RX23W Group

Sample Program for Highspeed Communication

Introduction

This application note provides sample program for highspeed communication using Bluetooth® low energy. Highspeed communication needs continuous transmit operation and suitable GAP parameter settings. This application note explains sample program to achieve highspeed communication and how it works.

Target Device

Target Board for RX23W

Related Documents

- RX23W Group Target Board for RX23W Quick Start Guide(R20QS0014)
- RX23W Group User's Manual: Hardware (R01UH0823)
- Bluetooth Low Energy Protocol Stack Basic Package User's Manual(R01UW0205)
- RX23W Group BLE Module Firmware Integration Technology Application Note(R01AN4860)
- RX23W Group Bluetooth Low Energy Profile Developer's Guide Application Note(R01AN4553)

Contents

1. Sample Program for Highspeed Communication 3
1.1 Projects 3
1.2 Running environment setup 3
1.3 Sample program operation 5
1.3.1 In case that the data is transmitted from server to client (Notification, Indication) 8
1.3.2 In case that the data is transmitted from client to server (Write Without Response, Write) 11
1.3.3 Changing GAP parameters 13
1.4 Custom (Throughput) Service 14
1.5 Software Structure 15
1.6 Source File Structure 16

2. Programs for highspeed communication. 18
2.1 Tx Flow control program 18
2.1.1 API Reference 20
2.2 Highspeed GAP Config Program 21
2.2.1 API Reference 23

3. Bluetooth Low Energy and Throughput 24
3.1 Generic Access Profile (GAP) 25
3.1.1 Device detection and connection establishment 25
3.1.2 Communication after establishing connection 26
3.1.3 Setting the connection interval 27
3.1.4 Setting the PHY 29
3.1.5 Setting the Maximum packet length 30
3.1.6 Setting the encryption of communication 31
3.2 Generic Attribute Profile (GATT) 32
3.2.1 No response operation (Notification / Write Without Response) 34
3.2.2 Response operation (Indication / Write) 34

4. BLE Solution tools 35
4.1 Designing profile by QE for BLE 37

5. Appendix 40
5.1 How to program to Target Board 40
5.2 How to add project to e^{2}studio workspace 45

Revision History 48

1. Sample Program for Highspeed Communication

By using the sample programs, two boards can connect and configure the settings suitable for highspeed communication automatically. The sample program measures throughput in bidirectional and continuous communication by the GATT.

1.1 Projects

This Application Note provides the program file as well as projects for the Central and Peripheral. Table 1.1 shows each program's role in the Bluetooth low energy communication. Section 5.1 shows how to import the projects into an e^{2}studio workspace. Table 1.2 shows the operation confirmation environment of this project.

Table 1.1 Project name and roles for sample program

Project name	GAP's role	GATT's role
tbrx23w_throughput_service_client	Central	GATT Client
tbrx23w_throughput_service_server	Peripheral	GATT Server

Table 1.2 The operation confirmation environment

Software	Version
e^{2}studio	7.6.0
CC-RX compiler	2.08

1.2 Running environment set-up

The two Target Boards are connected each other by sample programs and communicate.

Write the programs for peripheral (tbrx23w_throughput_service_server.mot) and for central (tbrx23w_throughput_service_client.mot) to each Target Board, respectively. The instructions for this can be found in section 5.2.

The sample programs are controlled via command line interface feature provided by the BLE protocol stack. Connect the device programmed central program to your PC via USB. The Device will be recognized as a serial port. Configure terminal emulator on your PC as in Table 1.3. Connect the device programmed to be the peripheral program to a PC port or power supply or since there is no need to use the command line interface feature (Figure 1.1).

Table 1.3 Terminal Emulator Settings

Setting	Configuration
New-line (Receive)	LF
New-line (Transmit)	CR
Terminal Mode	VT100
Baud rate	115200
Data	8bit
Parity	None
Stop bits	1bit
Flow Control	None

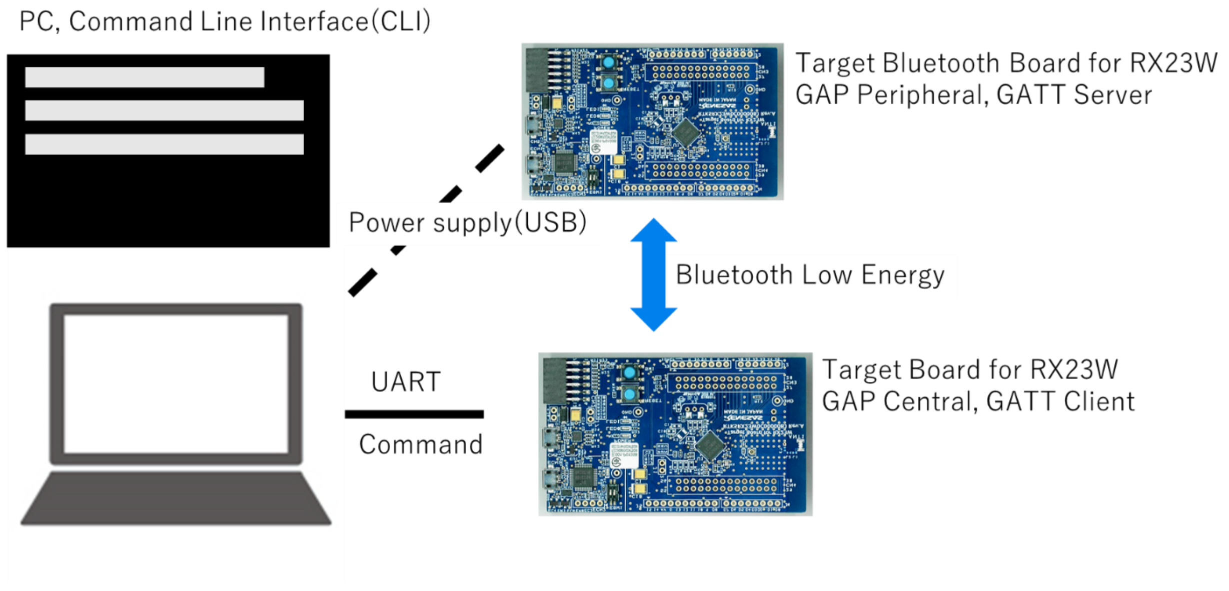


Figure 1.1 The setup for sample program

1.3 Sample program operation

Connect your PC to CN5 connector with an a micro B type USB cable and change emulator switch (2 of ESW1) to OFF as shown in Figure 1.2.

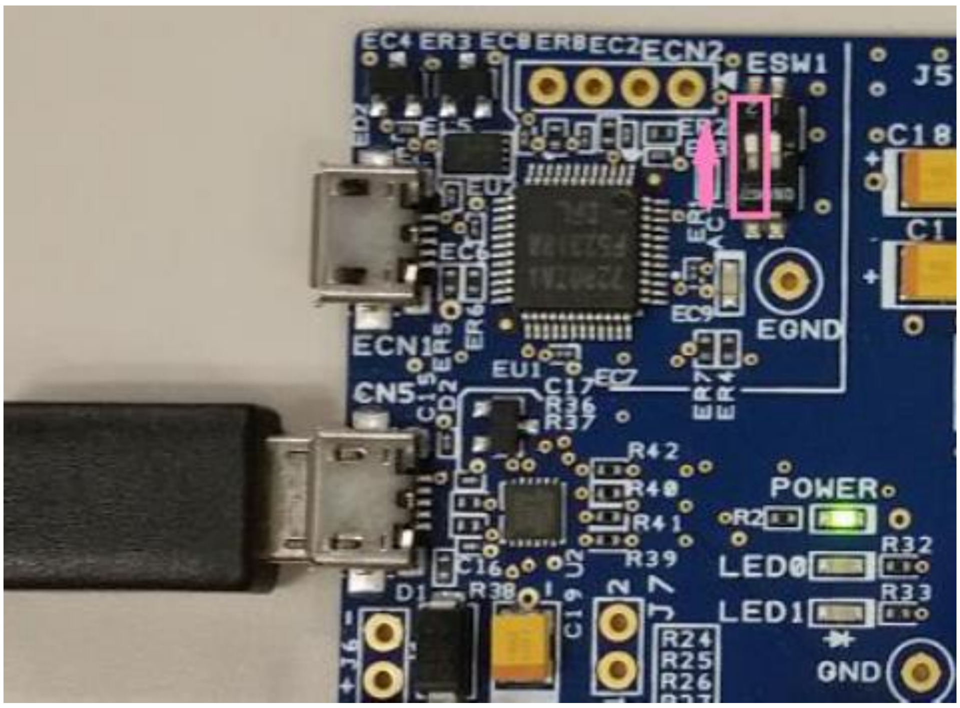


Figure 1.2 Target Device in Running Mode

Central and Peripheral will each complete scans and advertising and will automatically connect. The central will then initiate service discovery, then set the GAP parameters and initialize the MTU to achieve highspeed communication (Figure 1.3). Parameters and values that are set to achieve highspeed communications are shown in Table 1.4.

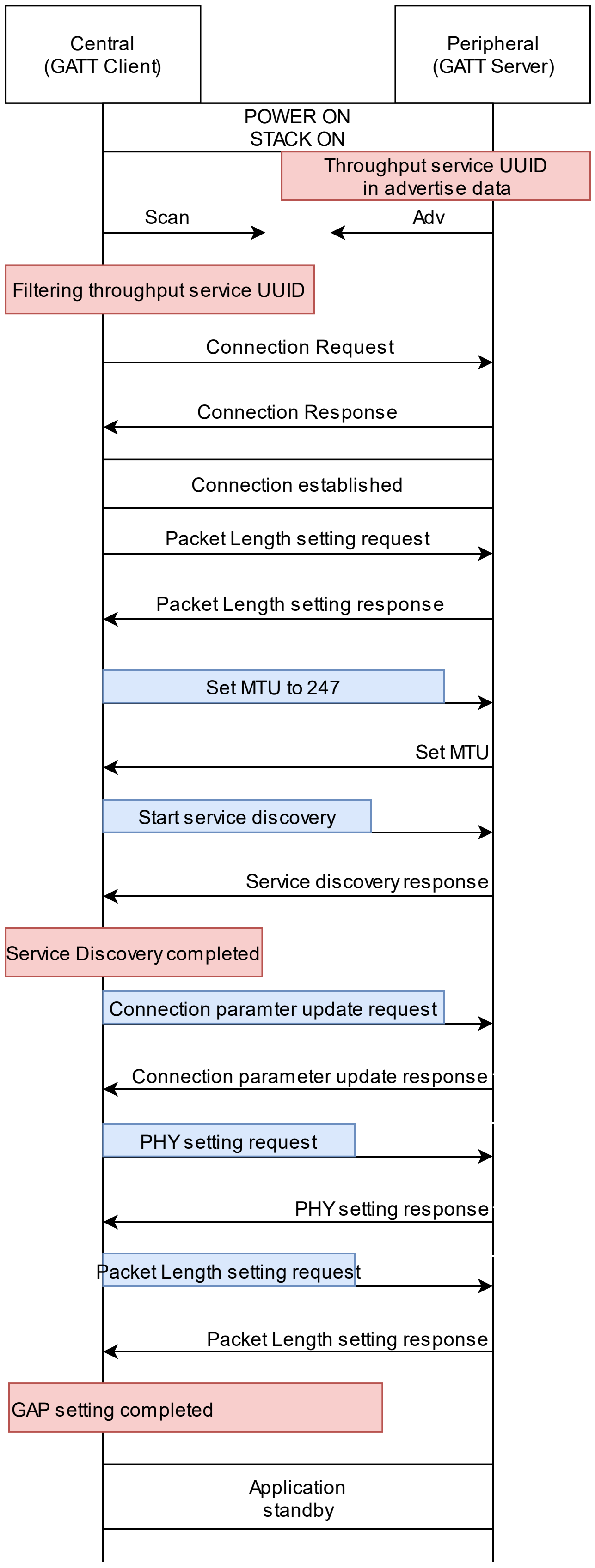
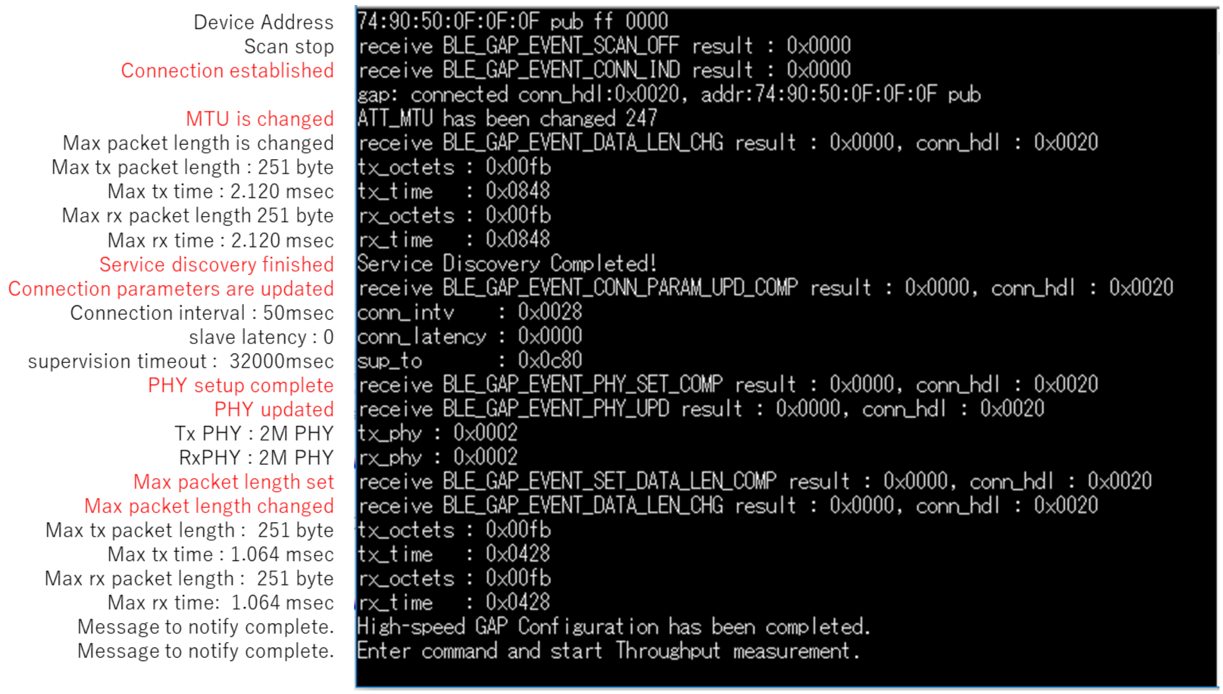


Figure 1.3 Start-up behavior of sample program

Table 1.4 Parameter and value configured at start-up.

parameter	Value
Connection Interval	50 (msec)
PHY	2M PHY
Max packet length	251 (byte)
MTU	247 (byte)

After highspeed communication initializations have been completed, the text "Enter command and start Throughput measurement" will be displayed in the CLI (Figure 1.4). Enter the command to CLI and start GATT communication using the throughput service. In the sample program the central is the client for the throughput service, while the peripheral is the server.


Device Address
Scan stop
Connection established
MTU is changed
Max packet length is changed
Max tx packet length : 251 byte
Max tx time : 2.120 msec
Max rx packet length 251 byte
Max rx time : 2.120 msec
Service discovery finished
Connection parameters are updated
Connection interval : 50msec
slave latency : 0
supervision timeout : 32000msec
PHY setup complete
PHY updated
Tx PHY : 2M PHY
RxPHY : 2M PHY
Max packet length set
Max packet length changed
Max tx packet length : 251 byte
Max tx time : 1.064 msec
Max rx packet length : 251 byte
Max rx time: 1.064 msec
Message to notify complete.
Message to notify complete.

```
74:90:50:0F:0F:0F pub ff 0000
receive BLE_GAP_EVENT_SCAN_OFF result : 0x0000
receive BLE_GAP_EVENT_CONN_IND result : 0x0000
gap: connected conn_hdl:0x0020, addr:74:90:50:0F:0F:0F pub
ATT_MTU has been changed 247
receive BLE_GAP_EVENT_DATA_LEN_CHG result : 0x0000, conn_hdl : 0x0020
tx_octets : 0x00fb
tx_time   : 0x0848
rx_octets : 0x00fb
rx_time   : 0x0848
Service Discovery Completed!
receive BLE_GAP_EVENT_CONN_PARAM_UPD_COMP result : 0x0000, conn_hdl : 0x0020
conn_intv    : 0x0028
conn_latency : 0x0000
sup_to       : 0x0c80
receive BLE_GAP_EVENT_PHY_SET_COMP result : 0x0000, conn_hdl : 0x0020
receive BLE_GAP_EVENT_PHY_UPD result : 0x0000, conn_hdl : 0x0020
tx_phy : 0x0002
rx_phy : 0x0002
receive BLE_GAP_EVENT_SET_DATA_LEN_COMP result : 0x0000, conn_hdl : 0x0020
receive BLE_GAP_EVENT_DATA_LEN_CHG result : 0x0000, conn_hdl : 0x0020
tx_octets : 0x00fb
tx_time   : 0x0428
rx_octets : 0x00fb
rx_time   : 0x0428
High-speed GAP Configuration has been completed.
Enter command and start Throughput measurement.
```

Figure 1.4 CLI display after Central device start-up.

The sample programs will communicate bidirectionally data from server to client and from client to server. Table 1.5 shows supported GATT operations and their respective control commands. These commands are registered as Throughput Client command in CLI. Once the start command is executed, throughput is measured at GATT communication. Note that there are differences in methods for measuring throughput for each transmitting direction.

Table 1.5 The control command each GATT communication operation.

GATT operation	Start command	Stop command
Notification	thc start notification	thc stop receive
Indication	thc start indication	thc stop receive
Write Without Response	thc start write_without_response	thc stop transmit
Write	thc start write	thc stop transmit

1.3.1 In the case that the data is transmitted from server to client (Notification, Indication)

Once the command is executed, the client enable Notification/Indication operation by writing the CCCD in server. In notification, the server continuously transmits data to client, because it doesn't require a response from the client (Figure 1.5). In indication, the server transmits data after it receives a confirmation from client, because it needs to wait response from client (Figure 1.6).

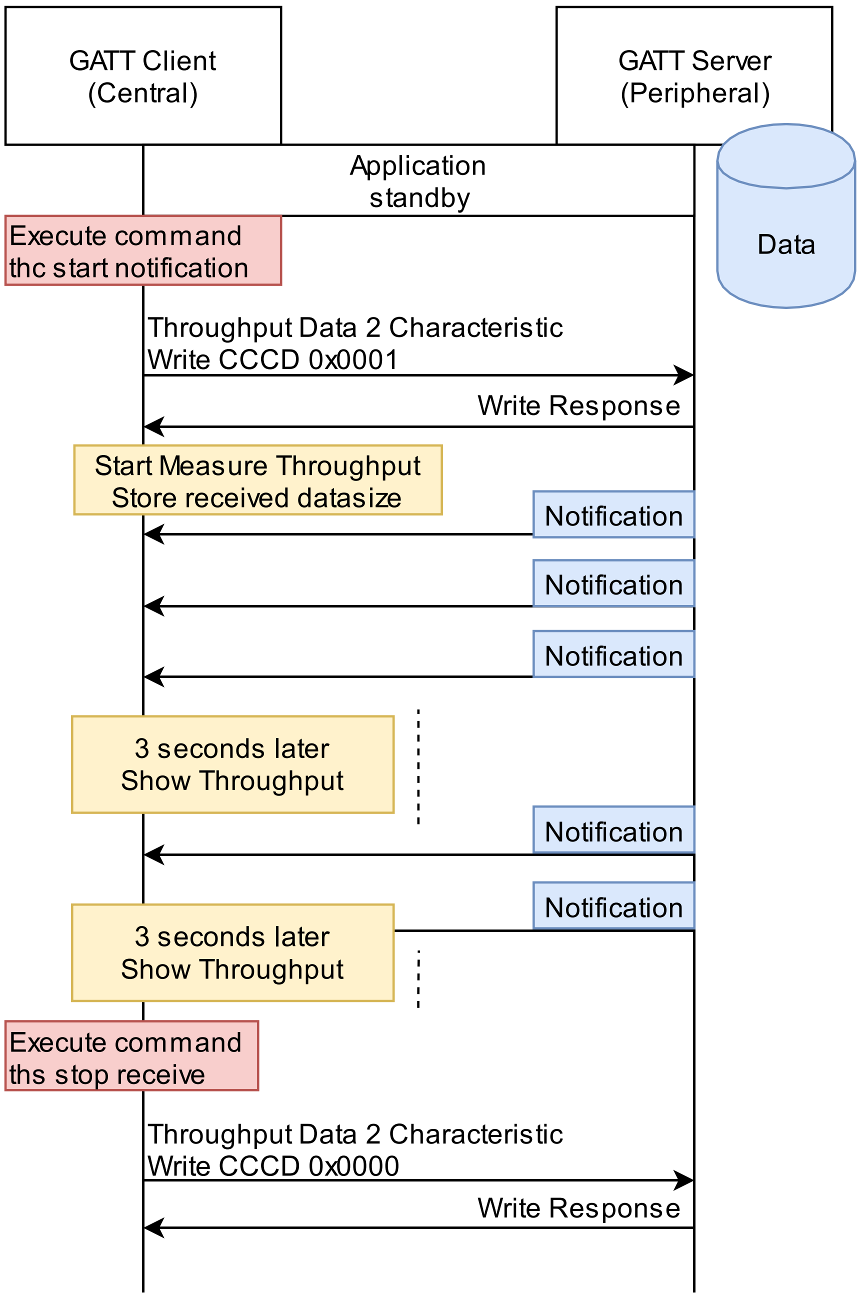

GATT Client
(Central)
GATT Server
(Peripheral)
Data
Application standby
Execute command thc start notification
Throughput Data 2 Characteristic Write CCCD 0x0001
Write Response
Start Measure Throughput Store received datasize
Notification
Notification
Notification
3 seconds later Show Throughput
Notification
Notification
3 seconds later Show Throughput
Execute command ths stop receive
Throughput Data 2 Characteristic Write CCCD 0x0000
Write Response

Figure 1.5 Sequence chart executed when executing notification executing command

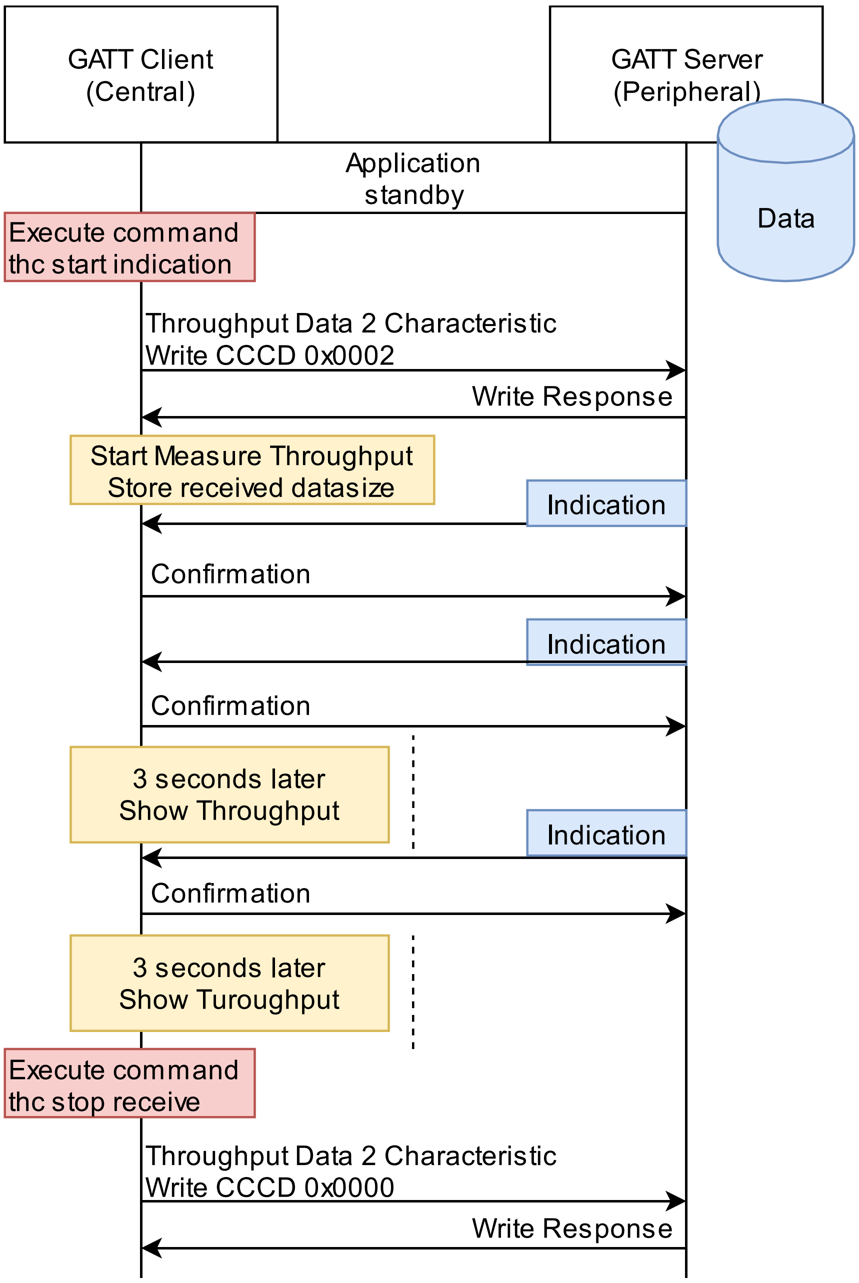
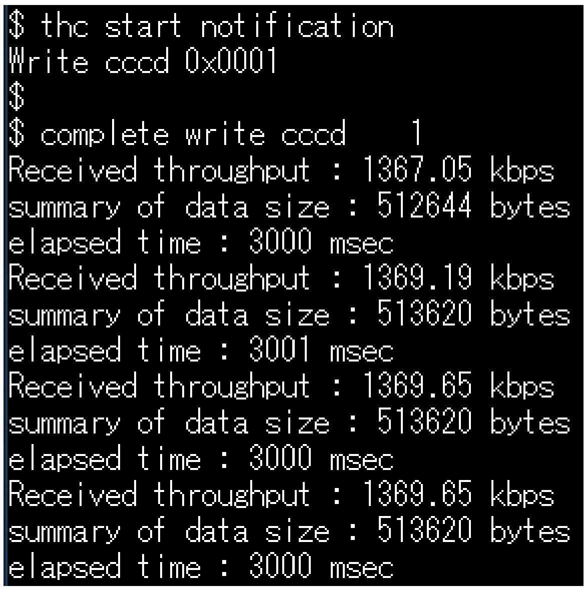


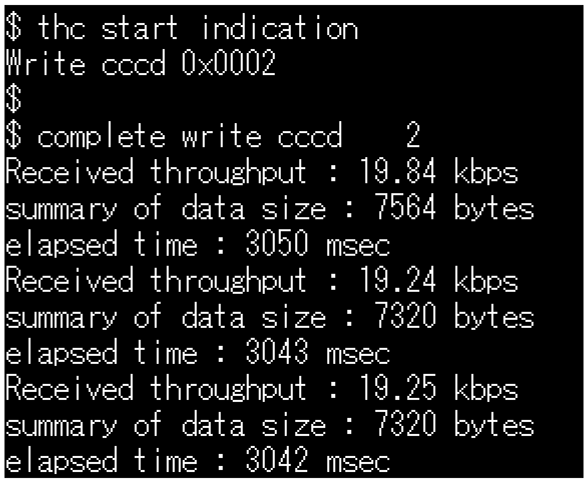
Figure 1.6 Sequence chart when executing indication command

The client calculates the throughput every 3 seconds from the total data size of Notification / Indication received and the elapsed time (3 seconds). The server will continue to send data until Notification / Indication is stopped. Figure 1.7 and Figure 1.8 show the display when Notification / Indication operation is performed.



```
$ thc start notification
Write cccd 0x0001
$
$ complete write cccd    1
Received throughput : 1367.05 kbps
summary of data size : 512644 bytes
elapsed time : 3000 msec
Received throughput : 1369.19 kbps
summary of data size : 513620 bytes
elapsed time : 3001 msec
Received throughput : 1369.65 kbps
summary of data size : 513620 bytes
elapsed time : 3000 msec
Received throughput : 1369.65 kbps
summary of data size : 513620 bytes
elapsed time : 3000 msec
```

Figure 1.7 CLI display when Notification operation is performed



```
$ thc start indication
Write cccd 0x0002
$
$ complete write cccd    2
Received throughput : 19.84 kbps
summary of data size : 7564 bytes
elapsed time : 3050 msec
Received throughput : 19.24 kbps
summary of data size : 7320 bytes
elapsed time : 3043 msec
Received throughput : 19.25 kbps
summary of data size : 7320 bytes
elapsed time : 3042 msec
```

Figure 1.8 CLI display when Indication operation is performed

1.3.2 In the case that the data is transmitted from client to server (Write Without Response, Write)

Figure 1.9 and Figure 1.10 show the sample programs behavior when a Write Without Response or Write operation is executed. Once these commands are executed, the client separates 5k bytes data allocated on ROM by MTU and transmit. After all data is transmitted, the client will show the throughput. Throughput is calculated by the elapsed time from command execution to storing all data to BLE protocol stack buffer. Due to this keep in mind that you will refer to a value that is slightly higher than the actual throughput.

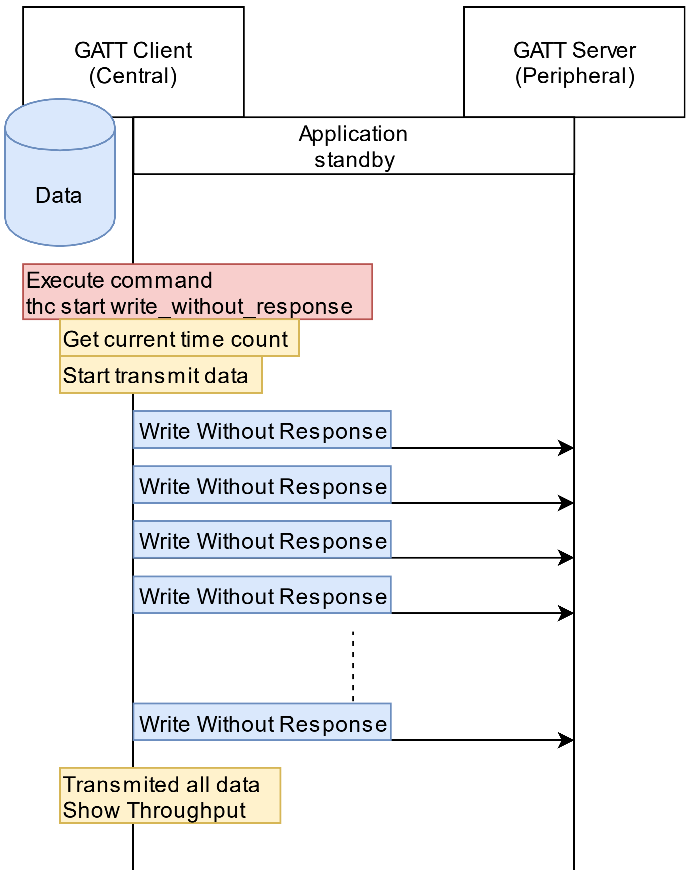


Figure 1.9 Sequence chart when executing Write Without Response command

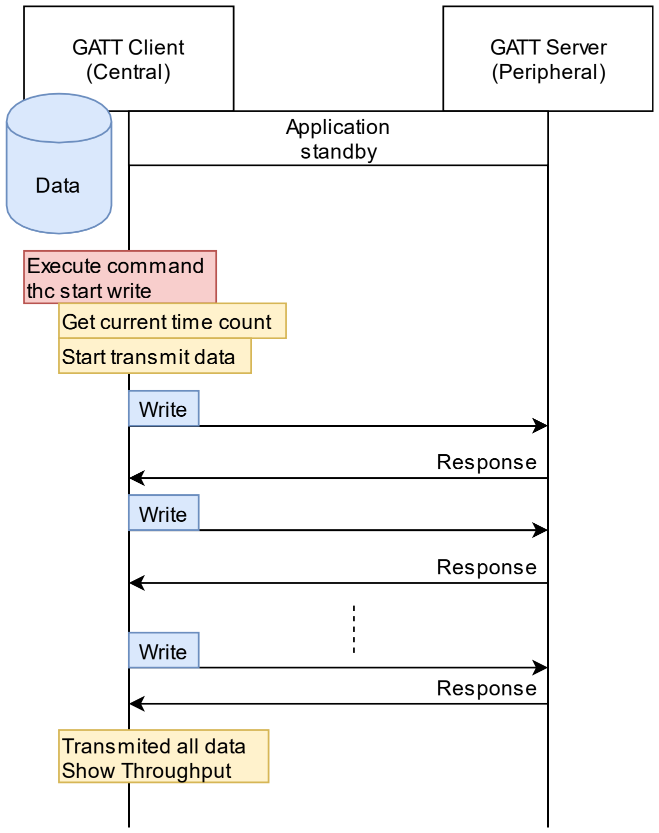
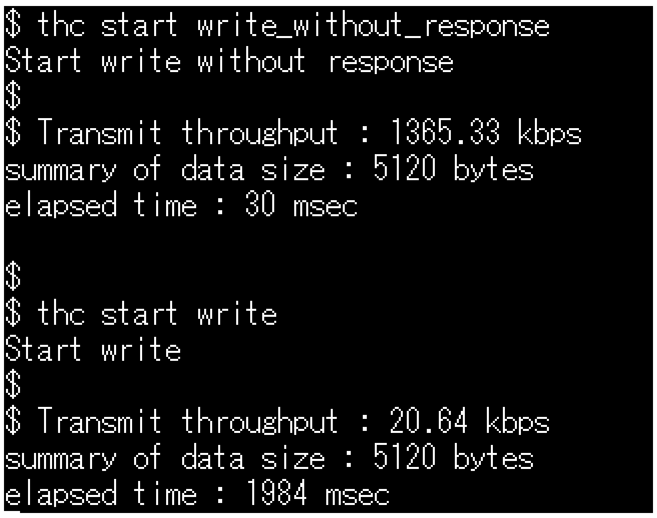


Figure 1.10 Sequence chart when executing Write command



```
$ thc start write_without_response
Start write without response
$
$ Transmit throughput : 1365.33 kbps
summary of data size : 5120 bytes
elapsed time : 30 msec

$
$ thc start write
Start write
$
$ Transmit throughput : 20.64 kbps
summary of data size : 5120 bytes
elapsed time : 1984 msec
```

Figure 1.11 CLI display when transmitting the data from client to server

1.3.3 Changing GAP parameters

The sample programs can control the GATT operation as well as change the GAP parameters that affect the throughput. The parameters and commands that can be changed from the CLI of the sample program are shown in Table 1.6. These commands are used in the CLI feature in the BLE Protocol Stack. For details, refer to [5.1.1.1 GAP command] in “Bluetooth Low Energy Protocol Stack Basic Package User’s Manual(R01UW0205)”.

Table 1.6 Parameters and commands that affect throughput

GAP parameters	commands	parameters
Connection Interval	gap conn_cfg update	connection handle connection interval slave latency supervision timeout
PHY	gap conn_cfg phy	connection handle tx phy rx phy coded scheme
Max Packet Length	gap conn_cfg data_len	connection handle max tx packet length max tx time
Encryption	gap auth start	connection handle

1.4 Custom (Throughput) Service

The Sample programs communicate to application data by using a custom (throughput) service showed in Table 1.7. These characteristics in application program interface with the st_ble_seq_data_t structure defined by the Profile Common Library in app_lib provided by BLE FIT Module.

Table 1.7 Throughput Service

name	UUID	Properties
Throughput Service	9CEF3D10-7FAB-49DC-AB89-762C9079FE96	---
Throughput Data 1 Characteristic	9CEF3D11-7FAB-49DC-AB89-762C9079FE96	Write Write Without Response
Throughput Data 2 Characteristic	9CEF3D12-7FAB-49DC-AB89-762C9079FE96	Indicate Notify
Client Characteristic Configuration Descriptor	0x2902	Read Write

1.5 Software Structure

The sample programs are developed by using BLE solution tool composed of QE for BLE and BLE FIT Module. Figure 1.12 shows a component diagram for the sample program. Callback functions to receive event form R_BLE_API, service API and other module are implemented in throughput application.

The sample programs use the TxFlow control program to realize continuous transmission operation. Also, the Highspeed GAP Config program is used for performing to make GAP settings for highspeed communication.

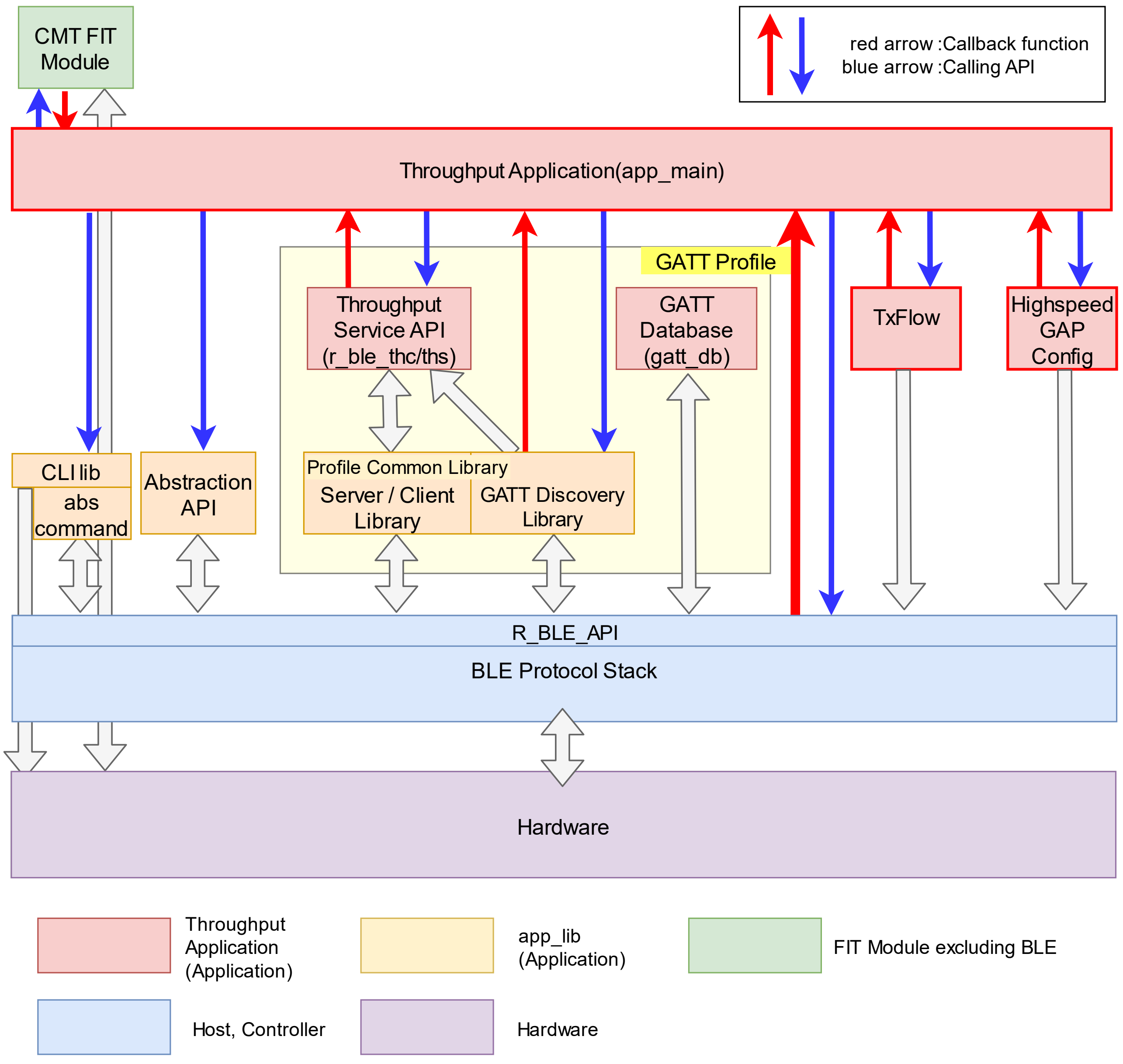


Figure 1.12 Component diagram in sample program

1.6 Source File Structure

Figure 1.13 shows the file structure of the sample program. The sample programs are created using Smart Configurator of e^2 studio. The description on the right side of the file name indicates the following.

(A)：Source code that added and modified to realize application.

(QE for BLE)：Source code generated by QE for BLE.

(SC)：Other source codes generated by Smart Configurator.

(Server)：The server programs.

(Client)：The client programs.

```
-- Config_BLE_PROFILE (QE for BLE)
        -- app_main.c                          (QE for BLE) (A) (Server, Client)
           Application framework.
        -- gatt_db.c                           (QE for BLE) (Server, Client)
        -- gatt_db.h                           (QE for BLE) (Server, Client)
           GATT Database.
        -- r_ble_gapc.c                        (QE for BLE) (Server, Client)
        -- r_ble_gapc.h                        (QE for BLE) (Server, Client)
        -- r_ble_gaps.c                        (QE for BLE) (Server, Client)
        -- r_ble_gaps.h                        (QE for BLE) (Server, Client)
        -- r_ble_gatc.c                        (QE for BLE) (Client)
        -- r_ble_gatc.h                        (QE for BLE) (Client)
        -- r_ble_gats.c                        (QE for BLE) (Server)
        -- r_ble_gats.h                        (QE for BLE) (Server)
           GAP, GATT Service API.
        -- r_ble_thc.c                         (QE for BLE) (Client)
        -- r_ble_thc.h                         (QE for BLE) (Client)
        -- r_ble_ths.c                         (QE for BLE) (Server)
        -- r_ble_ths.h                         (QE for BLE) (Server)
           Service API for throughput service.
        -- r_ble_highspeed_gap_config.c        (A) (Client)
        -- r_ble_highspeed_gap_config.h        (A) (Client)
           GAP configuration program for highspeed communication.
        -- r_ble_txflow.c                      (A) (Server, Client)
        -- r_ble_txflow.h                      (A) (Server, Client)
           Tx flow control program
-- general                        (SC)
-- r_ble_qe_utility               (SC)
-- r_ble_rx23w                    (SC)
        BLE FIT Module
-- r_bsp                          (SC)
-- r_byteq                        (SC)
-- r_cmt_rx                       (SC)
-- r_config                       (SC)
-- r_flash_rx                     (SC)
-- r_gpio_rx                      (SC)
-- r_irq_rx                       (SC)
-- r_lpc_rx                       (SC)
-- r_pincfg                       (SC)
-- r_sci_rx                       (SC)
```

Figure 1.13 File structure in sample program

2. Programs for highspeed communication.

In order to realize highspeed communication with Bluetooth Low Energy, firstly, it is necessary to make continuous transmission requests for communication by More Data, secondly to optimize GAP settings.

This application note provides two programs to facilitate these operations. The sample programs use these programs to achieve highspeed communication.

2.1 TxFlow control program

The TxFlow control program uses the Tx flow control feature of the Vendor Specific API provided by the BLE protocol stack. This program executes callback function to transmit when the transmit buffer has space.

For the Tx flow control feature of the BLE protocol stack, refer “R_BLE API document (r_ble_api_spec.chm)” included in the “RX23W Group BLE Module Firmware Integration Technology Application Note (R01AN4860)”.

The application realizes highspeed communication by receiving this callback function and continuously transmitting.

The type definitions and functions of this program are shown in Table 2.1 and Table 2.2.

Table 2.1 Type definition in TxFlow control program

```
typedef void(*r_txflow_cb_t)(void)
```

Table 2.2 The Function list in TxFlow control program

Function name	behavior
R_BLE_TxFlow_Init	Initialize the program and set up a callback function to receive transmittable event.
R_BLE_TxFlow_Start	Start notification of transmittable event.
R_BLE_TxFlow_Stop	Stop notification of transmittable event
R_BLE_TxFlow_TxFlowChg	This is a function for passing events from the Tx flow control feature. Called within the callback function of the Vendor Specific API.

Figure 2.1 shows an operation example of an application that performs continuous transmission using this program.

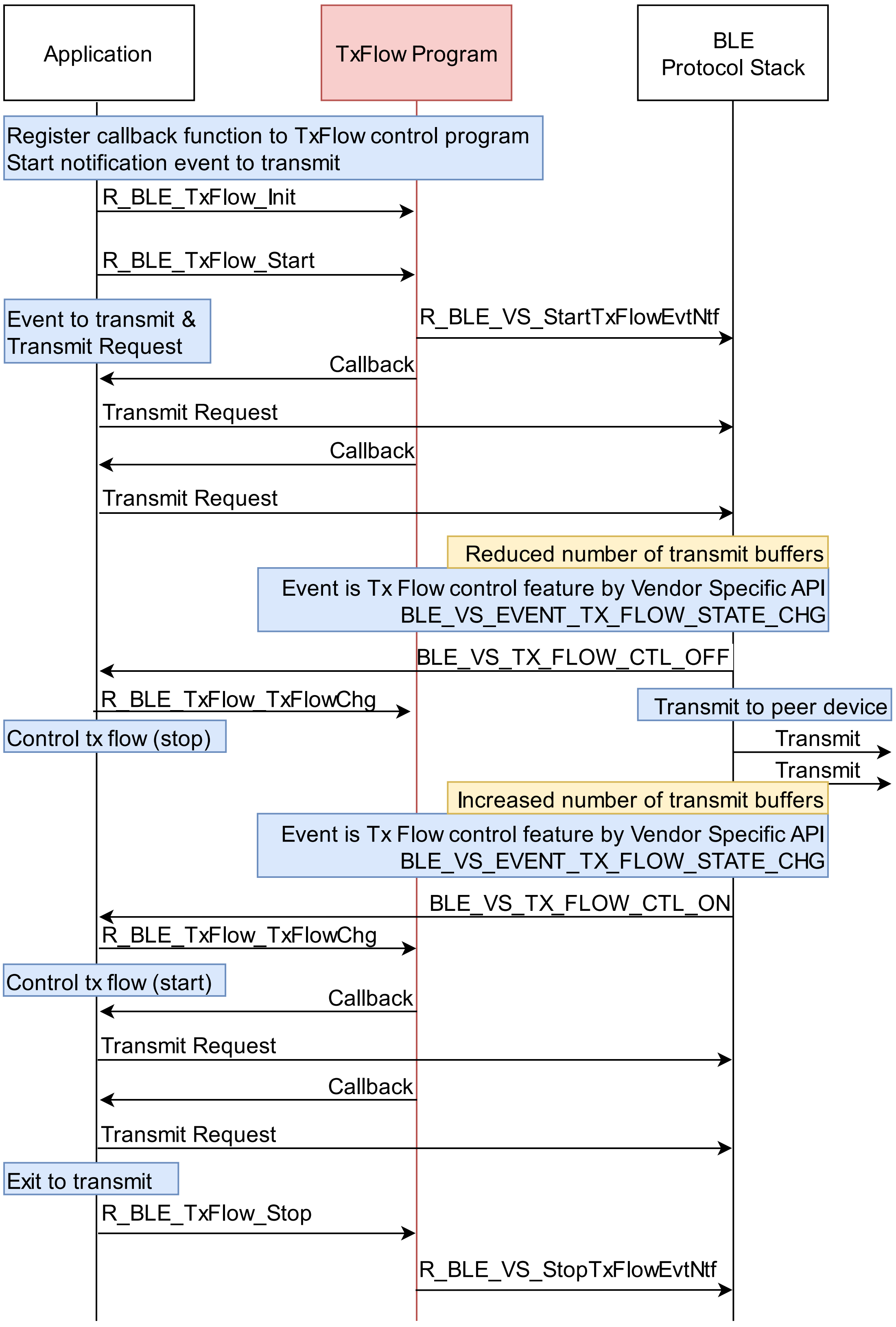

Application
TxFlow Program
BLE Protocol Stack
Register callback function to TxFlow control program
Start notification event to transmit
R_BLE_TxFlow_Init
R_BLE_TxFlow_Start
Event to transmit & Transmit Request
R_BLE_VS_StartTxFlowEvtNtf
Callback
Transmit Request
Callback
Transmit Request
Reduced number of transmit buffers
Event is Tx Flow control feature by Vendor Specific API BLE_VS_EVENT_TX_FLOW_STATE_CHG
BLE_VS_TX_FLOW_CTL_OFF
R_BLE_TxFlow_TxFlowChg
Transmit to peer device
Control tx flow (stop)
Transmit
Transmit
Increased number of transmit buffers
Event is Tx Flow control feature by Vendor Specific API BLE_VS_EVENT_TX_FLOW_STATE_CHG
BLE_VS_TX_FLOW_CTL_ON
R_BLE_TxFlow_TxFlowChg
Control tx flow (start)
Callback
Transmit Request
Callback
Transmit Request
Exit to transmit
R_BLE_TxFlow_Stop
R_BLE_VS_StopTxFlowEvtNtf

Figure 2.1 Operation example of TxFlow control program

2.1.1 API Reference

Table 2.3 shows details of the TxFlow control program functions.

Table 2.3 API Reference of this program

R_BLE_TxFlow_Init(r_txflow_cb_t cb)		
Initialize the program and register the callback function. The callback function registered here will continue to be called when there is space in the BLE protocol stack transmit buffer. * The registered callback function will be called repeatedly with one R_BLE_Execute () until notification is stopped by R_BLE_TxFlow_Stop or R_BLE_TxFlow_TxFlowChg.		
Parameters:		
r_txflow_cb_t	*cb*	Callback function that notifies events to transmit.
Return:		
r_ble_status_t	*BLE_SUCCESS*	Success

R_BLE_TxFlow_Start(void)		
Start notification of transmittable event.		
Parameters:		
None		
Return:		
r_ble_status_t	*BLE_SUCCESS*	Success
	BLE_ERR_INVALID_OPERATION	Notification has already started.
	BLE_ERR_INVALID_PTR	The callback function has not been registered.

R_BLE_TxFlow_Stop(void)		
Stop notification of transmittable event.		
Parameters:		
None		
Return:		
None		

R_BLE_TxFlow_TxFlowChg(st_ble_vs_tx_flow_chg_evt_t* evt_data)		
This function receives the variable data for the event (BLE_VS_EVENT_TX_FLOW_STATE_CHG) by the TxFlow control feature of the Vendor Specific API. This function is called in Vendor Specific callback function. If you do not use this function to acquire the transmission flow control event, the TxFlow control program will not operate normally.		
Parameters:		
st_ble_vs_tx_flow_chg_evt_t*	*evt_data*	BLE_VS_EVENT_TX_FLOW_STATE_CHG event data in Vendor Specific API.
Return: r_ble_status_t		
r_ble_status_t	*R_BLE_SUCCESS*	success

2.2 Highspeed GAP Config Program

The Highspeed GAP Config program changes the setting of GAP parameter to the optimum value to realize highspeed communication.

Table 2.4 shows the GAP parameters and the set values changed by this program.

Table 2.4 GAP settings for high-speed communication

parameter	value
Connection Interval	50 (msec)
PHY	2M PHY
Max packet length	251 (byte)

The type definitions and functions of this program are shown in Table 2.5 and Table 2.6.

Table 2.5 Type definition in Highspeed GAP Config program

```
typedef void(*r_highspeed_confgap_comp_cb_t)(uint16_t , ble_status_t);
```

Table 2.6 The Function list in Highspeed GAP Config program

Function name	behavior
R_BLE_Highspeed_ConfigGapConnPram	Register the callback function that receives the completion notification and start the GAP setting. The callback function is called when the GAP setting is completed, or it is disconnected.
R_BLE_Highspeed_GapCb	This function is called in the GAP API callback function and receive the GAP event.

Figure 2.2 shows the operation sequence of this program. This program changes the connection interval, PHY, and maximum packet length in this order. When the maximum packet length has set, the callback function registered in R_BLE_Highspeed_ConfigGapConnPram is called and is passed BLE_SUCCESS. If the connection is disconnected during setting, the callback function is called and is passed BLE_ERR_DISCONNECTED.

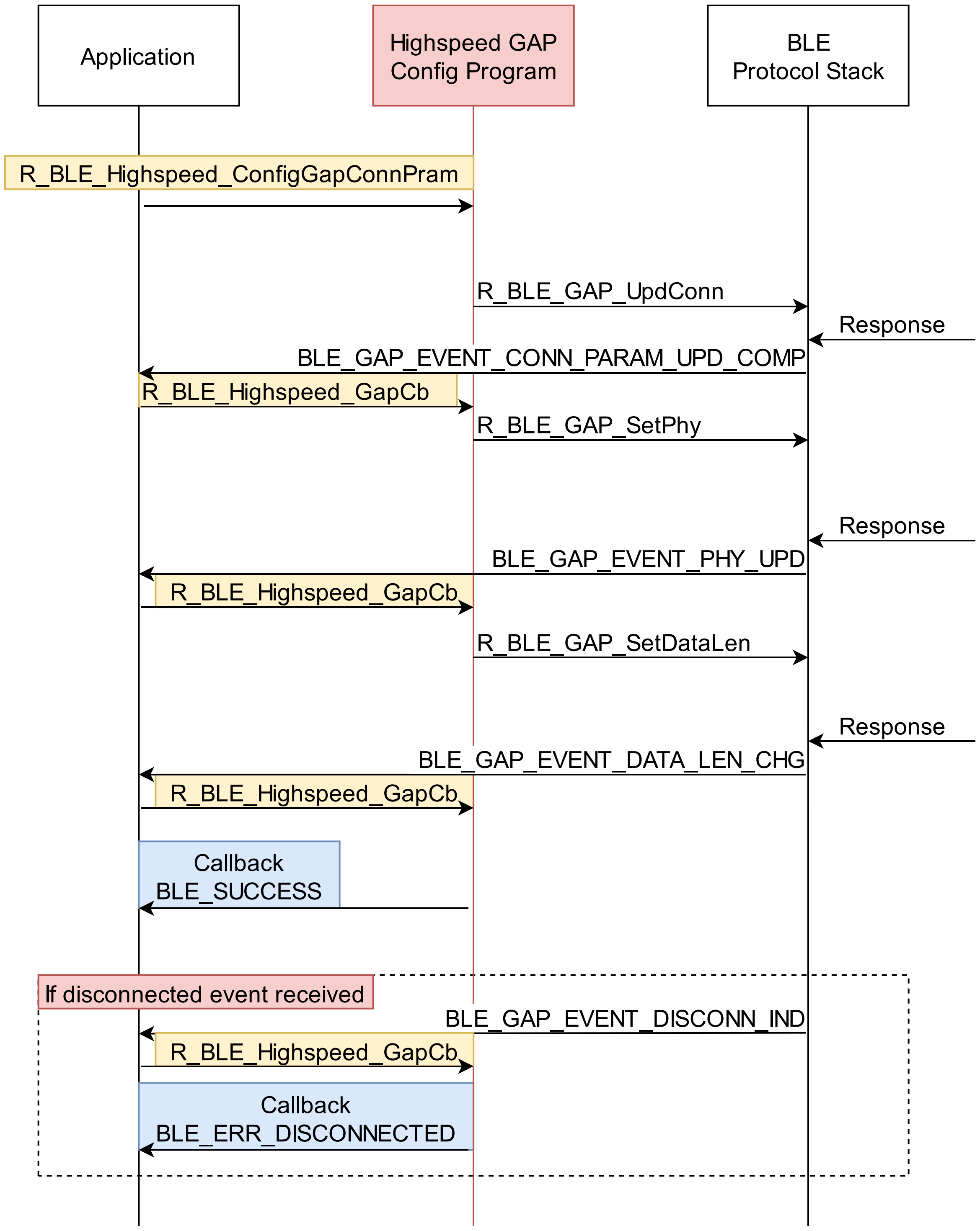


Figure 2.2 behavior in Highspeed GAP Config program

2.2.1 API Reference

Table 2.7 shows details of the Highspeed GAP Config program functions.

Table 2.7 API reference in Highspeed GAP Config program

R_BLE_Highspeed_ConfigGapConnParam(uint16_t conn_hdl, r_highspeed_confgap_comp_cb_t cb)		
Register the callback function that receives the completion notification and start the GAP setting.		
Parameters:		
uint16_t	*conn_hdl*	The connection handle of the connection for which GAP settings are made.
r_highspeed_confgap_comp_cb_t	*cb*	This parameter is the callback function pointer to receive the notification that the GAP setting has completed. BLE_SUCCESS is passed when GAP setting is completed BLE_ERR_DISCONNECTED is passed when disconnected during configuring.
Return:		
r_ble_status_t	*BLE_SUCCESS*	Success
	BLE_ERR_INVALID_OPERATION	Already started GAP setting.
	BLE_ERR_INVALID_PTR	Callback function pointer is null.

R_BLE_Highspeed_GapCb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)		
Receive events from the GAP API.		
Parameters:		
uint16_t	*type*	GAP event code defined by GAP API.
ble_status_t	*result*	The result of an event defined by the GAP API.
st_ble_evt_data_t *	*p_data*	This data corresponds to the event defined by GAP API.
Return:		
None		

3. Bluetooth Low Energy and Throughput

In this chapter, we explain briefly the relationship between the Bluetooth low energy communication mechanism and throughput. For details on communication standards, please refer to the Bluetooth specifications. Bluetooth Low Energy has three major layers. Controller and Host are connected by Host Controller Interface (HCI). Application and Host are connected by API (R_BLE_API in BLE FIT Module) (Figure 3.1)

In Bluetooth Low Energy, the Link Layer of the controller controls the actual communication path and transmission / reception interval. The operation of this Link Layer is important to achieve highspeed communication. The behavior of the Link Layer is set by the GAP of the Host Layer.

On the other hand, when sending meaningful data for application, the GATT of the host layer is used. In GATT, the profile determines the application data transmission procedure and the data structure to be transmitted and received. The design of this profile is also important for achieving highspeed communication.

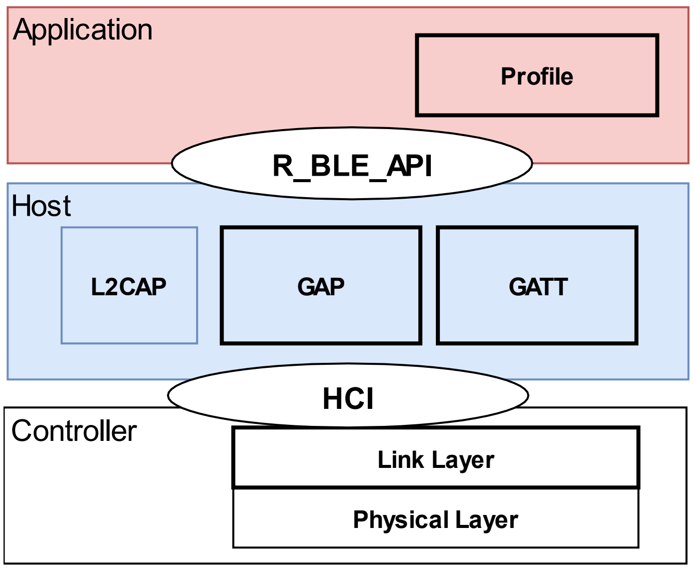


Figure 3.1 Three major layer in Bluetooth Low Energy

3.1 Generic Access Profile (GAP)

Generic Access Profile (GAP) defines the procedure for detecting connectable devices and establishing connections. GAP sets the operation of Link Layer and realizes these procedures.

3.1.1 Device detection and connection establishment

In Bluetooth Low Energy, a connection is established by one device transmitting (advertising) its own device information and the other device performing device detection (scanning) and connection request (initiating). The device that performs scanning and connection request is the central device, and the device that advertises is the peripheral device. Central determines parameters related to connection maintenance such as frequency map and communication interval (connection interval) after connection is established. The GAP will manage the following information.

- Connection Interval
- PHY
- Maximum Packet Length
- Information for Pairing
- etc.

3.1.2 Communication after establishing connection

In Bluetooth Low Energy, after the connection is established, the device exchanges radio frames with a connection event that occurs at each connection interval. In the Link Layer, the central is the master and the peripherals are the slaves. Radio frames are transmitted by the master in time with pre-shared connection events. (Figure 3.2)

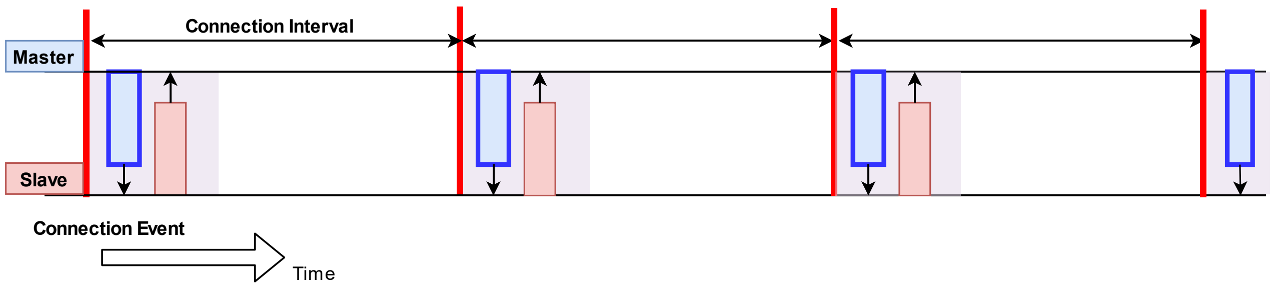


Figure 3.2 Exchange of connection event and radio frame

The connection is maintained by exchanging radio frames at the connection event. If there is additional data to send to either device, the More Data Bit in the radio frame will be set and the connection event will be extended. The connection event ends when the More Data Bit of each other is no longer set or when an error occurs in the received packet. Once the connection event ends, radio frames are not exchanged until the next connection event (Figure 3.3). In order to realize highspeed communication, it is important to communicate using this More Data.

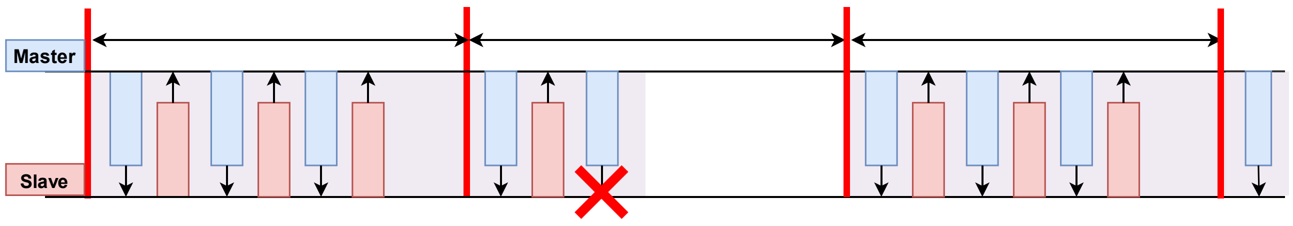


Figure 3.3 Communication by More Data

3.1.3 Setting the connection interval

Figure 3.4 shows a schematic diagram of Link Layer operation when the connection interval is changed. Even if the connection interval is changed, if communication by More Data is stable, there will be no significant change in throughput. Note that If the connection interval is shortened extremely, the overhead of waiting time for each interval will hinder throughput.

Figure 3.5 shows the relationship between the connection interval and throughput assuming that the communication environment is good and frame exchange is always successful. The settings of GAP and GATT are PHY: 2M PHY, maximum packet length is 251 bytes, MTU is 247 bytes, and 244 bytes of application data are always notified. If the connection interval is 7.5msec, it will be about 1040kbps.

The throughput per connection interval is calculated from the waiting time T_{overhead} immediately before the connection event, the minimum transfer time T_{frame} for the radio frame to make a round trip, and the application data length (L_{data}). If the packet length is 251 bytes, T_{frame} will be about 1.408 msec.

$$Throughput\ (kbps) = floor\left(\frac{connection\ interval - T_{overhead}}{T_{frame}}\right) * 8 * L_{\text{data}} * \frac{1}{connection\ interval}$$

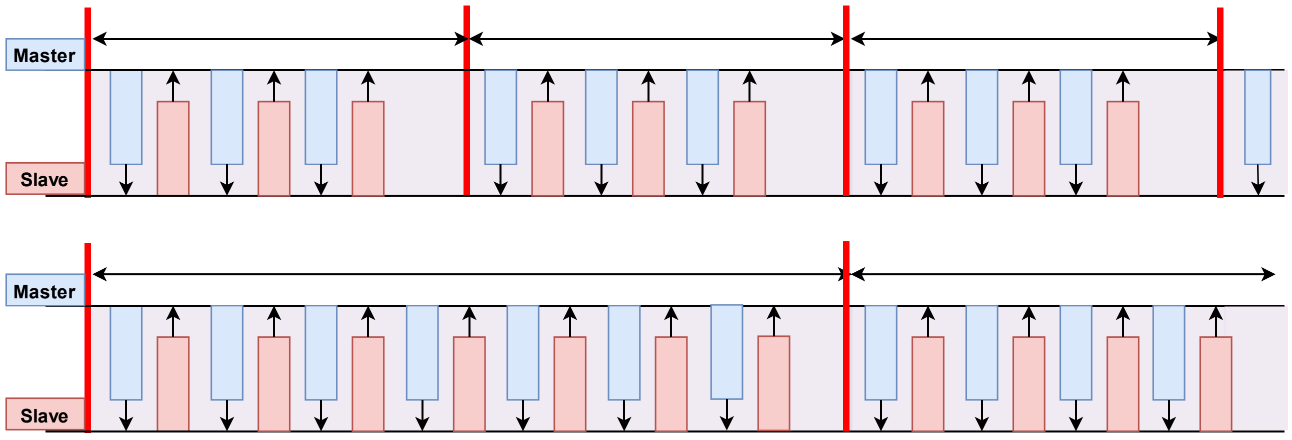


Figure 3.4 Change in connection interval and number of radio frames

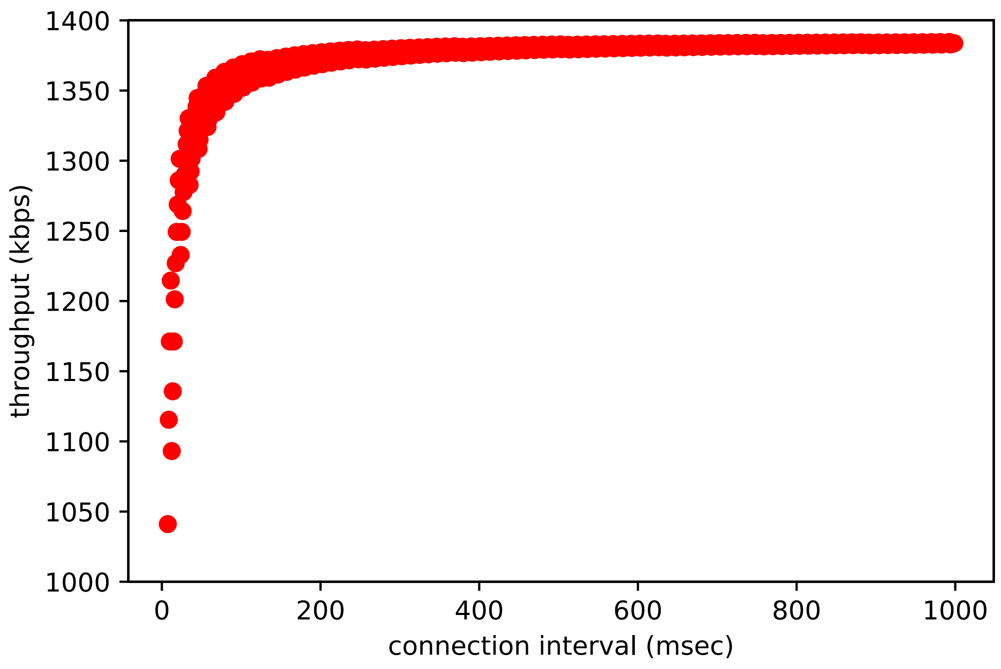


Figure 3.5 The relationship connection interval and throughput

If the communication environment is good, the throughput will not be affected even if the connection interval becomes long, but if the communication by using the More Data bit is interrupted due to a communication error, the difference in the connection interval will have a large effect (Figure 3.6). When a communication error occurs, each device waits until the next connection event, so if the connection interval increases, the throughput decreases.

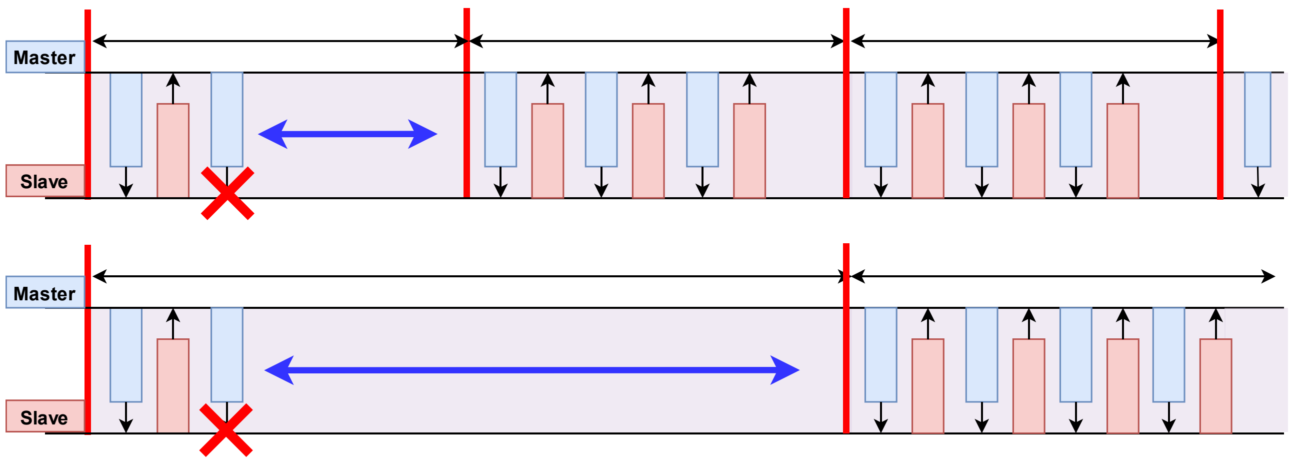


Figure 3.6 Connection interval and throughput when communication error occurs

Figure 3.7 shows the relationship between connection interval, probability of frame exchange failure, and throughput. GAP settings are PHY: 2M PHY, maximum packet length 251 bytes, MTU 247 bytes, and the value when 244 bytes of application data are always notified. The expected value of throughput per connection interval is plotted.

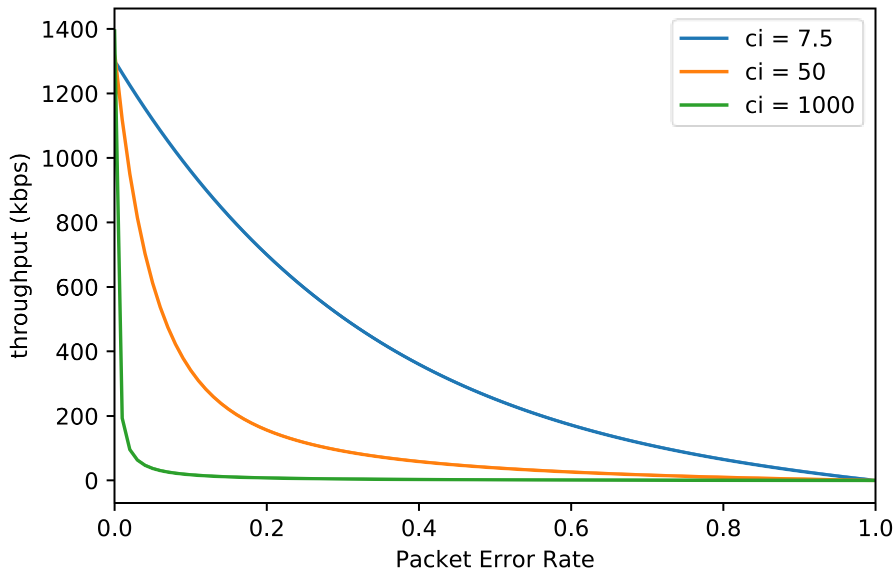


Figure 3.7 Relationship between frame exchange failure probability and throughput

To change the connection interval, execute the gap conn_cfg update command. To change using R_BLE_API, use R_BLE_GAP_UpdConn. For details about the API, refer to "R_BLE API document (r_ble_api_spec.chm)" included in "RX23W Group BLE Module Firmware Integration Technology Application Note (R01AN4860)".

3.1.4 Setting the PHY

Figure 3.8 shows a schematic diagram of Link Layer operation when the physical layer (PHY) settings are changed. When the physical layer PHY is changed, the air frame occupation time changes. If the data length is the same, the air occupation time will be about half in 1M PHY, as it is in 2M PHY. If the occupied time of one frame in the air is short, the number of packets transmitted / received per unit time increases, and the throughput improves.

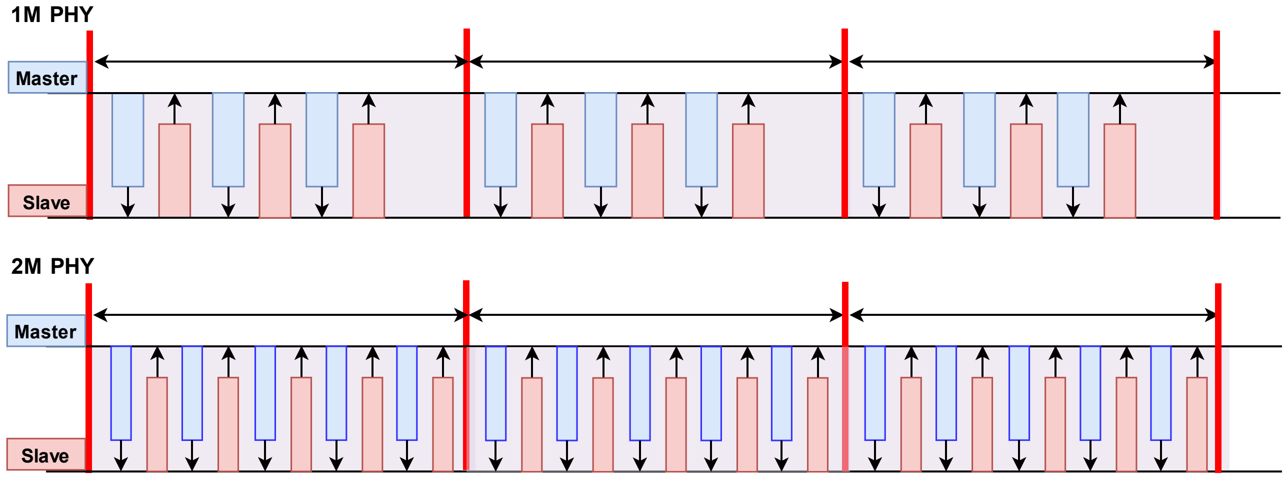


Figure 3.8 Schematic diagram when using 2M PHY

To change the PHY, execute the gap conn cfg phy command. To change using R_BLE_API, use R_BLE_GAP_SetPhy. For details about the API, refer to "R_BLE API document (r_ble_api_spec.chm)" included in "RX23W Group BLE Module Firmware Integration Technology Application Note (R01AN4860)".

3.1.5 Setting the Maximum packet length

Figure 3.9 shows a schematic diagram of Link Layer operation when the maximum packet length is set to a high value and data with a large packet length is transmitted. The application information can be efficiently transmitted by minimizing the header information of the radio frame and the transmission / reception interval.

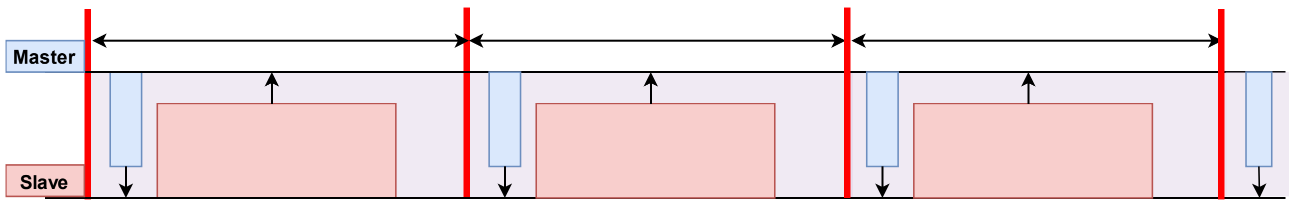


Figure 3.9 Schematic diagram of Link Layer when changing packet length

To change the Maximum packet length, execute the gap conn_cfg phy command. To change from R_BLE_API, use R_BLE_GAP_SetDatalen. For details about the API, refer to "R_BLE API document (r_ble_api_spec.chm)" included in "RX23W Group BLE Module Firmware Integration Technology Application Note (R01AN4860)".

3.1.6 Setting the encryption of communication

Figure 3.10 shows a schematic diagram of Link Layer operation when communication is encrypted. Through encryption, the data for checking packet integrity (4 bytes) is carried in the radio frame, which may reduce the throughput.

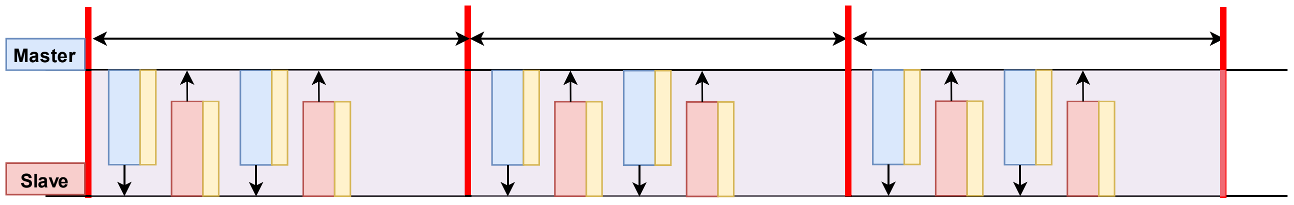


Figure 3.10 Schematic diagram of Link Layer in encrypted communication

To encrypt communication, execute the gap auth start command. When performing encryption from R_BLE_API, use R_BLE_GAP_StartEnc or R_BLE_GAP_StartPairing. You can also use R_BLE_ABS_StartAuth of Abstraction API of app_lib. For details on these APIs, refer to the “R_BLE API document (r_ble_api_spec.chm)” included in “RX23W Group BLE Module Firmware Integration Technology Application Note (R01AN4860)”.

3.2 Generic Attribute Profile (GATT)

In Bluetooth Low Energy, the information for detecting and connecting (advertising and scanning, initiating) a communication device and for continuous communication after the connection is made is managed by the GAP.

On one hand, the communication procedure of application data (sensor data etc.) is determined by Generic Attribute Profile (GATT). GATT implements a client-server architecture over the communication path established by GAP. The client reads / writes data from / to the GATT database held by the server using a predetermined procedure. At this time, the server returns a response to the client. On the other hand, it is also possible for the server to notify to the client (Figure 3.11). All applications that perform Bluetooth low energy data communication perform data communication according to GATT.

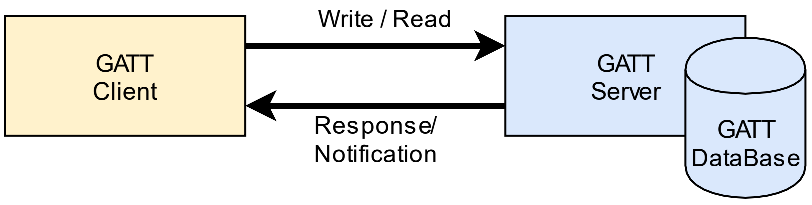


Figure 3.11 GATT architecture

In GATT, communication is focused on the feature of the application. The feature of an application is called a "service", and the data required for that feature is called a "characteristic." A "profile" is a set of features (services) required to realize an application and defines the communication specifications of the application.

When performing GATT communication, it is necessary to share in advance information about the feature (service) as an application and the data (characteristics) necessary to realize that feature. The server store information about the service it has and the characteristic that the service has in a database.

Figure 3.12 shows the relationship between profile, service, and characteristic when using a thermometer as an example. The features of the thermometer application are temperature measurement and device information. Features and data are kept on the GATT database as services and characteristics.

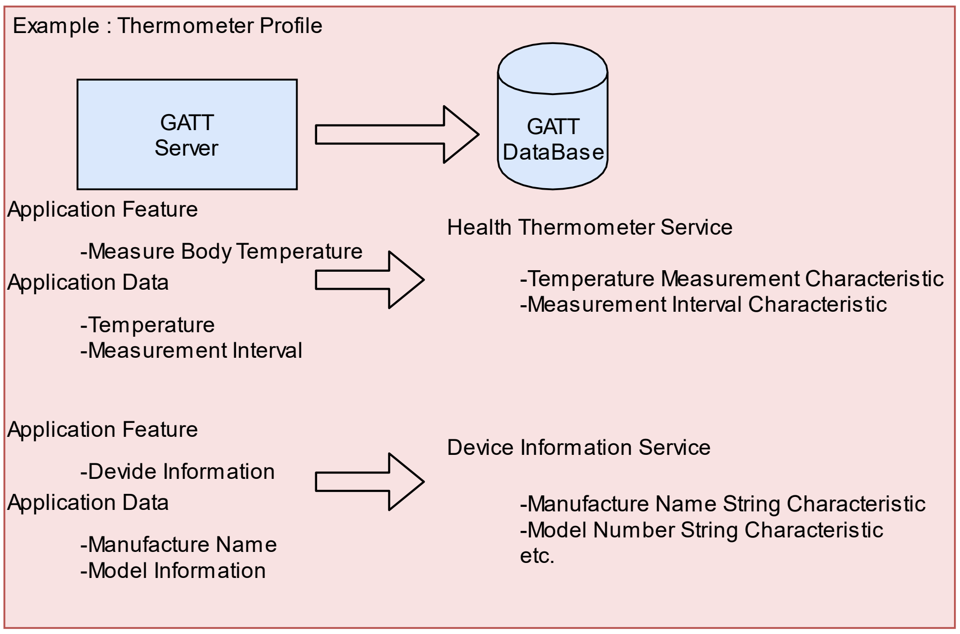

Example : Thermometer Profile
GATT Server
GATT DataBase
Application Feature
-Measure Body Temperature
Application Data
-Temperature
-Measurement Interval
Health Thermometer Service
-Temperature Measurement Characteristic
-Measurement Interval Characteristic
Application Feature
-Devide Information
Application Data
-Manufacture Name
-Model Information
Device Information Service
-Manufacture Name String Characteristic
-Model Number String Characteristic
etc.

Figure 3.12 Application Data and GATT database

Simply by establishing a connection, the client does not have service information for the server. The client queries the server for a particular service using a procedure called service discovery (Figure 3.13). This procedure gives the client information about the services the client wants on the server and handle information about the data in the database. The client uses this to handle information to read and write to the database.

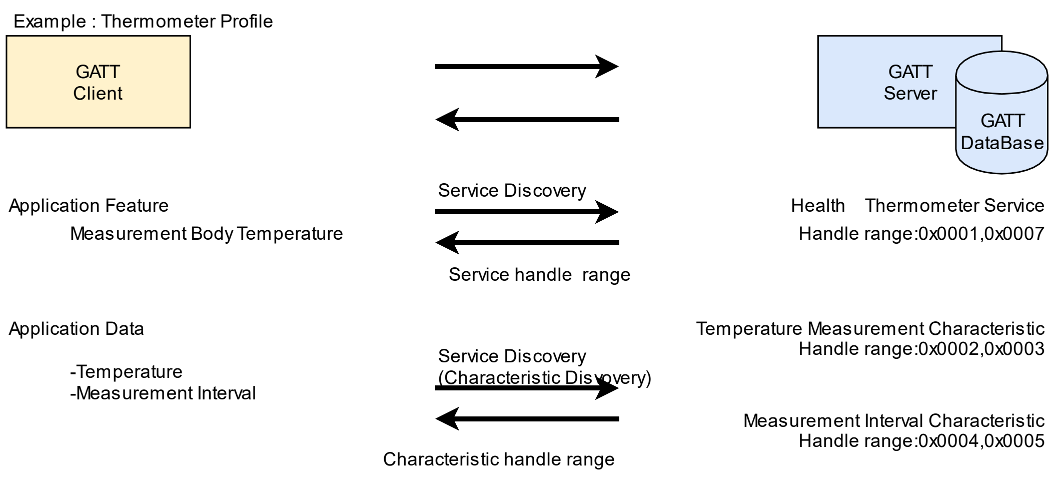

Example : Thermometer Profile
GATT Client
GATT Server
GATT DataBase
Application Feature
Measurement Body Temperature
Service Discovery
Service handle range
Health Thermometer Service
Handle range:0x0001,0x0007
Application Data
-Temperature
-Measurement Interval
Service Discovery (Characteristic Discovery)
Characteristic handle range
Temperature Measurement Characteristic
Handle range:0x0002,0x0003
Measurement Interval Characteristic
Handle range:0x0004,0x0005

Figure 3.13 Service discovery operation

In the characteristic, the data and its structure are decided, but in addition, the procedure of exchanging data between the client and the server is also decided. If there are additional options for the data and communication procedure determined by the characteristic, they are described in the characteristic descriptor.

Transmission and reception of characteristic data is performed according to the procedure determined by the characteristic. Table 3.1 summarizes the typical procedures. These procedures are characterized by the direction of data transmission and whether or not to wait for a response from the other. A procedure that requires a response cannot perform the same procedure before receiving a response from the other. Figure 3.14 shows a schematic diagram of the Read operation in which the client reads the server data. Data communication by GATT is performed based on the handle information.

Table 3.1 Typical communication procedure of GATT communication

Procedure name	operation	Direction to transmit	Response require
Read	Read	From client to server	Yes
Write	Write	From client to server	Yes
Write Without Response	Write	From client to server	No
Indication	Notify	From server to client	Yes
Notification	Notify	From server to client	No

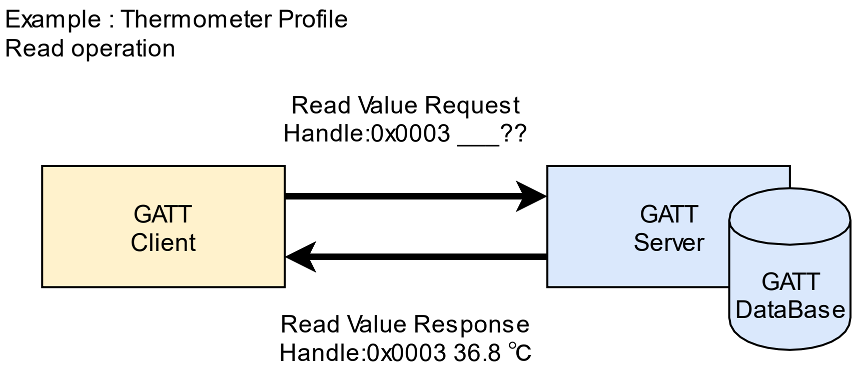


Figure 3.14 Read operation

3.2.1 No response operation (Notification / Write Without Response)

With Notification or Write Without Response operation, the next packet can be transmitted without waiting for the response from the opposition. Therefore, More Data communication can be performed by continuously sending transmission requests. Figure 3.15 shows a schematic diagram of the Notification operation in the Link Layer.

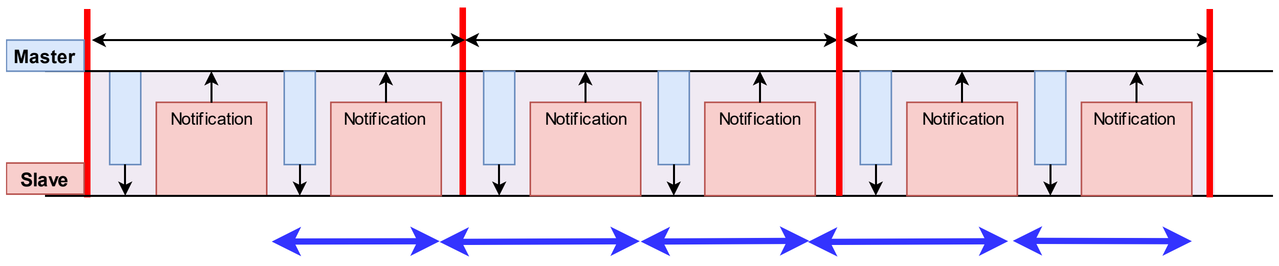


Figure 3.15 Schematic diagram of Link Layer during Notification operation

In the figure above, the slave acts as a server and performs a Notification. Slave and master are roles in the Link Layer and have no relationship with GATT role.

3.2.2 Response operation (Indication / Write)

In Indication and Write operations, after transmitting it is necessary to wait for the response from the other device before transmitting the next data. Therefore, a request to transmit the next data cannot be sent in one connection event, and more data communication cannot be performed. Figure 3.16 shows a schematic diagram of Link Layer operation for Indication. It takes twice as long as the connection interval to transmit one data packet.

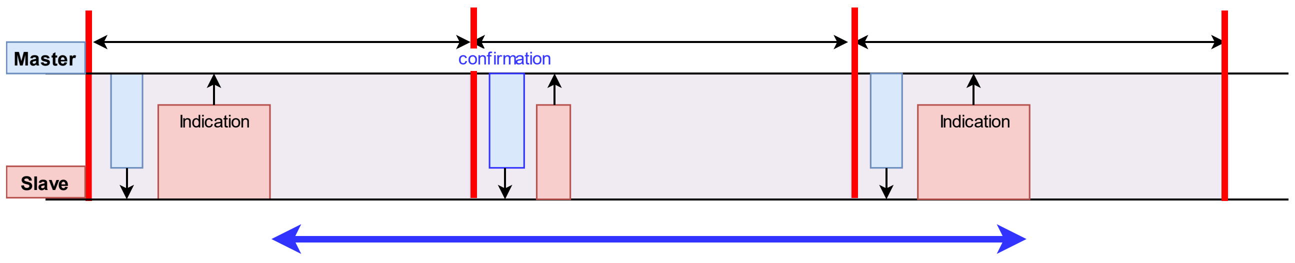


Figure 3.16 Schematic diagram of Link Layer during Indication operation

In the figure above, the slave acts as a server and performs Indication. Slave and master are roles in the Link Layer and have no relationship with GATT role.

4. BLE Solution tool

This sample program is created by using BLE FIT Module and QE for BLE. Generally, when developing a Bluetooth Low Energy application, in addition to developing the application, it is necessary to create a profile for exchanging application data. Some applications already have that profile defined by the Bluetooth SIG.

Figure 4.1 shows the relationship between BLE solution tool and Bluetooth Low Energy application.

BLE FIT Module is a FIT module for using the Bluetooth Low Energy feature of RX23W. The BLE FIT module consists of R_BLE_API for performing Bluetooth Low Energy communication and app_lib that assists application creation using this API.

QE for BLE is a tool for developing your GATT profile with a GUI. QE for BLE generates an application and profile framework to realize the developed profile by the code generation function.

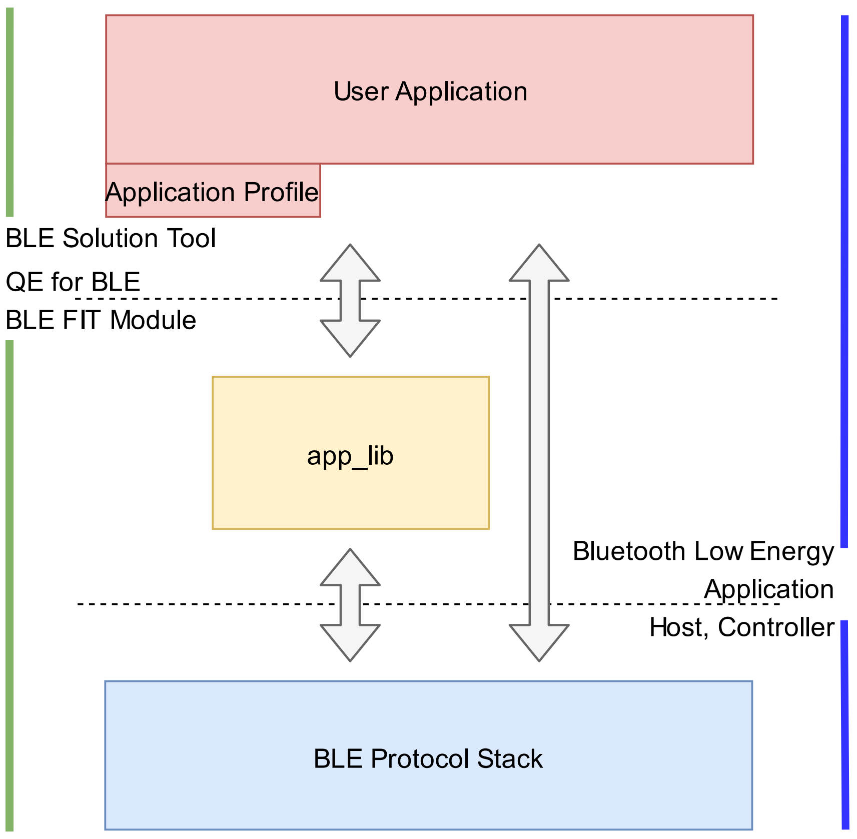


Figure 4.1 BLE Solution Tools and Bluetooth Low Energy Application Configuration

Figure 4.2 shows the relationship between the application framework generated by QE for BLE, app_lib, and R_BLE_API. The application framework has a program for the developed GATT profile and a part that performs the basic operation of GAP for Bluetooth Low Energy applications.

The basic operation of the GAP is realized by using the abstraction API of app_lib. The program for the developed GATT profile communicates using the generated service API and GATT database through Profile Common Library of app_lib.

Service API is an interface that connects the application and GATT. The Profile Common Library's GATT Discovery Library and the Profile Common Server / Client Library abstract GATT database information such as attribute handles and characteristic data structures. By using the service API, the application can communicate application data without being aware of the GATT database.

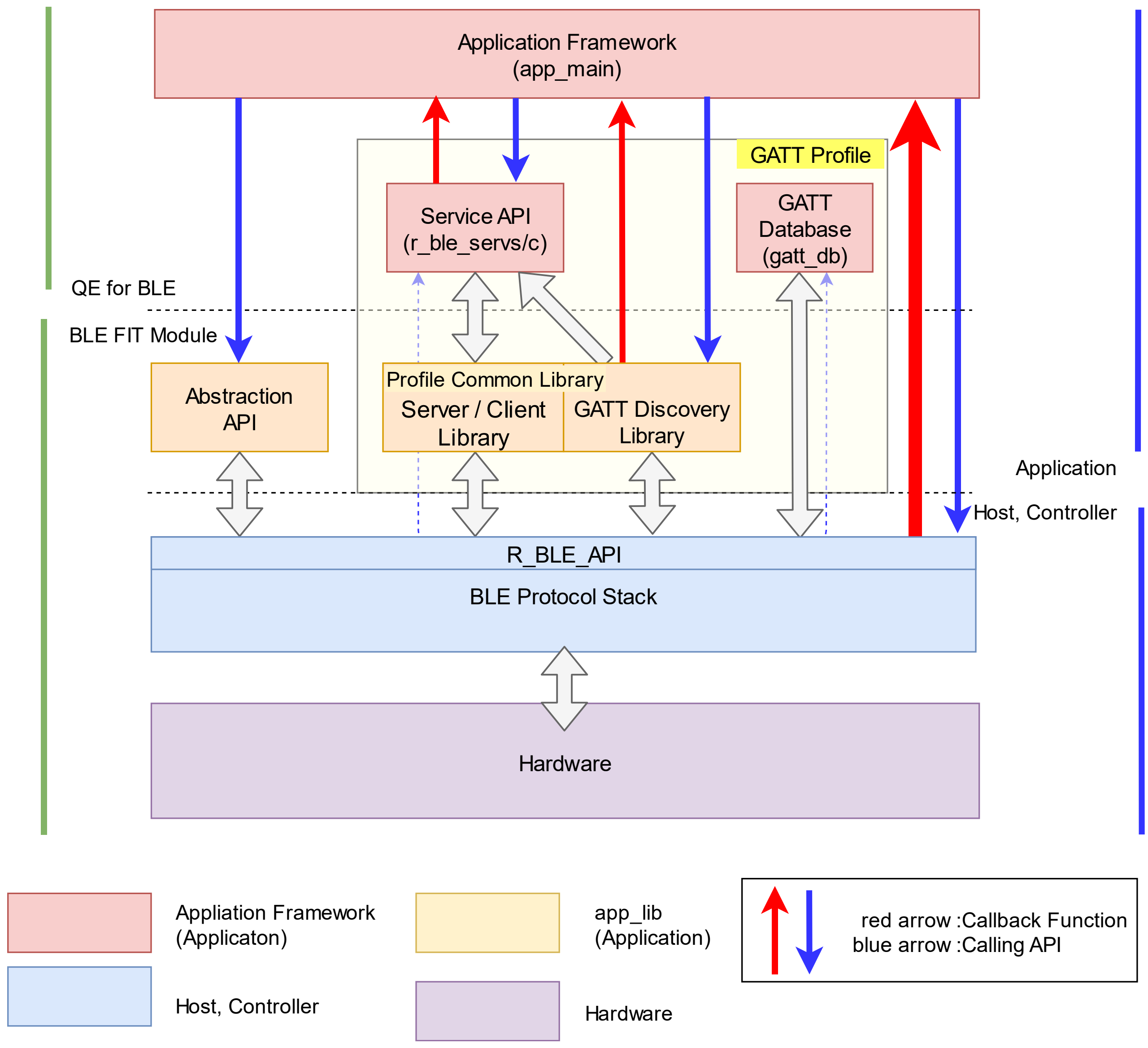


Figure 4.2 Application framework generated by QE for BLE

For app_lib, refer to Bluetooth Low Energy Protocol Stack Basic Package User's Manual (R01UW0205), and for QE for BLE, refer to "RX23W Group Bluetooth Low Energy Profile Developer’s Guide Application Note (R01AN4553)".

4.1 Designing profile by QE for BLE

This section introduces the method of developing profiles using QE for BLE as an example for the custom (throughput) service used in sample program

Throughput service has Throughput Data 1, the characteristic for Write / Write Without Response and Throughput Data 2, the characteristic for Notification / Indication operation (tab on the left side of Figure 4.3). These characteristics represent the data transmitted and received for throughput measurements.

To open this screen, open the {project_name} .scfg file from the Project Explorer and select [Config_BLE_Profile] from the component tab.

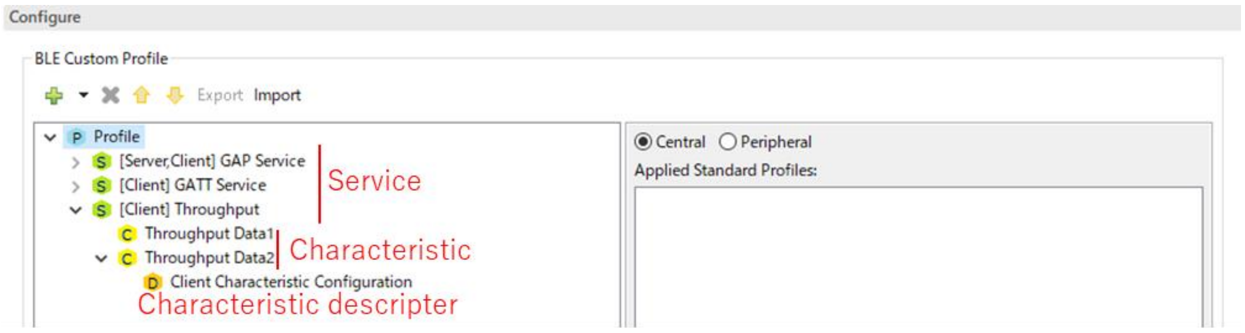


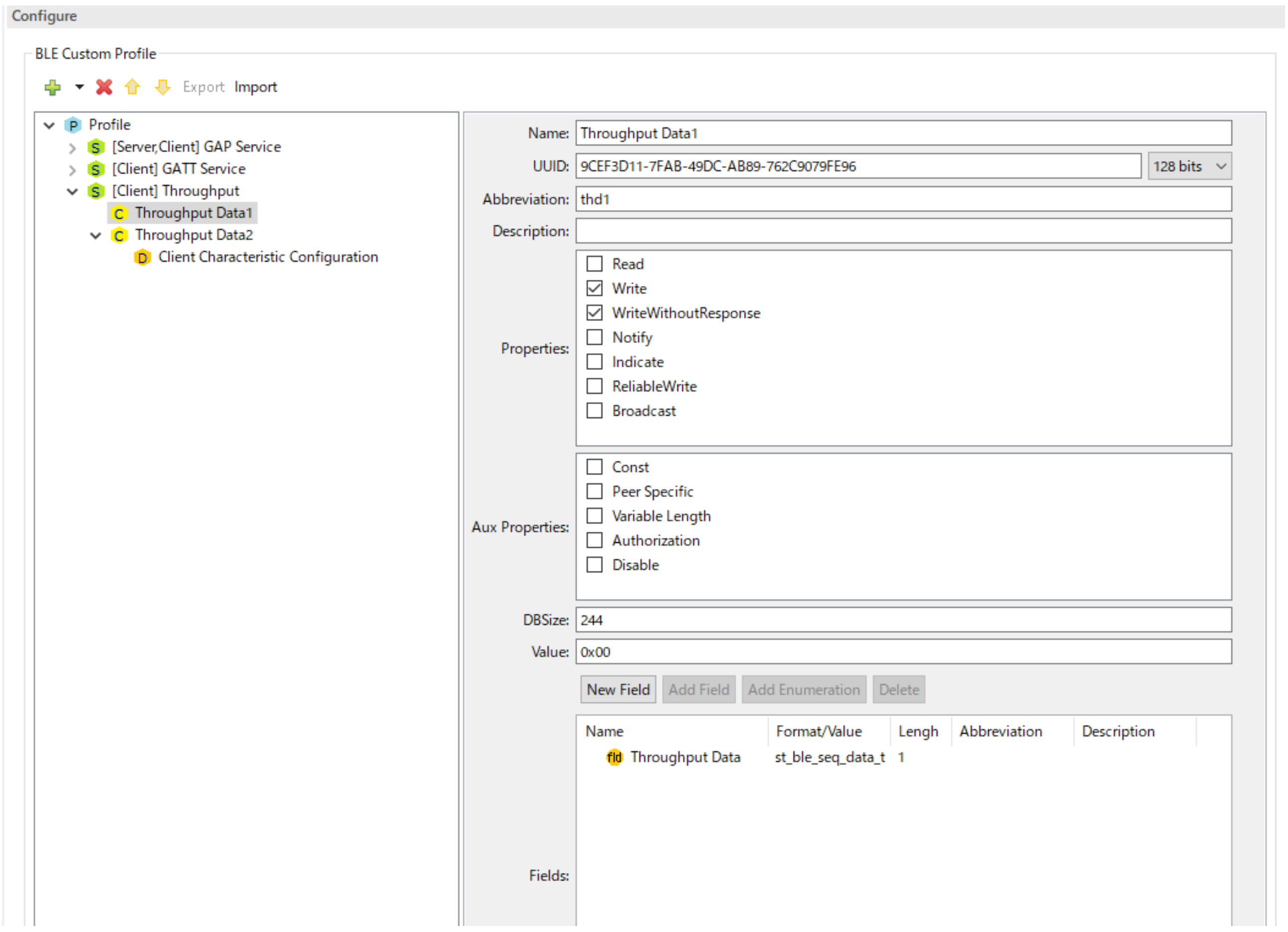
Figure 4.3 Profile configuration screen

Figure 4.4 shows the configuration screen of the Throughput Data 1 Characteristic of the throughput service. Set Name, UUID, Abbreviation and check Write and Write Without Response of Properties.

Aux Properties configure data management in the GATT database. There are no items to set for the throughput service.

Set DBSize and Value. DBSize sets the number of bytes reserved for storing data on the GATT database. The client does not have a GATT database, but this setting indicates the number of bytes reserved by the Profile Common Library when exchanging transmit / receive data from the host stack. Value represents the initial value of the characteristic.

Finally, set Fields. Field defines the characteristic structure handled in the application. This characteristic structure is reciprocally converted by the encode / decode functions set in the service API to packet data or GATT database data.



- Profile
  - [Server,Client] GAP Service
  - [Client] GATT Service
  - [Client] Throughput
    - Throughput Data1
    - Throughput Data2
      - Client Characteristic Configuration

Name: Throughput Data1
UUID: 9CEF3D11-7FAB-49DC-AB89-762C9079FE96 (128 bits)
Abbreviation: thd1
Description:
Properties: [ ] Read, [✓] Write, [✓] WriteWithoutResponse, [ ] Notify, [ ] Indicate, [ ] ReliableWrite, [ ] Broadcast
Aux Properties: [ ] Const, [ ] Peer Specific, [ ] Variable Length, [ ] Authorization, [ ] Disable
DBSize: 244
Value: 0x00
Fields:

Name	Format/Value	Lengh	Abbreviation	Description
Throughput Data	st_ble_seq_data_t	1		

Figure 4.4 Characteristic configuration screen of Throughput Data 1 Characteristic

The characteristics of the throughput service in sample application use the st_ble_seq_data_t structure defined by Profile Common Library. This structure has members that mean the start address of array data and length of array data. The encode / decode functions for this structure are implemented in the Profile Common Library. Therefore, you can transmit and receive array data without implementing the encode / decode functions.

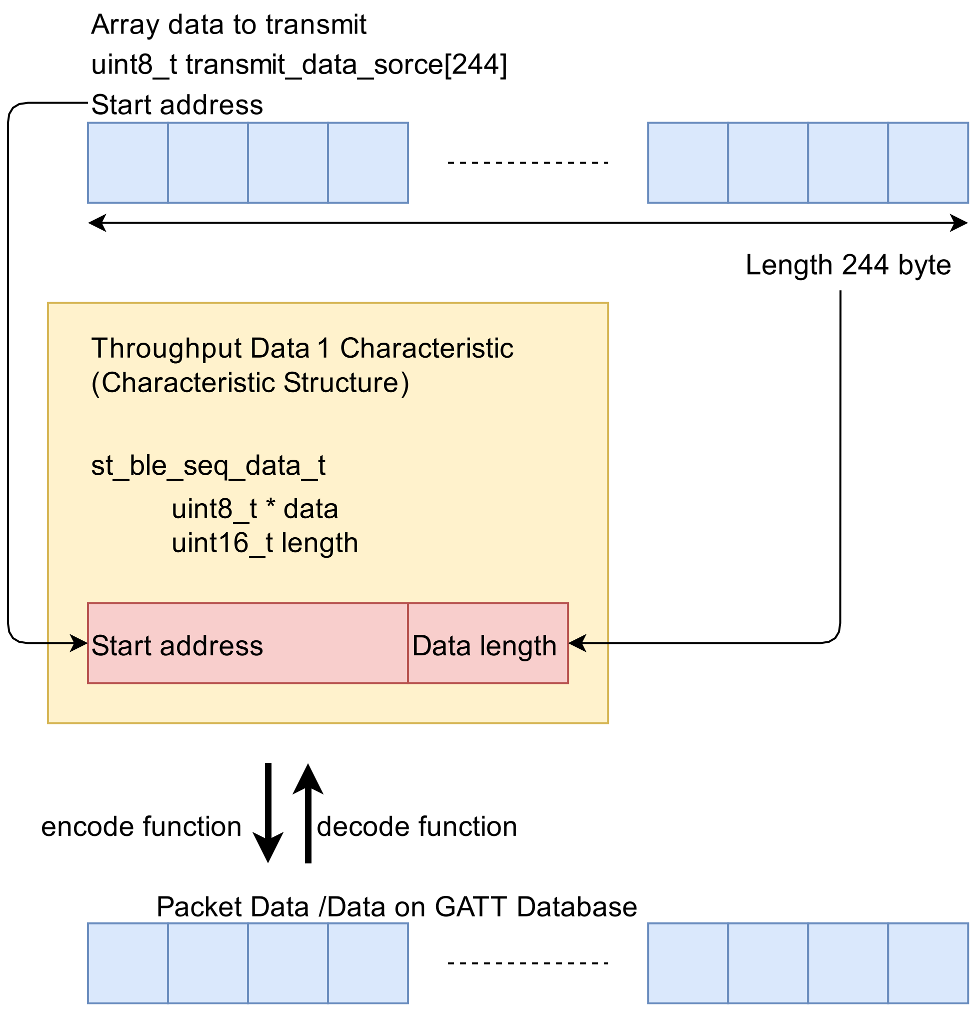


Figure 4.5 Transmitting array data using st_ble_seq_data_t structure

For details on how to develop a profile using QE for BLE, refer to "RX23W Group Bluetooth Low Energy Profile Developer’s Guide Application Note (R01AN4553)".

5. Appendix

5.1 How to program to the Target Board

The program can be written by connecting the USB cable by using the emulator circuit mounted on the target board.

The following shows how to write the program file to the Target Board using the Renesas Flash Programmer (RFP).

1. Change switch 2 of ESW1 to ON and connect your PC to the upper micro USB connector, as shown in Figure 5.1 below.

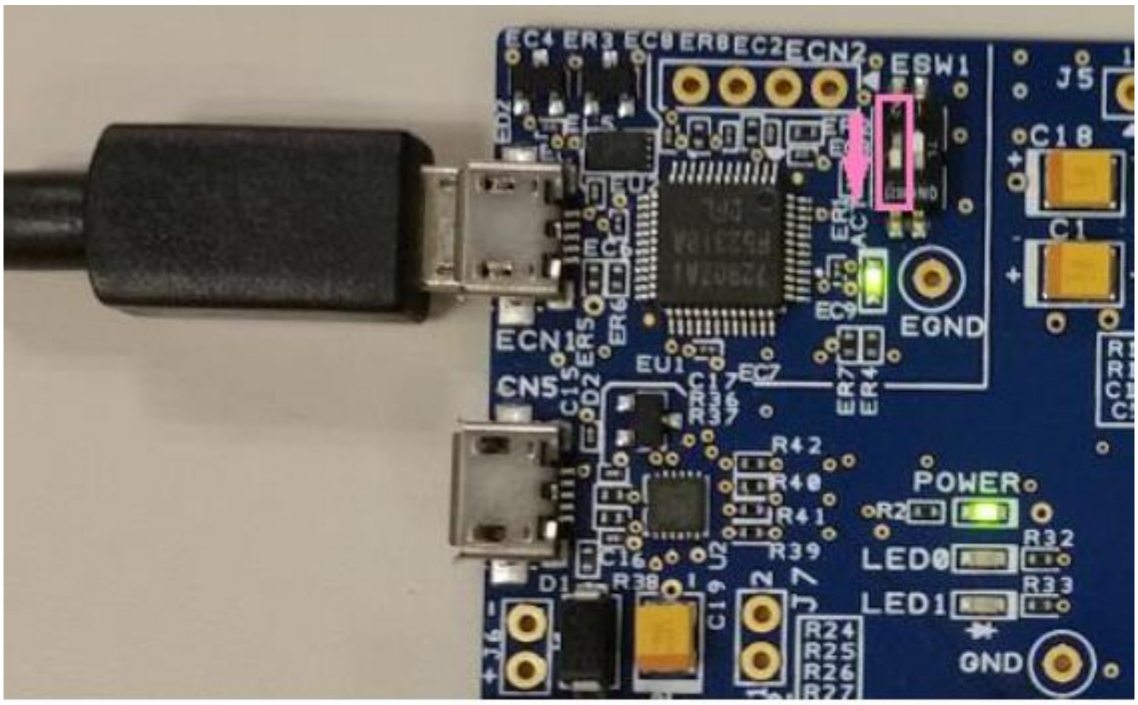


Figure 5.1 Write programs

2. Start RFP and select “File” → “New Project…”.

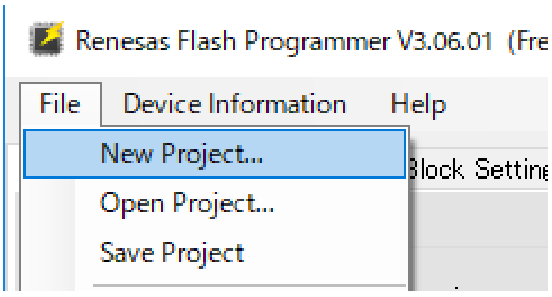
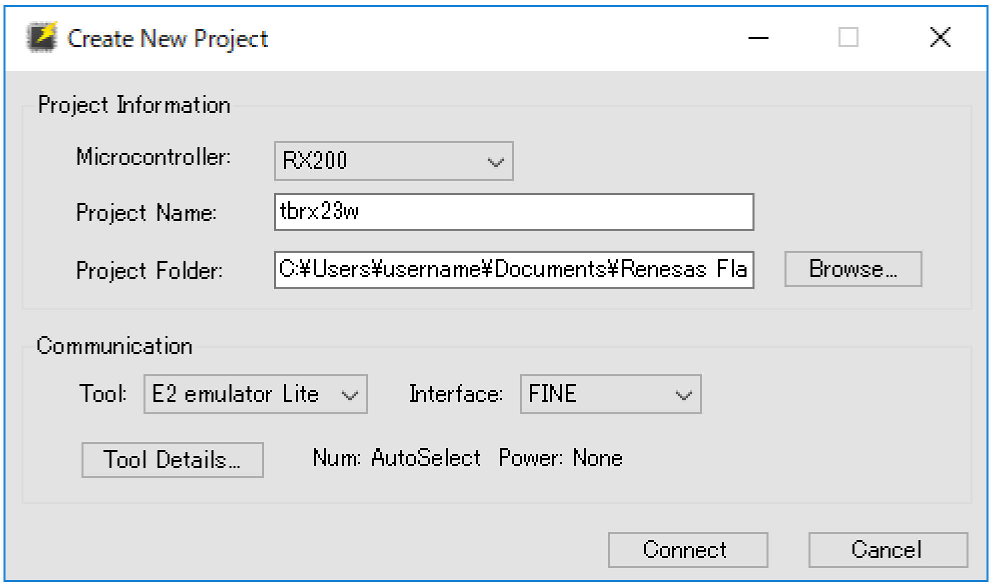


Figure 5.2 Create New Project

3. In the “Create New Project” window, make the following settings and click the “Connect” button.
   - Microcontroller: RX200
   - Project Name: Any name
   - Project Folder: Any folder
   - Communication Tool: E2 emulator Lite
   - Communication Interface: FINE
   - Power: None (default)


Create New Project

Project Information

Microcontroller: RX200

Project Name: tbrx23w

Project Folder: C:¥Users¥username¥Documents¥Renesas Fla

Communication

Tool: E2 emulator Lite

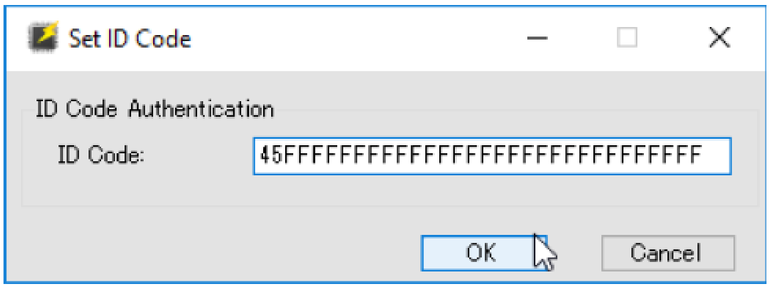
Interface: FINE

Num: AutoSelect Power: None

Figure 5.3 Project Setting

4. When prompted to enter the “Set ID Code” and click the “OK” button.

“45FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF” are setting in initialized Target Board.

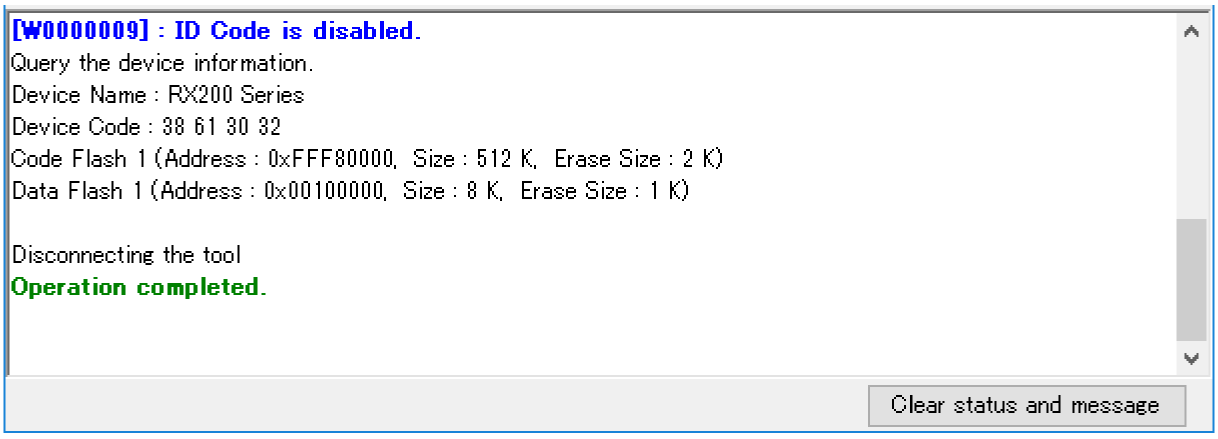

Set ID Code

ID Code Authentication

ID Code: 45FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Figure 5.4 Set ID Code

5. If the connection is successful, "Operation completed." will be displayed.

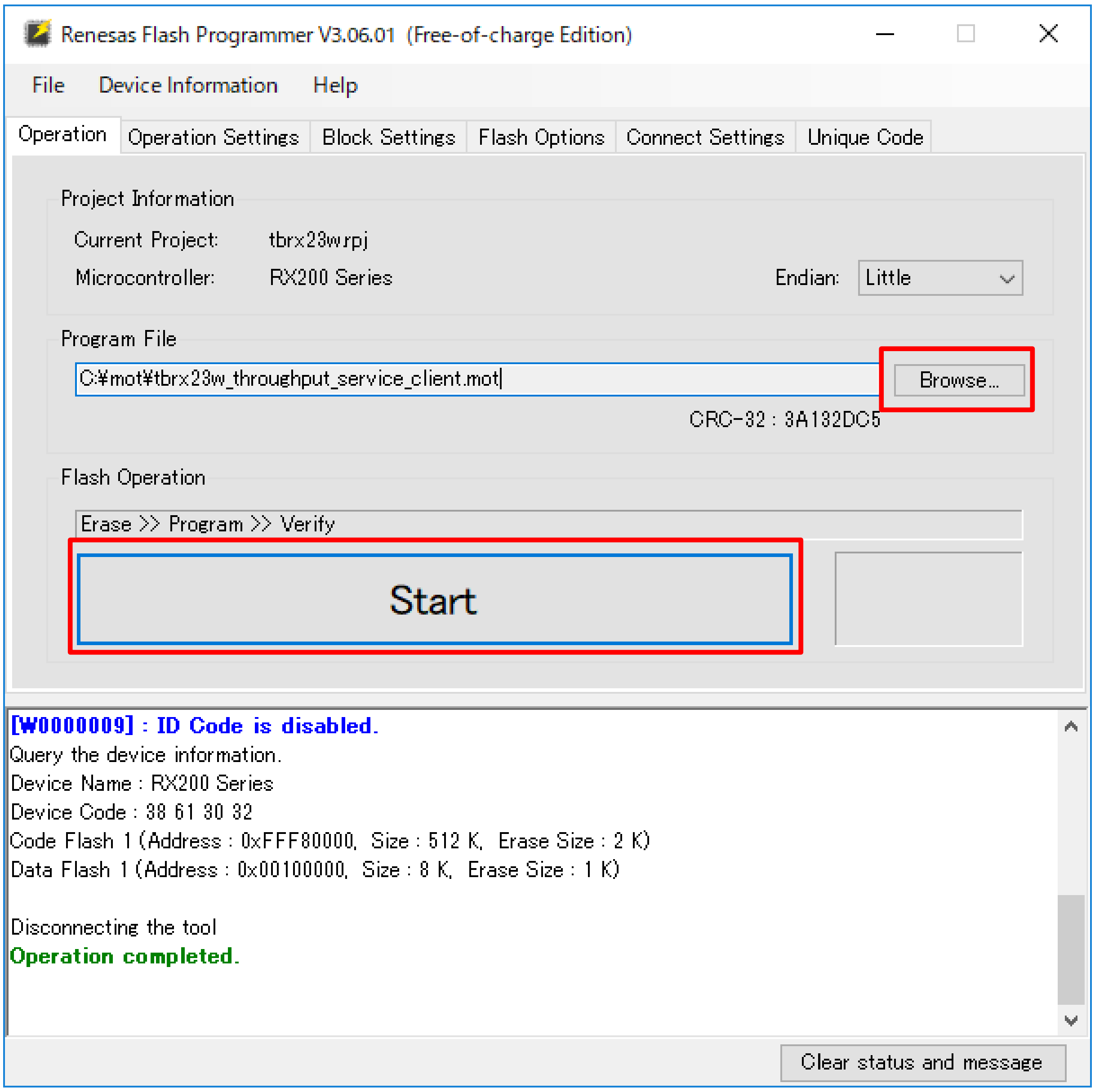


```
[W0000009] : ID Code is disabled.
Query the device information.
Device Name : RX200 Series
Device Code : 38 61 30 32
Code Flash 1 (Address : 0xFFF80000, Size : 512 K, Erase Size : 2 K)
Data Flash 1 (Address : 0x00100000, Size : 8 K, Erase Size : 1 K)

Disconnecting the tool
Operation completed.
```

Figure 5.5 Successful Connection

6. Click the “Browse…” button, select “rx23w_throughput_servce_xxxxxx.mot” in the mot folder in this application note, and click the “Open” button. Once you have selected the correct file, click the “Start”

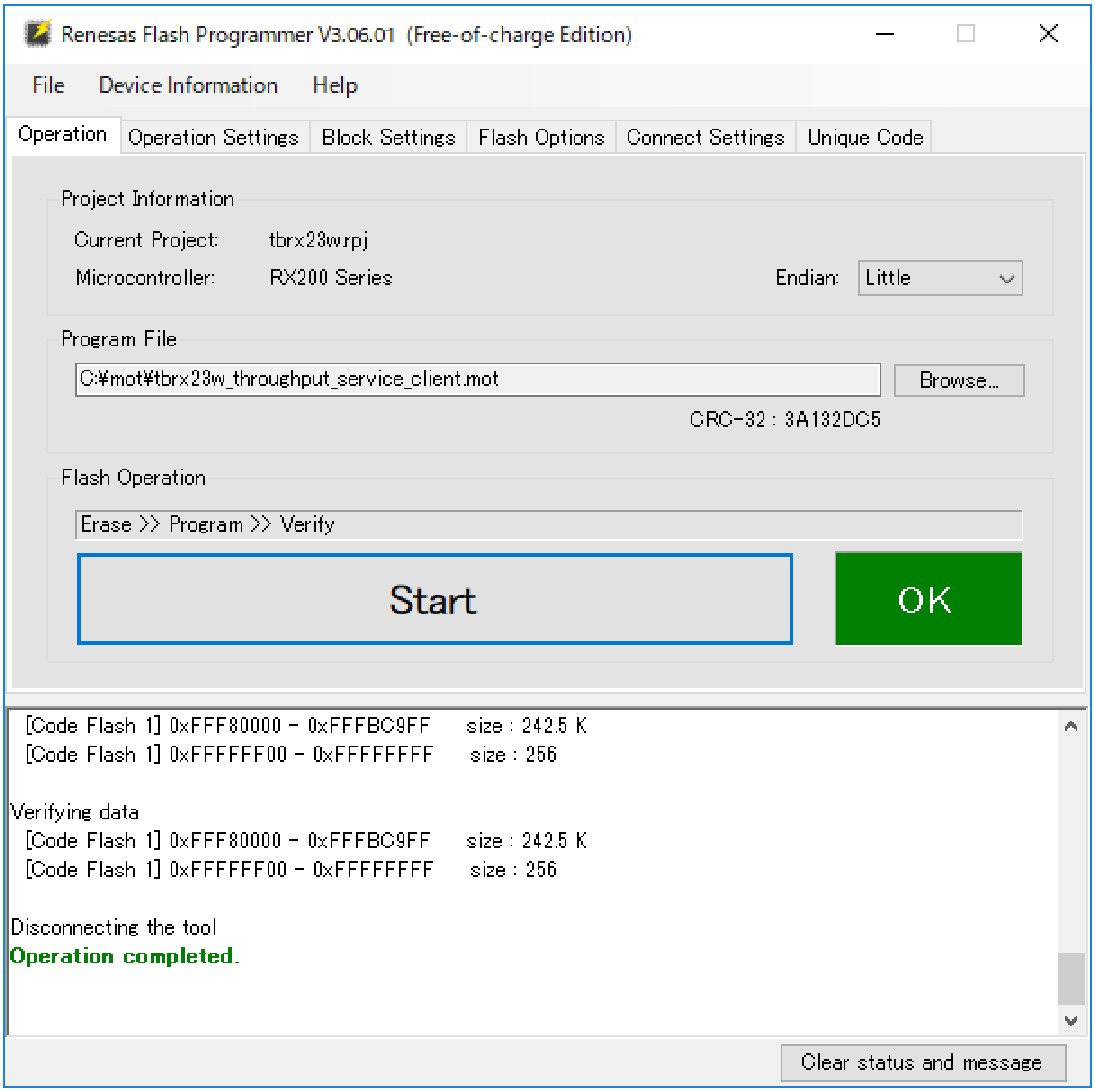


```
[W0000009] : ID Code is disabled.
Query the device information.
Device Name : RX200 Series
Device Code : 38 61 30 32
Code Flash 1 (Address : 0xFFF80000, Size : 512 K, Erase Size : 2 K)
Data Flash 1 (Address : 0x00100000, Size : 8 K, Erase Size : 1 K)

Disconnecting the tool
Operation completed.
```

Figure 5.6 Programming Firmware

7. When programming is completed normally, “Operation completed.” and “OK” are displayed.



Current Project: tbrx23w.rpj
Microcontroller: RX200 Series
Endian: Little
C:¥mot¥tbrx23w_throughput_service_client.mot
CRC-32 : 3A132DC5
Erase >> Program >> Verify

```
  [Code Flash 1] 0xFFF80000 - 0xFFFBC9FF    size : 242.5 K
  [Code Flash 1] 0xFFFFFF00 - 0xFFFFFFFF    size : 256

Verifying data
  [Code Flash 1] 0xFFF80000 - 0xFFFBC9FF    size : 242.5 K
  [Code Flash 1] 0xFFFFFF00 - 0xFFFFFFFF    size : 256

Disconnecting the tool
Operation completed.
```

Figure 5.7 Programming Completion

5.2 How to add project to e^{2}studio workspace

This chapter explain how to add sample program project to e^{2}studio workspace.

1. Select "File" → "Import".

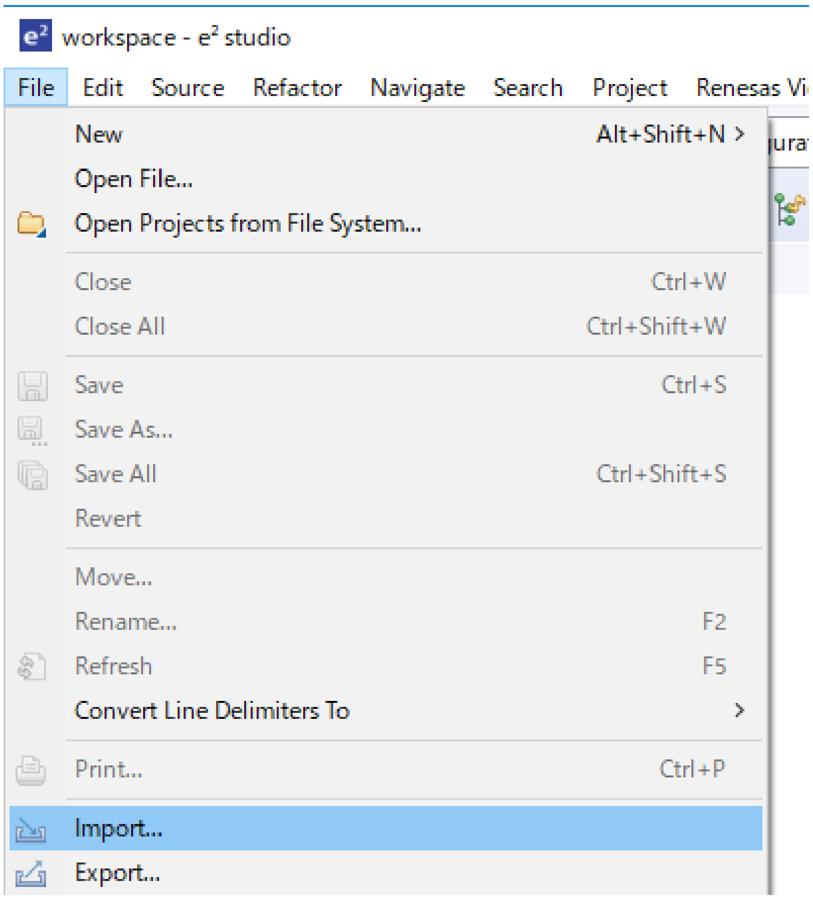


Figure 5.8 File menu

2. Select “Existing Projects into Workspace” and click “Next” button.

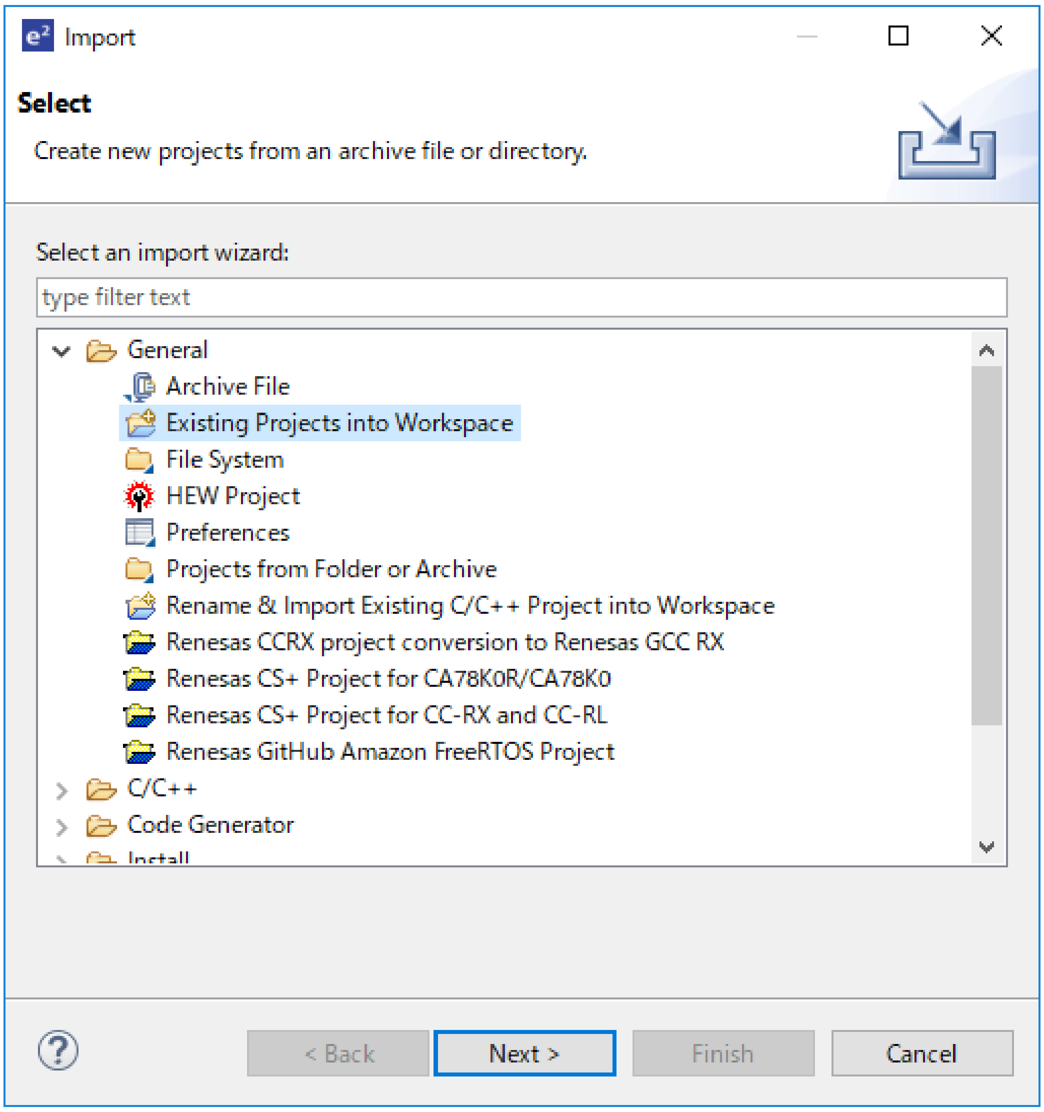
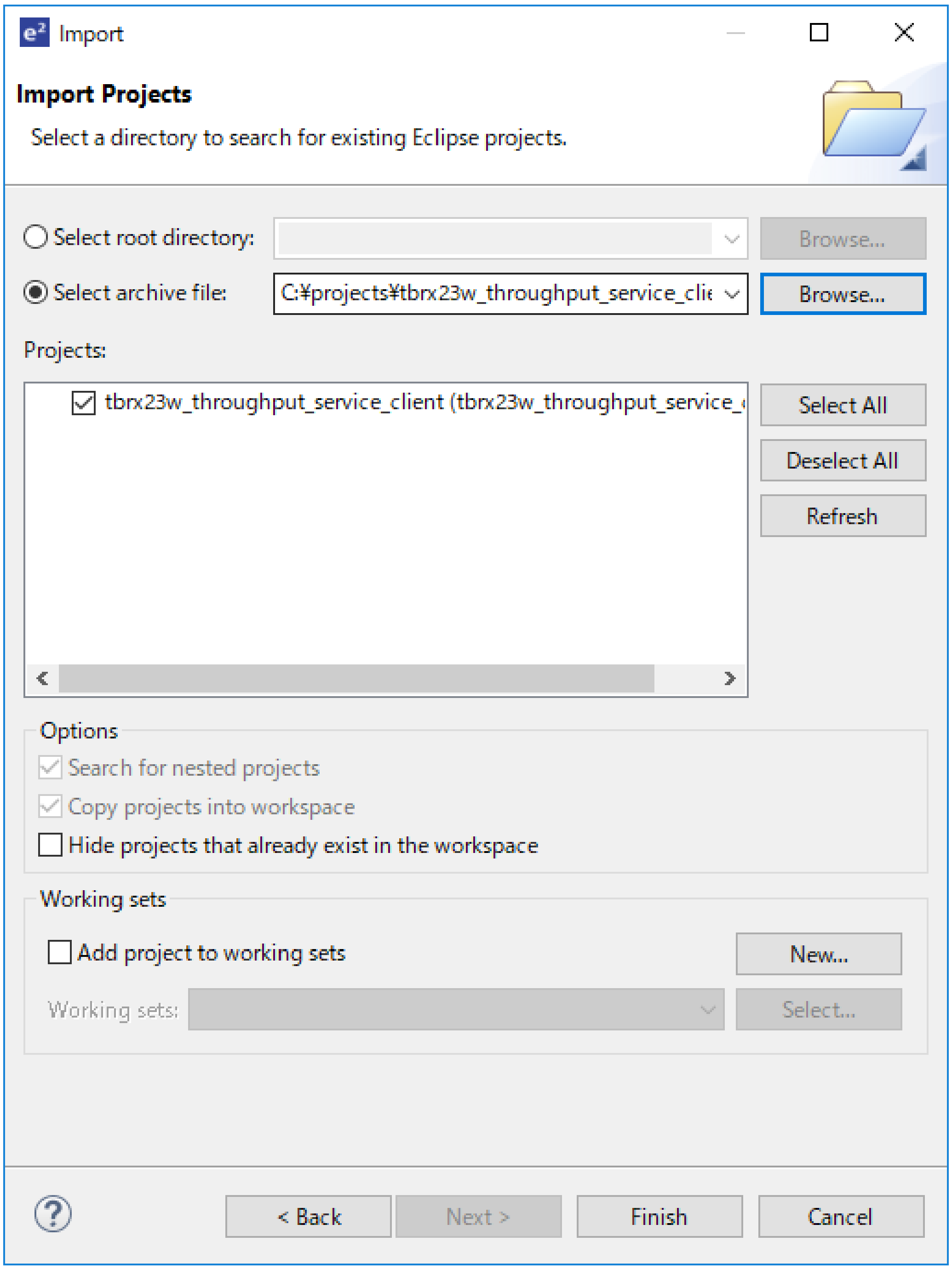

Import
Select
Create new projects from an archive file or directory.
Select an import wizard:
type filter text
General
Archive File
Existing Projects into Workspace
File System
HEW Project
Preferences
Projects from Folder or Archive
Rename & Import Existing C/C++ Project into Workspace
Renesas CCRX project conversion to Renesas GCC RX
Renesas CS+ Project for CA78K0R/CA78K0
Renesas CS+ Project for CC-RX and CC-RL
Renesas GitHub Amazon FreeRTOS Project
C/C++
Code Generator
Install
< Back
Next >
Finish
Cancel

Figure 5.9 Select an import wizard

3. Select “Select archive file”, click “Browse…” button and select the sample program project archive file. Click “Finish” button and the sample program project is imported.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

○ Select root directory:

● Select archive file: C:¥projects¥tbrx23w_throughput_service_clie

Projects:

☑ tbrx23w_throughput_service_client (tbrx23w_throughput_service_

Options

☑ Search for nested projects

☑ Copy projects into workspace

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

Working sets:

Figure 5.10 Import Projects

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	May.14.2020	—	First edition issued.

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

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

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Corporate Headquarters

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

Trademarks

Contact information

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