RX23W Group

Bluetooth Low Energy Profile Developer's Guide

Introduction

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

Target Device

- RX23W Group

Related Documents

- Bluetooth Core Specification <https://www.bluetooth.com>
- Core Specification Supplement <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)
- Bluetooth Low Energy Protocol Stack Basic Package: User's Manual (R01UW0205)
- RX23W Group Bluetooth Low Energy Application Developer's Guide (R01AN5504)
- QE for BLE [RA, RE, RX] V1.4.0 Release Note(R20UT5109)

Contents

1. Overview

1.1 Structure of profile

In Bluetooth LE Communication, Generic Attribute Protocol (GATT) is primarily used. GATT defines client and server roles, and profile communication is performed between client and server. Profiles are protocols developed for many applications consisting of one or more services. Profile communication is allowed between devices that supports same profile. Server device has profile data in GATT database, and client device accesses the profile data. The server device can also notify profile data to the client device using. By using them, it is possible to transmit and receive in Bluetooth LE communication. The server has the profile data in GATT database and the client accesses the profile data by Bluetooth LE communication.

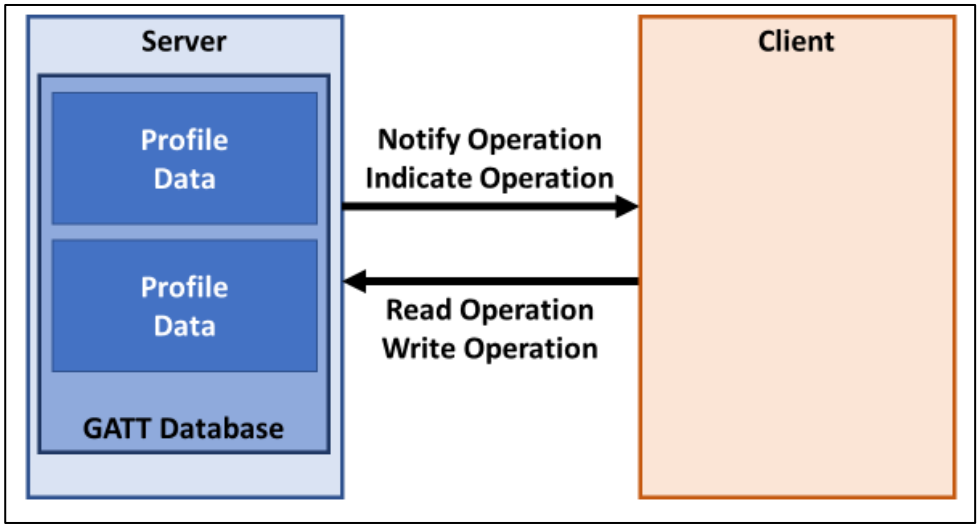

Figure1.1 Overview of profile communication

Figure1.2 shows Structure of profile in Bluetooth LE software.

In this Bluetooth LE software, user application and profiles run on the BLE Protocol Stack. Profile consists of 3 types of programs:

- Framework for using Bluetooth LE 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.

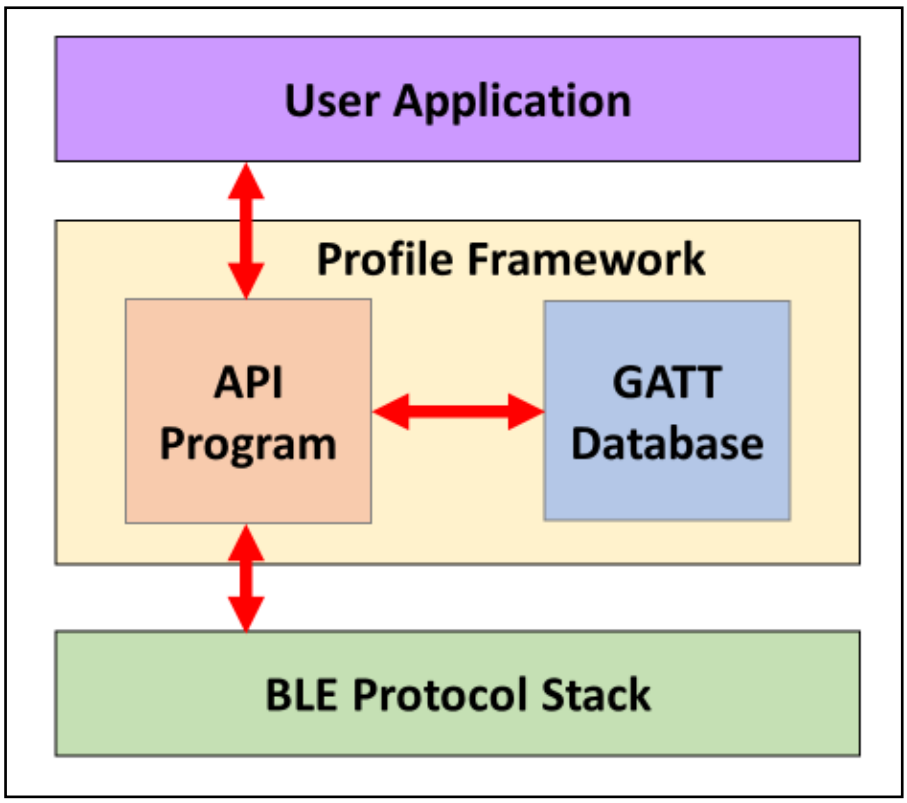

Figure1.2 Structure of profile

Bluetooth LE 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 Bluetooth LE 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

Figure1.3 shows flow of profile development.

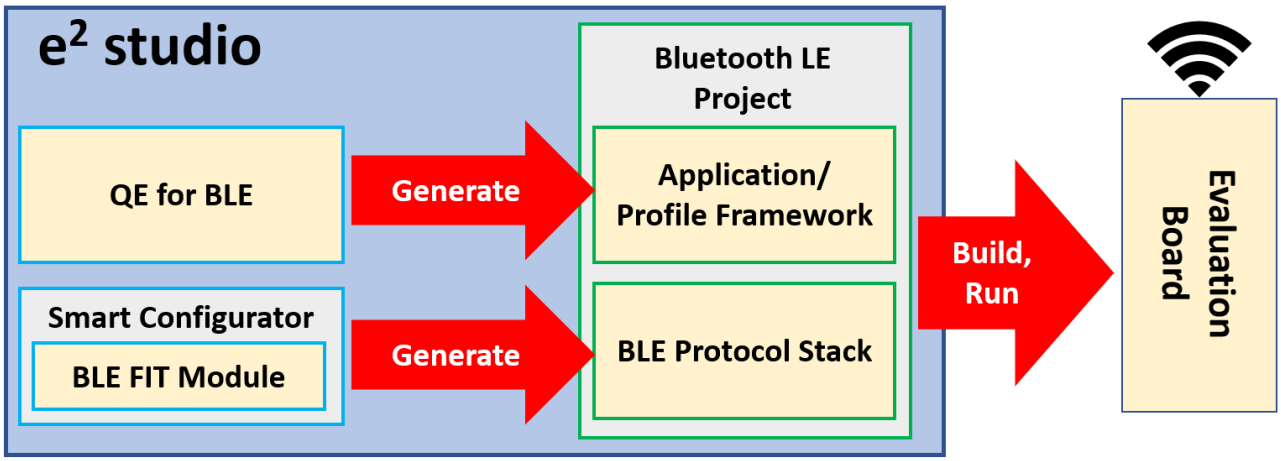

Figure1.3 Flow of profile development

The steps for profile development are as follows:

1) Create Bluetooth LE project

Add a Bluetooth LE communicable project for RX23W to e²studio. See [2 Development environment] for details.

2) Develop profile by QE for BLE

Operate the GUI of QE for BLE to design the profile. See [3 Profile Configuration in QE for BLE] for details. QE for BLE generates the API program of the designed profile, GATT database, and framework in the user project on e²studio.

3) Implement application

Implement the application on user project using program generated from QE for BLE. 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 runed on evaluation board. For more information, refer [5 Build and Run created profile].

2. Development environment

2.1 Software Requirements

This document is based on the following software environment.

- e2 studio 2022-01 (64bit)
- CC-RX compiler v2.08.00
- BLE FIT module v2.30
- QE for BLE [RA, RE, RX] v1.4.0
- QE for BLE [RA, RE, RX] Utility v1.40

For more information about QE for BLE [RA, RE, RX], refer [2.2 QE for BLE]. For guide on building development environment, refer [0

Building development environment].

2.2 QE for BLE

QE for BLE provides GUI to configure profile and generates program. Table 2.1 lists the program that QE for BLE generates.

Table 2.1 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 and notifying profile data. File is generated for each service that configure profile. 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]/qe_gen/ble.

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

For SIG adopted services, QE for BLE generates API program made by Renesas. These programs are almost authenticated by Bluetooth SIG. Table 2.2 shows the list of SIG adopted service that are supported by QE for BLE. Also, Table 2.3 shows list of profile that are supported by QE for BLE.

Specifications of each service is defined by Bluetooth SIG. Check Web page of Bluetooth SIG (https://www.bluetooth.com) for more information.

Table 2.2 SIG adopted service supported by BLE QE Utility Module

Service name	abbreviation	version	Service name	abbreviation	version
Alert Notification Service	ANS	1.0	Automation IO Service	AIOS	1.0
Battery Service	BAS	1.0	Blood Pressure Service	BLS	1.0
Body Composition Service	BCS	1.0	Bond Management Service	BMS	1.0
Continuous Glucose Monitoring Service	CGMS	1.0.1	Current Time Service	CTS	1.1
Cycling Power Service	CPS	1.1	Cycling Speed and Cadence Service	CSCS	1.0
Device Information Service	DIS	1.1	Environmental Sensing Service	ESS	1.0
Fitness Machine Service	FTMS	1.0	Glucose Service	GLS	1.0
Health Thermometer Service	HTS	1.0	Heart Rate Service	HRS	1.0
Human Interface Device Service	HIDS	1.0	Immediate Alert Service	IAS	1.0
Insulin Delivery Service	IDS	1.0	Link Loss Service	LLS	1.0.1
Location and Navigation Service	LNS	1.0	Next DST Change Service	NDCS	1.0
Object Transfer Service	OTS	1.0	Phone Alert Status Service	PASS	1.0
Pulse Oximeter Service	PLXS	1.0	Reconnection Configuration Service	RCS	1.0
Reference Time Update Service	RTUS	1.0	Running Speed and Cadence Service	RSCS	1.0
Scan Parameters Service	ScPS	1.0	Tx Power Service	TPS	1.0
User Data Service	UDS	1.0	Weight Scale Service	WSS	1.0
GATT Service	GATS	-	GAP Service	GAPS	-

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

Table 2.3 Profile supported by BLE QE Utility Module

Profile	Service that configure profile			
Blood Pressure Profile [BLP]	BLS	DIS		
Health Thermometer Profile [HTP]	HTS	DIS		
Heart Rate Profile [HRP]	HRS	DIS		
Glucose Profile [GLP]	GLS	DIS		
Pulse Oximeter Profile [PLXP]	PLXS	DIS	(BAS)	(CTS)
	(BMS)			
Continuous Glucose Monitoring Profile [CGMP]	CGMS	DIS	(BMS)	
Reconnection Configuration Profile [RCP]	RCS	(BMS)		
Insulin Delivery Profile [IDP]	IDS	DIS	(BAS)	(CTS)
	(BMS)	(IAS)		
Cycling Power Profile [CPP]	CPS	(DIS)	(BAS)	
Cycling Speed and Cadence Profile [CSCP]	CSCS	(DIS)		
Running Speed and Cadence Profile [RSCP]	RSCS	(DIS)		
Location and Navigation Profile [LNP]	LNS	(DIS)	(BAS)	
Weight Scale Profile [WSP]	WSS	DIS	(BCS)	(BAS)
	(CTS)	(UDS)		
Fitness Machine Profile [FTMP]	FTMS	(DIS)	(UDS)	
Environmental Sensing Profile [ESP]	ESS	(DIS)	BAS)	
Find Me Profile [FMP]	IAS			
Proximity Profile [PXP]	IAS	(LLS)	(TPS)	
Alert Notification Profile [ANP]	ANS			
Phone Alert Status Profile [PASP]	PASS			
Time Profile [TIP]	CTS	(NDCS)	(RTUS)	
HID over GATT Profile [HOGP]	HIDS	DIS	BAS	(ScPS)
Scan Parameters Profile [ScPP]	ScPS			
Automation IO Profile [AIOP]	AIOS			

Note: Services without () are mandatory services, and services with () are Optional services. If you add a profile in QE for BLE, only mandatory services are added to profile tree.

2.3 Building development environment

This section explains how to build a development environment using QE for BLE.

2.3.1 Add Bluetooth LE project

Refer to Chapter 4, "Project of BLE FIT Module" in the following document and add a project that can use the Bluetooth LE function to the workspace of e²studio.

- RX23W Group BLE Module Firmware Integration Technology(R01AN4860)

2.3.2 Install QE for BLE

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

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

Install method is as follows:

1. Activate e² studio.
2. Select [Renesas Views] → [Renesas Software Installer] menu to open the [Renesas Software Installer] dialog.
3. Select [Renesas QE] and click the [Next>] button.
4. Check the [QE for BLE [RA, RE, RX] (v1.4.0)] check box and click the [Finish (F)] button.
5. In the [Install] dialog, make sure that the [Renesas QE for BLE [RA, RE, RX]] check box and the [Renesas QE for BLE [RA, RE, RX] Utility] check box are checked, and then click [Click the Next>] button.
6. Confirm that the installation targets are [Renesas QE for BLE [RA, RE, RX]] and [Renesas QE for BLE [RA, RE, RX] Utility], and then [Next (N)>]. Press the button.
7. After confirming the license, if you agree to the license, select the [I accept the terms of the terms of use (A)] radio button and click the [Exit (F)] button.
8. If the dialog for selecting a trusted certificate is displayed, check the displayed certificate, and then click the [OK] button to continue the installation.
9. Restart e²studio.
10. Start this product from the [Renesas Views] - [Renesas QE] menu of e²studio. For how to use this product, refer to the QE item from the [Help] menu of e²studio.

3. Profile Configuration in QE for BLE

This section guides how to configure profiles in QE for BLE

3.1 Overview of profile configuration

This section describes the procedure for profile development using QE for BLE

3.1.1 Run QE for BLE

Launch QE for BLE by selecting [Renesas view]→[Renesas QE]→[R_BLE Custom Profile RA, RE, RX (QE)] in menu of the e²studio.

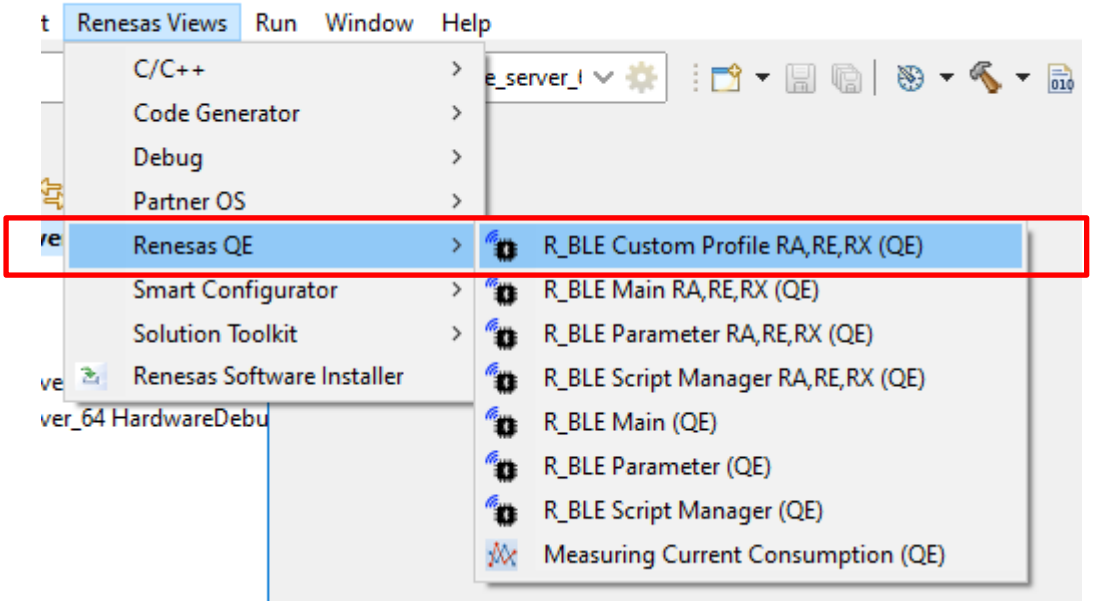

Figure 3.1 Open QE for BLE

Note: If your project contains an older version of QE for BLE, you will be prompted to migrate to the latest QE for BLE.

- QE for BLE [RA,RE,RX] V1.40 Relese note

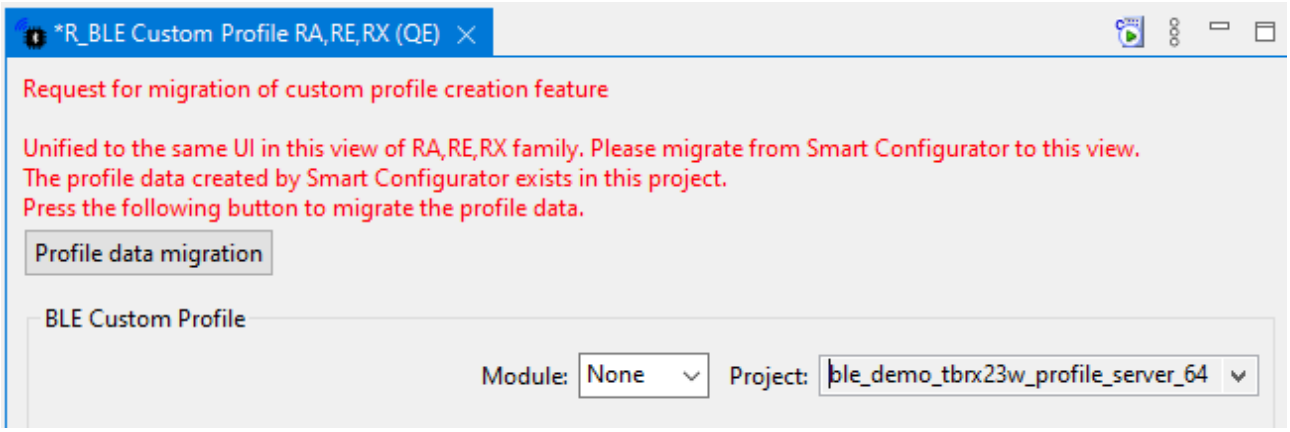

Figure 3.2 Profile updates when using older projects

From the project selection field in the upper right, select the project to which you want to add code.

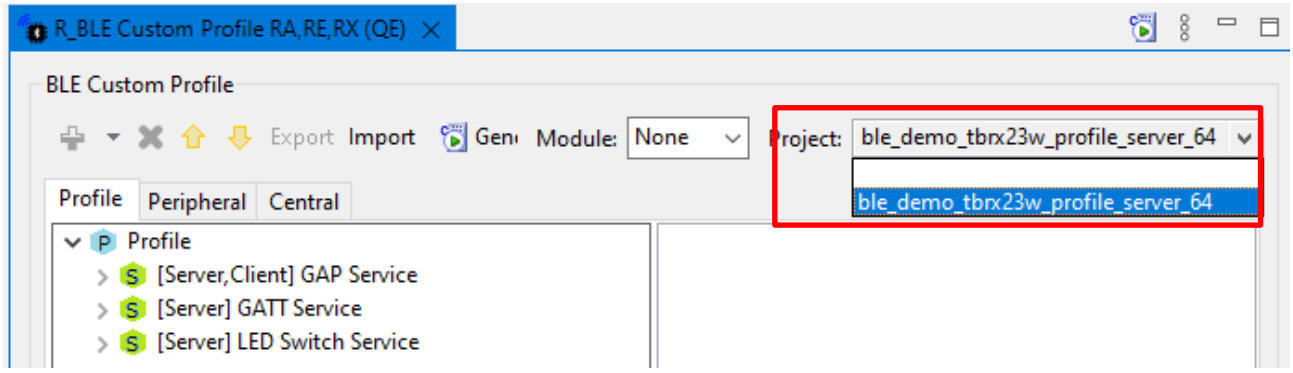

Figure 3.3 Select project.

3.1.2 Develop profile by QE for BLE

Configure settings in three tabs of QE for BLE: Profile, Peripheral, and Central tab.

- Profile tab: set Profiles including Services, Characteristics, and their Descriptors. You can also select GAP role in this tab. See Section 3.2.
- Peripheral tab: set Advertising information for working as a peripheral device. see Section 3.3.
- Central tab: set Scan and Connection information for working as a central device. see Section 3.4.

3.1.3 Code generation

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

The code is generated in the qe_ben / ble folder inside the project folder.

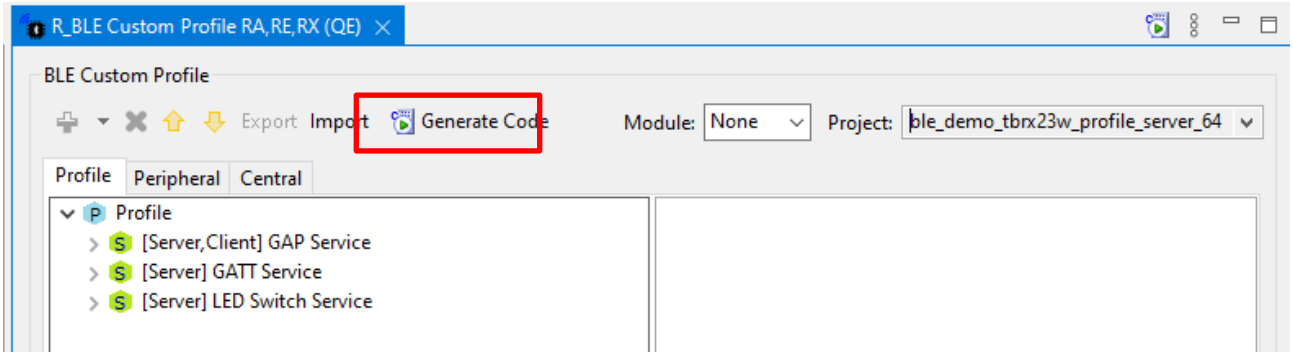

Figure 3.4 Generate Code button

3.1.4 Implement programs.

User application will be created using these generated programs. Refer [4.2 Implementation of custom service] for modifying API program of custom service, and refer [4.3 Implementation of app_main.c] for modifying application framework.

3.2 Configuration of Profile

You can configure profile for user application in Profile tab of QE for BLE. Figure3.5 shows configuration screen of QE for BLE.

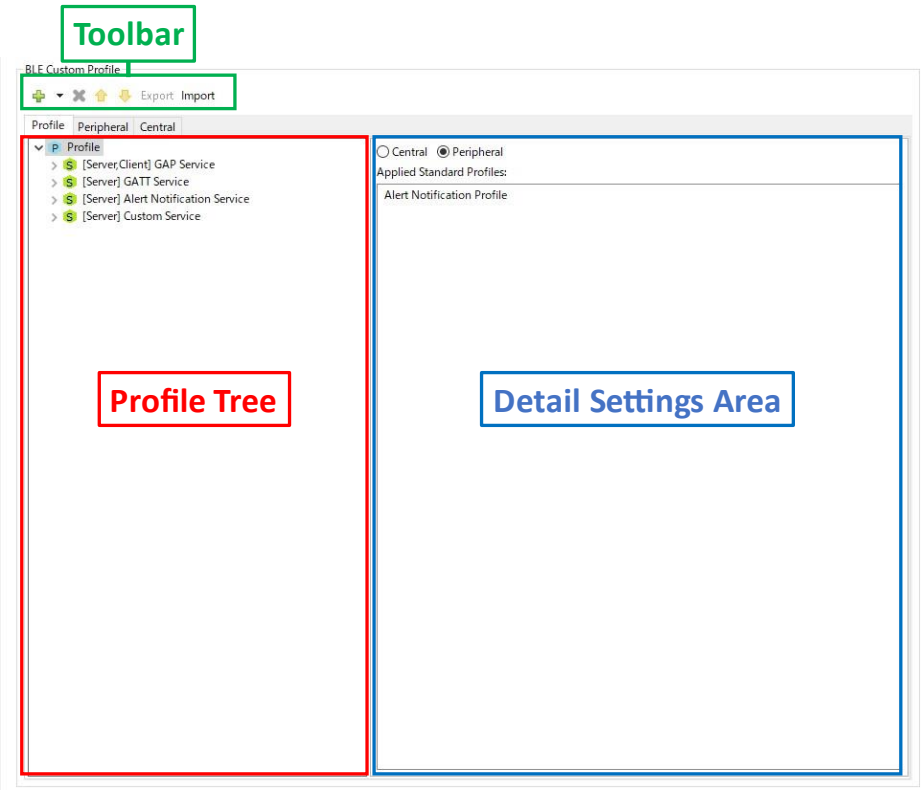

Figure3.5 QE for BLE configuration screen

Profile is a tree-like structure consisting of one or more services, service consisting of one or more characteristics, characteristic consisting of zero or more descriptors. You can check the configuration of the profile that you are currently designing from [Profile Tree].

When you select each element of [Profile Tree], the settings are displayed for each type of elements selected in [Detail Setting Area]. You can configure the functionality of elements added to [Profile Tree] by editing the items displayed in [Detail Setting Area]. Configured profile are stored in the project folder. If you want to save your profile to any folder, you can use [Export] to save it on a per-service basis.

[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 elements to [Profile Tree]. The elements added depends on the elements selected in [Profile Tree]
- [ ] : Deletes selected elements in [Profile Tree].
- [ ][ ] : Moves selected elements in [Profile Tree]. Use this to rearrange elements 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 Addition of elements

You can add elements to [Profile Tree] by clicking [] button in [Toolbar]. The elements that are added will change depending on current elements that are selected in [Profile Tree]. This section describes which elements are added by the selected elements.

You can add services by clicking [] button with the profile selected in [Profile Tree] (Figure3.6). Select [New service] to add custom service or [Add service] to add a SIG adopted service. SIG adopted service that can be added is listed in Table 2.2. If you select [Add profile], you can select adding services by profile. The list of services that configure profile is listed in Table 2.3. Optional services in profile will not be added by selecting profile. For these services, add to [Profile Tree] individually.

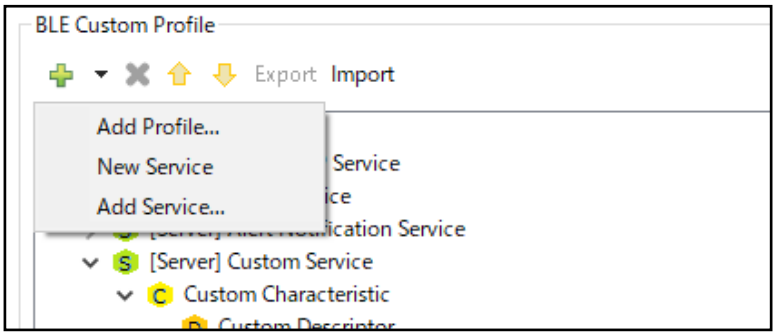

Figure3.6 Adding service

You can add a characteristic by clicking [] button with the service selected in [Profile Tree] (Figure3.7). Select [New Characteristic] to add a custom characteristic or [Add Characteristic] to add SIG adopted characteristic.

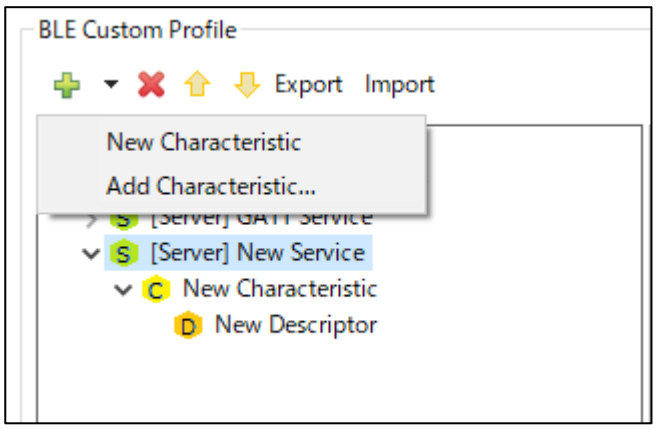

Figure3.7 Adding characteristic

You can add a descriptor by clicking [] button with the characteristic selected in [Profile Tree] (Figure3.8). Select [New Descriptor] to add a custom descriptor or [Add Descriptor] to add SIG adopted descriptor.

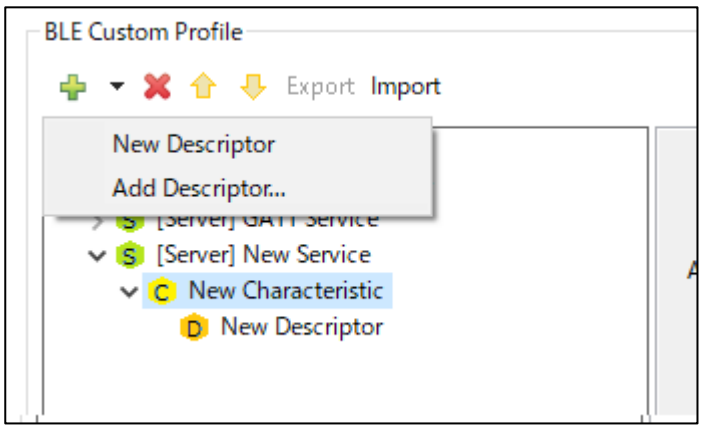

Figure3.8 Adding descriptor

3.2.2 Configuration of profile

When you select profile [P] in [Profile Tree], profile configuration screen (Figure3.9) 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.

[Applied Standard Profiles] shows profiles that applied based on services added to [Profile Tree]. Profile that are shown is listed in Table 2.3.

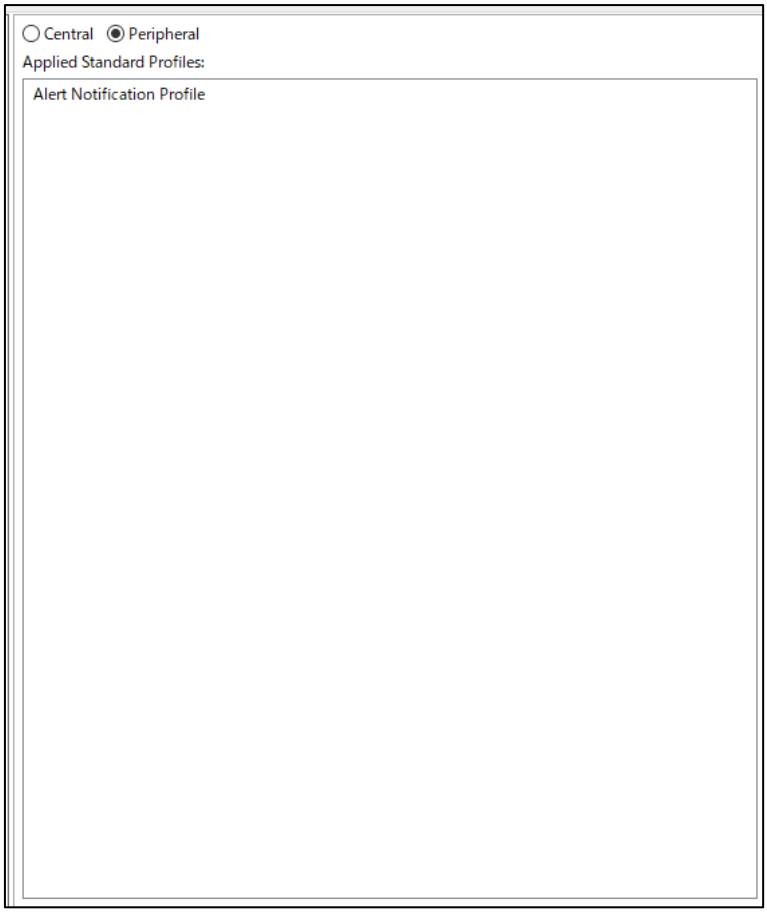

Figure3.9 Profile configuration screen

3.2.3 Configuration of service

When you select service [ ] in [Profile Tree], service configuration screen(Figure3.10) 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.

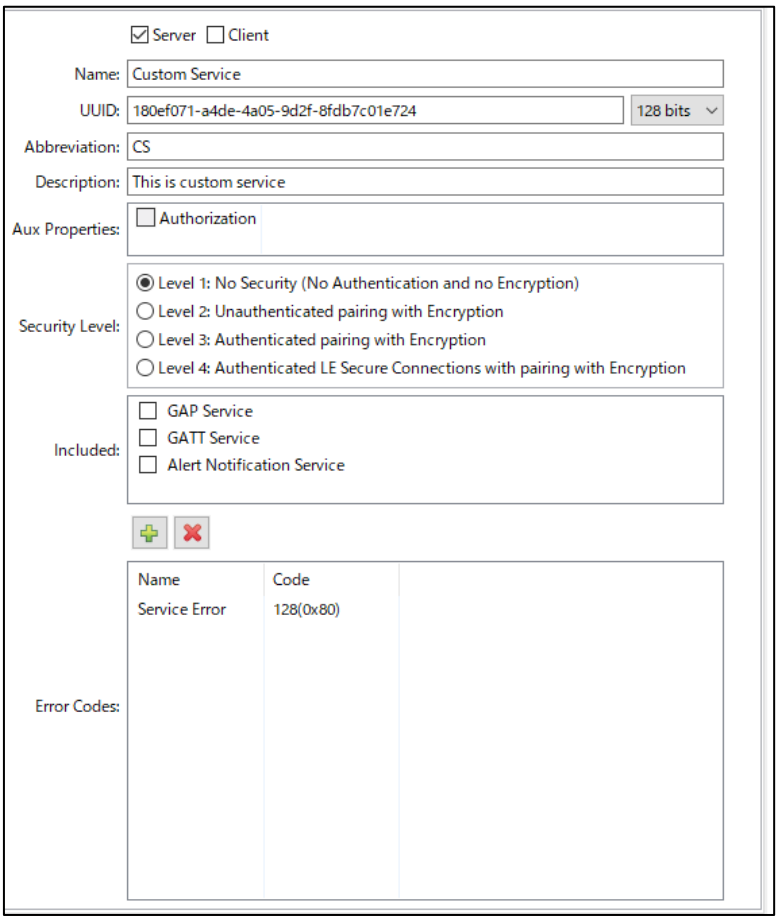

Figure3.10 Service configuration screen

Table 3.1 Service configuration

Item	Description	
Server [optional]	Set check on this item to generate service program as server. It also adds characteristic and descriptors to GATT database.	
Client [optional]	Set check on this item to generate service program as client.	
Name [mandatory]	Name of service. Example) Custom service	
UUID [mandatory]	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 Note: Lack of “0x” or “-” will not affect code generation.	
Abbreviation [mandatory]	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 [optional]	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 [optional]	AUX properties of service. Items below can be configured.	
	Authorization	Enable authorization. Use function R_BLE_GAP_AuthorizeDev( ) to authorize.
Security Level [mandatory]	Security level required for client to access service. Select from below.	
	Level 1: No Security	Client can access service without Pairing and communication will not be encrypted.
	Level 2: Unauthenticated pairing with Encryption	Client can access service after Pairing in Just Works method. Communication will be encrypted.
	Level 3: Authenticated pairing with Encryption	Client can access service after Pairing in authentication method using Passkey Entry or OOB mechanism. Communication will be encrypted.
	Level 4: Authenticated LE Secure Connections with pairing with Encryption	Client can access service after Pairing in LE Secure Connections method. Communication will be encrypted.
Included [optional]	Sets Included service. Select the service to be included from the list.	
Error Codes [optional]	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.

Figure3.11 shows the service configuration screen for SIG adopted service. In this state, only [server], [client], [Aux Properties], [Security Level], 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.

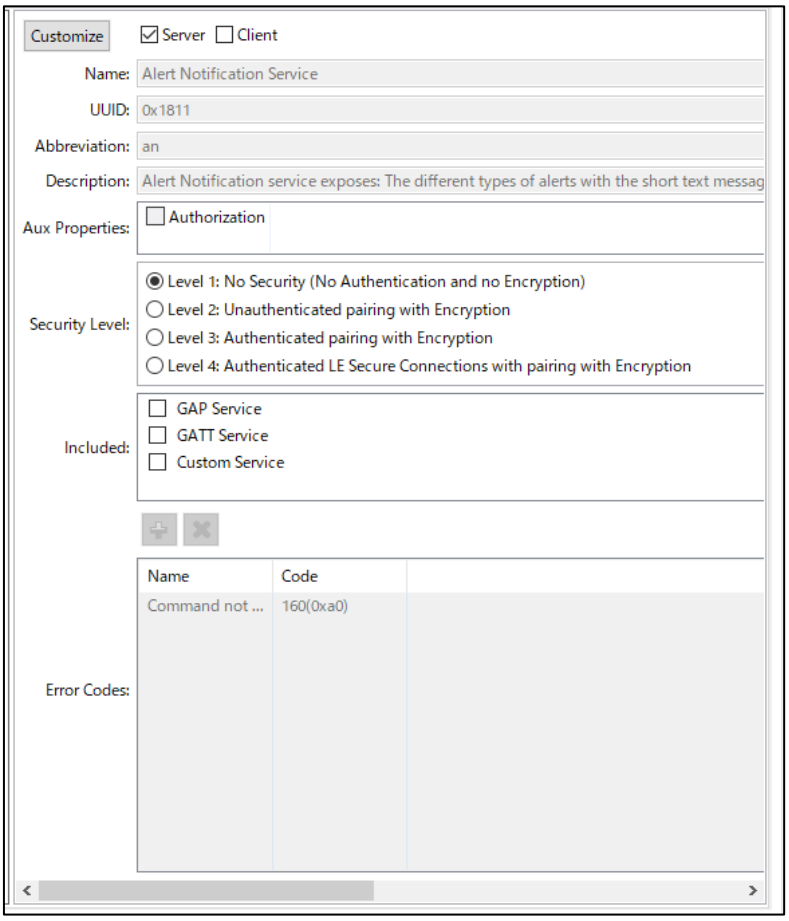

Figure3.11 SIG adopted service configuration screen

3.2.4 Configuration of characteristic

When you select characteristic [C] in [Profile Tree], characteristic configuration screen (Figure3.12) will be shown in [Detail Settings Screen]. Table 3.2 and Table 3.3 describes each item on the configuration screen.

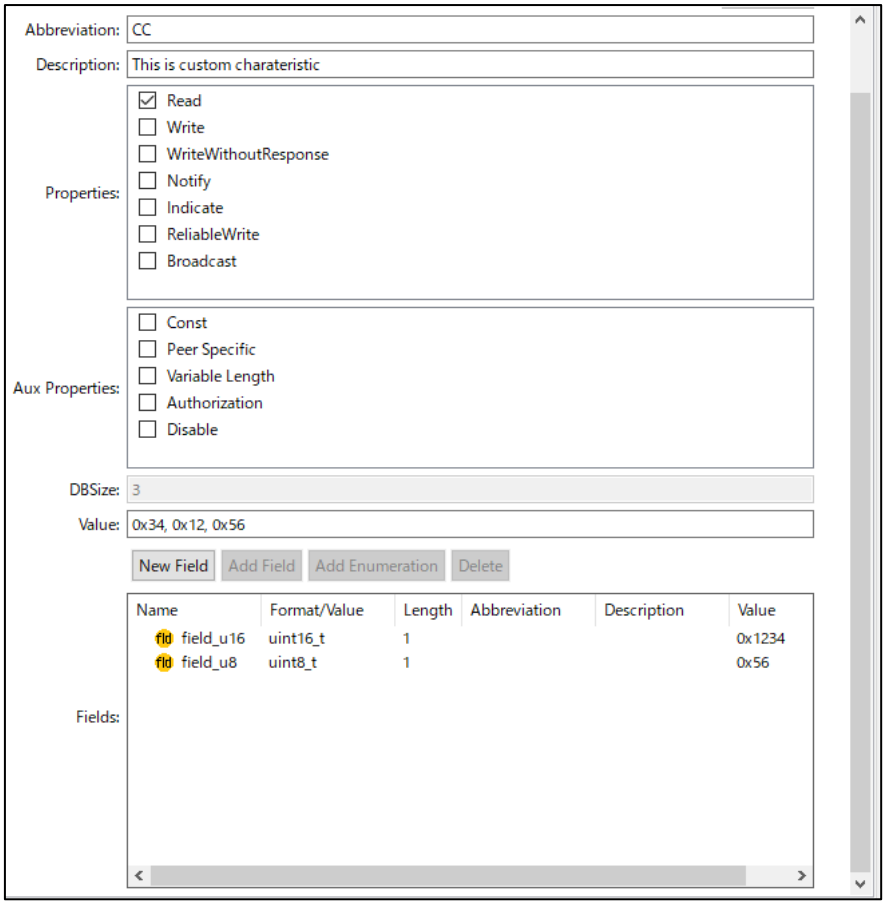

Figure3.12 Characteristic configuration screen

Table 3.2 Characteristic configuration

Item	Description	
Name [mandatory]	Name of characteristic. Example) Custom Characteristic	
UUID [mandatory]	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 Note: Lack of “0x” or “-” will not affect code generation.	
Abbreviation [mandatory]	Abbreviation of characteristic. This value is used in function name and variable name. Beware not to conflict with other characteristics. Example) cc	
Description [optional]	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 [mandatory]	Properties of characteristic which defines operation on Bluetooth LE communication. API and events will be generated for each item checked. [Broadcast] and [ReliableWrite] won’t generate API and events due to its method. Client Characteristic Configuration Descriptor will be added if [Notify] or [Indicate] is selected. Items below can be configured.	
	Read	Enable Read operation.
	Write	Enable Write operation.
	WriteWithoutResponse	Enable Write Without Response operation.
	Notify	Enable Notify operation.
	Indicate	Enable Indicate operation.
	ReliableWrite	Enable Reliable Write operation.
	Broadcast	Enable Broadcast operation.
Aux Properties [optional]	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 [mandatory]	Size of characteristic. Unit of value is byte. Size set in Field will be calculated automatically. If Field with [st_ble_seq_data_t] is set, put maximum size of data.	
Value [optional]	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 [mandatory]	Set value field used in application. Please refer Table 3.3 for configuration.	

Table 3.3 Field configuration

New Field	Add new field. Items below can be configured		
	Name [mandatory]	Name of field. Example) field_name	
	Format/Value [mandatory]	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 Bluetooth LE address data.
		st_ble_seq_data_t	Structure for variable length data. 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 [mandatory]	Data length of field.	
	Abbreviation [optional]	Abbreviation of field.	
	Description [optional]	Description of field. Explain usage if needed	
	Value	Initial value for each field. Value set here will apply to [Value] of descriptor.	
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 [mandatory]	Name of enumeration. Example) enable	
	Format/Value [mandatory]	Value code of enumeration. Example) 0x01	
	Description [optional]	Description of enumeration.	
Delete	Delete selected field.		

3.2.5 Configuration of descriptor

When you select descriptor [D] in [Profile Tree], descriptor configuration screen (Figure3.13) will be shown in [Detail Settings Screen]. Table 3.4 describe each item on the configuration screen.

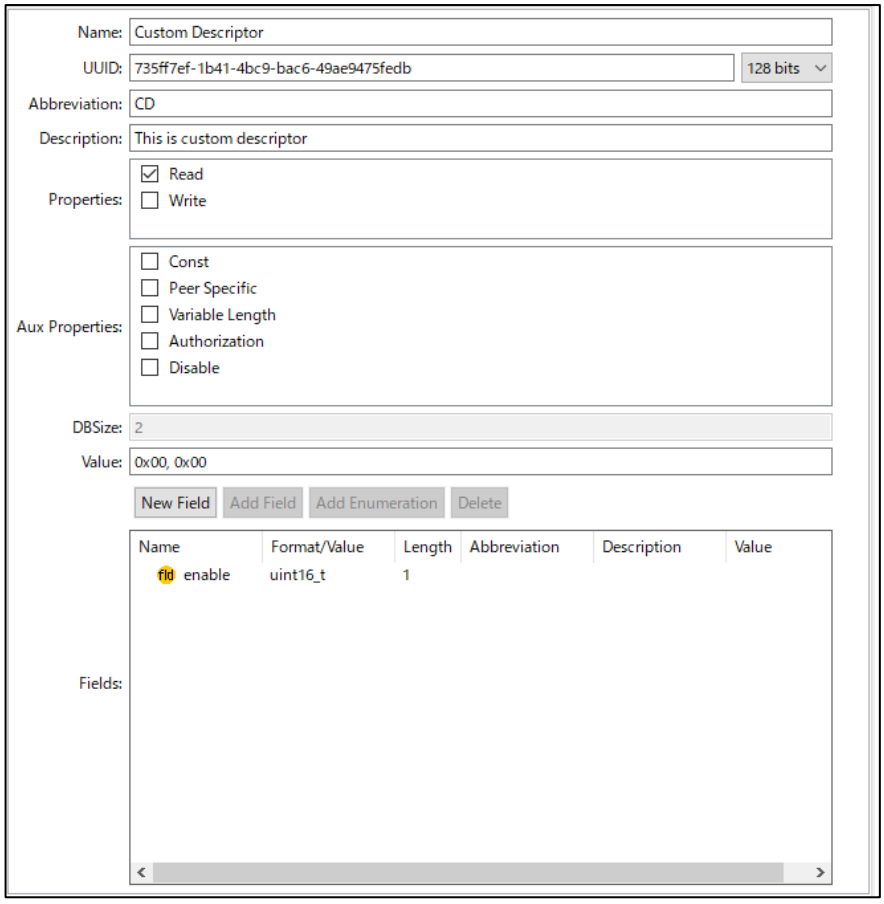

Figure3.13 Descriptor configuration screen

Table 3.4 Descriptor configuration

Item	Description	
Name [mandatory]	Name of descriptor. Example) Custom Descriptor	
UUID [mandatory]	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 Note: Lack of “0x” or “-” will not affect code generation.	
Abbreviation [mandatory]	Abbreviation of descriptor. This value is used in function name and variable name. Beware not to conflict with other descriptors. Example) cd	
Description [optional]	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 [mandatory]	Properties of descriptor which defines operation on Bluetooth LE communication. API and events will be generated for each item checked. Items below can be configured	
	Read	Enable Read operation.
	Write	Enable Write operation.
Aux Properties [optional]	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 [mandatory]	Size of descriptor. Unit of value is byte. Size set in Field will be calculated automatically. If Field with [st_ble_seq_data_t] is set, put maximum size of data.	
Value [optional]	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 [mandatory]	Set value field used in application. Please refer Table 3.3 for configuration.	

3.3 Configuration of peripheral

In [Peripheral] tab, you can configure parameters for GAP peripheral role. Parameters set in this tab are used in application framework when you select [Peripheral] in [Profile] tab. In this tab, you can configure following settings.

Table 3.5 Configurable items in Peripheral

Item	Description
Advertising Data	You can configure Advertising data that will be sent in Advertising event.
Scan Response Data	You can configure Scan response data that will be sent in Advertising event.
Advertising Parameter	You can set parameters for Advertising operation.

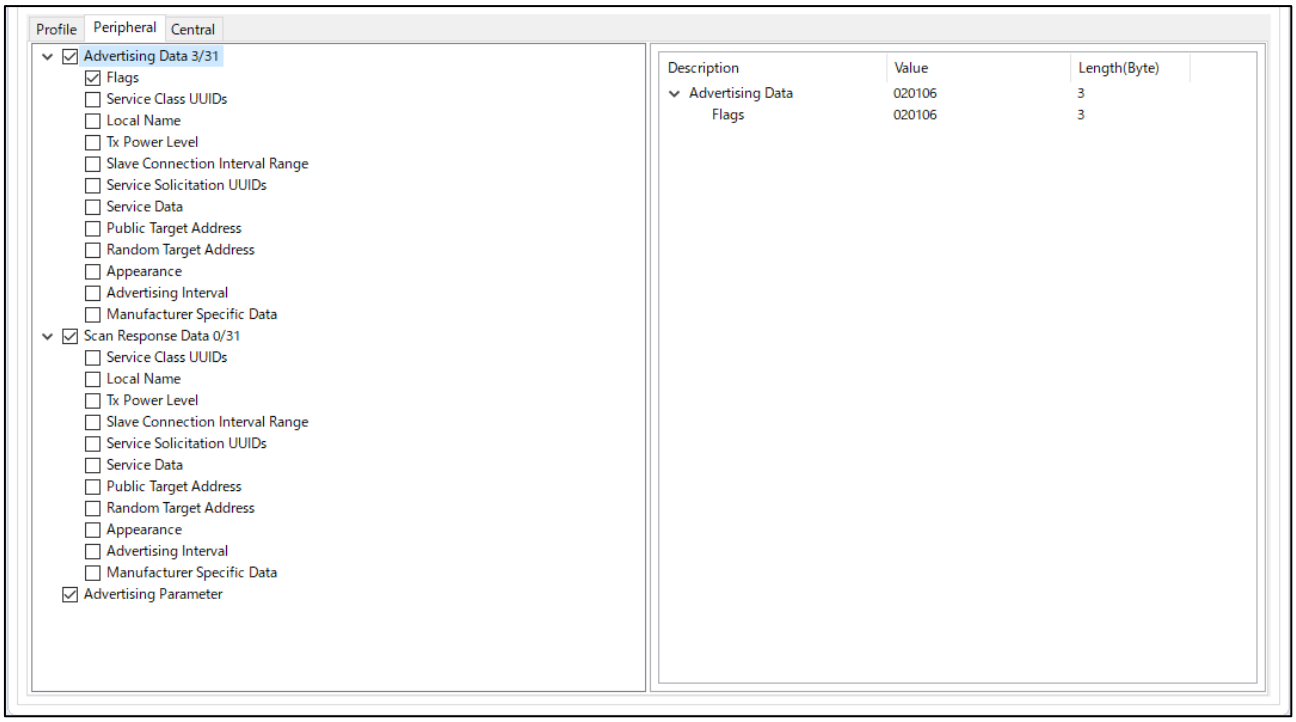

Figure3.14 Peripheral parameter configuration screen

3.3.1 Advertising Data

Advertising Data can be configured by this section. The Data type that are checked will be added as advertising data. User can also input data value by selecting each data type. Data type that user can select is listed in Table 3.6. Maximum size of Advertising data is 31 bytes, so please add data which will not exceed this size. Please configure [3.3.2 Scan Response Data] for additional data. Please refer [Core Specification Supplement <https://www.bluetooth.com>] for detail about Advertising data.

Table 3.6 List of Selectable Data Type

Item	Description	
Flags	This data describes flag of advertising data. Including this data type is necessary for connectable Advertising. This data type can't be selected for scan response data. Select discoverable mode and check for additional information.	
	LE Limited Discoverable Mode	Device will be discoverable for certain period.
	LE General Discoverable Mode	Device will be discoverable all the time.
	Non-Discoverable Mode	Device will not be discovered.
	BR/EDR Not Supported	Check if only Bluetooth LE function is supported.
	Simultaneous LE and BR/EDR to same Device Capable (Controller)	Check if function as Controller roll of Bluetooth LE and BR/EDR can be operated at same time.
	Simultaneous LE and BR/EDR to same Device Capable (Host)	Check if function as Host roll of Bluetooth LE and BR/EDR can be operated at same time.
Service Class UUIDs	This data shows the list of services device offers. You can select services that will be added to the list. Services those are added in Profile tab can be selected.	
Local Name	This data type describes name of advertising device. Select local name type and input the name. Local name can be selected from the below.	
	Short local name	This type describes shortened device name. Please use this type when device name is long and extends the size advertising data
	Complete local name	This type describes complete device name.
TX Power Level	This data type describes TX power of advertising device.	
Slave Connection Interval Range	This data type describes connection interval that is recommended from advertising device. Please input both MAX/MIN of connection interval.	
Service Solicitation UUIDs	This data type shows the list of service that advertising device requires. You can select services that will be added to the list. Services those are added in Profile tab can be selected.	
Service Data	This data type describes data of service. Value of this data type consists of service UUID and service Data. ex) Service UUID [0x1234] Service Data [0x56, 0x78, 0x9a, 0xbc] →Input data [123456789abc]	
Public Target Address	This data type describes Public BD Address of device that are target of advertising data. ex) Public BD Address [0x12:0x34:0x56:0x78:0x9a:0xbc] →Input data [12345678]	
Random Target Address	This data type describes Random BD Address of device that are target of advertising data. ex) Random BD Address [0x12:0x34:0x56:0x78:0x9a:0xbc] →Input data [12345678]	
Appearance	This data type describes appearance of Advertising device. The value of each appearance is listed in Assigned Numbers page in Bluetooth SIG. https://www.bluetooth.com	
Advertising Interval	This data type describes advertising interval of advertising event. The value input in this item will not be used as the advertising parameter.	
Manufacturer Specific Data	This data type describes data that manufacturer specifies by their own. Value of this data type consists of company ID and specific data. ex) Company ID [0x1234] Specific Data [0x56, 0x78, 0x9a, 0xbc] →Input data [341256789abc]	

3.3.2 Scan Response Data

Scan response data can be configured by this section. The data type that are checked will be added as scan response data. User can also input data value by selecting each data type. Data type that user can select is listed in Table 3.6.

3.3.3 Advertising Parameter

You can configure parameters used for Advertising operation. The list of parameters that can be configured

Note: If difficult to connect with default setting, please change parameter of “Slow Advertising Interval”.

Table 3.7 Configurable items of Advertising events

Item	Description	
Fast	You can configure timing information of advertising event. This parameter will be configurable if [Enable Fast Advertising] is checked. If not checked, parameter will be ignored. You can set following items.	
	Advertising Interval	Set Advertising Interval.
	Advertising period	Set Advertising Period. Parameters set in [Fast] will be used for this period.
Slow	You can configure timing information of advertising event. If [Enable Fast Advertising] is checked, this parameter will be used after Fast Advertising period. If not checked, this parameter will be used from the beginning of advertising operation. You can set following items.	
	Advertising Interval	Set Advertising Interval.
	Advertising period	Set Advertising Period. This parameter will be configurable if [Set Advertising Period] is checked. If you want to send Advertising only for certain period, please set this parameter.
Advertising channel	You can select Advertising channel that will be used in Advertising. Advertising event will be sent in all channels that is selected.	
Address type	You can select address type that will be used in advertising. Address type can be selected from below.	
	Public address	Public address will be used in advertising event.
	Random Address	Random address will be used in advertising event. Device static address will be used as BD address.

3.4 Configuration of central

In [Central] tab, you can configure parameters for GAP central role. Parameters set in this tab are used in application framework when you select [central] in [Profile] tab. In this tab, you can configure following settings.

Table 3.8 Configurable items of Central

Item	Description
Scan Parameter	You can set parameters for scan operation such as Scan Interval.
Scan filter data	You can configure Scan filter data that will be used during Scan operation.
Connection Parameter	You can set parameters for Connection such as Advertising Interval. Parameter set here will be used in Connection Request.

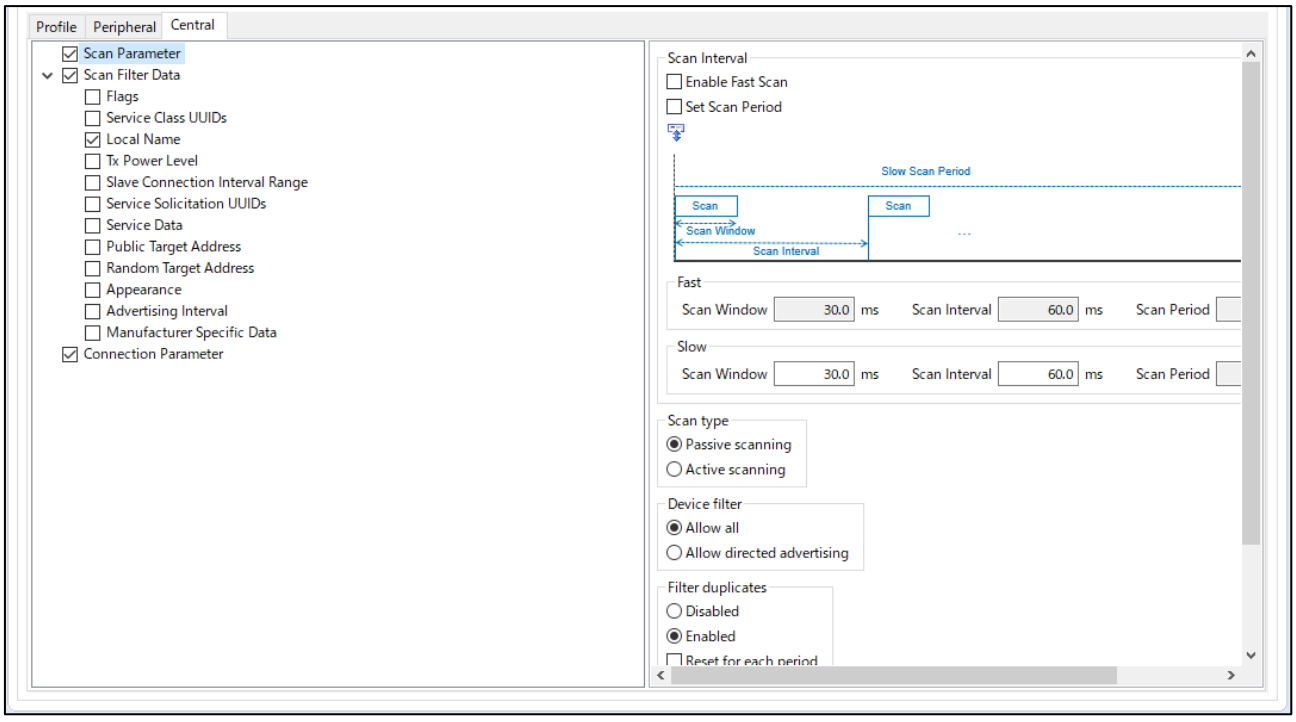

Figure3.15 Central parameter configuration screen

3.4.1 Scan Parameter

You can configure parameters that will be used in scan operation.

Note: If difficult to connect with default setting, please change parameter of “Slow Scan Interval” and “Slow Scan window”.

Table 3.9 Configurable items of Scan Parameter

Item	Description	
Fast	You can configure timing information of scan operation. This parameter will be configurable if [Enable Fast Scan] is checked. If not checked, parameter will be ignored. You can set following items.	
	Scan Window	Set Scan Window.
	Scan Interval	Set scan Interval.
	Scan Period	Set scan Period. Parameters set in [Fast] will be used for this period.
Slow	You can configure timing information of scan operation. If [Enable Fast Scan] is checked, this parameter will be used after Fast scan period. If not checked, this parameter will be used from the beginning. You can set following items.	
	Scan Window	Set Scan Window.
	Scan Interval	Set scan Interval.
	Scan Period	Set scan Period. This parameter will be configurable if [Set Scan Period] is checked. If you want to operate scan only for certain period, please set this parameter.
Scan type	You can select scan type. Scan type can be selected from below.	
	Passive Scanning	Passive scan will operate as scan operation.
	Active Scanning	Active scan will operate as scan operation.
Device filter	You can select device filter that will be used in scan operation. Device filter can be selected from below.	
	Allow all	Scan operation well accept all advertising and scan response PDUs except directed advertising PDUs not addressed to local device.
	Allow directed advertising	Scan operation will accept all advertising and scan response PDUs except directed advertising PDUs whose target address is identity address but doesn't address local device. However, directed advertising PDUs whose target address is the local resolvable private address are accepted.
Filter duplicates	You can select filter duplicate parameter that will be used in scan operation. Filter duplicates can be selected from below.	
	Disable	Duplicate filter will be disabled.
	Enabled	Duplicate filter will be enabled. If you check [Reset for each period], duplicate filter will reset for each scan period.

3.4.2 Scan Filter Data

Filter data for scan operation can be configured by this section. Only advertising event which has data that matches filter data will be notified to the application framework. The data type which is checked will be used as filter data. User can also input data value by selecting data type. Data type that user can select is listed in Table 3.6.

Note: Only one data type can be selected as Scan Filter Data

3.4.3 Connection Parameter

You can configure parameter used for connection event. This parameter will be used in connection request.

Table 3.10 Configurable item of connection

Item	Description	
Parameter	You can configure connection parameter. Parameter set here will be sent with connection request and used after connection established. You can set following items.	
	Connection Interval	Set connection interval.
	Connection Latency	Set slave latency.
	Connection Supervision Timeout	Set supervision timeout.
Connection cancel	You can configure connection cancel parameter. You can set following items.	
	Connection Timeout	Set connection timeout. If peripheral device doesn't respond to connection request for connection timeout, connection will be canceled.

3.5 Notice

3.5.1 Setting to connect between two evaluation boards

If you use two evaluation board, they can establish Bluetooth LE connection each other. For this operation, some configuration must be done in QE for BLE. Following is the points needed to be configured.

- **Set Peripheral for one device and Central for the other device**

Application framework which QE for BLE generates is made to communicate between Peripheral program and Central program. To communicate between two devices, each program must be written in different device.

- **Match Advertising Data and Scan Filter Data**

In Central Application Framework, only the advertising event that has advertising data which matches Scan Filter Data will be found. If advertising event is found, Central Application Framework tries to connect with device which sends advertising event. So, to connect between Central and Peripheral device, Advertising data and scan filter data must match. Preferred Adverting data type to match is “local name”.

- **Set [Enable Fast Advertising/Scan]**

With default setting, parameters which is set in “Slow” will be used for Advertising and Scan operation. This parameter is set to low duty and reduce energy consumption. So, using this parameter may result to be difficult to connect. For high duty, check [Enable Fast Advertising/Scan].

4. Implementation of program

This chapter guides you on how to add user applications to programs generated from QE for BLE. Figure4.1 shows an example of a program generated from QE for BLE.

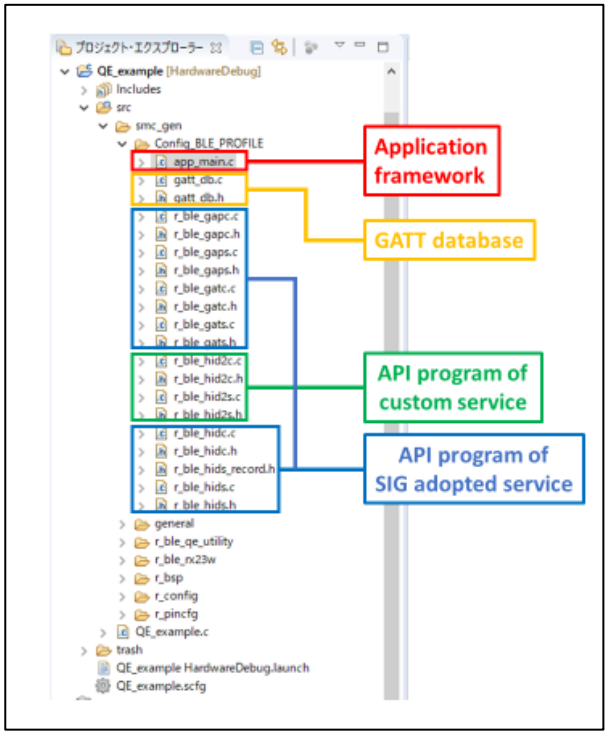

Figure4.1 Program generated from QE for BLE

Programs generated from QE for BLE are newly generated each time QE for BLE is used. If you are using version 1.10 or later of BLE QE Utility module, the application framework implements a comment line with code blocking capabilities to protect user-implemented code (Figure4.2). User can leave code implemented to new application framework generated from QE for BLE by implementing code between this comment line.

```
/* Start user code for XXXX. Do not edit comment generated here */

    Implement user code here

/* End user code. Do not edit comment generated here */
```

Figure4.2 User code blocking comment

When you re-generate code from QE for BLE, programs before regeneration will be copied to [trash] folder in the project (Figure4.3). Therefore, if you cannot add user code between comment line, copy the required code accordingly.

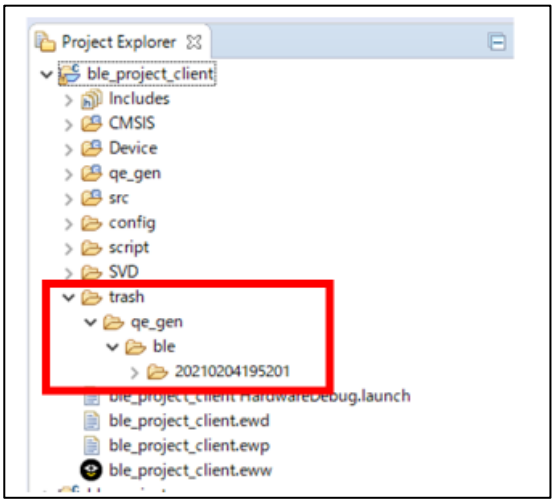

Figure4.3 trash folder

4.1 Service API program

This section will guide you through common specifications in the API program of SIG adopted services and custom service.

4.1.1 Function defined in service API program

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 operation of Bluetooth LE communication which is set to [properties] of characteristic or descriptor.

[operation] section Bluetooth LE operation for characteristic and descriptor will be set. Table 4.1 lists [operation] generated in the server side API program and Table 4.2 lists [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 4.1 Server API

operation	description
Get[characteristic] Get[characteristic][descriptor]	Get characteristic/descriptor value from GATT database. You can check GATT database value changed in write operation.
Set[characteristic] Set[characteristic][descriptor]	Set characteristic/descriptor value to GATT database. Value set in GATT database is used in operation such as read operation.
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 4.2 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 API.
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 4.3, 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 4.3 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.

4.1.2 Event defined in service API program

API program for all services, including custom service, have events defined for sending and receiving data in Bluetooth LE 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.

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

[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.

The string set to [event type] is determined by sending and receiving events in Bluetooth LE communication. The type of event that occurs in Bluetooth LE communication is different on the server side and client side. Table 4.4 lists the events that occur on the server side, and Table 4.5 lists the event that occur on the client side.

Table 4.4 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_CHAR_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 4.5 Client event

event	description
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_CHAR_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

Figure4.4 shows the example of events defined in custom service. In this example, [Switch State (abbreviation: switch_state)] characteristic and [LED Blink Rate (abbreviation: blink_rate)] characteristic from client side of service [LED Switch Service (abbreviation: ls)] is shown. [LED Switch Service] is service used in sample project of BLE FIT module.

```
  /* 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;
```

Figure4.4 Example of event defined in custom service

4.2 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.

4.2.1 Implementing encode/decode function

The application layer handles characteristic and descriptor value in accordance with the format specified by the “Fields” of QE for BLE. On the other hand, the GATT database maintains characteristic and descriptor value in 8-bit data array which size is specified by the [Dbsize] of QE for BLE, and data in the array is sent and received as bit-stream by the BLE Protocol Stack. For this reason, the API program has to convert the value between structured data format for application and 8bit-serialized data for GATT Database and BLE Protocol Stack by using the encode/decode function.

Figure4.5 shows the feature of encode/decode function.

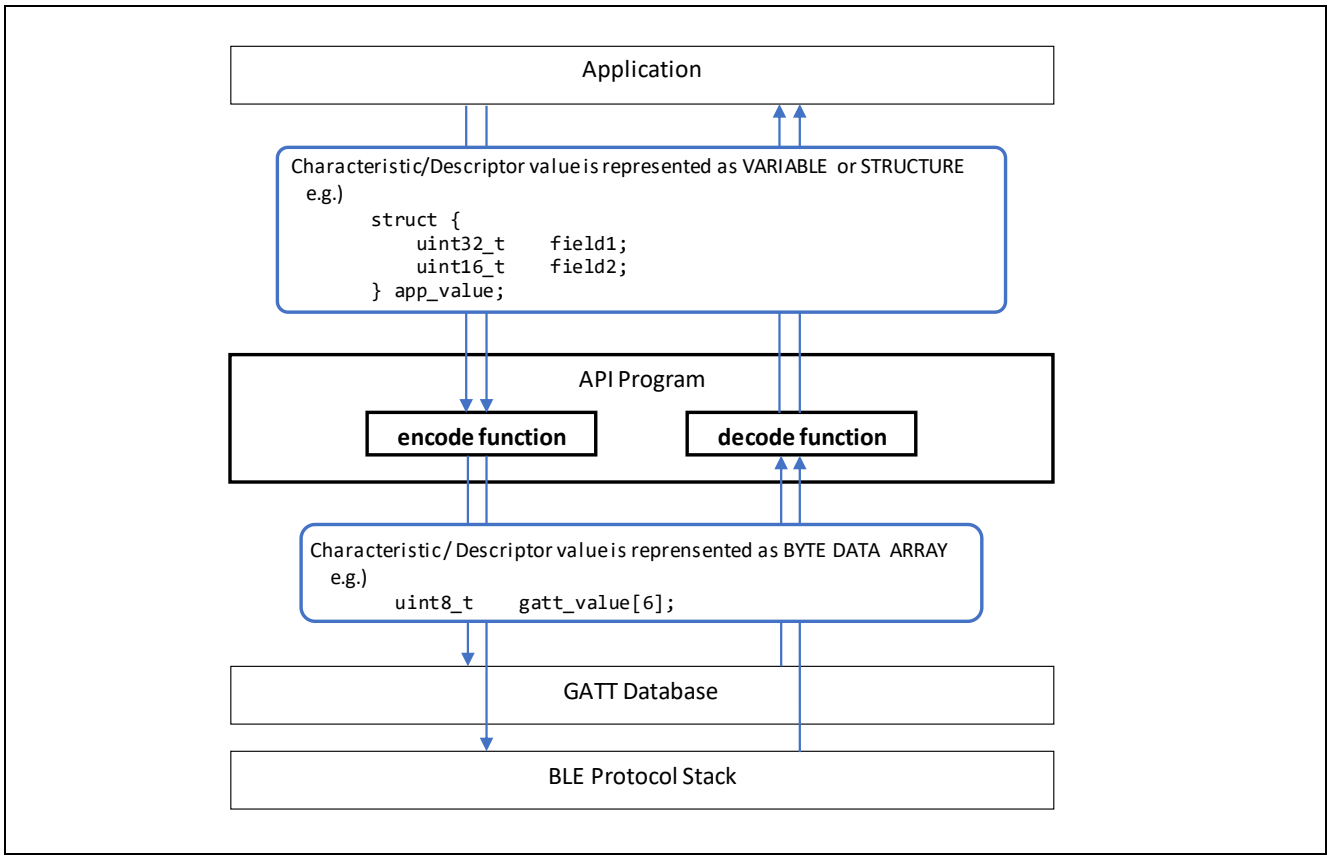

Figure4.5 Feature of encode/decode function

The encode function is used by the API Program when API to send characteristic or descriptor value or to change characteristic or descriptor value of own GATT Database is called. Also, the decode function is used by the API Program before callback function to notify characteristic or descriptor value received.

Figure4.6 shows a use-case of the encode/decode function that GATT Client writes new Characteristic value to peer GATT Server. The encode function is used by API Program of the client side and then the decode function is used by API Program of the server side.

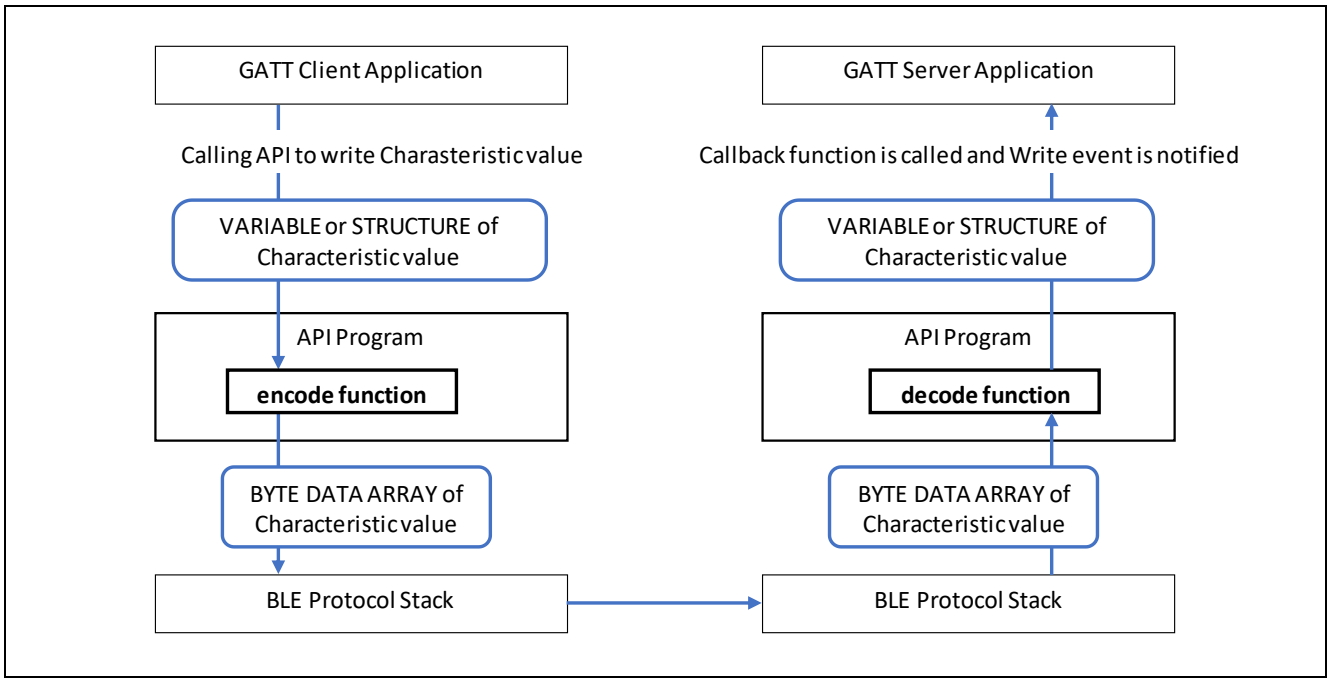

Figure4.6 Use-Case of the encode/decode Function writing Characteristic value

Similarly, Figure4.7 shows a use-case of the encode/decode function that GATT Server notifies new Characteristic value to peer GATT Client. The encode function is used by API Program of the server side and then the decode function is used by API Program of the client side.

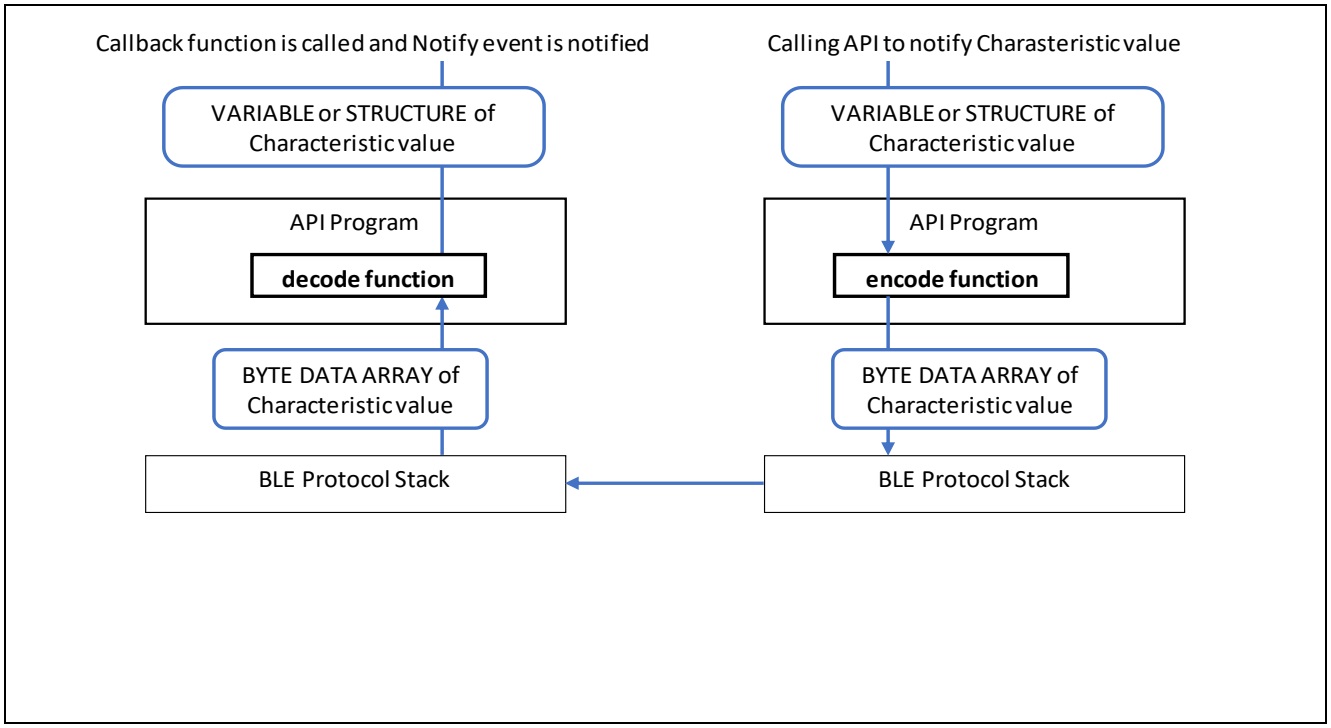

Figure4.7 Use-Case of the encode/decode Function notifying Characteristic value

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 4.6 describes the list of provided encode/decode macros and functions.

Table 4.6 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)

Figure4.8 shows implementation of a encode function for characteristic which has field shown in Figure4.9.In this encode function, encode macros and functions provided in Table 4.6 are used.

```
  typedef struct {
      uint16_t field_u16; /**<field_u16 */
      uint8_t field_u8; /**< field_u8 */
      st_ble_date_time_t field_date; /**< 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;
  }
```

Figure4.8 Example of implementing encode function

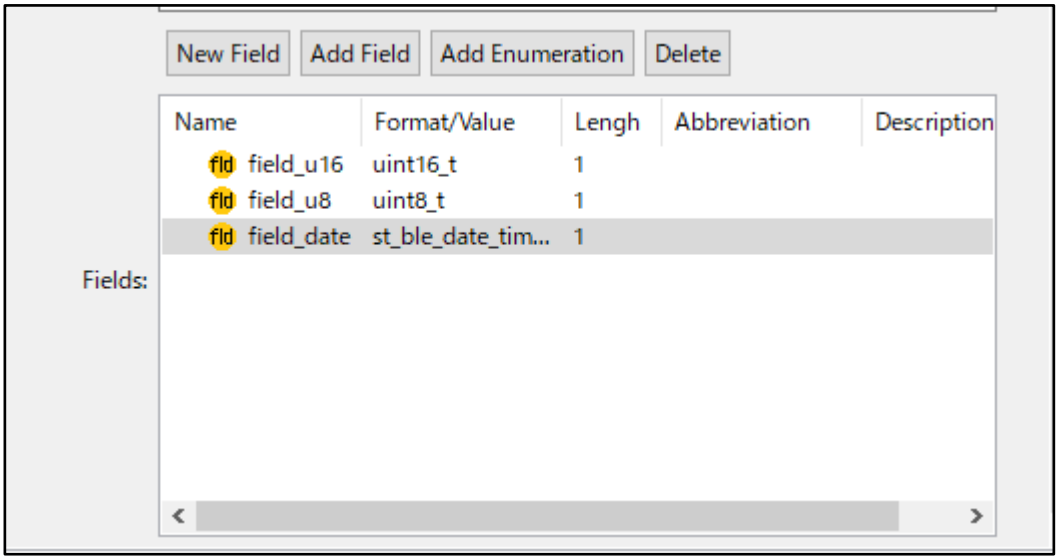

Figure4.9 Example of field

4.2.2 Implementing callback in service API program

Bluetooth LE software generates events when Bluetooth LE 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. For callback in application, refer [4.3.1Implementing callback in application framework]

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 shown in Figure4.10.

```
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),
};
```

Figure4.10 Characteristic structure in service API program

You can create a callback function for a characteristic event in a custom service by editing this structure as shown in Figure4.11.

```
  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,
  };
```

Figure4.11 Implementing callback function in service API program

The callbacks that can be registered are different in server program and client program. Table 4.7 shows callback functions that server program can register and Table 4.8 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 4.7 Callback available for server characteristic

Callback	Event
write_req_cb	This callback 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_cb	This callback 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_cb	This callback 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_cb	This callback 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_cb	This callback occurs when Handle Value Confirmation is received. It is used in Indication operation. GATT event: BLE_GATTS_EVENT_HDL_VAL_CNF
flow_control_cb	This callback occurs when TX flow event is noticed. VS event: BLE_VS_EVENT_TX_FLOW_STATE_CHG

Table 4.8 Callback available for client characteristic

Callback	Event
write_rsp_cb	This callback 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_CHAR_WRITE_RSP BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP
read_rsp_cb	This callback 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_LONG_CHAR_READ_COMP
hdl_val_ntf_cb	This callback occurs when Handle Value Notification is received. It is used in Notification operation. GATT event: BLE_GATTC_EVENT_HDL_VAL_NTF
hdl_val_ind_cb	This callback occurs when Handle Value Indication is received. It is used in Indication operation. GATT event: BLE_GATTC_EVENT_HDL_VAL_IND

Similar to characteristic, each descriptor has structure defined as shown inFigure4.12. 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,
};
```

Figure4.12 Descriptor structure in service API program

Descriptors can register different types of callbacks than characteristics. Table 4.9 shows callbacks that can be registered on the server side, and Table 4.10 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 4.9 Callback available for server descriptor

Callback	Event
write_req_cb	This callback 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_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	This callback 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
read_req_cb	This callback 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_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 4.10 Callback available for client descriptor

Callback	Event
write_rsp_cb	This callback 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_CHAR_WRITE_RSP BLE_GATTC_EVENT_LONG_CHAR_WRITE_RSP
read_rsp_cb	This callback 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_LONG_CHAR_READ_COMP

4.3 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.

4.3.1 Implementing callback in application framework

Bluetooth LE software generates events when Bluetooth LE 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 Bluetooth LE 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 [3. How to implement user code] in Application Note [RX23W Group Bluetooth Low Energy Application Developer's Guide (R01AN5504)].

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. Naming rule of callback function is follows.

[service][s or c]_cb

[service] is the string set to [abbreviation] of the service in QE for BLE. [s or c] is s if the service is set to server, c if the service is set to client.

For information about the events that occur, refer [4.1.2 Event defined in service API program]. Figure4.13 shows an example of implementing a custom service callback function. This example is server side of [LED switch service (abbreviation: ls)] used in sample program in [RX23W Group BLE Module Firmware Integration Technology(R01AN4860)]. The process of updating the software timer when receiving data sent by write data operation from client side is implemented. For software timer, refer [4.1 software timer] in [RX23W Group Bluetooth Low Energy Application Developer's Guide (R01AN5504)].

```
  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;
      }
  }
```

Figure4.13 Callback event of custom service

4.4 Notice

4.4.1 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.

4.4.2 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. Figure4.14 shows how to configure multiple SIG standard services on QE for BLE.

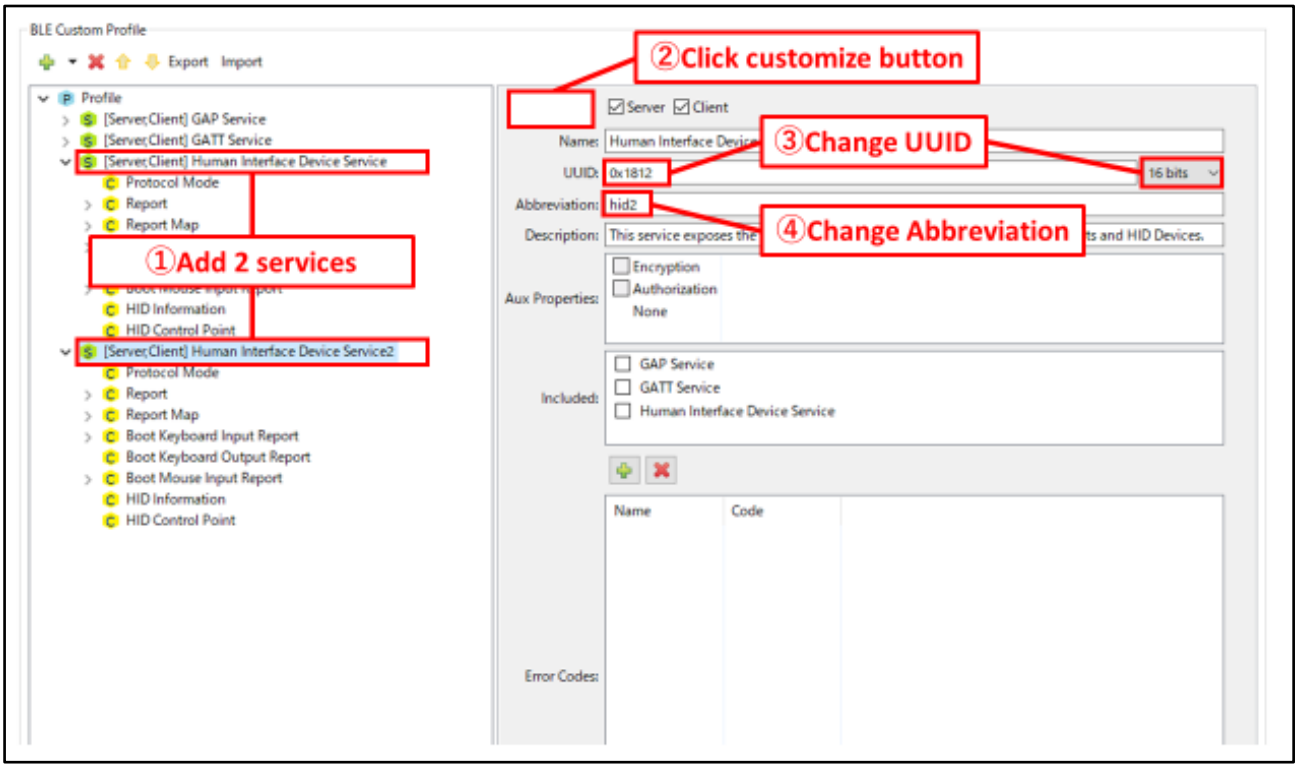

Figure4.14 Configure multiple service on QE for BLE

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 the profile has at least one service selected as a [client] except the GAP service, 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 */
    },
};
```

Figure4.15 Example of implementing 2 HIDSs

4.4.3 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 */

}
```

Figure4.16 GATT database of secondary service (1)

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: Make sure that the service which you changed to secondary service is included from at least one primary 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 */

  }
```

Figure4.17 GATT database of secondary service (2)

Client Side

If the profile has at least one service selected as a [client] except the GAP service, 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. When you need to discovery secondary service, perform discovery operation as the included service because secondary service is included from other primary service, Refer to [4.4.4 Implementation of discovery operation about included service]. When you perform secondary service discovery operation to debug, call R_BLE_GATTC_DiscAllSecondServ() in 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.

4.4.4 Implementation of discovery operation about included service

Specifying included service

If the profile has at least one service selected as a [client] except the GAP service, 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.

```
/*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,
    },
};
```

Figure4.18 Code generated by QE for BLE

```
/*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

     },
};
```

Figure4.19 code modified to discover included service

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 */
  }
```

Figure4.20 Discovery of included service

4.4.5 Guide for Connection Update

In Bluetooth LE communication, you can change the communication frequency during communication by updating the connection.

Connection update can be performed by using function [R_BLE_GAP_UpdConn]. To change frequency of communication, change the following parameters.

- Connection Interval
  - Sets frequency of communication. user can set maximum value and minimum value. Value is calculated by (set value) × 1.25ms.
  - variable: conn_intv_min, conn_intv_max
- Slave latency
  - Ignores communications by the number of value set. If set to 5, communication until the 6th reception will be ignored after first reception.
  - variable: conn_latency
- Supervision Timeout
  - Connection will be disconnected after the time set here. If user want to reduce the frequency of communication, this value needs to be changed accordingly. Value is calculated by (set value) × 10ms.

Figure4.21 shows the example of implementing connection update function in function disc_comp_cb.

```
  static void disc_comp_cb(uint16_t conn_hdl)
  {
  /* Hint: Input process such as GATT operation */
  /* Start user code for Discovery Complete callback function. Do not edit comment
generated here */
       st_ble_gap_conn_param_t conn_param = {
                     .conn_intv_min      = 0x0100,
                     .conn_intv_max      = 0x0100,
                     .conn_latency       = 0x0010,
                     .sup_to             = 0x0200,
                     .min_ce_length      = 0xFFFF,
                     .max_ce_length      = 0xFFFF,
       };
       R_BLE_GAP_UpdConn(conn_hdl, BLE_GAP_CONN_UPD_MODE_REQ, 0x00, conn_param);
  /* End user code. Do not edit comment generated here */
      return;
  }
```

Figure4.21 Example of using Connection Update function

5. Build and Run created profile

5.1 Build and Run for New Project

If you create a new project, the program generated from QE for BLE is generated ate the appropriate location in the project. You can build and run project without changing location of file and directory.

5.2 Build and Run for Sample Project

The BLE FIT module ships sample project using Bluetooth LE function under the name of “FITDemos”. User can import sample project and develop based on them.

5.2.1 When developing based on BLE FIT module version 2.31 or later

Sample project are created in structure that uses QE for BLE. Therefore, user can build and run project without changing location of file and directory.

5.2.2 When developing based on BLE FIT module version 2.31 or earlier

File conflicts occur as the QE for BLE code generation destination folder changes. Please delete the following folders.

- src/smc_gen/Config_BLE_PROFILE

5.2.3 When developing based on BLE FIT module version 1.10 or later

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” generated from QE for BLE → Replace with original “app_main.c”.
- “gatt_db.c” and “gatt_db.h” generated from QE for BLE → Replace with original “gatt_db.c” and “gatt_db.h”.
- “r_ble_[service]” generated from QE for BLE → Copy all files to folder “src”.
- “service” folder of sample project → Delete with all files inside.

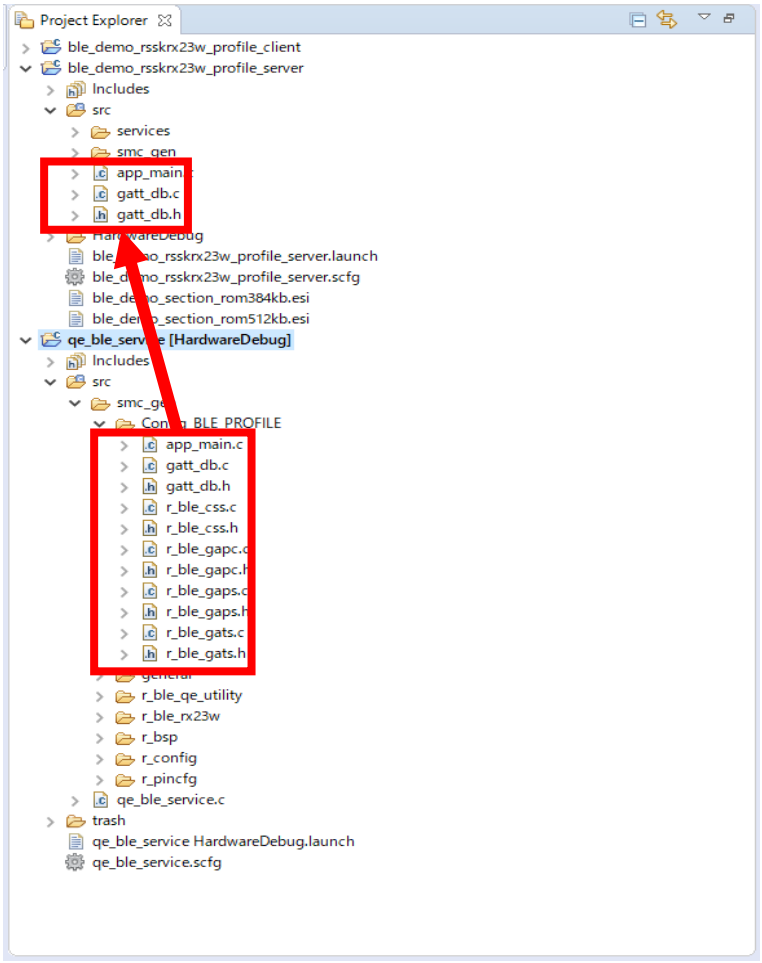

Figure5.1 Copy generated program files to sample program

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();
}
```

Figure5.2 Change function app_main

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov.27.19	—	First edition issued.
1.10	Mar.12.21	1	Added following document to Related Document. ・Bluetooth Low Energy Protocol Stack Basic Package: User's Manual (R01UW0205) ・RX23W Group Bluetooth Low Energy Application Developer’s Guide (R01AN5504)
		60	Added guide for using sample project included in version 1.20 or later BLE FIT module.
		33 22, 25	Added guide for function added in QE Utility V1.10 • Added code blocking function to protect user code. • Deleted “not supported” from Aux Properties of characteristic and descriptor.
		7	Added [2.1 Software Requirements].
		10	Added list of profile in [2.2 QE for BLE]
		13	Added [3.1.2 Download FIT Module].
		15	Added [3.2.1 Addition of elements].
		35	Added [4.1 Service API program]. changed section number of [4.1.1 Function defined in service API program] and [4.1.2 Event defined in service API program].
		59	Added [4.4.5 Using GAT Service and GAP service].
		59	Added [4.4.6 Guide for Connection Update].
1.20	Aug.18.21	26 29 13	Added explanation about QE for BLE[RX] V1.10: • Added [3.3 Configuration of peripheral] • Added [3.4 Configuration of central] • Changed explanation in [3.1.6 Configure QE for BLE]
1.30	Apr.13.22	6	Modified the content according to QE for BLE v1.40
		7	Changed Chapter 2 to QE for BLE v1.40 environment procedure
		12	Changed Chapter 3.1 to QE for BLE v1.40 environment procedure
		14	Change the save destination of the profile to project folder.
		60	Added the case of using the project of BLE FIT 2.30 or earlier.
		-	Deleted chapter "How to use GAT service and GAP service" since it was incorporated in QE for BLE 1.40,

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

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

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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