R01AN6324EJ0106
Rev.1.06
Jun.7.22

RX Family

RYZ014A Cellular Module Control Module Using Firmware Integration Technology

Introduction

This application note describes the usage of the RYZ014A Cellular module control module software, which conforms to the Firmware Integration Technology (FIT) standard.

In the following pages, the RYZ014A Cellular module control module is referred to collectively as “the RYZ014A Cellular FIT module” or “the FIT module”.

The FIT module supports the following Cellular module:

Renesas Electronics RYZ014A Cellular PMOD Module (RYZ014A).

In the following pages, the Renesas Electronics RYZ014A Cellular PMOD Module is referred to as “the RYZ014A Cellular module” or “the Cellular module.”

The FIT module makes use of the functionality of an RTOS. It is intended to be used in conjunction with an RTOS. In addition, the FIT module makes use of the following FIT modules:

- RX Family Board Support Package Module (R01AN1685)
- RX Family SCI Module (R01AN1815)
- RX Family BYTEQ, byte-based circular buffers, Module (R01AN1683)
- RX Family IRQ Module (R01AN1668)

Target Device

RX Family

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Related Documents

Firmware Integration Technology User’s Manual (R01AN1833)
RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
Adding Firmware Integration Technology Modules to Projects (R01AN1723)
Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)
RX Smart Configurator User’s Guide: e² studio (R20AN0451)
RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)
RX Family IRQ Module Using Firmware Integration Technology (R01AN1668)
RYZ014 Modules User's Manual: AT Command (R11UZ0093)

Contents

1. Overview

1.1 RYZ014A Cellular FIT Module

The FIT module is designed to be added to user projects as an API. For instructions on adding the FIT module, refer to “2.11 Adding the FIT Module to Your Project”.

1.2 Overview of RYZ014A Cellular FIT Module

This FIT module supports UART communication with the Cellular module.

The Cellular module driver is available in two implementation types, listed below, and the FIT module uses driver implementation type A.

1. Implementation type A:
   Driver software supporting the TCP/IP communication functionality of the module.
2. Implementation type B:
   Driver software supporting SSL communication functionality in addition to the functionality supported by implementation type A. The module handles processing for functions essential to SSL communication, such as protocol control and encryption.

1.2.1 Connection with RYZ014A Cellular Module

Examples of connections between RX65N Cloud Kit and RYZ014A Cellular module are shown in Figure 1.1.

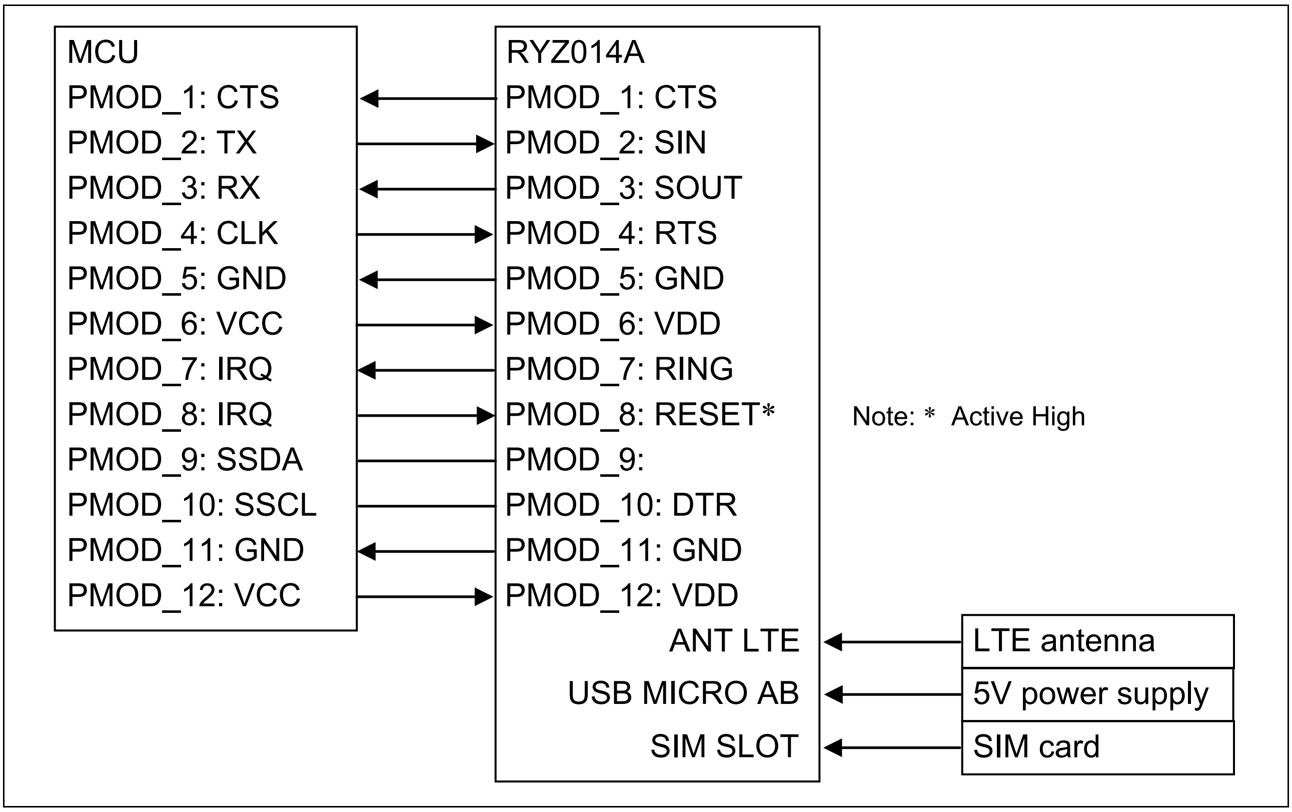

Figure 1.1 Example Connections between RX65N Cloud Kit and RYZ014A Cellular module

1.2.2 Software Configuration

Figure 1.2 shows the software configuration.

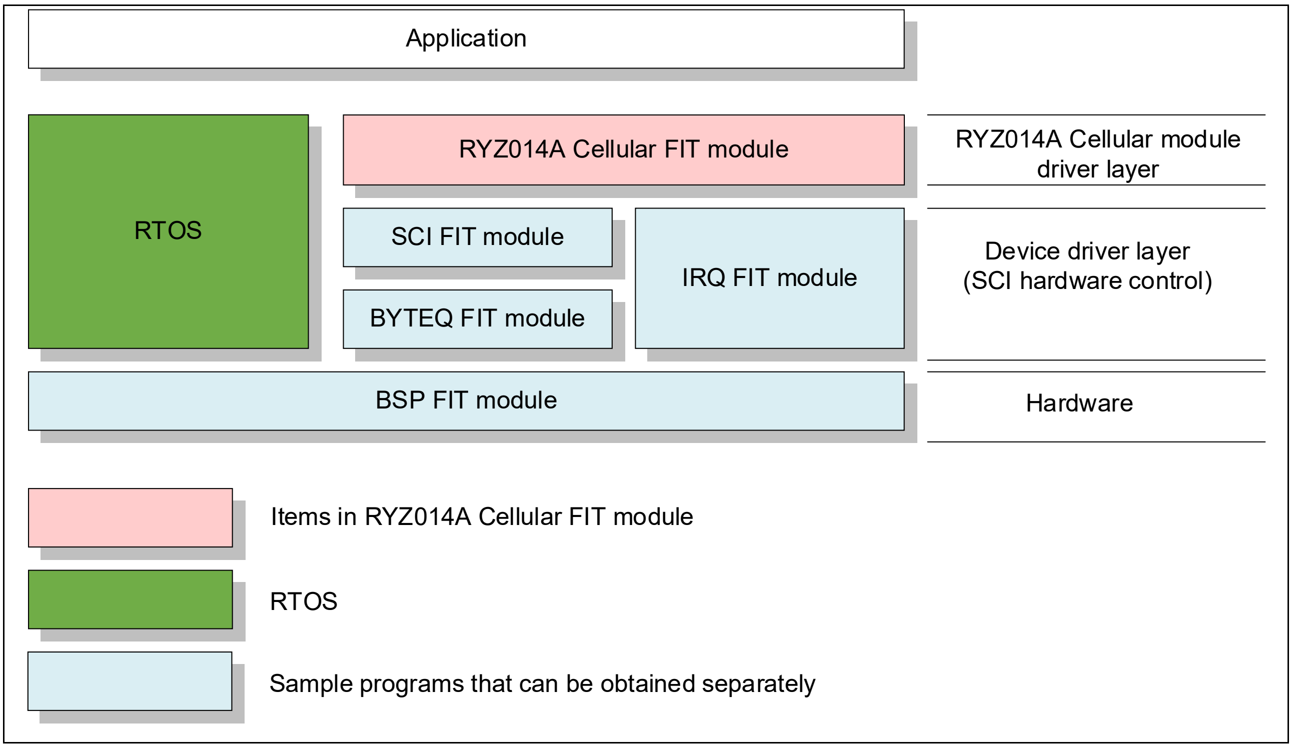

Figure 1.2 Software Configuration Diagram

(1) RYZ014A Cellular FIT module
The FIT module. This software is used to control the RYZ014A Cellular module.

(2) SCI FIT module
Implements communication between the RYZ014A Cellular module and the MCU. A sample program is available. Refer to "Related Documents" on page 1 and obtain the software.

(3) IRQ FIT module
Processes specific notifications from the RYZ014A Cellular module as interrupts. A sample program is available. Refer to "Related Documents" on page 1 and obtain the software.

(4) BYTEQ FIT module
Implements circular buffers used by the SCI FIT module. A sample program is available. Refer to "Related Documents" on page 1 and obtain the software.

(5) BSP FIT module
The Board Support Package module. A sample program is available. Refer to "Related Documents" on page 1 and obtain the software.

(6) RTOS
The RTOS manages the system overall. Operation of the FIT module has been verified using FreeRTOS and AzureRTOS.

1.2.3 Overview of API

Table 1.1 lists the API functions included in the FIT module.

Table 1.1 API functions

Function	Description
R_CELLULAR_Open	Initializes the FIT module and the Cellular module.
R_CELLULAR_Close	Closes communication between the FIT module and the Cellular module.
R_CELLULAR_APConnect	Connects the Cellular module to an access point.
R_CELLULAR_Disconnect	Disconnects the Cellular module from an access point.
R_CELLULAR_IsConnected	Obtains the FIT module’s access point connection status.
R_CELLULAR_CreateSocket	Creates a socket.
R_CELLULAR_ConnectSocket	Starts communication via a socket.
R_CELLULAR_CloseSocket	Closes a socket.
R_CELLULAR_ShutdownSocket	Ends socket communication.
R_CELLULAR_SendSocket	Transmits data via a socket.
R_CELLULAR_ReceiveSocket	Receives data via a socket.
R_CELLULAR_DnsQuery	Executes a DNS query.
R_CELLULAR_GetTime	Obtains the time information setting of the Cellular module.
R_CELLULAR_SetTime	Configures the time information setting of the Cellular module.
R_CELLULAR_SetEDRX	Enables extended discontinuous reception (eDRX).
R_CELLULAR_SetPSM	Enables power saving mode (PSM).
R_CELLULAR_GetICCID	Obtains the IC Card Identifier (ICCID).
R_CELLULAR_GetIMEI	Obtains the International Mobile Equipment Identifier (IMEI).
R_CELLULAR_GetIMSI	Obtains the International Mobile Subscriber Identity (IMSI).
R_CELLULAR_GetPhonenum	Obtains the phone number.
R_CELLULAR_GetRSSI	Obtains Received Signal Strength Indicator (RSSI) and Bit Error Rate (BER).
R_CELLULAR_GetSVN	Obtains Software Version Number (SVN).
R_CELLULAR_Ping	Pings the host.
R_CELLULAR_GetAPConnectState	Obtains the Cellular module’s access point connection status.
R_CELLULAR_GetCellInfo	Obtains information on cells.
R_CELLULAR_AutoConnectConfig	Enables autoconnect mode.
R_CELLULAR_SoftwareReset	Resets the Cellular module with the hardware reset AT command.

1.2.4 Status Transitions

Figure 1.3 shows the status transitions of the FIT module. The indications in the format "R_CELLULAR_XXX" in the figure designate calls to API functions listed in Table 1.1.

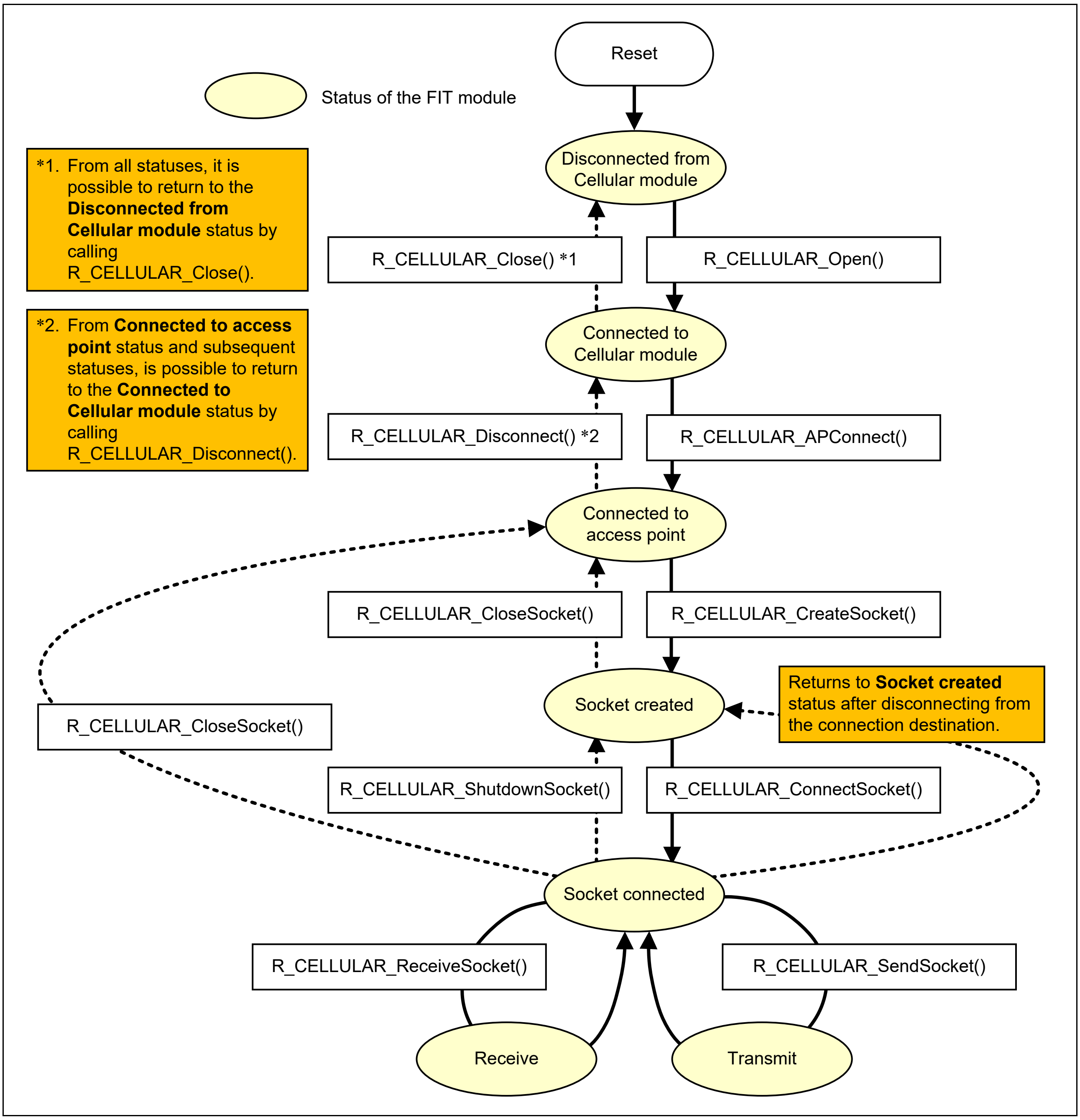

Figure 1.3 Status transitions of RYZ014A Cellular FIT module

2. API Information

The FIT module has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

The MCU used must support the following functions:

- Serial communication
- I/O ports
- IRQ
- One or more GPIO input pins that can be configured as interrupt sources

2.2 Software Requirements

The driver is dependent upon the following FIT module:

- r_bsp
- r_sci_rx
- r_byteq
- r_irq_rx

2.3 Supported Toolchain

The FIT module has been confirmed to work with the toolchain listed in 4.1, Confirmed Operation Environment.

2.4 Interrupt Vector

None

2.5 Header Files

All API calls and their supporting interface definitions are located in r_cellular_if.h.

2.6 Integer Types

The FIT module uses ANSI C99. These types are defined in stdint.h.

2.7 Compile Settings

The configuration option settings of the FIT module are contained in r_cellular_config.h.
The names of the options and their setting values are listed in Table 2.1.

Table 2.1 Configuration options (r_cellular_config.h)

Configuration options in r_cellular _config.h	
CELLULAR_CFG_AP_NAME Note: The default is "ibasis.iot".	Specifies the name of the access point to connect to. Set this option to match the SIM card used.
CELLULAR_CFG_AP_USERID Note: No default is defined.	Sets the username of the access point to connect to. Set this option to match the SIM card used. This setting may be omitted if there is no username.
CELLULAR_CFG_AP_PASSWORD Note: No default is defined.	Sets the password of the access point to connect to. Set this option to match the SIM card used. This setting may be omitted if there is no password.
CELLULAR_CFG_AP_PIN_CODE Note: No default is defined.	Sets the PIN code of the SIM card used. This setting may be omitted if no PIN code has been set.
CELLULAR_CFG_AUTH_TYPE Note: The default is 2.	Specifies authentication protocol used for PDP contexts. Set this option to match the SIM card used. 0: None, 1: PAP, 2: CHAP
CELLULAR_CFG_ATC_RETRY_GATT Note: The default is 100.	Specifies the maximum number of retries when connection to an access point fails. Set this option to a value of 0 to 100.
CELLULAR_CFG_EX_TIMEOUT Note: The default is 120.	Specifies the exchange timeout. Set this option to a value of 0 to 120.
CELLULAR_CFG_INT_PRIORITY Note: The default is 4.	Sets the interrupt priority of the serial module used for communication with the Cellular module. Set this option to a value of 2 to 15 to match the system priority.
CELLULAR_CFG_SEMAPHORE_BLOCK_TIME Note: The default is 15000.	Sets the API maximum wait time to prevent interference with the various functions. The unit is milliseconds. Set this option to a value of 1 to 15000.
CELLULAR_CFG_DEBUGLOG Note: The default is 0.	Configures the output setting for log information. The log information output setting of 1 to 4 can be used with FreeRTOS logging task. Set this option to a value of 0 to 4, as required. 0: Off, 1: Error log output, 2: Output of warnings in addition, 3: Output of status notifications in addition, 4: Output of Cellular module communication information in addition
CELLULAR_CFG_UART_SCI_CH Note: The default is 0.	Specifies the SCI port number used for communication with the Cellular module. The default value specifies SCI port number 0. Set this option to match the SCI port to be controlled.
CELLULAR_CFG_RESET_SIGNAL_LOGIC Note: The default is 1.	Changes the output format of the reset signal sent to the Cellular module. The default value specifies high-level reset signal output.
CELLULAR_CFG_RTS_PORT Note: The default is 2	Configures the port direction register (PDR) setting for the general-purpose port that controls the RTS pin of the Cellular module. The default value is suitable when port 22 is used. Set this option to match the port to be controlled.
CELLULAR_CFG_RTS_PIN Note: The default is 2.	Configures the port output data register (PODR) setting for the general-purpose port that controls the RTS pin of the Cellular module. The default value is suitable when port 22 is used. Set this option to match the port to be controlled.
CELLULAR_CFG_RESET_PORT Note: The default is D.	Configures the port direction register (PDR) setting for the general-purpose port that controls the PWD_L pin of the Cellular module. The

	default value is suitable when port D0 is used. Set this option to match the port to be controlled.
CELLULAR_CFG_RESET_PIN Note: The default is 0.	Configures the port output data register (PODR) setting for the general-purpose port that controls the PWD_L pin of the Cellular module. The default value is suitable when port D0 is used. Set this option to match the port to be controlled.

The configuration option settings of the SCI FIT module used by the FIT module are contained in r_cellular_config.h.
The names of the options for SCI FIT module and their setting values are listed in Table 2.2. Refer to the application note, “RX Family SCI Module Using Firmware Integration Technology (R01AN1815)” for details.

Table 2.2 Configuration options (r_sci_rx_config.h)

Configuration options in r_sci_rx_config.h	
SCI_CFG_CHx_INCLUDED Notes: 1. CHx = CH0 to CH12 2. The default values are as follows: CH0: 1, CH1 to CH12: 0	Each channel has resources such as transmit and receive buffers, counters, interrupts, other programs, and RAM. Setting this option to 1 assigns related resources to the specified channel.
SCI_CFG_CHx_TX_BUFSIZ Notes: 1. CHx = CH0 to CH12 2. The default value is 80 for all channels.	Specifies the transmit buffer size of an individual channel. The buffer size of the channel specified by CELLULAR_CFG_UART_SCI_CH should be set to 2048.
SCI_CFG_CHx_RX_BUFSIZ Notes: 1. CHx = CH0 to CH12 2. The default value is 80 for all channels.	Specifies the receive buffer size of an individual channel. The buffer size of the channel specified by CELLULAR_CFG_UART_SCI_CH should be set to 2048.
SCI_CFG_TEI_INCLUDED Note: The default is 0.	Enables the transmit end interrupt for serial transmissions. The FIT module uses the interrupt, then this option should be set to 1.

The configuration option settings of the IRQ FIT module used by the FIT module are contained in r_irq_rx_config.h.
The names of the options for IRQ FIT module and their setting values are listed in Table 2.3. Refer to the application note, “RX Family IRQ Module Using Firmware Integration Technology (R01AN1668)” for details.

Table 2.3 Configuration options (r_irq_rx_config.h)

Configuration options in r_sci_rx_config.h	
IRQ_CFG_FILT_EN_IRQx Notes: 1. IRQx = IRQ0 to IRQ15 2. The default value is 0 for all channels.	Specifies the channel to be used as IRQ. Set a value of 1 for the channel to which the RING pin of the Cellular module is connected.

The configuration option settings of the BSP FIT module used by the FIT module are contained in r_bsp_config.h.

The names of the options for BSP FIT module and their setting values are listed in Table 2.4. Refer to the application note, “RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)” for details.

Table 2.4 Configuration options (r_bsp_config.h)

Configuration options in r_bsp_config.h	
BSP_CFG_RTOS_USED Note: The default is 0.	Specifies the type of real-time operating system. When using this FIT module, set the following: FreeRTOS:1, AzureRTOS:5.

The configuration option settings of the RTOS used by the FIT module are contained in src/frtos_config/FreeRTOSConfig.h when FreeRTOS used or libs/threadx/tx_user.h when AzureRTOS used. The name of the option for FreeRTOS and its setting value is listed in Table 2.5. The names of the options for AzureRTOS and their setting values are listed in Table 2.6. Set the suitable value, as used RTOS.

Table 2.5 Configuration options (FreeRTOSConfig.h)

Configuration options in FreeRTOSConfig.h	
configTICK_RATE_HZ Note: The default is “(TickType_t)1000”.	Specifies RTOS tick interrupt cycle. When using this FIT module, set this option to “(TickType_t)1000”.

Table 2.6 Configuration options (tx_user.h)

Configuration options in tx_user.h	
USE_TX_TIMER_TICKS_PER_SECOND The default is 0.	Enables the user setting for RTOS tick interrupt cycle. When using this FIT module, set this option to 1.
TX_TIMER_TICKS_PER_SECOND The default is 100.	Specifies RTOS tick interrupt cycle. When using this FIT module, set this option to 1000.

2.8 Code Size

Table 2.5 lists the ROM size, RAM size, and maximum stack size used by the FIT module. The ROM (code and constants) and RAM (global data) sizes are determined by the configuration options specified at build time (see 2.7, Compile Settings).

The values listed in Table 2.5 have been confirmed under the following conditions.

FIT module revision: r_cellular rev1.06

Compiler version: Renesas Electronics C/C++ Compiler Package for RX Family V3.04.00 ("-lang = c99" option added to default settings of integrated development environment)

Configuration options: Default settings

Table 2.7 Code Sizes

ROM, RAM and Stack Code Sizes			
Device	**Category**	**Memory Used**	**Remarks**
RX65N RX72N	ROM	Approx. 30 KB	-
	RAM	Approx. 600 bytes	-
	Max. stack size used	Approx. 700 bytes	-

2.9 Parameters

This section describes the parameter structures used by the API functions in this module. The structures are defined in r_cellular_if.h.

Management structure (used by all APIs)

- st_cellular_ctrl_t

Configuration structure (used by R_CELLULAR_Open())

- st_cellular_cfg_t

Structure for setting to connect to access point (used by R_CELLULAR_APConnect())

- st_cellular_ap_cfg_t

Structures for setting and obtaining the time (used by R_CELLULAR_GetTime() and R_CELLULAR_SetTime())

- st_cellular_datetime_t

Structures for obtaining the ICCID (used by R_CELLULAR_GetICCID())

- st_cellular_iccid_t

Structures for obtaining the IMEI (used by R_CELLULAR_GetIMEI())

- st_cellular_imei_t

Structures for obtaining the IMSI (used by R_CELLULAR_GetIMSI())

- st_cellular_imsi_t

Structures for obtaining the phone number (used by R_CELLULAR_GetPhonenum())

- st_cellular_phonenum_t

Structures for obtaining the RSSI (used by R_CELLULAR_GetRSSI())

- st_cellular_rssi_t

Structures for obtaining the SVN (used by R_CELLULAR_GetSVN())

- st_cellular_svn_t

Structures for obtaining the Cellular module's access point connection status (used by R_CELLULAR_GetAPConnectState())

- st_cellular_notice_t

2.10 Return Values

The APIs returns enumerated types listed below. The enumerated types of return values are defined in r_cellular_if.h.

API error codes:

- e_cellular_err_t

2.11 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) or (5) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to "RX Smart Configurator User's Guide: e^2 studio (R20AN0451)" for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to "RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)" for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to "RX Smart Configurator User Guide: CS+ (R20AN0470)" for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to "RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)" for details.

(5) Adding the FIT module to your project using the Smart Configurator in IAREW
By using the Smart Configurator Standalone version, the FIT module is automatically added to your project. Refer to "RX Smart Configurator User Guide: IAREW (R20AN0535)" for details.

2.12 RTOS Usage Requirement

The FIT module utilizes RTOS functionality.

3. API Functions

3.1 R_CELLULAR_Open()

Initializes the RYZ014A Cellular FIT module and the Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_Open (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_cfg_t * const p_cfg
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_cfg (IN)	*Pointer to st_cellular_cfg_t structure defined by the user*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_ALREADY_OPEN	*/* R_CELLULAR_Open has already been run */*
CELLULAR_ERR_SERIAL_OPEN	*/* Serial initialization failure */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_SEMAPHORE_INIT	*/* Semaphore initialization failure */*
CELLULAR_ERR_EVENT_GROUP_INIT	*/* Event group initialization failure */*
CELLULAR_ERR_CREATE_TASK	*/* Failure creating task */*
CELLULAR_ERR_MEMORY_ALLOCATION	*/* Memory allocation failure */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Initializes the RYZ014A Cellular FIT module and the Cellular module to prepare for wireless communication.

If *p_cfg* is set to NULL, the default values of the FIT module and the values set by Smart Configurator are used.

Default values used when *p_cfg* is set to NULL:

<baud_rate = 921600 / ap_gatt_retry_count = 100 / sci_timeout = 10000 / tx_process_size = 1500

/ rx_process_size = 1500 / packet_data_size = 0 / exchange_timeout = 60

/ connect_timeout = 200 / send_timeout = 10 / creatable_socket = 6>

Reentrant

Reentrant operation is not possible.

Examples

[Using default configuration values]

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
```

[Setting new configuration values]

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctr = {0};
st_cellular_cfg_t cellular_cfg = {
                        "0000",      // PIN code of SIM card
                        921600,      // Baud rate for communication with module
                                        (921600 recommended)
                        0,           // Retry limit when connecting to access point
                        0xffff,      // MCU communication timeout setting
                        100,         // Data size of single transmission to Cellular module
                        100,         // Data size of single reception from Cellular module
                        100,         // Data size per packet
                        100,         // Exchange timeout
                        100,         // Socket connection timeout
                        100,         // Packet transmission timeout
                        3};          // Maximum socket creation count
ret = R_CELLULAR_Open(&cellular_ctrl, &cellular_cfg);
```

Special Notes

None

3.2 R_CELLULAR_Close()

Closes communication with the Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_Close (
    st_cellular_ctrl_t * const p_ctrl
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Closes communication with the Cellular module.

When a connection has been established to an access point, the connection to the access point is closed. When communication is taking place via a socket, the connection to the socket and the connection to the access point are closed.

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_Close(&cellular_ctrl);
```

Special Notes

None

3.3 R_CELLULAR_APConnect()

Connects the Cellular module to an access point.

Format

```
e_cellular_err_t R_CELLULAR_APConnect (
    st_cellular_ctrl_t * const p_ctrl
    const st_cellular_ap_cfg_t * const p_ap_cfg)
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_ap_cfg (IN)	*Pointer to st_cellular_ap_cfg_t structure defined by the user*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_ALREADY_CONNECT	*/* Already connected to access point */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Connects the Cellular module to an access point.

In order to connect to an access point, it is necessary to configure the required information, including access point information, before running this function, using either of the following methods.

(1) Configure the macros in r_cellular _config.h listed in Table 3.1 with appropriate values.
(Refer to Table 2.1 for details.)
Note1: In this case, set *p_ap_cfg* to NULL.
Note2: Configure macro settings in Smart Configurator. Do not edit macro values directly.

(2) While R_CELLULAR_Open() is running, set as the second argument a pointer to the st_cellular_cfg_t structure, with appropriate values configured for the members listed in Table 3.2. Then, while R_CELLULAR_APConnect() running, set as the second argument a pointer to the st_cellular_ap_cfg_t structure, with appropriate values configured for the members listed in Table 3.3.

The number of connection retries is the value of the CELLULAR_CFG_AT_COMMAND_RETRY_GATT macro or the value of the ap_gatt_retry_count member in the st_cellular_cfg_tstructure, with one-second intervals in between.

When a connection is successfully established to the access point, the network time is obtained automatically and stored in the nonvolatile memory of the Cellular module (RYZ014A). To obtain the time, run R_CELLULAR_GetTime().

Before running this API, run R_CELLULAR_Open() to initialize the RYZ014A Cellular FIT module and the Cellular module. If R_CELLULAR_Open() has not been run, a value of CELLULAR_ERR_NOT_OPEN is returned.

Table 3.1 Macros Required to Connect to an Access Point

Configuration options in r_cellular _config.h	
CELLULAR_CFG_AP_NAME	Connection destination access point name
CELLULAR_CFG_AP_USERID	Username of connection destination access point
CELLULAR_CFG_AP_PASSWORD	Password of connection destination access point
CELLULAR_CFG_PIN_CODE	PIN code of SIM card used
CELLULAR_CFG_AUTH_TYPE	Authentication protocol (0: None, 1: PAP, 2: CHAP)
CELLULAR_CFG_AT_COMMAND_RETRY_GATT	Maximum number of retries when connecting to an access point

Table 3.2 Members Required to Connect to an Access Point (R_CELLULAR_Open())

Members in st_cellular_cfg_t structure	
uint8_t sim_pin_code	PIN code of SIM card used
uint8_t ap_gatt_retry_count	Maximum number of retries when connecting to an access point

Table 3.3 Members Required to Connect to an Access Point (R_CELLULAR_APConnect())

Members in st_cellular_ap_cfg_t structure	
uint8_t ap_name	Connection destination access point name
uint8_t ap_user_name	Username of connection destination access point
uint8_t ap_pass	Password of connection destination access point
uint8_t auth_type	Authentication protocol (0: None, 1: PAP, 2: CHAP)

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
```

Special Notes

None

3.4 R_CELLULAR_IsConnected()

Obtains the FIT module's access point connection status.

Format

```
e_cellular_err_t R_CELLULAR_IsConnected (
    st_cellular_ctrl_t * const p_ctrl
)
```

Parameters

p_ctrl (IN/OUT) *Pointer to st_cellular_ctrl_t structure defined by the user*

Return values

CELLULAR_SUCCESS	*/* Connected to access point */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the FIT module's access point connection status.

Reentrant

Reentrant operation is possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_IsConnect(&cellular_ctrl);
```

Special Notes

None

3.5 R_CELLULAR_Disconnect()

Disconnects the Cellular module from an access point.

Format

```
e_cellular_err_t R_CELLULAR_Disconnect (
    st_cellular_ctrl_t * const p_ctrl
)
```

Parameters

p_ctrl (IN/OUT) *Pointer to st_cellular_ctrl_t structure defined by the user*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Disconnects the Cellular module from an access point.

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
ret = R_CELLULAR_Disconnect(&cellular_ctrl);
```

Special Notes

None

3.6 R_CELLULAR_Createsocket()

Creates a socket.

Format

```
int32_t R_CELLULAR_CreateSocket (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t protocol_type,
    const uint8_t ip_version
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
protocol_type (IN)	*Protocol type (specified as TCP = 6)*
ip_version (IN)	*IP version (specified as IPv4 = 4)*

Return values

Value of 1 to 6	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_SOCKET_CREATE_LIMIT	*/* Socket creation limit exceeded */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Sets the protocol type and IP version of a usable socket. For *protocol_type*, specify CELLULAR_SOCKET_IP_PROTOCOL_TCP (6), and for *ip_version*, specify CELLULAR_SOCKET_IP_VERSION_4 (4).

When the function is completed successfully, a socket is created and a number is returned as the return value. The number of the created socket is an integer value between 1 and 6.

Before running this API, run R_CELLULAR_APConnect() to connect to an access point. If R_CELLULAR_APConnect() has not been run, a value of CELLULAR_ERR_NOT_CONNECT is returned.

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
ret = R_CELLULAR_CreateSocket (&cellular_ctrl, CELLULAR_SOCKET_IP_PROTOCOL_TCP ,
                                    CELLULAR_SOCKET_IP_VERSION_4);
```

Special Notes

None

3.7 R_CELLULAR_Connectsocket()

Connects to the specified IP address and port.

Format

```
e_cellular_err_t R_CELLULAR_ConnectSocket (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t socket_no,
    const uint32_t ip_address,
    const uint16_t port
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
socket_no (IN)	*Socket number*
ip_address (IN)	*Connection destination IP address (expressed as a 32-bit value for IPv4)*
port *(IN)*	*Connection destination port number*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_ALREADY_SOCKET_CONNECT	*/* Already connected to socket */*
CELLULAR_ERR_SOCKET_NOT_READY	*/* Socket is in error status */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Using the socket specified by *socket_no*, connects to the IP address specified by *ip_address* and the port number specified by *port*. The IP address should be expressed as a 32-bit value for IPv4. To convert the address, use the CELLULAR_IP_ADDER_CONVERT macro.

Before running this API, run R_CELLULAR_CreateSocket() to create a socket. If R_CELLULAR_CreateSocket() has not been run, a value of CELLULAR_ERR_SOCKET_NOT_READY is returned.

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
int8_t socket_no;
uint16_t port_no = 33333;
uint32_t ip_adder = CELLULAR_IP_ADDER_CONVERT(192,168,0,10);

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, 6, 4);
if (0 < socket_no)
{
    ret = R_CELLULAR_ConnectSocket(&p_ctrl, socket_no, ip_adder, port_no);
}
```

Special Notes

None

3.8 R_CELLULAR_ShutdownSocket()

Shuts down socket communication.

Format

```
e_cellular_err_t R_CELLULAR_ShutdownSocket (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t socket_no
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
socket_no (IN)	*Socket number*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_SOCKET_NOT_READY	*/* Socket is in error status */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Shuts down communication on the socket specified by *socket_no*.

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
int8_t socket_no;
uint16_t port_no = 33333;
uint32_t ip_adder = CELLULAR_IP_ADDER_CONVERT(192,168,0,10);

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, 6, 4);
if (0 < socket_no)
{
    ret = R_CELLULAR_ConnectSocket(&p_ctrl, socket_no, ip_adder, port_no);
    ret = R_CELLULAR_ShutdownSocket(&p_ctrl, socket_no);
}
```

Special Notes

None

3.9 R_CELLULAR_CloseSocket()

Closes a socket.

Format

```
e_cellular_err_t R_CELLULAR_CloseSocket (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t socket_no
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
socket_no (IN)	*Socket number*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_SOCKET_NOT_READY	*/* Socket is in error status */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Closes the socket specified by *socket_no*. If communication is in progress on the socket, the socket is disconnected.

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
int8_t socket_no;
uint16_t port_no = 33333;
uint32_t ip_adder = CELLULAR_IP_ADDER_CONVERT(192,168,0,10);

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, 6, 4);
if (0 < socket_no)
{
    ret = R_CELLULAR_ConnectSocket(&p_ctrl, socket_no, ip_adder, port_no);
    ret = R_CELLULAR_CloseSocket(&p_ctrl, socket_no);
}
```

Special Notes

None

3.10 R_CELLULAR_SendSocket()

Transmits data via the specified socket.

Format

```
e_cellular_err_t R_CELLULAR_SendSocket (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t socket_no,
    uint8_t * const data,
    const int32_t length,
    const uint32_t timeout_ms
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
socket_no (IN)	*Socket number*
data (IN)	*Pointer to transmit data*
length (IN)	*Transmit data size (1 or more)*
timeout_ms (IN)	*Timeout setting (ms units, setting value = 0 to 0xffffffff, 0 = no timeout)*

Return values

Number of bytes transmitted	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_SOCKET_NOT_READY	*/* Socket is in error status */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Transmits the number of bytes specified by *length* of the data stored in *data* from the socket specified by *socket_no*.

Before running this API, run R_CELLULAR_ConnectSocket(). If R_CELLULAR_ConnectSocket() has not been run, a value of CELLULAR_ERR_SOCKET_NOT_READY is returned.

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
int8_t socket_no;
uint16_t port_no = 33333;
uint32_t ip_adder = CELLULAR_IP_ADDER_CONVERT(192,168,0,10);
uint8_t data[] = "TEST";
int32_t length = 4;

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, 6, 4);
if (0 < socket_no)
{
    ret = R_CELLULAR_ConnectSocket(&p_ctrl, socket_no, ip_adder, port_no);
    ret = R_CELLULAR_SendSocket(&p_ctrl, socket_no, data, length, timeout);
}
```

Special Notes

None

3.11 R_CELLULAR_ReceiveSocket()

Receives data via the specified socket.

Format

```
e_cellular_err_t R_CELLULAR_ReceiveSocket (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t socket_no,
    uint8_t * const data,
    const int32_t length,
    const uint32_t timeout_ms
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
socket_no (IN)	*Socket number*
data (IN)	*Pointer to transmit data*
length (IN)	*Transmit data size (1 or more)*
timeout_ms (IN)	*Timeout setting (ms units, setting value = 0 to 0xffffffff, 0 = no timeout)*

Return values

Number of bytes transmitted	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_SOCKET_NOT_READY	*/* Socket is in error status */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Acquires and stores the number of received bytes specified by *length* of the data in the receive data area *data* from the socket specified by *socket_no*.

Before running this API, run R_CELLULAR_ConnectSocket(). If R_CELLULAR_ConnectSocket() has not been run, a value of CELLULAR_ERR_SOCKET_NOT_READY is returned.

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
int8_t socket_no;
uint16_t port_no = 33333;
uint32_t ip_adder = CELLULAR_IP_ADDER_CONVERT(192,168,0,10);
uint8_t data[100] = {0};
int32_t length = 100;

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, 6, 4);
if (0 < socket_no)
{
    ret = R_CELLULAR_ConnectSocket(&p_ctrl, socket_no, ip_adder, port_no);
    ret = R_CELLULAR_ReceiveSocket(&p_ctrl, socket_no, data, length, timeout);
}
```

Special Notes

None

3.12 R_CELLULAR_DnsQuery()

Executes a DNS query.

Format

```
e_cellular_err_t R_CELLULAR_DnsQuery (
    st_cellular_ctrl_t * const p_ctrl,
    uint8_t *const domain_name,
    uint32_t * const ip_address
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
domain_name (IN)	*Pointer to domain name storage area*
ip_address (IN/OUT)	*Pointer to storage area for acquired IP address*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Executes a DNS query, obtains the IP address of the domain specified by *domain_name*, and saves it in *ip_address*.

Before running this API, run R_CELLULAR_APConnect() to connect to an access point. If R_CELLULAR_APConnect() has not been run, a value of CELLULAR_ERR_NOT_CONNECT is returned.

Reentrant

Reentrant operation is possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
uint8_t domain_name[] = “Renesas.com”;
uint32_t ip_address = 0;
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
ret = R_CELLULAR_DnsQuery (&cellular_ctrl, domain_name, &ip_address);
```

Special Notes

None

3.13 R_CELLULAR_GetTime()

Obtains the date and time information stored in the Cellular module (RYZ014A).

Format

```
e_cellular_err_t R_CELLULAR_GetTime (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_datetime_t * const p_time
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_time (IN/OUT)	*Pointer to structure for storing acquired date and time*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the date and time information setting of the Cellular module (RYZ014A) and saves it in the st_cellular_datetime_t structure referenced by the pointer *p_time*.

The value of the date and time information is “70/01/01/00:00:00+00” (year/month/date/hour:minute:second:time zone) when the Cellular module is activated. To use date and time information, run R_CELLULAR_SetTime() to obtain the current date and time.

Reentrant

Reentrant operation is possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_datetime_t cellular_time = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetTime(&cellular_ctrl, &cellular_time);
```

Special Notes

None

3.14 R_CELLULAR_SetTime()

This function configures the date and time information setting of the Cellular module (RYZ014A).

Format

```
e_cellular_err_t R_CELLULAR_SetTime (
    st_cellular_ctrl_t * const p_ctrl,
    const st_cellular_datetime_t * const p_time
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_time (IN/OUT)	*Pointer to structure for storing acquired date and time*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Configures the Cellular module (RYZ014A) with the date and time information stored in *p_time*.

If this function is run before connecting to an access point, the date and time information set by the function is overwritten by the network time obtained automatically when a connection to an access point is established.

Reentrant

Reentrant operation is possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_datetime_t cellular_time = {21,8,1,12,34,56,36};
                              // 2021, August, 1st, 12:34:56 + 36
                              // The time zone is specified in 15-minute units.
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_SetTime(&cellular_ctrl, &cellular_time);
```

Special Notes

None

3.15 R_CELLULAR_SetEDRX()

Configures the extended discontinuous reception (eDRX) parameter settings of the Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_SetEDRX (
    st_cellular_ctrl_t * const p_ctrl,
    const e_cellular_edrx_mode_t mode,
    const e_cellular_edrx_cycle_t edrx,
    const e_cellular_ptw_cycle_t ptw
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
mode (IN)	*Operating mode*
edrx (IN)	*eDRX cycle setting value*
ptw (IN)	*PTW cycle setting value*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Configures eDRX parameter settings in the Cellular module.

The values that may be set for the argument *mode* are as follows.

```
typedef enum
{
   CELLULAR_EDRX_MODE_INVALID = 0,          // Disable the edrx function
   CELLULAR_EDRX_MODE_ACTIVE = 1,           // Enable the edrx function
   CELLULAR_EDRX_MODE_ACTIVE_RESULT = 2,    // Activate the edrx function and return the results
   CELLULAR_EDRX_MODE_INIT = 3,             // Initialize and disable the edrx function
} e_cellular_edrx_mode_t;
```

The values that may be set for the argument *edrx* (eDRX cycle) are as follows.

```
typedef enum
{
    CELLULAR_EDRX_CYCLE_5_SEC = 0,      // edrx cycle (5.12sec)
    CELLULAR_EDRX_CYCLE_10_SEC,         // edrx cycle (10.24sec)
    CELLULAR_EDRX_CYCLE_20_SEC,         // edrx cycle (20.48sec)
    CELLULAR_EDRX_CYCLE_40_SEC,         // edrx cycle (40.96sec)
    CELLULAR_EDRX_CYCLE_81_SEC,         // edrx cycle (81.92sec)
    CELLULAR_EDRX_CYCLE_163_SEC,        // edrx cycle (163.84sec)
    CELLULAR_EDRX_CYCLE_327_SEC,        // edrx cycle (327.68sec)
    CELLULAR_EDRX_CYCLE_655_SEC,        // edrx cycle (655.36sec)
    CELLULAR_EDRX_CYCLE_1310_SEC,       // edrx cycle (1,310.72sec)
    CELLULAR_EDRX_CYCLE_2621_SEC        // edrx cycle (2,621.44sec)
} e_cellular_edrx_cycle_t;
```

The values that may be set for the argument *ptw* (paging time window cycle) are as follows.

```
typedef enum
{
    CELLULAR_PTW_CYCLE_NONE = 0,        // PTW (PTW not used)
    CELLULAR_PTW_CYCLE_1_SEC,           // PTW (1sec)
    CELLULAR_PTW_CYCLE_2_SEC,           // PTW (2sec)
    CELLULAR_PTW_CYCLE_3_SEC,           // PTW (3sec)
    CELLULAR_PTW_CYCLE_4_SEC,           // PTW (4sec)
    CELLULAR_PTW_CYCLE_5_SEC,           // PTW (5sec)
    CELLULAR_PTW_CYCLE_6_SEC,           // PTW (6sec)
    CELLULAR_PTW_CYCLE_7_SEC,           // PTW (7sec)
    CELLULAR_PTW_CYCLE_8_SEC,           // PTW (8sec)
    CELLULAR_PTW_CYCLE_9_SEC,           // PTW (9sec)
    CELLULAR_PTW_CYCLE_10_SEC,          // PTW (10sec)
    CELLULAR_PTW_CYCLE_12_SEC,          // PTW (12sec)
    CELLULAR_PTW_CYCLE_14_SEC,          // PTW (14sec)
    CELLULAR_PTW_CYCLE_16_SEC,          // PTW (16sec)
    CELLULAR_PTW_CYCLE_18_SEC,          // PTW (18sec)
    CELLULAR_PTW_CYCLE_20_SEC           // PTW (20sec)
} e_cellular_ptw_cycle_t;
```

Reentrant

Reentrant operation is not possible.

Examples

Setting the eDRX cycle to 20 seconds and PTW to 2 seconds (For details of setting values, refer to the comments in r_cellular_if.h.)

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_SetEDRX(&cellular_ctrl, CELLULAR_EDRX_MODE_ACTIVE
                        CELLULAR_EDRX_CYCLE_20_SEC, CELLULAR_PTW_CYCLE_2_SEC);
```

Special Notes

Figure 3.1 shows the eDRX parameters corresponding to the setting values of this function.

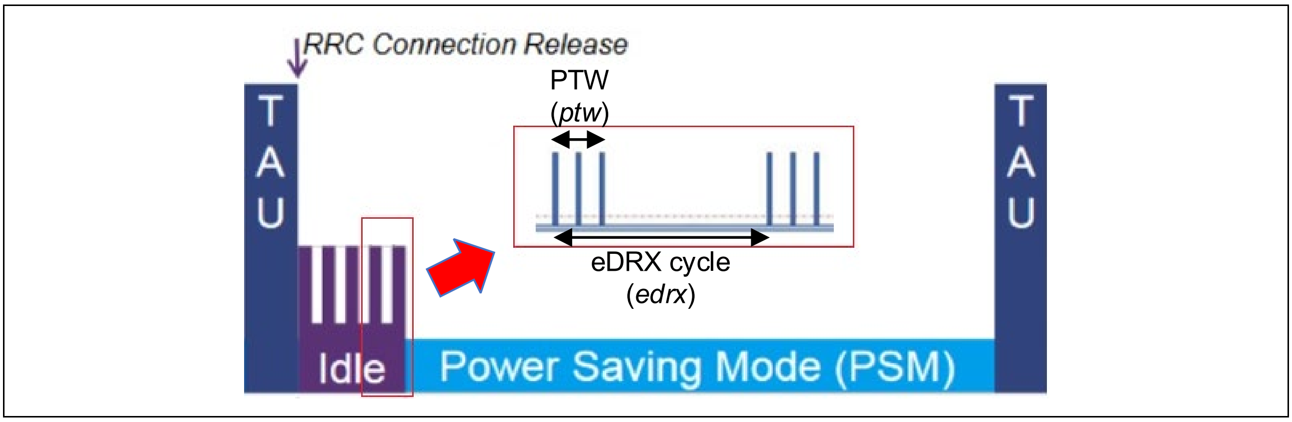

Figure 3.1 eDRX parameter

3.16 R_CELLULAR_SetPSM()

Configures the power saving mode (PSM) settings of the Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_SetPSM (
    st_cellular_ctrl_t * const p_ctrl,
    const e_cellular_psm_mode_t mode,
    const e_cellular_tau_cycle_t tau,
    const e_cellular_cycle_multiplier_t tau_multiplier,
    const e_cellular_active_cycle_t active,
    const e_cellular_cycle_multiplier_t active_multiplier
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
mode (IN)	*Operating mode*
tau (IN)	*TAU cycle setting value*
tau_multiplier (IN)	*Multiplier of TAU cycle setting value*
active (IN)	*Active time cycle setting value*
active_multiplier (IN)	*Multiplier of active time cycle setting value*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Configures the PSM parameters of the Cellular module.

The values set in the Cellular module are calculated using the following formula.

$$TAU = tau \times tau\_multiplier$$
$$ActiveTime = active \times active\_multiplier$$

The values that may be set for the argument *mode* are as follows.

```
typedef enum
{
   CELLULAR_PSM_MODE_INVALID        = 0,     // Disable the PSM function
   CELLULAR_PSM_MODE_ACTIVE         = 1,     // Enable the PSM function
   CELLULAR_PSM_MODE_INIT           = 2,     // Initialize and disable the PSM function
} e_cellular_psm_mode_t;
```

The values that may be set for the argument *tau* (tracking area update cycle) are as follows.

```
typedef enum
{
    CELLULAR_TAU_CYCLE_10_MIN = 0,          // TAU cycle (10min)
    CELLULAR_TAU_CYCLE_1_HOUR,              // TAU cycle (1hour)
    CELLULAR_TAU_CYCLE_10_HOUR,             // TAU cycle (10hour)
    CELLULAR_TAU_CYCLE_2_SEC,               // TAU cycle (2sec)
    CELLULAR_TAU_CYCLE_30_SEC,              // TAU cycle (30sec)
    CELLULAR_TAU_CYCLE_1_MIN,               // TAU cycle (1min)
    CELLULAR_TAU_CYCLE_320_HOUR,            // TAU cycle (320hour)
    CELLULAR_TAU_CYCLE_NONE,                // TAU cycle (Timer is deactivated)
} e_cellular_tau_cycle_t;
```

The values that may be set for the argument *active* (active time cycle) are as follows.

```
typedef enum
{
    CELLULAR_ACTIVE_CYCLE_2_SEC = 0,        // Active time (2sec)
    CELLULAR_ACTIVE_CYCLE_1_MIN,            // Active time (1min)
    CELLULAR_ACTIVE_CYCLE_6_MIN,            // Active time (6min)
    CELLULAR_ACTIVE_CYCLE_NONE,             // Active time (Timer is deactivated)
} e_cellular_active_cycle_t;
```

The values that may be set for the arguments *tau_multiplier* and *active_multiplier* are as follows.

```
typedef enum
{
    CELLULAR_CYCLE_MULTIPLIER_0 = 0,        // Multiplier 0
    CELLULAR_CYCLE_MULTIPLIER_1 = 1,        // Multiplier 1
    CELLULAR_CYCLE_MULTIPLIER_2 = 2,        // Multiplier 2
                        :
    CELLULAR_CYCLE_MULTIPLIER_30 = 30,      // Multiplier 30
    CELLULAR_CYCLE_MULTIPLIER_31 = 31,      // Multiplier 31
} e_cellular_cycle_multiplier_t;
```

Reentrant

Reentrant operation is not possible.

Examples

Setting the TAU to 10 minutes and the active time to 1 minute (For details of setting values, refer to the comments in r_cellular_if.h.)

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_SetPSM(&cellular_ctrl, CELLULAR_PSM_MODE_ACTIVE,
                  CELLULAR_TAU_CYCLE_10_MIN, CELLULAR_CYCLE_MULTIPLIER_1,
                  CELLULAR_ACTIVE_CYCLE_1_MIN, CELLULAR_CYCLE_MULTIPLIER_1);
```

- Argument *tau* set to 10 minutes and argument *tau_multiplier* set to a multiplier of 1 = 10 minutes × 1 = 10 minutes
- Argument *active* set to 1 minute and argument *active_multiplier* set to a multiplier of 1 = 1 minute × 1 = 1 minute

Special Notes

Figure 3.2 shows the PSM parameters corresponding to the setting values of this function.

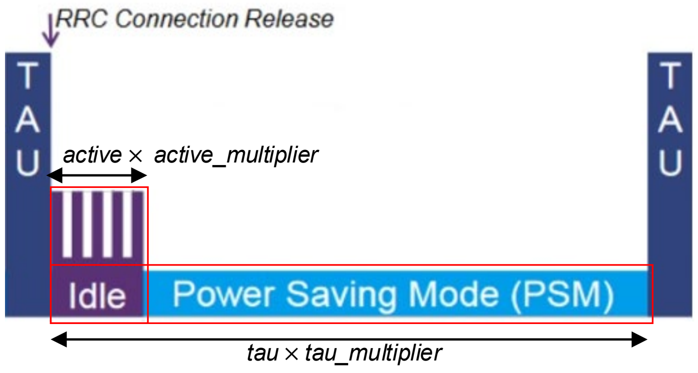

Figure 3.2 PSM parameter

3.17 R_CELLULAR_GetICCID()

Obtains the IC card identifier (ICCID) assigned to the SIM card used.

Format

```
e_cellular_err_t R_CELLULAR_GetICCID (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_iccid_t * const p_iccid
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_iccid (IN/OUT)	*Pointer to structure for storing acquired ICCID*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the ICCID from the SIM card used and saves it in the st_cellular_iccid_t structure referenced by the pointer *p_iccid*.

Reentrant

Reentrant operation is possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_iccid_t cellular_iccid = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetICCID(&cellular_ctrl, &cellular_iccid);
```

Special Notes

None

3.18 R_CELLULAR_GetIMEI()

Obtains the international mobile equipment identifier (IMEI) assigned to the RYZ014A Cellular module used.

Format

```
e_cellular_err_t R_CELLULAR_GetIMEI (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_imei_t * const p_imei
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_imei (IN/OUT)	*Pointer to structure for storing acquired IMEI*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the IMEI from the Cellular module used and saves it in the st_cellular_imei_t structure referenced by the pointer *p_imei*.

Reentrant

Reentrant operation is possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_imei_t cellular_imei = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetIMEI(&cellular_ctrl, &cellular_imei);
```

Special Notes

None

3.19 R_CELLULAR_GetIMSI()

Obtains the international mobile subscriber identity (IMSI) assigned to the SIM card used.

Format

```
e_cellular_err_t R_CELLULAR_GetIMSI (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_imsi_t * const p_imsi
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_imsi (IN/OUT)	*Pointer to structure for storing acquired IMSI*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the IMSI from the SIM card used and saves it in the st_cellular_imsi_t structure referenced by the pointer *p_imsi*.

Reentrant

Reentrant operation is possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_imsi_t cellular_imsi = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetIMSI(&cellular_ctrl, &cellular_imsi);
```

Special Notes

None

3.20 R_CELLULAR_GetPhonenum()

Obtains the phone number associated with the SIM card used.

Format

```
e_cellular_err_t R_CELLULAR_GetPhonenum (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_phonenum_t * const p_phonenum
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_phonenum (IN/OUT)	*Pointer to structure for storing acquired phone number*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the phone number from the SIM card used and saves it in the st_cellular_phonenum_t structure referenced by the pointer *p_phonenum*.

Reentrant

Reentrant operation is possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_phonenum_t cellular_phonenum = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetPhonenum(&cellular_ctrl, &cellular_phonenum);
```

Special Notes

None

3.21 R_CELLULAR_GetRSSI()

Obtains received strength indication (RSSI) and channel bit error rate (BER) from the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_GetRSSI (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_rssi_t * const p_rssi
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_rssi (IN/OUT)	*Pointer to structure for storing acquired RSSI and BER*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains RSSI and BER from the Cellular module and saves it in the st_cellular_rssi_t structure referenced by the pointer *p_rssi*.
If RSSI and BER are not known or not detectable due to such factors as the Cellular module is not connected to an access point, “99” is acquired.

Reentrant

Reentrant operation is possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_rssi_t cellular_rssi = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetRSSI(&cellular_ctrl, &cellular_rssi);
```

Special Notes

None

3.22 R_CELLULAR_GetSVN()

Obtains software version number (SVN) and revision information of the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_GetSVN (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_svn_t * const p_svn
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_svn (IN/OUT)	*Pointer to structure for storing acquired SVN and revision*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains SVN and revision information from the Cellular module and saves it in the st_cellular_svn_t structure referenced by the pointer *p_svn*.

Reentrant

Reentrant operation is possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_svn_t cellular_svn = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetSVN(&cellular_ctrl, &cellular_svn);
```

Special Notes

None

3.23 R_CELLULAR_Ping()

Pings the host.

Format

```
e_cellular_err_t R_CELLULAR_Ping (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t * const p_host,
    void(* const p_callback)(void * p_args)
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_host (IN)	*Pointer to structure for storing host name or host IP address*
p_callback (IN)	*Function pointer to receive ping status*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Pings the host. Either the host name or the host IP address, referenced by the pointer *p_host*, should be string type. Once registering callback function to *p_callback*, the ping status can be received.

Reentrant

Reentrant operation is not possible.

Examples

```
static void call_back (void * p_arg);
static void call_back(void * p_arg)
{
    st_cellular_ping_reply_t * cellular_ping_reply = NULL;
    cellular_ping_reply = (st_cellular_ping_reply_t *)p_arg;
}

e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
/* Setting a host name */
ret = R_CELLULAR_Ping(&cellular_ctrl, "renesas.com", call_back);
/* Setting an IP address */
ret = R_CELLULAR_Ping(&cellular_ctrl, "210.248.164.218", call_back);
```

Special Notes

None

3.24 R_CELLULAR_GetAPConnectState()

Obtains the RYZ014A Cellular module's access point connection status.

Format

```
e_cellular_err_t R_CELLULAR_GetAPConnectState (
    st_cellular_ctrl_t * const p_ctrl,
    const e_cellular_network_result_t level,
    st_cellular_notice_t * const p_result,
    void(* const p_callback)(void * p_args)
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
level(IN)	*Setting value to control the notification of occurrence of an event*
p_result (IN/OUT)	*Pointer to structure for storing the access point connection status*
p_callback (IN)	*Function pointer to receive notifications*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the Cellular module's access point connection status and saves it in the st_cellular_notice_t structure referenced by the pointer *p_result*. In addition, once setting *level* to enable the notification, a change in the Cellular module's access point connection status can be received via the callback function referenced by the pointer *p_callback*.

The values that may be set for the argument *level* are as follows.

```
typedef enum
{
    CELLULAR_DISABLE_NETWORK_RESULT_CODE = 0,
                  // Disable automatic notification of network connection status change
    CELLULAR_ENABLE_NETWORK_RESULT_CODE_LEVEL1,
                  // Enable automatic notification of network connection status change
    CELLULAR_ENABLE_NETWORK_RESULT_CODE_LEVEL2,
                  // Enable automatic detailed notification of network connection status change
} e_cellular_network_result_t;
```

Reentrant

Reentrant operation is not possible.

Examples

```
static void call_back (void * p_arg);
static void call_back(void * p_arg)
{
    st_cellular_cereg_reply_t * cellular_cereg_reply = NULL;
    cellular_cereg_reply = (st_cellular_cereg_reply_t *)p_arg;
}

e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_notice_t cellular_notice = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetAPConnectState(&cellular_ctrl,
            CELLULAR_ENABLE_NETWORK_RESULT_CODE_LEVEL2, &cellular_notice,
            call_back);
```

Special Notes

None

3.25 R_CELLULAR_GetCellInfo()

Obtains information on cells.

Format

```
e_cellular_err_t R_CELLULAR_GetCellInfo (
    st_cellular_ctrl_t * const p_ctrl,
    const e_cellular_info_type_t type,
    void(* const p_callback)(void * p_args)
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
type(IN)	*Setting value to configure the cell from which to report information.*
p_callback (IN)	*Function pointer to receive information for cells*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains information on the serving and neighbor cells and notifies the information via the callback function referenced by the pointer *p_callback*.

The values that may be set for the argument *type* are as follows.

```
typedef enum
{
    CELLULAR_INFO_TYPE0 = 0,    // Report information for the serving cell only
    CELLULAR_INFO_TYPE1 = 1,    // Report information for the intra-frequency cells only
    CELLULAR_INFO_TYPE2 = 2,    // Report information for the intra-frequency cells only
    CELLULAR_INFO_TYPE7 = 7,    // Report information for all cells
    CELLULAR_INFO_TYPE9 = 9,
                        // Report information for the serving cell only with RSRP/CINR on main antenna.
} e_cellular_info_type_t;
```

Reentrant

Reentrant operation is not possible.

Examples

```
static void call_back (void * p_arg);
static void call_back(void * p_arg)
{
st_cellular_cell_info_t * cellular_cell_info = NULL;
cellular_cell_info = (st_cellular_cell_info_t *)p_arg;
}

e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetCellInfo(&cellular_ctrl, CELLULAR_INFO_TYPE9, call_back);
```

Special Notes

None

3.26 R_CELLULAR_AutoConnectConfig()

Configures the autoconnect mode of the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_AutoConnectConfig (
    st_cellular_ctrl_t * const p_ctrl,
    e_cellular_auto_connect_t const type
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
type(IN)	*Autoconnect mode setting value*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Configures autoconnect mode of the Cellular module. When enabled, the Cellular module will automatically try to connect to an access point after each reboot. The setting is persistent across reboot.

The values that may be set for the argument *type* are as follows.

```
typedef enum
{
    CELLULAR_DISABLE_AUTO_CONNECT = 0,          // Disable automatic connection to an access point
    CELLULAR_ENABLE_AUTO_CONNECT,
                                                // Enables automatic connection to an access point (next start-up or reboot)
} e_cellular_auto_connect_t;
```

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_AutoConncetConfig(&cellular_ctrl,
                                    CELLULAR_ENABLE_AUTO_CONNECT);
```

Special Notes

None

3.27 R_CELLULAR_SoftwareReset()

Resets the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_SoftwareReset (
    st_cellular_ctrl_t * const p_ctrl,
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_ RUNNING	*/* Another AT command is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Resets the Cellular module with the hardware reset AT command.
The status of the FIT module after reboot depends on the autoconnect setting configured by R_CELLULAR_AutoConnectConfig(). If autoconnect is disabled, the FIT module goes to the status after completion of R_CELLULAR_Open(), called **Connect to Cellular module** status in Figure 1.3. If autoconnect is enabled, the Cellular module will automatically try to connect to an access point after this function is completed successfully, going to the status after completion of R_CELLULAR_APConnect(), called **Connected to access point** status in Figure 1.3. Refer to Figure 1.3 for the status transitions of the FIT module.

Note: To use autoconnect mode, R_CELLULAR_APConnect() must be completed once successfully.

Reentrant

Reentrant operation is not possible.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_AutoConncetConfig(&cellular_ctrl,
                                    CELLULAR_ENABLE_AUTO_CONNECT);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
ret = R_CELLULAR_SoftwareReset(&cellular_ctrl);
```

Special Notes

None

4. Appendices

4.1 Confirmed Operation Environment

The confirmed operation environment for the FIT module is listed in Table 4.1.

Table 4.1 Confirmed Operation Environment

Item		Contents
Integrated development environment		Renesas Electronics e² studio Ver.2022-04
C compiler	CC-RX	Renesas Electronics C/C++ Compiler for RX Family V3.04.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC	GCC for Renesas 8.3.0.202004-GNURX Toolchain
Endian order		Little endian
		Big endian
Revision of the module		Rev1.06
Board used		Renesas RX65N Cloud Kit (product No.: RTK5RX65N0SxxxxxBE)
		Renesas RX72N Envision Kit (product No.: RTK5RX72N0C00000BJ)
RTOS	FreeRTOS	202012.00
	Azure RTOS	v6.1.6_rel-rx-1.0.6
FIT	BSP FIT	Ver 6.20
	SCI FIT	Ver 3.50
	IRQ FIT	Ver 3.90

4.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file “platform.h”.

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:
  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”
- Using e² studio:
  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note “Board Support Package Module Using Firmware Integration Technology (R01AN1685)”.

(2) Q: I have added the FIT module to the project and built it. Then I got an error for when the configuration setting is wrong.

A: The setting in the file "r_cellular_config.h" may be wrong. Check the file "r_cellular_config.h". If there is a wrong setting, set the correct value for that. Refer to 2.7 Compile Settings for details.

5. Reference Documents

User's Manual: Hardware
(The latest versions can be downloaded from the Renesas Electronics website.)
Technical Update/Technical News
(The latest information can be downloaded from the Renesas Electronics website.)
User's Manual: Development Tools
RX Family CC-RX Compiler User's Manual (R20UT3248)
(The latest versions can be downloaded from the Renesas Electronics website.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.04	Mar. 31, 2022	-	First edition issued
1.05	Apr. 28, 2022	9	Added the following macro definitions in section 2.7 Compile Settings: - CELLULAR_CFG_EX_TIMEOUT
		11	Added Table 2.5 Configuration options. (FreeRTOSConfig.h). Added Table 2.6 Configuration options. (tx_user.h).
		18	Revised settings to connect to access point. (See section 3.3)
		19	Added setting authentication protocol. (See section 3.3)
1.06	Jun. 7, 2022	6	Added the following APIs to Table 1.1: - R_CELLULAR_GetICCID - R_CELLULAR_GetIMEI - R_CELLULAR_GetIMSI - R_CELLULAR_GetPhonenum - R_CELLULAR_GetRSSI - R_CELLULAR_GetSVN - R_CELLULAR_Ping - R_CELLULAR_GetAPConnectState - R_CELLULAR_GetCellInfo - R_CELLULAR_AutoConnectConfig - R_CELLULAR_SoftwareReset
		9	Added the following macro definitions in section 2.7 Compile Settings: - CELLULAR_CFG_AUTH_TYPE
		12	Code size is updated in 2.8 Code Size.
		13	Added the following structures in section 2.9 Parameters: - st_cellular_iccid_t - st_cellular_imei_t - st_cellular_imsi_t - st_cellular_phonenum_t - st_cellular_rssi_t - st_cellular_svn_t - st_cellular_notice_t
		45-60	Added the following APIs in section 3 API Functions: 3.17 R_CELLULAR_GetICCID 3.18 R_CELLULAR_GetIMEI 3.19 R_CELLULAR_GetIMSI 3.20 R_CELLULAR_GetPhonenum 3.21 R_CELLULAR_GetRSSI 3.22 R_CELLULAR_GetSVN 3.23 R_CELLULAR_Ping 3.24 R_CELLULAR_GetAPConnectState 3.25 R_CELLULAR_GetCellInfo 3.26 R_CELLULAR_AutoConnectConfig 3.27 R_CELLULAR_SoftwareReset

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Corporate Headquarters

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

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/.