

RL78/G22

LTE MQTT Communication

Introduction

This application note explains how to perform LTE communication using RL78/G22 and RYZ024A. RYZ024A is a cellular module capable of LTE Cat M1/NB1/NB2 communication. RYZ024A is a module capable of LTE Cat M1/NB1/NB2 communication, connected with host MCU via UART communication and controlled by AT commands. RL78/G22 works as host MCU of RYZ024A. The provided sample application utilizes RL78/G22 as a host MCU and provides a program for controlling RYZ024A to perform MQTT communication and implement power-saving features of the cellular network (eDRX, PSM). And, this sample application uses the AT command management framework to implement a program that sends AT commands from the RL78/G22 to the RYZ024A. By using the AT command management framework, it is possible to implement applications that use various communication protocols supported by the RYZ024A's LTE Cat M1 communication function. This document describes the MQTT communication application and AT command management framework implemented in this sample application.

SIM card activation is required when using the Truphone SIM card included in the PMOD Expansion Board for RYZ024A (hereafter PMOD-RYZ024A) ([RTKYZ024A0B00000BE](#)). Refer to "RA6M5 Group RYZ024A PMOD LTE Connectivity with RA6M5 MCU Quick Start Guide" (R21QS0007) to activate the SIM card.

Target Device

RL78/G22

RYZ024A

Related Documents

- RL78/G22 User's Manual: Hardware (R01UH0978)
- RL78/G22 Fast Prototyping Board User's Manual (R20UT5121)
- RL78/G22 Fast Prototyping Board Quick Start Guide (R20UT5123)
- RYZ024 Use Case for AT Commands(R19AN103)
- RYZ024 Modules AT Command User's Manual(R11UZ0110)
- RYZ024 Module System Integration Guide (R19AN0101)
- RA6M5 Group RYZ024A PMOD LTE Connectivity with RA6M5 MCU Quick Start Guide (R21QS0007)

Pmod™ is registered to Digilent Inc.

Contents

1. Overview.....	4
1.1 Operation overview	4
1.2 Description of the Software	5
1.2.1 Sample program Configuration	5
1.2.2 Hierarchy of sample programs	5
1.2.3 Used peripheral function	6
1.2.4 List of Option Byte Settings	6
1.2.5 Folder/File structure	7
1.2.6 Code size.....	7
2. MQTT Communication Application	8
2.1 Application environment	8
2.2 Application operation	16
2.2.1 Operation in 1 set	16
2.2.2 Operation in 2 sets	18
3. AT Command Management Framework	21
3.1 Framework Overview	21
3.2 API functions	23
3.2.1 Management API.....	23
3.2.1.1 R_LTE_Init	24
3.2.1.2 R_LTE_Execute	24
3.2.2 AT command API	25
3.2.2.1 R_LTE_OM_Config	26
3.2.2.2 R_LTE_NWK_Connect	27
3.2.2.3 R_LTE_NWK_Disconnect.....	27
3.2.2.4 R_LTE_MQTT_Connect.....	28
3.2.2.5 R_LTE_MQTT_Subscribe.....	28
3.2.2.6 R_LTE_MQTT_Publish	29
3.2.2.7 R_LTE_MQTT_RcvMessage	30
3.2.2.8 R_LTE_MQTT_Disconnect.....	30
3.2.2.9 R_LTE_SEC_CertificateAdd	31
3.2.2.10 R_LTE_SEC_CertificateRemove	31
3.2.2.11 R_LTE_SEC_PrivateKeyAdd	32
3.2.2.12 R_LTE_SEC_PrivateKeyRemove	32
3.2.2.13 R_LTE_NWK_ConnectionConfig	33
3.2.2.14 R_LTE_eDRX_Config	34
3.2.2.15 R_LTE_PSM_Config	36
3.3 Callback function	38
3.4 User Specific Configuration	44

3.5	Smart Configurator module used in the framework.....	47
3.5.1	UARTA module	47
3.5.2	TAU module.....	48
3.5.3	Interrupt function	48
3.5.4	UART0 module.....	48
3.6	Low Power Operation.....	49
3.6.1	Low power operation control of RL78/G22.....	49
3.6.2	Low power operation control of RYZ024A	50
4.	Application development using AT Command Management Framework.....	52
4.1	Overview of application development	52
4.2	Adding an AT command API.....	54
4.2.1	AT command API with Data receive operation	58
4.3	Guideline of error handling	59
4.4	PMOD-RYZ024A specific processing	62
4.5	Initializing PMOD-RYZ024A	64
	Revision History	65

1. Overview

1.1 Operation overview

The RYZ024A is a cellular module with Cat M1 technology support. This function can be controlled by AT commands via the UART from RL78/G22.

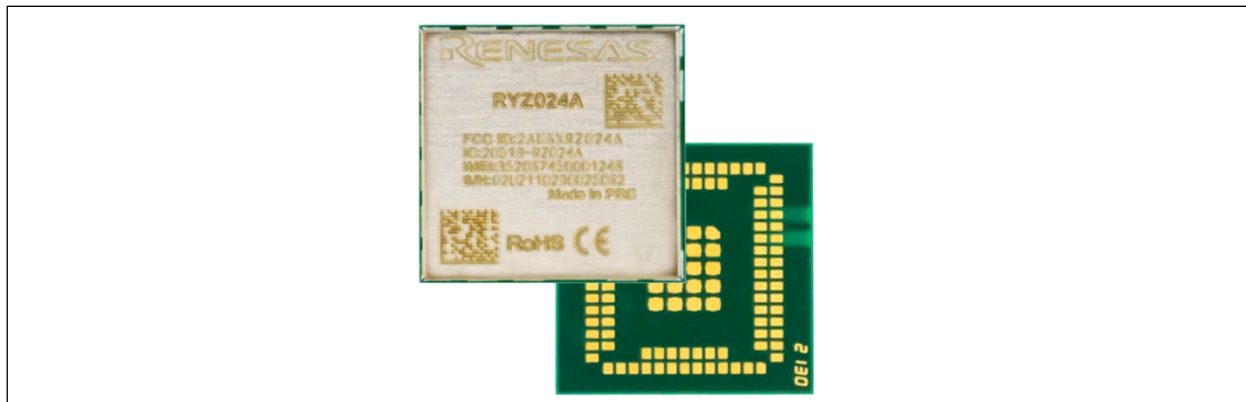


Figure 1-1 RYZ024A

In this sample application, AT command framework software, which controls LTE Cat M1 communication of RYZ024A using the RL78/G22 as the host MCU. The RL78/G22 sends AT command as string data to RYZ024A via UART communication. The response string data for the AT command is also received by UART communication. Through these exchanges, the RL78/G22 utilizes the LTE Cat M1 communication function of the RYZ024A.

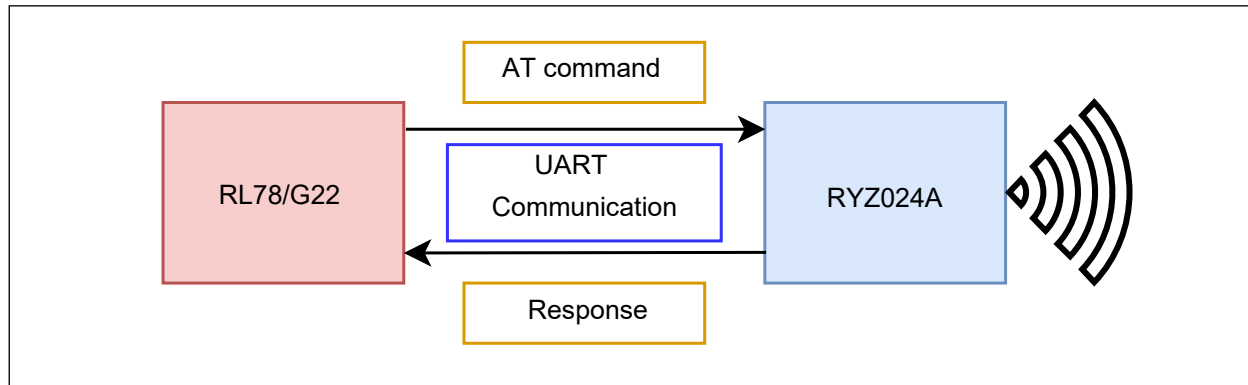


Figure 1-2 Communication between RL78/G22 and RYZ024A

Note: Regarding the use of PMOD-RYZ024A :

When the RYZ024A shifts to deep sleep, the CTS pin becomes Hi-Z. However, in the PMOD-RYZ024A, the CTS signal from the level shifter to the host microcomputer remains at low level due to the characteristics of the level shifter used. (The RXD pin also remains Low.) Therefore, be careful when waking up the RYZ024A from deep sleep and transmitting from the microcomputer to the UART.

In this sample application, dedicated processing is added to operate with PMOD-RYZ024A. For details, refer to [4.4 PMOD-RYZ024A specific processing].

1.2 Description of the Software

This sample application provides a sample program for checking the operation of Publish/Subscribe. This sample program can check the Publish/Subscribe operation with only 1 set (2.2.1 Operation in 1 set), or you can use two sets (2.2.2 Operation in 2 sets) and check the Publish/Subscribe operation with each other.

1.2.1 Sample program Configuration

[Table 1-1 Contents of this Application Note] shows the configuration of this sample program.

Table 1-1 Contents of this Application Note

File Name	Description
r01an6951xxrrrr--lte-sample.pdf xx: Language classification, Creation country rrrr: Revision number	This Document
sample_rl78g22_ryz024a	Publish/Subscribe sample application program

1.2.2 Hierarchy of sample programs

Software configuration of this sample framework is shown below.

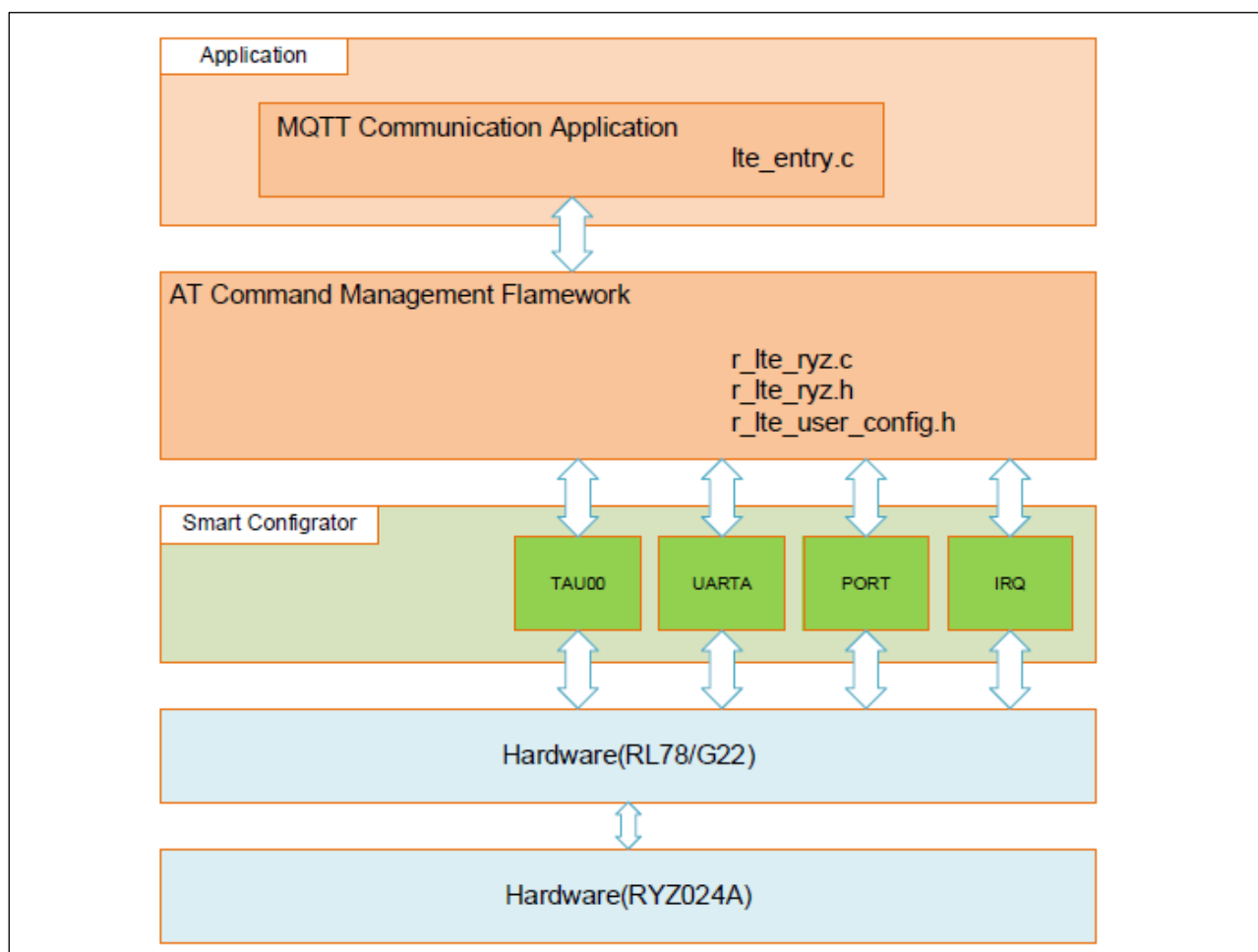


Figure 1-3 Software Configuration

The sample application is a program that performs MQTT communication using the RYZ024A module's TCP/IP stack. The application is implemented on the host MCU side. This program consists of two features, the MQTT communication application, which calls APIs for the MQTT communication and the AT command framework, which sends the AT command to the RYZ024A.

The MQTT application program connects to the MQTT server and sends data after user switch button on the RL78/G22 FPB is pressed. MQTT communication application is implemented using APIs from AT Command Management Framework. For a detailed description about the MQTT communication application, refer to [2 MQTT Communication Application].

The AT Command Management Framework is a framework for implementing the transmission of AT commands to the RYZ024A and the processing of responses received from the RYZ024A. By calling the API function implemented on the AT Command Management Framework, multiple AT command are sent to the RYZ024A, and application is notified by a callback function.

In this sample application, a framework-based program is implemented using a framework so that the RL78/G22 can perform MQTT communication through the RYZ024A. The AT Command Management Framework is intended to be used as a base for application development when using functions of the RYZ024A other than MQTT communication. For a detailed description about AT Command Management Framework, refer to [3 AT Command Management Framework].

1.2.3 Used peripheral function

[Table 1-2 Peripheral Functions to be used and their purpose] shows the peripheral functions used in this sample program.

Table 1-2 Peripheral Functions to be used and their purpose

Peripheral Function	Purpose
TAU00	AT command communication timeout
UARTA(P72/TxDA0)	UART TX
UARTA(P71/RxDA0)	UART RX
PORT(P70)	RTS signal for UART
PORT(P50)	CTS signal for UART
PORT(P17)	RYZ024A RESET control
SW(P137/INTP0)	User switch external pin interrupt
INTP(P51/INTP2)	RYZ024A RING signal external pin interrupt
UART(P12/TxD0)	UART TX for monitor log output

1.2.4 List of Option Byte Settings

[Table 1-3 Option Byte Settings] shows the option byte settings.

Table 1-3 Option Byte Settings

Address	Setting Value	Contents
000C0H/020C0H	11101111B	Watchdog time counter operation disable (Counting stopped after reset)
000C1H/020C1H	11111100B	LVD0 detection voltage: Reset mode At rising edge TYP. 2.67 V (2.59 V to 2.75 V) At falling edge TYP. 2.62 V (2.54 V to 2.70 V)
000C2H/020C2H	11101000B	HS mode, High-speed on-chip oscillator clock (f_{IH}): 32 MHz
000C3H/020C3H	10000100B	Enables on-chip debugging

1.2.5 Folder/File structure

[Table 1-4 Sample Program file structure] shows the file structure of the sample program provided in this application note.

Table 1-4 Sample Program file structure

Folder name, File name	Description
r01an6951_rl78g22_itemqtt	Program storage folder
└ src	Source files
├ └ ryz	RYZ024A related source files
├ └ smc_gen	Smart configurator generation
├ └ └ Config_INTC	
├ └ └ Config_PORT	
├ └ └ Config_TAU0_1	
├ └ └ Config_UART0	
├ └ └ Config_UARTA0	
├ └ └ general	
├ └ └ r_bsp	
├ └ └ r_config	
├ └ └ r_pincfg	
├ └ lte_entry.c	Main program of the sample application
├ └ lte_entry.h	
├ └ main.c	
├ └ main.h	

1.2.6 Code size

[Table 1-4 ROM/RAM Sizes] shows ROM/RAM Sizes of the sample program provided in this application note.

Table 1-5 ROM/RAM Sizes

Resource	Size
ROM	23,700 Byte
RAM	2,394 Byte

2. MQTT Communication Application

2.1 Application environment

This section describes the environment to operate the MQTT communication application.
This application operates in the following hardware environment.

Table 2-1 Hardware environment

Hardware	Description
RL78/G22 Fast Prototyping Board (RL78/G22 FPB)	Evaluation board with RL78/G22 (RTK7RLG220C00000BJ)
PMOD Expansion Board for RYZ024A (PMOD-RYZ024A)	PMOD board with RYZ024A module (RTKYZ024A0B00000BE)
Windows PC	RL78/G22's application development environment and debug console for operation conformation

Note: Regarding the use of PMOD-RYZ024A:

When the RYZ024A shifts to deep sleep, the CTS pin becomes Hi-Z. However, in the PMOD-RYZ024A, the CTS signal from the level shifter to the host microcomputer remains at low level due to the characteristics of the level shifter used. (The RXD pin also remains Low.) Therefore, be careful when waking up the RYZ024A from deep sleep and transmitting from the microcomputer to the UART.

In this sample application, dedicated processing is added to operate with PMOD-RYZ024A. For details, refer to [4.4 PMOD-RYZ024A specific processing].

This sample application has been developed and checked with the following software environment.

Table 2-2 Software environment of sample application

Item	Description
Integrated development environment (e2 studio)	Made by Renesas Electronics Corporation e2 studio V2024-01 (24.1.0)
C compiler (e2 studio)	Made by Renesas Electronics Corporation CC-RL V1.13.00
Integrated development environment (CS+)	Made by Renesas Electronics Corporation CS+ for CC V8.11
C compiler (CS+)	Made by Renesas Electronics Corporation CC-RL V1.13.00
Smart Configurator	Made by Renesas Electronics Corporation V1.9.0
Board support package (BSP)	Made by Renesas Electronics Corporation V1.62
Renesas Flash Programmer (RFP)	Made by Renesas Electronics Corporation V3.14.00

For application operation, follow these steps:

1. Connect RL78/G22 FPB and PMOD-RYZ024A with PMOD connector.
Please use PMOD2 connector for RL78/G22 FPB.

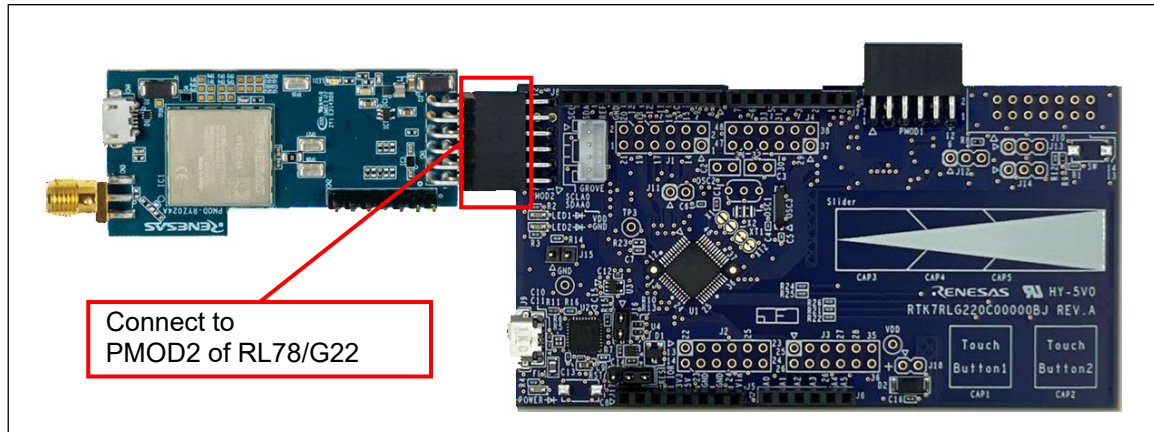


Figure 2-1 Connect RL78/G22 FPB and RYZ024A

2. Connect USB cables to RL78/G22 FPB and PMOD-RYZ024A.

Also connect the antenna to PMOD-RYZ024A.

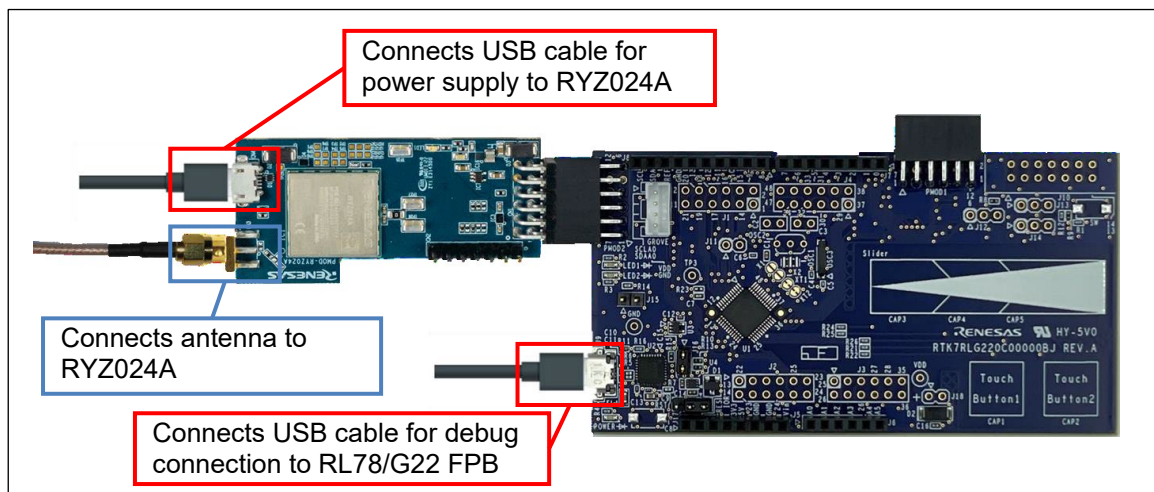


Figure 2-2 Connect USB cable and antenna

3. Import sample project

The sample program is provided in the project format of e² studio. This chapter shows how to import the project to e² studio and CS+.

- Importing a Project into e² studio

To use sample programs in e² studio, follow the steps below to import them into e² studio. In projects managed by e² studio, do not use space codes, multibyte characters, and symbols such as "\$", "#", "%" in folder names or paths to them.

(Note that depending on the version of e² studio you are using, the interface may appear somewhat different from the screenshots below.)

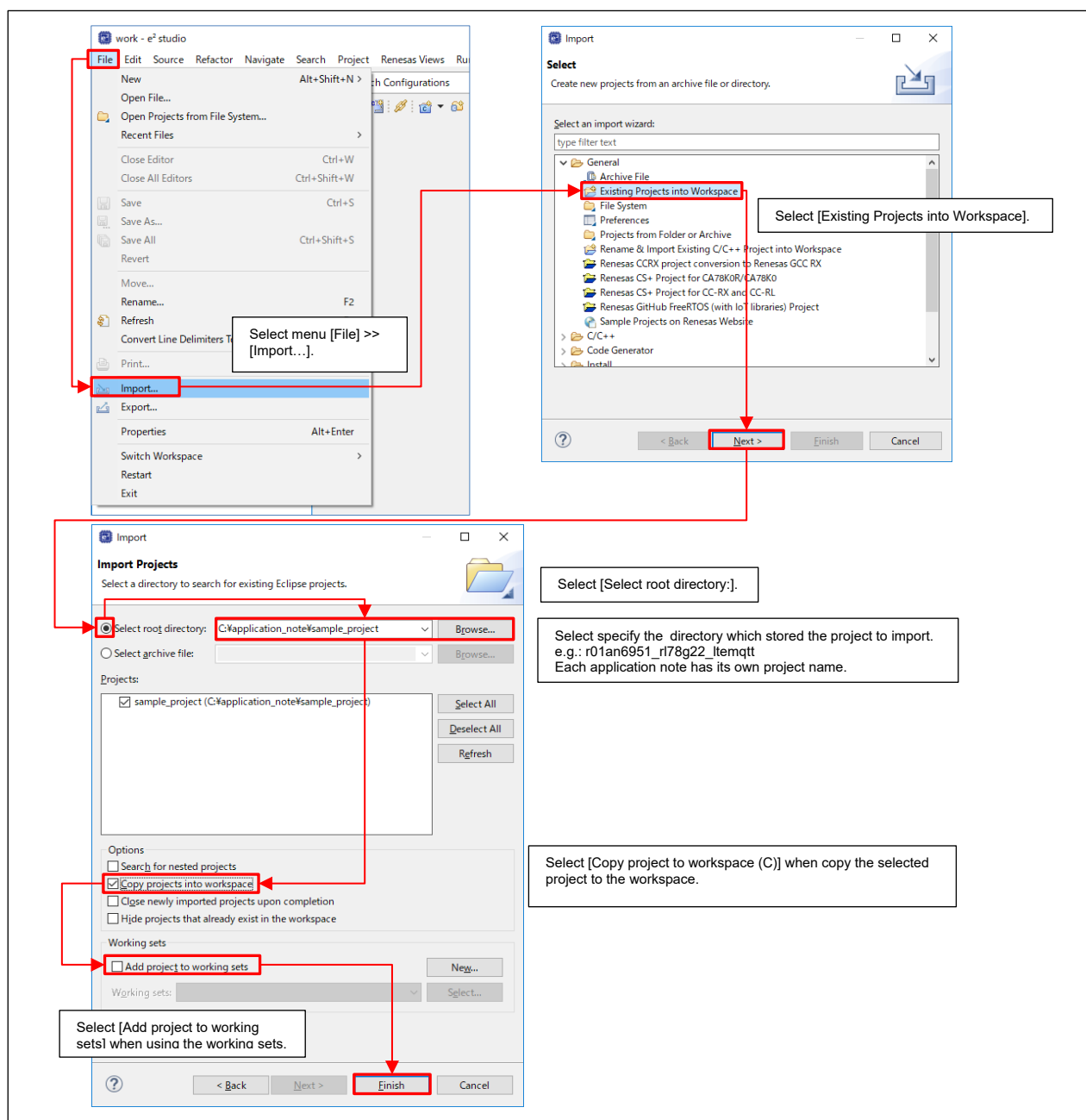


Figure 2-3 Importing a Project into e² studio

- Importing a Project into CS+

To use sample programs in CS+, follow the steps below to import them into CS+. In projects managed by CS+, do not use space codes, multibyte characters, and symbols such as "\$", "#", "%" in folder names or paths to them.

(Note that depending on the version of CS+ you are using, the interface may appear somewhat different from the screenshots below.)

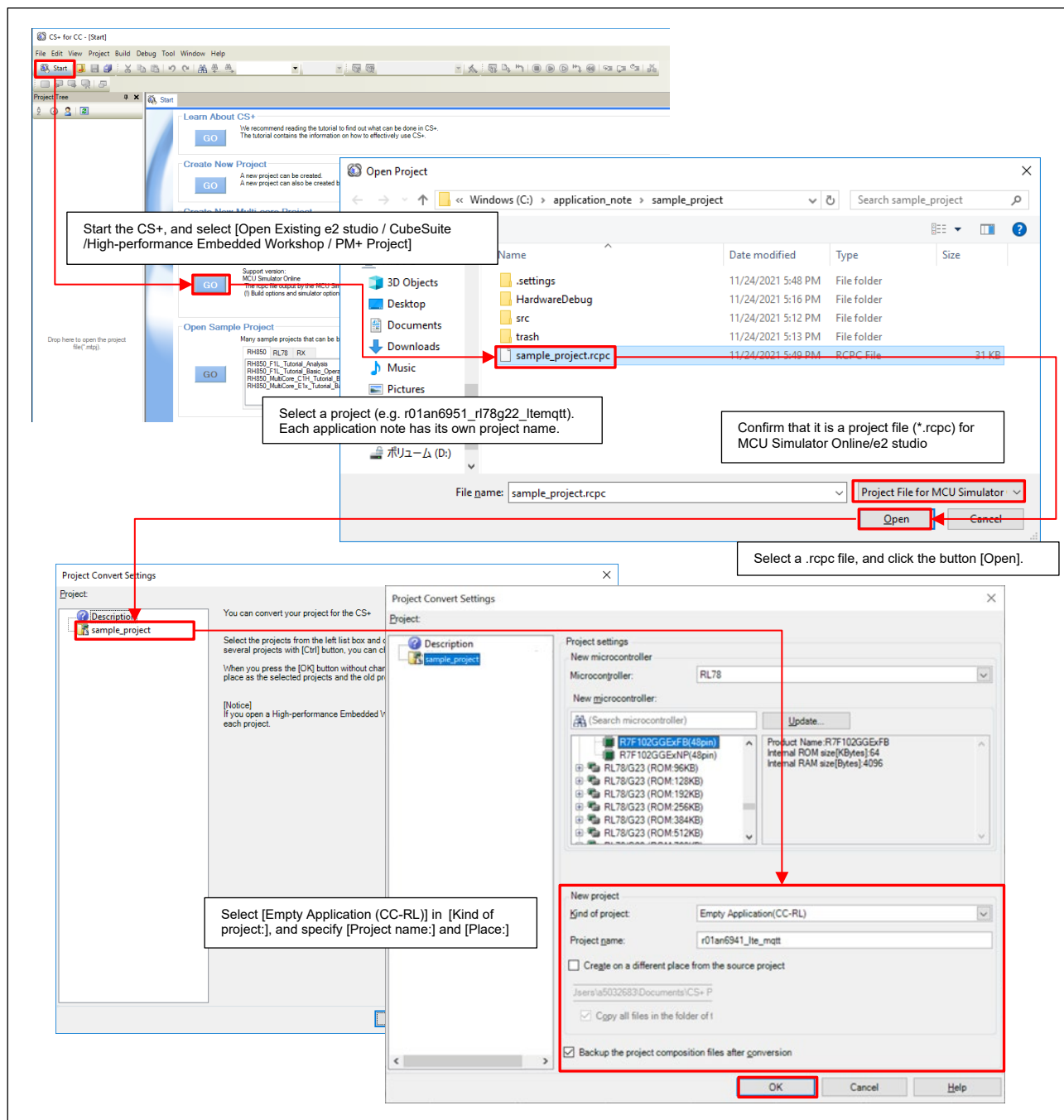


Figure 2-4 Importing a Project into CS+

4. Change Access Point Name, Authentication protocol, Username, Password, and the LTE bands by modifying the string data to match those of the user application.
The changes are needed in the below files:
 - lte_entry.c

Access Point Name (APN), Authentication protocol, Username, Password

The character string data to be changed are basically dependent on the SIM used. Please contact the SIM provider for APNs that can be used for user SIM. Username and password may be omitted. Please refer to the manual document of each kit for information about the SIM included with the kit provided from Renesas such as how to activate the SIM and available APNs.

LTE Bands

LTE bands differ depending on used region or operator. If you know the LTE bands to be used, specify that bands. The following is an example of the LTE bands:

- "1,19"
 - When user specifying DOCOMO bands.
- "2,4,12"
 - When user specifying AT&T bands.
- "1,2,3,4,5,8,12,13,17,18,19,20,25,26,28,66"
 - When user can't specify the bands

Note: The RYZ024A may take several minutes to connect when activating the SIM or connecting to the network for the first time. Restricting the band to be used is effective in shortening the time to connect.

In this application, operation is confirmed using the following APN and LTE bands.

- APN : ppsim.jp
- Authentication protocol : 0
- Username : -
- Password : -
- LTE bands : 1,19

```

36  /* ***** */
37  * @addtogroup r01an6951_rl78g22_ltemqtt
38  * @
39  /* ***** */
40
41  /* Application value definition */
42  extern uint8_t g_rbuf_guard[10];
43
44  uint8_t g_sw1_push_flag = 0;
45  uint8_t g_sw2_push_flag = 0;
46
47  static uint8_t s_sbuf[100];
48  static uint8_t s_sw1_count = 0;
49  static uint8_t s_reinitialize_flag = 0;
50  static uint8_t s_mqtt_conn_state = 0;
51
52  /* Application specific string data for network connection */
53  static uint8_t s_str_pdp_type[] = "IPv4V6";
54
55  /* Access Point Name using in network connection. Please select depending on SIM card */
56  static uint8_t s_str_pdp_apn[] = "ppsim.jp"; // ex. globaldata.iot , ibasis.iot , soracom.io
57
58  /* Authentication protocol, user id, password. Please modify depending on using LTE band */
59  static uint8_t s_str_pdp_protocol[] = "2";
60  static uint8_t s_str_pdp_userid[] = "pp@sim";
61  static uint8_t s_str_pdp_password[] = "jpn";
62
63  /* Band List for network connection. Please select depending on using LTE band
64  static uint8_t s_str_lte_bandlist[] = "1,19"; // Docomo bands in JP Region
65
66  /* static uint8_t s_str_lte_bandlist[] = "2,4,12"; AT&T bands in US Region */
67  /* static uint8_t s_str_lte_bandlist[] = "1,2,3,4,5,8,12,13,17,18,19,20,25,26,28,66"; All selectable bands. */

```

Change APN, protocol, username, and password

Change LTE Bands

Figure 2-5 Change APN, Authentication protocol, Username, Password and LTE bands (lte_entry.c)

- Disable low power operation of the RL78/G22 FPB. When using the dynamic printf that monitors the operation of this sample program, comment out the API (ryz_lte_lowpower_devspe) that executes the low power consumption mode because it is not recommended to use the low power consumption mode of the RL78/G22 FPB.
Comment out the line calling ryz_lte_lowpower_devspe() in the function shown below.

File name/function name: r_lte_ryz.c / void R_LTE_Execute(void)

```

1237     }
1238     ryz_lte_lowpower_devspe(lpm_mode);
1239 }

```

Figure 2-6 Disable low power operation

6. Set Motorola S-record file output.

(1): Right mouse click on the project.

(2): Select property.

(3): Select Settings → Converter Output, check "Output hex file", confirm that Output Motorola S-record file is selected, click Apply and Close.

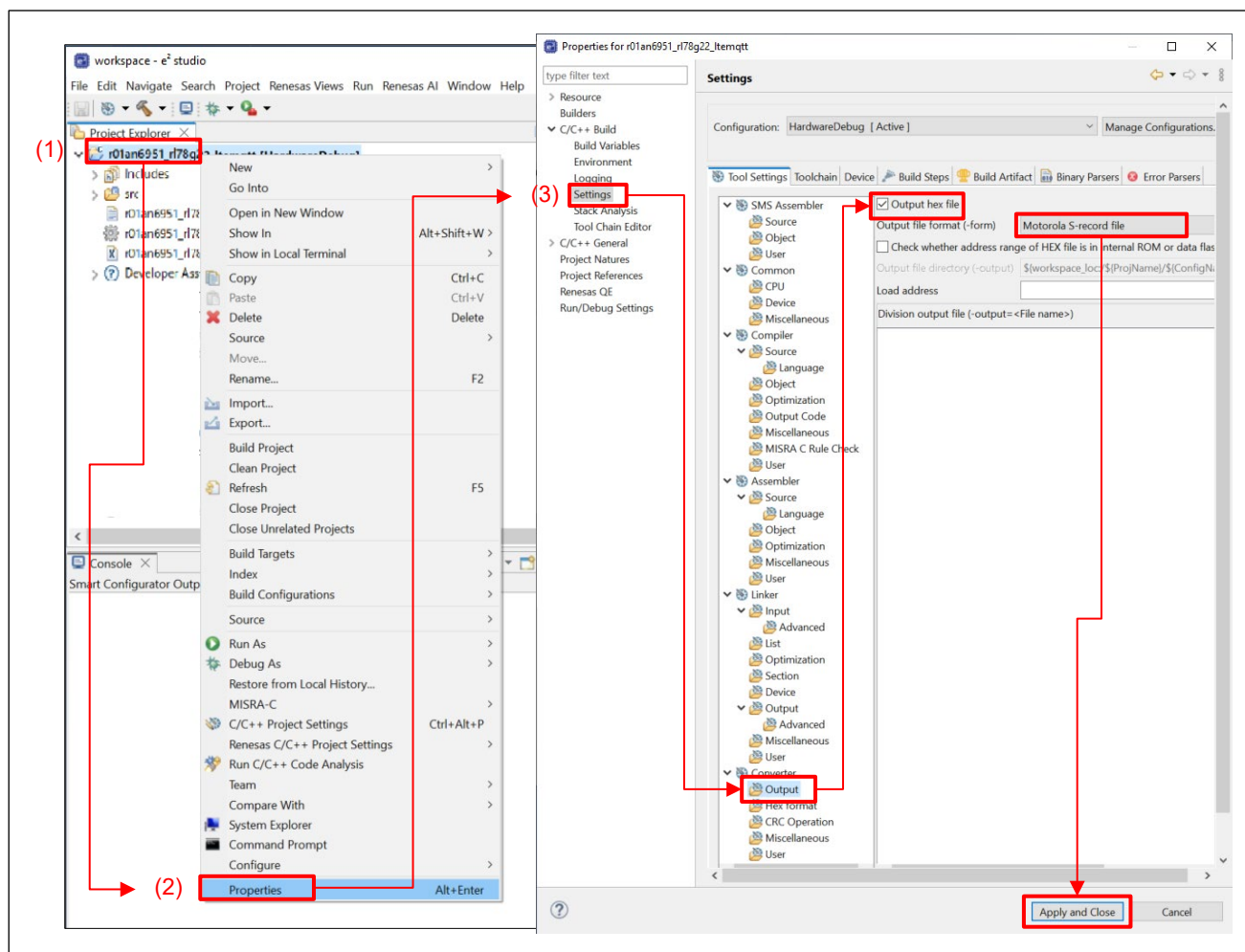


Figure 2-7 Motorola S-record file output setting

7. Build the sample project. When you build, Motorola S-record file (.mot) will be output.

8. Program the Motorola S-record file to the RL78/G22 FPB using the Renesas Flash Programmer.

- (1): Select File → New Project...
- (2): Select RL78/G2x from the pull-down menu.
- (3): Specify an arbitrary name and arbitrary folder for the Project Name and Project folder.
- (4): Select COM port and 2 wire UART from the pull-down menu.
- (5): Click Tool Details...
- (6): Specify COM port of RL78/G22 FPB
- (7): Click OK
- (8): Click Connect (If the connection is successful, proceed to (9). In case of error, please check the settings before (6).
- (9): Specify Motorola S-record file generated by build.
- (10): Click Start to start programing.

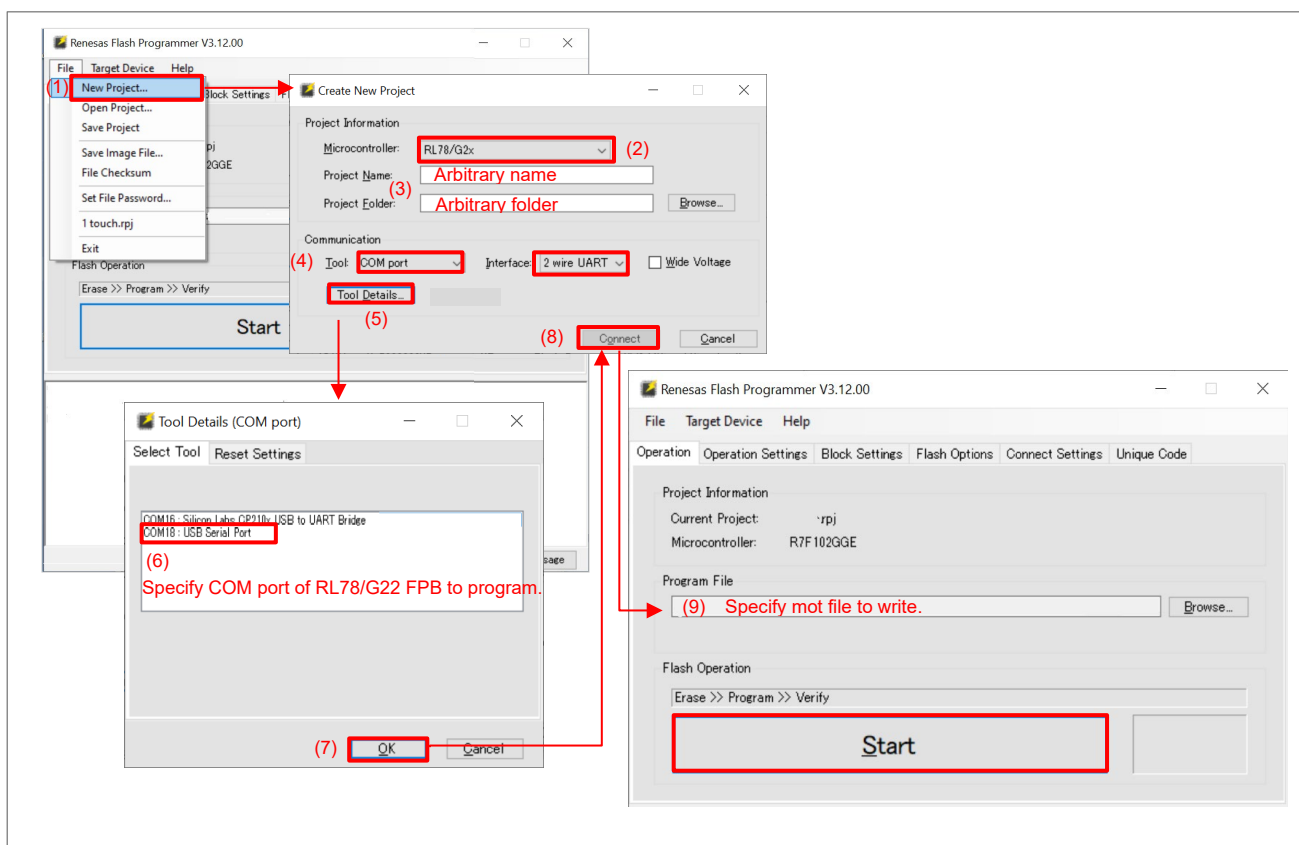


Figure 2-8 Programming file

9. Prepare for monitoring.

The execution result of the sample program is output from USB of RL78/G22 FPB. Specify the COM port of RL78/G22 FPB with terminal software such as TeraTerm.

Baud rate is 115200bps, data is 8bit, parity is none, stop bit is 1bit. The line feed code reception setting is "LF".

2.2 Application operation

This sample program uses the public MQTT server "test.mosquitto.org" to send and receive messages.

This section describes the operation of the provided program. With this sample program, you can check the Publish/Subscribe operation with only one set, or you can use two sets to check the Publish/Subscribe operation with each other.

2.2.1 Operation in 1 set

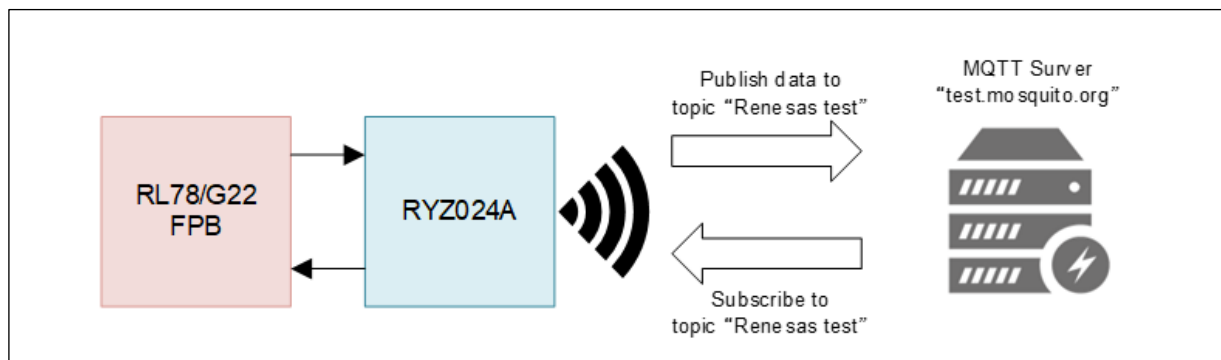


Figure 2-9 sample application system configuration in 1 set

In this sample program, after resetting RYZ024A module, connects to the cellular network via LTE and then connects to an MQTT server. After the connection to the MQTT server is completed and subscribe requests are made, the string "SW READY" is displayed in console. In this state, the user can operate the push button switches on the board.

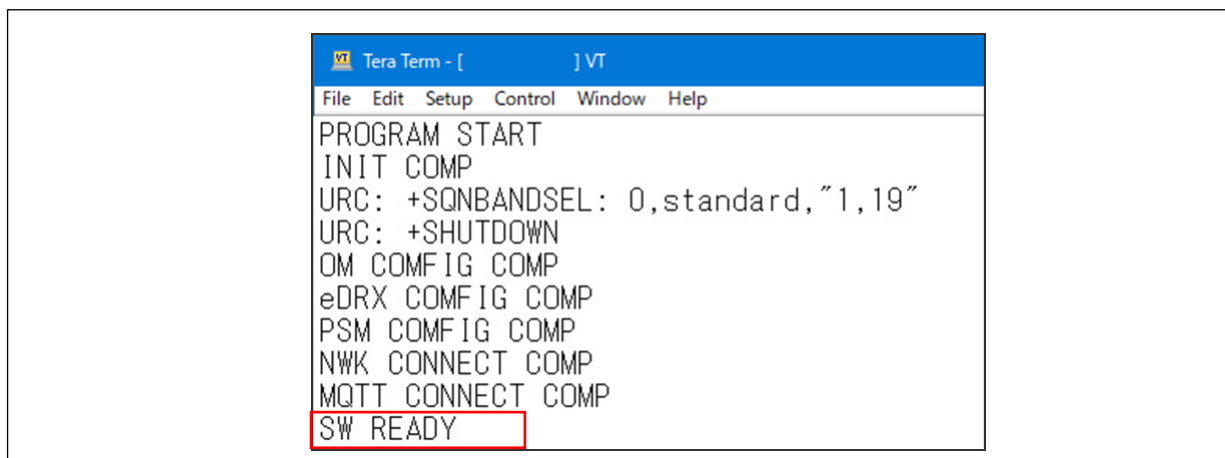
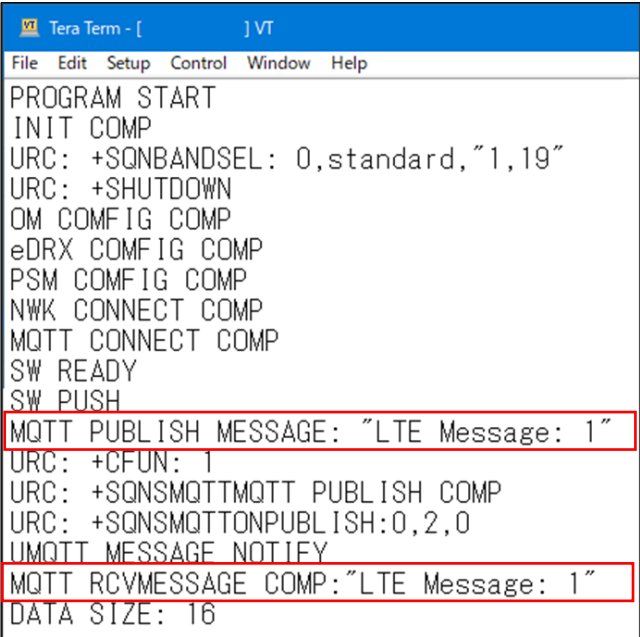


Figure 2-10 Connects MQTT server

After "SW READY" is displayed, press the push button user switch on the board to send a message to the MQTT server by publish request. RL78/G22 makes a Subscribe request to the MQTT server in advance. When the MQTT server receives Publish, it sends the ID of the message to the Subscriber. The Application sends message receive request using this subscribed message and displays the received message in console. Transmitting string data changes depending on the number count of user switch press.

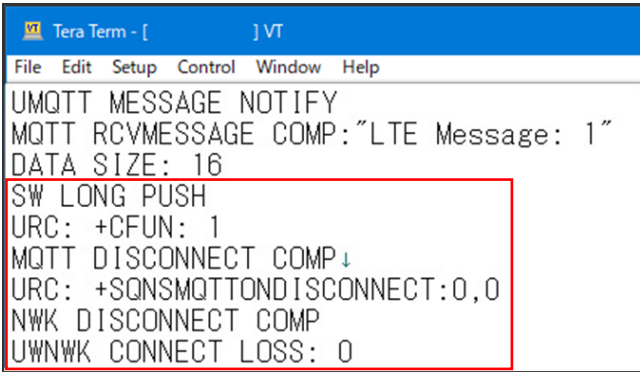


```

Tera Term - [ ] VT
File Edit Setup Control Window Help
PROGRAM START
INIT COMP
URC: +SQNBANDSEL: 0,standard,"1,19"
URC: +SHUTDOWN
OM COMFIG COMP
eDRX COMFIG COMP
PSM COMFIG COMP
NWK CONNECT COMP
MQTT CONNECT COMP
SW READY
SW PUSH
MQTT PUBLISH MESSAGE: "LTE Message: 1"
URC: +CFUN: 1
URC: +SQNSMQTTMQTT PUBLISH COMP
URC: +SQNSMQTTTONPUBLISH:0,2,0
UMQTT MESSAGE NOTIFY
MQTT RCVMESAGE COMP:"LTE Message: 1"
DATA SIZE: 16
  
```

Figure 2-11 Press user switch

Press and hold (1 second or more) the user switch to disconnect both from MQTT server and network.



```

Tera Term - [ ] VT
File Edit Setup Control Window Help
UMQTT MESSAGE NOTIFY
MQTT RCVMESAGE COMP:"LTE Message: 1"
DATA SIZE: 16
SW LONG PUSH
URC: +CFUN: 1
MQTT DISCONNECT COMP↓
URC: +SQNSMQTTTONDISCONNECT:0,0
NWK DISCONNECT COMP
UWNWK CONNECT LOSS: 0
  
```

Figure 2-12 Press and hold (1 second or more)

If you are disconnected from the network or MQTT server due to the network conditions or the connection signal strength, etc., this sample application will try to connect to the MQTT server again. Therefore, after the connection to the network is reestablished, "SW READY" is displayed after reconnecting to the MQTT server without pressing a button and makes a Subscribe request. After this, you can operate the switch again.

2.2.2 Operation in 2 sets

The same sample program is used when checking the operation with two sets.

However, change "s_str_mqtt_username" in lte_entry.c to a value other than renesas_device_001 (eg renesas_device_002).

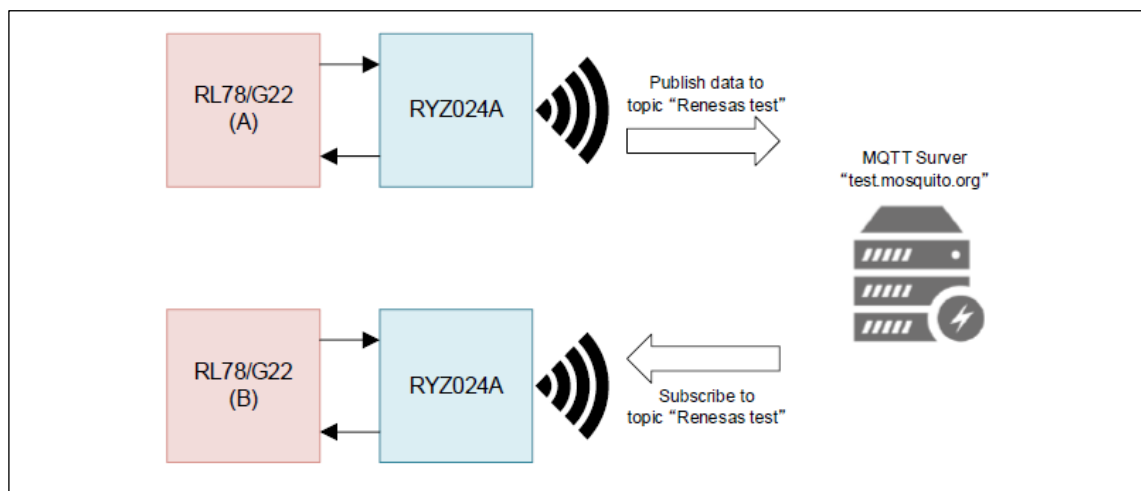


Figure 2-13 sample application system configuration in 2 sets

Code change location (lte_entry.c)

```

/* Application specific string data for MQTT Connection */
static uint8_t s_str_mqtt_serveraddress[] = "test.mosquitto.org";
static uint8_t s_str_mqtt_serverport[] = "1883";
static uint8_t s_str_mqtt_username[] = "renesas_device_001";
static uint8_t s_str_mqtt_topic[] = "renesas_test";
  
```

When the sample program is executed, RL78/G22 first resets the RYZ024A module. After resetting RYZ024A module, connects to the cellular network via LTE and then connects to an MQTT server. Next, after the connection to the MQTT server is completed and subscribe requests are made, the string "SW READY" is displayed in USB output. In this state, the user can operate the push button switches on the board.

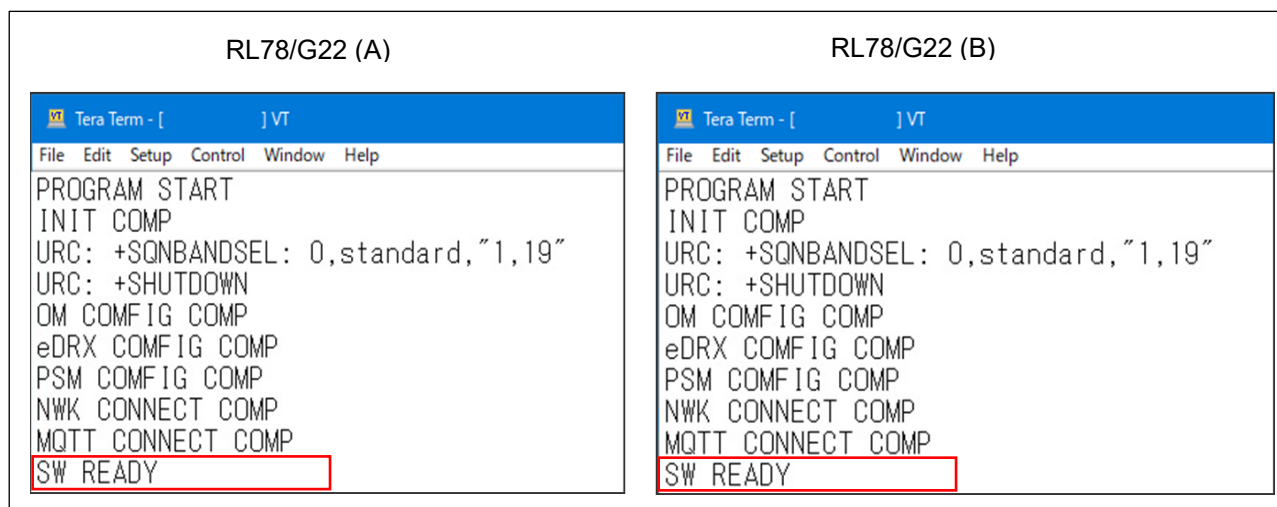


Figure 2-14 Connect MQTT server (2sets)

After "SW READY" is displayed, press the push button user switch of RL78/G22 (A) to send a message to the MQTT server by publish request. RL78/G22 (A) makes a Subscribe request to the MQTT server in advance. When the MQTT server receives Publish, it will send the ID of the message to Subscriber. The subscriber sends a request to receive a message and displays the received message on the console. Similarly, RL78/G22 (B) also makes a Subscribe request to the MQTT server, so it displays the received message on the console in the same way as RL78 (A). The character string data to be sent changes according to the number of times the user switch is pressed.

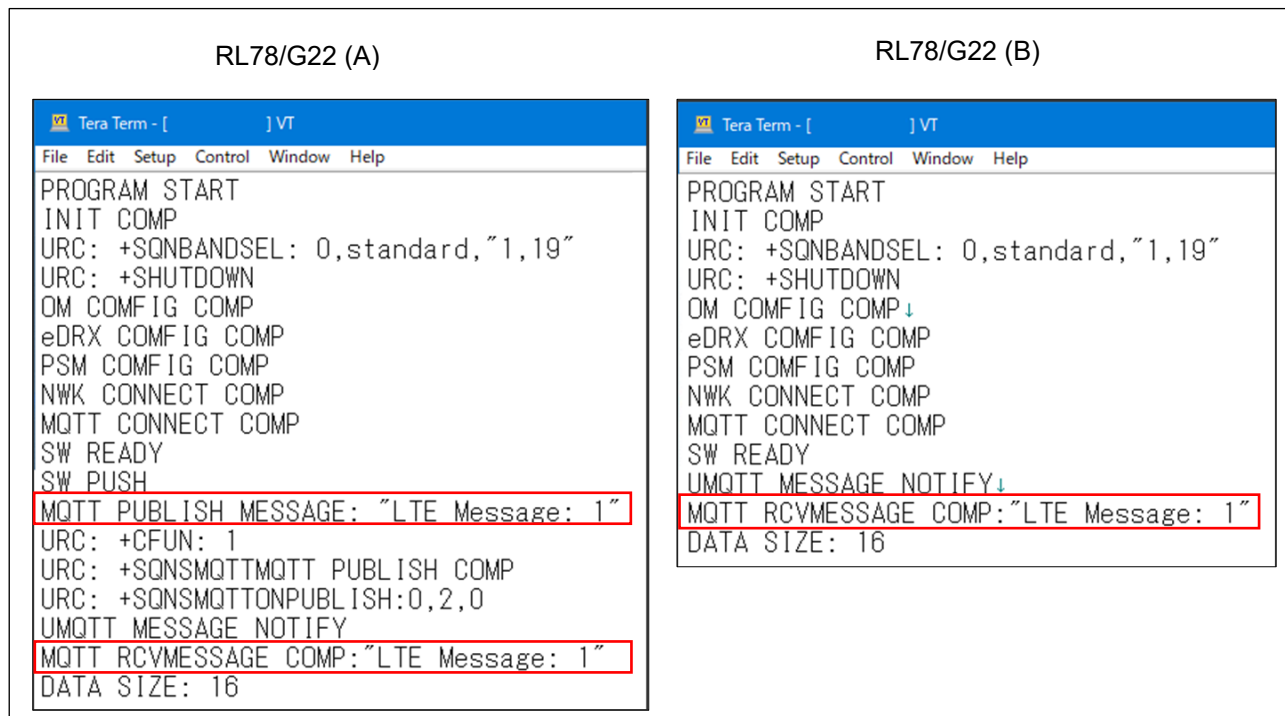


Figure 2-15 Press user switch (2sets)

Each time the user switch of RL78/G22 (A) is pressed, a message is sent from RL78/G22 (A) to RL78/G22 (B).

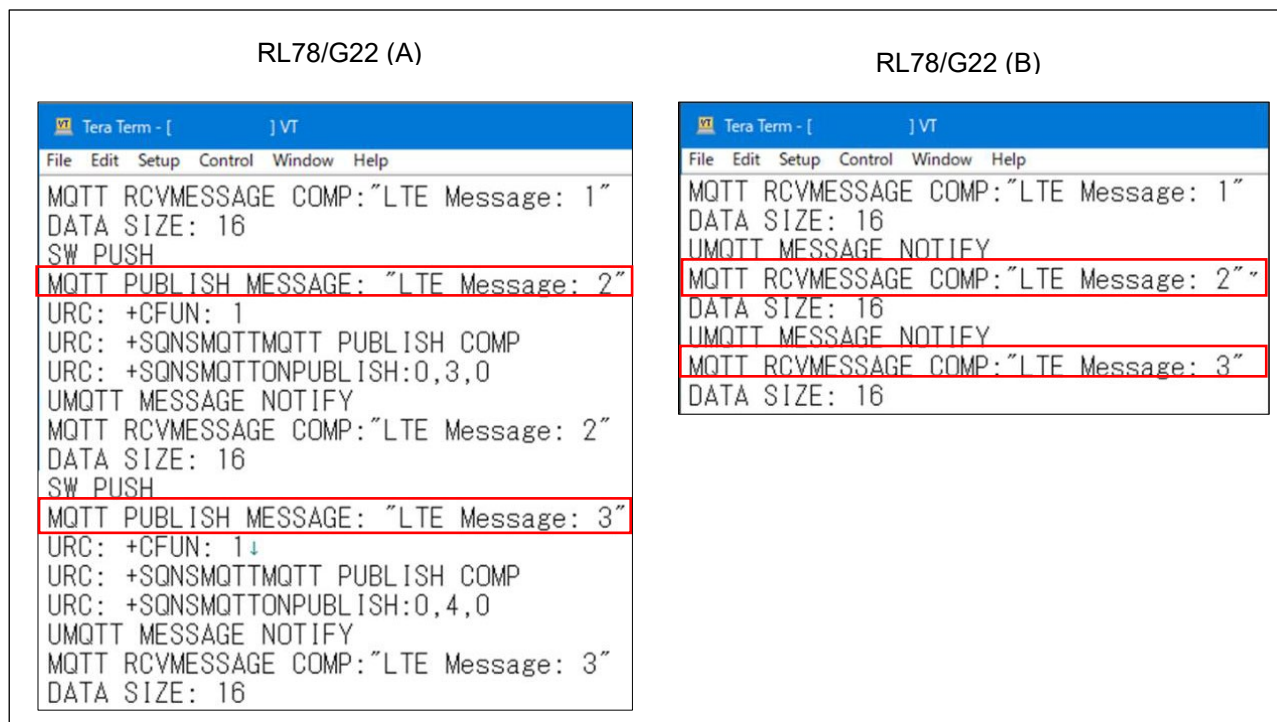


Figure 2-16 Publish from RL78/G22 (A) to RL78/G22 (B) (2sets)

By pressing the user switch of RL78/G22 (B), you can send a message from RL78/G22 (B) to RL78/G22 (A).

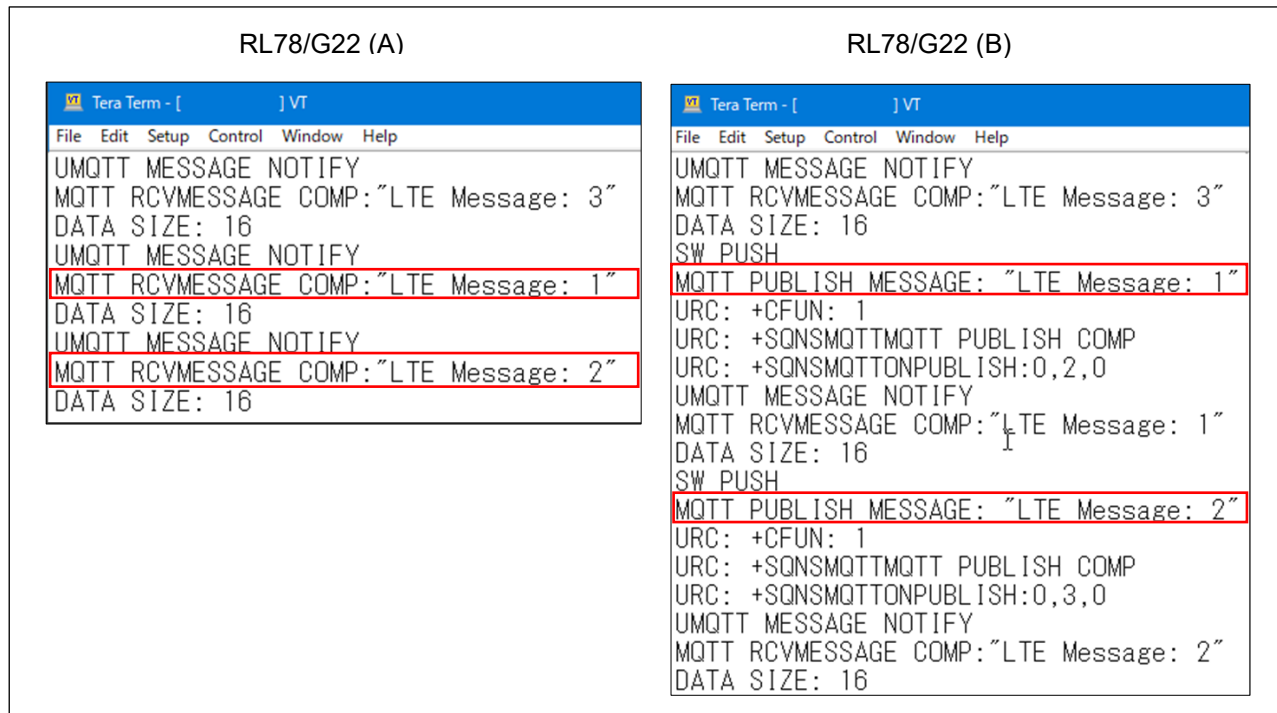


Figure 2-17 Publish from RL78/G22 (B) to RL78/G22 (A) (2sets)

3. AT Command Management Framework

3.1 Framework Overview

The RYZ024A is operated from the RL78/G22 through the transmission of AT commands and reception of responses using serial communication through the UART. The AT Command Management Framework is a framework for efficiently implementing the transmission and reception of AT commands and responses. This sample program implements a framework-based program for MQTT communication using the AT Command Management Framework.

The API implemented in the framework-based program of this sample program is classified into two types: management API and AT command API. The management API is an API for initializing framework-based programs and sending a series of AT commands in response to a response message. The AT Command API is an API for sending AT commands. The execution result of the AT command sent by the AT Command API is notified to the application as a callback function.

AT Command Management Framework is created using TAU, UARTA, PORT and interrupt controller generated by Smart Configurator.

- The UARTA is used to send AT commands to the RYZ024A and receive responses from the RYZ024A.
- The TAU is used to measure timeout condition after AT command is executed.
- The interrupt control is used for the RING signal interrupt that notifies that there is a Unsolicited Result Code (URC) from the RYZ024A.
- The Low Power module is used when the RL78/G22 is in the IDLE state or when it is waiting for a response after sending an AT command in the AT command API.

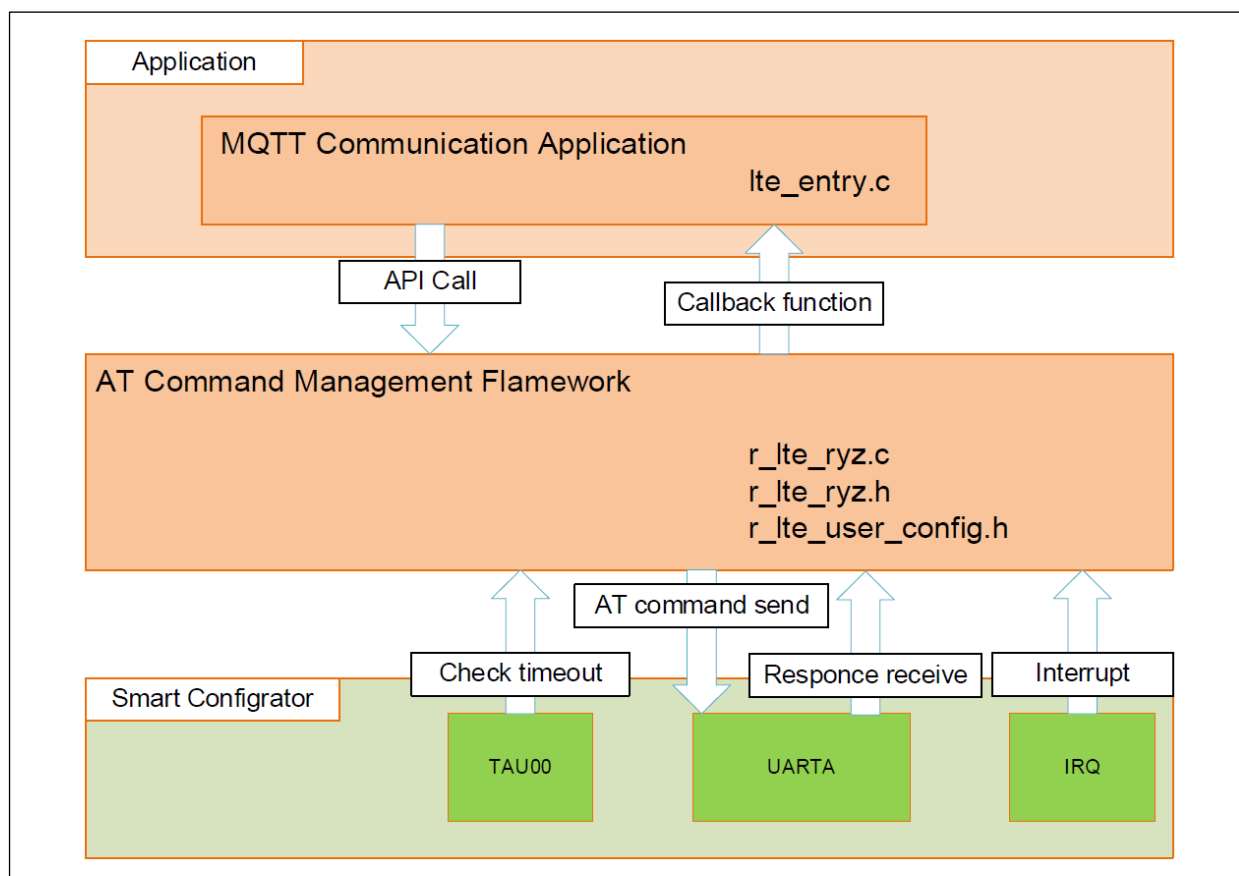


Figure 3-1 Overview of sample framework

The AT command framework implements an AT command API that sends AT commands to use the LTE communication function of the RYZ024A. The series of AT commands required to perform the desired action by calling the AT Command API in the application are added to the transmit waiting list. AT commands added to the transmit waiting list are sent to RYZ024A in order.

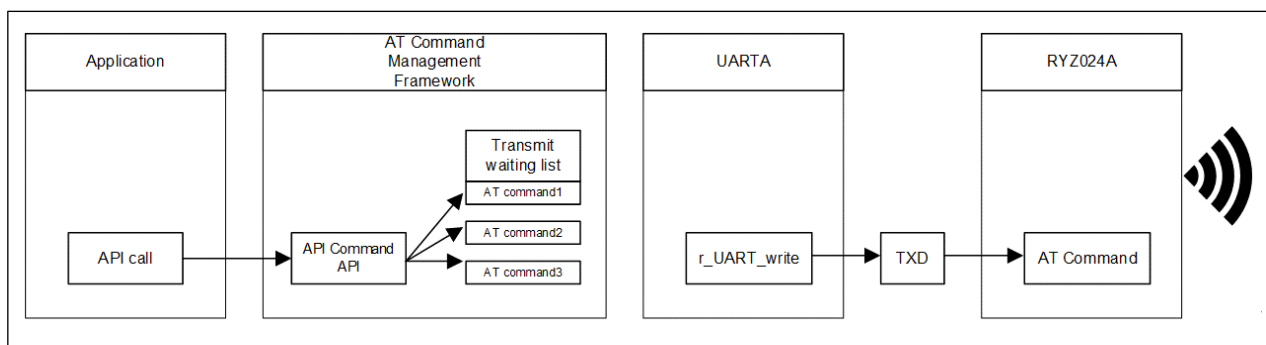


Figure 3-2 AT command API call

The result of executing the AT command on RYZ024A is sent as a response. This response data is received by the UART module's callback function and analyzed by the R_LTE_Execute function. If the execution result is correct, the R_LTE_Execute function sends the next AT command that is added to the transmit waiting list. This procedure is repeated until all AT commands added to the transmit waiting list have been sent.

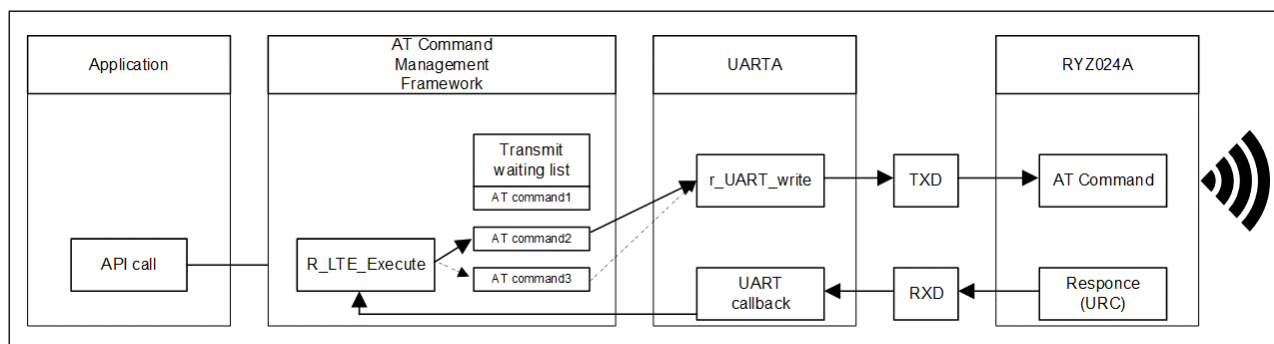


Figure 3-3 Receive response and AT command send

If all AT commands added to the send waiting list have been sent, the response from RYZ024A is an error, or other data unrelated to the AT command is received, the R_LTE_Execute function notifies the application as a callback function.

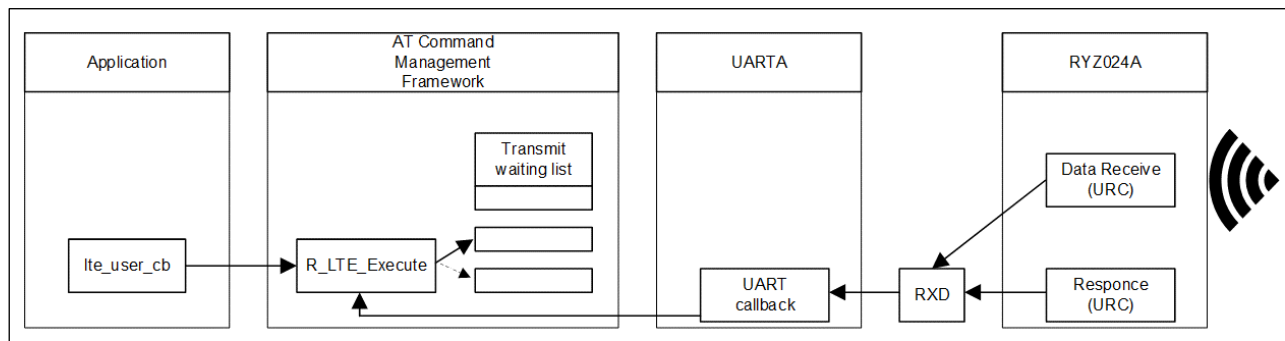


Figure 3-4 Notification in callback function

Executable API function provided from the framework-based program of this sample program is described in [3.2 API function].

The result of the API execution is notified to application through callback function. Details about callback function are described in [3.3 Callback function].

If user want to use the sample framework with other RL78 MCUs, the user only needs to change the “r_lte_user_config.h” file. Configurable values are described in [3.4 User Specific Configuration].

3.2 API functions

The API functions implemented in the framework-based program of this sample program are classified into two types: Management API and AT command API. The management API is an API for initializing the framework-based program and sending a series of AT commands in response to the response message. The AT command API is an API to send AT commands. The management API is described in [3.2.1 Management API] and AT command API is described in [3.2.2 AT command API].

3.2.1 Management API

The management API is an API for initializing the framework-based program and sending a series of AT commands in response to the response message. It must be implemented in the main routine of the application. Even when adding functions based on the AT Command Management Framework, basically there is no need to change the management API program (r_lte_ryz.c, r_lte_ryz.h, r_lte_user_config.c).

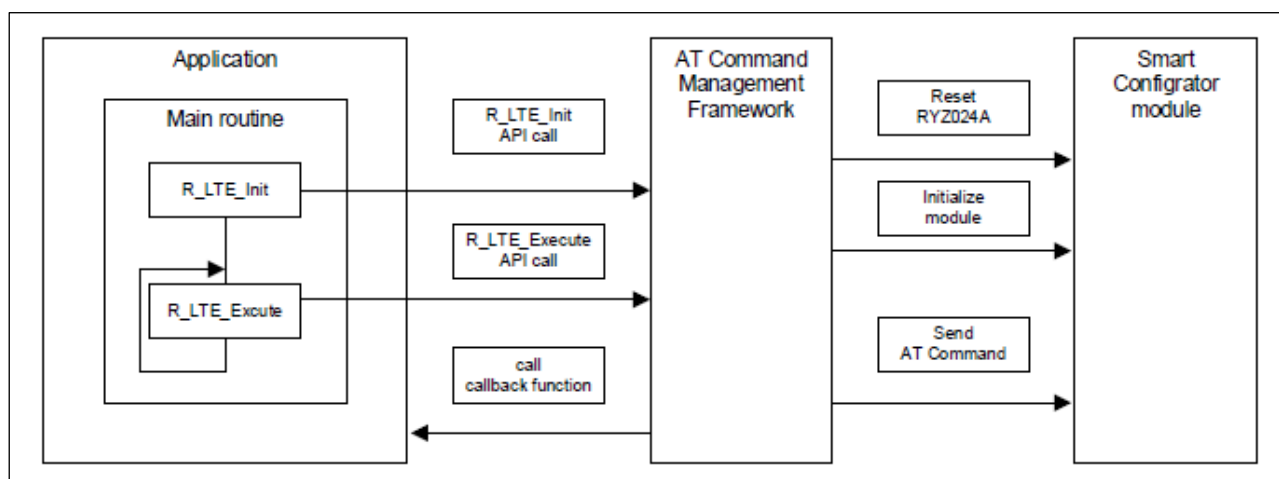


Figure 3-5 Management API

3.2.1.1 R_LTE_Init

Function name	R_LTE_Init	
Functional overview	Initialize framework-based program	
Argument	lte_cb_t * p_callback_fun (IN)	Callback function to register For information about type "lte_cb_t", refer [3.3 Callback function]
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
Advanced description	Initialize RYZ024A sample framework. As part of initialization, following operation are performed: • Initialization of Smart Configurator modules used in the framework. • Execute hardware reset of RYZ024A. • Registration of callback function to notify result of API to application. After this function is executed, AT command to reset RYZ024A will be sent. The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_INIT (0xFF) Please call this function before the main loop of your application.	

3.2.1.2 R_LTE_Execute

Function name	R_LTE_Execute	
Functional overview	Perform processing of the framework-based program.	
Argument	void	None
Return value	void	None
Advanced description	Execute various operations to be performed by the framework. The following operation are performed: • Send AT command specified by other API • Parse string data received from RYZ024A • Notifies the application of completion of API operation or receipt of errors or others by calling callback function Please call this function repeatedly in the main loop of your application.	

3.2.2 AT command API

The AT command API is an API for sending AT commands. The AT commands are added to the transmit waiting list by calling the AT command API from the application. AT commands added to the transmit waiting lists are sent sequentially to RYZ024A in response. When all AT commands specified in the AT command API have been sent, the AT command transmission result is notified to the application by callback function.

After calling the AT command API, the next AT command API cannot be called before the result is notified by the callback function. Also, the AT command API cannot be called from an interrupt handler. Call the AT command API only from main routine (including callback function of AT Command Management Framework).

The framework-based program of this sample program implements the API necessary for MQTT communication with the RYZ024A. If the user wants to implement a function that uses AT commands that are not used in MQTT communication applications, it is assumed that users will add a new AT Command API using this framework and develop an application.

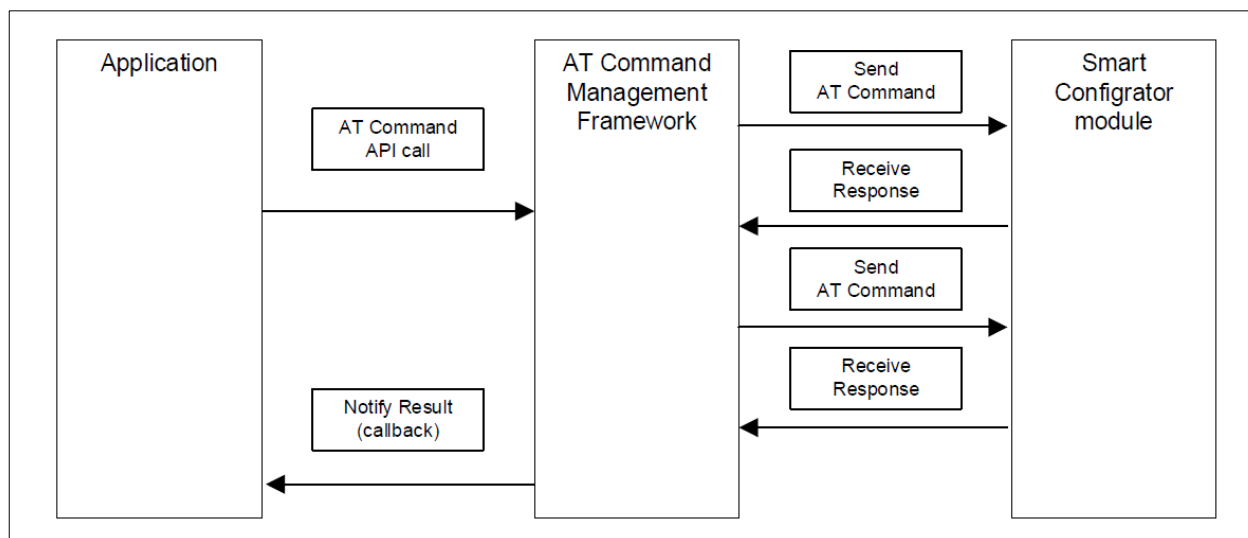


Figure 3-6 AT command API

3.2.2.1 R_LTE_OM_Config

Function Name	R_LTE_OM_Config	
Functional Overview	Configure operator mode	
Argument	uint8_t * p_pdp_type (IN)	Type of PDP context Example: "IPV4V6"
	uint8_t * p_pdp_apn (IN)	Access Point Name of PDP context Example: "ppsim.jp"
	uint8_t * p_pdp_protocol (IN)	Authentication protocol of PDP context Example: "0"
	uint8_t * p_pdp_userid (IN)	Username of PDP context Example: ""
	uint8_t * p_pdp_password (IN)	Password of PDP context Example: ""
	uint8_t * p_bandlist (IN)	List of authorized LTE bands Example: "1,2,3,4,5,8,12,13,17,18,19,20,25,26,28,66"
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	The data size of the argument exceeds the size that can be registered in transmit waiting list
Advanced description	Sends the following AT commands in order: 1. "AT+CFUN=0" 2. "AT+CGDCONT=1,[p_pdp_type],[p_pdp_apn]" 3. "AT+CGAUTH=1,[p_pdp_protocol],[p_pdp_userid],[p_pdp_password]" or "AT+CGAUTH=1,0" (Note) 4. "AT+SQNCTM="standard" 5. "AT+SQNBANDSEL=0," standard ",[p_bandlist]" 6. "AT^RESET" 7. "AT+CMEE=1" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_OM_CONFIG (0x01) Note: If "0" is specified for p_pdp_protocol, "AT+CGAUTH=1,0" is sent.	

3.2.2.2 R_LTE_NWK_Connect

Function Name	R_LTE_NWK_Connect	
Functional Overview	Connect to network	
Argument	uint8_t mode	Select sending AT command
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
Advanced description	Sends the following AT commands in order depending on value of “mode” (only 0 can be used): - Mode = 0 “AT+CEREG=5” “AT+CFUN=1” The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_NWK_CONNECT (0x02)	

3.2.2.3 R_LTE_NWK_Disconnect

Function Name	R_LTE_NWK_Disconnect	
Functional Overview	Disconnect from network	
Argument	uint8_t mode	Select sending AT command
Return value	LTE_SUCCESS (0x0000)	LTE_SUCCESS (0x0000)
	LTE_ERR_IN_PROCESS (0x0002)	LTE_FAIL_IN_PROCESS (0x0002)
Advanced description	Sends the following AT commands in order depending on value of “mode” (only 0 can be used): - Mode = 0 “AT+CFUN=0” The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_NWK_DISCONNECT (0x03)	

3.2.2.4 R_LTE_MQTT_Connect

Function Name	R_LTE_MQTT_Connect	
Functional Overview	Configure MQTT communication setting and connect to MQTT server	
Argument	uint8_t * p_username (IN)	Username used in MQTT communication Example: "renesas_device_001"
	uint8_t * p_host (IN)	Address of MQTT server to connect Example: "test.mosquitto.org"
	uint8_t * p_port (IN)	Port of MQTT server to connect Example: "1883"
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	The data size of the argument exceeds the size that can be registered in transmit waiting list
Advanced description	Sends the following AT commands in order: 1. "AT+SQNSMQTTTCFG=0,[p_username]" 2. "AT+SQNSMQTTCONNECT=0,[p_host],[p_port]" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_MQTT_CONNECT (0x04)	

3.2.2.5 R_LTE_MQTT_Subscribe

Function Name	R_LTE_MQTT_Subscribe	
Functional Overview	Specify topic to subscribe in MQTT communication	
Argument	uint8_t * p_topic (IN)	Topic to subscribe in MQTT communication Example: "renesas_test"
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	The data size of the argument exceeds the size that can be registered in transmit waiting list
Advanced description	Sends the following AT command: 1. AT+SQNSMQTTSUBSCRIBE=0,[p_topic],1" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_MQTT_SUBSCRIBE (0x05)	

3.2.2.6 R_LTE_MQTT_Publish

Function Name	R_LTE_MQTT_Publish	
Functional Overview	Publish message to MQTT server	
Argument	uint8_t * p_topic (IN)	Topic of publishing message Example: "renesas_test"
	uint16_t length (IN)	Publishing message size
	uint8_t * p_message (IN)	Publishing message
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	The data size of the argument exceeds the size that can be registered in transmit waiting list
Advanced description	Sends the following AT command: 1. "AT+SQNSMQTTPUBLISH=0,[p_topic],1,[length]" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_MQTT_PUBLISH (0x06)	

3.2.2.7 R_LTE_MQTT_RcvMessage

Function Name	R_LTE_MQTT_RcvMessage	
Functional Overview	Receive message from MQTT server	
Argument	uint8_t * p_topic (IN)	Topic of receiving message Example: "renesas_test"
	uint8_t message_id (IN)	Message ID of receiving message
	uint16_t message_size (IN)	Size of receiving message
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	The data size of the argument exceeds the size that can be registered in transmit waiting list
Advanced description	Sends the following AT commands in order depending on value of "message_id": - message_id = 0 - AT+SQNSMQTTTRCVMESSAGE=0,[p_topic]" - message_id = [other than 0] - AT+SQNSMQTTTRCVMESSAGE=0,[p_topic],[message_id]" The result of the AT command sent by this API is notified by the callback function. Received message will also be notified by callback function. Following API_ID is used in callback function. LTE_API_MQTT_RCVMESSAGE (0x07) This API is normally executed after receiving a URC of "+SQNSMQTTTONMESSAGE". If the MQTT message corresponding to the message_id specified in the argument has not yet been received, the error "LTE_CME_ERR_OPERATION_NOT_SUPPORTED" will be notified in the callback function.	

3.2.2.8 R_LTE_MQTT_Disconnect

Function Name	R_LTE_MQTT_Disconnect	
Functional Overview	Disconnect from MQTT server	
Argument	void	None
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
Advanced description	Sends the following AT command: "AT+SQNSMQTTDISCONNECT=0" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_MQTT_DISCONNECT (0x08)	

3.2.2.9 R_LTE_SEC_CertificateAdd

Function Name	R_LTE_SEC_CertificateAdd	
Functional Overview	Add certification information	
Argument	uint8_t cet_id	Adding certification ID
	uint16_t cet_len	Data length of adding certification
	uint8_t * p_cet_data	String data of certification
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	The data size of the argument exceeds the size that can be registered in transmit waiting list
Advanced description	Sends the following AT command: 1. "AT+SQNSNVW="certificate",[cet_id],[cet_len]" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_SEC_CERTIFICATEADD (0x09)	

3.2.2.10 R_LTE_SEC_CertificateRemove

Function Name	R_LTE_SEC_CertificateRemove	
Functional Overview	Remove certification information	
Argument	uint8_t cet_id	Removing certification ID
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
Advanced description	Sends the following AT command: 1. "AT+SQNSNVW="certificate",[cet_id],0" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_SEC_CERTIFICATEREMOVE (0x0A)	

3.2.2.11 R_LTE_SEC_PrivateKeyAdd

Function Name	R_LTE_SEC_PrivateKeyAdd	
Functional Overview	Add private key information	
Argument	uint8_t prk_id	Adding private key ID
	uint16_t prk_len	Data length of adding private key
	uint8_t * p_prk_data	String data of private key
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	The data size of the argument exceeds the size that can be registered in transmit waiting list
Advanced description	Sends the following AT command: 1. "AT+SQNSNVW="privatekey",[prk_id],[prk_len]" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_SEC_PRIVATEKEYADD (0x0B)	

3.2.2.12 R_LTE_SEC_PrivateKeyRemove

Function Name	R_LTE_SEC_PrivateKeyRemove	
Functional Overview	Remove private key information	
Argument	uint8_t prk_id	Remove private key ID
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
Advanced description	Sends the following AT command: 1. "AT+SQNSNVW="privatekey",[prk_id],0" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_SEC_PRIVATEKEYREMOVE (0x0C)	

3.2.2.13 R_LTE_NWK_ConnectionConfig

Function Name	R_LTE_NWK_ConnectionConfig	
Functional Overview	Configure connection and security	
Argument	uint8_t ca_cer_id	CA certification ID
	uint8_t client_cer_id	Client certification ID
	uint8_t prk_id	Private key ID
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
Advanced description	Sends the following AT commands in order: 1. "AT+SQNSCFG=1,1,1" 2. "AT+SQNSPCFG=1,2,,5,[ca_cer_id],[client_cer_id],[prk_id],""" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_NWK_CONNECTIONCONFIG (0x0D)	

3.2.2.14 R_LTE_eDRX_Config

Function Name	R_LTE_eDRX_Config	
Functional Overview	Set operation and parameters of eDRX	
Argument	uint8_t mode (IN)	eDRX mode
	uint8_t edrx_time_value (IN)	eDRX cycle
	uint8_t ptw_time_value (IN)	PTW(paging time window) time
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	The data size of the argument exceeds the size that can be registered in transmit waiting list
Advanced description	Generates and sends an AT command string from the specified arguments: AT+SQNEDRX=2,4,"0001","0000" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_EDRX_CONFIG (0x0E)	

(1) mode parameter

```
typedef enum
{
    LTE_EDRX_MODE_DISABLE = 0,
    LTE_EDRX_MODE_ENABLE,
    LTE_EDRX_MODE_ENABLE_WITH_URC,
    LTE_EDRX_MODE_RESET_PARAM,
} e_lte_edrx_mode_t;
```

(2) edrx_time_value parameter

```
typedef enum
{
    LTE_EDRX_TIME_VAL_5_SEC = 0,
    LTE_EDRX_TIME_VAL_10_SEC,
    LTE_EDRX_TIME_VAL_20_SEC,
    LTE_EDRX_TIME_VAL_40_SEC,
    LTE_EDRX_TIME_VAL_61_SEC,
    LTE_EDRX_TIME_VAL_81_SEC,
    LTE_EDRX_TIME_VAL_102_SEC,
    LTE_EDRX_TIME_VAL_122_SEC,
    LTE_EDRX_TIME_VAL_143_SEC,
    LTE_EDRX_TIME_VAL_163_SEC,
    LTE_EDRX_TIME_VAL_327_SEC,
    LTE_EDRX_TIME_VAL_655_SEC,
    LTE_EDRX_TIME_VAL_1301_SEC,
    LTE_EDRX_TIME_VAL_2621_SEC,
} e_lte_edrx_time_value_t;
```

(3) ptw_time_value parameter

```
typedef enum
{
    LTE_EDRX_PTW_TIME_VAL_1_SEC = 0,
    LTE_EDRX_PTW_TIME_VAL_2_SEC,
    LTE_EDRX_PTW_TIME_VAL_3_SEC,
    LTE_EDRX_PTW_TIME_VAL_5_SEC,
    LTE_EDRX_PTW_TIME_VAL_6_SEC,
    LTE_EDRX_PTW_TIME_VAL_7_SEC,
    LTE_EDRX_PTW_TIME_VAL_8_SEC,
    LTE_EDRX_PTW_TIME_VAL_10_SEC,
    LTE_EDRX_PTW_TIME_VAL_11_SEC,
    LTE_EDRX_PTW_TIME_VAL_12_SEC,
    LTE_EDRX_PTW_TIME_VAL_14_SEC,
    LTE_EDRX_PTW_TIME_VAL_15_SEC,
    LTE_EDRX_PTW_TIME_VAL_16_SEC,
    LTE_EDRX_PTW_TIME_VAL_17_SEC,
    LTE_EDRX_PTW_TIME_VAL_19_SEC,
    LTE_EDRX_PTW_TIME_VAL_20_SEC,
} e_lte_edrx_ptw_time_value_t;
```

3.2.2.15 R_LTE_PSM_Config

Function Name	R_LTE_PSM_Config	
Functional Overview	Set operation and parameters of PSM	
Argument	uint8_t mode (IN)	PSM mode
	uint8_t tau_time_value (IN)	TAU time
	uint8_t tau_multiplier (IN)	Multiplier for TAU time
	uint8_t active_time_value (IN)	Active time
	uint8_t active_multiplier (IN)	Multiplier for Active time
Return value	LTE_SUCCESS (0x0000)	API call success
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	The data size of the argument exceeds the size that can be registered in transmit waiting list
Advanced description	Generates and sends an AT command string from the specified arguments: AT+CPSMS=1,,,"10000010","00001111" The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. LTE_API_PSM_CONFIG (0x0F)	

(1) mode parameter

```
typedef enum
{
    LTE_PSM_MODE_DISABLE = 0,
    LTE_PSM_MODE_ENABLE,
    LTE_PSM_MODE_RESET_PARAM,
} e_lte_psm_mode_t;
```

(2) tau_time_value

```
typedef enum
{
    LTE_PSM_TAU_TIME_VAL_10_MIN = 0,
    LTE_PSM_TAU_TIME_VAL_1_HOUR,
    LTE_PSM_TAU_TIME_VAL_10_HOUR,
    LTE_PSM_TAU_TIME_VAL_2_SEC,
    LTE_PSM_TAU_TIME_VAL_30_SEC,
    LTE_PSM_TAU_TIME_VAL_1_MIN,
    LTE_PSM_TAU_TIME_VAL_320_HOUR,
} e_lte_psm_tau_time_value_t;
```

(3) active_time_value

```
typedef enum
{
    LTE_PSM_ACTIVE_TIME_VAL_2_SEC = 0,
    LTE_PSM_ACTIVE_TIME_VAL_1_MIN,
    LTE_PSM_ACTIVE_TIME_VAL_6_MIN,
    LTE_PSM_ACTIVE_TIME_VAL_NONE = 7,
} e_lte_psm_active_time_value_t;
```

(4) tau_multiplier, active_multiplier

```
typedef enum
{
    LTE_PSM_MULTIPLIER_0 = 0,
    LTE_PSM_MULTIPLIER_1,
    LTE_PSM_MULTIPLIER_2,
    LTE_PSM_MULTIPLIER_3,
    LTE_PSM_MULTIPLIER_4,
    LTE_PSM_MULTIPLIER_5,
    LTE_PSM_MULTIPLIER_6,
    LTE_PSM_MULTIPLIER_7,
    LTE_PSM_MULTIPLIER_8,
    LTE_PSM_MULTIPLIER_9,
    LTE_PSM_MULTIPLIER_10,
    LTE_PSM_MULTIPLIER_11,
    LTE_PSM_MULTIPLIER_12,
    LTE_PSM_MULTIPLIER_13,
    LTE_PSM_MULTIPLIER_14,
    LTE_PSM_MULTIPLIER_15,
    LTE_PSM_MULTIPLIER_16,
    LTE_PSM_MULTIPLIER_17,
    LTE_PSM_MULTIPLIER_18,
    LTE_PSM_MULTIPLIER_19,
    LTE_PSM_MULTIPLIER_20,
    LTE_PSM_MULTIPLIER_21,
    LTE_PSM_MULTIPLIER_22,
    LTE_PSM_MULTIPLIER_23,
    LTE_PSM_MULTIPLIER_24,
    LTE_PSM_MULTIPLIER_25,
    LTE_PSM_MULTIPLIER_26,
    LTE_PSM_MULTIPLIER_27,
    LTE_PSM_MULTIPLIER_28,
    LTE_PSM_MULTIPLIER_29,
    LTE_PSM_MULTIPLIER_30,
    LTE_PSM_MULTIPLIER_31,
} e_lte_psm_tau_multiplier_t;
```

3.3 Callback function

When an AT command is sent to the RYZ024A, string data is received as a response. Also, RYZ024A sends an Unsolicited Response Code (URC) that is not a result of an AT command. The framework-based program of this sample program receives string data from RYZ024A, and then parses the string data within the function R_LTE_Execute. If information is needed to be notified to the user application, the function R_LTE_Execute calls a callback function to notify the user application. This allows the application to check the execution result of the AT Command API and to check the URC of the RYZ024A. This section describes the structure of the callback function and the events and data that are signaled by the callback function.

The callback function has the following structure.

Type name	void * lte_cb_t	
Argument	uint16_t event_type (In)	Notified event ID Refer IDs in Table 3-1.
	uint16_t api_id (In)	ID identifying API which framework-based program is processing. Refer IDs in Table 3-2.
	uint16_t data_len (in)	Data size of "p_data"
	void * p_data (out)	Notified event data. Value changes depending on notified event type

The event_type and api_id values use values defined in macro formats within framework-based programs. The values for each are shown below.

Table 3-1 Event Type IDs (event_type) and value

Macro	Value	Description
LTE_EVENT_API_COMPLETE	0x0000	An event that notifies application that the operation specified in the API function has completed successfully. "p_data" is set according to the called API.
LTE_EVENT_ERROR	0x0001	An event that notifies application that an error has occurred in the behavior specified in the API function. Numeric data of error is set to "p_data".
LTE_EVENT_RCVURC	0x0002	An event that notifies application that a URC has been received. String data of URC is set to "p_data".
LTE_EVENT_TIMEOUT_ERROR	0x0003	An event that notifies application that timeout error has occurred for sending AT command and receiving a response. Timeout occurs when 60s has passed after sending an AT command.
LTE_EVENT_FATAL_ERROR	0x0004	An event that is notified when a fatal error occurs. Call the callback function when URC "+SYSSTART" is received at an unintended timing.

Table 3-2 API IDs (api_id) and value

Macro	Value	Corresponding API
LTE_API_NO_CURRENT_API	0x0000	None
LTE_API_OM_CONFIG	0x0001	R_LTE_OM_Config
LTE_API_NWK_CONNECT	0x0002	R_LTE_NWK_Connect
LTE_API_NWK_DISCONNECT	0x0003	R_LTE_NWK_Disconnect
LTE_API_MQTT_CONNECT	0x0004	R_LTE_MQTT_Connect
LTE_API_MQTT_DISCONNECT	0x0005	R_LTE_MQTT_Disconnect
LTE_API_MQTT_SUBSCRIBE	0x0006	R_LTE_MQTT_Subscribe
LTE_API_MQTT_PUBLISH	0x0007	R_LTE_MQTT_Publish
LTE_API_MQTT_RCVMESSAGE	0x0008	R_LTE_MQTT_RcvMessage
LTE_API_SEC_CERTIFICATEADD	0x0009	R_LTE_SEC_CertificateAdd
LTE_API_SEC_CERTIFICATEREMOVE	0x000A	R_LTE_SEC_CertificateRemove
LTE_API_SEC_PRIVATEKEYADD	0x000B	R_LTE_SEC_PrivateKeyAdd
LTE_API_SEC_PRIVATEKEYREMOVE	0x000C	R_LTE_SEC_PrivateKeyRemove
LTE_API_NWK_CONNECTIONCONFIG	0x000D	R_LTE_NWK_ConnectionConfig
LTE_API_EDRX_CONFIG	0x000E	R_LTE_eDRX_Config
LTE_API_PSM_CONFIG	0x000F	R_LTE_PSM_Config
LTE_API_INIT	0x00FF	R_LTE_Init

The callback function is called from R_LTE_Execute function in certain situations. The following is a list of when the callback function is called and the data to be set.

The source code containing the callback function is shown below.

Program: lte_entry.c

- When all AT commands specified by the AT Command API are sent and responses are received without error:
 - Value "LTE_EVENT_API_COMPLETE" is set to "event_type".
 - In "p_data", the data is set according to the AT command to be executed.
 - ✧ When URC is received as a response to AT command, string data of received URC is registered. The size of the string data to be notified is set to "data_len"
 - ✧ When calling the AT command API that starts data receive operation such as R_LTE_MQTT_RcvMessage, the received string data is registered. If the received data size exceeds "LTE_DATA_STR_SIZE", the excess data is discarded and the data of the first half is registered. The size of the string data to be notified is set to "data_len".
 - ✧ Otherwise, no data is set in "p_data". "data_len" is set to 0.

```
void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
    if(LTE_EVENT_API_COMPLETE == event_type)
    {
        switch (api_id)
        {
            case LTE_API_OM_CONFIG:
            {
                /* Connect to network after configuration of operation mode complete
                */
                sprintf(sbuf,"OM COMFIG COMP\n");
                debug_printf(sbuf);
                R_LTE_NWK_Connect(1);
            } break;

            /* Omission */

            case LTE_API_MQTT_RCVMESSAGE:
            {
                /* Display received message */
                sprintf(sbuf,"MQTT RCVMESSAGE COMP: ");
                for(uint8_t i = 0; i < data_len; i++)
                {
                    sbuf[21+i] = p_data[i] ;
                }
                sbuf[21+data_len+0]='\n';
                sbuf[21+data_len+1]='\0';
            } break;
        }
    }
}
```

LTE_EVENT_API_COMPLETE event notification

Define which AT command API result with api_id

Received data is registered in p_data

Figure 3-7 LTE_EVENT_API_COMPLETE event notification

- When the response to the AT command sent to the RYZ024A has an error in the expected response:
 - Value "LTE_EVENT_ERROR" is set to "event_type".
 - Value indicating an error is registered in "p_data". To check this value, use function "LTE_ERROR_DECODE" to check the value in 16-bit value.

```
void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
    /* Omission */

    if(LTE_EVENT_ERROR == event_type)
    {
        /* Display API ID and Error code when error response received */
        uint16_t err_code;
        LTE_ERROR_DECODE(&err_code, p_data);
        sprintf(sbuf, "ERROR RESPONSE\n");
        debug_printf(sbuf);
        sprintf(sbuf, "API ID: %d, ERROR CODE: %d\n", api_id, err_code);
        debug_printf(sbuf);
    }
}
```

LTE_EVENT_ERROR event notification

Analyze error code with LTE_ERROR_DECODE

Figure 3-8 LTE_EVENT_ERROR event notification

- When the AT command sent to the RYZ024A times out:
 - Value "LTE_EVENT_TIMEOUT_ERROR" is set to "event_type".
 - No data is set to "p_data".
 - When a timeout occurs, it is often assumed that the behavior of the RYZ024A is abnormal. Therefore, it is recommended to perform initialization.

```
void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
    /* Omission */
    if(LTE_EVENT_TIMEOUT_ERROR == event_type)
    {
        /* Set flag to initialize in main */
        gs_reinitialize_flag = 1;
    }
}
/* Omission */

void hal_entry(void)
{
    /* Omission */
    if(1 == gs_reinitialize_flag)
    {
        /* Initialize when timeout occur */
        R_LTE_Init(lte_user_cb);
        gs_reinitialize_flag = 0;
    }
}
```

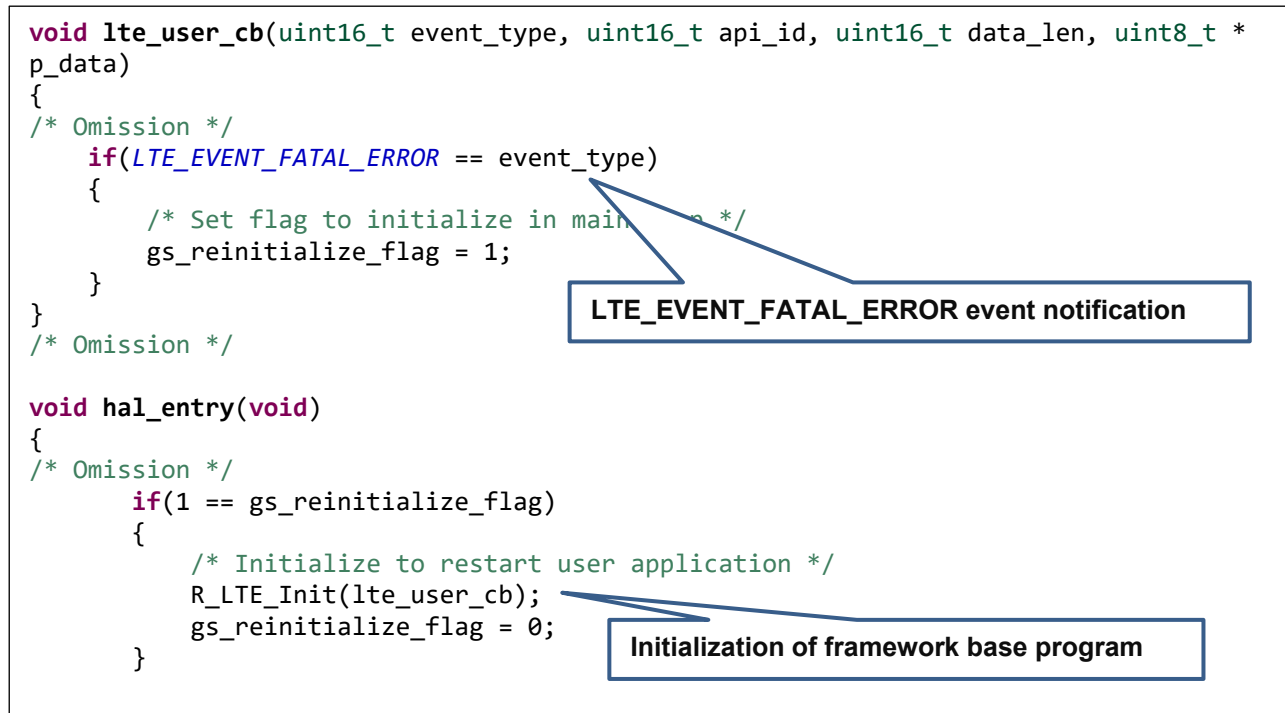
LTE_EVENT_TIMEOUT_ERROR event notification

Initialization of framework base program

Figure 3-9 LTE_EVENT_TIMEOUT_ERROR event notification

- When URC "+SYSSTART" is received from RYZ024A at an unintended timing:
 - Value "LTE_EVENT_FATAL_ERROR" is set to "event_type".
 - No data is set to "p_data".
 - When this event occurs, it is often assumed that the RYZ024A has restarted its operation. Therefore, it is recommended to perform initialization.

Note: In the normal operation of the RYZ024A, URC "+SYSSTART" will only be received if the modem reboots. This case is implemented for fail-safe purposes in case of occurrence.



```

void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
  /* Omission */
  if(LTE_EVENT_FATAL_ERROR == event_type)
  {
    /* Set flag to initialize in main program */
    gs_reinitialize_flag = 1;
  }
  /* Omission */

  void hal_entry(void)
  {
    /* Omission */
    if(1 == gs_reinitialize_flag)
    {
      /* Initialize to restart user application */
      R_LTE_Init(lte_user_cb);
      gs_reinitialize_flag = 0;
    }
  }

```

The diagram shows two callouts with arrows pointing to specific lines of code. The first callout, labeled "LTE_EVENT_FATAL_ERROR event notification", points to the line `if(LTE_EVENT_FATAL_ERROR == event_type)`. The second callout, labeled "Initialization of framework base program", points to the line `R_LTE_Init(lte_user_cb);`.

Figure 3-10 LTE_EVENT_FATAL_ERROR event notification

- When URC is sent from RYZ024A:
 - Value "LTE_EVENT_RCVURC" is set to "event_type".
 - Received URC string data is registered to "p_data". Execute user process according to the URC. Please execute the process according to the URC. If the data size of the received URC exceeds "LTE_DATA_STR_SIZE", the excess data is discarded and the data of the first half is registered. The size of the string data to be notified is set to "data_len"

```

void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
    /* Omission */

    if(LTE_EVENT_RCVURC == event_type)
    {
        /* Receive message from MQTT server when RYZ024A received subscribe
        notification */
        const uint8_t str_onmessage[] = "+SQNSMQTTONMESSAGE";

        /* Display received URC */
        sprintf(sbuf, "URC: %s", p_data);
        debug_printf(sbuf);

        if(0 == memcmp(p_data, str_onmessage, (sizeof(str_onmessage) - 1)))
        {
            uint8_t rcv_id = 0;
            char * ptr;
            uint8_t msg_count = 0;

            /* Display subscribed notification */
            sprintf(sbuf, "MQTT MESSAGE NOTIFY\n");
            debug_printf(sbuf);

            /* Get message ID from received URC string data */
            ptr = strtok((char *)p_data, ",");
            while(ptr != NULL)
            {
                ptr = strtok(NULL, ",");
                if(ptr != NULL)
                {
                    msg_count++;
                    if(2 == msg_count)
                    {
                        mqtt_rcvdata_len = (uint8_t )atoi(ptr);
                    }
                    if(4 == msg_count)
                    {
                        rcv_id = (uint8_t )atoi(ptr);
                    }
                }
            }
            /* request message receive */
            R_LTE_MQTT_RcvMessage(str_MQTT_topic, rcv_id);
        }
    }
}

```

LTE_EVENT_RCVURC event notification

**Check received URC
Execute process if received data is
+SQNSMQTTONMESSAGE"**

Analyze parameter of URC

**Call AT command API with
analyzed parameter**

Figure 3-11 LTE_EVENT_RCVURC event notification

3.4 User Specific Configuration

When users are developing applications based on this sample application, they need to change some settings depending on the RL78 MCU used. In the AT Command Management Framework, a program for setting these user-specific setting values is defined in "r_lte_ryz.c". Users can modify this file to use the AT Command Management Framework in the configuration that suits their environment. This section describes the values that can be set.

[Table 3-3 Pin function setting for RYZ024A] shows the setting items for the RL78/G22 pins connected to each pin of the RYZ024A. Please confirm when changing the board to be used as the host MCU.

Table 3-3 Pin function setting for RYZ024A

Function name	Description	Used pin
<code>void ryz_lte_rts_low_devspe(void)</code>	The RTS pin of RYZ024A to low	P70
<code>void ryz_lte_rts_high_devspe(void)</code>	The RTS pin of RYZ024A to high	P70
<code>void ryz_lte_reset_low_devspe(void)</code>	The RESET pin of RYZ024A to low	P17
<code>void ryz_lte_reset_high_devspe(void)</code>	The RESET pin of RYZ024A to high	P17
<code>uint8_t ryz_lte_cts_read_devspe(void)</code>	The CTS pin of RYZ024A to read	P50
<code>uint8_t ryz_lte_ring_read_devspe(void)</code>	The RING pin of RYZ024A to read	P51

[Table 3-4 Smart Configurator of RL78/G22] shows the setting items for using the Smart Configurator within the AT command management framework. Please check if you want to edit the Smart Configurator, change the RL78 MCU to use, etc.

Table 3-4 Smart Configurator of RL78/G22

Tag name	Component	Description
Clock	-	Operation mode: High-speed main mode 1.8 (V) to 5.5 (V) High-speed on-chip oscillator: 32MHz f _{OCO} start setting: Normal f _{IHP} : 32MHz f _{MAIN} : 32MHz f _{CLK} : 32000kHz
System	-	On-chip debug operation setting: Use emulator Emulator setting : E2 Lite Pseudo-RRM/DMM function setting: Used Start/Stop function setting: Unused Security ID setting: Use security ID Security ID: 0x00000000000000000000 Security ID authentication failure setting: Erase flash memory data

Tag name	Component	Description
Component	r_bsp	Start up select: Enable (use BSP startup) Control of invalid memory access detection: Disable RAM guard space (GRAM0-1): Disabled Guard of control registers of port function (GPORT): Disabled Guard of registers of interrupt function (GINT): Disabled Guard of control registers of clock control function, voltage detector, and RAM parity error detection function (GCSC): Disabled Data flash access control (DFLEN): Disables Initialization of peripheral functions by Code Generator/Smart Configurator: Enable API functions disable: Enable Parameter check enable: Enable Setting for starting the high-speed on-chip oscillator at the times of release from STOP mode and of transitions to SNOOZE mode: High-speed Enable user warm start callback (PRE): Unused Enable user warm start callback (POST): Unused Watchdog Timer refresh enable: Unused
	Config_TAU0_1	Component: Interval timer Operation mode: 16 bit count mode Resource: TAU0_1 Operation clock: CK00 Clock source: $f_{CLK}/2$ Interval value: 1ms Interval setting: Used Priority: Level 3 (low)
	Config_UART0	Component: UART Communication Operation: Transmission/reception Resource : UART0 Operation clock:CK00 Clock source: $f_{CLK}/2$ Transfer mode setting: Single transfer mode Data length setting: 8 bits Transfer direction setting: LSB Parity setting: None Stop bit length setting: 1bit Transfer data level setting: Non-reverse Transfer rate setting:115200bps Transmit end interrupt priority (INTST0): Level 3(low) Callback function setting: Transmission end

Tag name	Component	Description
Component	Config_UARTA0	Component: UART Communication Operation: Transmission/reception Resource: UARTA0 Operation clock: f_{SEL} Clock source: f_{SEL} clock select f_{IHP} Data length setting: 8 bits Transfer direction setting: LSB Parity setting: None Stop bit length setting: 1bit Transfer data level setting: Non-reverse Transmit mode setting: Continuous transmit by polling Receive error occurs setting: INTUR interrupt occurs Transfer rate setting: 115200bps Reception end interrupt priority (INTUR0): Level3(low) Callback function: Reception end, Reception error
	Config_INTC	INTP0 Valid edge: Falling edge, Priority: Level3(low) INTP2 Valid edge: Falling edge, Priority: Level3(low)
	Config_PORT	Port selection: PORT1, PORT5, PORT7 Port mode setting: Read Pmn register values P17: Out P50: In P70: Out

[Table 3-5 Size setting of AT command transmission waiting list] shows the size setting items for various data used within the AT command management framework. AT command management framework defines these setting values in "r_lte_user_config.h". Please change it according to the data size of the AT command and string used in the application and the stack size of the MCU to be used.

Table 3-5 Size setting of AT command transmission waiting list

Name	Default value	Description
LTE_ATC_STR_SIZE	100	Maximum length of the AT command string.
LTE_DATA_STR_SIZE	100	The maximum length of data to receive from the RYZ024A. If the data to be received exceeds this size, the excess data is discarded.
LTE_ATC_LIST_SIZE	8	The number of AT commands that can be added to the send waiting list. Define "maximum number of AT commands to be registered + 1".

3.5 Smart Configurator module used in the framework

The AT Command Management Framework uses Smart Configurator modules to implement its functionality. The Smart Configurator module is configured not only in code but also in the Smart Configurator. This section describes how to use and configure the Smart Configurator module used in the AT Command Management Framework.

3.5.1 UARTA module

The AT Command Management Framework uses the UARTA module to implement UART communication between the RL78/G22 and the RYZ024A.

When sending AT commands from the RL78/G22 to the RYZ024A, the write function (R_Config_UARTA0_Send) of the UARTA module is used. After calling the AT Command API from your application, a series of AT commands are registered in the Transmit waiting list in the framework. Transmission of AT commands from the waiting list are sequentially processed from the beginning of the list using the write function.

When sending a response from the RYZ024A to the RL78/G22, the data is received using the callback function of the UARTA module. This callback function receives a character data one by one and stores it in a ring buffer in the framework. Character data stored in the ring buffer is processed one character at a time R_LTE_Execute each function call.

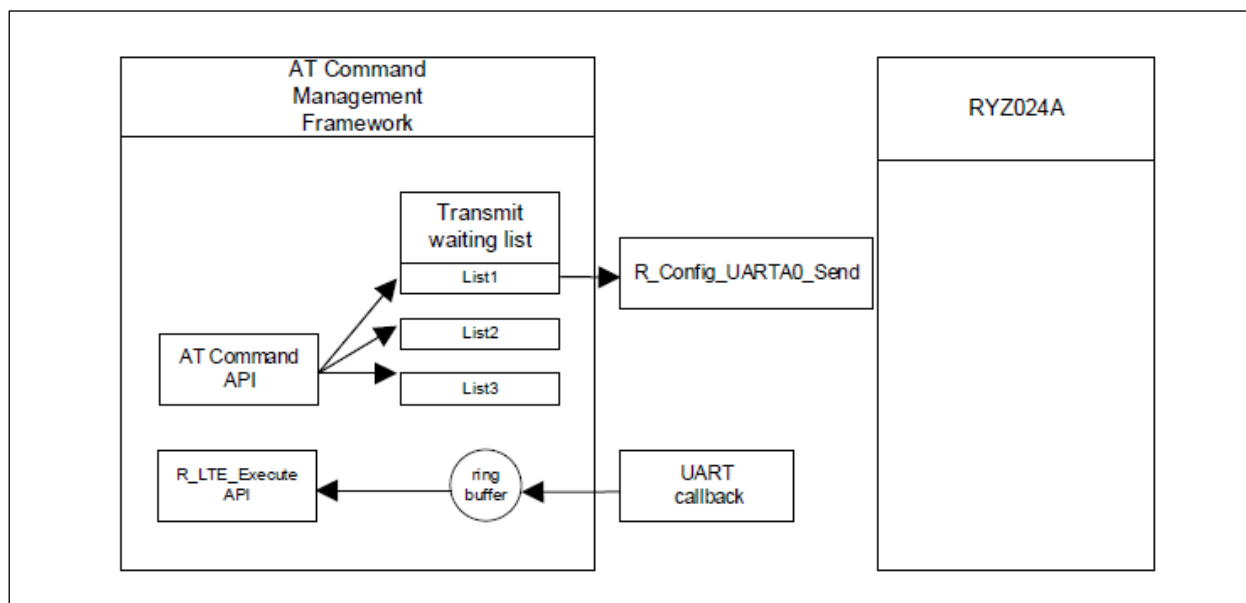


Figure 3-12 Using UARTA module

3.5.2 TAU module

The AT Command Management Framework uses the TAU module to implement the timeout function. After sending an AT command, a timeout occurs when 60 seconds elapse before receiving a response. [Table 3-6 AT command timeout setting (r_lte_ryz.c)] shows the definition that sets the timeout period.

Table 3-6 AT command timeout setting (r_lte_ryz.c)

Name	Default value	Description
AT_COMMAND_TIMETOUT	30	Timeout count of AT command. unit: 2sec e.g.) 2sec * 30 = 60sec

Framework starts the timer at the timing of sending the AT command. This timer stops when the response specified in the comp_msg is received or when an error response is received. If a response is not received for a certain period after sending an AT command, the timer callback function is called in the framework to signal a timeout has occurred. After the callback function is called, framework calls the user's callback function in the R_LTE_Execute function to notify the application that a timeout has occurred.

The timer count time is set in the Smart Configurator. To change the timeout period, use the Smart Configurator to change the timer count time and [Table 3-6 AT command timeout setting (r_lte_ryz.c)].

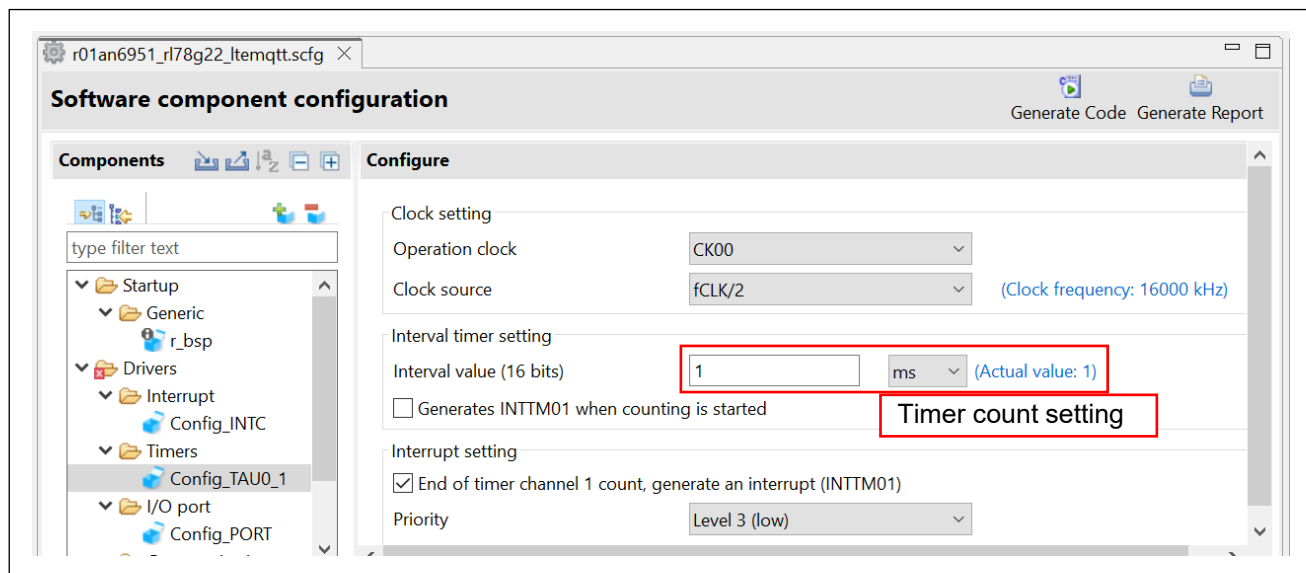


Figure 3-13 TAU module setting

3.5.3 Interrupt function

Use the interrupt function to generate an interrupt with a RING signal from the RYZ024A to notify that there is a URC. RL78/G22 uses INTP2 to detect this RING interrupt.

Also, INTP0 is used to detect that SW is pressed.

3.5.4 UART0 module

The UART0 module is used to output the execution results of the sample program. The execution results of the sample program are output via the RL78/G22 FPB's USB (COM port).

3.6 Low Power Operation

This sample application supports operation using the low power consumption function of the RL78/G22 and RYZ024A.

3.6.1 Low power operation control of RL78/G22

The RL78/G22 is in SNOOZE mode when the RL78/G22 is in the IDLE state, and in the HALT mode when it is waiting for a response after sending an AT command in the AT command API. The files and functions performing low power operation are listed below.

Application program: r_lte_ryz.c, R_LTE_Execute()

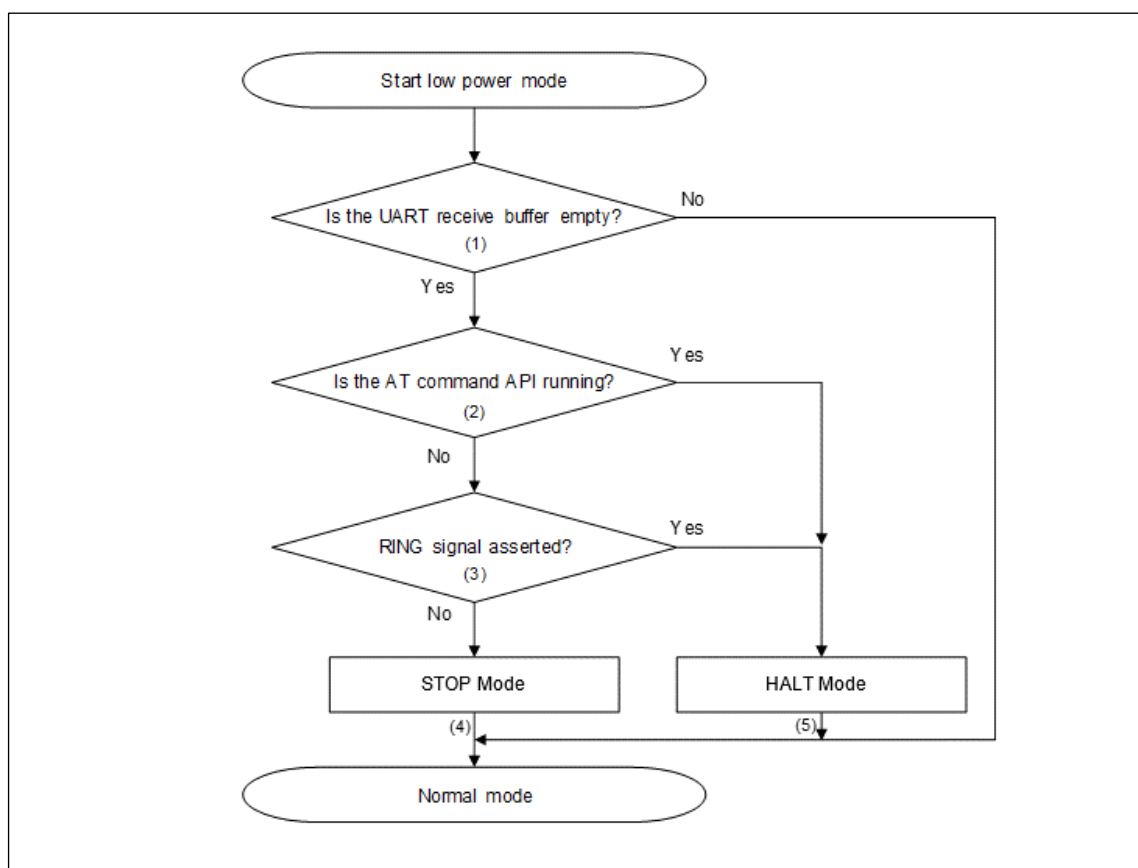


Figure 3-14 RL78/G22 low power consumption operation flow chart

- (1) If character strings or data are stored in the UART receive buffer, analysis processing is performed using the R_LTE_Execute() function without transitioning to low power consumption mode.
- (2) When the AT command API is executed, when the response of the transmitted AT command is received, it shifts to sleep mode so that it can return to normal mode with a UART reception interrupt.
- (3) While the RING signal is asserted, shift to HALT mode so that the URC from the RYZ024A can be received.
- (4) From STOP mode, SW is pressed, or an external pin interrupt is generated by asserting the RING signal to return to normal mode.
- (5) Returns to normal mode when a UART reception interrupt is generated by receiving a response to the transmitted AT command.

3.6.2 Low power operation control of RYZ024A

When the RYZ024A enables eDRX or PSM, you can operate the RYZ024A with low power consumption by controlling the RTS signal.

RTS=L: Disable low power consumption operation

RTS=H: Enable low power consumption operation

See also "RYZ024 Power Consumption Measurements on RYZ024-Based Modules" (R19AN0167) for low power operation of RYZ024A.

When sending an AT command from the RL78/G22, the RTS signal is controlled within the AT command management framework to wake up the RYZ024A in low power consumption mode. Specifically, RTS is set to Low in the AT command API and set to High after processing is completed. Also, when a RING interrupt occurs, RTS is set to Low so that the RYZ024A can transmit URC, and after the necessary processing is completed, it is set to High.

(1) When the AT command API is called

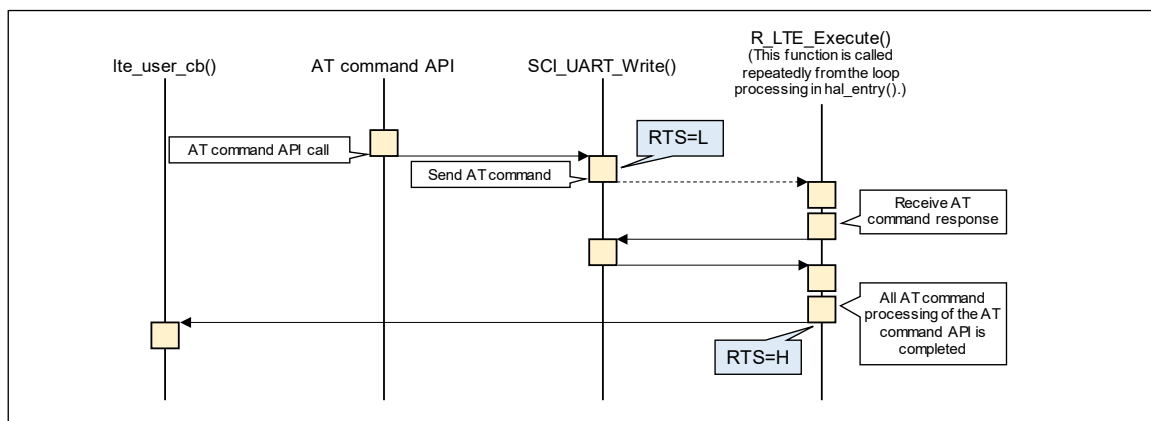


Figure 3-15 RTS signal control sequence (When the AT command API is called)

(2) When a RING interrupt occurs

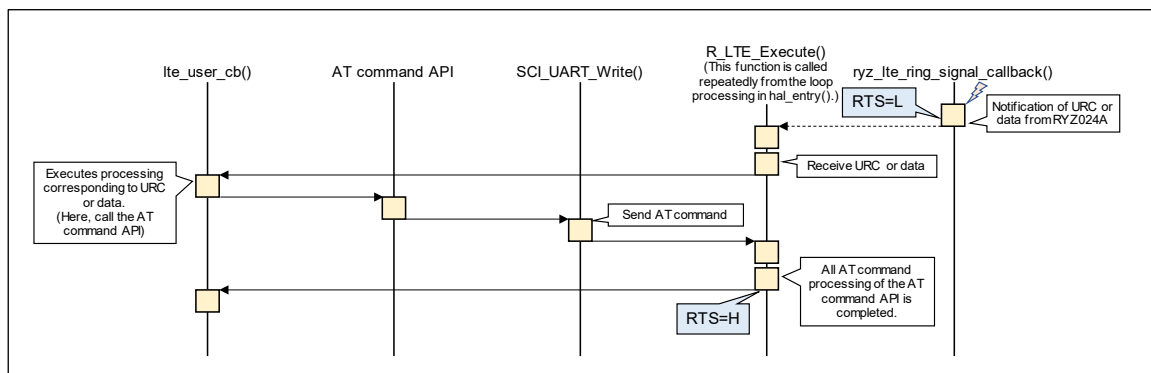
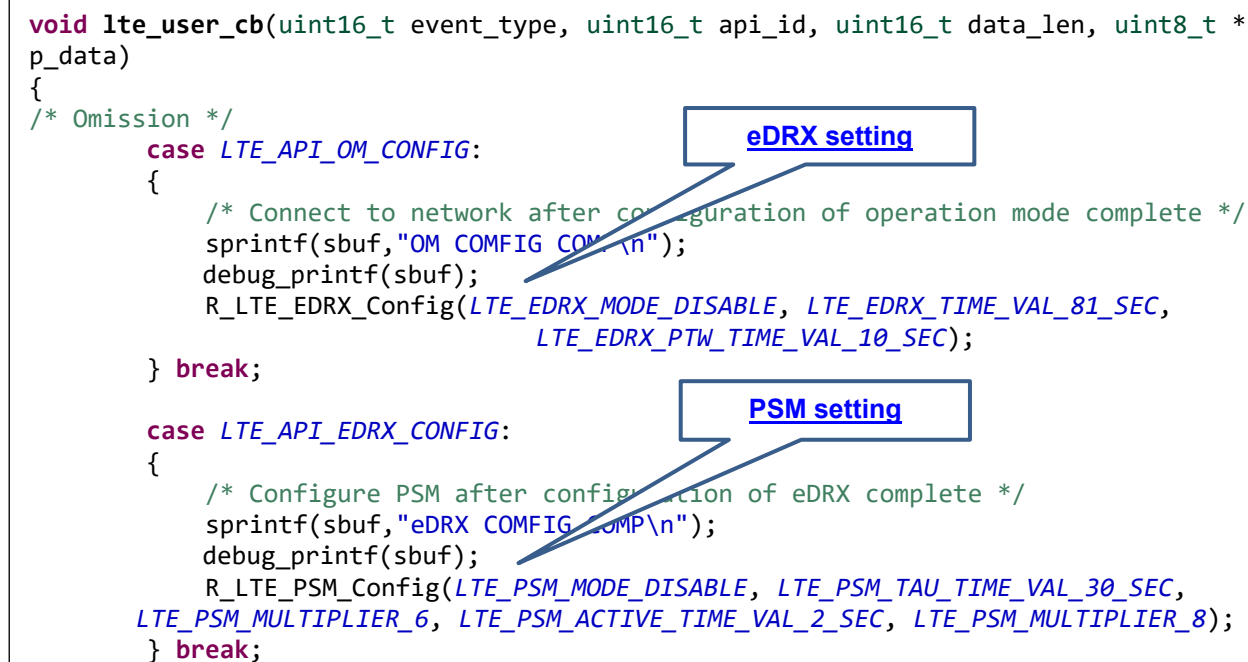


Figure 3-16 RTS signal control sequence (When a RING interrupt occurs)

Operation settings for eDRX and PSM are performed using the R_LTE_EDRX_Config function and R_LTE_PSM_Config function. In this sample application, it is executed within the callback function. Its source code is shown in Figure 3-20.

Program: lte_entry.c

eDRX and PSM operations are disabled by default. To enable it, refer to [3.2.2.14 R_LTE_eDRX_Config] and [3.2.2.15 R_LTE_PSM_Config] and change the first argument of each API.



```
void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
  /* Omission */
  case LTE_API_OM_CONFIG:
  {
    /* Connect to network after configuration of operation mode complete */
    sprintf(sbuf, "OM CONFIG COM\n");
    debug_printf(sbuf);
    R_LTE_EDRX_Config(LTE_EDRX_MODE_DISABLE, LTE_EDRX_TIME_VAL_81_SEC,
                     LTE_EDRX_PTW_TIME_VAL_10_SEC);
  } break;

  case LTE_API_EDRX_CONFIG:
  {
    /* Configure PSM after configuration of eDRX complete */
    sprintf(sbuf, "eDRX CONFIG COM\n");
    debug_printf(sbuf);
    R_LTE_PSM_Config(LTE_PSM_MODE_DISABLE, LTE_PSM_TAU_TIME_VAL_30_SEC,
                     LTE_PSM_MULTIPLIER_6, LTE_PSM_ACTIVE_TIME_VAL_2_SEC, LTE_PSM_MULTIPLIER_8);
  } break;
}
```

Figure 3-17 eDRX and PSM setting

4. Application development using AT Command Management Framework

The AT Command Management Framework is intended to be used as a base for user application development. By using the AT Command Management Framework, communication between the RL78/G22 and the RYZ024A can be efficiently implemented. In this section, we will describe how to develop user applications using this sample application as an example.

4.1 Overview of application development

The AT Command Management Framework is a specification that allows you to efficiently implement additional APIs within the framework. The API implemented in the framework is called in the application program to realize the operation desired by the user. In this sample application, operation is realized with the following file.

- Framework base program:
 - r_lte_ryz.c
 - r_lte_ryz.h
 - r_lte_user_config.h
- Bare metal Application program:
 - lte_entry.c

The APIs implemented in framework-based programs are classified into two types: management API and AT command API.

Management API

The Management API is the API for managing interactions with the RYZ024A. It must be implemented in the proper place in the application program. In addition, users do not need to change it during application development.

The following two APIs are implemented in the management API:

- R_LTE_Init
This is a function for initializing framework-based programs. This function performs initialization of the Smart Configurator and hardware reset of the RYZ024A. The RYZ024A sends a URC of "+SYSSTART" when initialization completes, and it is possible to accept AT commands. After "+SYSSTART", this function sends the AT command "AT+CMEE=1" to receive a detailed error response. After all, AT commands have finished executing, the callback function specified in the argument API_ID = "LTE_API_INIT" event is notified. This function should be executed first in all API implemented in the framework.
- R_LTE_Execute
This is a function that holds and parses the data received from RYZ024A, calls the callback function according to the data, and sends AT commands. Since this function processes each character stored in the ring buffer each time it is called, it is necessary to call it repeatedly in the main loop.

```

void lte_entry(void)
{
    sprintf(sbuf, "PROGRAM START\n");
    debug_printf(sbuf);

    /* SW interrupt driver open */
    R_ICU_ExternalIrqOpen(g_external_irq10.p_ctrl, g_external_irq10.p_cfg);
    R_ICU_ExternalIrqOpen(g_external_irq9.p_ctrl, g_external_irq9.p_cfg);

    /* Initialize framework-based program and register callback function */
    R_LTE_Init(lte_user_cb);

    while(1)
    {
        /* Execute variable process in framework-based program */
        R_LTE_Execute();

        /* Omission */
    }
}

```

Call R_LTE_Init before calling any other APIs in framework

Call R_LTE_Execute repeatedly in the main loop.

Figure 4-1 Implement management API (lte_entry.c)

AT command API

A set of AT commands necessary for the operation you want to perform is added to the transmit waiting list by calling the AT command API. The registered AT commands are sent sequentially in response to the response from the RYZ024A. The execution result of a series of AT commands is notified to the application by a callback function. Users develop applications by calling the AT command API in the order they want and implementing processing corresponding to callback functions. In addition, users can add a new AT command API by themselves and use AT commands not used in this sample application.

```

void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
    if(LTE_EVENT_API_COMPLETE == event_type)
    {
        switch (api_id)
        {
            case LTE_API_INIT:
            {
                /* Configure operation mode after association complete */
                sprintf(sbuf, "INIT COMP\n");
                debug_printf(sbuf);
                R_LTE_OM_Config(str_PDP_type, str_PDP_APN, str_LTE_bandlist);
            } break;

            case LTE_API_OM_CONFIG:
            {
                /* Connect to network after configuration of operation mode
                complete */
                sprintf(sbuf, "OM CONFIG COMP\n");
                debug_printf(sbuf);
                R_LTE_NWK_Connect(1);
            } break;
        }
    }
}

```

1. Call AT command API

2. Receive result with callback function

3. Call next AT command API

Figure 4-2 Implement AT command API (lte_entry.c)

4.2 Adding an AT command API

This framework assumes that the AT command API is added according to the user's application. This section explains how the AT command API implemented in this sample application and explains how to implement the new AT command API.

To add the AT command API, follow these steps:

1. Adding API IDs and Function Prototype Declarations

Add the API ID so that the added AT command API can be identified in the callback function. User also adds prototype declarations to the header file (r_lte_ryz.h) so that the AT Command API can be executed from the application program.

```
typedef enum
{
    LTE_API_NO_CURRENT_API = 0,
    LTE_API_OM_CONFIG,
    LTE_API_NWK_CONNECT,
    LTE_API_NWK_DISCONNECT,
    LTE_API_MQTT_CONNECT,
    LTE_API_MQTT_DISCONNECT,
    LTE_API_MQTT_SUBSCRIBE,
    LTE_API_MQTT_PUBLISH,
    LTE_API_MQTT_RCVMESSAGE,
    LTE_API_SEC_CERTIFICATEADD,
    LTE_API_SEC_CERTIFICATEREMOVE,
    LTE_API_SEC_PRIVATEKEYADD,
    LTE_API_SEC_PRIVATEKEYREMOVE,
    LTE_API_NWK_CONNECTIONCONFIG,
    LTE_API_EDRX_CONFIG,
    LTE_API_PSM_CONFIG,
    LTE_API_INIT = 0xff,
} e_lte_api_id_t;
```

Figure 4-3 API IDs of this sample application (r_lte_ryz.h)

2. Implementing the AT command API

Implement the actual state of the AT command API in the source file (r_lte_ryz.c). The AT command API of this sample application is implemented with the following configuration.

Checking arguments and checking the running AT command API

If the argument has a pointer, make sure you do not specify NULL. Also check "gs_process_api" to make sure that no other AT command API is running. If it is running, the AT command API cannot operate properly if you change the AT command transmit waiting list, so the error "LTE_ERR_IN_PROCESS" will be returned without executing any process. After that, to indicate that this AT command API is executing, register the API_ID in "gs_process_api".

```
e_lte_err_t R_LTE_OM_Config(uint8_t* p_pdp_type, uint8_t* p_pdp_apn, uint8_t*
p_bandlist)
{
    /* Check Argument and current state */
    if((NULL == p_pdp_type) || (NULL == p_pdp_apn) || (NULL == p_bandlist))
    {
        return LTE_ERR_POINTER_NULL;
    }

    if(LTE_API_NO_CURRENT_API != gs_process_api)
    {
        return LTE_ERR_IN_PROCESS;
    }

    /* Clear ATC list and set processing API ID */
    ryz_lte_clear_atc_list();
    gs_process_api = LTE_API_OM_CONFIG;

    /* Omission */
}
```

Check argument

Check running AT command API

Register running AT command API

Figure 4-4 Checking the arguments and running AT Command API of R_LTE_OM_Config (r_lte_ryz.c)

Registering AT commands in the Transmission Waiting List

Register the AT command as string data in the transmit waiting list "gs_atc_list". The following must be registered in the transmission waiting list "gs_atc_list" for one AT command.

- **atcommand:**
This is the string data of the AT command you want to execute. The length of the string should be registered in "atcommand_size". The maximum length of a string data that can be registered is 256 characters. If you want to use a larger AT command string data, change the "LTE_ATC_STR_SIZE" in the user configuration file (r_lte_user_config.h).
- **data:**
This is a pointer to register the address of the data string to be processed by the AT command. It is necessary for AT commands that send data. The data string registered here will be sent corresponding to the response of "> ". The length of the string must be registered in "data_size". It is assumed that the actual character string to be registered in this pointer is implemented in the application.
- **comp_msg:**
This is a response message that can be considered as the completion of the AT command you want to execute. Specify "OK" or URC. The length of the string should be registered in "comp_msg_size". The following AT command is sent immediately after receiving the string specified in the comp_msg. If "OK" and URC are sent consecutively, register the response to be sent last. In addition, the last comp_msg of a series of AT commands to be added to the send waiting list changes the data notified in the callback function. For details, see [3.3 Callback function].
- **data_exist_flag:**
This flag indicates that the AT command to be sent is set. R_LTE_Execute function checks this value to confirm that the AT command is registered. If you want to register the AT command, set it to "1".

The transmit waiting list "gs_atc_list" holds string data by fixed-length arrays. Therefore, if the string data to be registered exceeds the maximum length that can be registered in the transmit waiting list, an error due to a buffer overflow may occur. If the user expects the data size of string data that is being registered can exceed the maximum length, add processing to check the data size.

```

e_lte_err_t R_LTE_MQTT_Publish(uint8_t* p_topic, uint16_t length, uint8_t* p_message)
{
    /* Omission */

    /* Set AT command to ATC list */
    gs_atc_list[0].atcommand_size = (uint16_t)snprintf((char*)gs_atc_list[0].atcommand,
    LTE_ATC_STR_SIZE, "AT+SQNSMQTTPUBLISH=%s,\"%s\",%s,%d\r", "0",p_topic,"0",length);
    gs_atc_list[0].comp_msg_size = (uint16_t)snprintf((char*)gs_atc_list[0].comp_msg,
    LTE_ATC_STR_SIZE, "%s", "OK");
    gs_atc_list[0].data_exist_flag = 1;

    gs_atc_list[0].data = p_message;
    gs_atc_list[0].data_size = length;

    if(gs_atc_list[0].atcommand_size > LTE_ATC_STR_SIZE)
    {
        ryz_lte_clear_atc_list();
        return LTE_ERR_DATASIZE_OVERFLOW;
    }
}

```

Add following to first of Transmit waiting list:
atcommand =
"AT+SQNSMQTTPUBLISH=0,"[p_topic]"0,[length]
comp_msg = "OK"
data exist flag = 1

Set the address of the data you want to send in data

Check the Data Size to register the argument in the transmit waiting list

Figure 4-5 Register AT command of R_LTE_MQTT_Publish (r_lte_ryz.c)

Sending the first AT command

Send the AT command from the beginning of the registered transmit waiting list. Subsequent transmission of AT commands is done in R_LTE_Execute function corresponding to the response.

```

e_lte_err_t R_LTE_OM_Config(uint8_t* p_pdp_type, uint8_t* p_pdp_apn, uint8_t*
p_bandlist)
{
    /* Omission */

    /* Send first AT command from ATC list */
    ryz_lte_transmit_atc_list(LTE_TRANSMIT_ATCOMMAND);

    return LTE_SUCCESS;
}

```

Send first AT command registered in transmit waiting list

Figure 4-6 Sending the first AT command of R_LTE_OM_Config (r_lte_ryz.c)

4.2.1 AT command API with Data receive operation

To implement the AT command API that arbitrarily receives data after sending an AT command like the `R_LTE_MQTT_RcvMessage` function implemented in this sample application, it is necessary to rewrite the global variables in the framework.

When receiving data, it is necessary to change the global variables "gs_ryz_lte_receive_size" and "gs_ryz_lte_receive_flag". Set the size of the data you want to receive to "gs_ryz_lte_receive_size" and the macro "LTE_RCV_DATA_FLAG_ON" for "gs_ryz_lte_receive_flag".

```
e_lte_err_t R_LTE_MQTT_RcvMessage(uint8_t* p_topic, uint8_t message_id, uint16_t
message_size)
{
    /* Omission */

    /* Set receive flag and size for data receive operation */
    gs_ryz_lte_receive_size = message_size;
    gs_ryz_lte_receive_flag = LTE_RCV_DATA_FLAG_ON;

    /* Omission */
```



Figure 4-7 Global value setting of R_LTE_MQTT_RcvMessage (r_lte_ryz.c)

The data received with this AT command send notifies the application by callback function. The callback function is called when the "OK" response sent from RYZ024A is received after the data.

Note: "\r" or "\n" in the received data is converted to "\r\n" in RYZ024A and then transmitted to host MCU. As a result, some of the content and size of the received data may change.

4.3 Guideline of error handling

In a communication control system, it is necessary to develop an application assuming that various errors occur in the control of the communication controller and network operation. The following is a guideline for application development using this AT command Management Framework for detection and processing. In practice, the processing will vary depending on the requirements for the application product, so please handle it as reference information.

See also the "Connection Manager" description in the "RYZ024 Module System Integration Guide" (R19AN0101).

UART communication and RYZ024A behavior error

Defect status	Framework behavior	Application response
RYZ024A is restarted unintentionally	Receives URC "+SYSSTART". Since an unintended "+SYSSTART" is received, call callback function to notify application.	The callback function is called in event "LTE_EVENT_FATAL_ERROR". It is recommended to initialize using R_LTE_Init function for this event.
UART communication from the RYZ024A to the host MCU results in bit errors or character reception errors	If the character string does not match the string specified in the comp_msg, or if the string does not end with "\n", a timeout occurs and calls callback function to notify application.	The callback function is called in event "LTE_EVENT_TIMEOUT_ERROR". It is recommended to initialize using R_LTE_Init function for this event.
	If the received string matches the URC specified in the comp_msg in front, the application is notified by the callback function.	The callback function is called in event "LTE_EVENT_RCVURC". check the data registered in p_data because received string data is registered.
UART communication from the host MCU to the RYZ024A results in bit errors or character reception errors	If there is no response to the sent string, a timeout occurs and calls callback function to notify application.	The callback function is called in event "LTE_EVENT_TIMEOUT_ERROR". It is recommended to initialize using R_LTE_Init function for this event.
	If some of the AT commands sent are incorrect, an error response is received. After receiving the error response, notify the application with a callback function.	The callback function is called in event "LTE_EVENT_ERROR". Since the error code "LTE_CME_ERR_OPERATION_NOT_SUPPORTED" (0x04) is notified in the p_data, the corresponding processing needs to be added.
The MCU transmission and the transmission timing of the RYZ024A overlap, and the RYZ024A does not perform the expected operation	A timeout occurs when the operation stops. Framework calls callback function to notify application.	The callback function is called in event "LTE_EVENT_TIMEOUT_ERROR". It is recommended to initialize using R_LTE_Init function for this event.

CTS from RYZ024A does not enable for a long time	The response cannot be received from the RYZ024A for a long time, and a timeout occurs. Framework calls callback function to notify application.	The callback function is called in event "LTE_EVENT_TIMEOUT_ERROR". It is recommended to initialize using R_LTE_Init function for this event.
--	--	---

Network communication error

Defect status	Framework behavior	Application response
The network is disconnected due to deterioration of radio wave conditions, signal strength, etc.	Receive a "+CEREG" URC. The application is notified by a callback function.	The callback function is called in event "LTE_EVENT_RCVURC". Check the parameters of the URC and execute the corresponding processing. Check the AT command manual for URC parameters.
RYZ024A tried to connect to the network but could not connect due to an error such as incorrect Access Point Name.	Receive a "+CEREG" URC. The application is notified by a callback function.	The callback function is called in event "LTE_EVENT_RCVURC". Check the parameters of the URC and execute the corresponding processing. Check the AT command manual for URC parameters.
Otherwise, the connection is severed for some reason.	Receive a "+SQNSH" URC. The application is notified by a callback function.	The callback function is called in event "LTE_EVENT_RCVURC". Check the parameters of the URC and execute the corresponding processing. Check the AT command manual for URC parameters.

The communication status is notified by URC "+CEREG" etc. An example of URC of communication status notified from RYZ024A is explained below.

See also the description of "Connection Manager" in the "RYZ024 Module System Integration Guide" (R19AN0101).

- Received URC "+CEREG: 80" or "+CEREG: 4":
 - A URC that is notified when you are temporarily disconnected from the network. Since RYZ024A is trying to connect to the network again, if the radio wave condition improves, RYZ024A can reconnect to the network without executing the AT command API. At this time, MQTT communication is maintained in the RYZ024A, so MQTT communication can be resumed without executing R_LTE_MQTT_Connect function when reconnecting to the network.
- Received URC "+CEREG: 0":
 - A URC to be notified when disconnected from the network. If the radio wave conditions improve, the connection will be automatically reconnected. In the upper layer, for example, when the TCP socket is disconnected, a URC such as +SQNSH is notified, so processing such as TCP connection is required as necessary.
- Received URC "+SQNSMQTTTONCONNECT: 0,-7":
 - This is a URC that is notified when MQTT communication is also disconnected after a certain period after being disconnected from the network. Reconnecting to the network does not preserve MQTT communication, so you must execute the R_LTE_MQTT_Connect function again.

4.4 PMOD-RYZ024A specific processing

When the RYZ024A enters the deep sleep state, the UART CTS signal becomes Hi-Z. In a normal circuit, by pulling up the CTS signal and making it high level, when the RYZ024A is in the deep sleep state, it can be controlled by HW flow control so that AT commands cannot be sent from the RL78/G22.

However, in the PMOD-RYZ024A, due to the characteristics of the level shifter used, the CTS signal from the level shifter to the RL78/G22 remains low even when the RYZ024A enters deep sleep, making HW flow control impossible.

Therefore, in this sample application, when the RL78/G22 sends an AT command while the RYZ024A is in deep sleep, first send "AT+CFUN?" command to confirm that RYZ024A has woken up. When the response of "AT+CFUN?" command is not returned, it retries several times. Send the desired AT command (e.g. AT+SQNSMQTTPUBLISH) after receiving "OK". [Figure 4-8 Registration of "AT+CFUN?" command (r_lte_ryz.c)] shows how to add the "AT+CFUN?" command to the transmission waiting list.

[Figure 4-8 Registration of "AT+CFUN?" command (r_lte_ryz.c)] shows how to add the "AT+CFUN?" command to the transmission waiting list.

```
e_lte_err_t R_LTE_MQTT_Publish(uint8_t* p_topic, uint16_t length, uint8_t* p_message)
{
    /* Omission */

    /* Set AT command to ATC list */
    gs_atc_list[0].atcommand_size = (uint16_t)snprintf((char*)gs_atc_list[0].atcommand,
    LTE_ATC_STR_SIZE, "AT+CFUN?\r");
    gs_atc_list[0].comp_msg_size = (uint16_t)snprintf((char *)gs_atc_list[0].comp_msg,
    LTE_ATC_STR_SIZE, "OK");
    gs_atc_list[0].data_exist_flag = 1;
    gs_atc_list[0].at_polling_flag = 1;

    gs_atc_list[1].atcommand_size = (uint16_t)snprintf((char*)gs_atc_list[1].atcommand,
    LTE_ATC_STR_SIZE, "AT+SQNSMQTTPUBLISH=%s,\"%s\", \"%s\",%d\r", "0",p_topic,"0",length);
    gs_atc_list[1].comp_msg_size = (uint16_t)snprintf((char*)gs_atc_list[1].comp_msg,
    LTE_ATC_STR_SIZE, "%s", "OK");
    gs_atc_list[1].data_exist_flag = 1;
    gs_atc_list[1].data = p_message;
    gs_atc_list[1].data_size = length;

    if(gs_atc_list[1].atcommand_size > LTE_ATC_STR_SIZE)
    {
        ryz_lte_clear_atc_list();
        return LTE_ERR_DATASIZE_OVERFLOW;
    }

    /* Omission */
}
```

Add following to first of Transmit waiting list.

atcommand = "AT+CFUN?"
comp_msg = "OK"
data_exist_flag = 1
at_polling_flag=1

Add "AT+SQNSMQTTPUBLISH" to
2nd of Transmit waiting list.

Figure 4-8 Registration of "AT+CFUN?" command (r_lte_ryz.c)

By using an appropriate level shifter, even if the RYZ024A enters a deep sleep state, this processing is not necessary for boards that can perform HW flow control with the CTS signal.

Definitions for enabling or disabling PMOD-RYZ024A specific processing are shown in [Table 4.1 Definition of PMOD-RYZ024A specific processing (r_lte_ryz.c)].

Table 4-1 Definition of PMOD-RYZ024A specific processing (r_lte_ryz.c)

Name	Default value	Description
PMOD_RYZ024A	1	1: Enable PMOD-RYZ024A specific processing to send AT+CFUN? command. 0: Disable PMOD-RYZ024A specific processing.

Definitions for setting the operation of the "AT+CFUN?" command are shown in [Table 4-2 Operation setting of AT+CFUN? command (r_lte_ryz.c)].

Table 4-2 Operation setting of AT+CFUN? command (r_lte_ryz.c)

Name	Default value	Description
AT_POLLING_TIMETOUT	2	Timeout count for "AT+CFUN?" command. Unit: 2sec e.g.) 2sec * 2 = 4sec
AT_POLLING_RETRY_COUNT	2	Number of retries for "AT+CFUN?" command.

4.5 Initializing PMOD-RYZ024A

If you have been using the PMOD-RYZ024A before running the sample application in this application note, your settings may be saved in the RYZ024A's non-volatile memory. It can be initialized by executing the following AT command.

```
AT+CGDCONT=1,"IP","ppsim.jp"
OK
AT+CGDCONT?
+CGDCONT: 1,"IP","ppsim.jp",,,,0,0,0,0,0,0,0,0
OK
AT+SQNSFACTORYRESET
ERROR
AT^RESET
OK
+SHUTDOWN
+SYSSTART
OK
AT+CGDCONT?
+CGDCONT: 1,"IPV4V6","",,,,0,0,0,0,0,0,1,,0
OK
```

Set later to ensure that it has been initialized.

The response may not be ERROR.

Make sure the settings are initialized and show the default values.

Figure 4-9 Initializing PMOD-RYZ024A

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug. 30.23	-	1 st edition
1.10	Oct. 30.23	13	Changed API name for executing low power consumption mode (5 of 2.1 Application environment)
1.20	Feb. 09.24	7 8 45	Changed port settings and RTS setting conditions Updated 1.2.6 Code Size Updated 2.1 Application environment, Table 2-2 Software environment of sample application Changed 3.4 User Specific Configuration Table 3-4 Smart Configurator of RL78/G22 Changed of Config_INTC, Config_PORT (Changed Sample program(ryz_lte_rts_high_devspe))

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

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

1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity.

Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

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

3. Input of signal during power-off state

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

4. Handling of unused pins

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

5. Clock signals

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

6. Voltage application waveform at input pin

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

7. Prohibition of access to reserved addresses

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

8. Differences between products

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

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

Contact information

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

www.renesas.com/contact/.