint32_t	unsigned 32bit data type
		int8_t	signed 8bit data type
		int16_t	signed 16bit data type
		int32_t	signed 32bit data type
		st_ble_ieee_11073_float_t	IEEE-11073 32bit FLOAT type
		st_ble_ieee_11073_sfloat_t	IEEE-11073 16bit SFLOAT type
		st_ble_date_time_t	Structure for setting date and time information.
		st_ble_dev_addr_t	Structure for setting BLE address data.
		st_ble_seq_data_t	Structure for array. Select this when only one field is set, and length is set more than 2.
		struct	Structure type. Select this when adding field inside field.
	Length	Data length of field.	
	Abbreviation	Abbreviation of field.	
	Description	Description of field. Explain usage if needed.	
Add Field	Add a new Field inside the selected Field. Please use it if you configure data that has hierarchy. The Format/Value of the selected Field is set to [struct]. Added Field can be configured same items explained in [New Field].		
Add Enumeration	Defines enumeration usable for selected field. Items below can be configured.		
	Name	Name of enumeration. Example) enable	
	Format/Value	Value code of enumeration. Example) 0x01	
	Description	Description of enumeration.	
Delete	Delete selected field.		

4. Implementation of program

4.1 Implementation of custom service

If you want to use features that are not defined in the SIG adopted service, you must create a custom service. This chapter guides you how to implement API programs for custom services generated from QE for BLE.

4.1.1 Implementing encode/decode function

The GATT database maintains characteristic and descriptor values as 8-bit data array. The application layer handles characteristic and descriptor values in the format specified in the Field of QE for BLE. The API program defines encode/decode function to convert the value to fit each data format. Data sent and received by BLE communication is in same format as the GATT database, but is communicated to the application after processing the value with the encode/decode function.

Figure 4.1 shows the feature of encode/decode function.

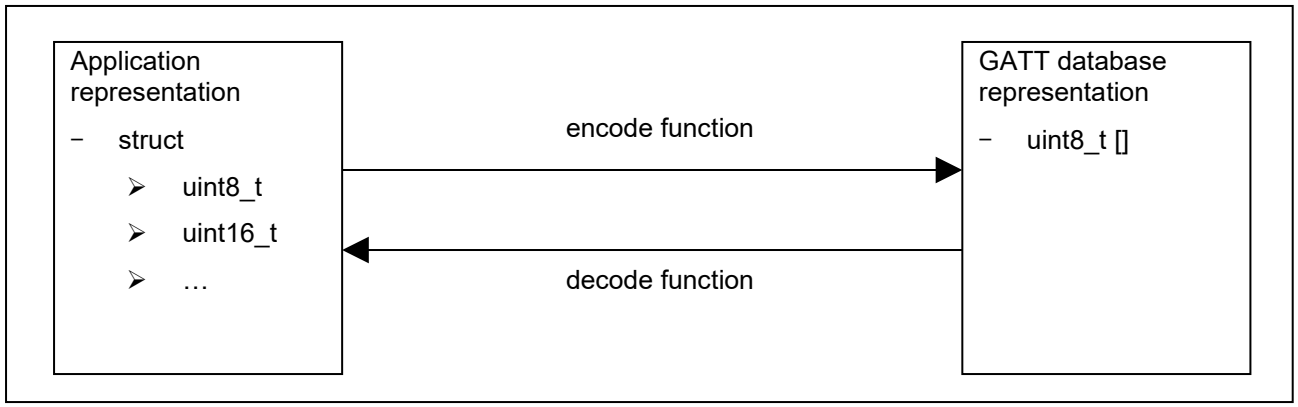

Figure 4.1 Feature of encode/decode function

In API program of custom service, encode/decode function is created but their contents are not implemented. Therefore, implementation of the encode/decode function for each data structure is needed. For basic data structures such as uint8_t type and commonly used data structures such as ieee11073 SFLOAT type, you can implement encode/decode function by calling appropriate encode/decode macros and functions. Table 4.1 describes the list of provided encode/decode macros and functions.

Table 4.1 encode/decode macro or function

Type of Field	encode	decode
char uint8_t int8_t	BT_PACK_LE_1_BYTE(dst, src)	BT_UNPACK_LE_1_BYTE(dst, src)
uint16_t int16_t	BT_PACK_LE_2_BYTE(dst, src)	BT_UNPACK_LE_2_BYTE(dst, src)
uint32_t int32_t	BT_PACK_LE_4_BYTE(dst, src)	BT_UNPACK_LE_4_BYTE(dst, src)
st_ble_ieee11073_sfloat_t	pack_st_ble_ieee11073_sfloat_t(*p_dst, *p_src)	unpack_st_ble_ieee11073_sfloat_t(*p_dst, *p_src)
st_ble_date_time_t	pack_st_ble_date_time_t(*p_dst, *p_src)	unpack_st_ble_date_time_t(*p_dst, *p_src)

Following example shows implementation of a encode function for characteristic which has field shown in Figure 4.2.In this encode function, encode macros and functions provided in Table 4.1 are used.

```
typedef struct {
    uint16_t field_u16[1]; /**<field_u16 */
    uint8_t field_u8[1]; /**< field_u8 */
    st_ble_date_time_t field_date[1]; /**< field_date */
} st_ble_css_cc_t;

static ble_status_t encode_st_ble_css_cc_t(const st_ble_css_cc_t *p_app_value,
st_ble_gatt_value_t *p_gatt_value)
{
    /* Start user code for Custom Characteristic characteristic value encode
function. Do not edit comment generated here */
       uint8_t pos = 0;
       BT_PACK_LE_2_BYTE(&p_gatt_value->p_value[pos], p_app_value->field_u16);
       pos += 2;

       BT_PACK_LE_1_BYTE(&p_gatt_value->p_value[pos], p_app_value->field_u8);
       pos += 1;

       pack_st_ble_date_time_t(&p_gatt_value->p_value[pos], p_app_value->field_date);
       pos += 7

       p_gatt_value->value_len = pos;
    /* End user code. Do not edit comment generated here */
    return BLE_SUCCESS;
}
```

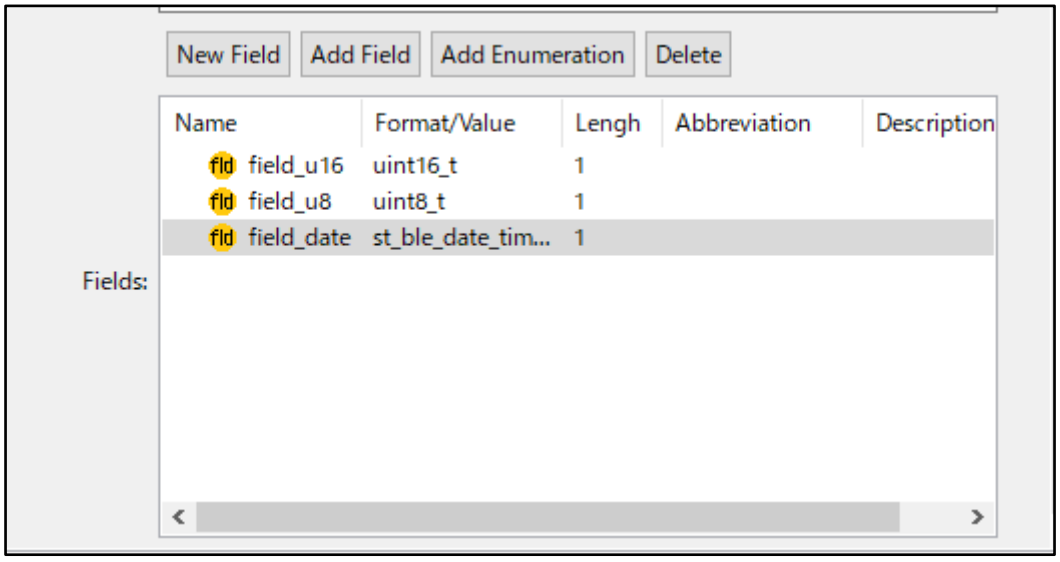

Figure 4.2 Example of field

4.1.2 Implementing callback

BLE software generates events when BLE communication such as receiving data or establishing connection occurs. You can implement application by implementing callback for those events. Callback for events can be implemented in 2 ways.

- Callback in the application.
- Callback in the service.

Beware that if you implement callback in the service, callback in application won't be called. This section guides you how to implement callback in the service.

Depending on the specifications of the custom service you implement, you may be required to implement following operations:

- Returns an error when an incorrect value is written to a characteristic or descriptor.
- Returns another characteristic value when specific instruction is written to a characteristic or descriptor.

Implementing these features in custom service API program improves portability and can be used for various applications.

Each characteristic has a structure defined as follows.

```
static const st_ble_servs_char_info_t gs_nc_char = {
    .start_hdl    = BLE_CSS_CC_DECL_HDL,
    .end_hdl      = BLE_CSS_CC_CLI_CNFG_DESC_HDL,
    .char_idx     = BLE_CSS_CC_IDX,
    .app_size     = sizeof(st_ble_css_cc_t),
    .db_size      = BLE_CSS_CC_LEN,
    .decode       = (ble_servs_attr_decode_t)decode_st_ble_css_cc_t,
    .encode       = (ble_servs_attr_encode_t)encode_st_ble_css_cc_t,
    .pp_descs     = gspp_cc_descs,
    .num_of_descs = ARRAY_SIZE(gspp_cc_descs),
};
```

You can create a callback function for a characteristic event in a custom service by editing this structure as follows.

```
static void css_cc_write_req_cb(const void *p_attr, uint16_t conn_hdl, ble_status_t
result, const void *p_app_value)
{
      /*........*/
}
static void css_cc_write_comp_cb(const void *p_attr, uint16_t conn_hdl, ble_status_t
result, const void *p_app_value)
{
      /*........*/
}

static const st_ble_servs_char_info_t gs_nc_char = {
    .start_hdl    = BLE_CSS_CC_DECL_HDL,
    .end_hdl      = BLE_CSS_CC_CLI_CNFG_DESC_HDL,
    .char_idx     = BLE_CSS_CC_IDX,
    .app_size     = sizeof(st_ble_css_cc_t),
    .db_size      = BLE_CSS_CC_LEN,
    .decode       = (ble_servs_attr_decode_t)decode_st_ble_css_cc_t,
    .encode       = (ble_servs_attr_encode_t)encode_st_ble_css_cc_t,
    .pp_descs     = gspp_cc_descs,
    .num_of_descs = ARRAY_SIZE(gspp_cc_descs),
    .write_req_cb = css_cc_write_req_cb,
    .write_comp_cb = css_cc_write_comp_cb,
};
```

The callbacks that can be registered are different in server program and client program. Table 4.2 shows callback functions that server program can register and Table 4.3 shows a callback functions that the client program can register. For more information about each event, refer the [R_BLE API Document (r_ble_api_spec.chm)] that is included in BLE FIT Module.

Table 4.2 Callback available for server characteristic

Callback	Event
write_req_cb	BLE_GATTS_OP_CHAR_PEER_WRITE_REQ
write_comp_cb	BLE_GATTS_EVENT_WRITE_RSP_COMP BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP
write_cmd_cb	BLE_GATTS_OP_CHAR_PEER_WRITE_CMD
read_req_cb	BLE_GATTS_OP_CHAR_PEER_READ_REQ
hdl_val_cnf_cb	BLE_GATTS_EVENT_HDL_VAL_CNF
flow_control_cb	BLE_VS_EVENT_TX_FLOW_STATE_CHG

Table 4.3 Callback available for client characteristic

Callback	Event
write_rsp_cb	BLE_GATTC_EVENT_CHAR_WRITE_RSP BLE_GATTC_EVENT_LONG_CHAR_WRITE_RSP
read_rsp_cb	BLE_GATTC_EVENT_CHAR_READ_RSP BLE_GATTC_EVENT_LONG_CHAR_READ_COMP
hdl_val_ntf_cb	BLE_GATTC_EVENT_HDL_VAL_NTF
hdl_val_ind_cb	BLE_GATTC_EVENT_HDL_VAL_IND

Similar to characteristic, each descriptor has structure defined as the follows. By editing this structure, you can also register callback functions in the descriptor.

```
static const st_ble_servs_desc_info_t gs_cc_cd = {
    .attr_hdl = BLE_CSS_CC_CD_DESC_HDL,
    .app_size = sizeof(uint8_t),
    .desc_idx = BLE_CSS_CC_CD_IDX,
    .db_size  = BLE_CSS_CC_CD_LEN,
    .decode   = (ble_servs_attr_decode_t)decode_uint8_t,
    .encode   = (ble_servs_attr_encode_t)encode_uint8_t,
};
```

Descriptors can register different types of callbacks than characteristics. Table 4.4 shows callbacks that can be registered on the server side, and Table 4.5 shows callbacks that can be registered on the client side. For more information about each event, refer [R_BLE API Document (r_ble_api_spec.chm)] that is included in BLE FIT Module.

Table 4.4 Callback available for server descriptor

Callback	Event
write_req_cb	BLE_GATTS_OP_CHAR_PEER_CLI_CNFG_WRITE_REQ BLE_GATTS_OP_CHAR_PEER_SER_CNFG_WRITE_REQ BLE_GATTS_OP_CHAR_PEER_USR_CNFG_WRITE_REQ BLE_GATTS_OP_CHAR_PEER_HLD_CNFG_WRITE_REQ
write_comp_cb	BLE_GATTS_EVENT_WRITE_RSP_COMP BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP
read_req_cb	BLE_GATTS_OP_CHAR_PEER_CLI_CNFG_READ_REQ BLE_GATTS_OP_CHAR_PEER_SER_CNFG_READ_REQ BLE_GATTS_OP_CHAR_PEER_USR_CNFG_READ_REQ BLE_GATTS_OP_CHAR_PEER_HLD_CNFG_READ_REQ

Table 4.5 Callback available for client descriptor

Callback	Event
write_rsp_cb	BLE_GATTC_EVENT_CHAR_WRITE_RSP BLE_GATTC_EVENT_LONG_CHAR_WRITE_RSP
read_rsp_cb	BLE_GATTC_EVENT_CHAR_READ_RSP BLE_GATTC_EVENT_LONG_CHAR_READ_COMP

4.1.3 Definition of service API

The APIs defined in SIG standard service API program and custom service API program are named according to certain rules. So, you can determine which API to use in user application just by checking the name of API.
API for operation about value of characteristics and descriptor is named as follows.

R_BLE_[service][S or C]_[operation]

[service] is the string set to [abbreviation] of the service in QE for BLE. For [S or C], S is set service is configured as server, C is set if service is configured as client.

The string set to [operation] is sending and receiving behavior of the value of characteristics and descriptors in BLE communication. Table 4.6 lists strings set to [operation] generated in the server side API program and Table 4.7 lists strings set to [operation] generated in the client side API program. In both tables, [characteristic] is the string set to [abbreviation] of the characteristic in QE for BLE, [descriptor] is the string set to the [abbreviation] of the descriptor in QE for BLE.

Table 4.6 Server API

operation	description
Get[characteristic] Get[characteristic][descriptor]	Get characteristic/descriptor value from GATT database. You can check database value after operation such as Write Characteristic Value.
Set[characteristic] Set[characteristic][descriptor]	Set characteristic/descriptor value to GATT database. Value set to database is used in operation such as Read Characteristic Value.
Notify[characteristic]	Start Notification operation by sending Handle Value Notification. Characteristic value will not be stored to GATT database by calling this API.
Indicate[characteristic]	Start Indication operation by sending Handle Value Indication. Characteristic value will not be stored to GATT database by calling this API.

Table 4.7 Client API

operation	description
Get[characteristic]AttrHdl	Get characteristic attribute handle discovered in Discovery operation. You can also get Attribute handle of descriptor included in characteristic. Complete Discovery operation before calling this function.
Write[characteristic] Write[characteristic][descriptor]	Start Write Characteristic Value operation by sending Write Request. If value length exceeds MTU size, this function will start Write Long Characteristic Value operation by sending Write Prepare request.
Read[characteristic] Read[characteristic][descriptor]	Start Read Characteristic Value operation by sending Read Request. If value length exceeds MTU size, this function will start Write Long Characteristic Value operation by sending Read Blob Request.

Each service generated from QE for BLE defines the function listed in Table 4.8, regardless of its configuration. In this table, [service] is sting set to [Abbreviation] of the service in QE for BLE, For [S or C], S is set service is configured as server, C is set if service is configured as client.

Table 4.8 API defined in each service API program

API	description
R_BLE_[service][S or C]_Init	Initialization function for the service. Calling this function is necessary before using service API program.
R_BLE_[service][S or C]_GetServAttrHdl	Returns service attribute handle which is discovered in discovery operation. Call this function after discovery operation is completed. This function is implemented only on client API program.
R_BLE_[service][S or C]_ServDiscCb	Function to operate discovery operation. This function is used as callback function when using discovery library. This function is implemented only on client API program.

4.2 Implementation of app_main.c

app_main.c is the underlying framework for implementing user applications and profiles. This chapter guides you on how to implement user applications and profiles.

4.2.1 Implementing callback

BLE software generates events when BLE communication such as receiving data or establishing connection occurs. You can implement application by implementing callback for those events. Callback for events can be implemented in 2 ways.

- Callback in the application.
- Callback in the service.

Beware that if you implement callback in the service, callback in application won't be called. This section guides you how to implement callback in the application.

Handling of basic events for BLE communication is implemented in application.

For events that comply with Bluetooth specifications, such as the establishment of connection or the completion of pairing, please refer [R_BLE API Document (r_ble_api_spec.chm)] that comes with the BLE FIT module.

For events that exchanges each data of characteristic or descriptor included in the profile is implemented in the callback function output as a skeleton program. For information about the events that occur, refer [4.2.1.1 Service events]. The following is an example of implementing a custom service callback function.

```
static void lss_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t
*p_data)
{
    switch (type)
    {
        case BLE_LSS_EVENT_BLINK_RATE_WRITE_REQ:
        {
            uint8_t rate = *(uint8_t *)p_data->p_param;
            R_BLE_TIMER_UpdateTimeout(gs_timer_hdl, rate*100);
        } break;

        default:
            break;
    }
}
```

4.2.1.1 Service events

API program for all services, including custom service, have events defined for sending and receiving data in BLE communications. Users can develop applications by implementing behavior responding to defined events in callback functions.

Each defined event is named based on the type of data and behavior in communication.

Events about characteristic value are named as follows.

BLE_[service][S or C]_EVENT_[characteristic]_[event type]

[service] is the string set to [abbreviation] of the service in QE for BLE, and [characteristic] is the string set to [abbreviation] of the characteristic in QE for BLE. [S or C] is S if the service is set to server, C if the service is set to client. [event type] is determined by the type of event described below.

Events about descriptor value are named as follows.

[service] is the string set to [abbreviation] of the service in QE for BLE, [characteristic] is the string set to [abbreviation] of the characteristic in QE for BLE, [descriptor] is the string set to [abbreviation] of the descriptor. [S or C] is S if the service is set to server, C if the service is set to client. [event type] is determined by the type of event described below.

BLE_[service][S or C]_EVENT_[characteristic]_[descriptor]_[event type]

The string set to [event type] is determined by sending and receiving events in BLE communication. The type of event that occurs in BLE communication is different on the server side and client side. Table 4.9 lists the events that occur on the server side, and Table 4.10 lists the event that occur on the client side.

Table 4.9 Server event

Event	description
WRITE_REQ	Event that occurs when Write Request or Prepare Write Request is received. It is used in Write Characteristic Value operation or Write Characteristic Long Value operation. GATT event: BLE_GATTS_OP_CHAR_PEER_WRITE_REQ
WRITE_COMP	Event that occurs when Write Response or Execute Write Response is sent. It is used in Write Characteristic Value operation or Write Characteristic Long Value operation. GATT event: BLE_GATTS_EVENT_WRITE_RSP_COMP BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP
WRITE_CMD	Event that occurs when Write Command or Signed Write Command is received. It is used in Write Characteristic Without Response operation or Signed Write operation. GATT event: BLE_GATTS_OP_CHAR_PEER_WRITE_CMD
READ_REQ	Event that occurs when Read Request is received. It is used in Read Characteristic Value operation or Read Characteristic Long Value operation. GATT event: BLE_GATTS_OP_CHAR_PEER_READ_REQ
HDL_VAL_CNF	Event that occurs when Handle Value Confirmation is received. It is used in Indication operation. GATT event: BLE_GATTS_EVENT_HDL_VAL_CNF

Table 4.10 Client event

イベント	説明
WRITE_RSP	Event that occurs when Write Response or Prepare Write Response is received. It is used in Write Characteristic Value operation or Write Characteristic Long Value operation. GATT event: BLE_GATTC_EVENT_WRITE_RSP BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP
READ_RSP	Event that occurs when Read Response or Read Blob Response is received. It is used in Read Characteristic Value operation or Read Characteristic Long Value operation. GATT event: BLE_GATTC_EVENT_CHAR_READ_RSP BLE_GATTC_EVENT_CHAR_PART_READ_RSP (If operation failed) BLE_GATTC_EVENT_LONG_CHAR_READ_COMP
HDL_VAL_NTF	Event that occurs when Handle Value Notification is received. It is used in Notification operation. GATT event: BLE_GATTC_EVENT_HDL_VAL_NTF
HDL_VAL_IND	Event that occurs when Handle Value Indication is received. It is used in Indication operation. GATT event: BLE_GATTC_EVENT_HDL_VAL_IND

Example of defined event in custom service is shown below.

```
/* LED Switch Service (Abbreviation:ls) Client Event Type Definition */
typedef enum {

    /* Switch State Characteristic (Abbreviation:switch state) */
    /* Handle Value Notification */
    BLE_LSC_EVENT_SWITCH_STATE_HDL_VAL_NTF
       = BLE_SERVC_ATTR_EVENT(BLE_LSC_SWITCH_STATE_IDX, BLE_SERVC_HDL_VAL_NTF),

    /* Client Characteristic Configuration Descriptor (Abbreviation:cli_cnfg) */
    /* Read response */
    BLE_LSC_EVENT_SWITCH_STATE_CLI_CNFG_READ_RSP
       = BLE_SERVC_ATTR_EVENT(BLE_LSC_SWITCH_STATE_CLI_CNFG_IDX, BLE_SERVC_READ_RSP),
    /* Write response */
    BLE_LSC_EVENT_SWITCH_STATE_CLI_CNFG_WRITE_RSP
       = BLE_SERVC_ATTR_EVENT(BLE_LSC_SWITCH_STATE_CLI_CNFG_IDX,
BLE_SERVC_WRITE_RSP),

    /* LED Blink Rate Characteristic (Abbreviation:blink rate) */
    /* Read Response */
    BLE_LSC_EVENT_BLINK_RATE_READ_RSP
       = BLE_SERVC_ATTR_EVENT(BLE_LSC_BLINK_RATE_IDX, BLE_SERVC_READ_RSP),
    /* Write Response */
    BLE_LSC_EVENT_BLINK_RATE_WRITE_RSP
       = BLE_SERVC_ATTR_EVENT(BLE_LSC_BLINK_RATE_IDX, BLE_SERVC_WRITE_RSP),

} e_ble_lsc_event_t;
```

4.3 Notice

4.3.1 Implementation of multiple services

When implementing multiple services, take care of the characteristic and descriptor code sizes contained in the service. If the code size exceeds the RAM/ROM size of target device, it cannot be compiled. Please refer [RX23W Group BLE Module Firmware Integration Technology(R01AN4860)] for ROM/RAM size that BLE Protocol Stack uses.

4.3.2 Implementation of same service

If you add multiple same SIG standard services to a profile, QE for BLE cannot correctly generate programs due to problem such as conflicts of file name. Therefore, if you want to implement multiple same services, you need to add only one service as SIG standard service and add the others as custom service on QE for BLE. For example, assume that you want to implement 2 Human Interface Device Service (HIDS), which is SIG standard service.

First, you need to add 2 HIDS as SIG standard service in QE for BLE. Change 1 of these HIDS from SIG standard service to custom service. To change from SIG standard service to custom service, click the customize button on the service setting screen. You need to make the following changes to the service that you changed to the custom service:

- Change [UUID] of service so that service UUID matches between the same service. If you want to treat the custom service as SIG standard service, set [UUID] to 16bit and change the value.
- Change [abbreviation] of service so that it does not conflict with other services. This is to prevent conflicts on file name, function name, and variable name because [abbreviation] is used for them. Similarly, set [abbreviation] of characteristic and descriptor to string which do not conflict with others.

Setting on QE for BLE is over. Figure 4.3 shows how to configure multiple SIG standard services on QE for BLE.

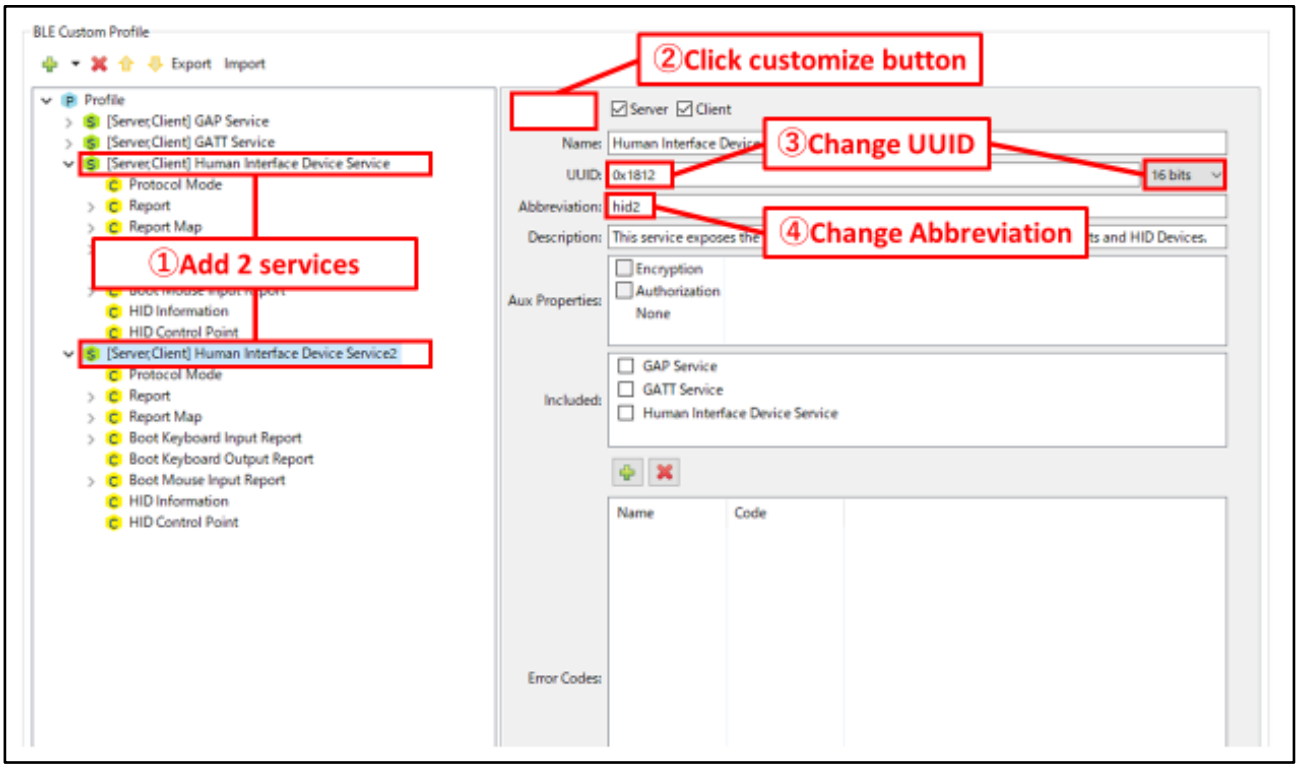

Figure 4.3 Configure multiple service on QE for BLE

Because the program generated from custom services are skeleton program, it is necessary to implement the actual state of process. Program generated from SIG standard services has same mechanism and is implemented according to the defined specification, so refer this program to implement skeleton program of custom service. The parts that must be implemented vary from service to service, but in many cases, following implementation is needed:

- Implements encode/decode function. Since the structure of the characteristic or descriptor remains the same, you can port many parts of implementation. Beware of differences in function name and variable name.
- Implements callback function in service. This is used when you want to automatically return error for invalid value written or automatically return certain value for specific value written. Implementation is needed according to functionality of each service.

In addition, if [central] is selected in the profile setting screen, discovery operation program using discovery library is implemented in file app_main.c. Among them, the array gs_disc_entries[] defines UUID and discovery callback function for each service included in profile. To discover services those have same service UUID, you need to add element idx which is index number for them. The following is example of implementing a program with 2 HIDS.

```
/* Human Interface Device Service UUID */
static uint8_t HIDC_UUID[] = { 0x12, 0x18 }; //HIDS specific service UUID
/* Human Interface Device Service2 UUID */
static uint8_t HID2C_UUID[] = { 0x12, 0x18 }; //Same service UUID

/* Service discovery parameters */
static st_ble_disc_entry_t gs_disc_entries[] = {
    {
       .p_uuid      = HIDC_UUID,
       .uuid_type   = BLE_GATT_16_BIT_UUID_FORMAT,
       .serv_cb     = R_BLE_HIDC_ServDiscCb,
       /* Add member [idx] */
       .idx         = 0, /* Set index number if service UUID is same */
    },
    {
       .p_uuid      = HID2C_UUID,
       .uuid_type   = BLE_GATT_16_BIT_UUID_FORMAT,
       .serv_cb     = R_BLE_HID2C_ServDiscCb,
       /* Add member [idx] */
       .idx         = 1, /* Set index number if service UUID is same */
    },
};
```

4.3.3 Implementation of secondary service

QE for BLE treats all services as primary services. Therefore, if you want to use secondary service, you need to modify the generated program. How to change program is different on the server side and client side.

Server Side

QE for BLE generates GATT database which stores information of services which have check in [server]. Since QE for BLE treats all services as primary service, generated GATT database defines all services as primary service. You need to modify service information defined in GATT database.

Change the array gs_gatt_type_table[ ] defined in file gatt_db.c. In this array, following 2 point needs to be changed:

- Add definition for secondary service. Refer to the other elements of the array and create element that has [UUID_Offset] is 2 and correct attribute handles of secondary services.
- Change element which defines [Primary Service Declaration]. Change it to specify the correct attribute handle.

The following is the example of implementation on array gs_gatt_type_table[ ].

```
static const st_ble_gatts_db_uuid_cfg_t gs_gatt_type_table[] =
{
    /* 0 : Primary Service Declaration */
    {
        /* UUID Offset */
        0,

        /* First Occurrence for type */
        /* Change this value to proper handle */
        0x000C,

        /* Last Occurrence for type */
        /* Change this value to proper handle */
        0x0026,
    },

    /* Add from here */
    /* 2 : Secondary Service Declaration */
    {
        /* UUID Offset */
        /* set 2 for this value */
        2,

        /* First Occurrence for type */
         /* Change this value to proper handle */
        0x0010,

        /* Last Occurrence for type */
        /* Change this value to proper handle */
        0x0000,
    },
    /* Add until here */

}
```

Also, change array gs_gatt_db_attr_table[ ]. n this array, following 2 point needs to be changed:

- Change [UUID_Offset] section of service declaration which you want to change to secondary service. [UUID_offset] determines attribute type of data. In [UUID_Offset], 0 stands for primary service and 2 stands for secondary service. Set 2 for [UUID_Offset].
- change element [Next Attribute Type Index] to indicate correct attribute handle. [Next Attribute Type Index] holds attribute handle of next data which has same attribute type. If modified data was the last data with same attribute type, enter 0x0000 for [Next Attribute Type Index].

The example of implementation on array gs_gatt_type_table[ ] is shown on the next page.

Note: If you are using GATT Browser for verifying implemented GATT database, make sure that the service which you changed to secondary service is included by other service.

```
static const st_ble_gatts_db_attr_cfg_t gs_gatt_db_attr_table[] =
{
    /* Handle: 0x000C */
    /* GATT Service: Primary Service Declaration */
    {
        /* Properties */
        BLE_GATT_DB_READ,
        /* Auxiliary Properties */
        BLE_GATT_DB_FIXED_LENGTH_PROPERTY,
        /* Value Size */
        2,

        /* Next Attribute Type Index */
        /* change this value to handle of next primary service declaration */
        0x0026,      /* 0x0010 → 0x0026 */

        /* UUID Offset */
        0,
        /* Value */
        (uint8_t *)(gs_gatt_const_uuid_arr + 20),
    },


    /* Example: Secondary Service Declaration */
    /* Handle: 0x0010 */
    /* Human Interface Device Service: Primary Service Declaration */
    {
       /* Properties */
       BLE_GATT_DB_READ,
       /* Auxiliary Properties */
       BLE_GATT_DB_FIXED_LENGTH_PROPERTY,
       /* Value Size */
       2,

       /* Next Attribute Type Index */
       /* Change this value to proper handle */
       /* Last secondary service declared: 0x0000 */
       /* Not last secondary service declared: handle of next secondary service
declaration */
      0x0000,      /* 0x0026 → 0x0000 */

       /* UUID Offset */
       /* Change this value to proper Attribute type */
       /* Primary service declaration: 0 */
       /* Secondary service declaration: 2 */
       2,     /* 0 → 2 */

       /* Value */
       (uint8_t *)(gs_gatt_const_uuid_arr + 26),
    },


    /* Handle: 0x0026 */
    /* Human Interface Device Service2: Primary Service Declaration */

}
```

Client Side
If you selected [Central] on the profile configuration screen, QE for BLE generate the code to perform the discovery operation. Generated program performs discovery operation only to primary service using Discovery Library provided by BLE Protocol Stack. Therefore, in order to perform discovery operation to secondary service, you need to call directly GATT Client API provided by BLE Protocol Stack. For more information about GATT Client API, refer the [R_BLE API document (r_ble_api_spec.chm)] that is included in BLE FIT module.

The following shows example for implementation to perform secondary service discovery operation.

Modify gs_disc_entries
QE for BLE treats all services as primary service. Remove secondary service from the array gs_disc_entries[ ] which is the list of primary services for correct discovery operation.

Implementation to perform discovery operation to secondary services using GATT Client API.

In order to perform secondary service discovery operation, use function R_BLE_GATTC_DiscAllSecondServ( ) to discover secondary service attribute handle. For more information about function, refer [R_BLE API document (r_ble_api_spec.chm)].

4.3.4 Implementation of discovery operation about included service

Specifying included service

If you selected [Central] on the profile configuration screen, QE for BLE generate the code to perform the discovery operation. Generated program performs discovery operation only to primary service using Discovery Library provided by BLE Protocol Stack.

If service has specific service as an included service, you need to confirm its structure to perform discovery operation to specific service. Discovery library provide feature to perform discovery operation confirming this structure. Discovery library perform discovery operation to attribute handle range that included service declaration has if included service entries are registered in discovery entry of parent service. Modify the variable gs_disc_entries in the app_main.c as the following, in order to register included service entries to discovery entry of parent service.

The code generated by QE for BLE

```
/*PRIMARY service entry */
static st_ble_disc_entry_t gs_disc_entries[] =
{
	{
		/*Weight Scale service disc entry */
		.p_uuid = (uint8_t *)BLE_WSC_UUID,
		.uuid_type = BLE_GATT_16_BIT_UUID_FORMAT,
		.serv_cb = R_BLE_WSC_ServDiscCb,
	},
	{	/*Body Composition service disc entry */
		.p_uuid = (uint8_t *)BLE_BCC_UUID,
		.uuid_type = BLE_GATT_16_BIT_UUID_FORMAT,
		.serv_cb = R_BLE_BCC_ServDiscCb,
	},
};
```

The code modified to discover included service.

```
/*Add INCLUDE service entry*/
static st_ble_disc_entry_t gs_disc_wsc_inc_entries[] =
{
	/*Body Composition service disc entry AS A INCLUDE SERVICE IN WSS*/
	{
		.p_uuid = (uint8_t *)BLE_BCC_UUID,
		.uuid_type = BLE_GATT_16_BIT_UUID_FORMAT,
		.serv_cb = R_BLE_BCC_ServDiscCb,
		.num_of_inc_servs = 0,
	},
};
/*PRIMARY service entry */
static st_ble_disc_entry_t gs_disc_entries[] =
{
	/*Weight Scale service disc entry as a primary service*/
	{
		.p_uuid = (uint8_t *)BLE_WSC_UUID,
		.uuid_type = BLE_GATT_16_BIT_UUID_FORMAT,
		.serv_cb = R_BLE_WSC_ServDiscCb,
		/* Register include service entry*/
		.inc_servs = gs_disc_wsc_inc_entries,
		.num_of_inc_servs = 1

	},
};
```

Store Attribute handle of included service

Discovered attribute handle of included service will be passed to parent service API program. But parent service API program don't store attribute handle of included service. Therefore, in case Service YYY is discovered as included service that Service XXX has, you can't get range of its attribute handle by calling service YYY's API R_BLE_YYY_GetServAttrhdl().

If service YYY's range of attribute handle is needed, modify service XXX's API program (r_ble_xxx.c) so that the notification that service YYY is discovered as a include service is delivered to service YYY's discovery callback function.

The following show example in case Service XXX have 16bit UUID and have service YYY as included service. Take care the data type is different in 128bit UUID and in 16bit UUID.

```
#include <string.h>
#include "r_ble_XXX.h"
#include "profile_cmn/r_ble_servc_if.h"

/* ADD : including discovery library and include service yyy */
#include "discovery/r_ble_disc.h"
#include "r_ble_yyy.h"

void R_BLE_XXX_ServDiscCb(uint16_t conn_hdl, uint8_t serv_idx, uint16_t type, void
*p_param)
{
	/* ADD : */
	uint16_t YYY_UUID = 0x0000;
	if (type == BLE_DISC_INC_SERV_FOUND)
	{
		st_disc_inc_serv_param_t * evt_param =
			(st_disc_inc_serv_param_t *)p_param;

		if (evt_param->uuid_type == BLE_GATT_16_BIT_UUID_FORMAT)
		{
			if(YYY_UUID == evt_param->value.inc_serv_16.service.uuid_16)
			{
				st_disc_serv_param_t serv_param = {
				.uuid_type            = BLE_GATT_16_BIT_UUID_FORMAT,
				.value.serv_16.range   =
							evt_param->value.inc_serv_16.service.range,
				.value.serv_16.uuid_16 =
							evt_param->value.inc_serv_16.service.uuid_16,
			    };
				R_BLE_YYY_ServDiscCb(
						/* Connection handle */
						conn_hdl,
						/* idx */
						0,
						/* Notify as a primary service */
						BLE_DISC_PRIM_SERV_FOUND,
						/* Service handle information */
						&serv_param);
			}
		}
	}
	/* Generated code */
}
```

5. Running created profile

5.1 Using code generated by Smart Configurator

If you are running program generated by Smart Configurator, you can build your project without changing configuration of files or directories. Project can be built after implementing the skeleton programs.

5.2 Using FITDemos

If you are developing profile based on the sample projects from FITDemos included in the BLE FIT module, follow these steps:

When QE for BLE is used in a sample project, the program will be generated replacing the files in the sample project. Therefore, a new project needs to be created for QE for BLE to generate the program. Next, copy the program generated from the new project to the sample project.

Copy each of the following files:

- app_main.c → Replace with original app_main.c.
- gatt_db.c and gatt_db.h → Replace with original gatt_db.c and gatt_db.h.
- r_ble_[service] → Copy all files to src.

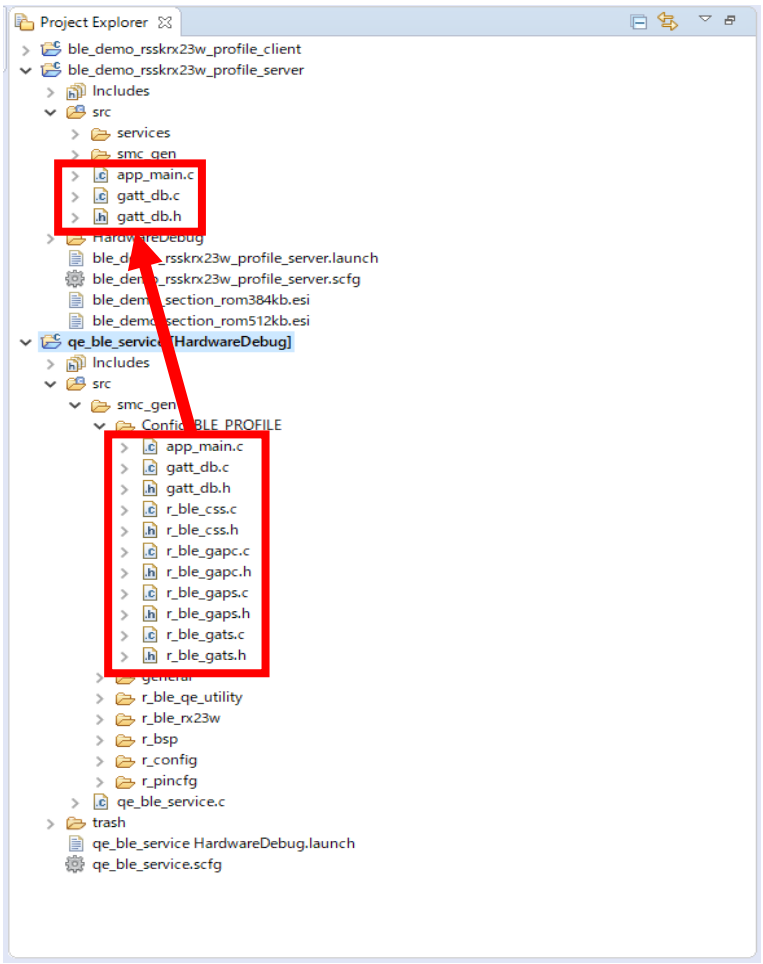

Figure 5.1 Copy generated program files to sample program

Edit app_main.c as follows to use in the sample project:

- Change app_main() to main()

```
//void app_main(void)
void main(void)
{
    R_BLE_Open();
    ble_app_init();

    while (1)
    {
        R_BLE_Execute();
    }

    R_BLE_Close();
}
```

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov.27.19	—	First edition issued.

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(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.4.0-1 November 2017)

Corporate Headquarters

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

Trademarks

Contact information

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