RA4W1 Group

Virtual UART Sample Application with Simple Connection APIs

Introduction

This application note provides the simple connection APIs which can simply establish Bluetooth Low Energy® connection and data exchange. And provides the virtual UART sample application as an example of simple connection APIs usage.

Target Device

RA4W1 Group

Related Document

Bluetooth Core Specification (https://www.bluetooth.com)

RA4W1 Group User's Manual: Hardware (R01UH0883)

Renesas Flexible Software Package (FSP) User's Manual

Renesas e^2 studio 2021-04 or higher User's Manual: Quick Start Guide (R20UT4989)

EK-RA4W1 Quick Start Guide (R20QS0015)

RA4W1 Group BLE sample application (R01AN5402)

Bluetooth LE Profile API Document User's Manual (R11UM0154)

Related Environments

Refer to section 1.3.

Contents

1. Overview

This application note provides the simple connection APIs which can simply establish Bluetooth Low Energy® connection and data exchange. And provides the virtual UART sample application as an example of simple connection APIs usage.

1.1 Simple connection APIs

Simple connection APIs (Hereafter SC APIs) are wrapper APIs for *RM_BLE_ABS* and *R_BLE* API, which provides as a part of the Flexible software package and QE for BLE. The SC APIs provide a simple way to establish Bluetooth LE connection and GATT-based data exchange. Refer to chapter 2 about specification, usage, and required GATT database for using the APIs.

1.2 Virtual UART sample application

The virtual UART sample application (Hereafter VUART App) is an example project using the SC APIs. The user can learn the usage of the SC APIs from the VUART App. And the user can use the SC APIs by importing the APIs into the user's application project based on the procedures shown in chapter 4. The sample application provides the following functionality.

- Simple AT command function to control and configure Bluetooth LE connection.
- Send/receive characters or binary data to/from a remote device over Bluetooth LE communication.

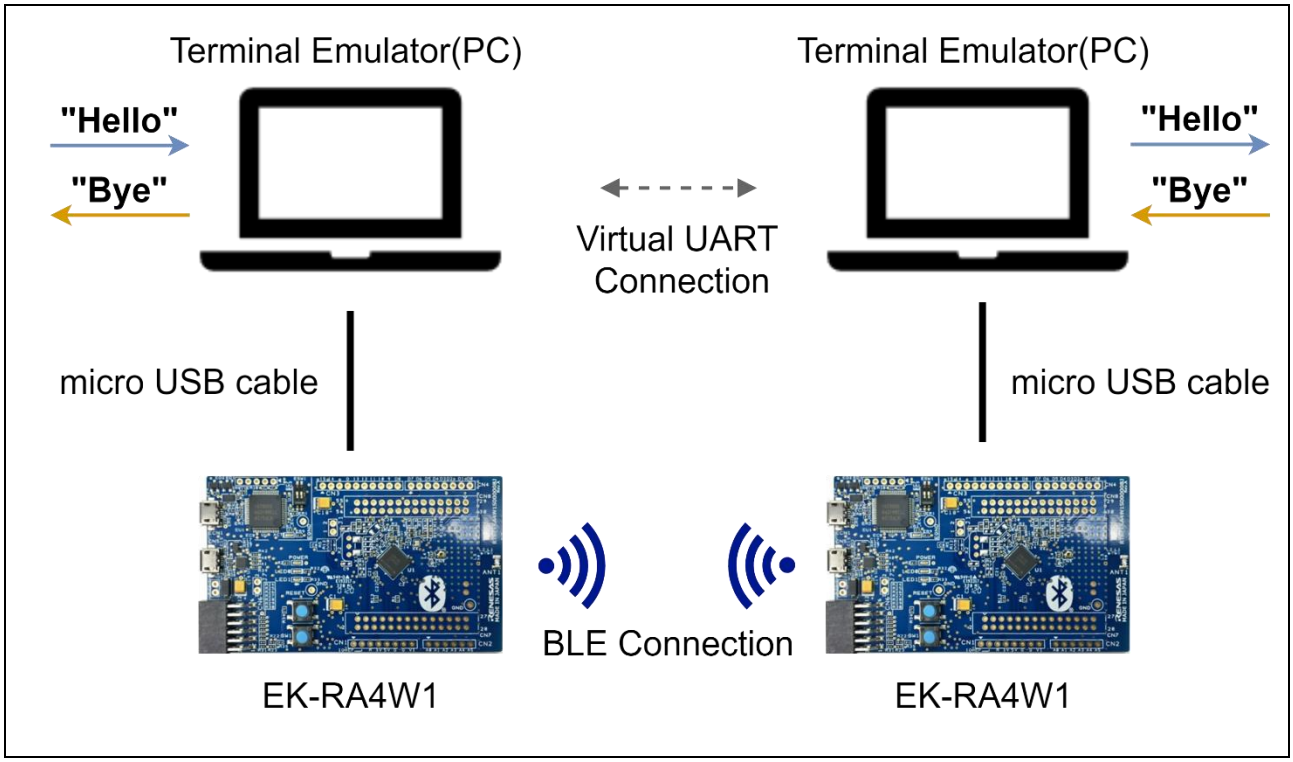

Figure 1. Operating environment of VUART App

This application note includes the following example project.

Table 1. Example projects

Example project	Description
ra4w1_simple_connection_baremetal.zip	Virtual UART sample application for baremetal environment. This application can perform both server and client roles.
ra4w1_simple_connection_freertos.zip	Virtual UART sample application for FreeRTOS environment. This application can perform both server and client roles.
ra4w1_simple_connection_azurertos.zip	Virtual UART sample application for AzureRTOS environment. This application can perform both server and client roles.

1.3 Directory/File Structure

The directory/file Structure of the VUART App is the following.

ra4w1_simple_connection_xxxxx

```
|    configuration.xml                  FSP configuration file
|
+---ra_xxx                              (Files generated by the FSP configurator)
|
\---src                                 VUART App
     |    app_main.c                    VUART App main process
     |    at_command.c                  AT command implementation file
     |
     +---app_lib                        (Application library for CLI)
     |
     \---SimpleConnection_API           Simple Connection API library
          |    simple_connection_api.c  Simple Connection API source
          |    simple_connection_api.h  Simple Connection API header
          |
          +---ble                       (GATT database, profiles)
```

1.4 Operating environments

Table 2 shows the hardware requirements for building and debugging BLE software.

Table 2. Hardware requirements

Hardware	Description
Host PC	Windows® 10 PC with USB interface.
MCU Board	The MCU used must support BLE functions. EK-RA4W1 [RTK7EKA4W1S00000BJ] **Notice** It is necessary to 2 pcs of EK-RA4W1 when confirming data exchange by using example project which listed in Table 1. Refer to *EK-RA4W1 Quick Start Guide (R20QS0015)* about usage of EK-FRA4W1.
On-chip debugging emulators	The EK-RA4W1 has an on-board debugger (J-Link OB). Therefore it is not necessary to prepare an emulator.
USB cables	Used to connect to the MCU board.

Table 3 shows the software requirements for build and debugging BLE software.

Table 3. Software requirements

Software		**Version**	**Description**
GCC environment	e² studio	2022-07	Integrated development environment (IDE) for Renesas devices.
	GCC ARM Embedded	V10.3.1	C/C++ Compiler. (download from e² studio installer)
	Renesas Flexible Software Package (FSP)	V4.0.0	Software package for making applications for the RA microcontroller series.
	SEGGER J-Flash	V7.68b	Tool for programming the on-chip flash memory of microcontrollers.
Integer types			It uses ANSI C99 “Exact width integer types”. These types are defined in stdint.h.
Endian			Little-endian
Terminal emuator			VT-100 compatible (e.g. teraterm)

2. Simple Connection API Specification

This chapter describes the specification of the simple connection APIs (SC APIs).

2.1 API reference

This section describes the *SC APIs* reference. The user can also refer to the Virtual UART sample application, which is listed in Table 1, about the actual usage of these SC APIs.

2.1.1 R_BLE_SC_Init

R_BLE_SC_Init API performs,

- Initialize Bluetooth LE protocol stack included with FSP.
  - Refer to item 5) about services and GATT database structure to use *SC APIs*.
- Register the user callback function.
  - Events that happened as a result of the user using *SC APIs* will be notified to this callback function.
- Start legacy advertising.
  - Will be broadcasted scannable and connectable advertising (*ADV_IND*). For details of advertising, refer to item 6) about detailed advertising parameters.

It is necessary to call this API before using other *SC APIs.*

1) Declaration

```
ble_status_t R_BLE_SC_Init (simple_connection_event_cb_t cb, char *p_DeviceName);
```

Type definition of *simple_connection_event_cb_t* is following.

```
typedef void (*simple_connection_event_cb_t)(uint16_t type,
                                             ble_status_t result,
                                             st_ble_seq_data_t *p_data);
```

Table 4. simple_connection_event_cb_t

Parameter	Direction	Description
type	[in]	The type of Simple Connection API event. Refer to the following sections for the detail of each event.
result	[in]	The result of the Simple Connection API event. Refer to *Renesas Flexible Software Package (FSP) User's Manual* about *ble_status_t* specification.
p_data	[in]	Event data notified by Simple Connection API event. Refer to the following sections for the detail of each event. And refer to *R11UM0154* about *st_ble_seq_data_t* definition.

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 5. Parameters

Parameter	Direction	Description
cb	in	Callback function to be registered with this API.
p_DeviceName	In	This parameter will appear in the local name field of scan response data and the device name characteristic of the GAP service. The length of this parameter should be 11 bytes or less. This parameter is mandatory and should be specified by ASCII characters except for the NULL character.

3) Return values

The return values of this API are the following.

Table 6. Return values

Error Code	Description
BLE_SUCCESS (0x0000)	Success
BLE_ERR_INVALID_PTR (0x0001)	*cb* or *DeviceName* is specified as NULL.
BLE_ERR_INVALID_ARG (0x0003)	The length of *DeviceName* is longer than 11 bytes. *DeviceName* is empty or includes the NULL character.
BLE_ERR_INVALID_OPERATION (0x0009)	Failed to initialize Bluetooth LE stack or profiles.

4) Events

None.

5) Services

In order to exchange data using *SC APIs*, the following service is required.

- **Simple Connection Service**
  - UUID : D68C0001-A21B-11E5-8CB8-0002A5D5C51B

The service includes the following four characteristics.

- **Simple Connection Indication**
  - UUID : D68C0002-A21B-11E5-8CB8-0002A5D5C51B (Indicatable)
- **Simple Connection Write**
  - UUID : D68C0003-A21B-11E5-8CB8-0002A5D5C51B (Writable)
- **Simple Connection Notification**
  - UUID : D68C0004-A21B-11E5-8CB8-0002A5D5C51B (Notifiable)
- **Simple Connection Write Without Response**
  - UUID : D68C0005-A21B-11E5-8CB8-0002A5D5C51B (Write without response)

R_BLE_SC_Init API will register the following GATT database, which includes those mandatory services and characteristics, to the Bluetooth LE stack.

Table 7. GATT database structure

Service	Attribute handle	Attribute type	Properties	Definition	Note
GAP service	0x0001	0x2800	Read	GAP Service Declaration	
	0x0002	0x2803	Read	Device Name characteristic Declaration	
	0x0003	0x2A00	Read/Write	Device Name characteristic Value	
	0x0004	0x2803	Read	Appearance characteristic Declaration	
	0x0005	0x2A01	Read	Appearance characteristic Value	
	0x0006	0x2803	Read	Peripheral Preferred Connection Parameters characteristic Declaration	
	0x0007	0x2A04	Read	Peripheral Preferred Connection Parameters characteristic value	
	0x0008	0x2803	Read	Central Address Resolution Characteristic Declaration	
	0x0009	0x2AA6	Read	Central Address Resolution characteristic value	
	0x000A	0x2803	Read	Resolvable Private Address Only characteristic Declaration	
	0x000B	0x2AC9	Read	Resolvable Private Address Only characteristic value	
GATT service	0x000C	0x2800	Read	GATT Service Declaration	
	0x000D	0x2803	Read	Service Changed characteristic Declaration	
	0x000E	0x2A05	Indication	Service Changed characteristic value	
	0x000F	0x2902	Read/Write	Client Characteristic Configuration descriptor	
Simple connection service	0x0010	0x2800	Read	Simple Connection Service Declaration	UUID : D68C0001-A21B-11E5-8CB8-0002A5D5C51B
	0x0011	0x2803	Read	Simple Connection Indication Characteristic Declaration	UUID : D68C0002-A21B-11E5-8CB8-0002A5D5C51B Indication (Server -> Client)
	0x0012	See Note	Read/Indication	Simple Connection Indication characteristic value	Attribute value length = 244 octets.
	0x0013	0x2902	Read/Write	Client Characteristic Configuration descriptor	

Service	Attribute handle	Attribute type	Properties	Definition	Note
	0x0014	0x2803	Read	Simple Connection Write Characteristic Declaration	UUID : D68C0003-A21B-11E5-8CB8-0002A5D5C51B Write (Client->Server)
	0x0015	See Note	Read/Write	Simple Connection Write characteristic value	Attribute value length = 244 octets.
	0x0016	0x2803	Read	Simple Connection Notification Characteristic Declaration	UUID : D68C0004-A21B-11E5-8CB8-0002A5D5C51B Notification (Server->Client)
	0x0017	See Note	Read/Notification	Simple Connection Notification characteristic value	Attribute value length = 244 octets.
	0x0018	0x2902	Read/Write	Client Characteristic Configuration descriptor	
	0x0019	0x2803	Read	Simple connection Write without Response characteristic Declaration	UUID : D68C0005-A21B-11E5-8CB8-0002A5D5C51B Write without response (Client->Server)
	0x001A	See Note	Read/Write without Response	Simple Connection Write without Response characteristic value	Attribute value length = 244 octets.

6) Advertising parameters and data structure

Scannable and connectable advertising (*ADV_IND*) PDU will be broadcasted after calling this API. The advertising parameters and advertising data structure are shown in Table 8. Advertising data and scan response data follow the *AD structure* format defined by *Bluetooth Core Specification.*

Table 8. Advertising parameters

Parameter	Value
Interval	20 [msec]
Channel Map	All channels (Ch.37, 38, 39) were used.
Own Bluetooth address type	The user can choose a public address or a static random address by configuring the *R_BLE_SC_ADDRESS_IN_USE* macro. Refer to section 2.2 about the macro specification.
Own Bluetooth address	One of the following will be used: • The identity address configured by R_BLE_SC_SetAddress API • The properties of the *BLE_Driver* FSP module, which describes in the FSP configuration Refer to section 2.1.5 about *R_BLE_SC_SetAddress* API and R01AN5402 section4.2.4 about FSP configuration.
Advertising Type	Scannable and connectable (*ADV_IND*)
Advertising data	Includes, • Simple connection service UUID D68C0001-A21B-11E5-8CB8-0002A5D5C51B
Scan response data	Includes • Simple connection service UUID D68C0001-A21B-11E5-8CB8-0002A5D5C51B • Complete local name Specified *p_Devicename*.

7) Note

- This API is blocking API. Therefore, this API will be occupied the MCU until the initialization is completed. And the user should not use this API in the interrupt handler.
- *R_BLE_Execute* and SC blocking APIs should not call in the user callback function to avoid a deadlock condition. See the note in the following section for which APIs are classified as blocking APIs.

2.1.2 R_BLE_SC_Scan

R_BLE_SC_Scan API performs an active scan and finds advertising PDU, which includes Simple Connection Service UUID (D68C0001-A21B-11E5-8CB8-0002A5D5C51B) as advertising/scan response data. Refer to item 2) about scan parameters. And the API returns the BD addresses and the device name of the discovered device(s).

1) Declaration

```
R_BLE_SC_Scan (uint8_t scan_device_limit, st_simple_find_device_t * p_Devices)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 9. Parameters

Direction	Parameter Name	Description
[in]	scan_device_limit	The number of the device to discover. When the number of the discovered device reaches this parameter, the scan will terminate. This parameter should be less than the *R_BLE_SC_DEV_MAX* macro.
[out]	p_Devices	BD address and device name of discovered devices.

3) Return values

The return values of this API are the following.

Table 10. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	*p_Devices* pointer is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	*scan_device_limit* is larger than the *R_BLE_SC_DEV_MAX* macro or specified as zero.
BLE_ERR_INVALID_STATE(0x0008)	The BLE stack is not running.
BLE_ERR_INVALID_OPERATION(0x0009)	The API has called in an interrupt handler or an event callback.
BLE_ERR_UNSPECIFIED(0x0013)	Other errors.

4) Events

None.

5) Scan parameters

This API will be performed the scan procedure according to the following parameters.

Table 11. Scan parameters

Parameter	Value
Scan interval	60 [msec]
Scan window	30 [msec]
Scan period	7.68 [sec] • This API will be occupied the MCU until 7.68 seconds after starting the scan.
Scan type	Active scanning
Scan filter	Simple connection service UUID (D68C0001-A21B-11E5-8CB8-0002A5D5C51B). • Find advertising PDU which includes Simple Connection Service UUID (D68C0001-A21B-11E5-8CB8-0002A5D5C51B) as advertising / scan response data.
Duplicate filter	Yes

6) Note

- This API is blocking API. Therefore, this API will be occupied the MCU until 7.68 seconds after starting the scan. And the user should not use this API in the interrupt handler and the callback function specified in *R_BLE_AC_Init* API.
- This API cannot use when choosing *compact* as *BLE_Driver* FSP module configuration.

2.1.3 R_BLE_SC_Connection

R_BLE_SC_Connection API performs,

- Establish a Connection with another device.
  - Refer to item 5) about connection parameters.
  - This API does not support connections with multiple devices.
- Stop legacy advertising that started when executing *R_BLE_SC_Init* API.
- Perform service discovery.
- Perform pairing.
- Enable Notification and Indication by configuring *Simple Connection Indication Characteristic Declaration* characteristic (D68C0002-A21B-11E5-8CB8-0002A5D5C51B) and *Simple Connection Notification Characteristic Declaration* characteristic (D68C0004-A21B-11E5-8CB8-0002A5D5C51B).

1) Declaration

```
ble_status_t R_BLE_SC_Connection (uint8_t type, uint8_t *p_address, uint16_t *p_conn_hdl)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 12. Parameters

Direction	Parameter Name	Description
[in]	type	The address type of target BD address. *BLE_GAP_ADDR_PUBLIC* (0x00) : Public address *BLE_GAP_ADDR_RAND* (0x01) : Static random address
[in]	p_address	The target BD address. The value specified by the *R_BLE_SetConnectionAddress* API will be used when All zero value is passed.
[out]	p_conn_hdl	The connection handle.

3) Return values

The return values of this API are the following.

Table 13. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	*p_address* or *p_conn_hdl* is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The Target BD address is invalid, or the type of target address is out of range.
BLE_ERR_INVALID_STATE(0x0008)	Already establish a connection with another device or BLE stack is not running.
BLE_ERR_INVALID_OPERATION(0x0009)	The API has called in an interrupt handler or an event callback.
BLE_ERR_CONTEXT_FULL(0x000B)	The API call could not be processed because the Host Stack queue was full.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Memory cannot allocate to proceed with this API.
BLE_ERR_NOT_FOUND(0x000D)	Encryption failed because the remote device had lost the LTK.
BLE_ERR_RSP_TIMEOUT(0x0011)	The connection has been canceled due to a connection procedure timeout.
BLE_ERR_UNSPECIFIED(0x0013)	Other error.

4) Events

This API will notify the following events to the callback function registered by the *R_BLE_SC_Init* API.

Table 14. Events

Event Code	Description
R_BLE_SCX_EVENT_CONN_IND (0x0041)	This event will be notified when, 1) Enabled indication by *Simple Connection Indication Characteristic Declaration* characteristic (D68C0002-A21B-11E5-8CB8-0002A5D5C51B). 2) Enabled notification by *Simple Connection Notification Characteristic Declaration* characteristic (D68C0004-A21B-11E5-8CB8-0002A5D5C51B). • **result** The *result* will be notified as *ble_status_t* type. Refer to *Renesas Flexible Software Package (FSP) User's Manual* about *ble_status_t* specification. • **Event Data** The *Event Data* will be notified as *st_ble_seq_t* type. In the case of items 1) and 2), the *data* member of Event Data includes configured CCCD value. Refer to *R11UM0154* about *st_ble_seq_data_t* definition.

5) Connection parameters

This API will be established a connection with another device according to the following parameters.

Table 15. Connection parameters

Parameter	Value
Connection Interval	Default value is 7.5 [msec]. the user can configure the connection interval by using *R_BLE_SC_SetConnectionInterval* API.
Supervision timeout	1,193 [msec]
Slave latency	0
Connection PHY	The default configuration is LE 1M PHY. The user can configure PHY by using *R_BLE_SC_SetgPHY()* API after a connection is established.
Own Bluetooth address type	The user can choose a public address or a static random address by using the *R_BLE_SC_ADDRESS_IN_USE* macro. Refer to section 2.2 about the macro specification.
Own Bluetooth address	One of the following will be used: • The identity address configured by *R_BLE_SC_SetAddress* API • The properties of the *BLE_Driver* FSP module, which describes in the FSP configuration Refer to section 2.1.5 about *R_BLE_SC_SetAddress* API and R01AN5402 section4.2.4 about FSP configuration.
Connection Timeout	10 [sec]. • If connectable advertising PDUs cannot find during this period, the connection request will be canceled.

6) Note

- This API is blocking API. Therefore, this API will be occupied the MCU until the connection has been completed or canceled due to *Connection Timeout,* which describes in item 5). And the user should not use this API in the interrupt handler and the callback function specified in *R_BLE_AC_Init* API.
- This API cannot use when choosing *compact* as *BLE_Driver* FSP module configuration.
- The address type and identity address of the connection target device will be specified by the argument of this API or *R_BLE_SC_SetConnectionAddress* API. Refer to section 2.1.7.

2.1.4 R_BLE_SC_DisConnection

R_BLE_SC_DisConnection API disconnects the current connection. After successfully disconnecting the connection, this API will automatically start legacy advertising. The restarted advertising parameter is based on Table 8.

1) Declaration

```
ble_status_t R_BLE_SC_DisConnection (uint16_t conn_hdl)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 16. Parameters

Direction	Parameter Name	Description
[in]	conn_hdl	Connection handle.

3) Return values

The return values of this API are the following.

Table 17. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The connection handle passed as the argument is invalid.
BLE_ERR_INVALID_STATE(0x0008)	The connection is not established, or the host stack is not running.
BLE_ERR_INVALID_OPERATION(0x0009)	The API has called in an interrupt handler or an event callback.
BLE_ERR_CONTEXT_FULL(0x000B)	The API call could not be processed because the Host Stack queue was full.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Memory cannot allocate to proceed with this API.

4) Events

This API will notify the following events to the callback function registered by the *R_BLE_SC_Init* API.

Table 18. Events

Event Code	Description
R_BLE_SCX_EVENT_DISCONN_IND (0x0042)	The connection has been disconnected. The following result code and event data will be notified to the callback function when this event has happened. • **result** The *result* will be notified as *ble_status_t* type. Refer to *Renesas Flexible Software Package (FSP) User's Manual* about *ble_status_t* specification. • **Event Data** None.

5) Note

- This API is blocking API. Therefore, this API will be occupied the MCU until the connection has been disconnected. And the user should not use this API in the interrupt handler and callback function specified in *R_BLE_AC_Init* API.
- This API cannot use when choosing *compact* as *BLE_Driver* FSP module configuration.
- The reason for disconnection is always 0x0013 (Remote User Terminated Connection).

2.1.5 R_BLE_SC_SetAddress

R_BLE_SC_SetAddress API configures the own identity address. The store destination of the identity address has been configured by the *Device Specific Data Flash Block* property of the *BLE_Driver* FSP module, which describes in the FSP configuration.

- Store in the volatile area
  - This API will restart the Host stack to activate the new address.

- Store in the non-volatile area
  - Immediately reflect the new address.

Refer to section 4.2.4 in R01AN5402 about identity address adaptation flow.

1) Declaration

```
ble_status_t R_BLE_SC_SetAddress (uint8_t type, uint8_t *p_address)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 19. Parameters

Direction	Parameter Name	Description
[in]	type	Identity address type. *BLE_GAP_ADDR_PUBLIC* (0x00) : Public address *BLE_GAP_ADDR_RAND* (0x01) : Static random address
[in]	p_address	Identity address. The length of the identity address should be 6 octets.

3) Return values

The return values of this API are the following.

Table 20. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	*p_address* is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The *type* parameter is out of range.
BLE_ERR_INVALID_STATE(0x008)	The host stack is not running or has already established a connection with another device.
BLE_ERR_INVALID_OPERATION(0x0009)	Failed to configure identity address, or the API has called in an interrupt handler or an event callback.
BLE_ERR_CONTEXT_FULL(0x000B)	The API call could not be processed because the Host Stack queue was full.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Memory cannot allocate to proceed with this API.

4) Events

None.

5) Note

- This API is blocking API. Therefore, this API will be occupied the MCU until the identity address has been configured. And the user should not use this API in the interrupt handler and the callback function specified in *R_BLE_AC_Init* API.
- This API cannot be used while a connection with another device exists..

2.1.6 R_BLE_SC_GetAddress

R_BLE_SC_GetAddress API gets the current own identity address.

1) Declaration

```
ble_status_t R_BLE_SC_GetAddress (uint8_t type, uint8_t *p_address)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 21. Parameters

Direction	Parameter Name	Description
[in]	type	Identity address type. *BLE_GAP_ADDR_PUBLIC* (0x00) : Public address *BLE_GAP_ADDR_RAND* (0x01) : Static random address
[out]	p_address	Pointer for identity address.

3) Return values

The return values of this API are the following.

Table 22. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	*p_address* is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The *type* parameter is out of range.
BLE_ERR_INVALID_STATE(0x008)	The host stack is not running or has already established a connection with another device.
BLE_ERR_INVALID_OPERATION(0x0009)	Failed to configure identity address, or the API has called in an interrupt handler or an event callback.
BLE_ERR_CONTEXT_FULL(0x000B)	The API call could not be processed because the Host Stack queue was full.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Memory cannot allocate to proceed with this API.

4) Events

None.

5) Note

This API is blocking API. Therefore, this API will be occupied the MCU until the identity address has been configured. And the user should not use this API in the interrupt handler and callback function specified in *R_BLE_AC_Init* API.

2.1.7 R_BLE_SC_SetConnectionAddress

R_BLE_SC_SetConnectionAddress API configures the connection target BD address.

The target address will be used to establish a connection with another device by using *R_BLE_SC_Connection* API.

1) Declaration

```
ble_status_t R_BLE_SC_SetConnectionAddress (uint8_t type, uint8_t * p_address)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 23. Parameters

Direction	Parameter Name	Description
[in]	type	Identity address type. *BLE_GAP_ADDR_PUBLIC* (0x00) : Public address *BLE_GAP_ADDR_RAND* (0x01) : Static random address
[in]	p_address	Identity address. The length of the identity address should be 6 octets.

3) Return values

The return values of this API are the following.

Table 24. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	*p_address* is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The *type* parameter is out of range.
BLE_ERR_INVALID_STATE(0x008)	Already establish a connection with another device.

4) Events

None.

5) Note

- This API cannot be used while a connection with another device exists.

2.1.8 R_BLE_SC_GetConnectionAddress

R_BLE_SC_GetConnectionAddress API gets the current target BD address.

1) Declaration

```
ble_status_t R_BLE_SC_GetConnectionAddress (uint8_t *p_type, uint8_t *p_address)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 25. Parameters

Direction	Parameter Name	Description
[out]	p_type	Identity address type. *BLE_GAP_ADDR_PUBLIC* (0x00) : Public address *BLE_GAP_ADDR_RAND* (0x01) : Static random address
[out]	p_address	Pointer for Identity address.

3) Return values

The return values of this API are the following.

Table 26. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	*p_type* or *p_address* is specified as NULL.

4) Events

None.

5) Note

None.

2.1.9 R_BLE_SC_SetConnectionInterval

R_BLE_SC_SetConnectionInterval API configures the connection interval, which will be used when establishing a connection with another device by using *R_BLE_SC_Connection* API.

1) Declaration

```
ble_status_t R_BLE_SC_SetConnectionInterval (uint16_t interval)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 27. Parameters

Direction	Parameter Name	Description
[in]	Interval	Connection interval. Connection interval [msec] = *interval* * 1.25. Valid range is 0x0006 - 0x0C80.

3) Return values

The return values of this API are the following.

Table 28. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The interval parameter is out of range.
BLE_ERR_INVALID_STATE(0x008)	Already establish a connection with another device.

4) Events

None.

5) Note

- This API cannot be used while a connection with another device exists.

2.1.10 R_BLE_SC_GetConnectionInterval

R_BLE_SC_GetConnectionInterval API gets the current connection interval.

1) Declaration

```
ble_status_t R_BLE_SC_GetConnectionInterval (uint16_t * p_interval)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 29. Parameters

In/out	Parameter Name	Description
[out]	p_interval.	Connection interval. Connection interval [msec] = *(*p_interval)* * 1.25.

3) Return values

The return values of this API are the following.

Table 30. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	*p_interval* is specified as NULL.

4) Events

None.

5) Note

None.

2.1.11 R_BLE_SC_SetPhy

R_BLE_SC_SetPhy API configures the Tx / Rx PHY.

1) Declaration

```
ble_status_t R_BLE_SC_SetPhy (uint16_t conn_hdl, uint8_t phy)
```

2) Parameter

To use this API, it is necessary to specify the following parameters as an argument.

Table 31. Parameters

Direction	Parameter Name	Description
[in]	conn_hdl	Connection handle.
[in]	phy	The PHY to be used. The valid range is, *BLE_GAP_SET_PHYS_HOST_PREF_1M* (0x01) : LE 1M PHY *BLE_GAP_SET_PHYS_HOST_PREF_2M* (0x02) : LE 2M PHY *BLE_GAP_SET_PHYS_HOST_PREF_CD* (0x04) : LE Coded PHY (S=8)

3) Return values

The return values of this API are the following.

Table 32. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The *phy* parameter is out of range, or the *conn_hdl* parameter is invalid.
BLE_ERR_UNSUPPORTED(0x0007)	*BLE_Driver* FPS module's current configuration does not support this feature.
BLE_ERR_INVALID_STATE(0x0008)	The connection has not been established, or the host stack is not running.
BLE_ERR_CONTEXT_FULL(0x000B)	The API call could not be processed because the Host Stack queue was full.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Memory cannot allocate to proceed this API.

4) Events

None.

5) Note

- The user can use this API after establishing a connection with another device.
- This API does not support asymmetric PHY.
- This API does not support S=2 coding.
- Coded PHY can be used when choosing *extended* as *BLE_Driver* FSP module configuration.
- LE 2M PHY can be used when choosing *extended* or *balance* configuration as *BLE_Driver* FSP module configuration.
- This API cannot use when choosing *compact* as *BLE_Driver* FSP module configuration.

2.1.12 R_BLE_SC_GetPhy

R_BLE_SC_SetPhy API gets the current Tx / Rx PHY.

1) Declaration

```
ble_status_t R_BLE_SC_GetPhy (uint16_t conn_hdl, uint8_t * p_phy_in_use)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 33. Parameters

Direction	Parameter Name	Description
[in]	conn_hdl	Connection handle.
[out]	p_phy_in_use	The PHY to be used. 0x01: LE 1M PHY 0x02: LE 2M PHY 0x03: LE Coded PHY (S=8)

3) Return values

The return values of this API are the following.

Table 34. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	*p_phy_in_use* is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The *conn_hdl* parameter is invalid.
BLE_ERR_UNSUPPORTED(0x0007)	*BLE_Driver* FPS module's current configuration does not support this feature.
BLE_ERR_INVALID_STATE(0x0008)	The PHY update is in progress, or no active connection exists.
BLE_ERR_CONTEXT_FULL(0x000B)	The API call could not be processed because the Host Stack queue was full.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Memory cannot allocate to proceed with this API.

4) Events

None.

5) Note

- The user can use this API after establishing a connection with another device.
- This API does not support when choosing *compact* as *BLE_Driver* FSP module configuration.

2.1.13 R_BLE_SC_SetMode

R_BLE_SC_SetMode API chooses the GATT communication method from *Notification* / *Write without response* or *Indication* / *Write*.

1) Declaration

```
ble_status_t R_BLE_SC_SetMode (uint8_t mode)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 35. Parameters

Direction	Parameter Name	Description
[in]	mode	GATT communication method. The valid range is, *R_BLE_SC_RESPONSE* (0x00, default setting) Server -> Client : Indication Client -> Server : Write *R_BLE_SC_NORESPONSE* (0x01) Server -> Client : Notification Client -> Server : Write without response

3) Return values

The return values of this API are the following.

Table 36. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The *mode* parameter is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The connection has not been established, or the host stack is not running.

4) Events

None.

5) Note

The user can use this API after establishing a connection with another device.

2.1.14 R_BLE_SC_GetMode

R_BLE_SC_GetMode API gets the GATT communication method in use.

1) Declaration

```
ble_status_t R_BLE_SC_GetMode (uint8_t * p_mode)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 37. Parameters

Direction	Parameter Name	Description
[out]	p_mode	GATT communication method. The valid range is, *R_BLE_SC_RESPONSE* (0x00, default setting) Server -> Client : Indication Client -> Server : Write *R_BLE_SC_NORESPONSE* (0x01) Server -> Client : Notification Client -> Server : Write without response

3) Return values

The return values of this API are the following.

Table 38. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	*p_mode* is specified as NULL.

4) Events

None.

5) Note

- The user can use this API after establishing a connection with another device.

2.1.15 R_BLE_SC_SendData

R_BLE_SC_SendData API transmits the PDU to the opposite device according to the GATT communication method specified by *R_BLE_SC_SetMode* API.

1) Declaration

```
ble_status_t R_BLE_SC_SendData ((uint16_t conn_hdl, uint8_t *p_data, uint16_t len)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 39. Parameters

Direction	Parameter Name	Description
[in]	conn_hdl	Connection handle.
[in]	p_data	Pointer to send data.
[in]	len	Length of send data. The maximum data length is (MTU size - 3) octets. MTU size can be configured by the *MTU Size Configured* property of the *BLE_Drievr* FSP module.

3) Return values

The return values of this API are the following.

Table 40. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	*p_data* is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The *conn_hdl* parameter is invalid, or the *len* parameter is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The connection has not been established, or the host stack is not running.
BLE_ERR_CONTEXT_FULL(0x000B)	The API call could not be processed because the Host Stack queue or the internal buffer was full.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Memory cannot allocate to proceed with this API.

4) Events

This API will notify the following events to the callback function registered by the *R_BLE_SC_Init* API..

Table 41. Events (Server side)

Event Code	Description
R_BLE_SCS_EVENT_WRITE_COMP (0x0001)	This event will be notified when the server has received a write request PDU to *Simple Connection Write* characteristic (D68C0003-A21B-11E5-8CB8-0002A5D5C51B). • **result** The *result* will be notified as *ble_status_t* type. Refer to *Renesas Flexible Software Package (FSP) User's Manual* about *ble_status_t* specification. • **Event Data** The Event Data will be notified as *st_ble_seq_data_t* type and included characteristic value. Refer to *R11UM0154* about *st_ble_seq_data_t* definition.
R_BLE_SCS_EVENT_WRITE_WO_RESP_COMP (0x0002)	This event will be notified when the server has received a write without request PDU to *Simple Connection Write without Response* characteristic (D68C0005-A21B-11E5-8CB8-0002A5D5C51B). • **result** The *result* will be notified as *ble_status_t* type. Refer to *Renesas Flexible Software Package (FSP) User's Manual* about *ble_status_t* specification. • **Event Data** The Event Data will be notified as *st_ble_seq_data_t* type and included characteristic value.Refer to *R11UM0154* about *st_ble_seq_data_t* definition.

Event Code	Description
R_BLE_SCS_EVENT_INDICATION_CFM (0x0003)	This event will be notified when the server has received indication confirmation PDU to *Simple Connection Indication* characteristic (D68C0005-A21B-11E5-8CB8-0002A5D5C51B). • **result** The *result* will be notified as *ble_status_t* type. Refer to *Renesas Flexible Software Package (FSP) User's Manual* about *ble_status_t* specification. • **Event Data** The Event Data will be notified as *st_ble_seq_data_t* type and included characteristic value.Refer to *R11UM0154* about *st_ble_seq_data_t* definition.

Table 42. Events (Client side)

Event Code	Description
R_BLE_SCC_EVENT_WRITE_RSP (0x0011)	This event will be notified when the client has received a write response PDU to *Simple Connection Write* characteristic (D68C0003-A21B-11E5-8CB8-0002A5D5C51B). • **result** The *result* will be notified as *ble_status_t* type. Refer to *Renesas Flexible Software Package (FSP) User's Manual* about *ble_status_t* specification. • **Event Data** The Event Data will be notified as *st_ble_seq_data_t* type and included characteristic value. Refer to *R11UM0154* about *st_ble_seq_data_t* definition.

Event Code	Description
R_BLE_SCC_EVENT_INDICATION_COMP (0x0012)	This event will be notified when the client has received indication PDU from *Simple Connection Indication* characteristic (D68C0002-A21B-11E5-8CB8-0002A5D5C51B). • **result** The *result* will be notified as *ble_status_t* type. Refer to *Renesas Flexible Software Package (FSP) User's Manual* about *ble_status_t* specification. • **Event Data** The Event Data will be notified as *st_ble_seq_data_t* type and included characteristic value.Refer to *R11UM0154* about *st_ble_seq_data_t* definition.
R_BLE_SCC_EVENT_NOTIFICATION_COMP (0x0013)	This event will be notified when the client has received notification PDU from *Simple Connection Notification* characteristic (D68C0004-A21B-11E5-8CB8-0002A5D5C51B). • **result** The *result* will be notified as *ble_status_t* type. Refer to *Renesas Flexible Software Package (FSP) User's Manual* about *ble_status_t* specification. • **Event Data** The Event Data will be notified as *st_ble_seq_data_t* type and included characteristic value.Refer to *R11UM0154* about *st_ble_seq_data_t* definition.
R_BLE_SCC_EVENT_ERROR_RSP (0x0031)	The event will be notified when the client has received an error response from the server side. • **result** The *result* will be notified as *ble_status_t* type. Refer to *Renesas Flexible Software Package (FSP) User's Manual* about *ble_status_t* specification. • **Event Data** None.

5) Internal Buffer

It is not possible to send data within the period between the *Write Request* and *Response* or the *Indication* and *Confirmation*. The data should be lost when the user tries to send data within the period.

To avoid such a data loss, This API has a 244 octets internal buffer. The behavior of the internal buffer is shown below.

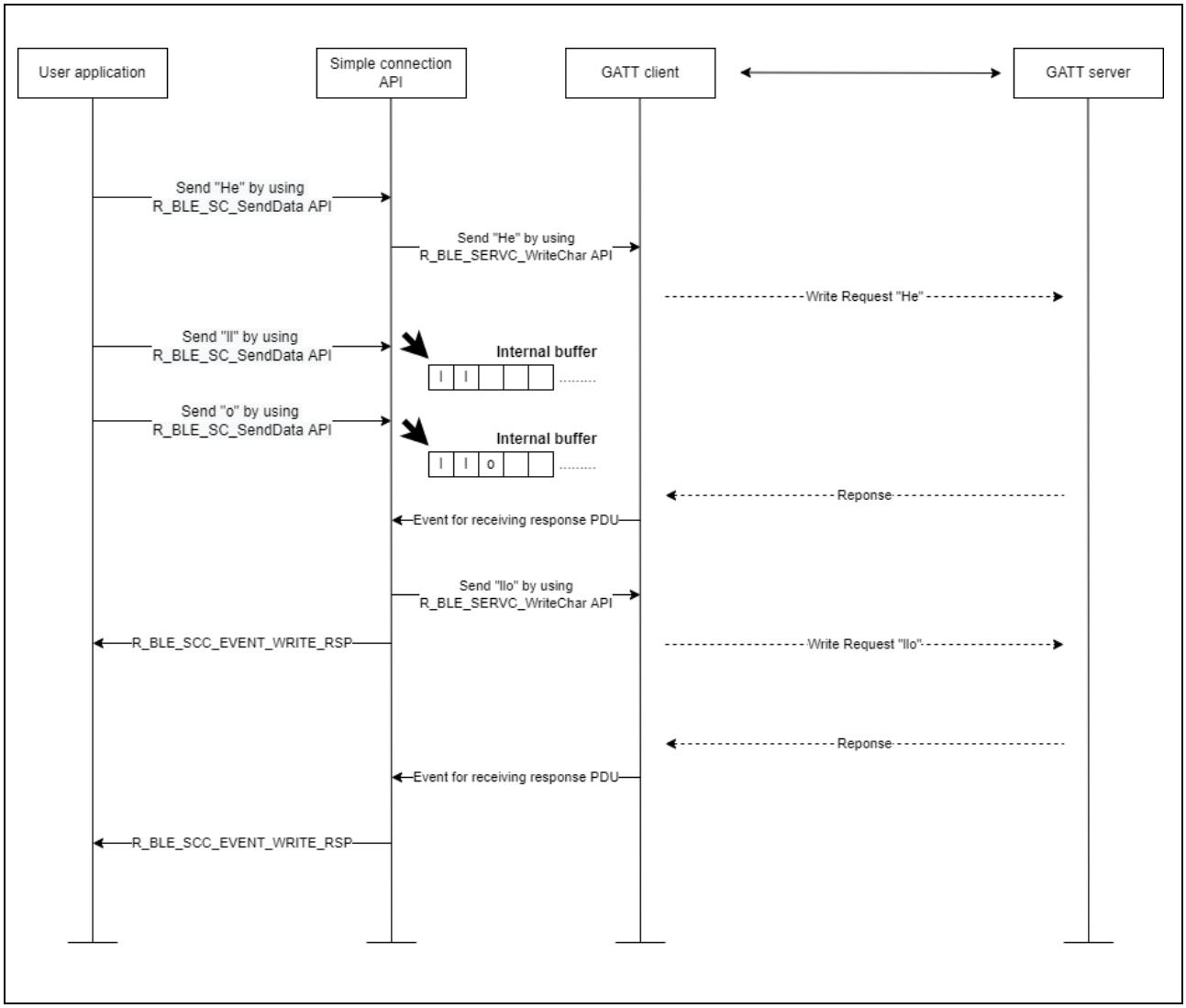

Figure 2. The behavior of the internal buffer

6) Note

- The user can use this API after establishing a connection with another device.
- The user should not use this API in the interrupt handler and the callback function specified in *R_BLE_AC_Init* API.

2.1.16 R_BLE_SC_GetConnDevice

R_BLE_SC_GetConnDevice API returns following variables.

- Connection handle
- Remote address type
- Remote address
- Exchanged MTU size

1) Declaration

```
ble_status_t R_BLE_SC_GetConnDevice(uint16_t *p_conn_hdl,
                                    uint8_t *p_type,
                                    uint8_t *p_address,
                                    uint16_t *mtu)
```

2) Parameters

To use this API, it is necessary to specify the following parameters as an argument.

Table 43. Parameters

Direction	Parameter Name	Description
[out]	*p_conn_hdl	Pointer to store connection handle.
[out]	*p_type	Pointer to store remote address type.
[out]	*p_address	Pointer to store remote address.
[out]	*mtu	Pointer to store MTU size.

3) Return values

The return values of this API are the following.

Table 44. Return value

Error Code	Description
BLE_SUCCESS(0x0000)	Success

4) Events

None.

2.1.17 R_BLE_SC_GetSemaphoreHandle

R_BLE_SC_GetSemaphoreHandle API returns the task handle when the user makes their own application on FreeRTOS or AzureRTOS environment.

Bluetooth LE application needs to call *R_BLE_Execute* API periodically. A task that periodically calls *R_BLE_Execute* API is created by *R_BLE_SC_Init* API. Task handle obtained by *R_BLE_SC_GetSemaphoreHandle* API is used when the user explicitly switches the context to the task that periodically calls *R_BLE_Execute* API.

1) Declaration

```
void *R_BLE_SC_GetSemaphoreHandle(void)
```

2) Parameters

None.

3) Return values

The return value of *R_SC_BLE_GetSemaphoreHandle* API varies depending on the type of RTOS as follows.

Table 45. Return value

RTOS	Synchronization Type (See Note)	Return value	Type
FreeRTOS	Event Group	Event group handle	EventGroupHandle_t
FreeRTOS	Semaphore	Semaphore handle	SemaphoreHandle_t
AzureRTOS	N/A	Semaphore handle	TX_SEMAPHORE

4) Events

None.

5) Note

In FSP4.0 or later, in order to improve the overhead of the task switching in the FreeRTOS environment, the context switch method to the task that periodically calls *R_BLE_Execute* API can be selected from Event Group and Semaphore by *Synchronization Type* property of *BLE Driver* FSP module.

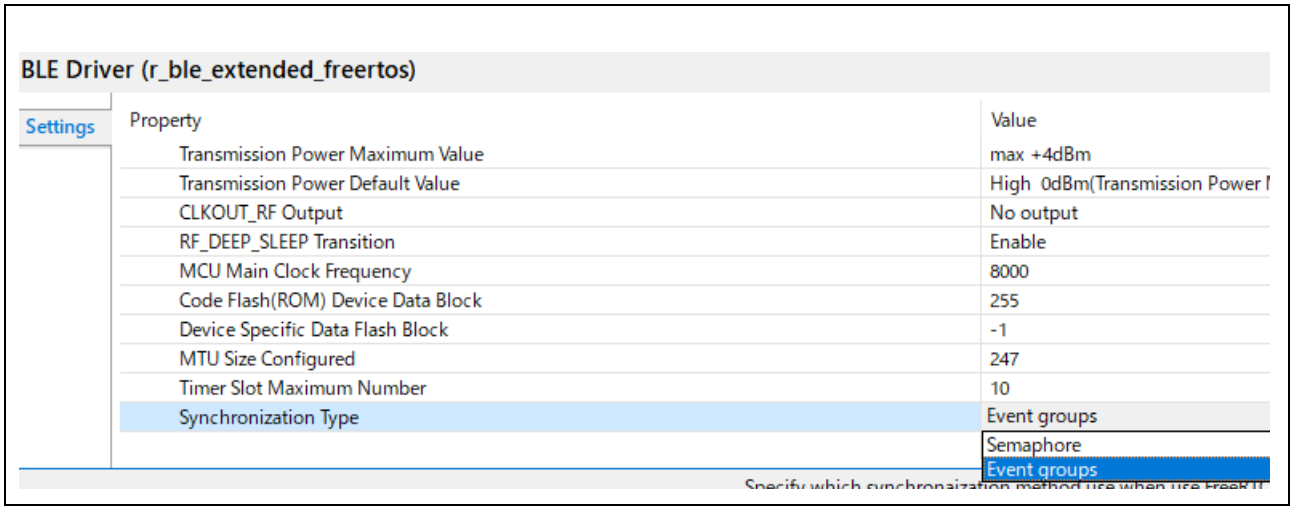

Figure 3. Synchronization Type property of BLE Driver FSP module

2.1.18 R_BLE_SC_Close

R_BLE_SC_Close performs,

- Terminate the BLE stack by using *RM_BLE_ABS_Close* API.
- Clear the callback registration configured by *R_BLE_SC_Init* API.

1) Declaration

```
ble_status_t R_BLE_SC_Close(void)
```

2) Parameters

None.

3) Return values

None.

4) Events

None.

2.2 Macros

This section describes the macro definition, which can be changed by the user. In the attached sample project listed in Table 1. Example projects, these macros are described in *simple_connection_api.h*.

Table 46. Macro definition for user settings

Macro	Description
R_BLE_SC_LOG_OUTPUT	Specify whether output GAP/GATT operation log or not. 0 : Log output is disabled 1 : Log output is enabled (Default)
R_BLE_SC_PAIRING_REQUIRE	Specify whether it requires pairing or not when the opposite device tries to access to own device's characteristic. 0 : Pairing is not required 1 : Pairing is required (Default)
R_BLE_SC_ADDRESS_IN_USE	Specify whether to use a public address or a static random address as the own address. 0x00 : Public address 0x01 : Static random address (Default)
R_BLE_SC_DEV_MAX	The maximum number of devices that can be found by *R_BLE_SC_Scan()* API. The default value is 10.

2.3 Required peripheral module

SC APIs require one channel of General PWM Timer (32bit).

3. Virtual UART Sample Application

This chapter describes the Virtual UART Sample Application(hereafter VUART App).

3.1 What is Virtual UART sample application

The virtual UART sample application (hereafter VUART App.) is an example project using the SC APIs. The VUART App. provides following functionality,

- Simple AT command function to control and configure Bluetooth LE connection.
  - Refer to section 3.4 about AT command specification.
  - Each AT command is a wrapper function for SC APIs.
- Send/receive characters or binary data to/from a remote device over Bluetooth LE communication.
  - Refer to section 2.1.1 about the required GATT database.

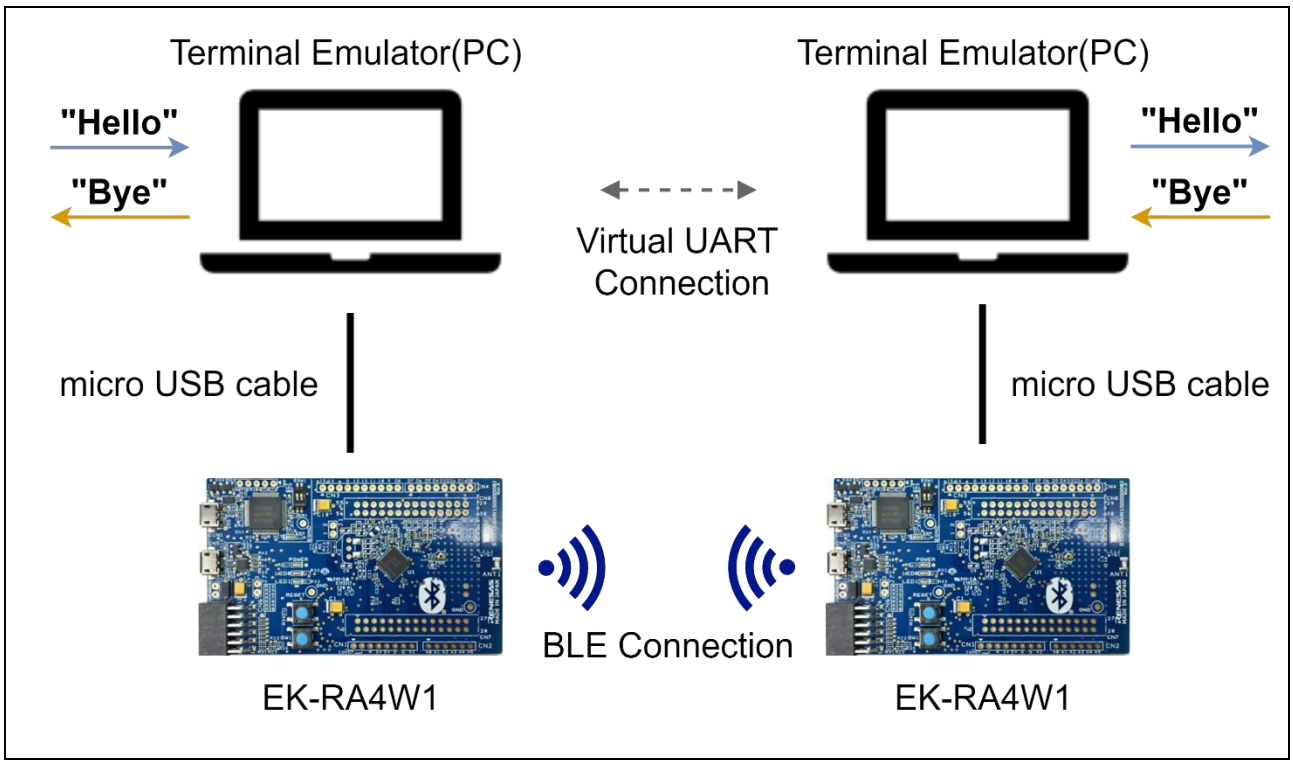

Figure 4. Operating environment of VUART App

3.2 Operation Flow

The VUART App. has the following states shown in Table 47.

Table 47. States of the VUART App

State	Description
Advertising	Advertising will be automatically started after power-ON.
Scanning	Scanning will be started by *at -ds* command and find the device which supports the Simple Connection Service.
Initiating	Establish a connection with another device by using *at -c* command.
Client	A client of the Simple Connection Service.
Server	A server of the Simple Connection Service.

Figure 5 shows state transition by AT commands.

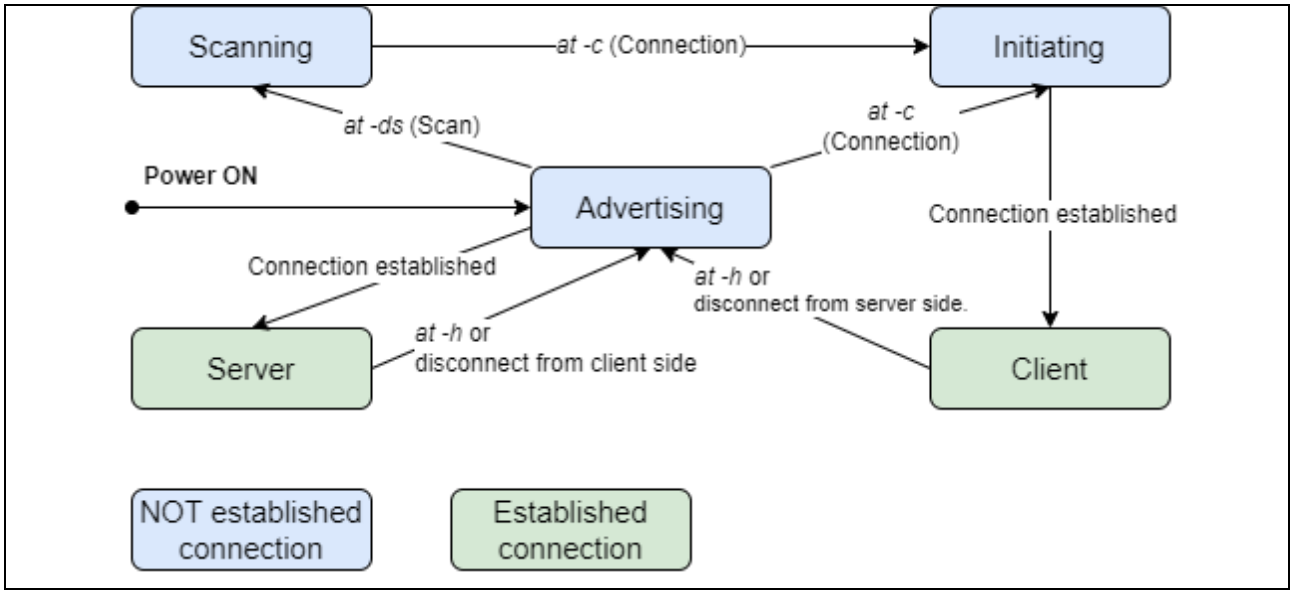

Figure 5. State transition diagram of the VUART App

3.3 AT command and VUART modes

The VUART App. has the following operation modes:

- **AT command mode**
  - Parses and executes AT commands input via terminal emulator.
- **VUART mode**
  - Send characters entered from the terminal emulator to the opposite device by using *R_BLE_SC_SendData* API.

When the device is in the Advertising, Scanning, or Initiating state, only AT command mode is available. When the device is in the Connection State(Client, Server), both AT command and VUART modes are available.

There are two methods to switch operation modes:

- Execute ***at -v*** command: can be used when entering VUART mode.
- Push **SW1** of the EK-RA4W1 board: can be used when entering both VUART mode and AT command mode.

Figure 6 shows the transition between AT command and VUART modes.

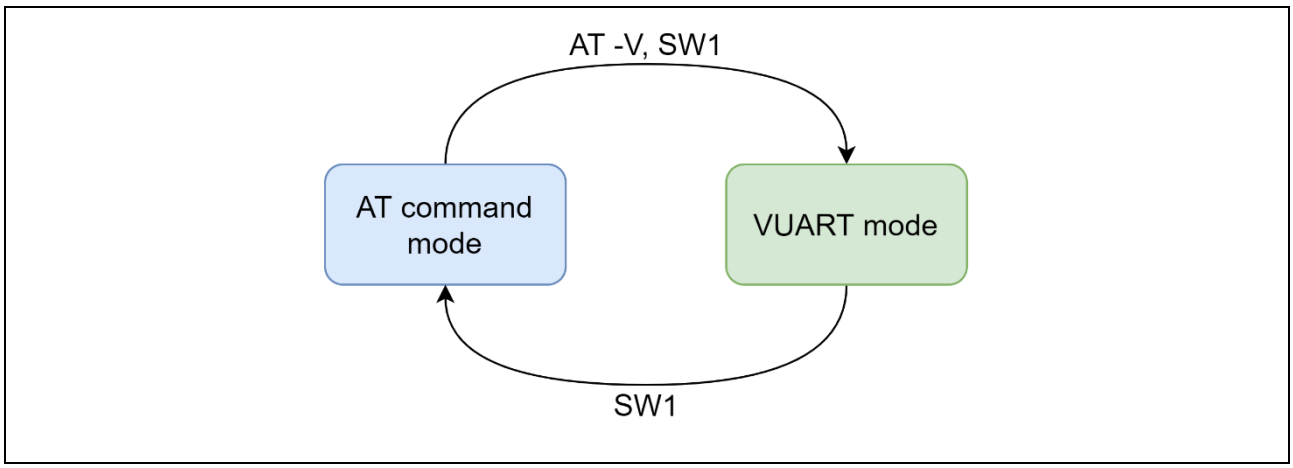

Figure 6. The transition between AT command and VUART modes

3.4 AT command reference

The following sections describe AT command specification. Most of the AT commands are a wrapper of SC APIs. The correspondence between AT command and SC APIs are following.

Table 48. The correspondence between AT command and SC APIs

AT command	Wrapping SC APIs	features
at -ds	*R_BLE_SC_Scan*	Scanning,
at -c	*R_BLE_SC_Connection*	Initiating.
at -h	*R_BLE_SC_Disconnection*	Disconnect,
at -ap	*R_BLE_SC_SetConnectionAddress* *R_BLE_SC_GetConnectionAddress*	Set/Get initiation target device address.
at -cI	*R_BLE_SetConnectionInterval* *R_BLE_GetConnectionInterval*	Set/Get connection interval.
at -as	*R_BLE_SC_SetAddress* *R_BLE_SC_GetAddress*	Set/Get the own device address.
at -p	*R_BLE_SC_SetPHY* *R_BLE_SC_GetPHY*	Set/Get PHY to use.
at -m	*R_BLE_SC_SetMode* *R_BLE_SC_GetMode*	Set/Get procedure to write Simple connection service characteristic from, *write* or *write without response (client)* indication or notification *(server)*
at -s	None	Display the current connection state.
at -e	None	Enable/Disable local echo,
at -v	None	Enter the VUART mode.
at -r	None	Reset .

Note: It is necessary to use lowercase when entering the AT commands on the console. ex) at -ds

3.4.1 AT -DS

at -ds command is a wrapper of the *R_BLE_SC_Scan* API. This command performs scanning to detect the device(s) that support the Simple Connection Service.

The scan will stop when:

- The number of discovered device(s) reaches *scan_device_limit,* which is specified by the user as AT command parameter.
- 7.68 [sec] have passed from the AT command entered.

When the scan stops, the following information will be displayed to the terminal emulator:

- BD address and its type.
- Device Name.

Table 49. at-ds command

AT -DS command		
Format :	at -ds (scan_device_limit)	
Parameters :	scan_device_limit	The number of device(s) to discover. If omitted, #R_BLE_SC_DEV_MAX (10) will be applied.
Usage :	$at -ds XX:XX:XX:XX:XX:XX rnd VUART#1 YY:YY:YY:YY:YY:YY pub VUART#2 ZZ:ZZ:ZZ:ZZ:ZZ:ZZ rnd VUART#3 at -ds : success --- The address and address type of the detected devices will be displayed at the end.	
	$at -ds 1 XX:XX:XX:XX:XX:XX rnd VUART#1 at -ds : success --- The scan stops when one device has been detected.	

3.4.2 AT -C

at -c command is a wrapper of the *R_BLE_SC_Connection API*. This AT command performs initiating procedure. This command requires the BD address of the device to connect. There are two ways to specify the address:

- Specify as an argument of at -c command.
- Specify by using *at -ap* command before using at -c command.

Table 50. at-c command

AT -C command		
Format :	at -c (addr) (addr_type)	
Parameters :	(addr)	The address of the device to connect.
	(addr_type)	The address type of the device to connect. pub : Public Address rnd : Random Address
Usage :	$at -c at -c : connection handle = 0xXX at -c : success --- Connects to the device specified by the AT-AP command in advance.	
	$at -c D1:45:9E:1D:AD:C7 rnd at -c : connection handle = 0xXX at -c : success --- Connect to the device that random address is D1:45:9E:1D:AD:C7.	

3.4.3 AT -H

at -h command is a wrapper of the *R_BLE_SC_DisConnection* API. This command will disconnect the current connection.

Table 51. at-h command

AT -H command		
Format :	at -h [conn_hdl]	
Parameters :	[conn_hdl]	The connection handle of the connection to disconnect.
Usage :	$at -h 0x20 at -h : disconnected handle = 0x20 at -h: success ---------- Disconnects the connection that connection handle is 0x20.	

3.4.4 AT -AP

at -ap command is a wrapper of the *R_BLE_SC_SetConnectionAddress* API and *R_BLE_SC_GetConnectionAddress* API. This command specifies the BD address of the device to connect. This command should be executed before *at -c* command.

Table 52. at-ap command

AT -AP command		
Format :	at -ap [addr] [addr_type] at -ap ?	
Parameters :	[addr]	The address of the device to connect.
	[addr_type]	The address type of the device to connect. **pub** : Public Address **rnd** : Random Address
	?	Get the current address and address type that is specified.
Usage :	$at -ap 74:90:50:FF:FF:FF pub at -ap : set success --- Set 74:90:50:FF:FF:FF as the public address of the connection target.	
	$at -ap ? Target Address = 74:90:50:FF:FF:FF Target Address Type = Public at -ap : get success --- Get the current address and address type that is specified.	

3.4.5 AT -CI

at -ci command is a wrapper of the *R_BLE_SC_SetConnectionInterval* API and the *R_BLE_SC_GetConnectionInterval* API. This command performs,

- Configuring the connection interval used when establishing a connection.
- Getting the current connection interval configuration.

The connection interval value configured by this *at -ci* command will only be applied when establishing a connection. If the user wants to change the connection interval by using this command, it is necessary to disconnect the current connection.

Table 53. at-ci command

AT -CI command		
Format :	at -ci [conn_intv] at -ci ?	
Parameters :	[conn_intv]	Set the connection interval value. Time(ms) = conn_intv * 1.25.
	?	Get the current connection interval setting.
Usage:	$at -ci 0x28 at -ci : set success --- Set the connection interval to 50ms(0x28 * 1.25 = 50ms).	
	$at -ci ? at -ci : get success (Connection Interval = 0x28) --- Get the current connection interval setting.	

3.4.6 AT -AS

at -as command is a wrapper of the *R_BLE_SC_SetAddress* API and *R_BLE_SC_GetAddress* API.
This command performs:

- Setting the own identify address and its type.
- Getting the current configuration of the own identify address and its type.

Table 54. at-as command

AT -AS command		
Format :	at -as [addr] [addr_type] at -as ?	
Parameters :	[addr]	The address to set.
	[addr_type]	The type of address. **pub** : Public Address **rnd** : Random Address
	?	Get the current configuration of the own address and its type.
Usage :	$at -as 74:90:50:FF:FF:FF pub Public Address = 74:90:50:FF:FF:FF at -as : set success ---------- Set 74:90:50:FF:FF:FF as the own public address.	
	$at -as ? Public Address = XX:XX:XX:XX:XX:XX Static Address = YY:YY:YY:YY:YY:YY at -as : get success ---------- Get the current own address and address type.	

3.4.7 AT -P

at -p command is a wrapper of the *R_BLE_SC_SetPhy* API and the *R_BLE_SC_GetPhy* API.
This command performs:

- Updates the PHY of the current connection.
- Gets the PHY used in the current connection.

This command is available when a connection is already established.

Table 55. at-p command

AT -P command		
Format :	at -p [conn_hdl] [phy] at -p [conn_hdl] ?	
Parameters :	[conn_hdl]	The handle of the connection to change PHY.
	[phy]	The PHY to set. **1M** : 1M PHY. **2M** : 2M PHY. **CD** : Coded PHY.
	?	Get the current PHY.
Usage :	$at -p 0x20 2M at -p : set success ---------- Sets the PHY of the connection with handle 0x20 to 2M PHY.	
	$at -p 0x20 ? Phy = 2M at -p : get success ----------	

3.4.8 AT -M

at -mAT -M command is a wrapper of the *R_BLE_SC_SetMode* API and the *R_BLE_SC_GetMode* API.
This command performs:

- Sets the GATT communication method.
- Gets the current configuration of the GATT communication method.

Table 56. at-m command

AT -M command		
Format :	at -m [mode] at -m ?	
Parameters :	[mode]	The data sending mode to set. **resp** : Use the GATT communication method with a response (Client: Write, Server: Indication). **noresp** : Use the GATT communication method without a response (Client: Write w/o Resp, Server: Notification).
	?	Get the current configuration of the GATT communication method.
Usage :	$at -m resp at -m : set success ---------- Sets the GATT communication method to "resp".	
	$at -m ? Response at -m : get success ----------	

3.4.9 AT -S

at -s command displays the current connection status.

Table 57. at-s command

AT -S command	
Format :	at -s
Parameters :	None.
Usage :	$at -s at-s: DISCONNECT ---------- No connection exists.
	$at -s at-s: CONNECT ---------- Connection exists.

3.4.10 AT -E

at -e command sets the local echo.

Table 58. at-e command

AT -E command		
Format :	at -e [on/off]	
Parameters :	[on/off]	**on** : Enable local echo. **off** : Unbale local echo.
Usage :	$at -e on at-e: Success. ---------- Enable local echo.	
	$at -e off at-e: Success. ---------- Unbale local echo.	

3.4.11 AT -V

at-v command enters the VUART mode. Also, refer to section 3.3 about the usage of this command.

Table 59. at-v command

AT -V command	
Format :	at -v
Parameters :	None
Usage :	$at -v app_main : VUART mode. ---------- Enter the VUART mode.

3.4.12 AT -R

at -v command resets the MCU.

Table 60. at-r command

AT -R command	
Format :	at -r
Parameters :	None
Usage :	$at -r !! MCU software reset !! BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8 addr:D2:8F:B9:18:F9:0F rnd on current register receive BLE_GAP_EVENT_ADV_ON result : 0x0000, adv_hdl : 0x0000 app_main : Simple Connection API successfully initialized. ---------- Resets the MCU.

3.5 How to use the VUART App

This section describes how to use the VUART App. Refer to section 1.4 about the required environment to use the VUART App.

3.5.1 Setup

3.5.1.1 Import projects

To import sample projects listed in Table 1, follow the steps below:

1. Launch the e²studio and select the workspace directory.

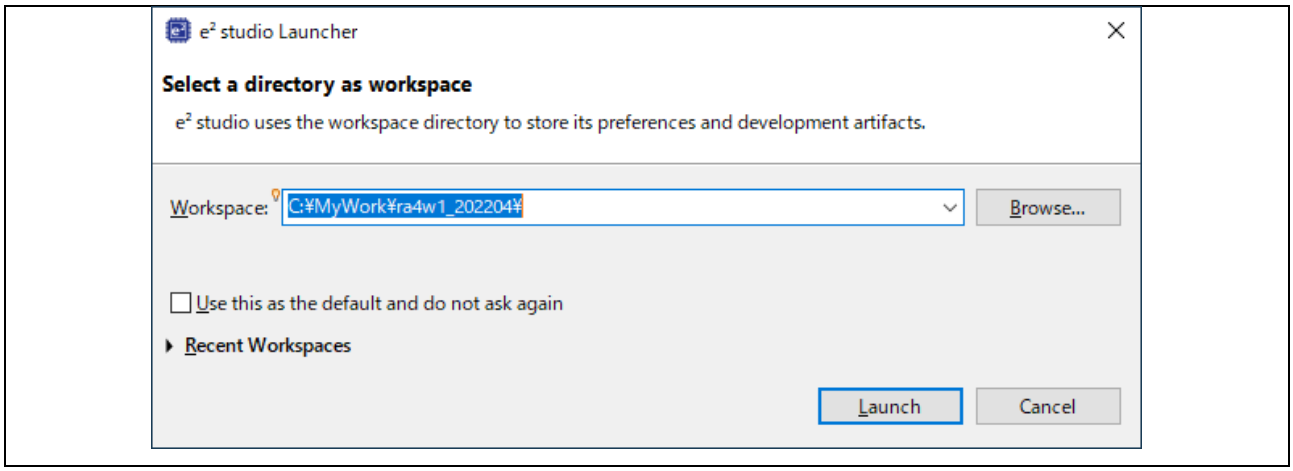

Figure 7. Select workspace

2. Select **File -> Import**.

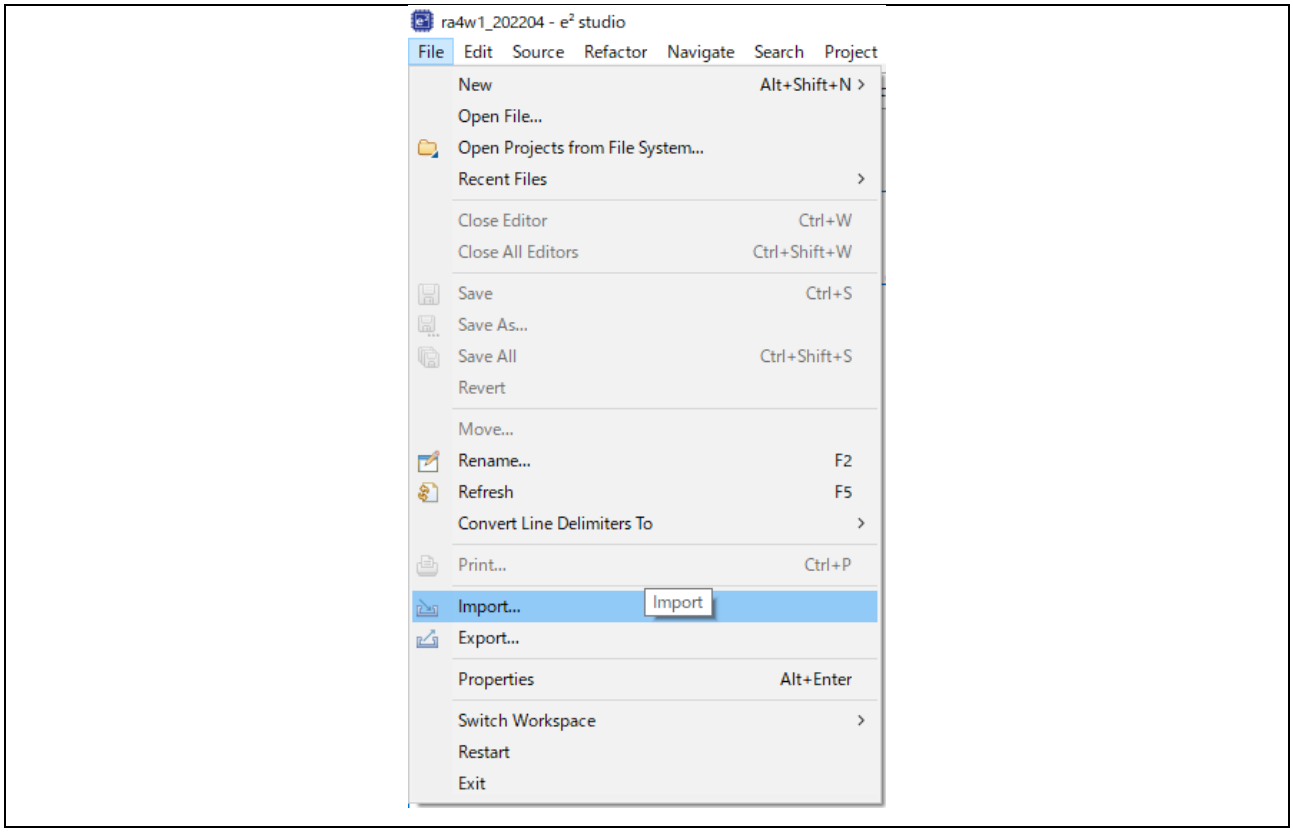

Figure 8. File menu

3. Select **Existing Projects into Workspace** and click **Next** button.

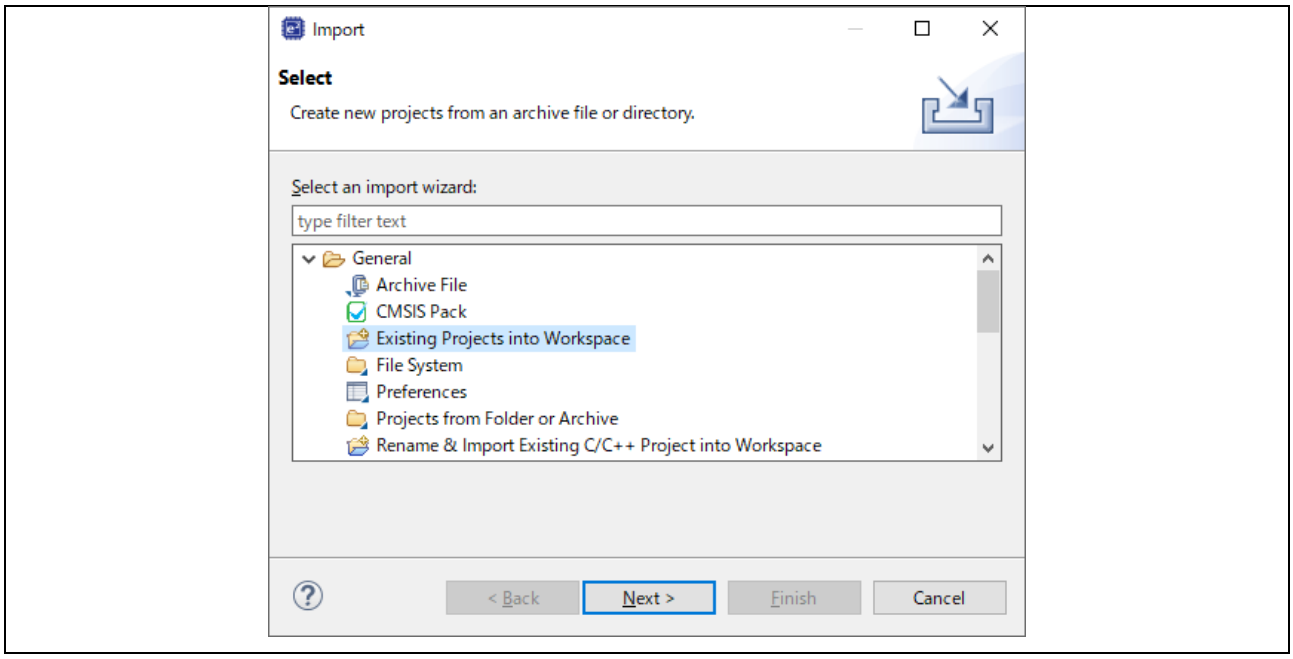

Figure 9. Select an import wizard

4. Select the root directory. Click **Browse…** button, and select the demo project folder. Click **Finish** button to import the demo project.

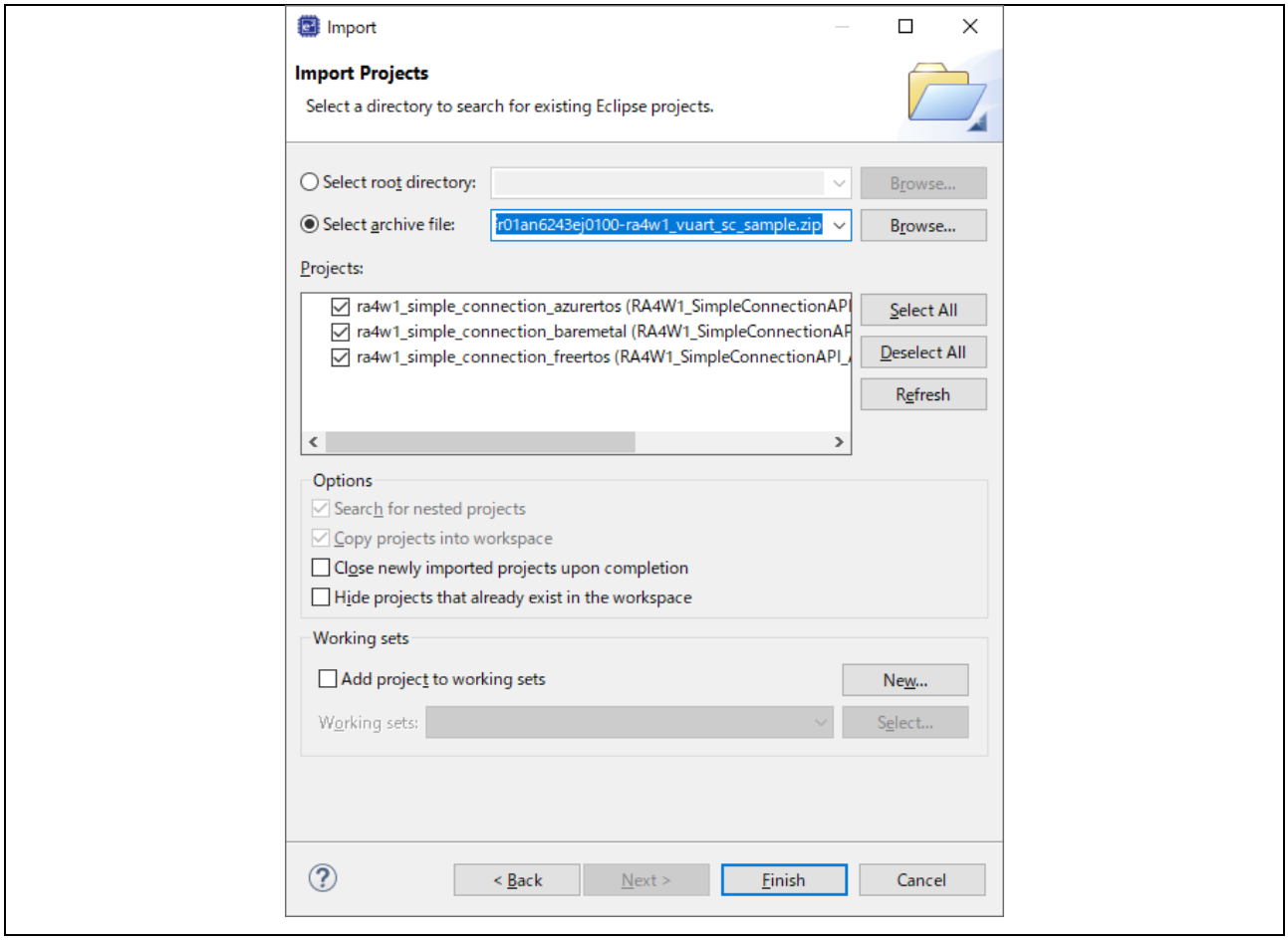

Figure 10. Import projects

3.5.1.2 Generate with FSP configurator

Generate source code of FSP modules before building the sample project.

1. Click **configuration.xml** to open the FSP configurator.

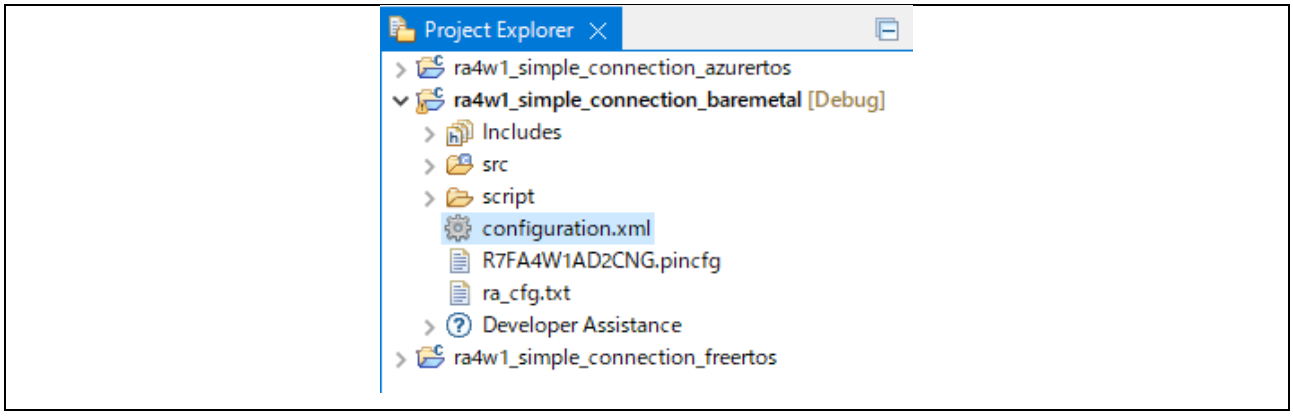

Figure 11. Click configuration.xml

2. Click "***Generate Project Content***" button.

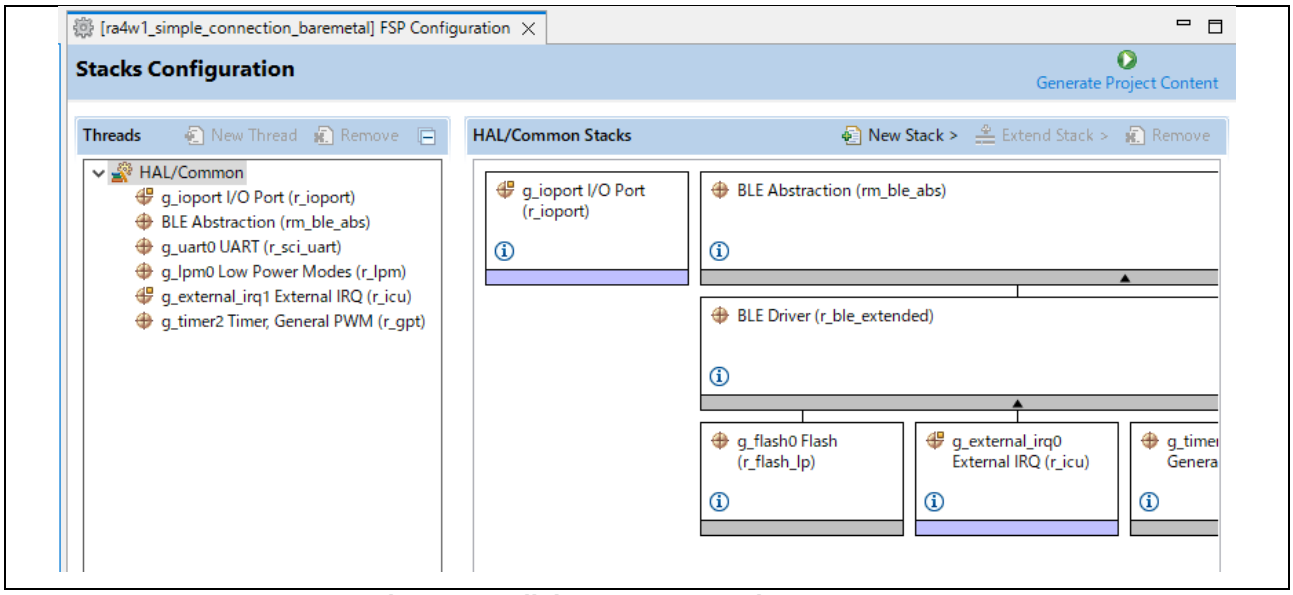

Figure 12. Click "Generate Project Content"

3. Check the following folders are generated: *ra, ra_gen, ra_cfg*.

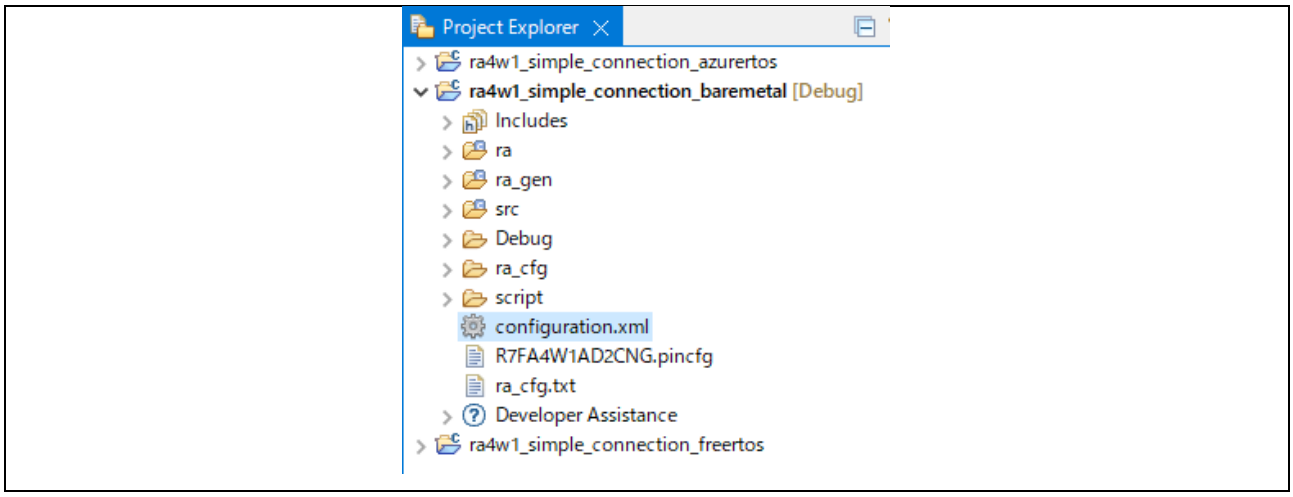

Figure 13. Check generated folders

3.5.1.3 Build and programming

Refer to Chapter 4 and Chapter 5 in " *Renesas e² studio 2021-04 or higher User's Manual: Quick Start Guide*" (R20UT4989).

3.5.2 Demo

This section describes the behaviors of the VUART App. It is necessary to 2pcs of EK-RA4W1 to perform data exchange.

3.5.2.1 Power on and advertising

Power on both of EK-RA4W1. After the power of the EK-RA4W1 is turned on, Advertising will be automatically started. Then, the following message will be displayed on the terminal emulator.

```
BLE_VS_EVENT_GET_ADDR_COMP result:0x0000, param_len:8
  addr:D2:8F:B9:18:F9:0F rnd on current register
receive BLE_GAP_EVENT_ADV_ON result : 0x0000, adv_hdl : 0x0000
app_main : Simple Connection API successfully initialized.
```

Figure 14. Power on and advertising

3.5.2.2 Scan

Execute *at -ds* command either one of EK-RA4W1 turned on in section 3.5.2.1 to find a device that supports simple connection service. Refer to section 3.4.1 for detail of *at -ds* command. After executing *at -ds* command, the following message will be displayed on the terminal emulator.

```
$ at -ds 1                                          Start scanning
DD:51:F4:60:70:7C rnd ff 0000
DD:51:F4:60:70:7C rnd ff 0000
receive BLE_GAP_EVENT_SCAN_OFF result : 0x0000
DD:51:F4:60:70:7C rnd VUART#1                       The discovered device(its address,
                                                    address type, device name)
at-ds: Success.
$
```

Figure 15. Scanning(Scanner)

3.5.2.3 Connect

Execute *at -c* command to connect to another EK-RA4W1 VUART App device. Refer to section 3.4.2 about the usage of *at -c* command. The following example connects to the device whose address is **DD:51:F4:60:70:7C rnd**(Static random address). The device which executes ***at -c*** command becomes GATT Client. The opposite device becomes GATT Server. After executing ***at -c*** command, the following message will be displayed on the Client side of the terminal emulator.

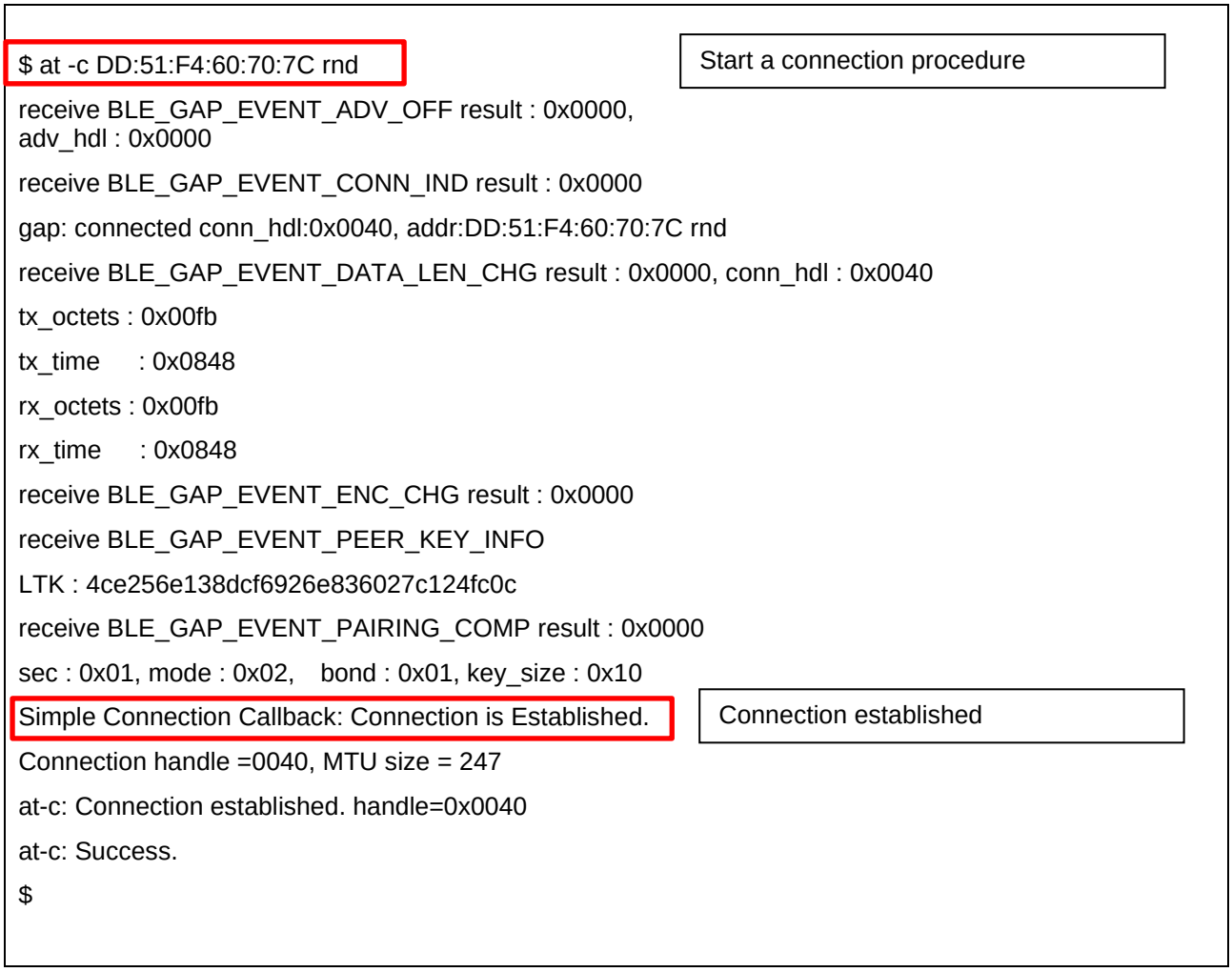

Figure 16. Connecting(Client)

Then, the following message will be displayed on the Server side of the terminal emulator.

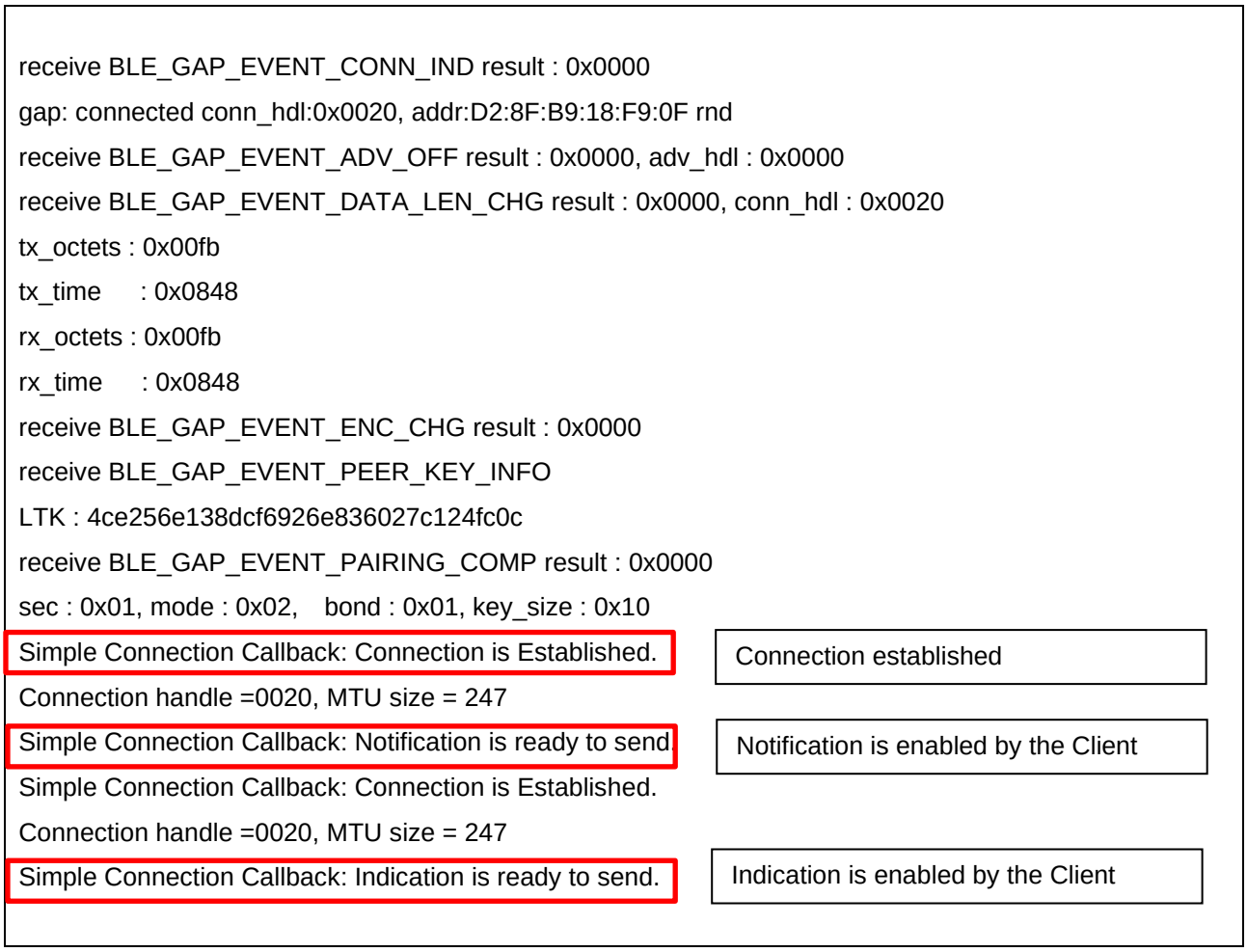

Figure 17. Connecting(Server)

3.5.2.4 Send message

Following is an example of sending a string from client to server.

1) To send messages from client to server, change the operation mode of the client side from AT command mode to VUART mode by using *at -v* command.

```
$ at -v
$ app_main : VUART mode.
```

Figure 18. Enter the VUART mode by at -v command(Client)

2) Type "**vuart 1234567890**" in the client side of the terminal emulator. The client side will send the string by the Write procedure.

```
$ at -v
$ app_main : VUART mode.


vuart 1234567890
```

Figure 19. Send a message(Client)

3) The string "**vuart 1234567890**" will be displayed on the server side of the terminal emulator.

```
vuart 1234567890
```

Figure 20. Receive the message on the opposite device(Server)

4) To resume the AT command mode, push **SW1** on the client side of the EK-RA4W1 board.

```
$ at -v
$ app_main : VUART mode.


vuart 1234567890
$ app_main : AT command mode.
```

Figure 21. Resume to the AT command mode(Client)

Note: If the user wants to send a string from server to client, do the same steps on the server side. In that case, the server will send a string by using the Indication procedure.

3.5.2.5 Disconnect

To disconnect the current connection, enter ***at -h*** command. Both the Client and the Server can be started the disconnection procedure. In this demo, the Client starts the disconnection procedure.

After executing ***at -h*** command, the following message will be displayed on the client side of the terminal emulator.

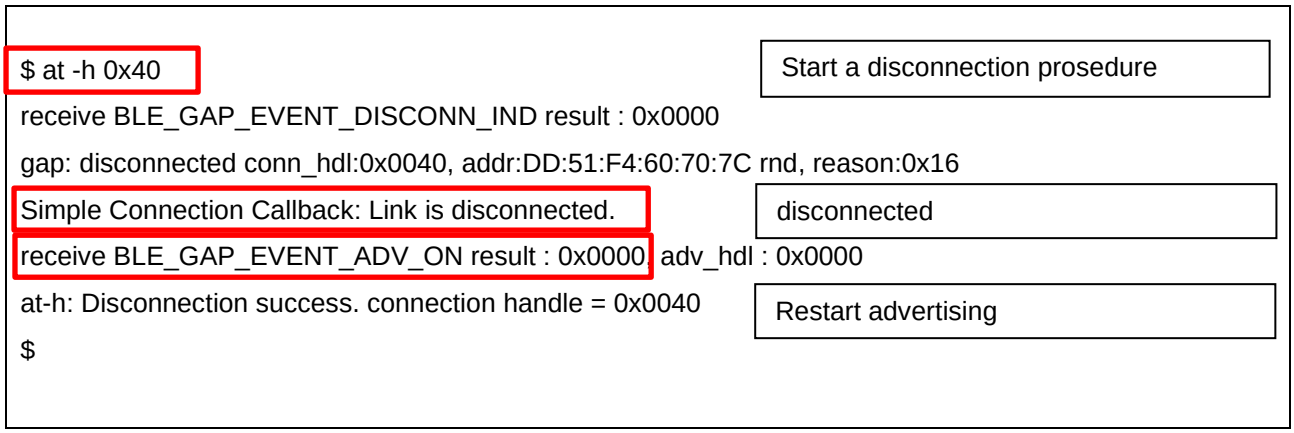

Figure 22. Disconnected(Client)

The following message will be displayed on the server side of the terminal emulator.

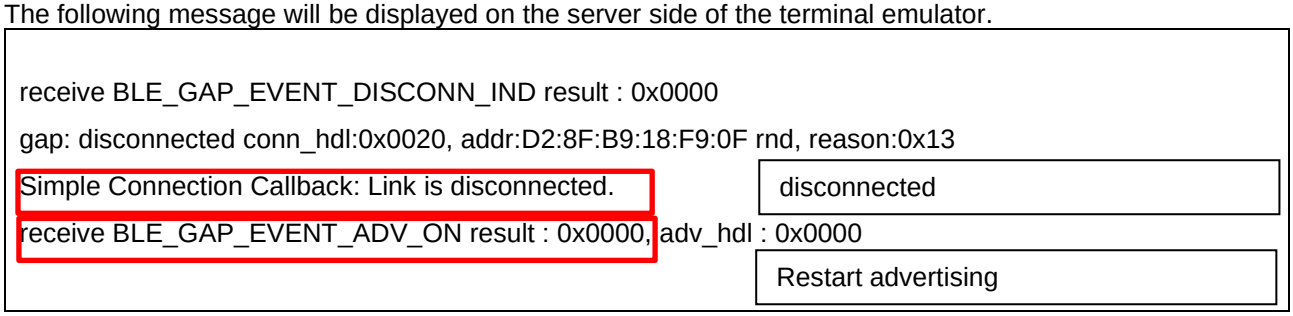

Figure 23. Disconnected(Server)

4. How to use SC APIs in the user application

This section describes how to use *SC APIs* in the user application.

4.1 Import related codes

1. Create an application project by referring "*RA4W1 Group BLE sample application*" (R01AN5402) chapter 4 and "*Bluetooth LE Profile API Document User's Manual*" (R11UM0154).

2. Copy the ***SimpleConnection_API*** folder from the attached example project with this APN to the *src* folder of the user application project made in item 1. If the user wants to use the command line interface, also copy the *app_lib* folder to the src folder of the application project.

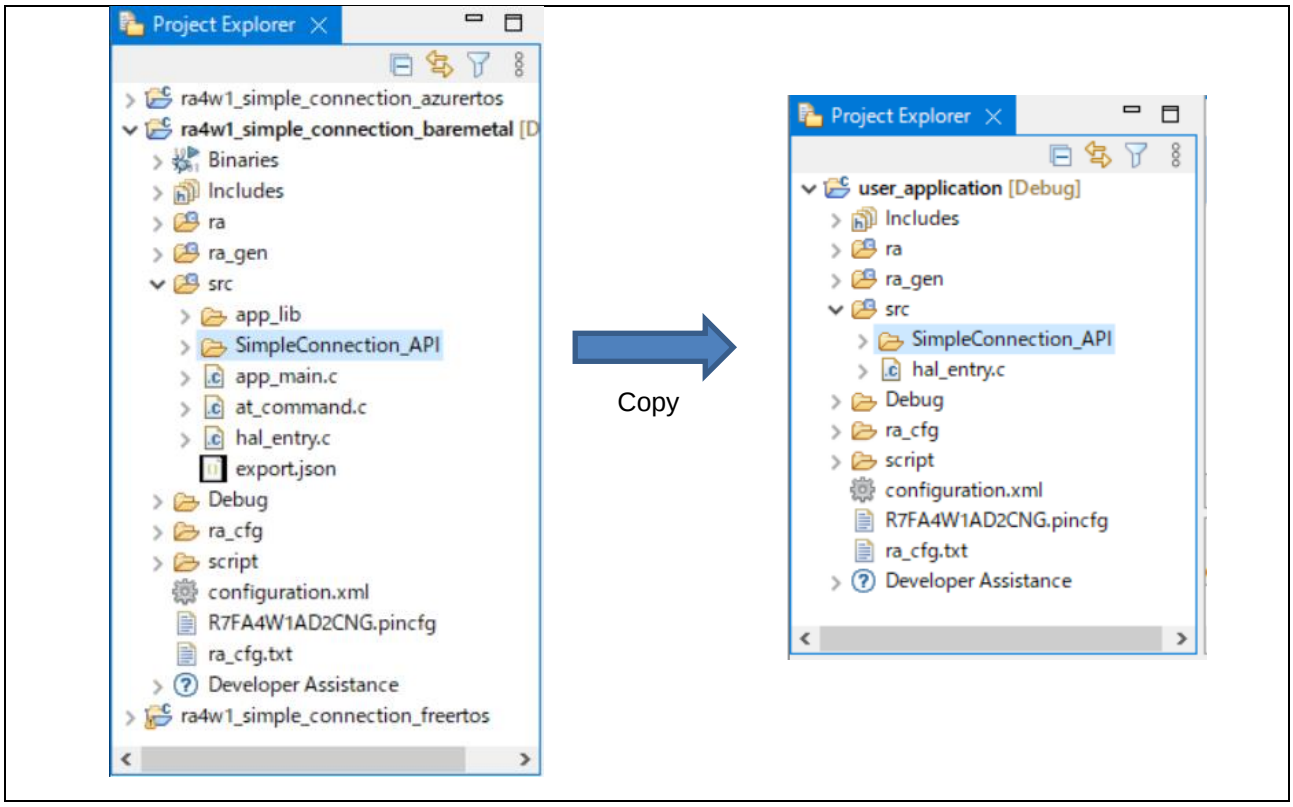

Figure 24. Copy the SimpleConnection_API folder

3. Add the ***Simple_Connection_API*** folder to the include path by editing the user application project property.

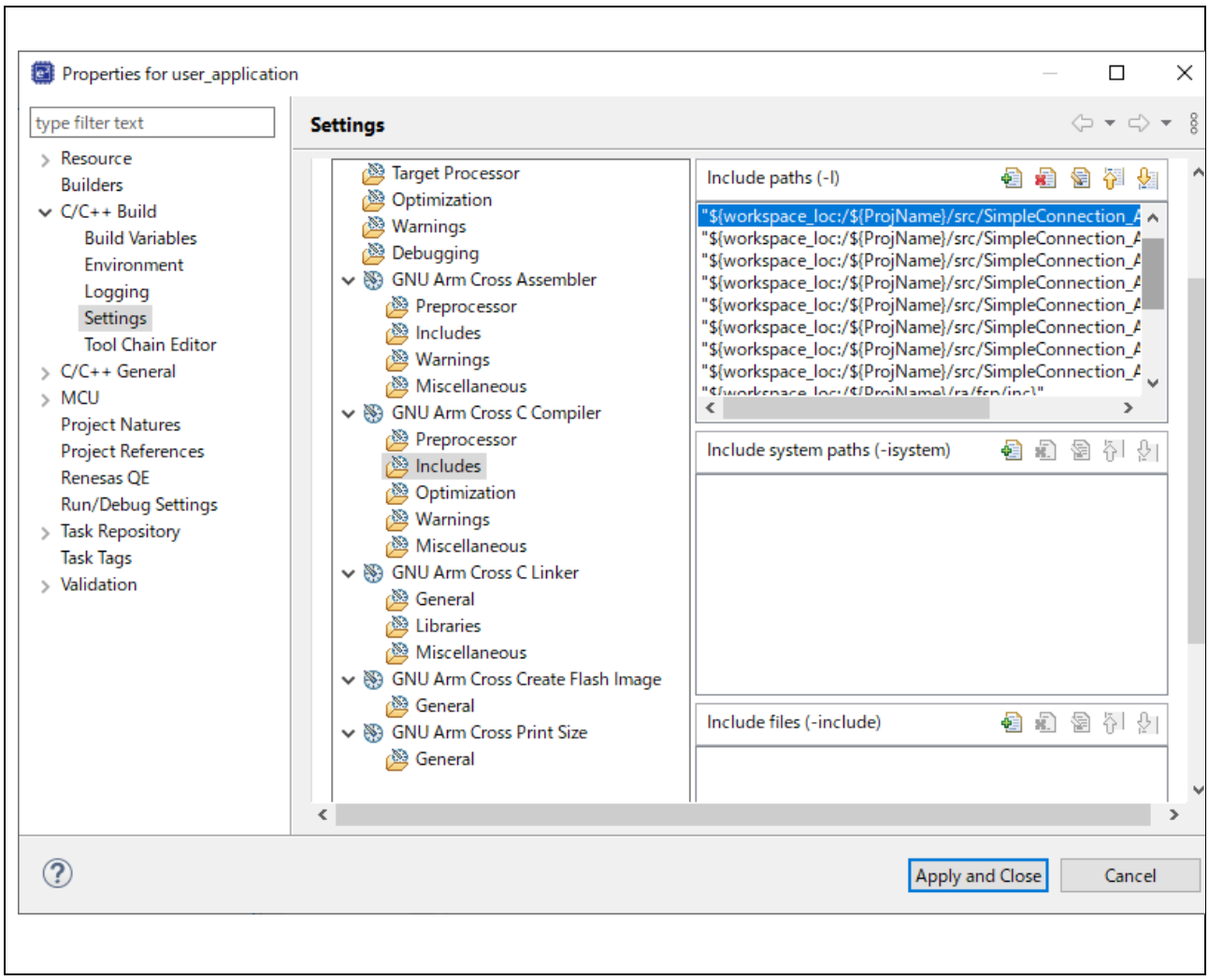

Figure 25. Add include path

4.2 Configure the required peripheral module

The SC APIs require one channel of General PWM Timer (32bit), as mentioned in section 2.3. The user needs to add/configure the timer according to the following steps.

1. Open the FSP configurator. Click **configuration.xml**.

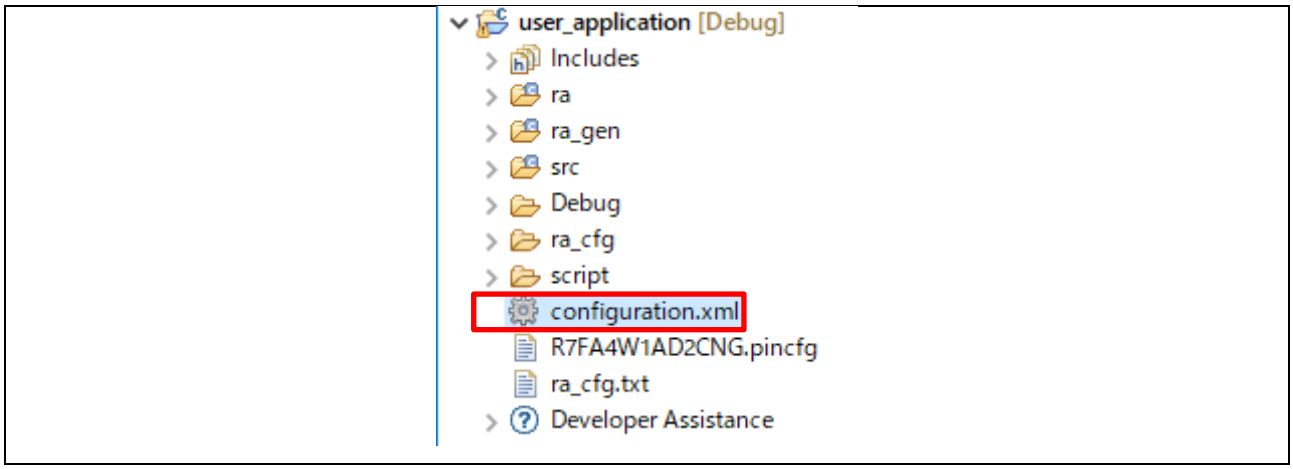

Figure 26. Open the FSP configurator

2. Add a timer for SC APIs. Click **New Stack -> Timers -> Timer, General PWM (r_gpt)**.

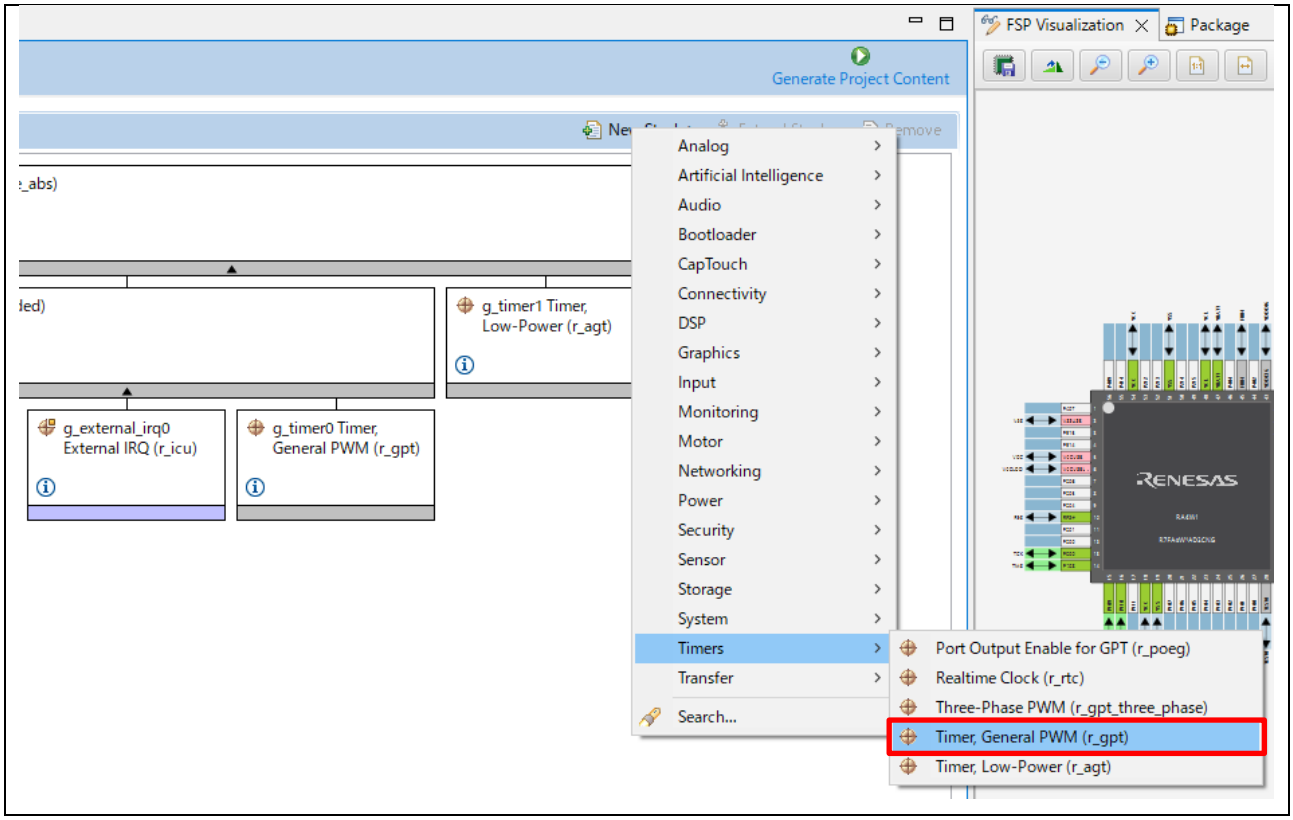

Figure 27. Add a timer for SC APIs

The timer properties are the following. Properties not listed in the table can be left as default values.

Table 61. General PWM Timer configurartions

Item	Value
General / Name	The user can assign a preferred name. Need to assign the same value to the *R_BLE_SC_TIMER_INSTANCE* macro of *Simple_Connection_API.c*.
General /Channel	Select from Channel 0 to 3.
General /Mode	Periodic
General /Period	1
General / Period Unit	Seconds
Interrupts / Overflow Crest Interrupt Priority	7
Interrupts / Callback	NULL

3. Click *Generate Project Content* button.

4.3 [THIS SECTION IS NO LONGER USED]

4.4 Implementation

The sample code is shown in Code 1, Code 2 and Code 3. The examples are the minimum implementation that performs Initialize(and Advertising), Scanning, Connection, and sending data from client to server on the bearmetal environment. *R_BLE_SCS_EVENT_WRITE_COMP* event will be notified on the server side of the application when receiving data from the client side. The user can add the following Code 1, Code 2 and Code 3 to *hal_entry.c*.

```
#include “simple_connection_api.h”

#define ROLE_CLIENT    (1)

void user_callback(uint16_t type, ble_status_t result, st_ble_seq_data_t *p_param);

uint8_t *g_p_received_data;

const char g_message[] = "MESSAGE\n";
```

Code 1. The sample code (declaration)

```
void hal_entry(void)
{
    ble_status_t ret;
    uint16_t conn_hdl;
    st_simple_find_device_t p_devices;

    /* Initialize and Advertising */
    ret = R_BLE_SC_Init(user_callback, "SC_DEVICE#1");
    if(BLE_SUCCESS != ret){ return; }
#if 1 == ROLE_CLINET
    /* Scanning the SC service device */
    ret = R_BLE_SC_Scan(1, &p_devices);
    if(BLE_SUCCESS != ret){ return; }

    /* Initiate a connection to the device detected */
    ret = R_BLE_SC_Connection(p_devices.find_device[0].addr_type,
                              p_devices.find_device[0].address,
                              &conn_hdl);
    if(BLE_SUCCESS != ret){ return; }

    /* Change the PHY to 2M */
    ret = R_BLE_SC_SetPhy(conn_hdl, BLE_GAP_SET_PHYS_HOST_PREF_2M);

    /* Wait for the PHY change complete */
    ret = BLE_ERR_INVALID_STATE;
    while(BLE_SUCCESS != ret)
    {
        R_BLE_Execute();
        ret = R_BLE_SC_GetPhy(conn_hdl, &phy_in_use);
    }

    /* Send a message */
    ret = R_BLE_SC_SendData(conn_hdl, (uint8_t*)g_message, (uint16_t)strlen(g_message));
    if(BLE_SUCCESS != ret){ return; }
#endif
    while(1)
    {
        /* Add your own code */
        R_BLE_Execute();
    }
}
```

Code 2. The sample code (main routine)

```
void user_callback(uint16_t type, ble_status_t result, st_ble_seq_data_t *p_param)
{
    FSP_PARAMETER_NOT_USED(result);

    switch (type) {
        case R_BLE_SCX_EVENT_CONN_IND:
            /* Add your code when establishing connection */
            break;
        case R_BLE_SCX_EVENT_DISCONN_IND:
            /* Add your code when disconnection has been happened */
            break;
       case R_BLE_SCS_EVENT_WRITE_COMP:
       {
            /* Add your code when write procedure has been completed (Server side only) */
           g_p_received_data = p_param->data;
       } break;
       case R_BLE_SCS_EVENT_WRITE_WO_RESP_COMP:
            /* Add your code when write without response procedure has been completed (Server side only) */
            break;
       case R_BLE_SCS_EVENT_INDICATION_CFM:
            /* Add your code when receive confirmation PDU (Server side only) */
            break;
       case R_BLE_SCC_EVENT_WRITE_RSP:
            /* Add your code when receive write response PDU (Client side only) */
            break;
       case R_BLE_SCC_EVENT_INDICATION_COMP:
            /* Add your code when receive indication PDU (Client side only) */
            break;
       case R_BLE_SCC_EVENT_NOTIFICATION_COMP:
            /* Add your code when receive notification PDU (Client side only) */
            break;
       case R_BLE_SCC_EVENT_ERROR_RSP:
            /* Add your code when receive error response PDU (Client side only) */
            break;
    }
}
```

Code 3. The sample code(user_callback)

R_BLE_Execute and SC blocking APIs should not call in the user callback function to avoid a deadlock condition. See the note in section 2.1.X for which APIs are classified as blocking API.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.28.2022	—	First edition issued.
1.01	Nov.30.2022	—	• The attached VUART sample application has been updated for FSP4.0 or later. • The following sections were updated. 2.1.1 R_BLE_SC_Init ➢ Add note on the usage of the user callback function. ➢ Fix the description of the *R_BLE_SCX_EVENT_CONN_IND* event notification timing on table 14. 2.1.12 R_BLE_SC_GetPhy ➢ Fix the description of the *BLE_ERR_INVALID_STATE*(0x0008) error on table 34. 2.1.17 R_BLE_SC_GetSemaphoreHandle ➢ Change Table 45. • Remove section 4.3. • Add PHY update procedure from LE 1M PHY to LE 2M PHY to Code 2.

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

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

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

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Contact information

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

Trademarks

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