R01AN6324EJ0111
Rev.1.11
May.31.23

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

RX65N/RX651 Group

RX66N Group

RX72M Group

RX72N Group

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

Related Documents

[1] Firmware Integration Technology User’s Manual (R01AN1833)
[2] RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
[3] Adding Firmware Integration Technology Modules to Projects (R01AN1723)
[4] Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)
[5] RX Smart Configurator User’s Guide: e² studio (R20AN0451)
[6] RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
[7] RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)
[8] RX Family IRQ Module Using Firmware Integration Technology (R01AN1668)

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 three 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 Cellular module. The Cellular module assumes functions essential to SSL communication such as protocol control and encryption used in firmware upgrade process called Firmware upgrade Over-The-Air (FOTA).
2. Implementation type B:
   Driver software supporting SSL communication functionality in addition to the functionality supported by implementation type A. The Cellular module handles processing for functions essential to SSL communication, such as protocol control and encryption.
3. Implementation type C:
   Driver software supporting Point to Point Protocol (PPP) communication functionality of the Cellular module.

This FIT module does not support Point to Point Protocol (PPP). In addition, it does not use the WDT feature of the Cellular module.

1.2.1 Connection with PMOD Expansion Board for RYZ014A

Examples of connections between RX65N Cloud Kit and PMOD Expansion Board for RYZ014A are shown in Figure 1.1. The PMOD_9 pin and the PMOD_10 pin are not used by the FIT module.

The FIT module can be used with the connection Type 1 and Type 2 described in **RYZ014 Module System Integration Guide** to connect the RYZ014A to external host MCU.

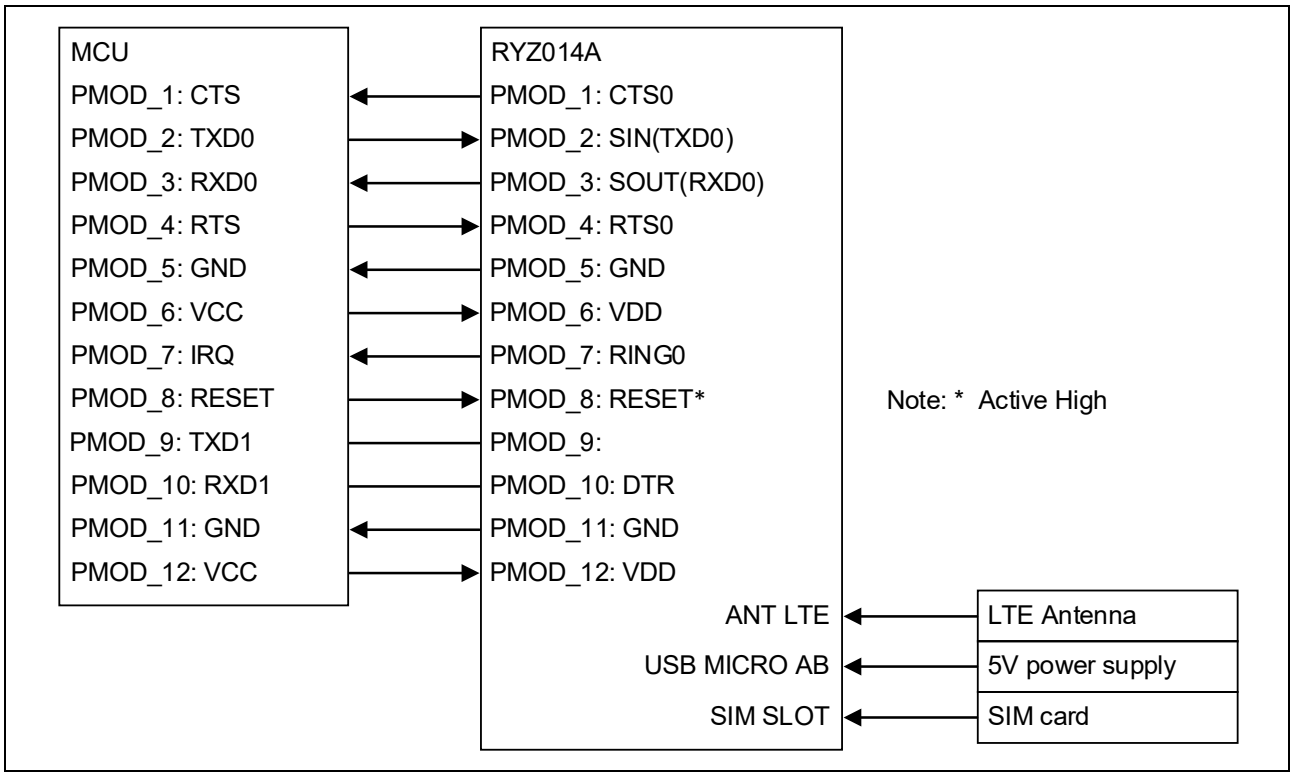

Figure 1.1 Example Connections between RX65N Cloud Kit and PMOD Expansion Board for RYZ014A

1.2.2 Software Configuration

Figure 1.2 shows the software configuration.

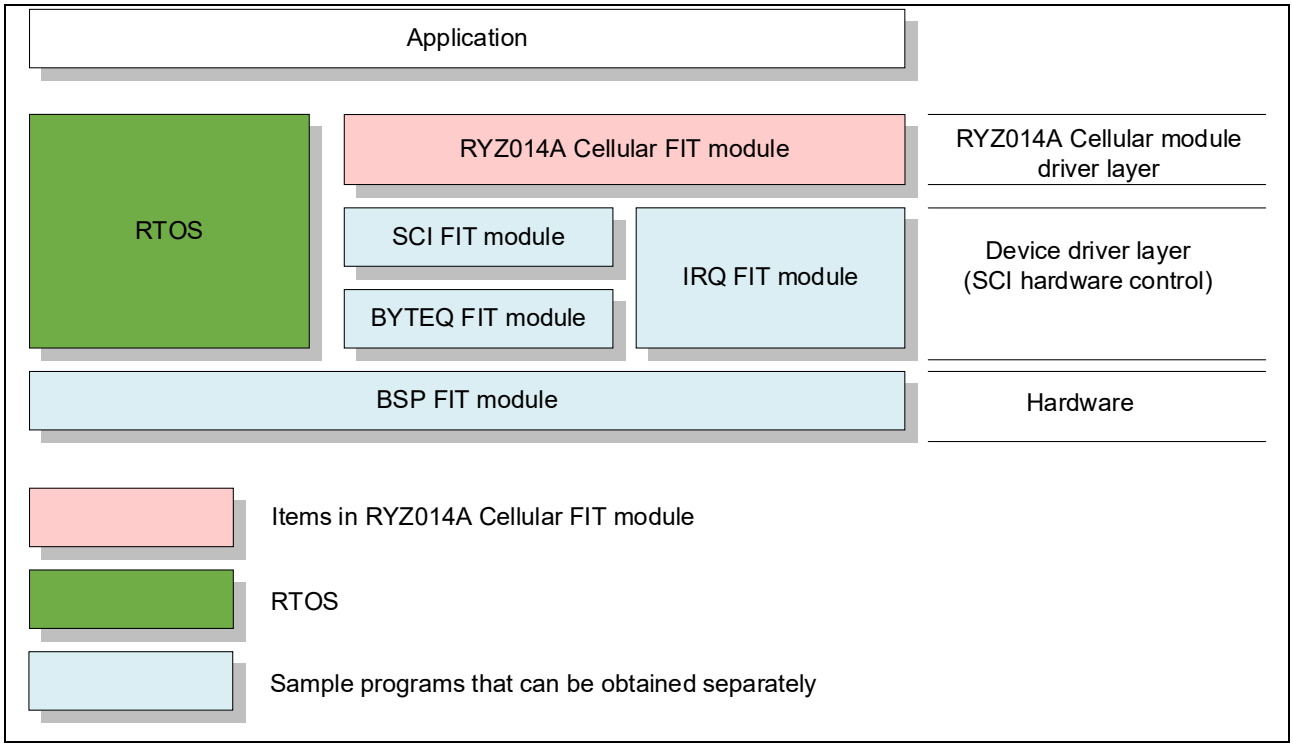

Figure 1.2 Software Configuration Diagram

(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.

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_GetEDRX	Obtains the extended discontinuous reception (eDRX) parameters used.
R_CELLULAR_SetPSM	Enables power saving mode (PSM).
R_CELLULAR_GetPSM	Obtains the power saving mode (PSM) parameters used.
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) and revision information.
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_SetOperator	Sets operator mode which configures the Cellular module according to the requirements of a specific network and/or carrier.
R_CELLULAR_SetBand	Configures list of 4G LTE bands the Cellular module is allowed to use.
R_CELLULAR_GetPDPAddress	Obtains a PDP address.
R_CELLULAR_FirmUpgrade	Initiates Firmware upgrade Over-The-Air (FOTA) to upgrade the Cellular module firmware.
R_CELLULAR_FirmUpgradeBlocking	Upgrades the Cellular module firmware with Firmware upgrade Over-The-Air (FOTA) configured using synchronous upgrade.
R_CELLULAR_GetUpgradeState	Obtains a status code of Firmware upgrade Over-The-Air (FOTA).
R_CELLULAR_UnlockSIM	Enter a PIN code.
R_CELLULAR_WriteCertificate	Writes a certificate or secret key in the non-volatile memory of the Cellular module.
R_CELLULAR_EraseCertificate	Removes a certificate or secret key from the non-volatile memory of the Cellular module.

R_CELLULAR_GetCertificate	Reads a certificate or secret key from the non-volatile memory of the Cellular module.
R_CELLULAR_ConfigSSLProfile	Configures SSL security profile.
R_CELLULAR_SoftwareReset	Resets the Cellular module with the hardware reset AT command.
R_CELLULAR_HardwareReset	Resets the Cellular module with the RESET pin.
R_CELLULAR_FactoryReset	Perform a factory reset to reset the Cellular module to the factory default configuration.
R_CELLULAR_RTS_Ctrl	Controls the RTS0 pin of the Cellular module.

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. The notes indicated by bold red font in the figure are processes that should be implemented in the user application to implement the connection manager described in section "2.2.8.1 RRC Idle" of **RYZ014 Module System Integration Guide**.

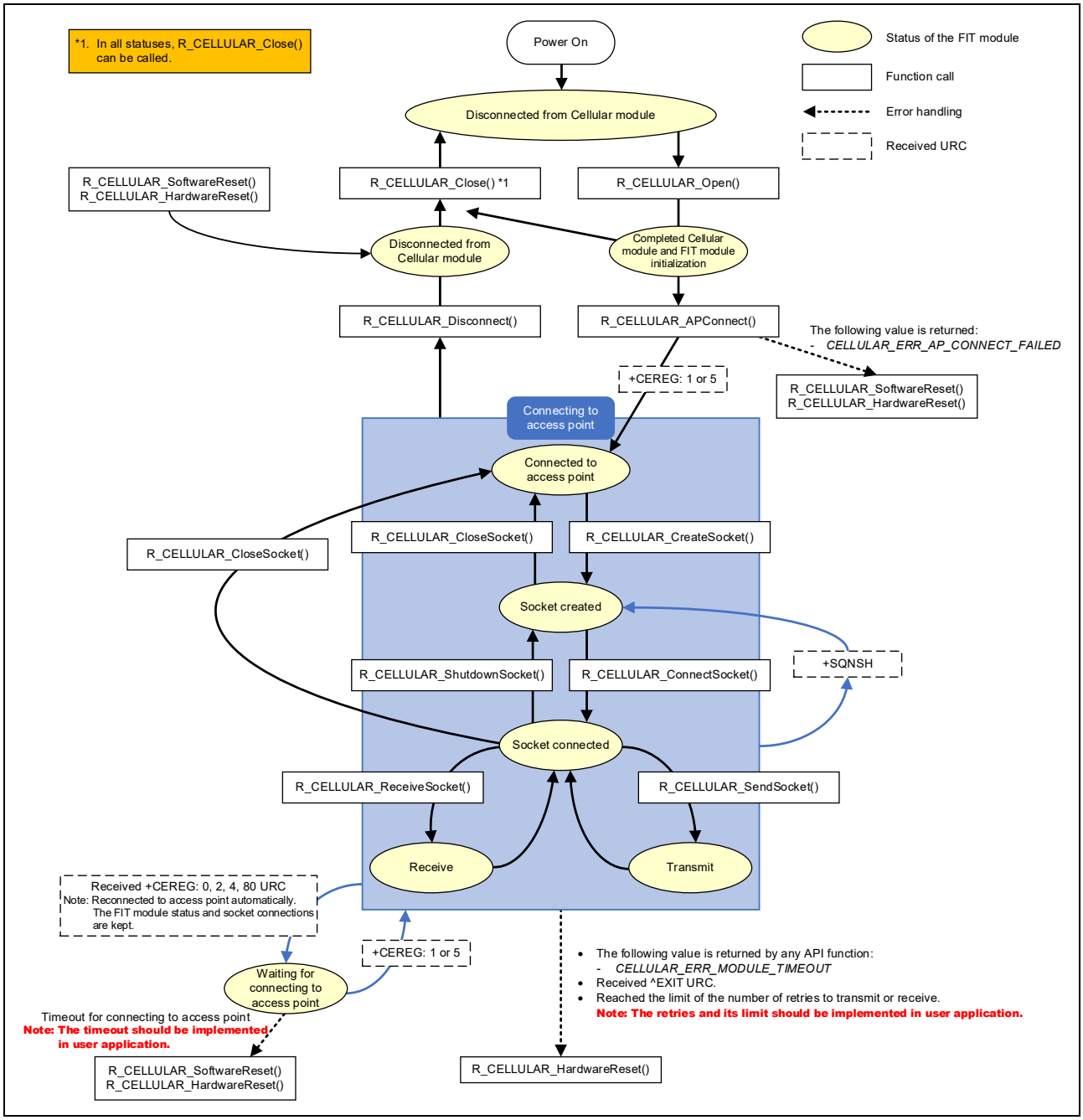

Figure 1.3 Status transitions of RYZ014A Cellular FIT module

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 IRQ 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 “iot.truphone.com”.	Specifies the name of the access point to connect to. Set this option to match the SIM 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 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 used. This setting may be omitted if there is no password.
CELLULAR_CFG_PIN_CODE Note: No default is defined.	Sets the PIN code of the SIM used. This setting may be omitted if no PIN code has been set.
CELLULAR_CFG_AUTH_TYPE Note: The default is 1.	Specifies authentication protocol used for PDP contexts. Set this option to match the SIM used. 0: None, 1: PAP, 2: CHAP
CELLULAR_CFG_NETWORK_NOTIFY_LEVEL Note: The default is 2.	Configures value to control the presentation of an unsolicited result code (URC) +CEREG which notifies occurrence of an event such as registration and location information. Set this option to a value of 0 to 5. In order to output the unsolicited result code to log information, set CELLULAR_CFG_DEBUGLOG to 4. 0: Disabled, 1: Enable registration unsolicited result code, 2: Enable registration and location information unsolicited result code, 3: Enable registration, location information and EMM cause value information unsolicited result code, 4: Enable registration, location information and PSM parameters unsolicited result code, 5: Enable registration, location information, EMM cause value information and PSM parameters unsolicited result code.
CELLULAR_CFG_ATC_RETRY_CGATT Note: The default is 600.	Specifies the maximum number of retries to confirm connectivity to an access point before it established. The retry interval is 1 second. Set this option to a value of 0 to 65535.
CELLULAR_CFG_EX_TIMEOUT Note: The default is 0.	Specifies the exchange timeout. Set this option to a value of 0 to 120. Use a value of 0 to perform no exchange timeout.
CELLULAR_CFG_SCI_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_PSM_PREPARATION_TIME Note: The default is 100.	Configures guard period during which no character should be received on UART before module entering in sleep mode when UART power saving is activated. The unit is milliseconds. Set this option to a value of 100 to 10000.
CELLULAR_CFG_PSM_WAKEUP_LATENCY Note: The default is 5000.	Maximum wake-up latency to come back to operational mode whenever host needs access to the Cellular module’s services. The unit is milliseconds. Set this option to a value of 0 to 10000.
CELLULAR_CFG_RING_LINE_ACTIVE_TIME Note: The default is 1000.	Configures how long the RING0 line is activated in order to indicate an unsolicited result code or pending data. The unit is milliseconds. Set this option to a value of 1000 to 5000.

CELLULAR_CFG_UPGRADE_TIME Note: The default is 60.	Configures timeout period for completion of the firmware upgrade triggered by R_CELLULAR_FirmUpgradeBlocking(). The unit is minutes. Set this option to a value of 1 to 600.
CELLULAR_CFG_URC_CHARGET_ENABLED Note: The default is 0.	Specifies whether to use a callback function to receive an AT command response from the Cellular module such as unsolicited result code (URC). 0: Not used, 1: Used.
CELLULAR_CFG_URC_CHARGET_FUNCTION Note: The default is “my_sw_urc_charget_function”.	Specifies function name of the callback function called when an AT command response from the Cellular module such as unsolicited result code (URC) is received.
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_CTS_SW_CTRL Note: The default is 0.	Configures the UART flow control mode. 0: CTS hardware flow control is enabled, RTS flow control is performed by the FIT module using GPIO, 1: RTS hardware flow control is enabled, CTS flow control is performed by the FIT module using GPIO.
CELLULAR_CFG_CTS_PORT Note: The default is 2.	Configures the port direction register (PDR) setting for the general port that controls the CTS 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. This option takes effect when CELLULAR_CFG_CTS_SW_CTRL is set to 1.
CELLULAR_CFG_CTS_PIN Note: The default is 2.	Configures the port output data register (PODR) setting for the general port that controls the CTS 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. This option takes effect when CELLULAR_CFG_CTS_SW_CTRL is set to 1.
CELLULAR_CFG_PFS_SET_VALUE Note: The default is 0x0BU.	Specifies the pin function control register (PFS) setting value to select the peripheral function of the MCU pin used to control the RTS pin of the Cellular module. Set this option to match the pin to be used. This option takes effect when CELLULAR_CFG_CTS_SW_CTRL is set to 1.
CELLULAR_CFG_RTS_PORT Note: The default is 2	Configures the port direction register (PDR) setting for the general 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 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 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 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_IRQ_NUM Note: The default is 4.	Specifies the IRQ pin number corresponding to the IRQ pin which is connected to the RING0 pin of the Cellular module to use its output as an external pin interrupt signal. The default value is suitable when IRQ4 is used. Set this option to match the MCU used.

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 RING0 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.

The configuration option settings of the RTOS used by the FIT module are contained in src/frtos_config/FreeRTOSConfig.h when FreeRTOS used.
The name of the option for FreeRTOS and its setting value is listed in Table 2.5. 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”.

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 rev1.11

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 2.6 Code Sizes

ROM, RAM and Stack Code Sizes			
Device	**Category**	**Memory Used**	**Remarks**
RX65N RX72N	ROM	Approx. 36 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 (Refer to Table 3.1.)

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

- st_cellular_ap_cfg_t (Refer to Table 3.4.)

Structures for obtaining the IP addresses (used by R_CELLULAR_DnsQuery() and R_CELLULAR_GetPDPAddress())

- st_cellular_ipaddr_t (Refer to Table 3.5.)

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

- st_cellular_datetime_t (Refer to Table 3.6.)

Structures for setting and obtaining eDRX configuration (Used by R_CELLULAR_SetEDRX())

- st_cellular_edrx_config_t (Refer to Table 3.7.)

Structures for setting and obtaining PSM configuration (Used by R_CELLULAR_SetPSM())

- st_cellular_psm_config_t (Refer to Table 3.8.)

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

- st_cellular_iccid_t (Refer to Table 3.9.)

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

- st_cellular_imei_t (Refer to Table 3.10.)

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

- st_cellular_imsi_t (Refer to Table 3.11.)

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

- st_cellular_phonenum_t (Refer to Table 3.12.)

Structures for obtaining RSSI (used by R_CELLULAR_GetRSSI())

- st_cellular_rssi_t (Refer to Table 3.13.)

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

- st_cellular_svn_t (Refer to Table 3.14.)

Structures for setting ping parameters (used by R_CELLULAR_Ping())

- st_cellular_ping_cfg_t (Refer to Table 3.15.)

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

- st_cellular_notice_t (Refer to Table 3.16.)

Structures for obtaining FOTA status (used by R_CELLULAR_GetUpgradeState())
- st_cellular_updatestate_t (Refer to Table 3.18.)

Structures for obtaining certificate or secret key (used by R_CELLULAR_GetCertificate())
- st_cellular_certificate_t (Refer to Table 3.19.)

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_MODULE_TIMEOUT	*/* Timeout 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 */*
CELLULAR_ERR_RECV_TASK	/* Serial reception failure */

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

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

The entire st_cellular_ctrl_t structure, defined by the user, referenced by the pointer *p_ctrl* should be initialized with zero before running this function.

If *p_cfg* is set to a pointer to the st_cellular_cfg_t structure defined by the user, the values stored the structure are used. The members in st_cellular_cfg_t structure are listed in Table 3.1.

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_cgatt_retry_count = 600 / sci_timeout = 60000 / tx_process_size = 1500
/ rx_process_size = 1500 / packet_data_size = 0 / exchange_timeout = 60
/ connect_timeout = 200 / send_timeout = 10 / creatable_socket = 6>

The value of sim_pin_code is set to CELLULAR_CFG_PIN_CODE.

Once setting to use the callback function using the Smart Configurator, an AT command response from the Cellular module such as unsolicited result code (URC) can be received with the callback function. (Refer to Table 2.1 for details.)

The Cellular module may reboot unexpectedly and notifies an unsolicited result code +SYSSTART. In that case, recovery operation should be performed. Refer to section 4.3.2 for details. It is recommended to detect the unsolicited result code +SYSSTART with the callback function.

Table 3.1 Members in configuration structure

Members in st_cellular_cfg_t structure	
uint8_t sim_pin_code[]	PIN code of SIM
uint32_t baud_rate	Baud rate for communication with module
uint16_t ap_cgatt_retry_count	Retry limit when confirming connectivity to access point
uint32_t sci_timeout	MCU communication timeout setting
uint16_t tx_process_size	Data size of single transmission to Cellular module
uint16_t rx_process_size	Data size of single reception from Cellular module
uint16_t packet_data_size	Data size per packet
uint16_t exchange_timeout	Exchange timeout
uint16_t connect_timeout	Socket connection timeout
uint8_t send_timeout	Packet transmission timeout
uint8_t creatable_socket	Maximum socket creation count

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is not thread safe.

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]

```
/* Callback function example (Using the default function name) */
uint8_t portBuffer[2049] = {'\0'};
void my_sw_urc_charget_function(void * p_arg)
{
   //Obtains an AT command response such as URC.
   sprintf((char *)portBuffer, "%.2048s", (char *)p_arg);
}

e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctr = {0};
st_cellular_cfg_t cellular_cfg = {
                        "0000",     // PIN code of SIM
                        921600,     // Baud rate for communication with module
                                    (921600 recommended)
                        600,        // Retry limit when confirming connectivity 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 and shuts down it.

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 */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_MODULE_COM	*/* Failure communicating with Cellular module */*
CELLULAR_ERR_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*
CELLULAR_ERR_RECV_TASK	*/* Serial reception failure */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Closes communication with the Cellular module and shuts down it.

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.

If return values described below are returned, a shutdown of the Cellular module has not been completed. In that case, the following operation should be run depending on the value returned.

(1) If a value of CELLULAR_ERR_OTHER_API_RUNNING is returned, this function should be rerun.

(2) If one of values listed below is returned, this function should be rerun after running R_CELLULAR_Open().

- CELLULAR_ERR_MODULE_COM
- CELLULAR_ERR_MODULE_TIMEOUT
- CELLULAR_ERR_RECV_TASK

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

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);

/* Recovery operation if specific values are returned */
if ((CELLULAR_ERR_MODULE_COM == ret)     ||
    (CELLULAR_ERR_MODULE_TIMEOUT == ret) ||
    (CELLULAR_ERR_RECV_TASK == ret))
{
   ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
   ret = R_CELLULAR_Close(&cellular_ctrl);
}
else if (CELLULAR_ERR_OTHER_API_RUNNING == ret)
{
   ret = R_CELLULAR_Close(&cellular_ctrl);
}
```

Special Notes

This function sends AT commands to the Cellular module. Therefore, this function may not be completed successfully when running it after getting timeout on another API function. If the timeout occurs, R_CELLULAR_HardwareReset() should be called before running this function.

3.3 R_CELLULAR_APConnect()

Connects the RYZ014A 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_ALREADY_CONNECT	*/* Already connected to access point */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*
CELLULAR_ERR_AP_CONNECT_FAILED	*/* Failure connecting to access point */*

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 3.3. Then, while R_CELLULAR_APConnect() is 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 3.4.

Retries to confirm connection to an access point are performed up to the number of times specified by the value of CELLULAR_CFG_ATC_RETRY_CGATT macro or the value of ap_cgatt_retry_count member in the st_cellular_cfg_t structure at 1 second intervals. To complete connection to an access point could take several minutes or more. The time to complete the connection depends on the SIM used and network conditions.

When a connection is successfully established to an access point, the network time is obtained automatically and stored in the non-volatile memory of the Cellular module. To obtain the time, run R_CELLULAR_GetTime(). In addition, the IP address assigned when a connection to an access point is established is stored in the pdp_addr member in the st_cellular_ctrl_t structure referenced by the pointer *p_ctrl*.

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

Table 3.2 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 used
CELLULAR_CFG_AUTH_TYPE	Authentication protocol (0: None, 1: PAP, 2: CHAP)
CELLULAR_CFG_ATC_RETRY_CGATT	Maximum number of retries to confirm connectivity to an access point before it established

Table 3.3 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 used
uint8_t ap_cgatt_retry_count	Maximum number of retries to confirm connectivity to an access point before it established

Table 3.4 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.

Thread Safety

This function is thread safe.

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

It can take more than 10 minutes to connect to an access point after running R_CELLULAR_FactoryReset() or for the first time after changing the SIM used. If a value of CELLULAR_ERR_AP_CONNECT_FAILED is returned, the value of the CELLULAR_CFG_ATC_RETRY_CGATT macro or ap_cgatt_retry_count member in the st_cellular_cfg_t structure should be increased to extend the duration to wait for the connection established.

Restricting LTE bands allowed to use using R_CELLULAR_SetBand(), the time taken to connect to an access point may decrease.

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 */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

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.

Thread Safety

This function is thread safe.

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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.

Thread Safety

This function is thread safe.

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

R_CELLULAR_Close() should be called after running this function.

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 or IPv6 = 6)*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_SOCKET_CREATE_LIMIT	*/* Socket creation limit exceeded */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*
CELLULAR_ERR_SIM_NOT_SUPPORT_IPV6	*/* IPv6 is not available */*

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_PROTOCOL_TCP (6), and for *ip_version*, specify CELLULAR_PROTOCOL_IPV4 (4) or CELLULAR_PROTOCOL_IPV6 (6). If *ip_version* is set to IPv6 (6) when IPv6 is not available, a value of CELLULAR_ERR_SIM_NOT_SUPPORT_IPV6 is returned.

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.

The value to be set for the *protocol_type* is as follows.

```
#define CELLULAR_PROTOCOL_TCP          (6)      // TCP
```

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

```
#define CELLULAR_PROTOCOL_IPV4         (4)      // IPv4
#define CELLULAR_PROTOCOL_IPV6         (6)      // IPv6
```

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
int32_t socket_no;
st_cellular_ctrl_t cellular_ctrl = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, CELLULAR_PROTOCOL_TCP,
                                     CELLULAR_PROTOCOL_IPV4);
```

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 uint8_t * const p_ip_addr,
    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*
p_ip_addr (IN)	*Pointer to connection destination address in string type* *Using an IPv4 address, it should be period separated decimal format,* *e.g., “104.120.11.132”* *Using an IPv6 address, it should be colon separated hexadecimal format,* *e.g., ”1234:5678:9ABC:DEF0:1234:5678:9ABC:DEF0”* *A fully qualified domain name (FQDN) as a host name also can be used,* *e.g., “www.renesas.com”*
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_MODULE_TIMEOUT	*/* Timeout 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 */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Using the socket specified by *socket_no*, connects to the destination specified by either the IP address or the host name referenced by the pointer *p_ip_addr*, and contacts to the port specified by *port*. The IP address and the host name should be expressed as a string type data.

Before running this function, call 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.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
int8_t socket_no;
uint16_t port_no = 33333;
uint8_t ip_adder[] = "104.120.11.132";

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, CELLULAR_PROTOCOL_TCP,
                                     CELLULAR_PROTOCOL_IPV4);
if (0 < socket_no)
{
   ret = R_CELLULAR_ConnectSocket(&cellular_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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_SOCKET_NOT_READY	*/* Socket is in error status */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
int8_t socket_no;
uint16_t port_no = 33333;
uint8_t ip_adder[] = "104.120.11.132";

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, CELLULAR_PROTOCOL_TCP,
                                     CELLULAR_PROTOCOL_IPV4);
if (0 < socket_no)
{
   ret = R_CELLULAR_ConnectSocket(&cellular_ctrl, socket_no, ip_adder,
                                  port_no);
   ret = R_CELLULAR_ShutdownSocket(&cellular_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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_SOCKET_NOT_READY	*/* Socket is in error status */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Closes the socket specified by *socket_no*. If the socket is communicating, the socket is disconnected.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
int8_t socket_no;
uint16_t port_no = 33333;
uint8_t ip_adder[] = "104.120.11.132";

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, CELLULAR_PROTOCOL_TCP,
                                     CELLULAR_PROTOCOL_IPV4);
if (0 < socket_no)
{
   ret = R_CELLULAR_ConnectSocket(&cellular_ctrl, socket_no, ip_adder,
                                  port_no);
   ret = R_CELLULAR_CloseSocket(&cellular_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 = 1 to 0xffffffff)*

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_MODULE_TIMEOUT	*/* Timeout 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 */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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 function, run R_CELLULAR_ConnectSocket(). If R_CELLULAR_ConnectSocket() has not been run, a value of CELLULAR_ERR_SOCKET_NOT_READY is returned.

If a value of CELLULAR_ERR_MODULE_TIMEOUT is returned, the Cellular module may be unresponsive to subsequent AT commands. In that case, R_CELLULAR_HardwareReset() should be called to recover the Cellular module. Refer to section 3.41 for the FIT module's status after running R_CELLULAR_HardwareReset().

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
int8_t socket_no;
uint16_t port_no = 33333;
uint8_t ip_adder[] = "104.120.11.132";

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, CELLULAR_PROTOCOL_TCP,
                                     CELLULAR_PROTOCOL_IPV4);
if (0 < socket_no)
{
   ret = R_CELLULAR_ConnectSocket(&cellular_ctrl, socket_no, ip_adder,
                                  port_no);
   ret = R_CELLULAR_SendSocket(&cellular_ctrl, socket_no, data, length,
                               timeout);
}
```

Special Notes

None

3.11 R_CELLULAR_ReceiveSocket()

Receives data via the specified socket.

Format

```
int32_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 received data*
length (IN)	*Received 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 received	*/* 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_SOCKET_NOT_READY	*/* Socket is in error status */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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*. If there is no received data, zero is returned as the return value.

Before running this function, 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.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
int32_t socket_no;
int32_t recv_ret;
uint16_t port_no = 33333;
uint8_t ip_adder[] = "104.120.11.132";

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
socket_no = R_CELLULAR_CreateSocket (&cellular_ctrl, CELLULAR_PROTOCOL_TCP,
                                     CELLULAR_PROTOCOL_IPV4);
if (0 < socket_no)
{
   ret = R_CELLULAR_ConnectSocket(&cellular_ctrl, socket_no, ip_adder,
                                  port_no);
   recv_ret = R_CELLULAR_ReceiveSocket(&cellular_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,
    const uint8_t ip_version,
    st_cellular_ipaddr_t * const p_addr
)
```

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_version (IN)	*IP version (specified as IPv4 = 4 or IPv6 = 6)*
p_addr (IN/OUT)	*Pointer to st_cellular_ipaddr_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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*
CELLULAR_ERR_SIM_NOT_SUPPORT_IPV6	*/* IPv6 is not available */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Executes a DNS query, obtains the IP address of the IP version specified by *ip_version* for the domain specified by *domain_name*, and saves it in the st_cellular_ipaddr_t structure referenced by the pointer *p_addr*. The members in st_cellular_ipaddr_t structure are listed in Table 3.5.

Before running this function, 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.

If *ip_version* is set to 6 (IPv6) when IPv6 is not available, a value of CELLULAR_ERR_SIM_NOT_SUPPORT_IPV6 is returned.

Table 3.5 Members in structure for obtaining the IP addresses

Members in st_cellular_ipaddr_t structure	
uint8_t ipv4[]	IPv4 address
uint8_t ipv6[]	IPv6 address

Reentrant

Reentrant operation is possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_ipaddr_t addr = {0};
uint8_t domain_name[] = "www.renesas.com";

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
ret = R_CELLULAR_DnsQuery(&cellular_ctrl, domain_name, CELLULAR_PROTOCOL_IPV4,
                          &addr);
```

Special Notes

None

3.13 R_CELLULAR_GetTime()

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

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the date and time information setting of the Cellular module and saves it in the st_cellular_datetime_t structure referenced by the pointer *p_time*. The members in st_cellular_datetime_t structure are listed in Table 3.6.

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. Running R_CELLULAR_APConnect() to attach to a network, the current date and time is retrieved from network automatically.

Table 3.6 Members in structure for obtaining the time

Members in st_cellular_datetime_t structure	
uint8_t year	Year (two last digits)
uint8_t month	Month
uint8_t day	Day
uint8_t hour	Hour
uint8_t min	Minutes
uint8_t sec	Seconds
int8_t timezone	Time zone

Reentrant

Reentrant operation is possible.

Thread Safety

This function is thread safe.

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()

Configures the date and time information setting of the RYZ014A Cellular module.

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Configures the Cellular module with the date and time information stored in *p_time*. The members in st_cellular_datetime_t structure are listed in Table 3.6.

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.

Thread Safety

This function is thread safe.

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 RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_SetEDRX (
    st_cellular_ctrl_t * const p_ctrl,
    const st_cellular_edrx_config_t * const p_config,
    st_cellular_edrx_config_t * const p_result
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_config (IN)	*Pointer to st_cellular_edrx_config_t structure defined by the user (For setting eDRX parameters)*
p_result (IN/OUT)	*Pointer to st_cellular_edrx_config_t structure defined by the user (For setting eDRX parameters)*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Configures the eDRX parameter settings in the Cellular module with the parameters stored in the st_cellular_edrx_config_t structure referenced by the pointer *p_config*. The members in st_cellular_edrx_config_t structure are listed in Table 3.7.

Obtains the applied eDRX cycle and Paging Time Window (PTW) and saves them in the st_cellular_edrx_config_t structure referenced by the pointer *p_result*.

Table 3.7 Members in structure for setting and obtaining eDRX configuration

Members in st_cellular_edrx_config_t structure	
e_cellular_edrx_mode_t edrx_mode	eDRX mode
e_cellular_edrx_cycle_t edrx_cycle	eDRX cycle
e_cellular_ptw_cycle_t ptw_cycle	PTW cycle

The values that may be set for the edrx_mode member in the st_cellular_edrx_config_t structure 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 edrx_cycle (eDRX cycle) member in the st_cellular_edrx_config_t structure 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 ptw_cycle (paging time window cycle) member in the st_cellular_edrx_config_t structure are as follows.

```
typedef enum
{
    CELLULAR_PTW_CYCLE_1_SEC,         // PTW (1.28sec)
    CELLULAR_PTW_CYCLE_2_SEC,         // PTW (2.56sec)
    CELLULAR_PTW_CYCLE_4_SEC,         // PTW (3.84sec)
    CELLULAR_PTW_CYCLE_5_SEC,         // PTW (5.12sec)
    CELLULAR_PTW_CYCLE_6_SEC,         // PTW (6.40sec)
    CELLULAR_PTW_CYCLE_7_SEC,         // PTW (7.68sec)
    CELLULAR_PTW_CYCLE_9_SEC,         // PTW (8.96sec)
    CELLULAR_PTW_CYCLE_10_SEC,        // PTW (10.24sec)
    CELLULAR_PTW_CYCLE_11_SEC,        // PTW (11.52sec)
    CELLULAR_PTW_CYCLE_13_SEC,        // PTW (12.80sec)
    CELLULAR_PTW_CYCLE_14_SEC,        // PTW (14.08sec)
    CELLULAR_PTW_CYCLE_15_SEC,        // PTW (15.36sec)
    CELLULAR_PTW_CYCLE_16_SEC,        // PTW (16.64sec)
    CELLULAR_PTW_CYCLE_18_SEC,        // PTW (17.92sec)
    CELLULAR_PTW_CYCLE_19_SEC,        // PTW (19.20sec)
    CELLULAR_PTW_CYCLE_20_SEC         // PTW (20.48sec)
} e_cellular_ptw_cycle_t;
```

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

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};
st_cellular_edrx_config_t edrx_cfg = {CELLULAR_EDRX_MODE_ACTIVE,
                                      CELLULAR_EDRX_CYCLE_20_SEC,
                                      CELLULAR_PTW_CYCLE_2_SEC};
st_cellular_edrx_config_t edrx_ret = {0};

/* The third parameter is not used */
ret = R_CELLULAR_Open(&cellular_ctrl, &edrx_cfg, NULL);

/* Obtaining applied parameters using edrx_ret */
ret = R_CELLULAR_SetEDRX(&cellular_ctrl, &edrx_cfg, &edrx_ret);
```

Special Notes

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

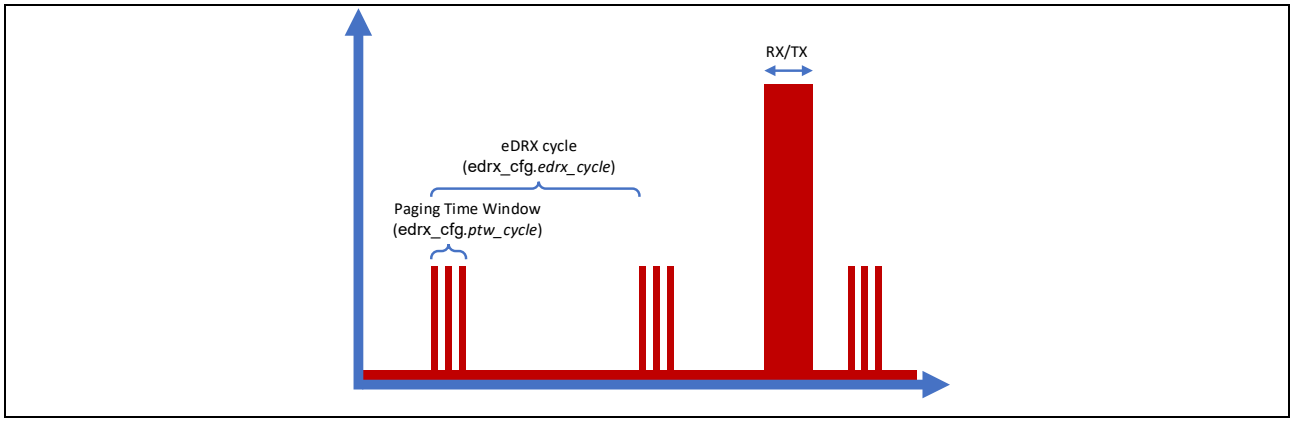

Figure 3.1 eDRX parameter

3.16 R_CELLULAR_GetEDRX()

Obtains the applied extended Discontinuous Reception (eDRX) parameters from the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_GetEDRX (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_edrx_config_t * const p_result
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_result (IN/OUT)	*Pointer to st_cellular_edrx_config_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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the eDRX parameters applied and saves them in the st_cellular_edrx_config_t structure referenced by the pointer *p_result*. The members in st_cellular_edrx_config_t structure are listed in Table 3.7.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_edrx_config_t edrx_ret = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetEDRX(&cellular_ctrl, &edrx_ret);
```

3.17 R_CELLULAR_SetPSM()

Configures the power saving mode (PSM) parameter settings of the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_SetPSM (
    st_cellular_ctrl_t * const p_ctrl,
    const st_cellular_psm_config_t * const p_config,
    st_cellular_psm_config_t * const p_result
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_config (IN)	*Pointer to st_cellular_psm_config_t structure defined by the user (For setting PSM parameters)*
p_result (IN/OUT)	*Pointer to st_cellular_psm_config_t structure defined by the user (For obtaining applied PSM parameters)*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*
CELLULAR_ERR_IRQ_OPEN	*/* IRQ port initialization failure */*
CELLULAR_ERR_RECV_TASK	*/* Serial reception failure */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Configures the PSM parameters in the Cellular module with the parameters stored in the st_cellular_psm_config_t structure referenced by the pointer *p_config*.

Obtains the applied PSM parameters and saves them in the st_cellular_psm_config_t structure referenced by the pointer *p_result*.

Running R_CELLULAR_Close() with PSM enabled, PSM operation of the FIT module is disabled. This function must be run each time after running R_CELLULAR_Open() to enable the PSM operation of the FIT module.

Table 3.8 Members in structure for setting and obtaining PSM configuration

Members in st_cellular_psm_config_t structure		
e_cellular_psm_mode_t	psm_mode	PSM mode
e_cellular_tau_cycle_t	tau_cycle	TAU cycle
e_cellular_cycle_multiplier_t	tau_multiplier	TAU cycle multiplier
e_cellular_active_cycle_t	active_cycle	Active time
e_cellular_cycle_multiplier_t	active_multiplier	Active time multiplier

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 psm_mode member in the st_cellular_psm_config_t structure 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 tau_cycle (tracking area update cycle) member in the st_cellular_psm_config_t structure 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 active_cycle (active time) member in the st_cellular_psm_config_t structure 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 tau_multiplier and the active_multiplier member in the st_cellular_psm_config_t structure 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.

Thread Safety

This function is thread safe.

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};
st_cellular_psm_config_t psm_cfg = {CELLULAR_PSM_MODE_ACTIVE,
                                    CELLULAR_TAU_CYCLE_10_MIN,
                                    CELLULAR_CYCLE_MULTIPLIER_1,
                                    CELLULAR_ACTIVE_CYCLE_1_MIN,
                                    CELLULAR_CYCLE_MULTIPLIER_1};
st_cellular_psm_config_t psm_ret = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);

/* The third parameter is not used */
ret = R_CELLULAR_SetPSM(&cellular_ctrl, &psm_cfg, NULL);

/* Obtaining applied parameters using psm_ret */
ret = R_CELLULAR_SetPSM(&cellular_ctrl, &psm_cfg, &psm_ret);
```

- The tau_cycle field of psm_cfg set to 10 minutes and the tau_multiplier field of psm_cfg set to a multiplier of 1 = 10 minutes × 1 = 10 minutes
- The active_cycle field of psm_cfg set to 1 minute and the active_multiplier field of psm_cfg 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.

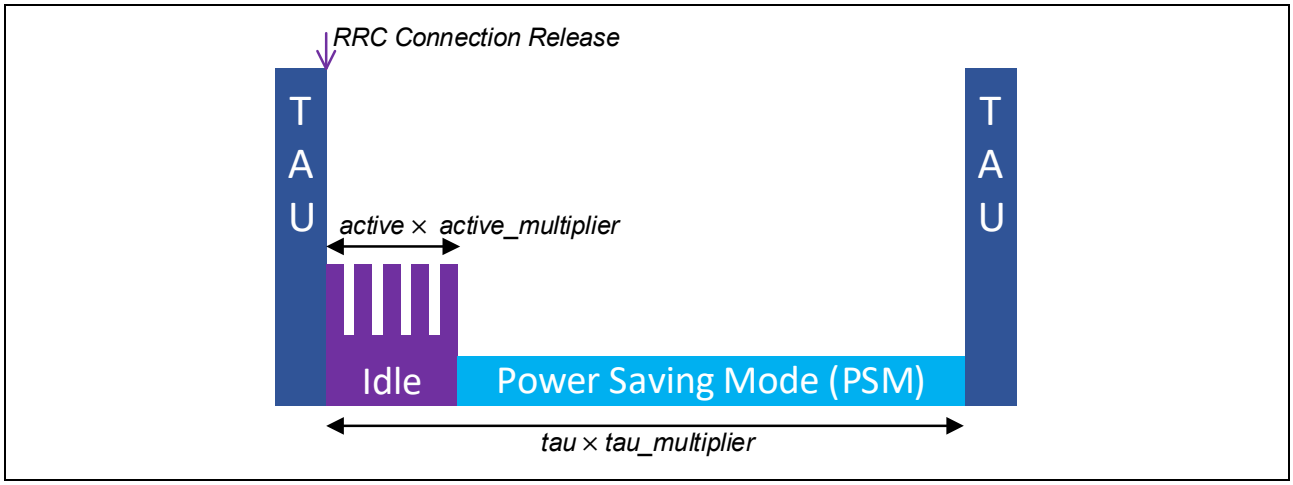

Figure 3.2 PSM parameter

3.18 R_CELLULAR_GetPSM()

Obtains the applied power saving mode (PSM) parameters from the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_GetPSM (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_psm_config_t * const p_result
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_result (IN/OUT)	*Pointer to st_cellular_psm_config_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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the PSM parameters applied and saves them in the st_cellular_psm_config_t structure referenced by the pointer *p_result*. The members in st_cellular_psm_config_t structure are listed in Table 3.8.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_psm_config_t psm_ret = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetPSM(&cellular_ctrl, &psm_ret);
```

3.19 R_CELLULAR_GetICCID()

Obtains the IC card identifier (ICCID) assigned to the SIM 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the ICCID from the SIM used and saves it in the st_cellular_iccid_t structure referenced by the pointer *p_iccid*. The member in st_cellular_iccid_t structure is listed in Table 3.9. The ICCID obtained is up to 22 digits.

Table 3.9 Member in structures for obtaining the ICCID

Member in st_cellular_iccid_t structure	
uint8_t iccid[]	ICCID

Reentrant

Reentrant operation is possible.

Thread Safety

This function is thread safe.

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

3.20 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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*. The member in st_cellular_imei_t structure is listed in Table 3.10. The IMEI obtained is up to 15 digits.

Table 3.10 Member in structures for obtaining the IMEI

Member in st_cellular_imei_t structure	
uint8_t imei[]	IMEI

Reentrant

Reentrant operation is possible.

Thread Safety

This function is thread safe.

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

3.21 R_CELLULAR_GetIMSI()

Obtains the international mobile subscriber identity (IMSI) assigned to the SIM 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the IMSI from the SIM used and saves it in the st_cellular_imsi_t structure referenced by the pointer p_imsi. The member in st_cellular_imsi_t structure is listed in Table 3.11. The IMSI obtained is up to 15 digits.

Table 3.11 Member in structures for obtaining the IMSI

Member in st_cellular_imsi_t structure	
uint8_t imsi[]	IMSI

Reentrant

Reentrant operation is possible.

Thread Safety

This function is thread safe.

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

3.22 R_CELLULAR_GetPhonenum()

Obtains the phone number associated with the SIM 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains the phone number from the SIM used and saves it in the st_cellular_phonenum_t structure referenced by the pointer *p_phonenum*. The member in st_cellular_phonenum_t structure is listed in Table 3.12. The phone number obtained is up to 15 digits.

Table 3.12 Member in structures for obtaining the phone number

Member in st_cellular_phonenum_t structure	
uint8_t phonenum[]	Phone number

Reentrant

Reentrant operation is possible.

Thread Safety

This function is thread safe.

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

3.23 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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*. The members in st_cellular_rssi_t structure are listed in Table 3.13.

If RSSI and BER are not known or not detectable owing to such factors as the Cellular module is not connected to an access point, “99” is acquired.

Table 3.13 Members in structures for obtaining RSSI

Members in st_cellular_rssi_t structure	
uint8_t rssi[]	RSSI
uint8_t ber[]	BER

Reentrant

Reentrant operation is possible.

Thread Safety

This function is thread safe.

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

3.24 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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*. The members in st_cellular_svn_t structure are listed in Table 3.14.

Table 3.14 Members in structures for obtaining SVN

Members in st_cellular_svn_t structure	
uint8_t svn[]	Software version number
uint8_t revision[]	Revision
uint8_t lr_svn[]	LR version number

Reentrant

Reentrant operation is possible.

Thread Safety

This function is thread safe.

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

3.25 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,
    const st_cellular_ping_cfg_t * const p_cfg
)
```

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_cfg (IN/OUT)	*Pointer to st_cellular_ping_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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Pings the host. Either the host IP address or the host name referenced by the pointer *p_host* should be string type.

This function can be run with one of the following two settings of *p_cfg*. (Refer to Table 3.15 for setting value details.)

(1) Sets *p_cfg* to NULL, then pings with the default configuration.
(2) Sets *p_cfg* to the pointer which references the st_cellular_ping_cfg_t structure with members set to the applicable value.

Once setting CELLULAR_CFG_URC_CHARGET_ENABLED to 1 using the Smart Configurator to use the callback function, the ping status can be received with the callback function. (Refer to Table 2.1 for details.)

Table 3.15 Members to Configure Pings (R_CELLULAR_Ping())

Members in st_cellular_ping_cfg_t structure		
uint8_t	count	Number of ping echo request to send. (Default: 4, minimum: 1, maximum: 64)
uint16_t	len	Length of ping echo request message in bytes. (Default: 32, minimum: 32, maximum: 1400)
uint16_t	interval	Length of ping echo request message in seconds. (Default: 1, minimum: 1, maximum: 600)
uint8_t	timeout	Time to wait for an echo reply in seconds. (Default: 10, minimum: 1, maximum: 60)

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
/* Callback function example (Using the default function name) */
void my_sw_urc_charget_function(void * p_arg)
{
   uint8_t * p_urc = p_arg;
   printf("URC = %s\n", p_urc);
}

e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_ping_cfg_t ping_cfg = {64, 1400, 1, 10};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);

/* Setting a host name */
ret = R_CELLULAR_Ping(&cellular_ctrl, www.renesas.com, NULL);

/* Setting an IP address */
ret = R_CELLULAR_Ping(&cellular_ctrl, "104.120.11.132", NULL);

/* Setting with the st_cellular_ping_cfg_t structure */
ret = R_CELLULAR_Ping(&cellular_ctrl, "www.renesas.com", &ping_cfg);
```

Special Notes

None

3.26 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
)
```

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*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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*. The members in st_cellular_notice_t structure are listed in Table 3.16.

In addition, once setting CELLULAR_CFG_URC_CHARGET_ENABLED to 1 using the Smart Configurator to use the callback function, a change in the Cellular module's access point connection status can be received with the callback function. (Refer to Table 2.1 for details.)

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

```
typedef enum
{
    CELLULAR_DISABLE_NETWORK_RESULT_CODE = 0,
            // Disable notification of network registration status change
    CELLULAR_ENABLE_NETWORK_RESULT_CODE_LEVEL1,
            // Enable notification of network registration status change
    CELLULAR_ENABLE_NETWORK_RESULT_CODE_LEVEL2,
            // Enable detailed notification of network registration status change
    CELLULAR_ENABLE_NETWORK_RESULT_CODE_LEVEL3,
            // Enable network registration, location information and EMM cause value information
    CELLULAR_ENABLE_NETWORK_RESULT_CODE_LEVEL4,
            // For a UE that wants to apply PSM, enable network registration and location information
    CELLULAR_ENABLE_NETWORK_RESULT_CODE_LEVEL5
            // For a UE that wants to apply PSM, enable network registration, location information and
            // EMM cause value information
} e_cellular_network_result_t;
```

Table 3.16 Members in structures for obtaining the Cellular module's access point connection status

Members in st_cellular_notice_t structure		
e_cellular_network_result_t	level	Network status notification level
e_cellular_reg_stat_t	stat	EPS registration status
uint8_t	ta_code[]	Tracking area code in hexadecimal format, string.
uint8_t	cell_id[]	E-UTRAN cell ID in hexadecimal format, string.
e_cellular_access_tec_t	access_tec	Access technology of the serving cell
uint8_t	active_time[]	Active time value (T3324), string.
uint8_t	tau[]	Periodic TAU value (T3412), string.

The values that may be set for the stat member in the st_cellular_notice_t structure are as follows.

```
typedef enum
{
    CELLULAR_REG_STATS_VALUE0 = 0,    // Not registered
    CELLULAR_REG_STATS_VALUE1,        // Registered, home network
    CELLULAR_REG_STATS_VALUE2,
                    // Not registered, but MT is currently trying to attach or searching an operator to register to
    CELLULAR_REG_STATS_VALUE3,        // Registration denied
    CELLULAR_REG_STATS_VALUE4,        // Unknown (for example, out of E-UTRAN coverage)
    CELLULAR_REG_STATS_VALUE5,        // Registered, roaming
    CELLULAR_REG_STATS_VALUE6,        // Registered for "SMS only", home network (not applicable)
    CELLULAR_REG_STATS_VALUE7,        // Registered for "SMS only", roaming (not applicable)
    CELLULAR_REG_STATS_VALUE8,        // Attached for emergency bearer services only
    CELLULAR_REG_STATS_VALUE9,        // Registered for "CSFB not preferred", home network (not applicable)
    CELLULAR_REG_STATS_VALUE10,       // Registered for "CSFB not preferred", roaming (not applicable)
    CELLULAR_REG_STATS_VALUE80        // Registered, temporary connection lost
} e_cellular_reg_stat_t;
```

The values that may be set for the access_tec member in the st_cellular_notice_t structure are as follows.

```
typedef enum
{
    CELLULAR_ACCESS_TEC0 = 0,   // GSM
    CELLULAR_ACCESS_TEC1,       // GSM Compact
    CELLULAR_ACCESS_TEC2,       // UTRAN
    CELLULAR_ACCESS_TEC3,       // GSM w/EGPRS
    CELLULAR_ACCESS_TEC4,       // UTRAN w/HSDPA
    CELLULAR_ACCESS_TEC5,       // UTRAN w/HSUPA
    CELLULAR_ACCESS_TEC6,       // UTRAN w/HSDPA and HSUPA
    CELLULAR_ACCESS_TEC7        // E-UTRAN
} e_cellular_access_tec_t;
```

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
/* Callback function example (Using the default function name) */
void my_sw_urc_charget_function(void * p_arg)
{
   uint8_t * p_urc = p_arg;
   printf("URC = %s\n", p_urc);
}

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_APConnect(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetAPConnectState(&cellular_ctrl,
            CELLULAR_ENABLE_NETWORK_RESULT_CODE_LEVEL2, &cellular_notice);
```

Special Notes

None

3.27 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
)
```

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*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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 with the callback function. The functionality level of the Cellular module is set to 1 (full functionality) if the functionality level is not 1 when running this function. Refer to section “3.11 Set Phone Functionality: AT+CFUN” of **RYZ014 Modules User's Manual: AT Command** for details of functionality level.

Before running this function, set CELLULAR_CFG_URC_CHARGET_ENABLED to 1 using the Smart Configurator to use the callback function. (Refer to Table 2.1 for details.) If the callback function has not been set to use, a value of CELLULAR_ERR_PARAMETER is returned.

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.

Thread Safety

This function is thread safe.

Examples

```
/* Callback function example (Using the default function name) */
void my_sw_urc_charget_function(void * p_arg)
{
   uint8_t * p_urc = p_arg;
   printf("URC = %s\n”, p_urc);
}

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_GetCellInfo(&cellular_ctrl, CELLULAR_INFO_TYPE9);
```

Special Notes

None

3.28 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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.

Thread Safety

This function is thread safe.

Examples

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

Special Notes

None

3.29 R_CELLULAR_SetOperator()

Sets the operator mode to the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_SetOperator (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t * const p_operator
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_operator (IN)	*Operator 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Sets the operator mode which configures the module according to the requirements of a specific network and/or carrier. The operator mode referenced by the pointer *p_operator* should be string type. The RYZ014A Cellular module only supports ‘standard’ mode of the operator. Refer to section “2.2.2 Operator Modes” of **RYZ014 Module System Integration Guide** for details.

If using a SIM with a PIN code, R_CELLULAR_UnlockSIM() should be run to enter the PIN code after running this function. Refer to section 3.35 for details.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_SetOperator(&cellular_ctrl, “standard”);
```

Special Notes

None

3.30 R_CELLULAR_SetBand()

Configures list of 4G LTE bands the Cellular module is allowed to use.

Format

```
e_cellular_err_t R_CELLULAR_SetBand(
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t * const p_band
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_band (IN)	*List of authorized LTE bands*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Configures list of 4G LTE bands the Cellular module is allowed to use. The list of bands configured is corresponding to the operator mode set by R_CELLULAR_SetOperator(). The list of authorized LTE bands referenced by the pointer *p_band* should be string type. If the list contains more than one band, separate them with a comma. For instance, when specifying bands 1, 2, and 3, the syntax of the list is "1,2,3".

The possible bands the RYZ014A Cellular module is allowed to use are 1, 2, 3, 4, 5, 8, 12, 13, 14, 17, 18, 19, 20, 25, 26, 28 and 66. Specify an appropriate list according to the carrier and destination to be used. The most frequently used bands for each region of the world are listed in Table 3.17. The band 3 must not be used in Japan.

If using a SIM with a PIN code, R_CELLULAR_UnlockSIM() should be run to enter the PIN code after running this function. Refer to section 3.35 for details.

Table 3.17 Most frequently used bands for each region

Region	Bands
North America	2, 4, 5, 12, 13, 25
EMEA	1, 3, 8, 20, 28
Japan	1, 8, 18, 19, 26
Australia	1, 3, 8, 28

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_SetBand(&cellular_ctrl, "1,2,3");
```

Special Notes

None

3.31 R_CELLULAR_GetPDPAddress()

Obtains a PDP address.

Format

```
e_cellular_err_t R_CELLULAR_GetPDPAddress (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_ipaddr_t * const p_addr
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_addr (IN/OUT)	*Pointer to st_cellular_ipaddr_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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains a PDP address that identifies the Cellular module in the address space applicable to the PDP. If IPv6 is available, IPv4 address and IPv6 address are obtained and stored in the ipv4 member and ipv6 member in the st_cellular_ipaddr_t structure referenced by the pointer *p_addr*, respectively. If IPv6 is not available owing to factors such that the SIM used does not support IPv6, only IPv4 address is obtained and stored in the ipv4 member in the structure, then the ipv6 member in the structure is filled with zero. The members in st_cellular_ipaddr_t structure are listed in Table 3.5.

Before running this function, run R_CELLULAR_APConnect() to connect to an access point. If the Cellular module has not been connected to an access point when this function is run, a value of CELLULAR_ERR_NOT_CONNECT is returned.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_ipaddr_t addr = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetPDPAddress (&cellular_ctrl, &addr);
```

Special Notes

None

3.32 R_CELLULAR_FirmUpgrade()

Initiates Firmware upgrade Over-The-Air (FOTA) to upgrade the RYZ014A Cellular module firmware with a firmware fetched from an external server.

Format

```
e_cellular_err_t R_CELLULAR_FirmUpgrade (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t * const p_url,
    const e_cellular_firmupgrade_t command,
    const uint8_t spid
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_url (IN/OUT)	*URL of the firmware*
command (IN)	*Command for upgrade process (specified as upgrade launch = 1 or cancel = 2)*
spid (IN)	*Security profile identifier (setting value = 0 to 6, 0 = unused)*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Initiates Firmware upgrade Over-The-Air (FOTA) to upgrade the RYZ014A Cellular module firmware. Running R_CELLULAR_GetUpgradeState(), the upgrade status can be obtained.

For *spid*, when connecting to an SSL/TLS server to perform FOTA, specify the security profile identifier configured by R_CELLULAR_ConfigSSLProfile().

Before running this function, run R_CELLULAR_APConnect() to connect to an access point. If the Cellular module has not been connected to an access point when this function is run, a value of CELLULAR_ERR_NOT_CONNECT is returned.

Running R_CELLULAR_GetUpgradeState(), a status code of FOTA can be obtained. It also can be obtained from an unsolicited result code +SQNSUPGRADE including <upgrade_status> with the callback function. Once setting CELLULAR_CFG_URC_CHARGET_ENABLED to 1 using the Smart Configurator to use the callback function, the unsolicited result code can be received with the callback function. (Refer to Table 2.1 for details.)

This function is non-blocking to perform FOTA in asynchronous upgrade. Initiating FOTA, this function exits with return value CELLULAR_SUCCESS. Receiving “”downloading”, 100” status of FOTA, reboot the Cellular module using the reset functions to finalize the upgrade. Any reset function such as R_CELLULAR_SoftwareReset() is acceptable. Refer to “6.1 Device Upgrade: AT+SQNSUPGRADE” of **RYZ014 Modules User's Manual: AT Command** for details. An unsolicited result code +SYSSTART will be received after the reboot. Receiving the +SYSSTART, run R_CELLULAR_Open() following R_CELLULAR_Close().

After initiating FOTA using this function, any reset should not be run before receiving “”downloading”, 100” status of FOTA. To cancel the upgrade, run this function specifying 2 for *command*.

The FIT module does not detect and handle timeouts for FOTA. If necessary, error handling with timeouts should be implemented in the user application.

The value that may be set for the *command* is as follows.

```
typedef enum
{
    CELLULAR_FIRM_UPGRADE_NONBLOCKING = 1,
                                                // Performs firmware upgrade in non-blocking mode
    CELLULAR_FIRM_UPGRADE_CANCEL = 2,       // Cancel firmware upgrade
} e_cellular_firmupgrade_t;
```

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

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_FirmUpgrade (&cellular_ctrl, “https://www.renesas.com”,
                              CELLULAR_FIRM_UPGRADE_NONBLOCKING, 0);
```

Special Notes

FOTA with asynchronous upgrade is not supported depending on firmware revisions of the Cellular module. This function is not available if using the Cellular module with the firmware not supporting asynchronous upgrade.

3.33 R_CELLULAR_FirmUpgradeBlocking()

Upgrades the RYZ014A Cellular module firmware with Firmware upgrade Over-The-Air (FOTA) in synchronous upgrade.

Format

```
e_cellular_err_t R_CELLULAR_FirmUpgradeBlocking (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t * const p_url,
    const uint8_t spid
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_url (IN/OUT)	*URL of the firmware*
spid (IN)	*Security profile identifier (setting value = 0 to 6, 0 = unused)*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_NOT_CONNECT	*/* Not connected to access point */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Upgrades the RYZ014A Cellular module firmware with Firmware upgrade Over-The-Air (FOTA) in synchronous upgrade. This function cannot be canceled. While running this function, the other functions cannot be run.

For *spid*, when connecting to an SSL/TLS server to perform FOTA, specify the security profile identifier configured by R_CELLULAR_ConfigSSLProfile().

Before running this function, run R_CELLULAR_APConnect() to connect to an access point. If the Cellular module has not been connected to an access point when this function is run, a value of CELLULAR_ERR_NOT_CONNECT is returned.

Once setting CELLULAR_CFG_URC_CHARGET_ENABLED to 1 using the Smart Configurator to use the callback function, the unsolicited result code can be received with the callback function. (Refer to Table 2.1 for details.)

The upgrade is completed successfully, a value of CELLULAR_SUCCESS is returned. After this function exits with return value CELLULAR_SUCCESS, run R_CELLULAR_Open() following R_CELLULAR_Close().

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

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_FirmUpgradeBlocking(&cellular_ctrl,
                                     "https://www.renesas.com", 0);
```

Special Notes

None

3.34 R_CELLULAR_GetUpgradeState()

Obtains the RYZ014A Cellular module's FOTA status code.

Format

```
e_cellular_err_t R_CELLULAR_GetUpgradeState (
    st_cellular_ctrl_t * const p_ctrl,
    st_cellular_updatestate_t * const p_state
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_state (IN/OUT)	*Pointer to st_cellular_updatestate_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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Obtains a status code of FOTA and saves it in the st_cellular_updatestate_t structure referenced by the pointer *p_state*. The member in st_cellular_updatestate_t structure is listed in Table 3.18.

Table 3.18 Member in structures for obtaining FOTA status

Member in st_cellular_updatestate_t structure	
uint8_t state[]	Status code of FOTA

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_updatestate_t state = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetUpgradeState (&cellular_ctrl, &state);
```

Special Notes

None

3.35 R_CELLULAR_UnlockSIM()

Enters PIN code for the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_UnlockSIM (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t * const p_pass
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
p_pass (IN)	*PIN code in string type*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Enters PIN code for the Cellular module. The functionality level of the Cellular module is set to 4 (Disable RF) if the functionality level is not 1 (full functionality) or 4 when running this function. Refer to section “3.11 Set Phone Functionality: AT+CFUN” of **RYZ014 Modules User's Manual: AT Command** for details of functionality level. If running this function when using a SIM with no PIN code, a value of CELLULAR_SUCCESS is received.

If using a SIM with a PIN code, this function should be run to enter the PIN code after running the functions that cause the Cellular module to reboot during the processing. If the PIN code is not entered, the Cellular module cannot access the SIM and connect to an access point. Refer to section 4.3.2 for the functions that reboot the Cellular module during the processing.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_UnlockSIM (&cellular_ctrl, "0000");
```

Special Notes

None

3.36 R_CELLULAR_WriteCertificate()

Writes a certificate or secret key in the non-volatile memory of the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_WriteCertificate (
    st_cellular_ctrl_t * const p_ctrl,
    const e_cellular_nvm_type_t data_type,
    const uint8_t index,
    const uint8_t * p_data,
    const uint32_t size
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
data_type (IN)	*Data type to be written (specified as certificate = 0 or private key = 1)*
index (IN)	*Index corresponding to the written data (setting value = 0 to 19)*
p_data (IN)	*Data to be written*
size (IN)	*Data size to be written in bytes (setting value = 1 to 16384)*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Writes a certificate or secret key in the non-volatile memory of the Cellular module. Up to 20 certificates and secret keys can be written.

The value that may be set for the *data_type* is as follows.

```
typedef enum
{
    CELLULAR_NVM_TYPE_CERTIFICATE = 0,      //Certificate
    CELLULAR_NVM_TYPE_PRIVATEKEY            //Private key
} e_cellular_nvm_type_t;
```

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
const uint8_t CERTIFICATE[] = "-----BEGIN CERTIFICATE-----"\
                              "AAAAAAAAAAAAAAAAAAAAAAAAA\n"\
                              "-----END CERTIFICATE-----\n";
const uint32_t CERTIFICATE_LENGTH = sizeof( CERTIFICATE );
const uint8_t PRIVATEKEY[] = "-----BEGIN RSA PRIVATE KEY-----\n"\
                             "AAAAAAAAAAAAAAAAAAAAAAAAA\n"\
                             "-----END RSA PRIVATE KEY-----\n";
const uint32_t PRIVATEKEY_LENGTH = sizeof( PRIVATEKEY );

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);

/* Writes a certificate */
ret = R_CELLULAR_WriteCertificate (&cellular_ctrl,
                                   CELLULAR_NVM_TYPE_CERTIFICATE, 0,
                                   CERTIFICATE, CERTIFICATE_LENGTH);

/* Writes a secret key */
ret = R_CELLULAR_WriteCertificate (&cellular_ctrl,
                                   CELLULAR_NVM_TYPE_PRIVATEKEY, 0,
                                   PRIVATEKEY, PRIVATEKEY_LENGTH);
```

Special Notes

None

3.37 R_CELLULAR_EraseCertificate()

Removes a certificate or secret key from the non-volatile memory of the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_EraseCertificate (
    st_cellular_ctrl_t * const p_ctrl,
    const e_cellular_nvm_type_t data_type,
    const uint8_t index
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
data_type (IN)	*Data type to be written (specified as certificate = 0 or private key = 1)*
index (IN)	*Index corresponding to the written data (setting value = 0 to 19)*

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Removes a certificate or secret key from the non-volatile memory of the Cellular module.

Refer to section 3.36 for the value that may be set for the *data_type*.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_EraseCertificate (&cellular_ctrl,
                                   CELLULAR_NVM_TYPE_CERTIFICATE, 0);
```

Special Notes

None

3.38 R_CELLULAR_GetCertificate()

Reads a certificate or secret key from the non-volatile memory of the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_GetCertificate (
    st_cellular_ctrl_t * const p_ctrl,
    const e_cellular_nvm_type_t data_type,
    const uint8_t index,
    st_cellular_certificate_t * const p_cert
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
data_type (IN)	*Data type to be written (specified as certificate = 0 or private key = 1)*
index (IN)	*Index corresponding to the written data (setting value = 0 to 19)*
p_cert (IN/OUT)	*Pointer to st_cellular_certificate_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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Reads a certificate or secret key from the non-volatile memory of the Cellular module and saves it in the st_cellular_certificate_t structure referenced by the pointer *p_cert*. The member in st_cellular_certificate_t structure is listed in Table 3.19. The data size obtained is up to 2048 bytes.

Refer to section 3.36 for the value that may be set for the *data_type*.

Table 3.19 Member in structures for obtaining certificate or secret key

Member in st_cellular_certificate_t structure	
uint8_t certificate[]	Certificate or secret key

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_certificate_t cert = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_GetCertificate (&cellular_ctrl,
                                 CELLULAR_NVM_TYPE_CERTIFICATE, 0, &cert);
```

Special Notes

None

3.39 R_CELLULAR_ConfigSSLProfile()

Configures SSL security profile.

Format

```
e_cellular_err_t R_CELLULAR_ConfigSSLProfile (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t security_profile_id,
    const e_cellular_cert_validate_level_t cert_valid_level,
    const uint8_t ca_certificate_id,
    const uint8_t client_certificate_id,
    const uint8_t client_privatekey_id
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
security_profile_id (IN)	*Security profile identifier (setting value = 1 to 6)*
cert_valid_level (IN)	*Server certificate validation level*
ca_certificate_id (IN)	*Trusted Certificate Authority certificate identifier*
	(Setting value = 0 to 19, otherwise no certificate is referenced)
client_certificate_id (IN)	*Client certificate identifier*
	(Setting value = 0 to 19, otherwise no certificate is referenced)
client_privatekey_id (IN)	*Client private key identifier*
	(Setting value = 0 to 19, otherwise no key is referenced)

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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Configures SSL security profile. Each security profile contains the certificate and the secret key stored using R_CELLULAR_WriteCertificate().

The security profile identifiers configured are used with R_CELLULAR_FirmUpgrade() and R_CELLULAR_FirmUpgradeBlocking().

The value that may be set for the *cert_valid_level* as follows.

```
typedef enum
{
    CELLULAR_NO_CERT_VALIDATE                   = 0,    //certificate not validated
    CELLULAR_VALIDATE_CERT_EXPDATE              = 1,    //Validate certificate chain, validity period
    CELLULAR_VALIDATE_CERT_EXPDATE_CN = 5,
                                        //Validate certificate chain, validity period, common name
} e_cellular_cert_validate_level_t;
```

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};
st_cellular_certificate_t cert = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);

/* Connects to an SSL server using a host name (e.g., www.renesas.com) */
ret = R_CELLULAR_ConfigSSLProfile(&cellular_ctrl,
                                  1,
                                  CELLULAR_VALIDATE_CERT_EXPDATE_CN,
                                  0,  //CA certificate identifier (required)
                                  255,
                                  255);

/* Connects to a server that requires a client certificate */
ret = R_CELLULAR_ConfigSSLProfile(&cellular_ctrl,
                                  1,
                                  CELLULAR_VALIDATE_CERT_EXPDATE,
                                  0,  //CA certificate identifier
                                  1,  //Client certificate identifier (required)
                                  1); //Client private key identifier (required)

/* Connects to a server that requires no client certificate */
ret = R_CELLULAR_ConfigSSLProfile(&cellular_ctrl,
                                  1,
                                  CELLULAR_NO_CERT_VALIDATE,
                                  255,  //Not used
                                  255,  //Not used
                                  255); //Not used
```

Special Notes

None

3.40 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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API 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 **Completed Cellular module and FIT module initialization** 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. To use autoconnect mode, R_CELLULAR_APConnect() must be completed successfully once.

If using a SIM with a PIN code, R_CELLULAR_UnlockSIM() should be run to enter the PIN code after running this function. Refer to section 3.35 for details.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_AutoConnectConfig(&cellular_ctrl,
                                   CELLULAR_ENABLE_AUTO_CONNECT);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
ret = R_CELLULAR_SoftwareReset(&cellular_ctrl);
```

Special Notes

None

3.41 R_CELLULAR_HardwareReset()

Resets the RYZ014A Cellular module with the RESET pin.

Format

```
e_cellular_err_t R_CELLULAR_HardwareReset (
    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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*
CELLULAR_ERR_RECV_TASK	*/* Serial reception failure */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Resets the Cellular module with the RESET pin.

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 **Completed Cellular module and FIT module initialization** status in Figure 1.3. R_CELLULAR_Open() should be run following R_CELLULAR_Close() after running this function.

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

If using a SIM with a PIN code, R_CELLULAR_UnlockSIM() should be run to enter the PIN code after running this function. Refer to section 3.35 for details.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_AutoConnectConfig(&cellular_ctrl,
                                   CELLULAR_ENABLE_AUTO_CONNECT);
ret = R_CELLULAR_APConnect(&cellular_ctrl, NULL);
ret = R_CELLULAR_HardwareReset(&cellular_ctrl);
```

Special Notes

None

3.42 R_CELLULAR_FactoryReset()

Performs a factory reset to reset the RYZ014A Cellular module to the factory default configuration.

Format

```
e_cellular_err_t R_CELLULAR_ FactoryReset (
    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_MODULE_TIMEOUT	*/* Timeout communicating with Cellular module */*
CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING	*/* Another AT command is running */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*
CELLULAR_ERR_IRQ_OPEN	*/* IRQ port initialization failure */*
CELLULAR_ERR_RECV_TASK	*/* Serial reception failure */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Reset the Cellular module to the factory default configuration.

This function uses an unused PDP context identifier <cid> to confirm that the processing has been completed successfully. This function should be run with one or more <cid>s unused. The <cid>=8 is used if all <cid>s are used. Refer to “8.8 Define PDP Context: AT+CGDCONT” of **RYZ014 Modules User's Manual: AT Command** for details of <cid>.

R_CELLULAR_Open() should be run following R_CELLULAR_Close() after running this function.

Creating an OEM restoration point is not supported by the FIT module.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_FactoryReset(&cellular_ctrl);
```

Special Notes

None

3.43 R_CELLULAR_RTS_Ctrl()

Controls the RTS0 pin of the RYZ014A Cellular module.

Format

```
e_cellular_err_t R_CELLULAR_RTS_Ctrl (
    st_cellular_ctrl_t * const p_ctrl,
    const uint8_t lowhigh
)
```

Parameters

p_ctrl (IN/OUT)	*Pointer to st_cellular_ctrl_t structure defined by the user*
lowhigh (IN)	*Logic level of the RTS0 pin*

Return values

CELLULAR_SUCCESS	*/* Normal end */*
CELLULAR_ERR_PARAMETER	*/* Invalid argument value */*
CELLULAR_ERR_NOT_OPEN	*/* Open function has not been run */*
CELLULAR_ERR_OTHER_API_RUNNING	*/* Another API is running */*

Properties

Prototype declarations are contained in r_cellular_if.h.

Description

Controls the RTS0 pin of the Cellular module.

In order to set the RTS0 pin to high-level signal output, set the argument *lowhigh* to 1. In order to set the RTS0 pin to low-level signal output, set the argument *lowhigh* to 0.

Reentrant

Reentrant operation is not possible.

Thread Safety

This function is thread safe.

Examples

```
e_cellular_err_t ret;
st_cellular_ctrl_t cellular_ctrl = {0};

ret = R_CELLULAR_Open(&cellular_ctrl, NULL);
ret = R_CELLULAR_RTS_Ctrl (&cellular_ctrl, 1);  //RTS0 is high
ret = R_CELLULAR_RTS_Ctrl (&cellular_ctrl, 0);  //RTS0 is low
```

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 e^2 studio Ver.2022-10
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. 2022002-GNURX Toolchain
Endian order		Little endian
Revision of the module		Rev1.11
Board used		Renesas CK-RX65N Cloud Kit (product No.: RTK5CK65N0SxxxxxBE)
		Renesas RX72N Envision Kit (product No.: RTK5RX72N0C00000BJ)
RTOS	FreeRTOS	10.4.3-rx-1.0.1 (kernel only)
	Azure RTOS	v6.1.12_rel-rx-1.0.0
FIT	BSP FIT	Ver 7.10
	SCI FIT	Ver 4.30
	IRQ FIT	Ver 4.00

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 e^2 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.

(3) Q: I have run R_CELLULAR_APConnect() in order to connect to an access point. Then, connection to the access point fails and a value of CELLULAR_ERR_AP_CONNECT_FAILED is returned. I checked the unsolicited result codes notified while running R_CELLULAR_APConnect() using a callback function, then found that an unsolicited result code “+CEREG: 80” is notified.

A: The LTE bands used may not be suitable. Run R_CELLULAR_SetBand() to configure suitable list of LTE bands for the region where the Cellular module is used. Refer to section 3.30 for details.

4.3 Recovery operation

If the events described in this section are encountered when using the RYZ014A Cellular module and the FIT module, perform the recovery operation.

4.3.1 RYZ014A Cellular module notifying unsolicited result code ^EXIT

The Cellular module might perform a self-reset with notifying an unsolicited result code ^EXIT. It is recommended to detect the ^EXIT with a callback function. If the ^EXIT is detected, the following operation should be run.

(1) Detect an unsolicited result code +SYSSTART.
The +SYSSTART following the ^EXIT should be detected.
(2) Run R_CELLULAR_Close().
(3) Run R_CELLULAR_Open().

The operation will recover the Cellular module and the FIT module, the FIT module going to **Completed Cellular module and FIT module initialization** status in Figure 1.3.

4.3.2 RYZ014A Cellular module notifying unsolicited result code +SYSSTART

The Cellular module notifies an unsolicited result code +SYSSTART when it is ready to operate. An unsolicited result code +SYSSTART is notified with a reboot while running the following functions including R_CELLULAR_Open().

- R_CELLULAR_Open()
- R_CELLULAR_SetPSM()
- R_CELLULAR_SetOperator()
- R_CELLULAR_SetBand()
- R_CELLULAR_FirmUpgradeBlocking()
- R_CELLULAR_SoftwareReset()
- R_CELLULAR_HardwareReset()
- R_CELLULAR_FactoryReset()

If the Cellular module reboots unexpectedly, an unsolicited result code +SYSSTART is notified while running the above functions as well. It is recommended to detect the +SYSSTART with a callback function. If getting the unexpected reboots, the following operation should be run.

(1) Run R_CELLULAR_Close().
(2) Run R_CELLULAR_Open().

The operation will recover the Cellular module and the FIT module, the FIT module going to **Completed Cellular module and FIT module initialization** status in Figure 1.3.

4.3.3 Getting timeout on API processing

If an API processing times out, a value of CELLULAR_ERR_MODULE_TIMEOUT is returned to notify the user of the timeout. If getting the timeout, the following operation should be run.

(1) Run R_CELLULAR_HardwareReset().
Resets the Cellular module with the RESET pin.
(2) Run R_CELLULAR_Close().
(3) Run R_CELLULAR_Open().

The operation will recover the Cellular module and the FIT module, the FIT module going to **Completed Cellular module and FIT module initialization** status in Figure 1.3.

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.)
RYZ014 Module System Integration Guide (R19AN0074)
(The latest versions can be downloaded from the Renesas Electronics website.)
RYZ014 Modules User's Manual: AT Command (R11UZ0093)
(The latest versions 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
1.07	Jul. 21, 2022	6	Added the following APIs to Table 1.1: - R_CELLULAR_SetBand - R_CELLULAR_GetPDPAddress
		9	Added the following macro definitions in section 2.7 Compile Settings: - CELLULAR_CFG_NETWORK_NOTIFY_LEVEL - CELLULAR_CFG_PSM_PREPARATION_TIME - CELLULAR_CFG_RING_LINE_ACTIVE_TIME

		61-62	Added the following APIs in section 3 API Functions: - 3.28 R_CELLULAR_SetBand - 3.29 R_CELLULAR_GetPDPAddress
		Program	The number of available digits of the phone number obtained with R_CELLULAR_GetPhonenum() is modified to up to 15.
1.08	Sep. 30, 2022	6	Added the following APIs to Table 1.1: - R_CELLULAR_HardwareReset - R_CELLULAR_FactoryReset - R_CELLULAR_RTS_Ctrl
		9	Added the following macro definitions in section 2.7 Compile Settings: - CELLULAR_CFG_PSM_WAKEUP_LATENCY - CELLULAR_CFG_URC_CHARGET_ENABLED - CELLULAR_CFG_URC_CHARGET_FUNCTION - CELLULAR_CFG_IRQ_NUM
		14	Added the following structures in section 2.9 Parameters: - st_cellular_notice_t - st_cellular_edrx_config_t - st_cellular_psm_config_t
		16-17	Revised the contents in section 3.1 R_CELLULAR_Open owing to the modified specification of callback function. The callback function can be set by Table 2.1 Configuration options (r_cellular_config.h).
		18	Revised the description part in section 3.2 R_CELLULAR_Close owing to the modified specification of R_CELLULAR_Close().
		41-43	Revised the contents in section 3.15 R_CELLULAR_SetEDRX owing to the modified argument of R_CELLULAR_SetEDRX().
		44-46	Revised the contents in section 3.16 R_CELLULAR_SetPSM owing to the modified argument of R_CELLULAR_SetPSM().
		53-58	Revised the contents in the following sections owing to the modified specification of callback function: - 3.23 R_CELLULAR_Ping - 3.24 R_CELLULAR_GetAPConnectState - 3.25 R_CELLULAR_GetCellInfo
		65-66	Moved section 3.27 R_CELLULAR_SoftwareReset to section 3.29.
		67-70	Added the following APIs in section 3 API Functions: - 3.30 R_CELLULAR_HardwareReset - 3.31 R_CELLULAR_FactoryReset - 3.32 R_CELLULAR_RTS_Ctrl
		72	Added the following section: - 4.3 Recovery operation
1.09	Nov. 30, 2022	4	Revised Figure 1.1.
		9-10	Revised the default values of the following macro definition in section 2.7: - CELLULAR_CFG_AP_NAME - CELLULAR_CFG_AUTH_TYPE Revised the following macro definition in section 2.7: - CELLULAR_CFG_SCI_PRIORITY Added the following macro definitions in section 2.7: - CELLULAR_CFG_CTS_SW_CTRL - CELLULAR_CFG_CTS_PORT - CELLULAR_CFG_CTS_PIN - CELLULAR_CFG_PFS_SET_VALUE
		14	Added the following structures in section 2.9 Parameters:

			- st_cellular_ipaddr_t - st_cellular_ping_cfg_t
		16-80	All API functions except R_CELLULAR_Open() defined in section 3 are made thread safe. Added the following return value to the thread-safe API functions: - CELLULAR_ERR_OTHER_API_RUNNING
		19	Revised the contents in the following section owing to the FIT module supporting IPv6: - 3.3 R_CELLULAR_APConnect
		25	Revised the contents in the following section owing to the FIT module supporting IPv6: - 3.6 R_CELLULAR_CreateSocket Added the following return value to R_CELLULAR_CreateSocket(): - CELLULAR_ERR_SIM_NOT_SUPPORT_IPV6
		27	Revised the contents in the following section owing to the FIT module supporting IPv6: - 3.7 R_CELLULAR_ConnectSocket
		37	Revised the contents in the following section owing to the FIT module supporting IPv6: - 3.12 R_CELLULAR_DnsQuery Added the following return value to R_CELLULAR_DnsQuery(): - CELLULAR_ERR_SIM_NOT_SUPPORT_IPV6
		49	Revised the description part in section 3.17 R_CELLULAR_GetICCID to add that of the number of available digits of ICCID.
		51	Revised the description part in section 3.18 R_CELLULAR_IMEI to add that of the number of available digits of IMEI.
		53	Revised the description part in section 3.19 R_CELLULAR_IMSI to add that of the number of available digits of IMSI.
		55	Revised the description part in section 3.20 R_CELLULAR_GetPhonenum to add that of the number of available digits of phone number.
		61	Revised the contents in the following section owing to R_CELLULAR_Ping() being configurable: - 3.23 R_CELLULAR_Ping
		71	Revised the contents in the following section owing to the FIT module supporting IPv6: - 3.28 R_CELLULAR_GetPDPAddress Added the following return value to R_CELLULAR_GetPDPAddress(): - CELLULAR_ERR_SIM_NOT_SUPPORT_IPV6
1.10	Apr. 7, 2023	-	Removed the descriptions for AzureRTOS.
		4-9	Revised the contents in section 1.2 Overview of RYZ014A Cellular FIT Module: Added the description for implementation type C. Added the description for WDT. Revised the description in section 1.2.1. Revised Figure 1.3 in section 1.2.4.
		11	Revised the contents in section 2.7 Compile Settings: Revised the default value of the following macro: - CELLULAR_CFG_ATC_RETRY_CGATT Revised the following macro definition: - CELLULAR_CFG_ATC_RETRY_CGATT
		18-20	Revised the contents in section 3.1 R_CELLULAR_Open(): Added the description for procedure when receiving unsolicited result code +SYSSTART. Revised the description for the default values used when *p_cfg* is NULL.
		21-22	Revised the contents in section 3.2 R_CELLULAR_Close():

			Added a return value in Return Value part. Revised the description in Description part. Revised the description in Examples part.
		37	Revised the contents in section 3.10 R_CELLULAR_SendSocket(): Added the description for procedure when CELLULAR_ERR_MODULE_TIMEOUT is returned.
		50-51	Added the following API in section 3 API Functions: - 3.16 R_CELLULAR_GetEDRX()
		52-53	Revised the contents in section 3.17 R_CELLULAR_SetPSM(): Added a return value in Return Value part. Revised the description in Description part.
		56-57	Added the following API in section 3 API Functions: - 3.18 R_CELLULAR_GetPSM()
		79-80	Added the following API in section 3 API Functions: - 3.29 R_CELLULAR_SetOperator()
		81	Revised the contents in section 3.30 R_CELLULAR_SetBand(): Revised the description in Description part.
		85-101	Added the following API in section 3 API Functions: - 3.32 R_CELLULAR_FirmUpgrade() - 3.33 R_CELLULAR_FirmUpgradeBlocking() - 3.34 R_CELLULAR_GetUpgradeState() - 3.35 R_CELLULAR_UnlockSIM() - 3.36 R_CELLULAR_WriteCertificate() - 3.37 R_CELLULAR_EraseCertificate() - 3.38 R_CELLULAR_GetCertificate() - 3.39 R_CELLULAR_ConfigSSLProfile()
		104	Revised the contents in section 3.41 R_CELLULAR_HardwareReset(): Added a return value in Return Value part.
		106	Revised the contents in section 3.42 R_CELLULAR_FactoryReset (): Added a return value in Return Value part.
		111	Revised the contents in section 4.2 Troubleshooting: Added a new item (3).
		112	Revised the contents in section 4.3 Recovery operation: Added section 4.3.2 RYZ014A Cellular module notifying unsolicited result code +SYSSTART
1.11	May. 31, 2023	27	Revised the contents in section 3.5 R_CELLULAR_Disconnected(): Removed a return value in Return Value part.
		29	Revised the contents in section 3.6 R_CELLULAR_CreatedSocket(): Removed a return value in Return Value part. Revised the description in Description part.
		33	Revised the contents in section 3.8 R_CELLULAR_ShutdownSocket(): Removed a return value in Return Value part.
		35	Revised the contents in section 3.9 R_CELLULAR_CloseSocket(): Removed a return value in Return Value part.
		37	Revised the contents in section 3.10 R_CELLULAR_SendSocket(): Revised the minimum value of *timeout_ms* to 1.
		39	Revised the contents in section 3.11 R_CELLULAR_ReceivedSocket(): Removed a return value in Return Value part. Revised the description in Description part.
		Program	Revised R_CELLULAR_SendSocket() implementation: If an error response is received from the Cellular module, the function aborts without waiting until timeout, and the number of bytes that has been sent is returned as the return value. Revised R_CELLULAR_ReceiveSocket() implementation: When *timeout_ms* is set to 0 (no timeout), if the socket is disconnected from an access point for a certain period of time, the function aborts, and the number of bytes that has been received is returned as the return value.

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

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

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

www.renesas.com

Trademarks

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

Contact information

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