

---

RYZ014A and RA MCU

## **RYZ014A LTE Communication Sample Application**

---

### **Introduction**

The RYZ014A is a cellular module capable of LTE Cat M1 communication. The RYZ014A connects to the host MCU through a UART communication, and its operation can be controlled by AT commands. In this sample application, a program that sends AT commands from the host MCU to the RYZ014A is implemented using the AT Command Management Framework. The user can use the AT Command Management Framework as a basis of development not only for this sample application, but also applications using various communication protocols supported by the LTE Cat M1 communication function of the RYZ014A. In this sample application, the telemetry data is transmitted after the connection to the MQTT server. The data transmit is triggered by a pushbutton switch. This document describes the MQTT communication application and AT Command Management Framework implemented in this sample application.

### **Target Devices**

RYZ014A

EK-RA6M5

## Contents

1. Overview .....	3
1.1 Overview of Sample Framework .....	3
2. MQTT Communication Application .....	4
2.1 Application Environment.....	4
2.2 Application Operation .....	8
3. AT Command Management Framework .....	9
3.1 Framework Overview.....	9
3.2 API Functions .....	11
3.2.1 Management API.....	11
3.2.2 AT Command API .....	12
3.3 Callback Function .....	18
3.4 User-Specific Configuration.....	23
3.5 FSP Modules used in Framework .....	24
3.5.1 SCI UART Module .....	24
3.5.2 AGT Timer Module .....	25
4. Application Development using AT Command Management Framework.....	26
4.1 Overview of Application Development .....	26
4.2 Adding an AT Command API .....	28
4.2.1 AT Command API with Data Receive Operation .....	31
4.3 Guideline of Error Handling .....	31
Revision History .....	34

## 1. Overview

### 1.1 Overview of Sample Framework

The RYZ014A is a cellular module with LTE Cat M1 communication function. This function can be controlled by entering AT commands as string data through the UART.

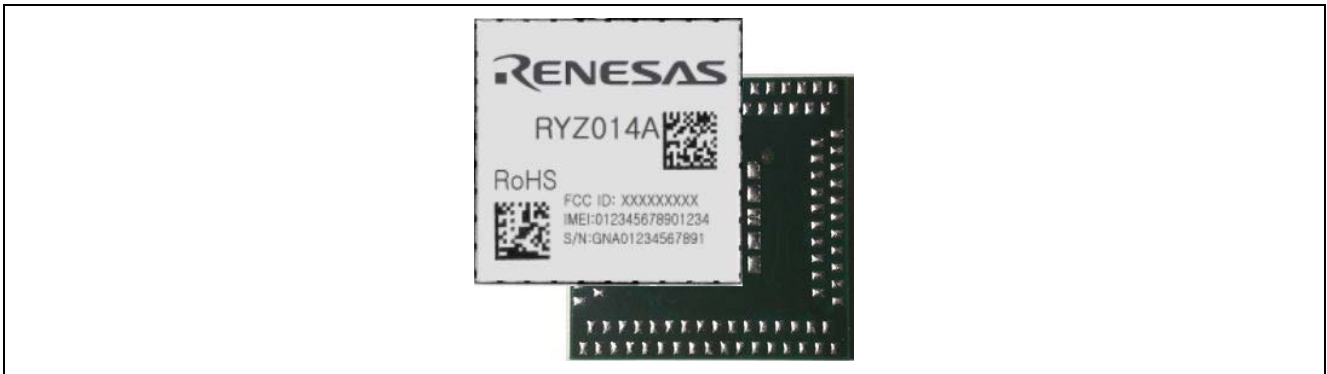


Figure 1. RYZ014A

In this sample application, AT command framework software controls RYZ014A using the RA MCU as the host MCU. The RA MCU sends AT command as string data to the RYZ014A through 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 RYZ014A.

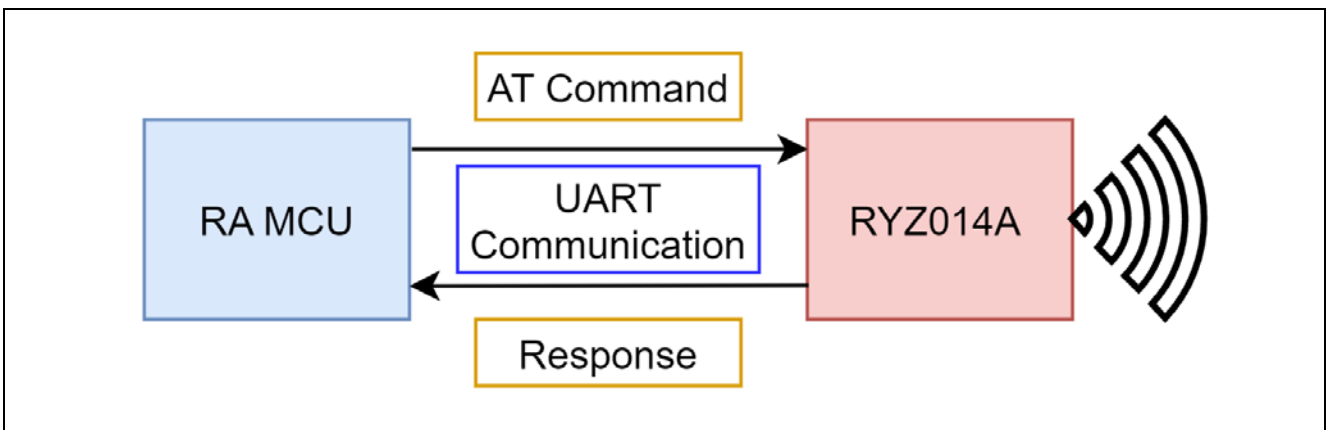


Figure 2. Communication of RYZ014A and host MCU

## 2. MQTT Communication Application

### 2.1 Application Environment

This section describes the environment to operate MQTT communication application.

This application operates in the hardware environment described in Table 1.

**Table 1. Hardware environment**

Hardware	Description
PMOD Expansion Board for RYZ014A	RYZ014A module (RTKY014A0B00000BE)
EK-RA6M5	Host RA MCU (RTKY014A0B00000BE)
Windows PC	RA MCU's application development environment and debug console for operation conformation

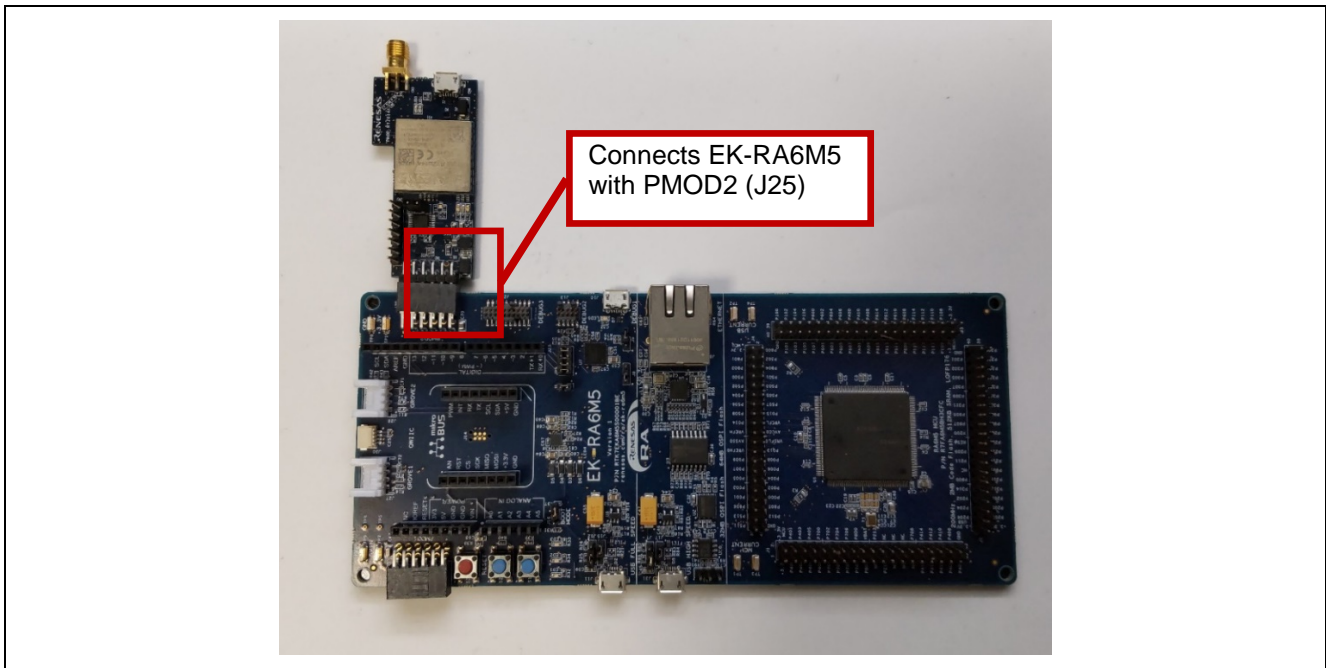
The developed application works with the software environment described in Table 2.

**Table 2. Software environment**

Software	Version	Description
e2 studio	2022-04	Renesas IDE ( <a href="https://www.renesas.com/e2studio">https://www.renesas.com/e2studio</a> )
FSP	3.8.0	Driver that can be used with RA MCU ( <a href="http://www.renesas.com/fsp">http://www.renesas.com/fsp</a> )
SEGGER RTT Viewer	7.64	Debug write display viewer ( <a href="https://www.segger.com/products/debug-probes/j-link/tools/rtt-viewer/">https://www.segger.com/products/debug-probes/j-link/tools/rtt-viewer/</a> )

For application operation, perform the following steps:

1. Connect the EK-RA6M5 and RYZ014A through the PMOD connector. Use the PMOD2 (J25) connector for EK-RA6M5.



**Figure 3. Connect EK-RA6M5 and RYZ014A**

2. Connect USB cables to EK-RA6M5 and RYZ014A. Also connect the antenna to the RYZ014A.

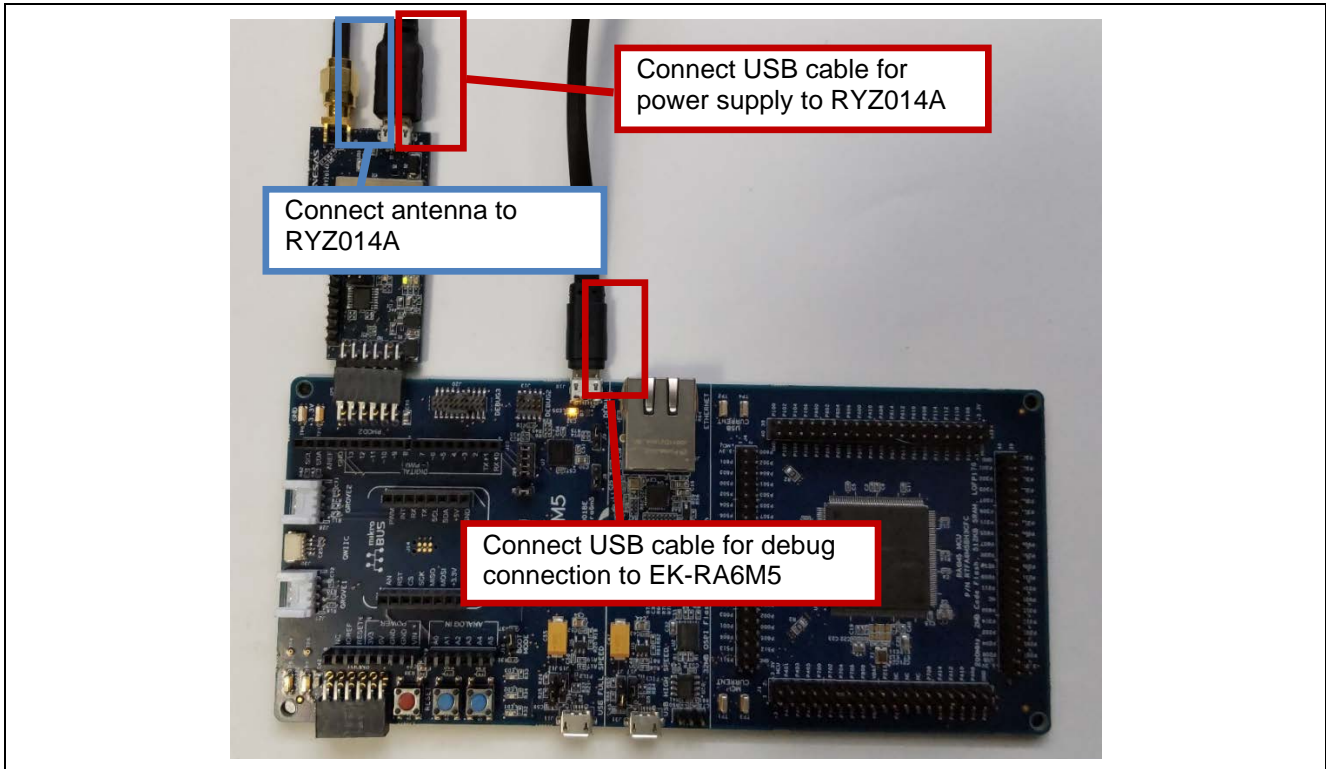


Figure 2.1 Connect USB cable and antenna

3. Import sample project into e2 studio.

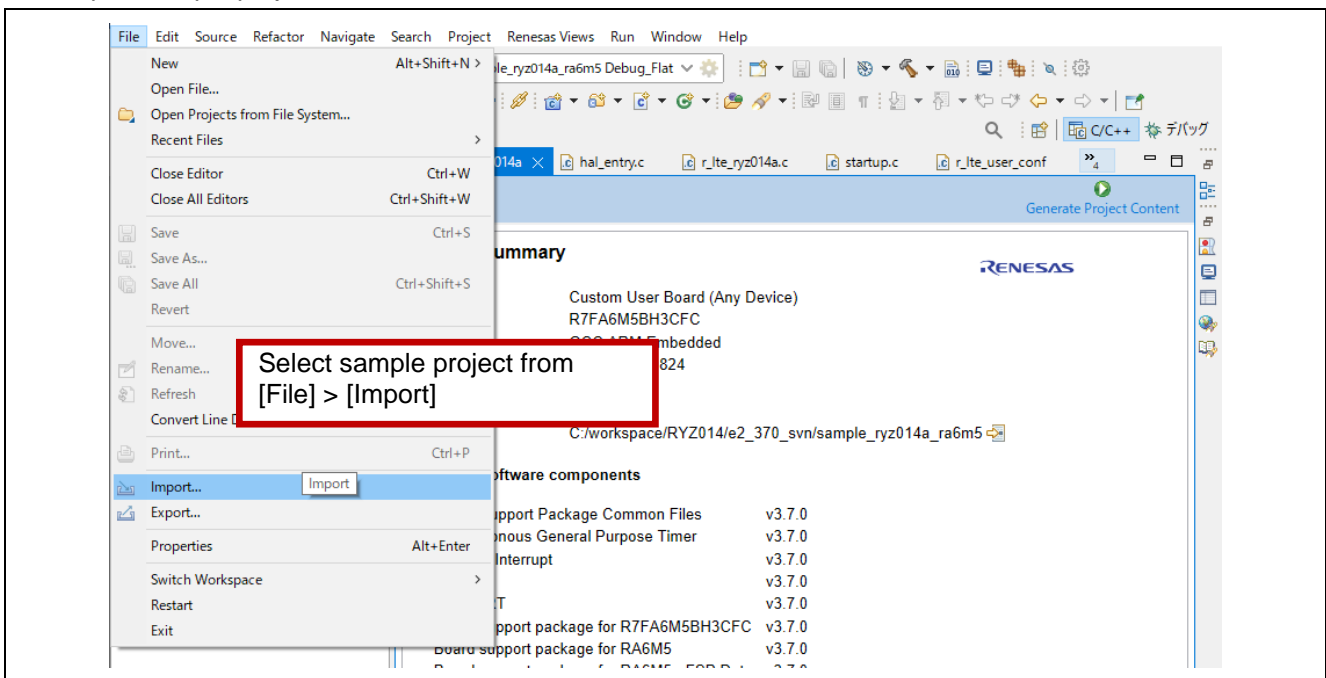


Figure 2.2 Add sample project

4. Change the Access Point Name and LTE bands. The “hal\_entry.c” file specifies the Access Point Name and the LTE bands to be used when connecting to the LTE network. These values must be changed to match the user application.

#### Access Point Name (APN)

The APN changes depending on the SIM in use. Contact the SIM provider for APNs that can be used for a user SIM. Refer to the manual of your Renesas kit for information about the SIM included with the kit, such as how to activate the SIM and available APNs.

#### LTE Bands

LTE bands change depending on region or network in use. If you know the LTE bands to be used, specify those bands. The following is an example of the LTE bands:

— “1,19”

- When specifying DOCOMO bands.

— “2,4,12”

- When specifying AT&T bands.

— “1,2,3,4,5,8,12,13,17,18,19,20,25,26,28,66”

- Select this value when you cannot specify the bands to be used. If you specify this value, it may take a few minutes for the band to be selected for the first time.

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

— APN: soracom.io

— LTE bands: 1,19

```

1  #include "hal_data.h"
2  #include "r_lte_ryz_lte.h"
3  #include "SEGGER_RTT/SEGGER_RTT.h"
4
5  #include <string.h>
6  #include <stdio.h>
7
8  FSP_CPP_HEADER
9  void R_BSP_WarmStart(bsp_warm_start_event_t event);
10 FSP_CPP_FOOTER
11
12 /* Application value definition */
13 void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t * p_data);
14 static uint8_t sw1_count = 0;
15 static uint8_t sw1_push_flag = 0;
16 static uint8_t sw2_push_flag = 0;
17 static uint8_t reinitialize_flag = 0;
18 static uint8_t mqtt_conn_state = 0;
19
20 /* APN for network connection */
21 static uint8_t str_PDP_type[] = "IPv4v6";
22
23 /* Access Point Name using in network connection. Please select depending on SIM card */
24 #if 1
25 static uint8_t str_PDP_APN[] = "globaldata.iot"; //RYZ014A-PMOD kit SIM card
26 #endif
27 #if 0
28 static uint8_t str_PDP_APN[] = "ibasis.iot"; //RYZ014A-EVK kit SIM card
29 #endif
30
31 /* Band List for network connection. Please select depending on using LTE band */
32 #if 1
33 static uint8_t str_LTE_bandlist[] = "1,19"; //Docomo bands in JP Region
34 #endif
35 #if 0
36 static uint8_t str_LTE_bandlist[] = "2,4,12"; //AT&T bands in US Region
37 #endif
38 #if 0
39 static uint8_t str_LTE_bandlist[] = "1,2,3,4,5,8,12,13,17,18,19,20,25,26,28,66"; //All selectable band
40 #endif
41
42 /* data for MQTT Connection */
43 static uint8_t str_MQTT_serveraddress[] = "test.mosquitto.org";
44 static uint8_t str_MQTT_serverport[] = "1883";

```

Figure 4. Change APN and LTE bands in hal\_entry.c

- Build the sample project. In this sample project, the user monitors the execution status using SEGGER J-Link RTT Viewer. For this setting, check the address of “\_SEGGER\_RTT” from the .map file in the Debug folder, which is generated after build.

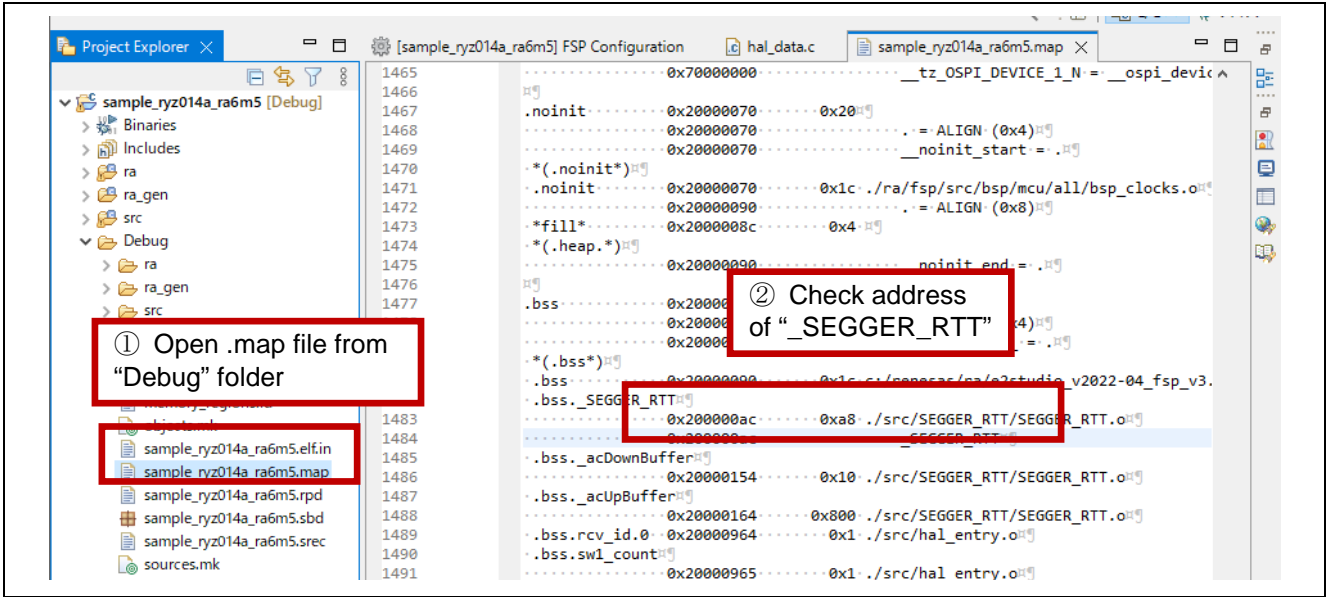


Figure 5. Check .map file

- Start the RTT Viewer and connect to the EK-RA56M5. Enter address of “\_SEGGER\_RTT” for connection.

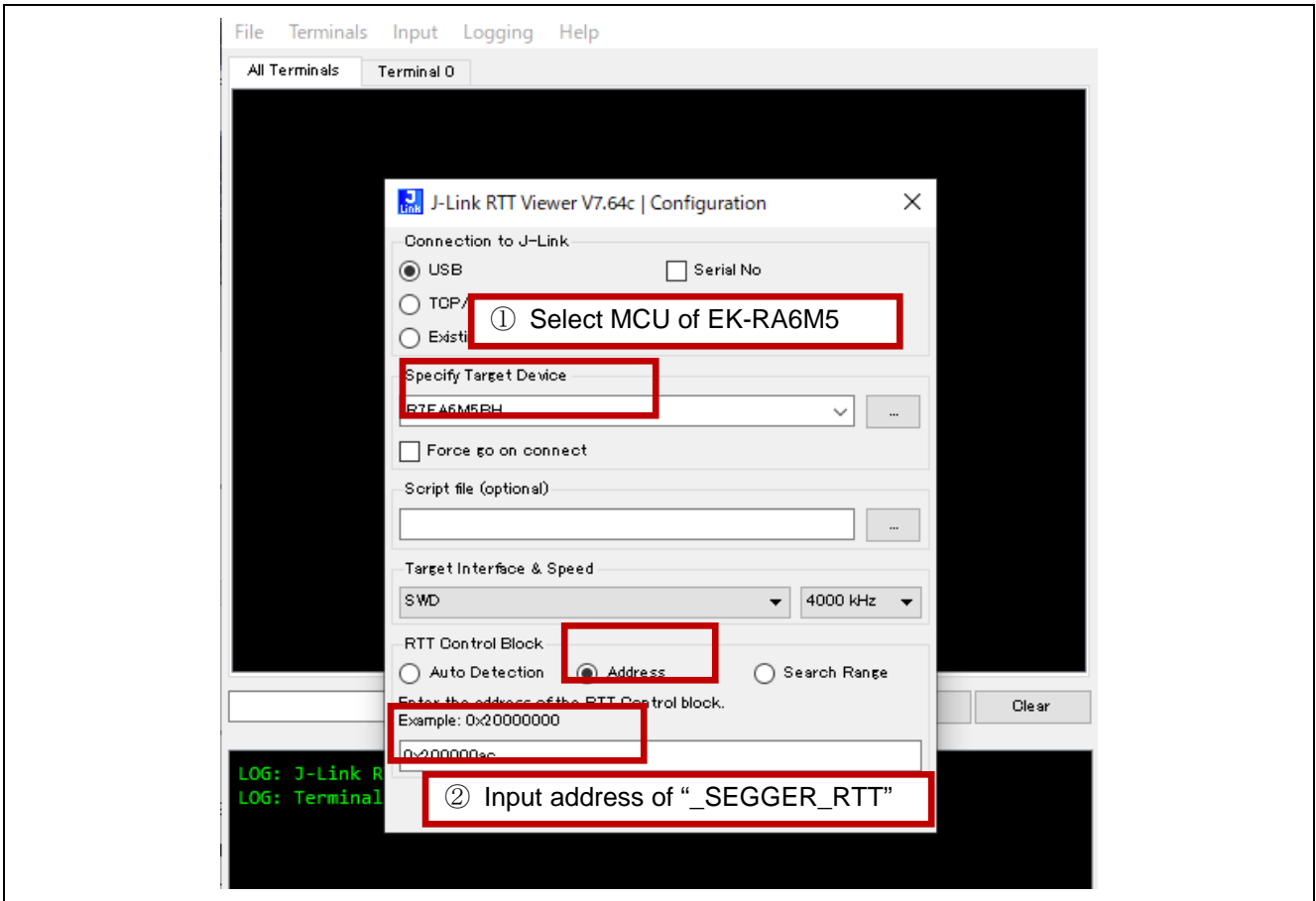
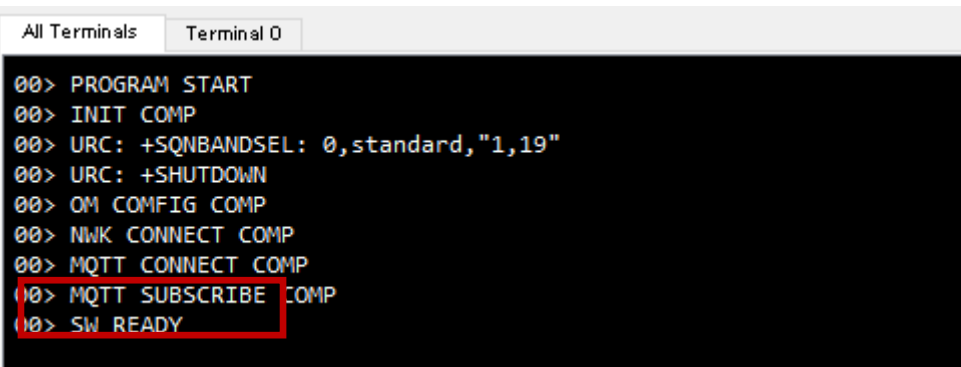


Figure 6. Start RTT Viewer



## 2.2 Application Operation

In this sample program, after resetting, the RYZ014A module connects to the cellular network through LTE, then connects to an MQTT server (public MQTT server "test.mosquitto.org" is used). After the connection to the MQTT server is completed and subscribe requests are made, the string "SW READY" is displayed in the RTT Viewer. In this state, the user can operate the pushbutton switches on the board.



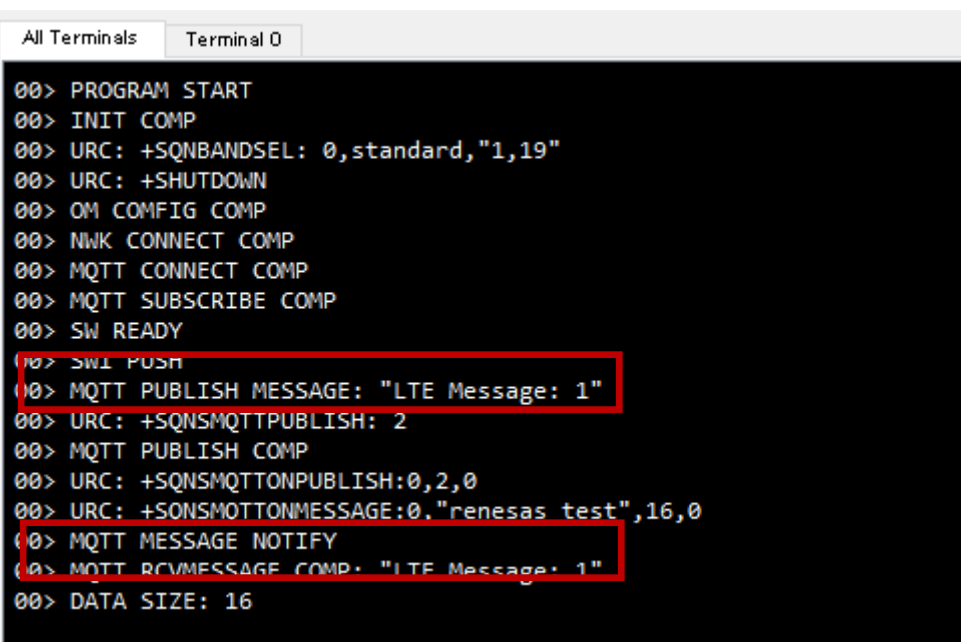
```

All Terminals Terminal 0
00> PROGRAM START
00> INIT COMP
00> URC: +SQNBANDSEL: 0,standard,"1,19"
00> URC: +SHUTDOWN
00> OM CONFIG COMP
00> NWK CONNECT COMP
00> MQTT CONNECT COMP
00> MQTT SUBSCRIBE COMP
00> SW READY

```

Figure 7. Connecting to MQTT server

After "SW READY" is displayed, press pushbutton 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 a message, it can be seen on the host as a subscribed message. The application sends message receive request using this subscribed message and displays the received messages in the RTT Viewer. Transmitting string data changes depending on the number count of SW1 pressings.



```

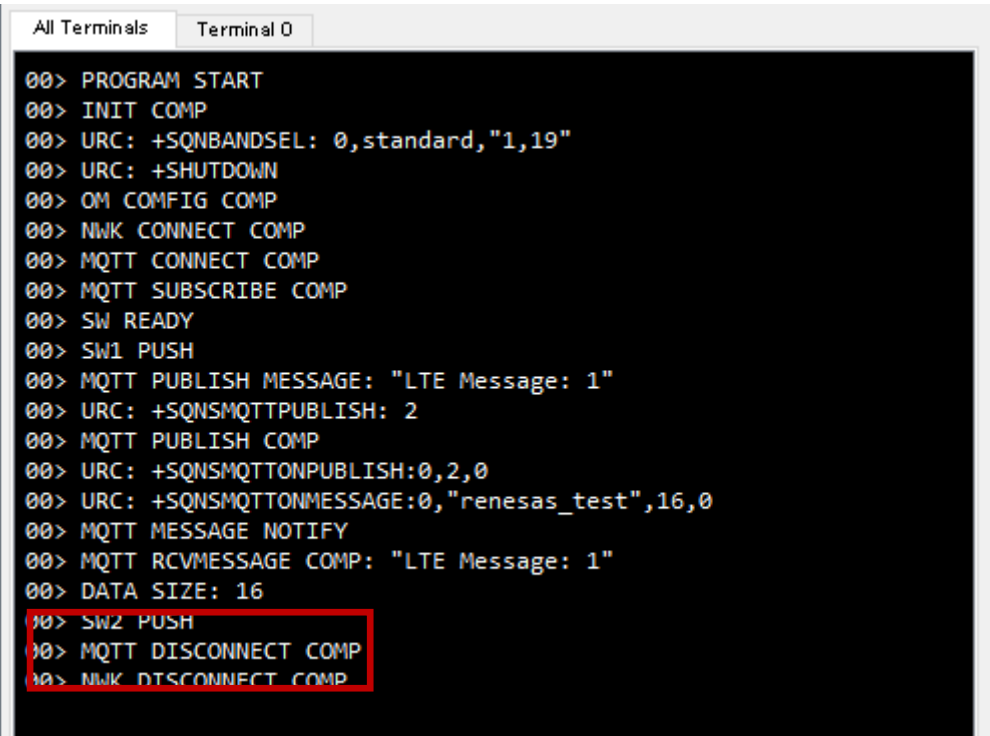
All Terminals Terminal 0
00> PROGRAM START
00> INIT COMP
00> URC: +SQNBANDSEL: 0,standard,"1,19"
00> URC: +SHUTDOWN
00> OM CONFIG COMP
00> NWK CONNECT COMP
00> MQTT CONNECT COMP
00> MQTT SUBSCRIBE COMP
00> SW READY
00> SW1 PUSH
00> MQTT PUBLISH MESSAGE: "LTE Message: 1"
00> URC: +SQNSMQTTPUBLISH: 2
00> MQTT PUBLISH COMP
00> URC: +SQNSMQTTONPUBLISH:0,2,0
00> URC: +SONSMOTTONMESSAGE:0,"renesas test",16,0
00> MQTT MESSAGE NOTIFY
00> MQTT RCVMESAGE COMP: "LTE Message: 1"
00> DATA SIZE: 16

```

Figure 8. Press SW1



Press pushbutton switch SW2 to disconnect both from the MQTT server and network.



```
All Terminals Terminal 0
00> PROGRAM START
00> INIT COMP
00> URC: +SQNBANDSEL: 0,standard,"1,19"
00> URC: +SHUTDOWN
00> OM COMFIG COMP
00> NWK CONNECT COMP
00> MQTT CONNECT COMP
00> MQTT SUBSCRIBE COMP
00> SW READY
00> SW1 PUSH
00> MQTT PUBLISH MESSAGE: "LTE Message: 1"
00> URC: +SQNSMQTTPUBLISH: 2
00> MQTT PUBLISH COMP
00> URC: +SQNSMQTTONPUBLISH:0,2,0
00> URC: +SQNSMQTTONMESSAGE:0,"renesas_test",16,0
00> MQTT MESSAGE NOTIFY
00> MQTT RCVMESSAGE COMP: "LTE Message: 1"
00> DATA SIZE: 16
00> SW2 PUSH
00> MQTT DISCONNECT COMP
00> NWK DISCONNECT COMP
```

Figure 9. Press SW2

If you are disconnected from the network or MQTT server due to a condition such as deterioration of radio wave conditions or radio signal strength, this sample application tries to connect to the MQTT server again. Therefore, after the radio wave condition is restored, "SW READY" is displayed after reconnecting to the MQTT server and making a Subscribe request, without pressing a button. After this, you can operate the switch again.

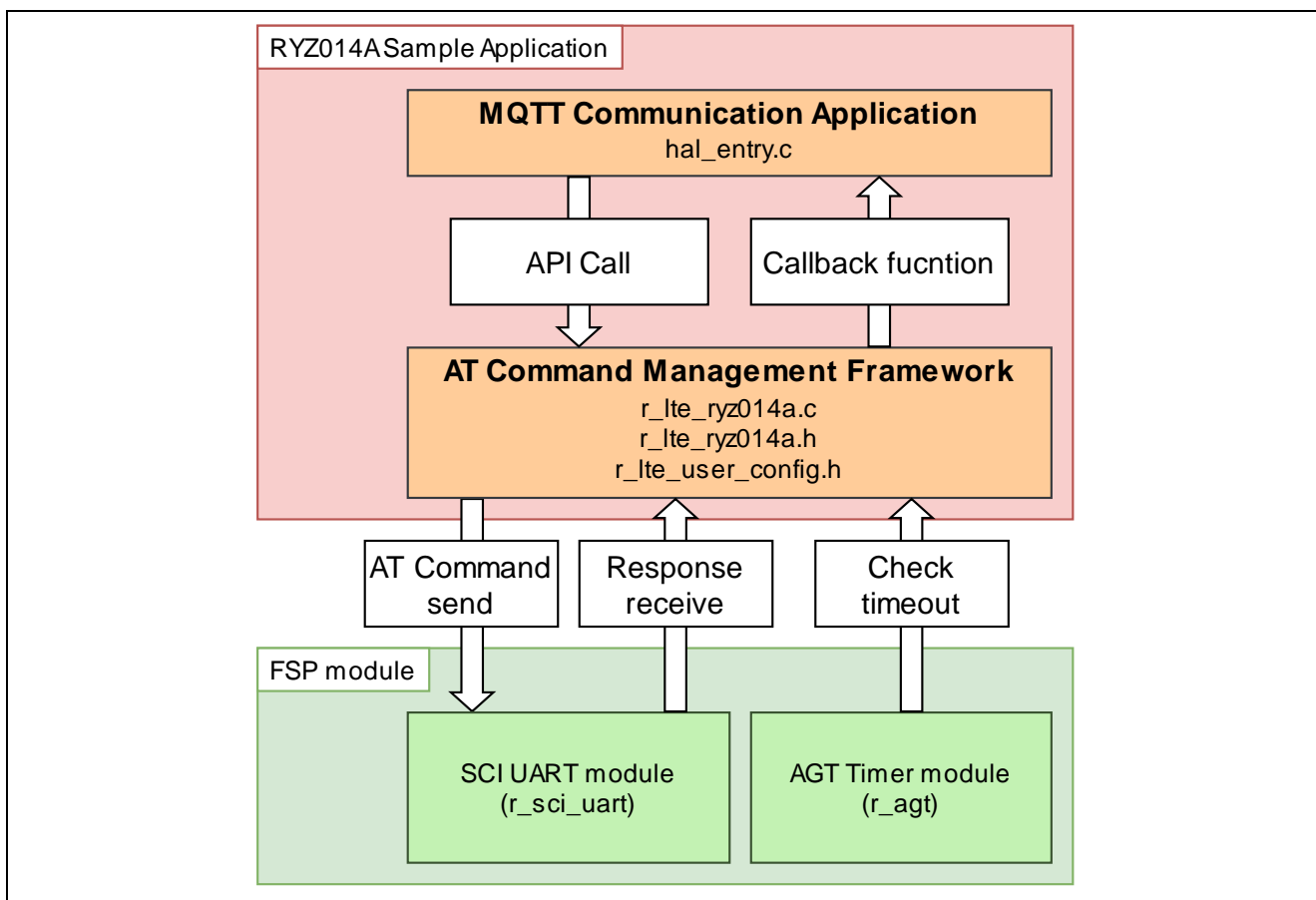
### 3. AT Command Management Framework

#### 3.1 Framework Overview

The RYZ014A is operated from the host MCU through the transmission and reception of AT commands and responses using UART communication. 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.

The AT Command Management Framework is implemented using the SCI UART and AGT timer modules of the FSP. The SCI UART module is used to send AT commands to the RYZ014A and receive responses from the RYZ014A. The AGT timer module is used to measure timeout condition after an AT command is executed.



**Figure 10. Overview of sample framework**

The executable API functions provided from the framework-based program of this sample program are described in section 3.2, API Functions.

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

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 section 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.

#### 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, there is no need to change the management API program.

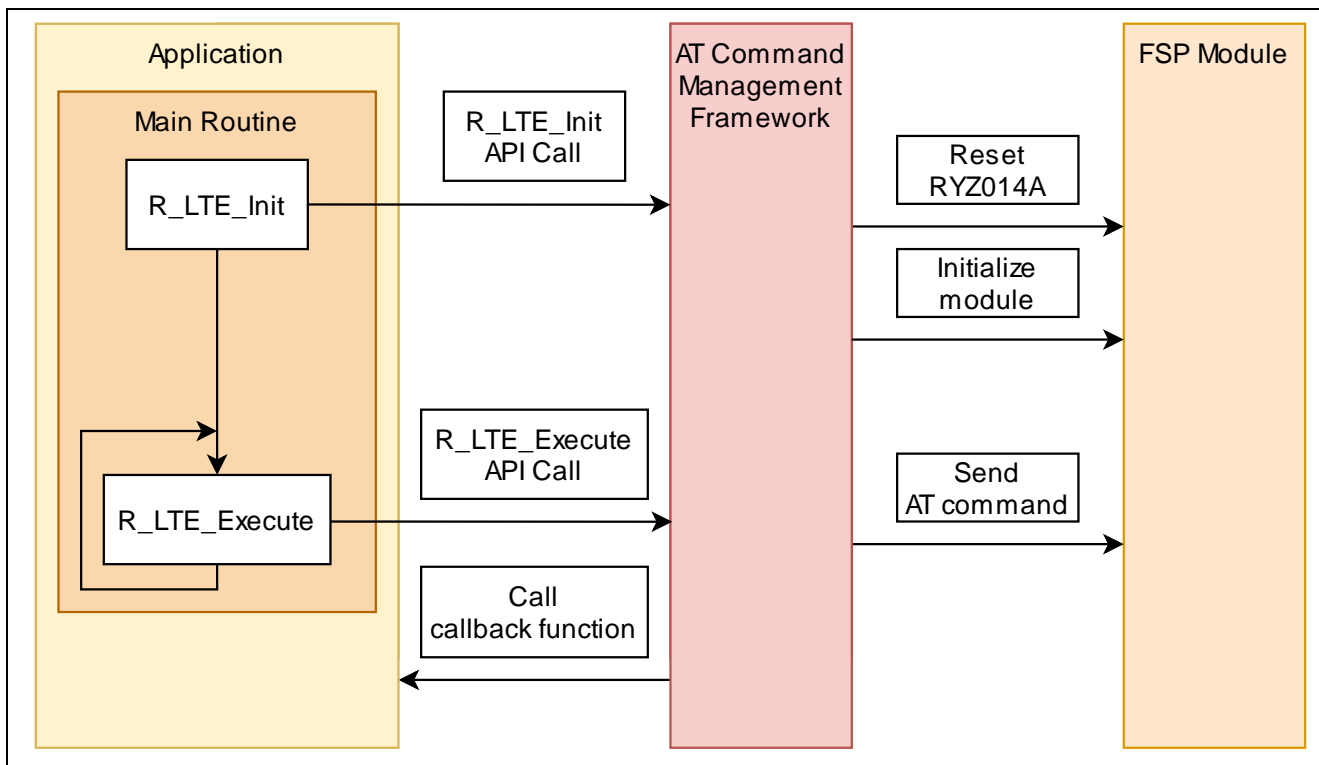


Figure 11. Management API

##### 3.2.1.1 R\_LTE\_Init

<b>Function name</b>	R_LTE_Init	
<b>Functional overview</b>	Initialize framework-based program.	
<b>Argument</b>	lte_cb_t * p_callback_fun (IN)	Callback function to register. For information about type "lte_cb_t", refer to section 3.3, Callback Function.
<b>Return value</b>	LTE_SUCCESS (0x0000)	API call success.
	LTE_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL.
<b>Detailed description</b>	Initialize RYZ014A sample framework. As part of initialization, the following operations are performed: <ul style="list-style-type: none"> <li>• Initialization of FSP modules used in the framework.</li> <li>• Execute hardware reset of RYZ014A.</li> <li>• Registration of callback function to notify result of API to application.</li> <li>• After this function is executed, AT command to reset RYZ014A will be sent.</li> </ul> The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in the callback function: LTE_API_INIT (0xFF). Call this function before the main loop of your application.	

3.2.1.2 R\_LTE\_Execute

<b>Function name</b>	R_LTE_Execute	
<b>Functional overview</b>	Perform processing of the framework-based program.	
<b>Argument</b>	void	None
<b>Return value</b>	void	None
<b>Detailed description</b>	Execute various operations to be performed by the framework. The following operation are performed: <ul style="list-style-type: none"> <li>• Send AT command specified by other API.</li> <li>• Parse string data received from RYZ014A.</li> <li>• Notifies the application of completion of API operation or receipt of errors or others by calling callback function.</li> </ul> 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 the RYZ014A 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, do not call the next AT command API before the callback function notifies the result. Also do not execute the AT command API 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 RYZ014A. 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.

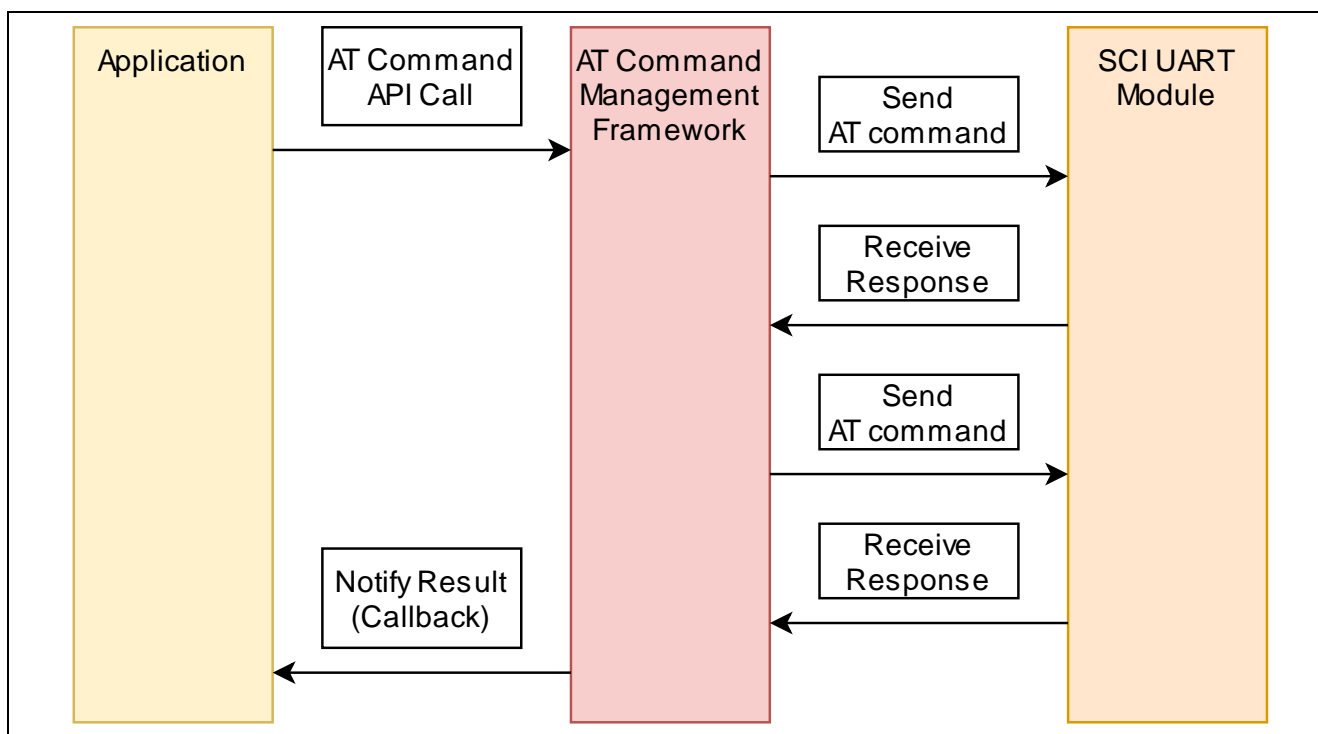


Figure 12. AT command API

## 3.2.2.1 R\_LTE\_OM\_Config

<b>Function Name</b>	R_LTE_OM_Config	
<b>Functional Overview</b>	Configure operator mode.	
<b>Argument</b>	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: "soracom.io"
	uint8_t * p_bandlist (IN)	List of authorized LTE bands. Example: "1,19"
<b>Return value</b>	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 the transmit waiting list.
<b>Detailed description</b>	<p>Sends the following AT commands in order:</p> <ul style="list-style-type: none"> <li>• "AT+CFUN=0"</li> <li>• "AT+CGDCONT=1,[ p_pdp_type],[ p_pdp_apn]"</li> <li>• "AT+SQNCTM="standard"</li> <li>• "AT+SQNBANDSEL=0," standard ",[p_bandlist]"</li> <li>• "AT^RESET"</li> <li>• "AT+CMEE=1"</li> </ul> <p>The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in the callback function: LTE_API_OM_CONFIG (0x01).</p>	

## 3.2.2.2 R\_LTE\_NWK\_Connect

<b>Function Name</b>	R_LTE_NWK_Connect	
<b>Functional Overview</b>	Connect to network.	
<b>Argument</b>	uint8_t mode	Select sending AT command.
<b>Return value</b>	LTE_SUCCESS (0x0000)	API call success.
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process.
<b>Detailed description</b>	<p>Sends the following AT commands in order depending on value of "mode" (only 0 can be used):</p> <ul style="list-style-type: none"> <li>• Mode = 0</li> <li>• "AT+CEREG=5"</li> <li>• "AT+CFUN=1"</li> </ul> <p>The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in the callback function: LTE_API_NWK_CONNECT (0x02).</p>	

## 3.2.2.3 R\_LTE\_NWK\_Disconnect

<b>Function Name</b>	R_LTE_NWK_Disconnect	
<b>Functional Overview</b>	Disconnect from network.	
<b>Argument</b>	uint8_t mode	Select sending AT command.
<b>Return value</b>	LTE_SUCCESS (0x0000)	LTE_SUCCESS (0x0000).
	LTE_ERR_IN_PROCESS (0x0002)	LTE_FAIL_IN_PROCESS (0x0002).
<b>Detailed description</b>	<p>Sends the following AT commands in order depending on value of “mode” (only 0 can be used):</p> <ul style="list-style-type: none"> <li>• Mode = 0</li> <li>• “AT+CFUN=1”</li> </ul> <p>The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in callback function: LTE_API_NWK_DISCONNECT (0x03).</p>	

## 3.2.2.4 R\_LTE\_MQTT\_Connect

<b>Function Name</b>	R_LTE_MQTT_Connect	
<b>Functional Overview</b>	Configure MQTT communication setting and connect to MQTT server.	
<b>Argument</b>	uint8_t * p_username (IN)	Username used in MQTT communication. Example: “sqn/gm01q”
	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”
<b>Return value</b>	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 the transmit waiting list.
<b>Detailed description</b>	<p>Sends the following AT commands in order:</p> <ul style="list-style-type: none"> <li>• “AT+SQNSMQTTTCFG=0,[p_username]”</li> <li>• “AT+SQNSMQTTTCONNECT=0,[p_host] ,[p_port]”</li> </ul> <p>The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in the callback function: LTE_API_MQTT_CONNECT (0x04).</p>	

### 3.2.2.5 R\_LTE\_MQTT\_Subscribe

<b>Function Name</b>	R_LTE_MQTT_Subscribe	
<b>Functional Overview</b>	Specify topic to subscribe in MQTT communication.	
<b>Argument</b>	uint8_t * p_topic (IN)	Topic to subscribe in MQTT communication. Example: "sqn/test"
<b>Return value</b>	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 the transmit waiting list.
<b>Detailed description</b>	<p>Sends the following AT command:</p> <ul style="list-style-type: none"> <li>• "AT+SQNSMQTTSUBSCRIBE=0,[p_topic],1"</li> </ul> <p>The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in callback function: LTE_API_MQTT_SUBSCRIBE (0x05).</p>	

### 3.2.2.6 R\_LTE\_MQTT\_Publish

<b>Function Name</b>	R_LTE_MQTT_Publish	
<b>Functional Overview</b>	Publish message to MQTT server.	
<b>Argument</b>	uint8_t * p_topic (IN)	Topic of publishing message. Example: "sqn/test"
	uint16_t length (IN)	Publishing message size.
	uint8_t * p_message (IN)	Publishing message.
<b>Return value</b>	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 the transmit waiting list.
<b>Detailed description</b>	<p>Sends the following AT command:</p> <ul style="list-style-type: none"> <li>• "AT+SQNSMQTTPUBLISH=0,[p_topic],1,[length]"</li> </ul> <p>The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in callback function: LTE_API_MQTT_PUBLISH (0x06).</p>	



### 3.2.2.7 R\_LTE\_MQTT\_RcvMessage

<b>Function Name</b>	R_LTE_MQTT_RcvMessage	
<b>Functional Overview</b>	Receive message from MQTT server.	
<b>Argument</b>	uint8_t * p_topic (IN)	Topic of receiving message. Example: "sqn/test"
	uint8_t message_id (IN)	Message ID of receiving message.
	uint16_t message_size (IN)	Size of receiving message.
<b>Return value</b>	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 the transmit waiting list.
<b>Detailed description</b>	<p>Sends the following AT commands in order depending on value of "message_id":</p> <p>message_id = 0</p> <ul style="list-style-type: none"> <li>• "AT+SQNSMQTTRCVMESSAGE=0,[p_topic]"</li> </ul> <p>message_id = [other than 0]</p> <ul style="list-style-type: none"> <li>• "AT+SQNSMQTTRCVMESSAGE=0,[p_topic],[message_id]"</li> </ul> <p>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.</p> <p>The following API_ID is used in callback function: LTE_API_MQTT_RCVMESSAGE (0x07).</p> <p>Note: If the user tries to receive a message that does not exists or a message that have already received, an error is notified in the callback function.</p>	

### 3.2.2.8 R\_LTE\_MQTT\_Disconnect

<b>Function Name</b>	R_LTE_MQTT_Disconnect	
<b>Functional Overview</b>	Disconnect from MQTT server.	
<b>Argument</b>	void	None.
<b>Return value</b>	LTE_SUCCESS (0x0000)	API call success.
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process.
<b>Detailed description</b>	<p>Sends the following AT command:</p> <ul style="list-style-type: none"> <li>• "AT+SQNSMQTTDISCONNECT=0"</li> </ul> <p>The result of the AT command sent by this API is notified by the callback function.</p> <p>The following API_ID is used in callback function: LTE_API_MQTT_DISCONNECT (0x08).</p>	

## 3.2.2.9 R\_LTE\_SEC\_CertificateAdd

<b>Function Name</b>	R_LTE_SEC_CertificateAdd	
<b>Functional Overview</b>	Add certification information.	
<b>Argument</b>	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.
<b>Return value</b>	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 the transmit waiting list.
<b>Detailed description</b>	<p>Sends the following AT command:</p> <ul style="list-style-type: none"> <li>“AT+SQNSNVW="certificate",[cet_id],[cet_len]”</li> </ul> <p>The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in callback function: LTE_API_SEC_CERTIFICATEADD (0x09).</p>	

## 3.2.2.10 R\_LTE\_SEC\_CertificateRemove

<b>Function Name</b>	R_LTE_SEC_CertificateRemove	
<b>Functional Overview</b>	Remove certification information.	
<b>Argument</b>	uint8_t cet_id	Removing certification ID.
<b>Return value</b>	LTE_SUCCESS (0x0000)	API call success.
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process.
<b>Detailed description</b>	<p>Sends the following AT command:</p> <ul style="list-style-type: none"> <li>“AT+SQNSNVW="certificate",[cet_id],0”</li> </ul> <p>The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in callback function: LTE_API_SEC_CERTIFICATEREMOVE (0x0A).</p>	

## 3.2.2.11 R\_LTE\_SEC\_PrivateKeyAdd

<b>Function Name</b>	R_LTE_SEC_PrivateKeyAdd	
<b>Functional Overview</b>	Add private key information.	
<b>Argument</b>	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.
<b>Return value</b>	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 the transmit waiting list.
<b>Detailed description</b>	<p>Sends the following AT command:</p> <ul style="list-style-type: none"> <li>“AT+SQNSNVW="privatekey",[prk_id],[prk_len]”</li> </ul> <p>The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in callback function: LTE_API_SEC_PRIVATEKEYADD (0x0B).</p>	

### 3.2.2.12 R\_LTE\_SEC\_PrivateKeyRemove

<b>Function Name</b>	R_LTE_SEC_PrivateKeyRemove	
<b>Functional Overview</b>	Remove private key information.	
<b>Argument</b>	uint8_t prk_id	Remove private key ID.
<b>Return value</b>	LTE_SUCCESS (0x0000)	API call success.
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process.
<b>Detailed description</b>	Sends the following AT command: <ul style="list-style-type: none"> <li>“AT+SQNSNVW="privatekey",[prk_id],0”</li> </ul> The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in callback function: LTE_API_SEC_PRIVATEKEYREMOVE (0x0C).	

### 3.2.2.13 R\_LTE\_NWK\_ConnectionConfig

<b>Function Name</b>	R_LTE_NWK_ConnectionConfig	
<b>Functional Overview</b>	Configure connection and security.	
<b>Argument</b>	uint8_t ca_cer_id	CA certification ID.
	uint8_t client_cer_id	Client certification ID.
	uint8_t prk_id	Private key ID.
<b>Return value</b>	LTE_SUCCESS (0x0000)	API call success.
	LTE_ERR_IN_PROCESS (0x0002)	Other API is in process.
<b>Detailed description</b>	Sends the following AT commands in order: <ul style="list-style-type: none"> <li>“AT+SQNSCFG=1,1,1”</li> <li>“AT+SQNSPCFG=1,2,,5,[ca_cer_id],[client_cer_id],[prk_id],””</li> </ul> The result of the AT command sent by this API is notified by the callback function. The following API_ID is used in callback function: LTE_API_NWK_CONNECTIONCONFIG (0x0D).	

## 3.3 Callback Function

When an AT command is sent to the RYZ014A, string data is received as a response. Also, the RYZ014A sends an Unsolicited Response Code (URC) if the state of the RYZ014A changes. The framework-based program of this sample program receives string data from RYZ014, 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 RYZ014A. This section describes the structure of the callback function and the events and data that are signaled by the callback function.

<b>Type name</b>	lte_cb_t	
<b>Argument</b>	uint16_t event_type (In)	Notified event type ID (see Table 3).
	uint16_t api_id (In)	ID identifying API that framework-based program is processing (see Table 4).
	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 in Table 3 and Table 4.

**Table 3. Event type IDs and values**

Macro	Value	Description
LTE_EVENT_API_COMPLETE	0x0000	An event that notifies the 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 the 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 the 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 the 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 4. API IDs and values**

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

The callback function is called from the 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:

- When all AT commands specified by the AT Command API are sent and responses to them are received and responses do not have an 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 a URC is received as a response to an AT command, string data of the 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 CONFIG COMPLETE\n");
                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);
            } break;
        }
    }
}

```

Figure 13. LTE\_EVENT\_API\_COMPLETE event notification (hal\_entry.c)

- When the response to the AT command sent to the RYZ014A is an error:
  - 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 16-bit value.

```

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

    if(LTE_EVENT_ERROR == event_type)
    {
        /* Display API ID and Error code when error occurs */
        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 */
}

```

Figure 14. LTE\_EVENT\_ERROR event notification (hal\_entry.c)

- When the AT command sent to the RYZ014A 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 RYZ014A 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 */
        timeout_flag = 1;
    }
}
/* Omission */

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

```

Figure 15. LTE\_EVENT\_TIMEOUT\_ERROR event notification (hal\_entry.c)

- When URC “+SYSSTART” is received from RYZ014A 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 RYZ014A has restarted its operation. Therefore, it is recommended to perform initialization.

Notes: In the normal operation of the RYZ014A, URC “+SYSSTART” will not be received at an unintended timing. 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 */
        reinitialize_flag = 1;
    }
}
/* Omission */

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

```

Figure 16. LTE\_EVENT\_FATAL\_ERROR event notification (hal\_entry.c)

- When URC is sent from RYZ014A:
  - Value "LTE\_EVENT\_RCVURC" is set to "event\_type".
  - The received URC string data is registered to "p\_data". Execute user 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 RYZ014A received subscribe
notification */
    const uint8_t str_onmessage[] = "+SQNSMQTTONMESSAGE";
    if(0 == memcmp(p_data, str_onmessage, (sizeof(str_onmessage) - 1)))
    {
      uint8_t rcv_id = 0;
      char * ptr;
      uint8_t msg_count = 0;

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

      /* 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 */
}

```

**LTE\_EVENT\_RCVURC event notification**

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

**Analyze parameter of URC**

**Call AT command API with  
analyzed parameter**

Figure 17. LTE\_EVENT\_RCVURC event notification (hal\_entry.c)



### 3.4 User-Specific Configuration

When developing applications based on this sample application, the user needs to change some settings depending on the using RA MCU. 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.

The settings in Table 5 specify the pin corresponding to each pin of the RYZ014A. Confirm the values in Table 5 when changing the board to be used as the host MCU.

**Table 5. Pin function setting for RYZ014A**

Name	Default value	Description
RYZ014A_RESET_PIN	BSP_IO_PORT_04_PIN_04	Pin corresponding to the reset pin of the RYZ014A. The default value corresponds to PMOD2 of the EK-RA6M5.
RYZ014A_RESET_ENABLE	BSP_IO_LEVEL_HIGH	Signal settings to enable the reset pin of the RYZ014A. The default value is Low → High to enable.
RYZ014A_RESET_DISABLE	BSP_IO_LEVEL_LOW	Signal settings to disable the reset pin of the RYZ014A. The default value is High → Low to disable.

The settings in Table 6 are for using FSP modules within the AT Command Management Framework. Edit the settings in Table 6 if you want to edit module configurations or use a different RA MCU.

**Table 6. FSP module setting for RYZ014A**

Name	Default value	Description
RYZ014A_UART_CTRL	g_uart0.p_ctrl	SCI UART Module Control Structure. By default, PMOD2 of the EK-RA6M5 uses UART value.
RYZ014A_UART_CFG	g_uart0.p_cfg	SCI UART Module Configuration Structure. By default, PMOD2 of the EK-RA6M5 uses UART value.
RYZ014A_TIMER_CTRL	g_timer0.p_ctrl	AGT timer Module Control Structure. By default, PMOD2 of the EK-RA6M5 uses channel 0 value.
RYZ014A_TIMER_CFG	g_timer0.p_cfg	AGT timer Module Configuration Structure. By default, PMOD2 of the EK-RA6M5 uses channel 0 value.

The settings in Table 7 are the size settings of various data used within the AT command management framework. Change the value 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 7. Data size settings**

Name	Default value	Description
LTE_ATC_STR_SIZE	100	Maximum length of the AT command string.
LTE_DATA_STR_SIZE	100	The maximum length of data to receive from the RYZ014A. If the data to be received exceeds this size, the excess data is discarded.
LTE_ATC_LIST_SIZE	6	The number of AT commands that can be added to the send waiting list. The default value is set to the maximum number registered by the AT command API of this sample application.

### 3.5 FSP Modules used in Framework

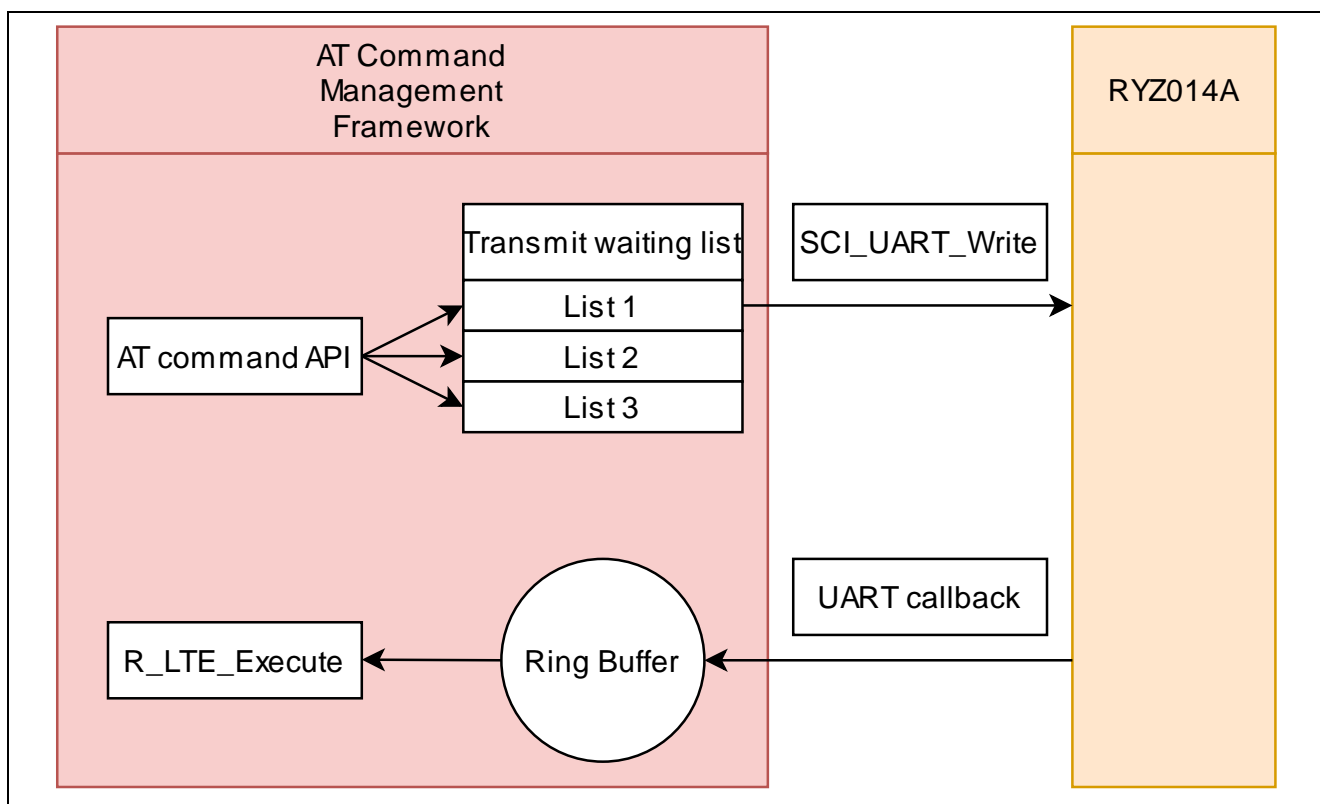
The AT Command Management Framework uses FSP modules to implement its functionality. The FSP modules are configured not only in code but also in the RA configurator. This section describes how to use and configure the FSP module used in the AT Command Management Framework.

#### 3.5.1 SCI UART Module

The AT Command Management Framework uses the SCI UART module to implement UART communication between the RYZ014A and the host MCU.

When sending AT commands from the host MCU to the RYZ014A, 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 RYZ014A to the host MCU, the data is received using the callback function of the SCI UART module. This callback function receives 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 using the R\_LTE\_Execute function call.



**Figure 18. 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.

The framework starts the timer when 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 as if a timeout has occurred. After the callback function is called, framework calls the user's callback function in the R\_LTE\_Execute function to notify the application that a timeout has occurred.

After sending an AT command, a timeout occurs after 60 seconds have elapsed. The time until the timeout occurs is set in the RA Configurator. If you want to change the time until the timeout occurs, change it from the RA Configurator.

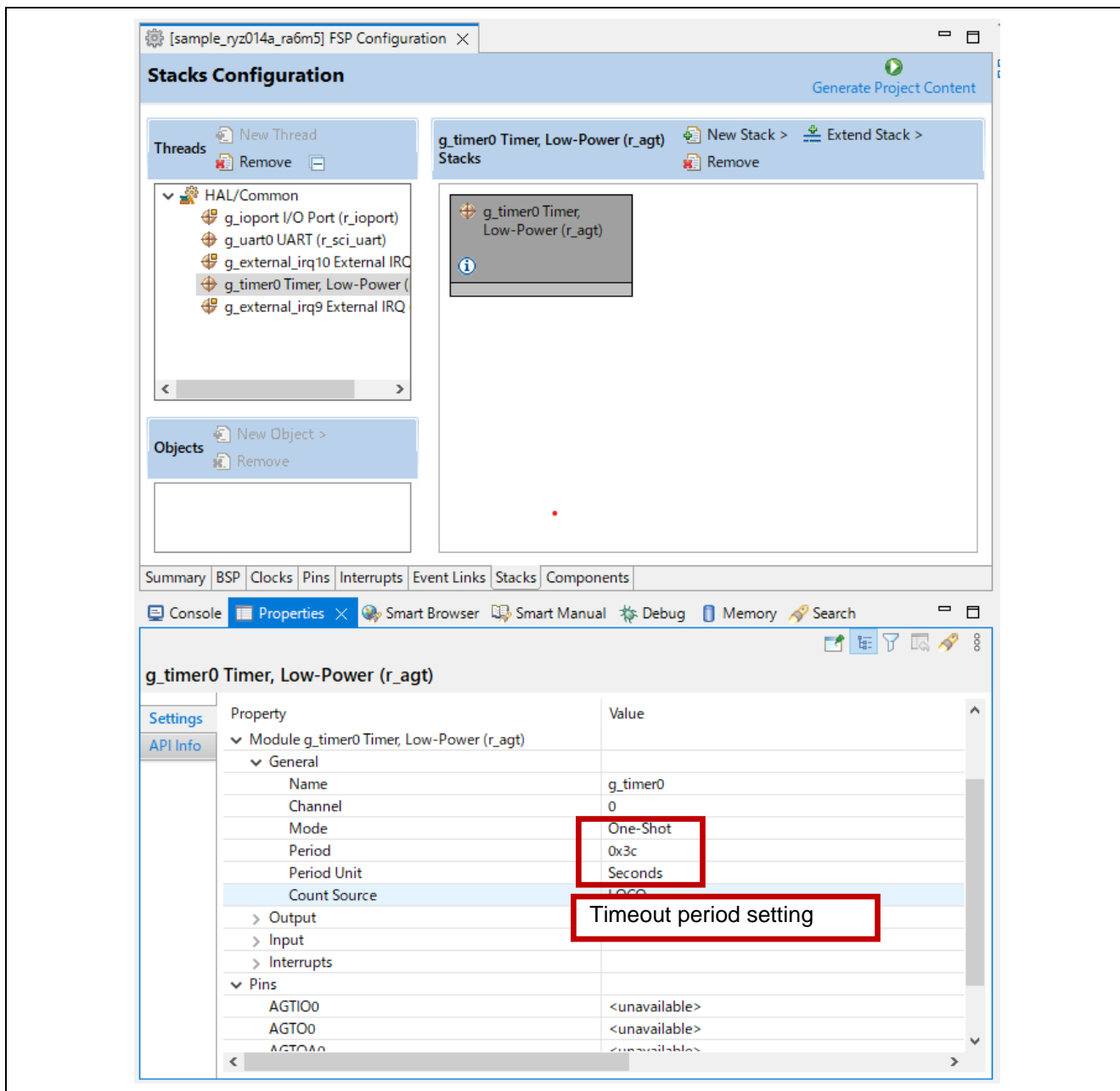


Figure 19. AGT Timer module setting

## 4. Application Development using AT Command Management Framework

The AT Command Management Framework is intended to be used as a base for user application development. By using the AT Command Management Framework, communication between the RYZ014A 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 files:

- Application program:
  - hal\_entry.c
- Framework base program:
  - r\_lte\_ryz014a.c
  - r\_lte\_ryz014a.h
  - r\_lte\_user\_config.h

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 RYZ014A. 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 RYZ014A. The RYZ014A 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 APIs implemented in the framework.
- R\_LTE\_Execute  
This is a function that holds and parses the data received from RYZ014A, 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(&R_ICU_ExternalIrq10, &R_ICU_ExternalIrq10.p_cfg);
    R_ICU_ExternalIrqOpen(&R_ICU_ExternalIrq9, &R_ICU_ExternalIrq9.p_cfg);

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

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

        /* Omission */
    }
}

```

**Call R\_LTE\_Init before calling any other APIs in framework**

**Call R\_LTE\_Execute repeatedly in the main loop.**

Figure 20. 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 RYZ014A. 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 APIs in the order they want and implementing processing corresponding to callback functions. In addition, users can add a new AT command API and use AT commands not used in this sample application.

```

void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
    if(LTE_EVENT_API_COMPLETE == event_type)
    {
        switch (api_id)
        {
            case LTE_API_INIT:
            {
                /* Configure operation mode after 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 21. 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 is 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. Also add prototype declarations to the header file (`r_lte_ryz014a.h`) so that the AT Command API can be executed from the application program.

```
typedef enum
{
    LTE_API_NO_CURRENT_API = 0,
    LTE_API_OM_CONFIG,
    LTE_API_NWK_CONNECT,
    LTE_API_NWK_DISCONNECT,
    LTE_API_MQTT_CONNECT,
    LTE_API_MQTT_DISCONNECT,
    LTE_API_MQTT_SUBSCRIBE,
    LTE_API_MQTT_PUBLISH,
    LTE_API_MQTT_RCVMESSAGE,
    LTE_API_SEC_CERTIFICATEADD,
    LTE_API_SEC_CERTIFICATEREMOVE,
    LTE_API_SEC_PRIVATEKEYADD,
    LTE_API_SEC_PRIVATEKEYREMOVE,
    LTE_API_NWK_CONNECTIONCONFIG,

    LTE_API_INIT = 0xff,
} e_lte_api_id_t;
```

**Figure 22. API IDs of this sample application (`r_lte_ryz014a.h`)**

### 2. Implementing the AT Command API

Implement the actual state of the AT Command API in the source file (`r_lte_ryz014a.c`). The AT command API of this sample application is implemented with the following configuration.

#### Checking arguments and checking the running AT Command API

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

```

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

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

    /* Clear ATC list and set processing API ID */
    ryz014a_clear_atc_list();
    gs_process_api = LTE_API_OM_CONFIG;

    /* Omission */
}

```

**Figure 23. Checking the arguments and running AT Command API of R\_LTE\_OM\_Config (r\_lte\_ryz014a.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 items 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 section 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)
  {
    ryz014a_clear_atc_list();
    return LTE_ERR_DATASIZE_OVERFLOW;
  }
}

```

**Add following to first of Transmit waiting list:**  
atcommand =  
"AT+SQNSMQTTPUBLISH=0,"[p\_topic]"0,[length]  
comp\_msg = "OK"  
data\_exist\_flag = 1

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

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

Figure 24. Register AT command of R\_LTE\_MQTT\_Publish (r\_lte\_ryz014a.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 the 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 */
  ryz014a_transmit_atc_list(LTE_TRANSMIT_ATCOMMAND);

  return LTE_SUCCESS;
}

```

**Send first AT command registered in transmit waiting list**

Figure 25. Sending the first AT command of R\_LTE\_OM\_Config (r\_lte\_ryz014a.c)

### 4.2.1 AT Command API with Data Receive Operation

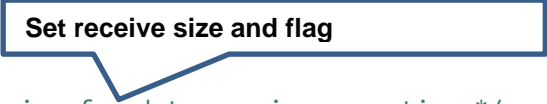
To implement the AT command API that arbitrarily receives data after sending an AT command like the `R_LTE_MQTT_RcvMessage` function implemented in this sample application, it is necessary to rewrite the global variables in the framework.

When receiving data, it is necessary to change the global variables `"gs_ryz014a_receive_size"` and `"gs_ryz014a_receive_flag"`. Set the size of the data you want to receive to `"gs_ryz014a_receive_size"` and the macro `"LTE_RCV_DATA_FLAG_ON"` for `"gs_ryz014a_receive_flag"`.

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

    /* Set receive flag and size for data receive operation */
    gs_ryz014a_receive_size = message_size;
    gs_ryz014a_receive_flag = LTE_RCV_DATA_FLAG_ON;

    /* Omission */
}
```



**Figure 26. Global value setting of `R_LTE_MQTT_RcvMessage (r_lte_ryz014a.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 RYZ014A is received after the data.

Note: "\r" or "\n" in the received data is converted to "\r\n" in the RYZ014A, 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 error detection and processing. In practice, the processing will vary depending on the requirements for the application product, so use the following information as reference information.

**Table 8. UART communication and RYZ014A behavior error**

Defect status	Framework behavior	Application response
RYZ014A is restarted unintentionally.	Receives URC "+SYSSTART". Since an unintended "+SYSSTART" is received, call the 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 RYZ014A 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 the callback function to notify application.	The callback function is called in event "LTE_EVENT_TIMEOUT_ERROR". It is recommended to initialize using R_LTE_Init function for this event.
	If the received string matches the URC specified in the comp_msg in front, the application is notified by the callback function.	The callback function is called in event "LTE_EVENT_RCVURC ". check the data registered in p_data because received string data is registered.
UART communication from the host MCU to the RYZ014A results in bit errors or character reception errors.	If there is no response to the sent string, a timeout occurs and calls the 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 RYZ014A overlap, and the RYZ014A does not perform the expected operation.	A timeout occurs when the operation stops. Framework calls the 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 RYZ014A does not enable for a long time.	The response cannot be received from the RYZ014A for a long time, and a timeout occurs. Framework calls the 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 9. Network communication error**

Defect status	Framework behavior	Application response
The network is disconnected due to a condition such as deterioration of radio wave conditions or signal strength.	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.
RYZ014A 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.
The connection is severed for some other 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" and so forth. As an example of an application implementation that supports communication disconnection, here we will explain the process of returning to MQTT communication implemented in this sample application:

- Received URC "+CEREG: 80" or "+CEREG: 4":
  - A URC that is notified when you are temporarily disconnected from the network. Since RYZ014A is trying to connect to the network again, if the radio wave condition improves, RYZ014A can reconnect to the network without executing the AT command API. At this time, MQTT communication is maintained in the RYZ014A, so MQTT communication can be resumed without executing R\_LTE\_MQTT\_Connect function when reconnecting to the network.
- Received URC "+CEREG: 0":
  - A URC that is notified when you are disconnected from the network. Since RYZ014A is trying to connect to the network again, if the radio wave condition improves, RYZ014A can reconnect to the network without executing the AT command API. When you connect to the network again, you will be notified of URC "+SQNSMQTTONCONNECT: 0,0". You can resume MQTT communication without executing R\_LTE\_MQTT\_Connect functions, but the connection to the MQTT server has been reset, so you need to run the R\_LTE\_MQTT\_Subscribe function again, for example.
- 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.

**Revision History**

Rev.	Date	Description	
		Page	Summary
1.00	Aug.31.22	-	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.

## Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
  - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
  - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.5.0-1 October 2020)

## Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,  
Koto-ku, Tokyo 135-0061, Japan  
[www.renesas.com](http://www.renesas.com)

## Trademarks

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

## Contact information

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