RYZ024A and RA MCU

LTE Communication Sample Application

Introduction

RYZ024A is a cellular module capable of LTE Cat M1/NB1/NB2 communication. RYZ024A connects to the host MCU via UART communication, and its operation can be controlled by AT commands. This sample application uses RA MCU as a host MCU and provides a program to control RYZ024A and perform MQTT communication and implement power-saving features of the cellular network (eDRX, PSM). This sample application uses the AT command management framework to implement a program that sends AT commands from the host MCU 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 RYZ024A PMOD board (RTKYZ024A0B00000BE). Refer to "RA6M5 Group RYZ024A PMOD LTE Connectivity with RA6M5 MCU Quick Start Guide" (R21QS0007) to activate the SIM card.

Target Device

RYZ024A

EK-RA6M5

Related Documents

- RYZ024 Module System Integration Guide (R19AN0101)
- RYZ024 Modules AT Command User's Manual(R11UZ0110)
- RA6M5 Group User's Manual: Hardware (R01UH0891)
- RA6M5 Group Evaluation Kit for RA6M5 Microcontroller Group EK-RA6M5 v1 User's Manual (R20UT4829)
- Renesas Flexible Software Package (FSP) User's Manual (R11UM0155)
- RA6M5 Group RYZ024A PMOD LTE Connectivity with RA6M5 MCU Quick Start Guide (R21QS0007)
- EK-RA6M5 v1 – User's Manual (R20UT4829)

Pmod^{TM} is registered to Digilent Inc.

Contents

1. Overview

1.1 Overview of Sample Framework

The RYZ024A is a cellular module with CATM1 technology support. The modem can be controlled by AT commands via the UART.
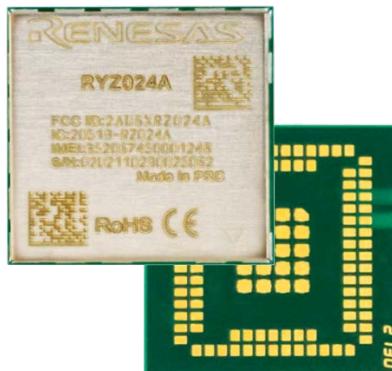

Figure 1. RYZ024A

In this sample application, AT command framework software controls LTE Cat M1 communication of RYZ024A using the RA MCU as the host MCU. The RA MCU 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 RA MCU utilizes the LTE Cat M1 communication function of the RYZ024A.
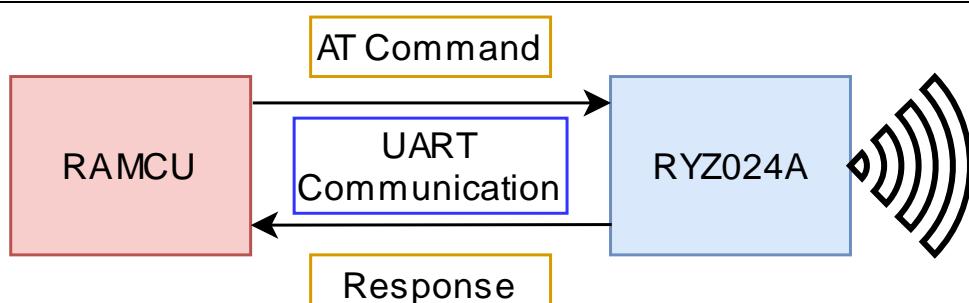

Figure 2. Communication of RYZ024A and Host MCU

Note: Regarding the use of PMOD Expansion Board for RYZ024A (RTKYZ024A0B00000BE):

When RYZ024A shifts to Deep Sleep mode, the CTS pin becomes Hi-Z. However, in the PMOD Expansion Board for RYZ024A (hereafter PMOD-RYZ024A), the CTS signal from the level shifter to the host microcomputer remains at a 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 section 4.4, PMOD-RYZ024A Specific Processing.

1.2 Software Configuration

This sample application provides two types of programs: baremetal program and FreeRTOS Kernel (hereafter FreeRTOS) program.

Table 1. Application Note Contents

R01AN6579xxxxrrrr-RYZ024A-ra-lte-sample.pdf **xx: Language classification, Creation country** **rrrr: Revision number**	This document
sample_RYZ024A_ra6m5	Baremetal sample program
sample_RYZ024A_ra6m5_rtos	FreeRTOS sample program

The file structure of the sample program provided in this application note is as follows.

Table 2. Sample Program File Structure

Folder structure			Description	
• sample_ryz024a_ra6m5			Sample project for baremetal	
• sample_ryz024a_ra6m5_rtos			Sample project for FreeRTOS	
	\		Project files for GCC RA configurator files	
	.settings\		e² studio setting files	
	script\		Linker setting files	
	src\	hal_entry.c	Main program of sample application	
		lte_task_entry.c	Main program of LTE communication task • FreeRTOS only	
		mqtt_app_task_entry.c	Main program of MQTT communication application task • FreeRTOS only	
		r_lte_ryz.c r_lte_ryz.h r_lte_user_config.h	AT command management framework program	
		SEGGER_RTT\	SEGGER J-Link RTT Viewer source codes	

Software configuration of this sample framework is shown in the following figure.
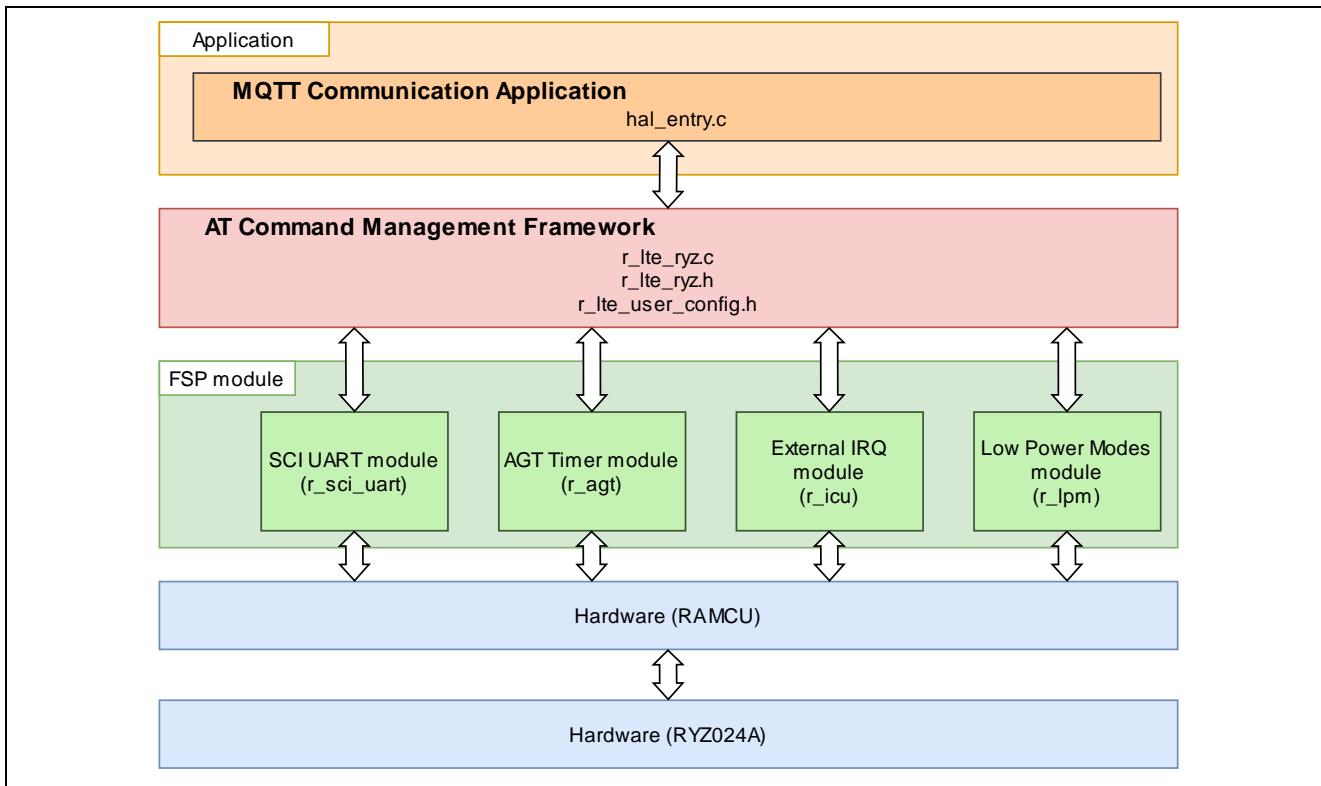
Figure 3. Software Configuration

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 a button on the EK-RA6M5 board 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 section 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 EK-RA6M5 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 section 3, AT Command Management Framework.

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 3. Hardware Environment

PMOD Expansion Board for RYZ024A	RYZ024A module (RTKYZ024A0B00000BE)
EK-RA6M5	Host RA MCU (RTKYZ014A0B00000BE)
Windows® PC	RA MCU's application development environment and debug console for operation conformation

The developed application works with the following software environment.

Table 4. Software Environment

e² studio	2022-10	Renesas IDE (https://www.renesas.com/e2studio)
FSP	4.2.0	Driver that can be used with RA MCU (http://www.renesas.com/fsp)
SEGGER RTT Viewer	7.66	Debug write display viewer (https://www.segger.com/products/debug-probes/j-link/tools/rtt-viewer/)

**Table 5.   EK-RA6M5 Peripheral Functions Used in this Sample Application**

Function Name		Description
Serial Communication Interface	SCI0 (UART0)	UART communication with RYZ024A Baudrate : 115200 bps Data length : 8 bits Parity : none Stop bit : 1 bit Flow control : Hardware CTS/Software RTS
Low Power Asynchronous General Purpose Timer	AGT0	Timeout for AT command communication
I/O Port	P404	Reset pin control of RYZ024A (Reset at low level)
	P412	RTS signal used for UART communication with RYZ024A (Connect to RTS0 of RYZ024A)
	P413	CTS signal used for UART communication with RYZ024A (Connect to CTS0 of RYZ024A)
	P400	Ring signal pin of RYZ024A
External Interrupt	IRQ10	External interrupt of user button S1
	IRQ9	External interrupt of user button S2
	IRQ0	External interrupt of RYZ024A RING signal

For application operation, follow these steps:
1. Connect EK-RA6M5 and RYZ024A with PMOD connector. Please use PMOD2 (J25) connector for EK-RA6M5.


Connects EK-RA6M5 with PMOD2 (J25)
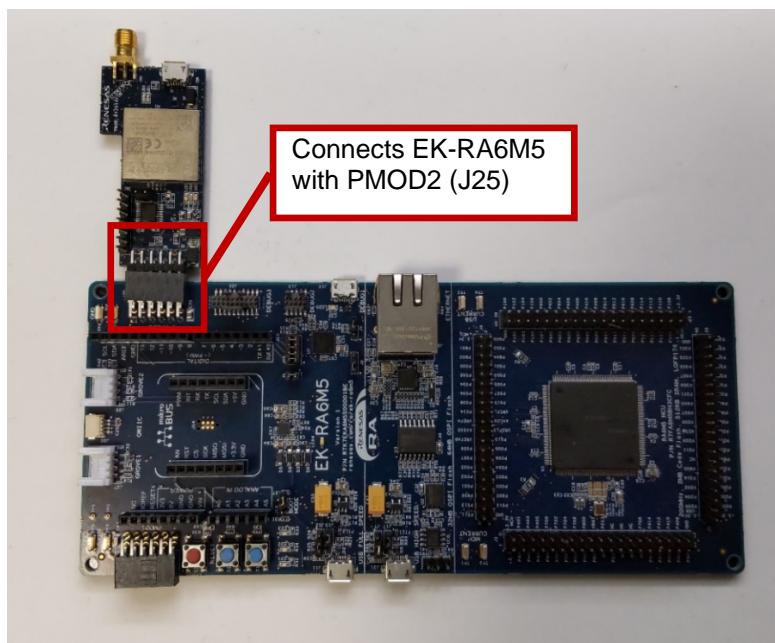
Figure 4.   Connect EK-RA6M5 and RYZ024A

2.   Connect USB cables to EK-RA6M5 and RYZ024A. Also connect the antenna to RYZ024A.
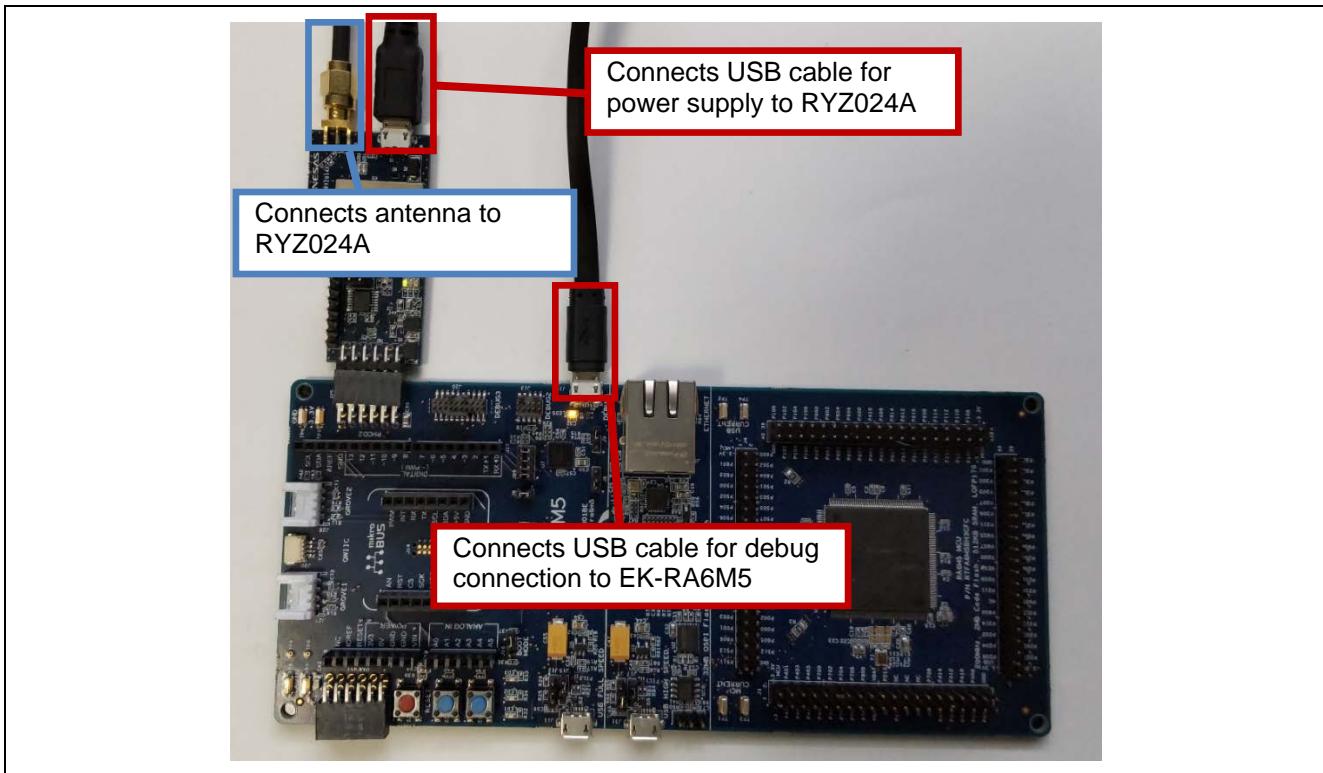
Figure 5.   Connect USB Cable and Antenna

3.   Import sample project to e² studio.
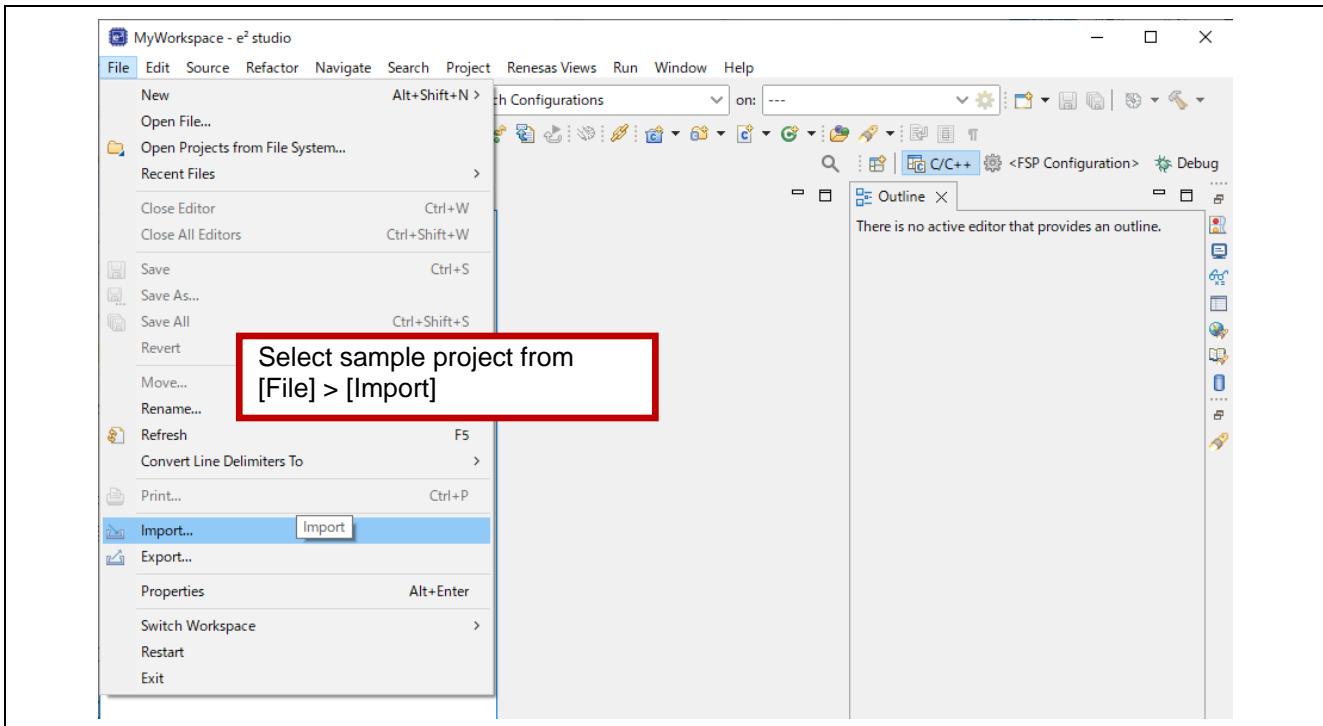
Figure 6.   Add Sample Project

4. Generate the necessary files for sample project.
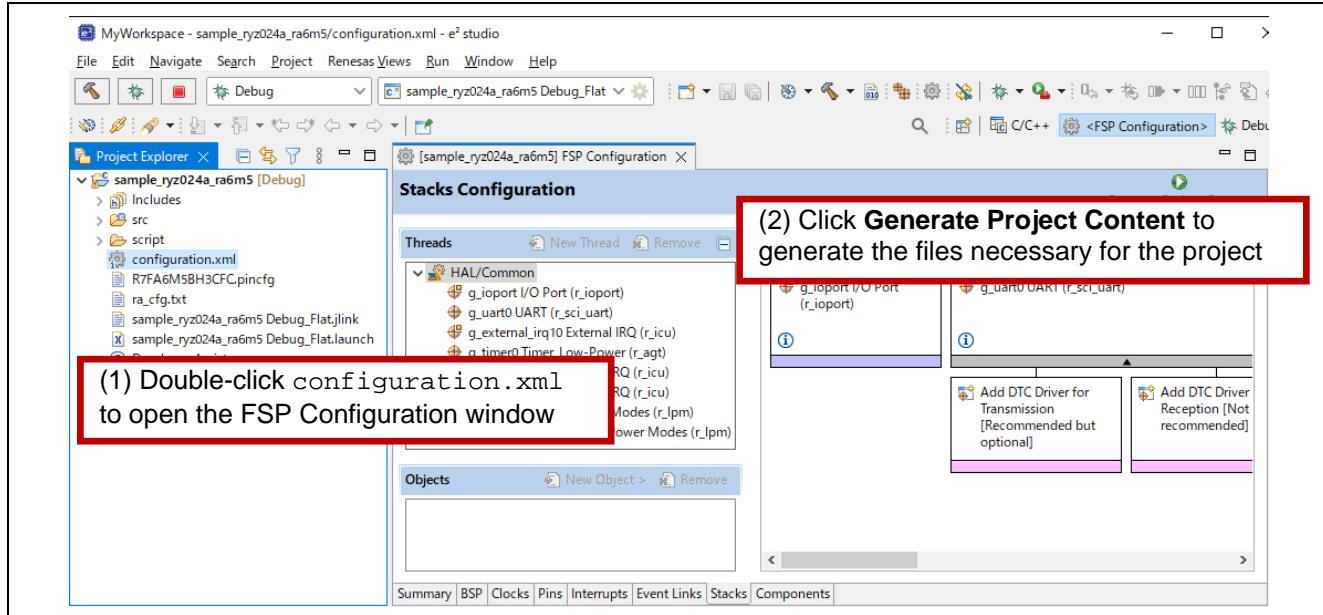
Figure 7.   Generate the Necessary Files for Sample Project

5. 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 following files:

   Baremetal program: `hal_entry.c`

   FreeRTOS program: `lte_task_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, use the default as shown above. Bear in mind that the connection may take longer.

   In this application, operation is confirmed using the following APN and LTE bands.
   — APN   : iot.truphone.com
   — Authentication protocol   : 0
   — Username   : -
   — Password   : -
   — LTE bands   : 1,2,3,4,5,8,12,13,17,18,19,20,25,26,28,66

```
31    ⊖/***********************************************************************************//**
32      * @addtogroup sample_ryz024a_ra6m5
33      * @{
34      ************************************************************************************/
35
36    FSP_CPP_HEADER
37    void R_BSP_WarmStart(bsp_warm_start_event_t event);
38    FSP_CPP_FOOTER
39
40    /* Application value definition */
41    void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t * p_data);
42    static uint8_t gs_sw1_count = 0;
43    static uint8_t gs_sw1_push_flag = 0;
44    static uint8_t gs_sw2_push_flag = 0;
45    static uint8_t gs_reinitialize_flag = 0;
46    static uint8_t gs_mqtt_conn_state = 0;
47
48    /* Application specific string data for network connection */
49    static uint8_t gs_str_PDP_type[] =              "IPV4V6";
50
51   #if 1
52    /* Access Point Name using in network connection. Please select depending on SIM card */
53    static uint8_t gs_str_PDP_APN[] =               "iot.truphone.com";
54
55    /* Authentication protocol, user id, password. Please modify depending on using LTE band */
56    static uint8_t gs_str_PDP_protocol[] =          "0";
57    static uint8_t gs_str_PDP_userid[] =            "";
58    static uint8_t gs_str_PDP_password[] =          "";
59    #endif
60   ⊖#if 0
61    /* Access Point Name using in network connection. Please select depending on SIM card */
62    static uint8_t gs_str_PDP_APN[] =               "soracom.io";
63
64    /* Authentication protocol, user id, password. Please modify depending on using LTE band */
65    static uint8_t gs_str_PDP_protocol[] =          "2";
66    static uint8_t gs_str_PDP_userid[] =            "sora";
67    static uint8_t gs_str_PDP_password[] =          "sora";
68    #endif
69
70    /* Band List for network connection. Please select depending on u
71   ⊖#if 0
72    static uint8_t gs_str_LTE_bandlist[] =          "1,19";
73    #endif
74   ⊖#if 0
75    static uint8_t gs_str_LTE_bandlist[] =          "2,4,12";           /* AT&T bands in US Region */
76    #endif
77   ⊖#if 1
78    static uint8_t gs_str_LTE_bandlist[] =          "1,2,3,4,5,8,12,13,17,18,19,20,25,26,28,66"; /* All selectable bands */
79    #endif
```

Change APN, protocol, username, and password

Change LTE Bands

Figure 8.   Change APN, Authentication protocol, Username, Password and LTE bands (hal_entry.c or lte_task_entry.c)

6. Disable low power operation of the host MCU. When using the SEGGER J-Link RTT Viewer that monitors the operation of this sample program, comment out the API (R_LPM_LowPowerModeEnter) that executes the low power consumption mode because it is not recommended to use the low power consumption mode of the host MCU.
Comment out the line calling R_LPM_LowPowerModeEnter() in the function shown below.

Baremetal program: file name/function name: `r_lte_ryz.c` / void R_LTE_Execute(void)

```
853
854              /* enter Low Power Mode */
855              R_LPM_LowPowerModeEnter(&gs_lpm_ctrl_instance_ctrls[lpm_mode]);
856      }
```

FreeRTOS program: file name/function name: `r_lte_ryz.c` / void vApplicationIdleHook(void)

```
823
824              /* enter Low Power Mode */
825              R_LPM_LowPowerModeEnter(&gs_lpm_ctrl_instance_ctrls[lpm_mode]);
826      }
```

7. Build sample project. In this sample project, user can monitor the execution status flow using SEGGER J-Link RTT Viewer. For this setting, please check address of ".bss._SEGGER_RTT" from .map file in Debug folder which is generated after build.
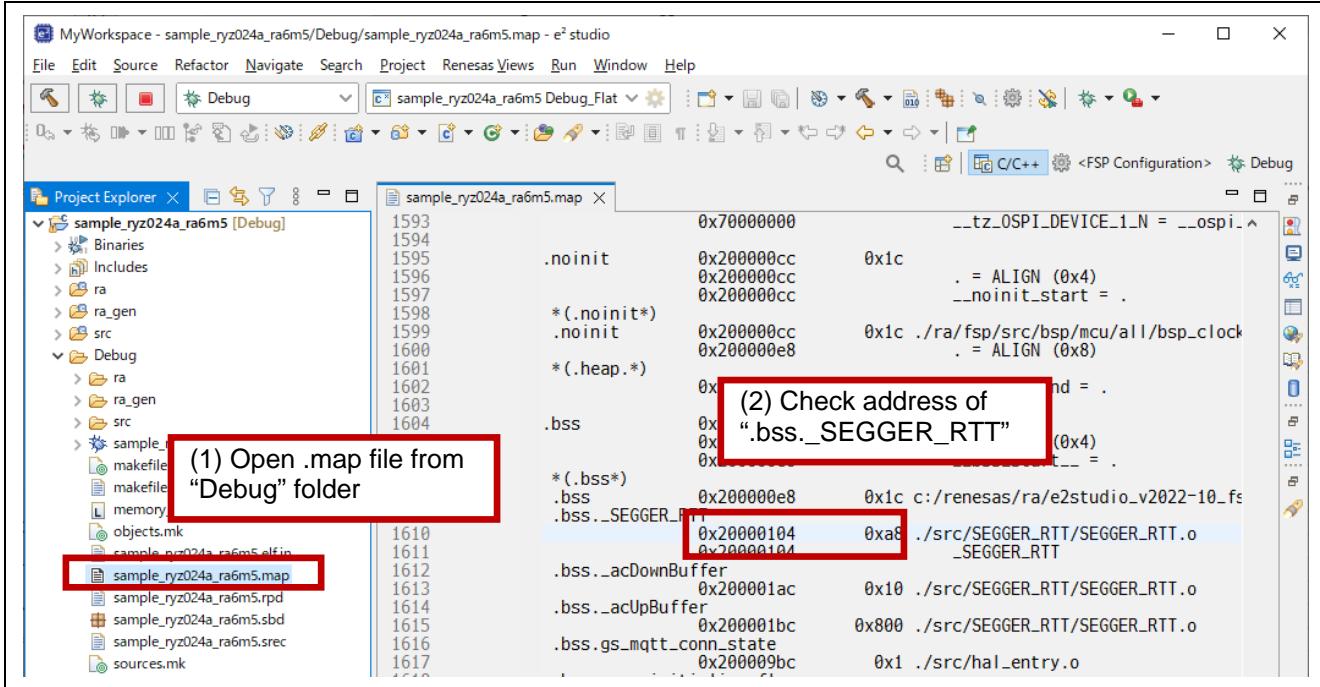

Figure 9.   Check .map file

8. Start the debugging first and then run the RTT viewer and connect to the EK-RA56M5. Enter address of "_SEGGER_RTT" for connection.
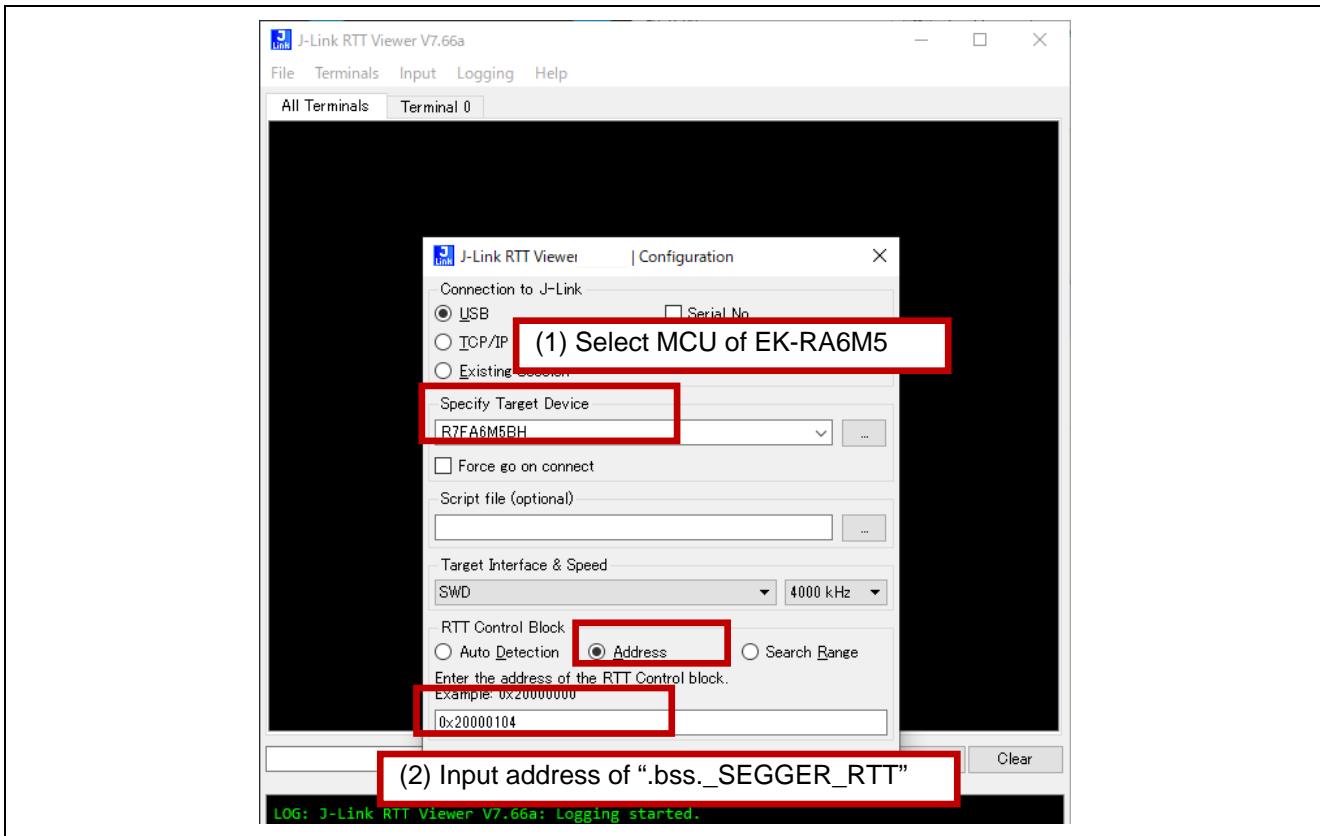

Figure 10.   Start RTT Viewer

2.2 Application Operation

This sample program uses the public MQTT server "test.mosquitto.org" to send and receive messages. Using the MQTT communication control commands of RYZ024A, the sample program connect to the MQTT server, publish MQTT data, receive MQTT data, and display the results on the RTT Viewer. The flow of program operation is explained below.
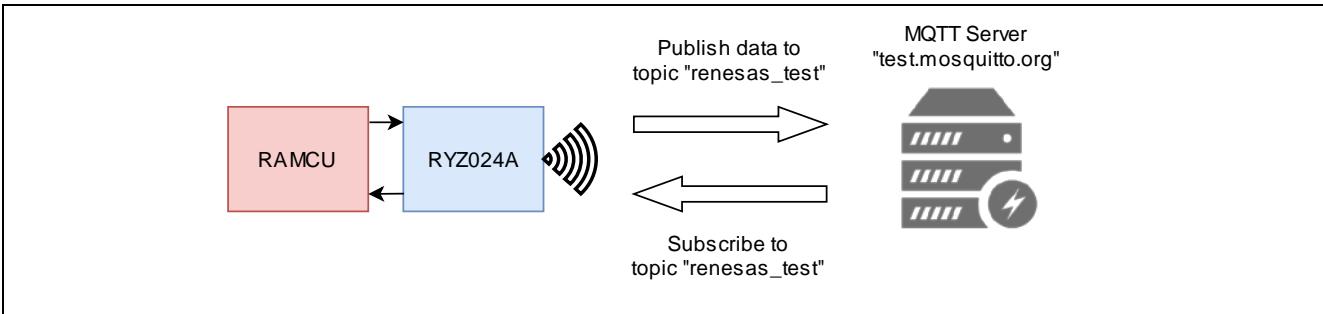
Figure 11. Sample Application System Configuration

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 RTT Viewer. In this state, the user can operate the push button switches on the board.
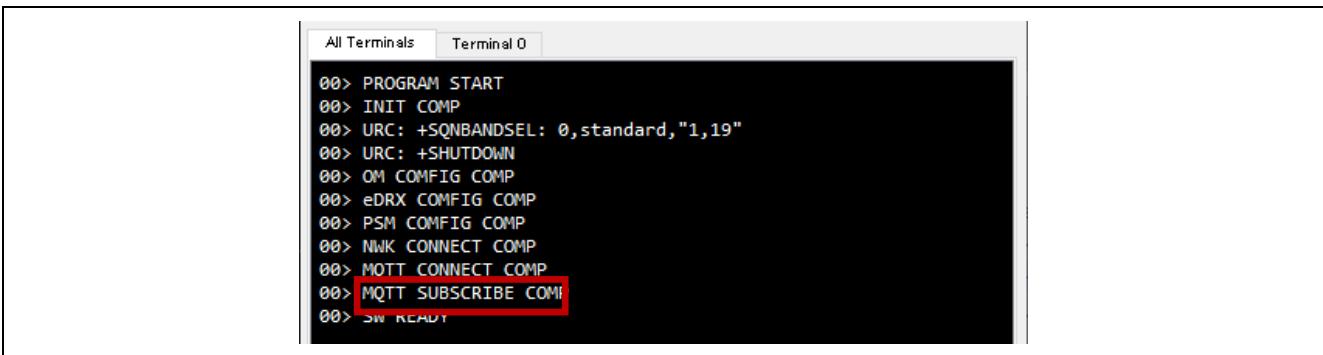
Figure 12. Connects MQTT Server

After "SW READY" is displayed, press the push button switch SW1 to send a message to the MQTT server by publish request. Since the subscribe requests are made from the host to the MQTT server, when the server sends message, it can be seen on the host as subscribed messages. The Application sends message receive request using this subscribed message and displays the received messages in RTT viewer. Transmitting string data changes depending on the number count of SW1 press.
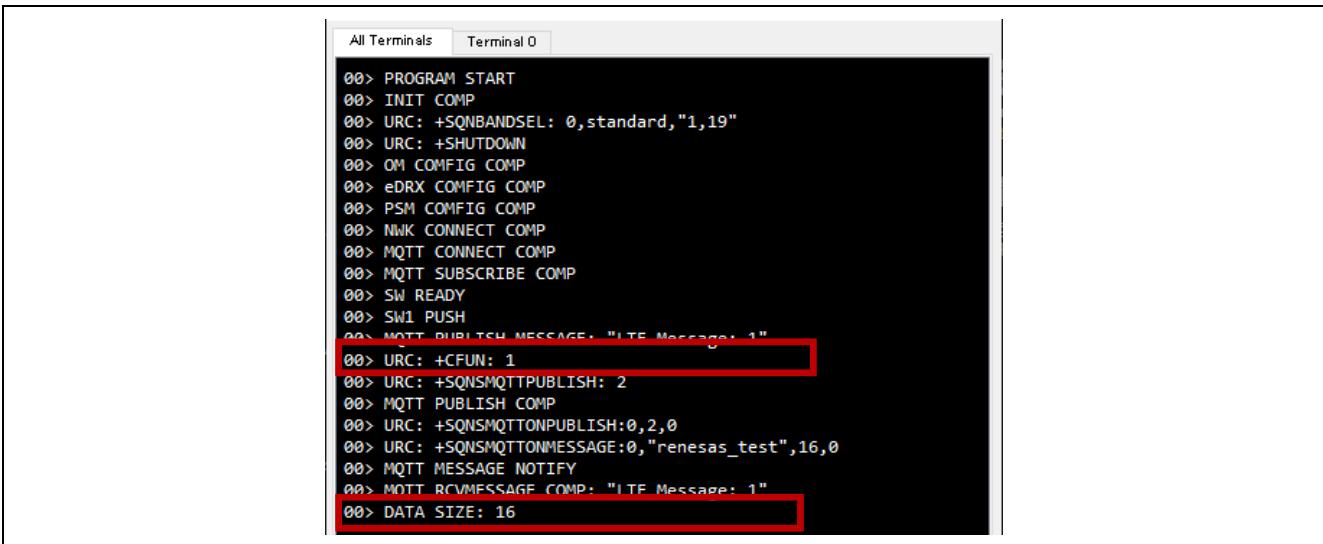
Figure 13. Press SW1

Press push button switch SW2 to disconnect both from MQTT server and network.
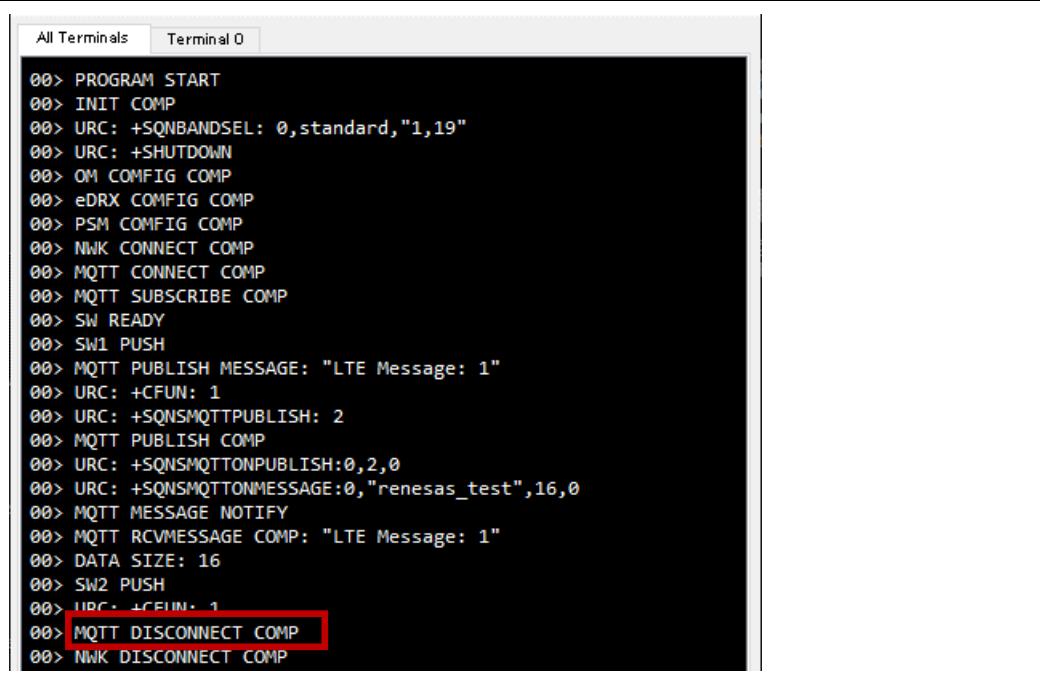
Figure 14.   Press SW2

If you are disconnected from the network or MQTT server due to the network conditions or the connection signal strength, or so forth, 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.

## 3.   AT Command Management Framework

3.1   Framework Overview

The RYZ024A is operated from the host MCU through the transmission of AT commands and reception of responses using serial communication through the MCU 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 implemented using the SCI UART module, AGT timer module, External IRQ module and Low Power Modes module from FSP module.

- The SCI UART module is used to send AT commands to the RYZ024A and receive responses from the RYZ024A.
- The AGT timer module is used to measure timeout condition after AT command is executed.
- The External IRQ module is used for the RING signal interrupt that notifies that there is a Unsolicited Result Code (URC) from the RYZ024A.
- The Low Power Modes module is used when the host MCU is in the IDLE state or when it is waiting for a response after sending an AT command in the AT command API.

This chapter description is mainly based on the bare metal version of the AT command management framework. For the framework included in the FreeRTOS sample program provided with this sample program, see [3.6 FreeRTOS Framework].
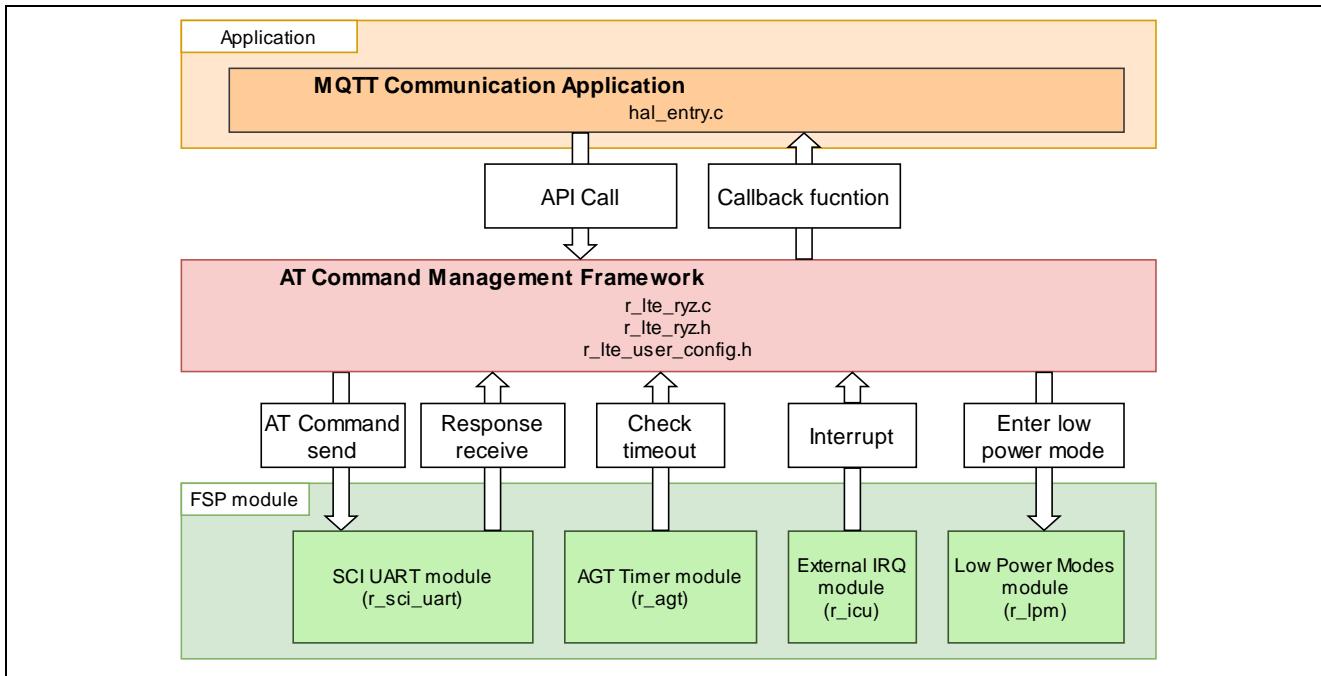
Figure 15.   Overview of Sample Framework

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.
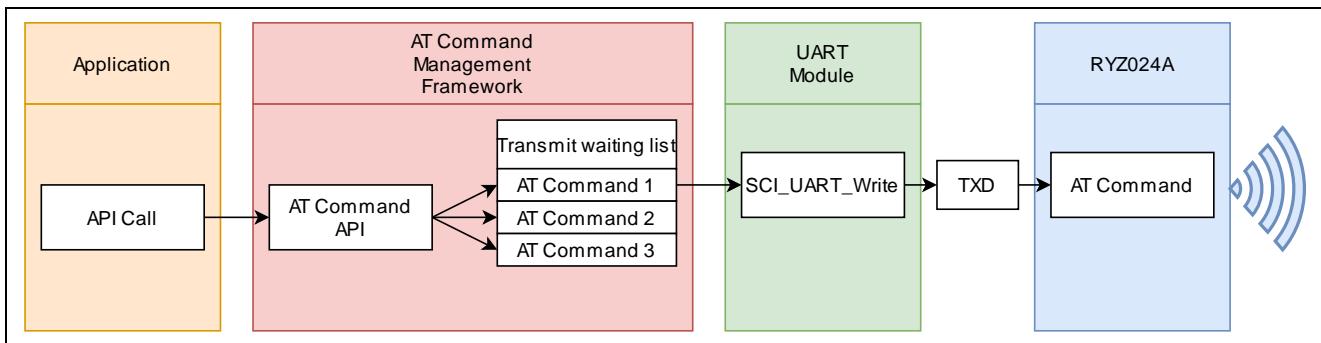

Figure 16.   AT Command API Call

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.
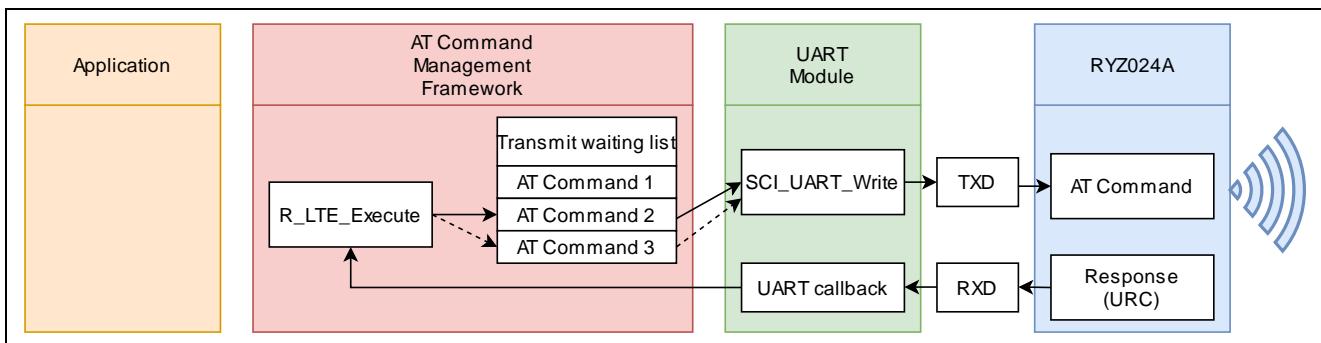
Figure 17.   Receive Response and AT Command Send

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.
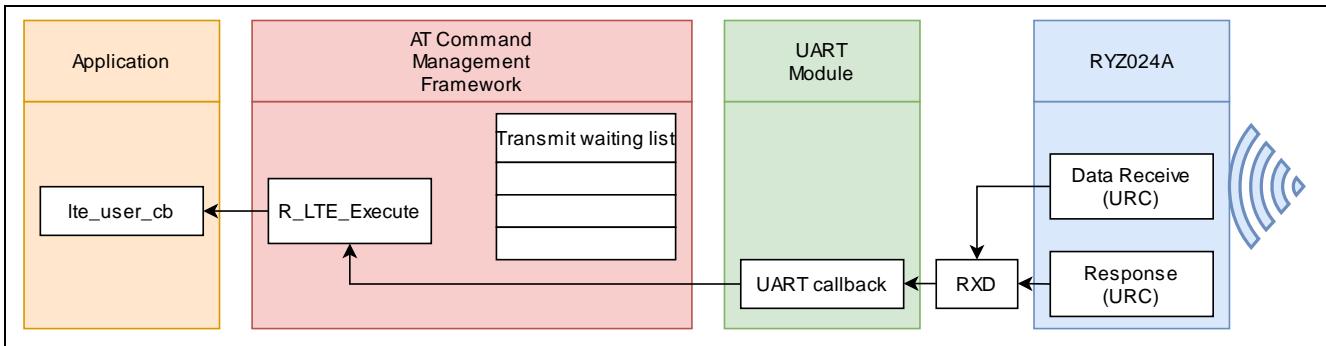
Figure 18.   Notification in Callback Function

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 RA 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).
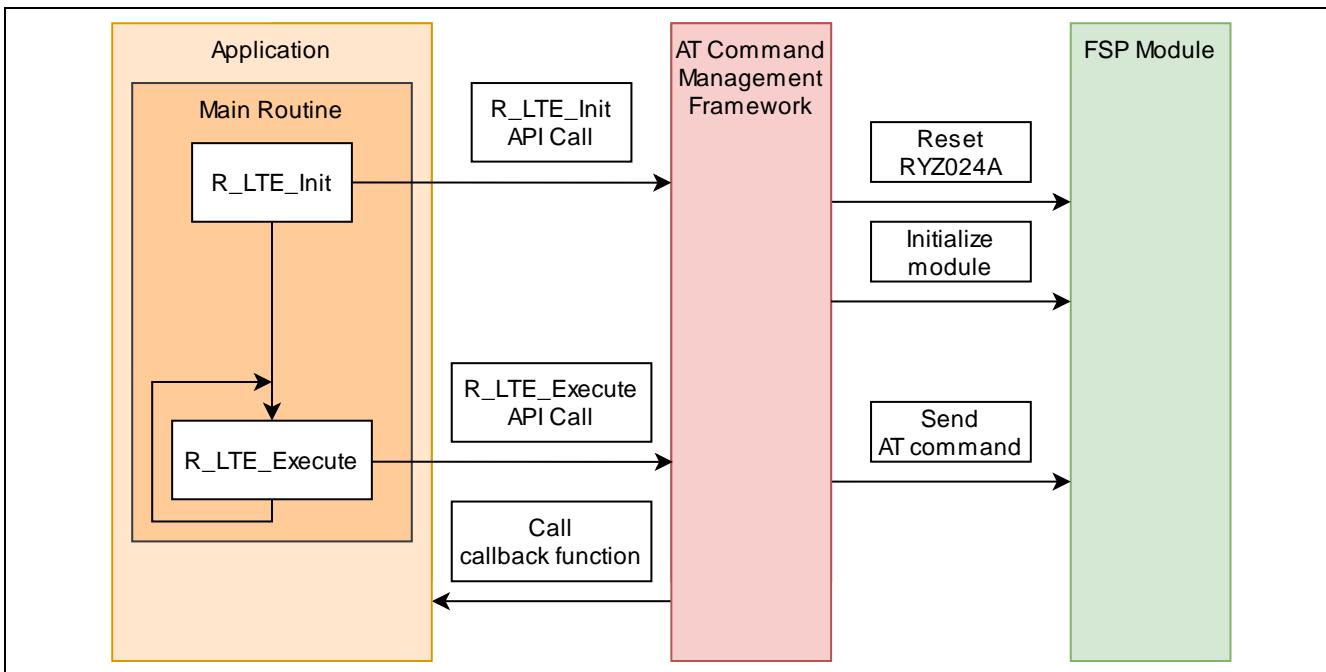
Figure 19.   Management API

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 FSP 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.
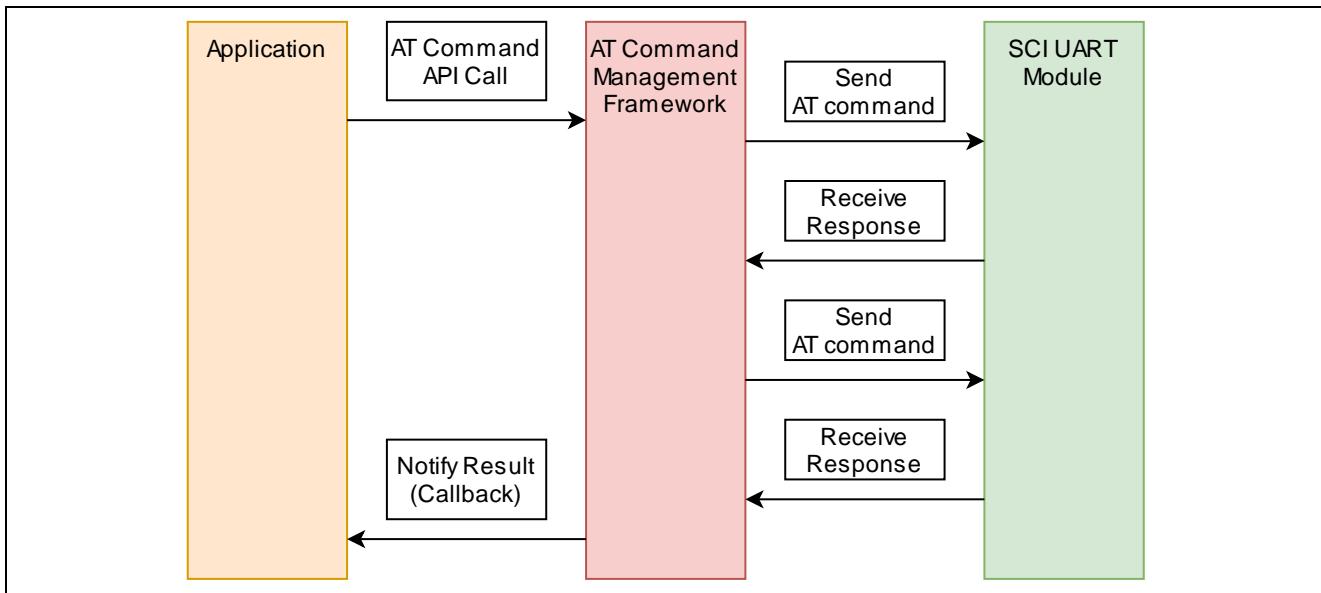

Figure 20. AT Command API

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: "iot.truphone.com"
	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 1. "AT+CEREG=5" 2. "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+SQNSMQTTCFG=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 **(1) AT+SQNSMQTTRCVMESSAGE=0,[p_topic]"** • message_id = [other than 0] **(2) AT+SQNSMQTTRCVMESSAGE=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)` Note: If user tried to receive message that does not exists or same message that have already received, an error 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: 1. "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: "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: "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: "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: "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 (0x003)	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 (0x003)	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 URC that is not a result of an AT command. The framework-based program of this sample program receives string data from RYZ024And 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.

Type name	lte_cb_t	
Argument	uint16_t event_type (In)	Notified event ID Refer IDs in Table 6. .
	uint16_t api_id (In)	ID identifying API which framework-based program is processing. Refer IDs in Table 7. .
	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 as follows.

Table 6. Event Type IDs and Value

Event Type	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 7.   API IDs and Value

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.

Baremetal program: `hal_entry.c`

FreeRTOS program: `mqtt_app_task_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.

```
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
*/
                SEGGER_RTT_printf(0, "OM COMFIG COMP (
                R_LTE_NWK_Connect(1);
            } break;

/* Omission */

            case LTE_API_MQTT_RCVMESSAGE:
            {
                /* Display received message */
                SEGGER_RTT_printf(0, "MQTT RCVMESSAGE COMP: %s\n",p_data);
                for(uint8_t i = 0; i < data_len; i++)
                {
                    SEGGER_RTT_printf(0, "%c", p_data[i]);
                }
            } break;
```

LTE_EVENT_API_COMPLETE event notification

Define which AT command API result with api_id

Received data is registered in p_data

Figure 21. 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 erro
        uint16_t err_code;
        LTE_ERROR_DECODE(&err_code, p_data);
        SEGGER_RTT_printf(0, "Error Response\n");
        SEGGER_RTT_printf(0, "API ID: %d, Error Code: %d\n", api_id, err_code);
    }

/* Omission */
```

LTE_EVENT_ERROR event notification

Analyze error code with LTE_ERROR_DECODE

Figure 22. 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 23. 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.

Notes: 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 loop */
        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;
        }
}
```

LTE_EVENT_FATAL_ERROR event notification

Initialization of framework base program

Figure 24. 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 */
        SEGGER_RTT_printf(0, "URC: %s",p_data);

        if(0 == memcmp(p_data, str_onmessage, (si
        {
            uint8_t rcv_id = 0;
            char * ptr;
            uint8_t msg_count = 0;

            /* Display subscribed notification */
            SEGGER_RTT_printf(0, "MQTT MESSAGE NOTIFY\n");

            /* 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);
        }
    }
/* Omission */
```

Callout: **LTE_EVENT_RCVURC event notification**

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

Callout: **Analyze parameter of URC**

Callout: **Call AT command API with analyzed parameter**

Figure 25.   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 RA MCU used. In the AT Command Management Framework, a program for setting these user-specific setting values is defined in "r_lte_user_config.h". 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 8 shows the setting items for the RA MCU pins connected to each pin of the RYZ024A. Please confirm when changing the board to be used as the host MCU.

Table 8.   Pin Function Setting for RYZ024A

RYZ024A_RESET_PIN	BSP_IO_PORT_04_PIN_04	Pin corresponding to the reset pin of the RYZ024A.
RYZ024A_RESET_ENABLE	BSP_IO_LEVEL_HIGH	Signal settings to enable the reset pin of the RYZ024A.
RYZ024A_RESET_DISABLE	BSP_IO_LEVEL_LOW	Signal settings to disable the reset pin of the RYZ024A.
RYZ_LTE_RTS0_PIN	BSP_IO_PORT_04_PIN_12	Pin corresponding to the RTS0 pin of the RYZ024A.
RYZ_LTE_CTS0_PIN	BSP_IO_PORT_04_PIN_13	Pin corresponding to the CTS0 pin of the RYZ024A.
RYZ_LTE_RING0_PIN	BSP_IO_PORT_04_PIN_00	Pin corresponding to the RING0 pin of the RYZ024A.

Table 9 shows the setting items for using the FSP module within the AT command management framework. Please check if you want to edit the RA Configurator, change the RA MCU and so forth.

Table 9.   FSP Module Setting for RYZ024A

RYZ024A_UART_CTRL	g_uart0.p_ctrl	SCI UART Module Control Structure. Used for UART communication between host MCU and RYZ024A. Channel : 0
RYZ024A_UART_CFG	g_uart0.p_cfg	SCI UART Module Configuration Structure.
RYZ024A_TIMER_CTRL	g_timer0.p_ctrl	AGT timer Module Control Structure. Used for AT command timeout. Channel : 0
RYZ024A_TIMER_CFG	g_timer0.p_cfg	AGT timer Module Configuration Structure.
RYZ_LTE_RING_CTRL	g_external_irq0.p_ctrl	External IRQ Module Control Structure. Used for RING signal interrupt from RYZ024A. Channel : 0 Pin : P400
RYZ_LTE_RING_CFG	g_external_irq0.p_cfg	External IRQ Module Configuration Structure.

Table 10 shows the size setting items for various data used within the AT command management framework. 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 10. Size Setting of AT Command Transmission Waiting List

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   FSP module Used in Framework

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

3.5.1   SCI UART Module

The AT Command Management Framework uses the SCI UART module to implement UART communication between the RYZ024A and the host MCU.

When sending AT commands from the host MCU to the RYZ024A, the write function (R_SCI_UART_Write) of the SCI UART 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 host MCU, the data is received using the callback function of the SCI UART 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.
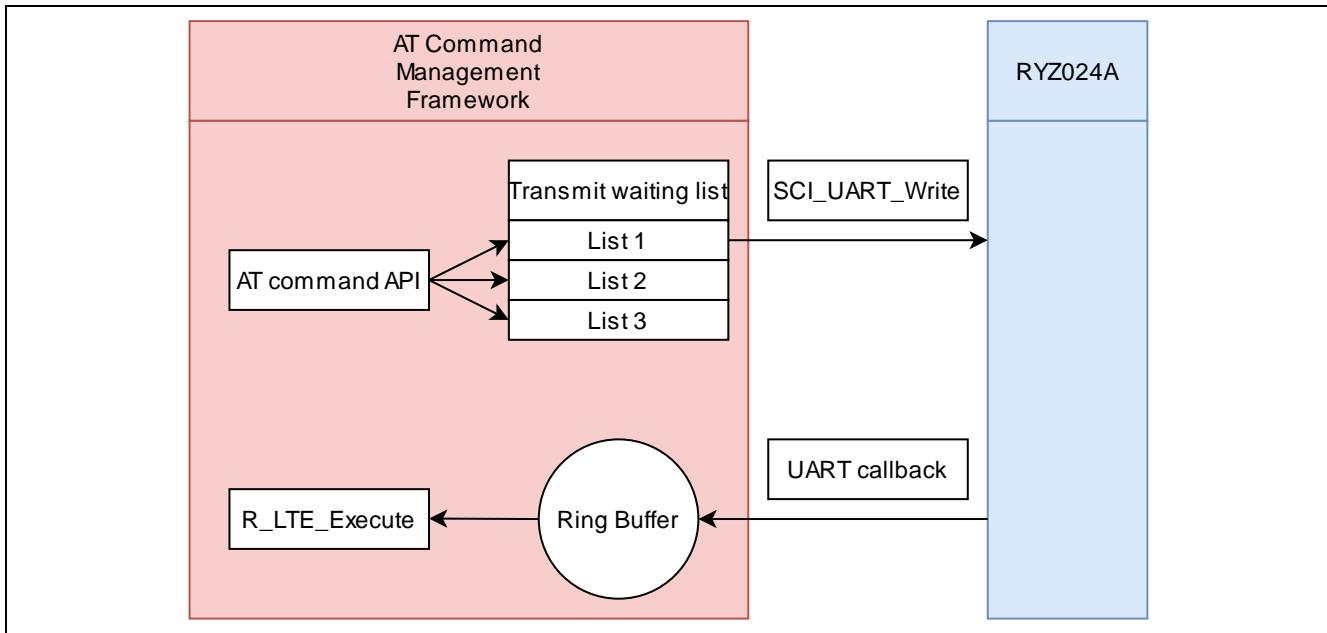
Figure 26.   Using SCI UART Module

3.5.2 AGT Timer module

The AT Command Management Framework uses the AGT Timer module to implement the timeout function. After sending an AT command, a timeout occurs when 60 seconds elapse before receiving a response.

Table 11. AT Command Timeout Setting (r_lte_ryz.c)

AT_COMMAND_TIMETOUT	30	Timeout count of AT command. unit: 2 sec for example, 2 sec * 30 = 60 sec

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 occured. 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 RA configurator. To change the timeout period, use the RA Configurator to change the timer count time and [Table 11. AT Command Timeout Setting (r_lte_ryz.c)].
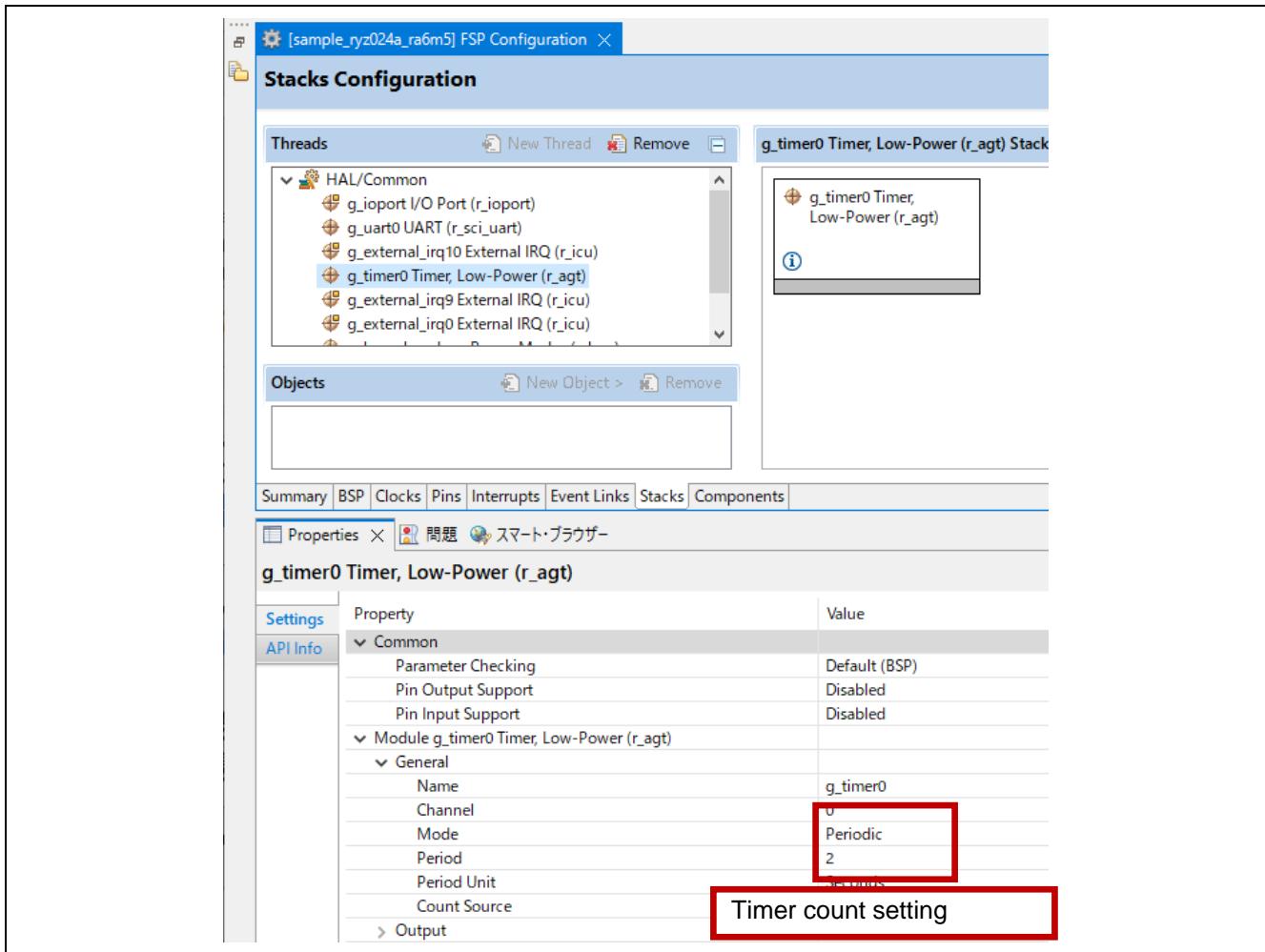
Figure 27.   AGT Timer Module Setting

3.5.3 External IRQ Module

Use the External IRQ module to generate an interrupt with a RING signal from the RYZ024A to notify that there is a URC.

3.5.4 Low Power Mode Module

Use the Low Power Mode module to control low power operation of the host MCU. For the low power consumption operation of the host MCU, refer to section, [3.7.2 Low Power Operation Control of Host MCU].

3.6 FreeRTOS Framework

This sample application provides a program using FreeRTOS. In the FreeRTOS sample program user can easily implement an application by dividing programs that need to be called repeatedly in the main loop, such as R_LTE_Execute function, into tasks. User can also stop programs that do not need to work with FreeRTOS. In order to realize these FreeRTOS operations, some changes are implemented to the framework in the FreeRTOS sample program. This section describes the operation of the AT command management framework included in the FreeRTOS sample program.

The software configuration of the FreeRTOS sample program is shown as follows.
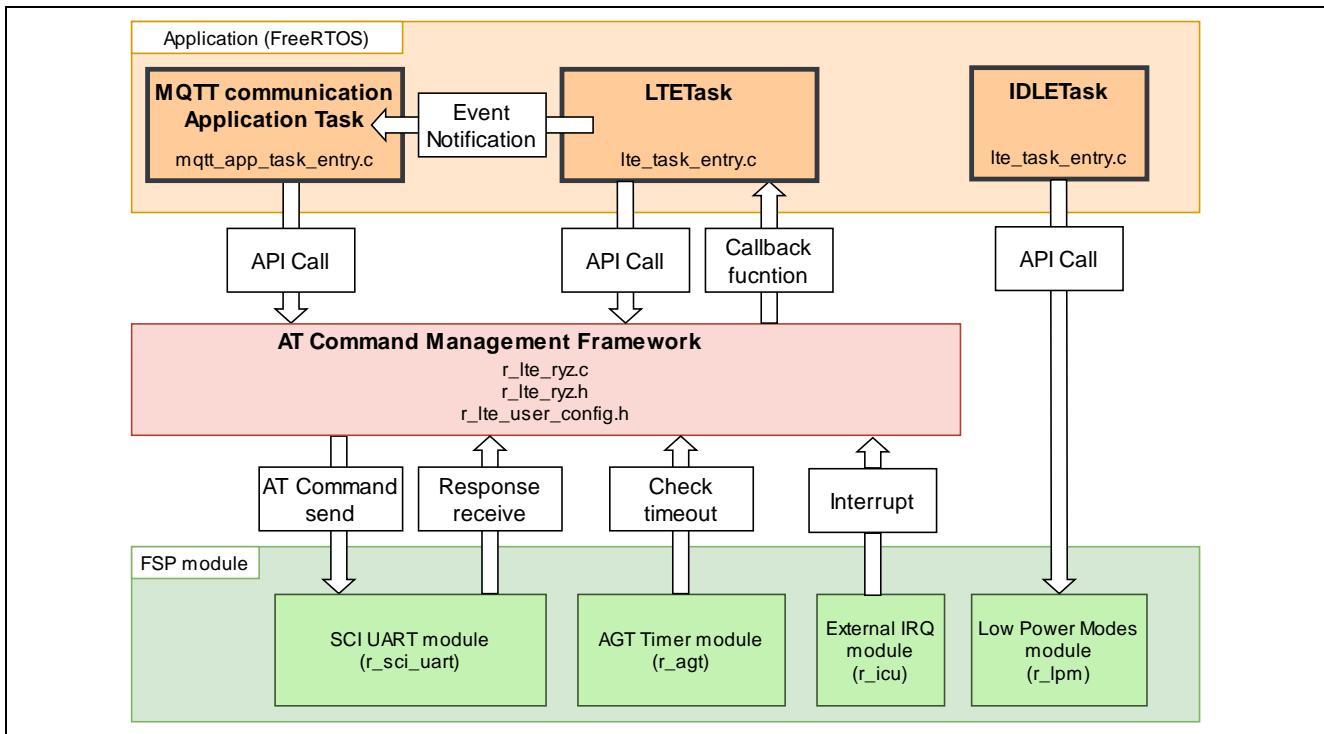
Figure 28.   FreeRTOS Software Configuration

The application of this sample program consists of two tasks: LTE task and MQTT application task. LTE tasks are R_LTE_Execute tasks for repeatedly calling functions. The callback function to be called in the R_LTE_Execute function will also be handled in this task. The MQTT application task implements MQTT communication corresponding to the input of the switch implemented in the sample program. The status of MQTT communication is notified by inter-task communication from LTE tasks as events received in callback functions.

The management API of the AT command management framework and its accompanying callback functions are assumed to be called only in LTE tasks, so exclusive processing is not implemented. On the other hand, the AT command API implements exclusive processing using semaphores so that it can be used from multiple tasks. Therefore, please use the AT command API from any task according to user application.

3.6.1 LTE Task

After calling the R_LTE_Init function, which initializes the framework, the LTE task repeatedly calls the R_LTE_Execute function to operate the framework.

If the initialization flag is changed, call the R_LTE_Init function to restart the sample application.
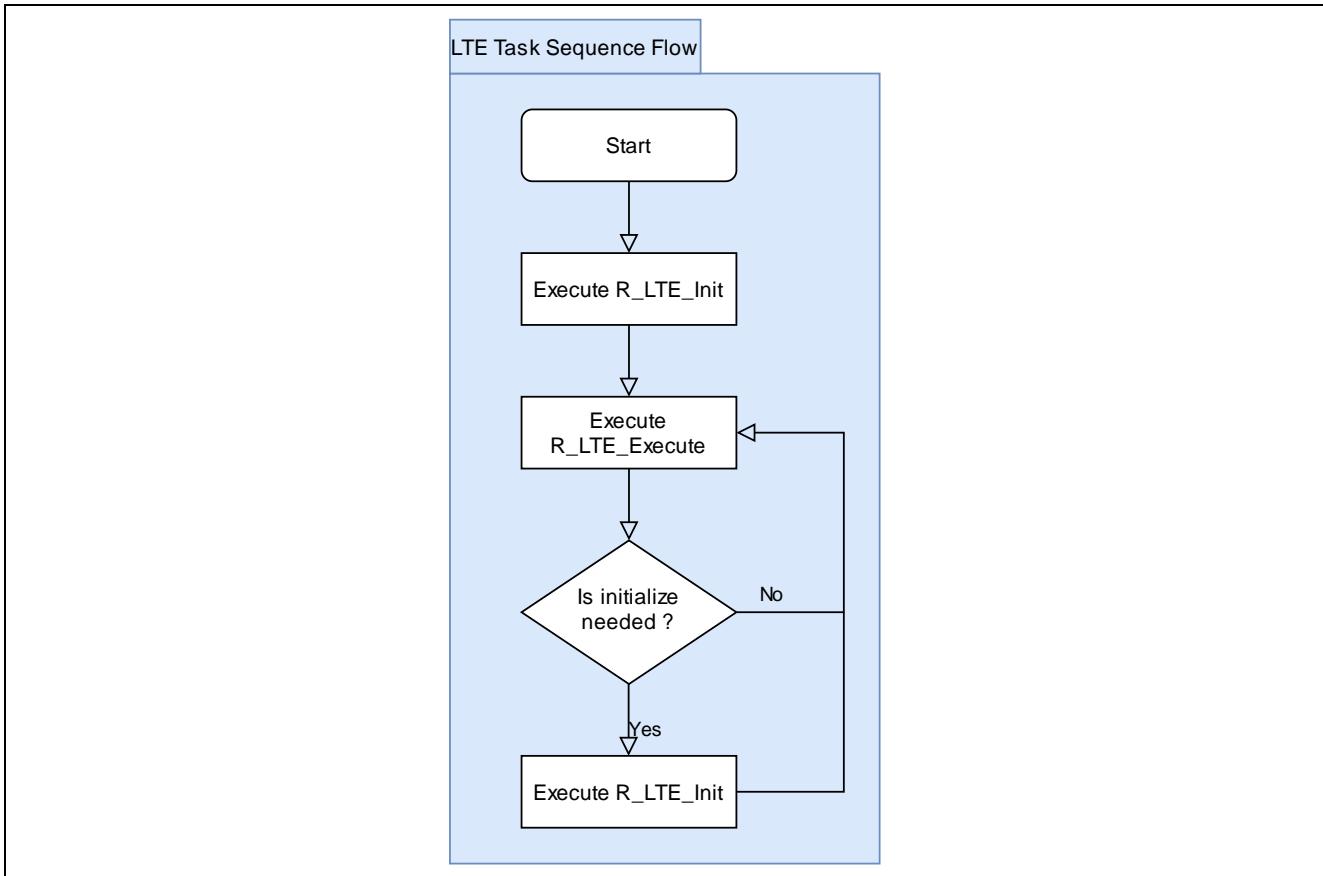
Figure 29.   Workflow of LTE Task

The R_LTE_Execute function processes all data received from RYZ024A. Call the callback function or send the AT command according to the data. It continues to operate until data cannot be retrieved from the ring buffer that temporarily holds data and suspends the task when the data runs out.
Call the Resume function from the UART interrupt handler activated by receiving data from the RYZ024A to set the LTE task to Running state.
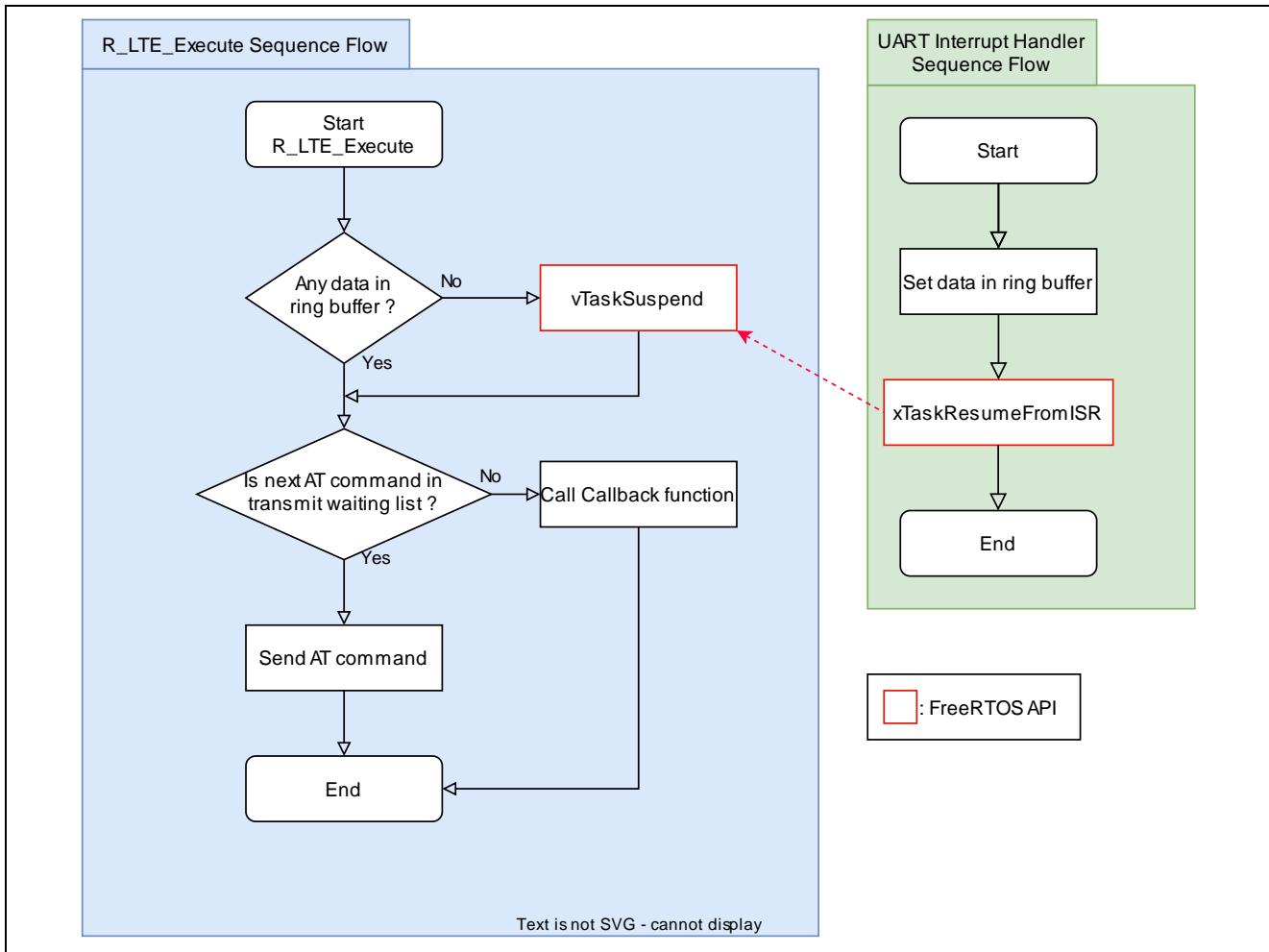
Figure 30.   Workflow of R_LTE_Execute Function

3.6.2   MQTT Communication Application Task

After starting operation, the MQTT communication application task will be in Suspended state using EventGroupWaitBits and will wait until RYZ024A MQTT communication is completed. After connecting to the MQTT server with the LTE task and completing the Subscribe request, EventGroupSetBits resumes operation. After initializing the switch, it becomes Suspend state and waits for switch input.

Call the Resume function in the switch interrupt handler that is called when the switch is pressed and set the MQTT communication application task to the Running state. Depending on the type of switch pressed, MQTT publish or Disconnect is executed, and EventGroupWaitBits is used to wait until switch processing is completed. When the switching process is completed, the LTE task callback function is called, and the operation is restarted by EventGroupSetBits.
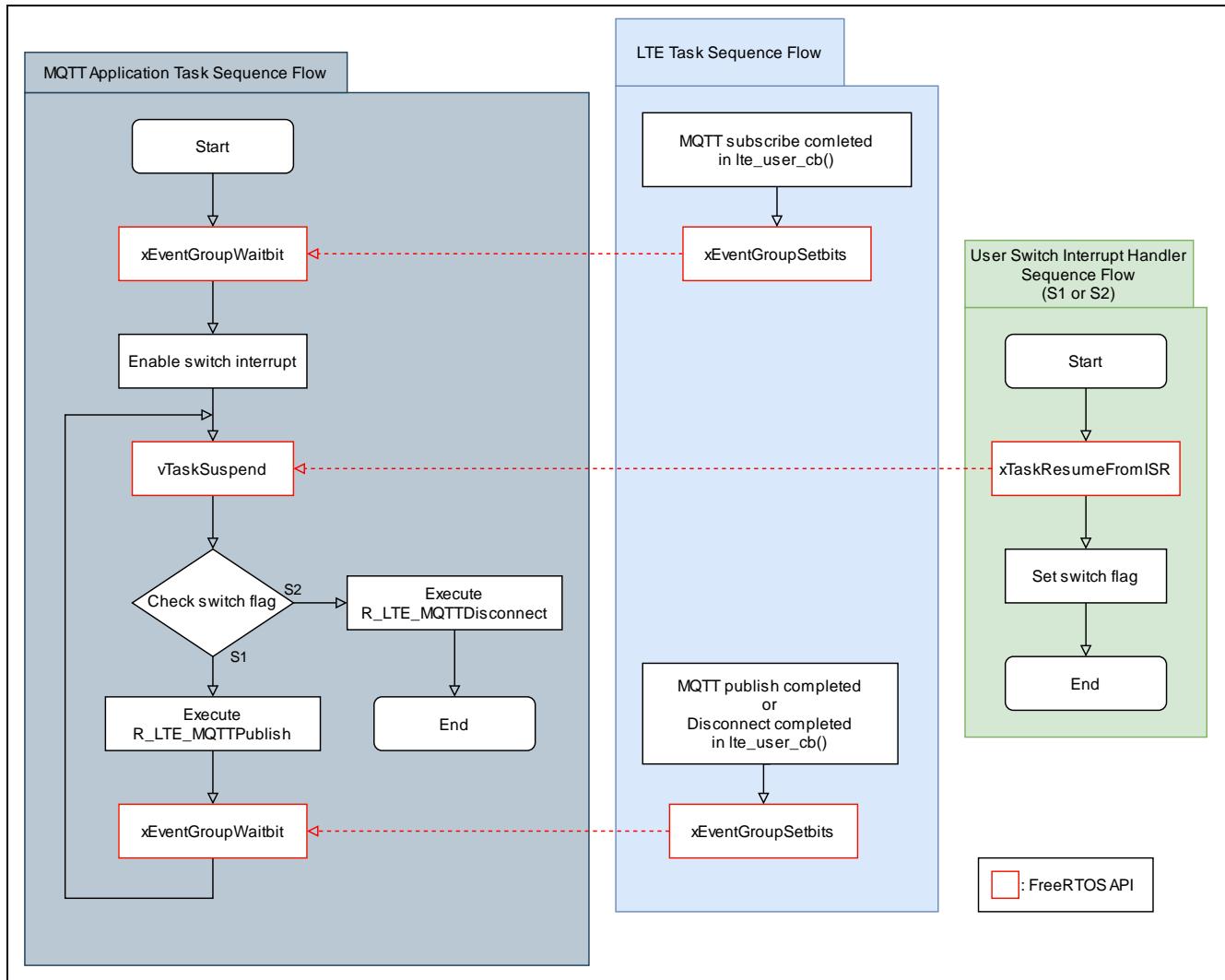
Figure 31.   Workflow of MQTT Communication Application Task

3.6.3 IDLE Task

The IDLE task is called when the LTE task, MQTT communication application task and interrupt processing are not running and puts the RA6M5 into low power consumption mode.

Refer to [3.7.2 Low Power Operation Control of Host MCU] for the processing to shift RA6M5 to low power consumption mode.

3.6.4 Task Setting Value

The settings for each task are shown below.

Table 12. LTE task Setting Value

Symbol	lte_task
Stack Size	1024
Priority	4 (Max priorities = 5)
Thread context	NULL
Memory Allocation	Static
Allocate Secure Context	Enable

Table 13. MQTT Communication Application Task Setting Value

Symbol	mqtt_app_task
Stack Size	1024
Priority	1 (Max priorities = 5)
Thread context	NULL
Memory Allocation	Static
Allocate Secure Context	Enable

Table 14. DLE Task Setting Value

Symbol	vApplicationIdleHook
Priority	0

3.7 Low Power Operation

This sample application supports operation using the low power consumption function of the RYZ024A and host MCU (RA6M5).

3.7.1 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 host MCU, 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
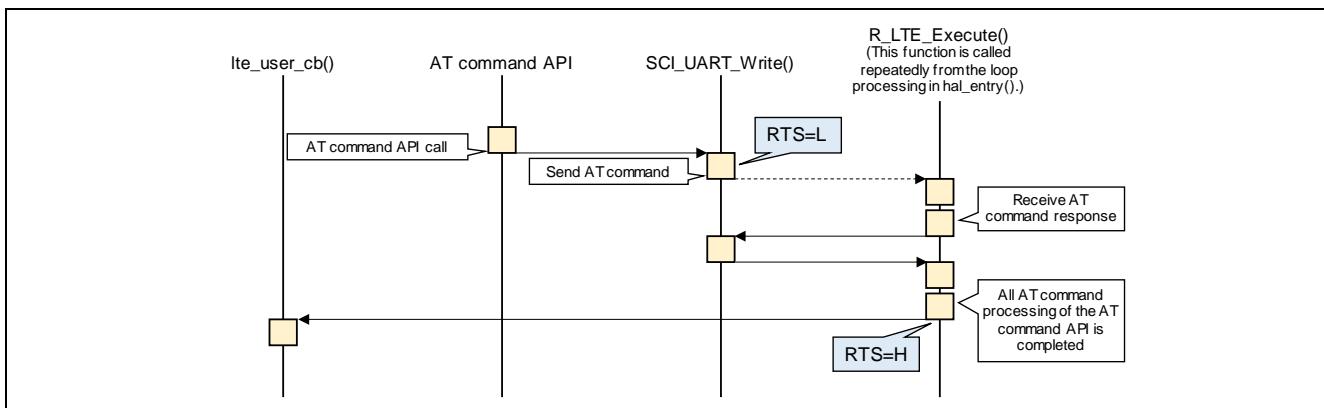
Figure 32. RTS Signal Control Sequence (when the AT command API is called)

(2) When a RING interrupt occurs
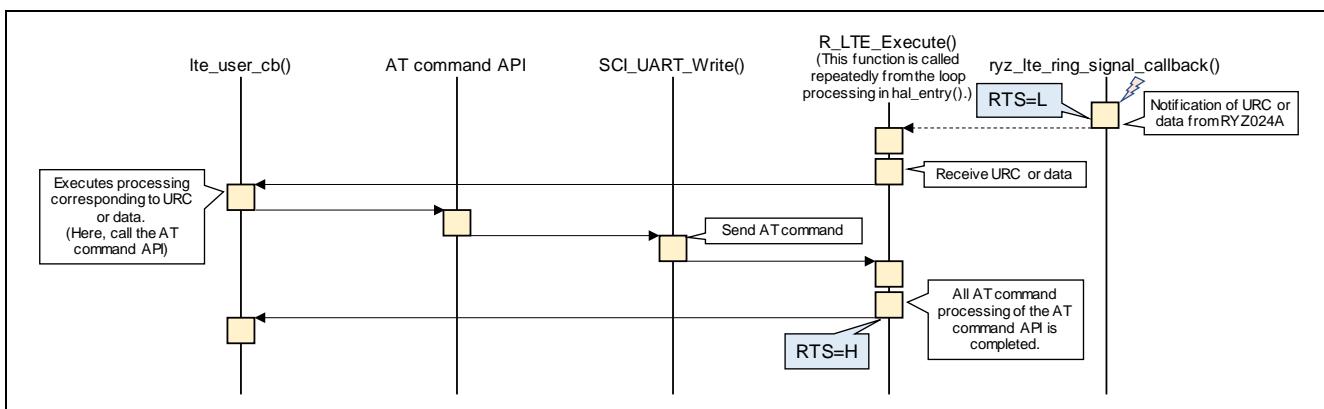
Figure 33. RTS Signal Control Sequence (when a RING interrupt occurs)

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 34. .

Baremetal application program : hal_entry.c

FreeRTOS application program : lte_task_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.

```c
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:                    eDRX setting
        {
            /* Connect to network after configuration of operation mode complete */
            SEGGER_RTT_printf(0, "OM COMFIG COMP\n");
            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:                  PSM setting
        {
            /* Configure PSM after configuration of eDRX complete */
            SEGGER_RTT_printf(0, "eDRX COMFIG COMP\n");
            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;


/* Omission */
```

Figure 34.   eDRX and PSM Setting

3.7.2 Low Power Operation Control of Host MCU

The host MCU enters software standby mode when the host MCU is in the IDLE state, and low power consumption operation in sleep 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.

Baremetal application program : `r_lte_ryz.c`, `R_LTE_Execute()`

FreeRTOS application program : `r_lte_ryz.c`, `vApplicationIdleHook()`
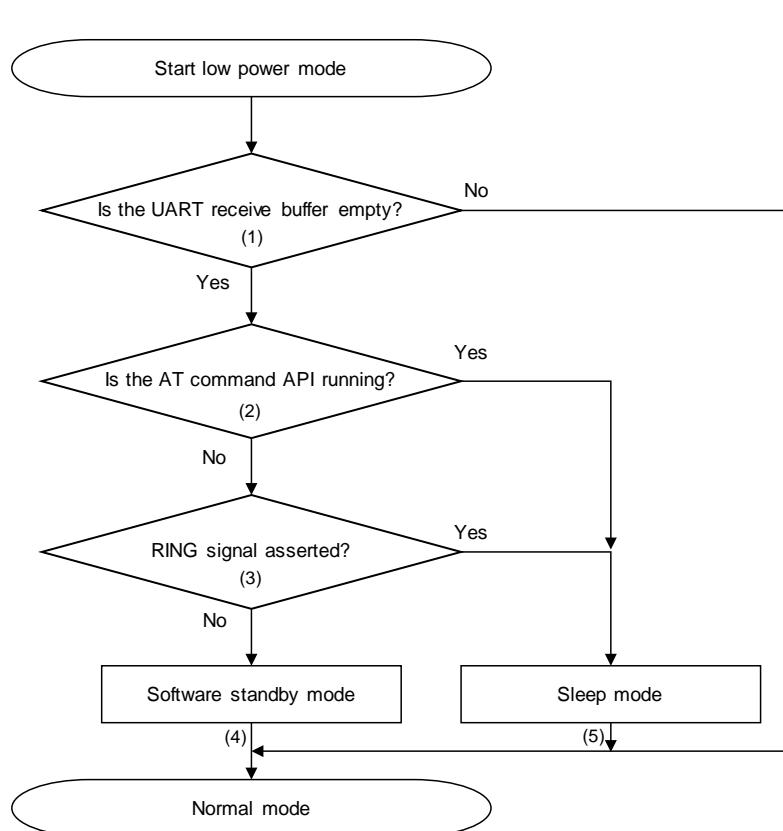
Figure 35.   Host MCU Low Power Consumption Operation Flow Chart

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 sleep mode so that the URC from the RYZ024A can be received.
4. From software standby, the S1 or S2 button 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.

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 RYZ024A and the host MCU 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
- Baremetal application program:
    — hal_entry.c
- FreeRTOS Application program:
    — lte_task_entry.c
    — mqtt_app_task_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 FSP module 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 hal_entry(void)
{
    SEGGER_RTT_printf(0, "PROGRAM START\n");

    /* SW interrupt driver open */
    R_ICU_ExternalIrqOpen(g              l_irq10.p_cfg);
    R_ICU_ExternalIrqOpen(g              irq9.p_cfg);

    /* Initialize framework-ba   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 36.   Implement Management API (hal_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 m    after initiation complete */
                SEGGER_RTT_printf(0     INIT COMP\n");
                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 */
                SEGGER_RTT_printf(0, "OM COMFIG COMP\n");
                R_LTE_NWK_Connect(1);
            } break;

/* Omission */
```

> 1. Call AT command API

> 2. Receive result with callback function

> 3. Call next AT command API

Figure 37.   Implement AT Command API (hal_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.

```c
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 38.   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;                    Check argument
    }

    if(LTE_API_NO_CURRENT_API != gs_process_api)
    {
        return LTE_ERR_IN_PROCESS;                  Check running
    }                                               AT command API

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

 /* Omission */
```

Figure 39. 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;
    }
```

Callout (pointing to "Set AT command to ATC list"):
Add following to first of Transmit waiting list:
atcommand =
"AT+SQNSMQTTPUBLISH=0,"[p_topic]",0,[length]
comp_msg = "OK"
data_exist_flag = 1

Callout (pointing to data/data_size):
Set the address of the data you want to send in data

Callout (pointing to if check):
Check the Data Size to register the argument in the transmit waiting list

Figure 40.   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;
```

Callout (pointing to ryz_lte_transmit_atc_list):
Send first AT command registered in transmit waiting list

Figure 41.   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 size and flag ─┐

    /* 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;

/* Ommission */
```

Figure 42. 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).

Table 15. UART Communication and RYZ024A Behavior Error

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_ERRO R". 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	The callback function is called in event "LTE_EVENT_RCVURC ". check the data registered in

	in front, the application is notified by the callback function.	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.

Table 16. Network Communication Error

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 "+SQNSMQTTONCONNECT: 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 host computer.

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

Therefore, in this sample application, when the host MCU 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 (for example, AT+SQNSMQTTPUBLISH) after receiving "OK". [Figure 43.   Registration of "AT+CFUN?" Command (r_lte_ryz.c)] shows how to add the "AT+CFUN?" command to the transmission waiting list.

[Figure 43.   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 */
```

Callout 1:
Add following to first of Transmit waiting list.

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

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

Figure 43.   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 17. Definition of PMOD-RYZ024A Specific Processing (r_lte_ryz.c)].

Table 17. Definition of PMOD-RYZ024A Specific Processing (r_lte_ryz.c)

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 18. Operation setting of AT+CFUN? Command (r_lte_ryz.c)].

Table 18. Operation setting of AT+CFUN? Command (r_lte_ryz.c)

AT_POLLING_TIMETOUT	2	Timeout count for "AT+CFUN?" command.
		Unit: 2 sec
		for example, 2 sec * 2 = 4 sec
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","soracom.io"            Set later to ensure that it has been initialized.
OK
AT+CGDCONT?
+CGDCONT: 1,"IP","soracom.io",,,,0,0,0,0,0,0,0,,0

OK
AT+SQNSFACTORYRESET
ERROR                    The response may not be ERROR.
AT^RESET
OK

+SHUTDOWN

+SYSSTART

OK                       Make sure the settings are initialized and show the default values.
AT+CGDCONT?
+CGDCONT: 1,"IPV4V6","",,,,0,0,0,0,0,0,1,,0

OK
```

Figure 44.   Initializing PMOD-RYZ024A

Revision History

		Description	
Rev.	Date	Page	Summary
1.00	Apr.27.23	-	Initial release

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.