US159-DA16XXXMEVZ Wi-Fi Control Module Using Firmware Integration Technology

Introduction

This application note describes the usage of the US159-DA16XXXMEVZ Wi-Fi control module, which conforms to the Firmware Integration Technology (FIT) standard.

In the following pages, the US159-DA16XXXMEVZ Wi-Fi control module software is referred to collectively as “the DA16XXX Wi-Fi FIT module” or “the FIT module.”

The FIT module supports the following Wi-Fi Pmod modules:

- DA16200MOD (US159-DA16200MEVZ)
- DA16600MOD (US159-DA16600MEVZ)

In the following pages, the DA16XXXMOD is referred to as “the Wi-Fi module”. The DA16200 and DA16600 products will collectively be referred to as “DA16XXX”.

Target Device

- RX Family
- RX600 Series
- RX65N Group

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

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX

For details of the confirmed operation contents of each compiler, refer to 6.1 Confirmed Operation Environment.

Related Documents

[1] Firmware Integration Technology User’s Manual (R01AN1833)
[2] RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
[3] RX Smart Configurator User’s Guide: e² studio (R20AN0451)
[4] RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
[5] RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)

Contents

1. Overview

1.1. DA16XXX FIT Module

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

1.2. Overview of the DA16XXX Wi-Fi FIT Module

DA16XXX is a low-power Wi-Fi networking SoC that delivers a dramatic breakthrough in battery life even for devices that are continuously connected to the Wi-Fi network. The module comes readily equipped with radio certification for Japan, North America, and Europe.

The Wi-Fi FIT module supplies these features:

- Supports connect/disconnect to a b/g/n (2.4GHz) Wi-Fi Access Point using Open, WPA, and WPA2 security. Encryption types can be either TKIP, or CCMP(AES).
- Supports retrieval of the module device MAC address.
- Supports retrieval of the module device IP address once connected to an Access Point.
- Supports a Wi-Fi network scan capability to get a list of local Access Points.
- Supports a Ping function to test network connectivity.
- Supports a DNS Query call to retrieve the IPv4 address of a supplied URL.
- Supports a SNTP Client to synchronize the local time with a server that provides time services.
- Supports TCP client sockets.
- Supports TLS client for secure sockets.
- Supports MQTT on-chip client.
  - Supports connect/disconnect to an MQTT broker via hostname, port, and user credentials.
  - Supports unsecure and secure connection via TLS encryption.
  - Supports the MQTT subscribe/publish model for multiple topics.
  - Supports other optional configurations such as MQTT v3.1.1, Quality-of-service (QoS) level, TLS cipher suites, and ALPNs.
- Supports HTTP on-chip client.
  - Supports sending a request header (GET, PUT, and POST) to an HTTP server and receiving a response header.
  - Supports unsecure and secure connection via TLS encryption.
  - Supports parsing of the response header and returning to the user.
  - Supports other optional configurations such as Server Name Indication (SNI) and ALPNs.
- Supports 1 UART for interfacing with the DA16XXX module.
- Supports FreeRTOS-based user applications.

1.2.1. Connection with the DA16XXX Wi-Fi Module

Examples of connection to the DA16XXX Wi-Fi module are shown below.

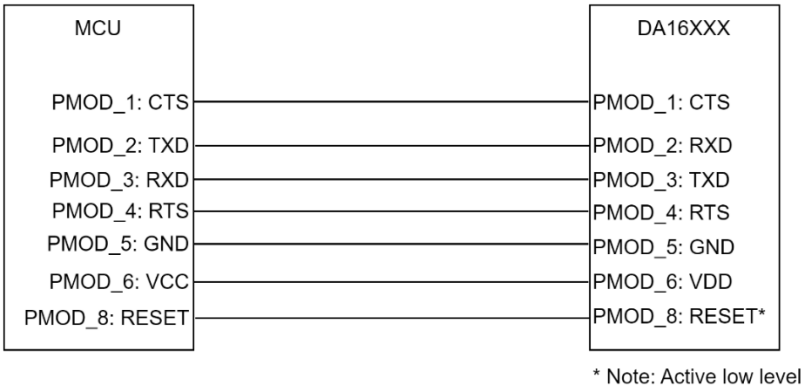

Figure 1.1 Example Connection to the DA16XXX Wi-Fi Module

1.2.2. Software Configuration

Figure 1.2 shows the software configuration.

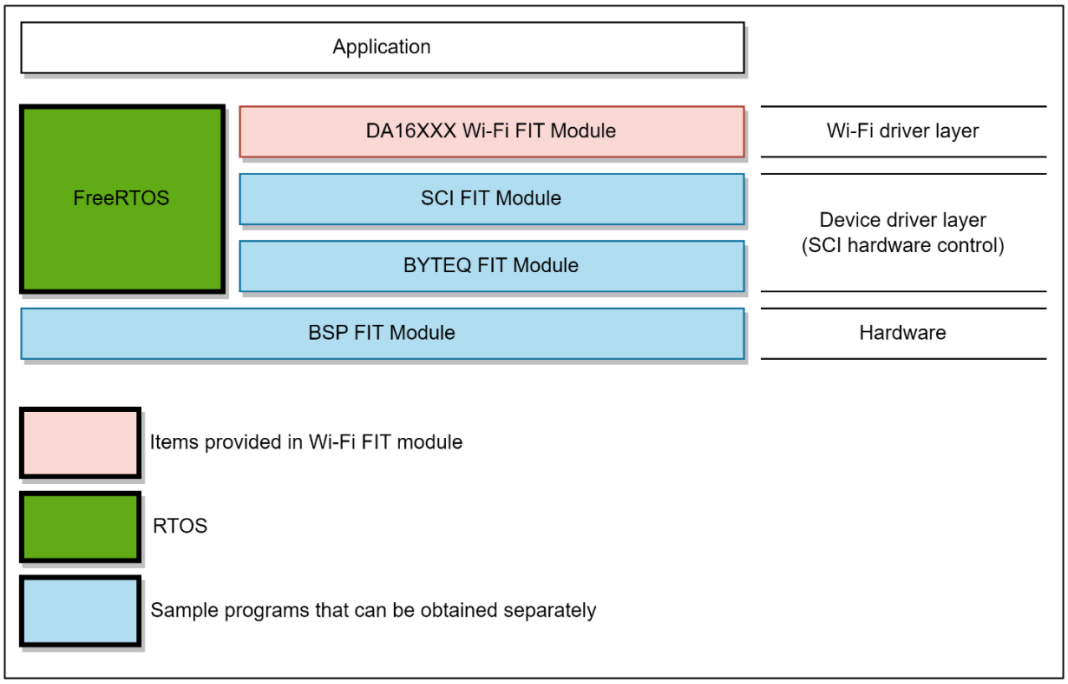

Figure 1.2 Software Configuration Diagram

1. DA16XXX Wi-Fi FIT module
   The FIT module. This software is used to control the Wi-Fi module.
2. SCI FIT module
   Implements communication between the Wi-Fi module and the MCU. A sample program is available.
   Refer to “Related Documents” on page 1 and obtain the software.
3. 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.
4. BSP FIT module
   The Board Support Package module. A sample program is available.
   Refer to “Related Documents” on page 1 and obtain the software.
5. RTOS
   The RTOS manages the system overall. Operation of the FIT module has been verified using FreeRTOS.

1.3. API Overview

Table 1.1 lists the API functions included in the FIT module. The required memory sizes are lists in 2.8 Code Size.

Table 1.1 API Functions

Function	Function Description
Wi-Fi Common API	
R_WIFI_DA16XXX_Open()	Initialize the Wi-Fi module
R_WIFI_DA16XXX_IsOpened()	Check Wi-Fi is opened
R_WIFI_DA16XXX_Close()	Close the Wi-Fi module
R_WIFI_DA16XXX_Ping()	Pings a specified IP address
R_WIFI_DA16XXX_Scan()	Scan Access points
R_WIFI_DA16XXX_Connect()	Connects to an access point
R_WIFI_DA16XXX_Disconnect()	Disconnects from an access point
R_WIFI_DA16XXX_IsConnected()	Check connected access point
R_WIFI_DA16XXX_DnsQuery()	Execute DNS query
R_WIFI_DA16XXX_SntpServerIpAddressSet()	Set SNTP server IP address
R_WIFI_DA16XXX_SntpEnableSet()	Enable or disable SNTP client service
R_WIFI_DA16XXX_SntpTimeZoneSet()	Set SNTP time zone
R_WIFI_DA16XXX_LocalTimeGet()	Get the local time based on current time zone
R_WIFI_DA16XXX_SetDnsServerAddress()	Set DNS Server Address
R_WIFI_DA16XXX_GetMacAddress()	Get MAC Address
R_WIFI_DA16XXX_GetIpAddress()	Get IP Address
R_WIFI_DA16XXX_HardwareReset()	Reset the Wi-Fi module
R_WIFI_DA16XXX_GetVersion()	Returns version information for the module
Wi-Fi TCP Client API	
R_WIFI_DA16XXX_GetAvailableSocket()	Get the next available socket ID
R_WIFI_DA16XXX_GetSocketStatus()	Get the socket status
R_WIFI_DA16XXX_CreateSocket()	Create a new socket instance
R_WIFI_DA16XXX_TcpConnect()	Connect to a specific IP and Port using socket
R_WIFI_DA16XXX_SendSocket()	Send data on connecting socket
R_WIFI_DA16XXX_ReceiveSocket()	Receive data on connecting socket
R_WIFI_DA16XXX_CloseSocket()	Disconnect a specific socket connection
R_WIFI_DA16XXX_TcpReconnect()	Reconnect TCP socket
Wi-Fi TLS Client API	
R_WIFI_DA16XXX_GetAvailableTlsSocket()	Get the next available socket ID
R_WIFI_DA16XXX_GetTlsSocketStatus()	Get the socket status
R_WIFI_DA16XXX_CreateTlsSocket()	Create a new socket instance
R_WIFI_DA16XXX_TlsConnect()	Connect to a specific IP and Port using socket
R_WIFI_DA16XXX_SendTlsSocket()	Send data on connecting socket
R_WIFI_DA16XXX_ReceiveTlsSocket()	Receive data on connecting socket
R_WIFI_DA16XXX_CloseTlsSocket()	Disconnect a specific socket connection

R_WIFI_DA16XXX_TlsReconnect()	Reconnect TLS socket
R_WIFI_DA16XXX_RegistServerCertificate()	Register server certificate on Wi-Fi module
R_WIFI_DA16XXX_RequestTlsSocket()	Request TLS socket communication
R_WIFI_DA16XXX_GetServerCertificate()	Get stored server certificate on Wi-Fi module
R_WIFI_DA16XXX_WriteCertificate()	Write certificate on Wi-Fi module
R_WIFI_DA16XXX_DeleteCertificate()	Delete certificate on Wi-Fi module
Wi-Fi MQTT On-chip Client API	
R_WIFI_DA16XXX_MqttOpen()	Initialize MQTT on-chip Client service
R_WIFI_DA16XXX_MqttDisconnect()	Disconnect from MQTT on-chip Client service
R_WIFI_DA16XXX_MqttConnect()	Configure and connect the MQTT on-chip Client service
R_WIFI_DA16XXX_MqttPublish()	Publish a message for a given MQTT topic
R_WIFI_DA16XXX_MqttSubscribe()	Subscribe to MQTT topics
R_WIFI_DA16XXX_MqttUnSubscribe()	Unsubscribe from MQTT topics
R_WIFI_DA16XXX_MqttReceive()	Receive data subscribed from MQTT Client service
R_WIFI_DA16XXX_MqttClose()	Close the MQTT on-chip Client service
Wi-Fi HTTP On-chip Client API	
R_WIFI_DA16XXX_HttpOpen()	Initialize the HTTP on-chip Client service
R_WIFI_DA16XXX_HttpClose()	Close the HTTP Client service
R_WIFI_DA16XXX_HttpSend()	Send the HTTP request with the configured buffers

1.4. Status Transitions

1.4.1. Status Transitions of TCP Client

Figure 1.3 shows the status transitions of the FIT module up to communication status using TCP sockets.

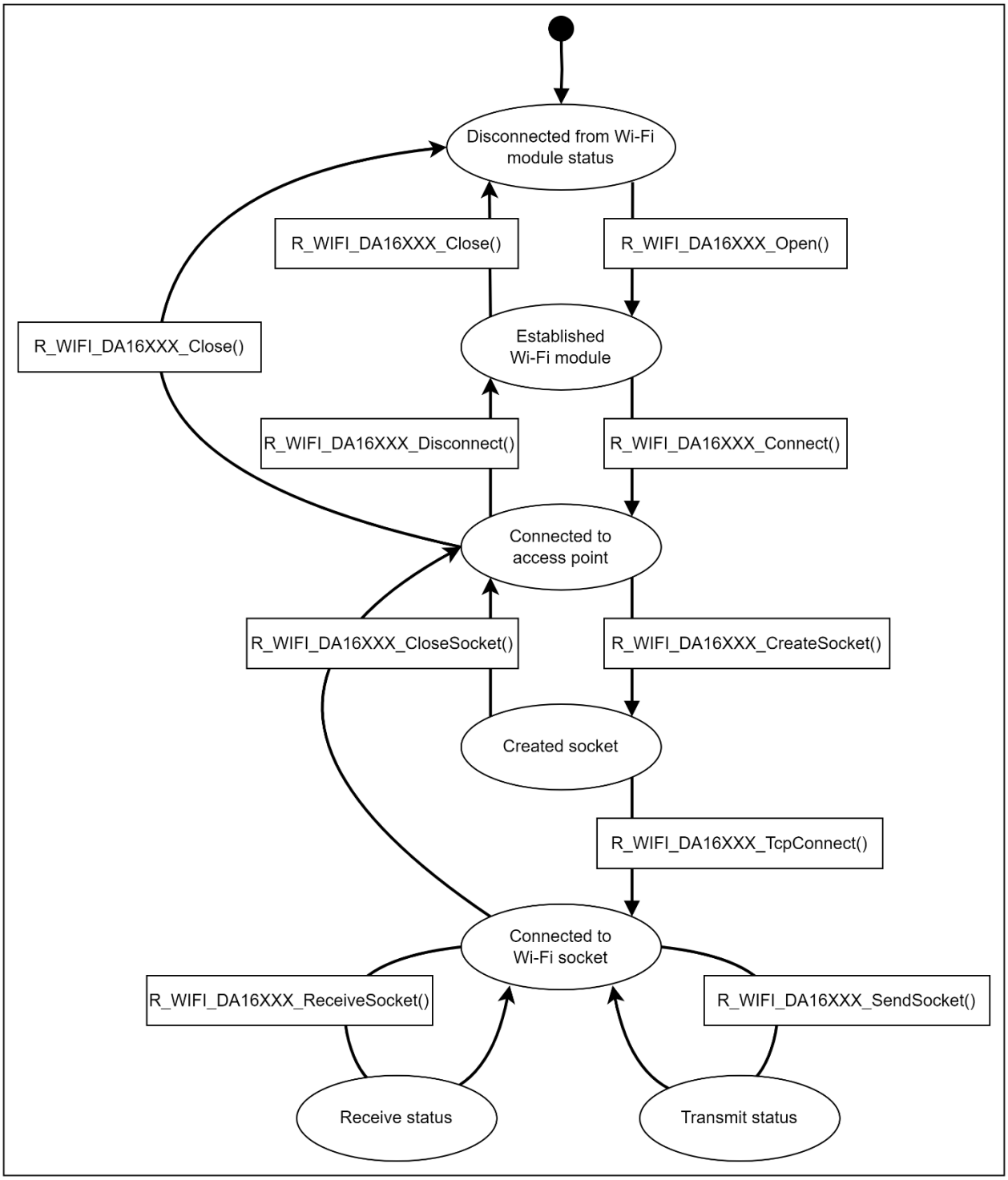

Figure 1.3 Status Transitions When Using TCP Socket

1.4.2. Status Transitions of TLS Client

Figure 1.4 shows the status transitions of the FIT module up to communication status using TLS sockets.

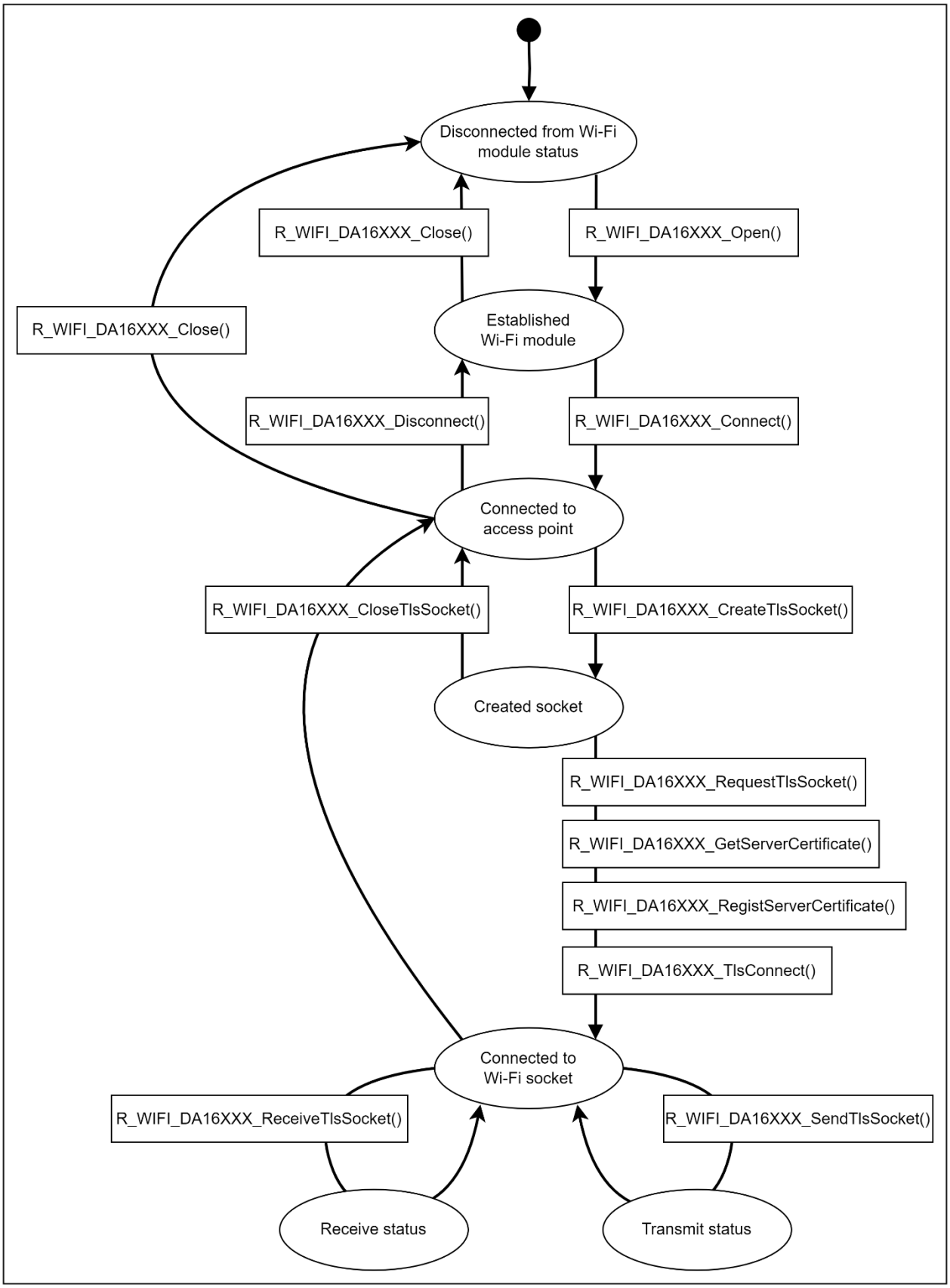

Figure 1.4 Status Transitions When Using TLS Socket

1.4.3. Status Transitions of MQTT On-Chip Client

Figure 1.5 shows the status transitions of the FIT module up to communication status using the MQTT on-chip client.

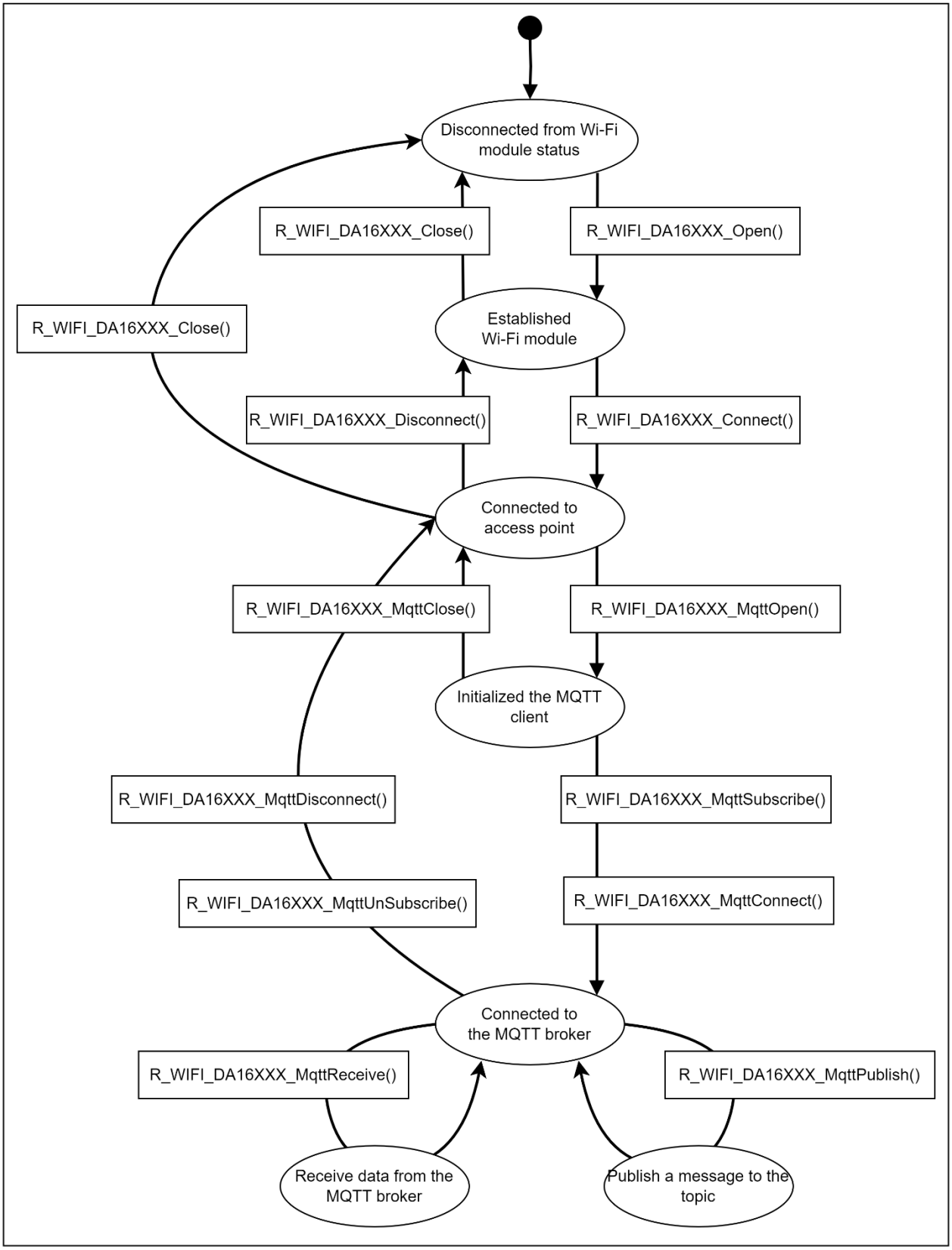

Figure 1.5 Status Transitions When Using the MQTT On-Chip Client

1.4.4. Status Transitions of HTTP On-Chip Client

Figure 1.6 shows the status transitions of the FIT module up to communication status using the HTTP on-chip client.

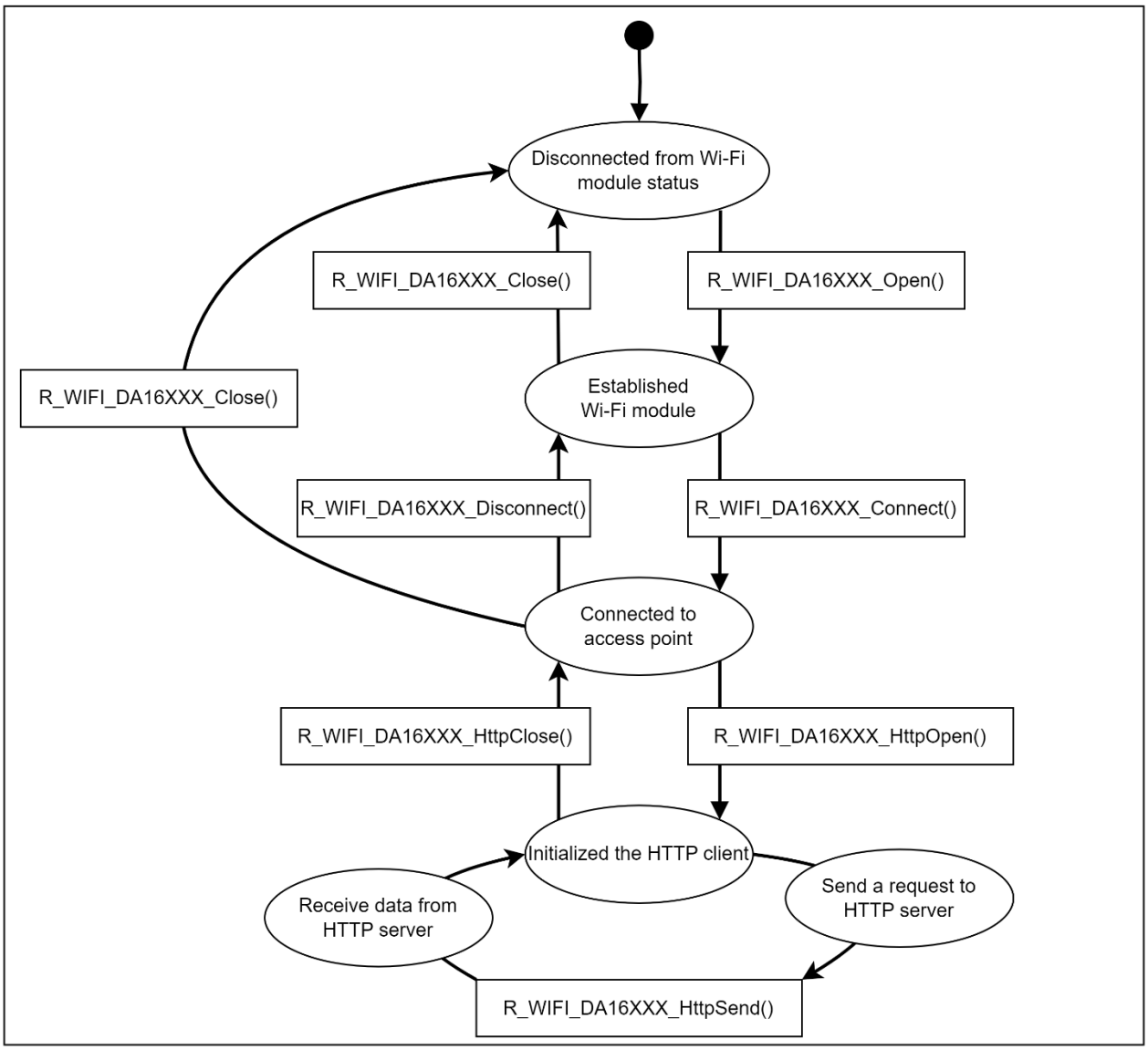

Figure 1.6 Status Transitions When Using the HTTP On-Chip Client

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

2.2. Software Requirements

The driver is dependent upon the following FIT modules:

- r_bsp
- r_sci_rx
- r_byteq_rx
- FreeRTOS

2.3. Supported Toolchain

The FIT module has been confirmed to work with the toolchain listed in 6.1 Confirmed Operation Environment.

2.4. Interrupt Vector

None

2.5. Header Files

All API calls and their supporting interface definitions are located in r_wifi_da16xxx_if.h.

2.6. Integer Types

This project 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_wifi_da16xxx_config.h.
The names of the options and their setting values are listed in the table below.

Table 2.1 Configuration Options (r_wifi_da16xxx_config.h)

Configuration Options in r_wifi_da16xxx_config.h	
Wi-Fi common configuration	
WIFI_CFG_DA16600_SUPPORT Default: “0”	Use DA16600 module. 1 = enabled, 0 = disabled.
WIFI_CFG_SCI_CHANNEL Default: “6”	SCI Channel for AT command communication. Set this option to match the SCI port to be controlled.
WIFI_CFG_SCI_INTERRUPT_LEVEL Default: “4”	Interrupt priority of the serial module used for communication with the Wi-Fi module. Set this option to a value of 1 to 15 to match the system priority.
WIFI_CFG_SCI_PCLK_HZ Default: “60000000”	Peripheral clock speed for WIFI_CFG_SCI_CHANNEL
WIFI_CFG_SCI_BAUDRATE Default: “115200”	Communication baud rate for WIFI_CFG_SCI_CHANNEL. Set this option to a value of 115200, 230400, 460800 or 921600.
WIFI_CFG_CTS_SW_CTRL Default: “1”	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.
WIFI_CFG_CTS_PORT Default: “J”	Configures the port direction register (PDR) setting for the general port that controls the CTS pin of the Wi-Fi module. Set this option to match the port to be controlled. This option takes effect when WIFI_CFG_CTS_SW_CTRL is set to 1.
WIFI_CFG_CTS_PIN Default: “3”	Configures the port output data register (PODR) setting for the general port that controls the CTS pin of the Wi-Fi module. Set this option to match the port to be controlled. This option takes effect when WIFI_CFG_CTS_SW_CTRL is set to 1.
WIFI_CFG_RTS_PORT Default: “J”	Configures the port direction register (PDR) setting for the general port that controls the RTS pin of the Wi-Fi module. Set this option to match the port to be controlled.
WIFI_CFG_RTS_PIN Default: “3”	Configures the port output data register (PODR) setting for the general port that controls the RTS pin of the Wi-Fi module. Set this option to match the port to be controlled.
WIFI_CFG_PFS_SET_VALUE Default: “0x0AU”	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 Wi-Fi module. Set this option to match the pin to be used. This option takes effect when WIFI_CFG_CTS_SW_CTRL is set to 1.
WIFI_CFG_RESET_PORT Default: “5”	Configures the port direction register (PDR) setting for the general port that controls the RESET pin of the Wi-Fi module. Set this option to match the port to be controlled.
WIFI_CFG_RESET_PIN Default: “5”	Configures the port output data register (PODR) setting for the general port that controls the RESET pin of the Wi-Fi module. Set this option to match the port to be controlled.
WIFI_CFG_AT_CMD_TX_BUFFER_SIZE Default: “512”	AT command transfer buffer size. Set this value in range from 1 to 8192.
WIFI_CFG_AT_CMD_RX_BUFFER_SIZE Default: “3000”	AT command receive buffer size. Set this value in range from 1 to 8192.

WIFI_CFG_USE_CALLBACK_FUNCTION Default: “0”	Enables or disables the user Wi-Fi callback function. 0 = Unused, 1 = Used.
WIFI_CFG_CALLBACK_FUNCTION_NAME Default: “NULL”	Specifies function name of the Wi-Fi callback function called when an error occurs.
WIFI_CFG_MAX_SSID_LEN Default: “32”	Configures max SSID Length
WIFI_CFG_MAX_BSSID_LEN Default: “6”	Configures max BSSID Length
WIFI_CFG_SNTP_ENABLE Default: “0”	Enables or disables the SNTP client service. 1 = enabled, 0 = disabled
WIFI_CFG_SNTP_SERVER_IP Default: “0.0.0.0”	Configures SNTP server IP address string.
WIFI_CFG_SNTP_UTC_OFFSET Default: “7”	Configures time zone offset in hours (-12 ~ 12).
WIFI_CFG_COUNTRY_CODE Default: “”	Configures a country code. The country code defined in ISO3166-1 alpha-2 standard.
WIFI_CFG_USE_FREERTOS_LOGGING Default: “0”	Enables or disables FreeRTOS logging. 1 = enabled, 0 = disabled
WIFI_CFG_DEBUG_LOG Default: “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 module communication information in addition.
Wi-Fi TCP Client configuration	
WIFI_CFG_TCP_SUPPORT Default: “1”	Enables or disables TCP protocol. 1 = enabled, 0 = disabled.
WIFI_CFG_TCP_CREATABLE_SOCKETS Default: “1”	Configures the number of TCP client socket. Set this value in range from 1 to 4.
WIFI_CFG_TCP_SOCKET_RECEIVE_BUFFER_SIZE Default: “4096”	Configures the receive buffer size for the socket. Set this value in range from 1 to 8192.
Wi-Fi MQTT on chip configuration	
WIFI_CFG_MQTT_SUPPORT Default: “0”	Enables or disables MQTT on-chip protocol. 1 = enabled, 0 = disabled.
MQTT_CFG_MQTT_CERTS Default: “0”	Flag to use MQTT Certificates. 1 = Used, 0 = Unused.
WIFI_CFG_MQTT_CERTS_HEADER Default: “NULL”	Name of header file that will contain certificates (macros). User must create header file. Example: "cert_storage.h"
WIFI_CFG_MQTT_ROOT_CA Default: “NULL”	Links to user-defined macro of the same name for Root CA which user must define in application header.
WIFI_CFG_MQTT_CLIENT_CERT Default: “NULL”	Links to user-defined macro of the same name for client certificate which user must define in application header.
WIFI_CFG_MQTT_PRIVATE_KEY Default: “NULL”	Links to user-defined macro of the same name for private key which user must define in application header.
WIFI_CFG_MQTT_CMD_TX_BUF_SIZE Default: “512”	Configures the MQTT buffer used for sending commands and publishing data. Maximum publishing length is 2063 bytes. Set this value in range from 200 to 2064 and must be less than or equal to WIFI_CFG_AT_CMD_TX_BUFFER_SIZE.
WIFI_CFG_MQTT_CMD_RX_BUF_SIZE Default: “512”	Configures MQTT buffer used for receiving subscribed data. Set this value in range from 1 to 3000 and must be less than or equal to WIFI_CFG_AT_CMD_TX_BUFFER_SIZE.

WIFI_CFG_MQTT_USE_MQTT_V311 Default: “1”	Flag to use MQTT version 3.1.1. 1 = Used, 0 = Unused.
WIFI_CFG_MQTT_RX_TIMEOUT Default: “1000”	Timeout for the MQTT Receive function to check the buffer for incoming MQTT messages in milliseconds
WIFI_CFG_MQTT_TX_TIMEOUT Default: “1000”	Timeout for publishing MQTT messages in milliseconds.
WIFI_CFG_MQTT_CLEAN_SESSION Default: “1”	Flag to use MQTT clean session. 1 = Used, 0 = Unused.
WIFI_CFG_MQTT_ALPN1 Default: “NULL”	Select 1st Application Layer Protocol Negotiation (ALPN).
WIFI_CFG_MQTT_ALPN2 Default: “NULL”	Select 2nd ALPN.
WIFI_CFG_MQTT_ALPN3 Default: “NULL”	Select 3rd ALPN.
WIFI_CFG_MQTT_KEEP_ALIVE Default: “60”	MQTT ping period to check if connection is still active.
WIFI_CFG_MQTT_CLIENT_IDENTIFIER Default: “NULL”	Configures client identifier.
WIFI_CFG_MQTT_HOST_NAME Default: “NULL”	Configures MQTT Host Name (or IP address).
WIFI_CFG_MQTT_PORT Default: “1883”	Configures MQTT Port for communication.
WIFI_CFG_MQTT_USER_NAME Default: “NULL”	Configures MQTT Username.
WIFI_CFG_MQTT_PASSWORD Default: “NULL”	Configures MQTT Password.
WIFI_CFG_MQTT_WILL_TOPIC Default: “NULL”	Configures Topic for MQTT Last Will message.
WIFI_CFG_MQTT_WILL_MESSAGE Default: “NULL”	Configures Payload for MQTT Last Will message.
WIFI_CFG_MQTT_SNI_NAME Default: “NULL”	Configures Server Name Indication (SNI).
WIFI_CFG_MQTT_WILL_QOS Default: “0”	Configures Quality-of-Service. 0: At most once (QoS 0). 1: At least once (QoS 1). 2: Exactly once (QoS 2).
WIFI_CFG_MQTT_TLS_CIPHER_SUITES Default: “0”	Flag to use TLS Cipher Suites. 1 = Used, 0 = Unused.
WIFI_CFG_MQTT_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA Default: “0”	Select TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
WIFI_CFG_MQTT_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA Default: “0”	Select TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
WIFI_CFG_MQTT_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 Default: “0”	Select TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
WIFI_CFG_MQTT_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 Default: “0”	Select TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
WIFI_CFG_MQTT_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 Default: “0”	Select TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
WIFI_CFG_MQTT_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 Default: “0”	Select TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

WIFI_CFG_MQTT_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA Default: “0”	Select TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
WIFI_CFG_MQTT_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA Default: “0”	Select TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
WIFI_CFG_MQTT_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 Default: “0”	Select TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
WIFI_CFG_MQTT_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 Default: “0”	Select TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
WIFI_CFG_MQTT_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 Default: “0”	Select TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
WIFI_CFG_MQTT_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 Default: “0”	Select TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 Cipher. Unused: 0. Used: WIFI_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
WIFI_CFG_MQTT_P_CALLBACK Default: “1”	Enables or disables the user MQTT callback function. 0 = Unused, 1 = Used.
WIFI_CFG_MQTT_P_CALLBACK_FUNCTION_NAME Default: “mqtt_userCallback”	Specifies function name of the MQTT callback function called when receive data subscribed.
Wi-Fi TLS Client configuration	
WIFI_CFG_TLS_SUPPORT Default: “0”	Enables or disables TLS on-chip protocol. 1 = enabled, 0 = disabled.
WIFI_CFG_TLS_CREATABLE_SOCKETS Default: “1”	Configures the number of TLS client socket. Set this value in range from 1 to 2.
WIFI_CFG_TLS_SOCKET_RECEIVE_BUFFER_SIZE Default: “4096”	Configures the receive buffer size for the socket. Set this value in range from 1 to 8192.
WIFI_CFG_TLS_USE_CA_CERT Default: “1”	Flag to use CA certificates. 0 = Unused, 1 = Used.
WIFI_CFG_TLS_CERT_MAX_NAME Default: “32”	Configures length for certificate’s name.
WIFI_CFG_TLS_CERT_CA_NAME Default: “NULL”	Configures CA certificate name.
WIFI_CFG_TLS_CERT_CLIENT_NAME Default: “NULL”	Configures Client certificate name.
WIFI_CFG_TLS_CERT_PRIVATE_NAME Default: “NULL”	Configures Private certificate name.
Wi-Fi HTTP on chip configuration	
WIFI_CFG_HTTP_SUPPORT Default: “0”	Enables or disables HTTP on-chip protocol. 1 = enabled, 0 = disabled.
WIFI_CFG_HTTP_SNI_NAME Default: “NULL”	Configures Server Name Indication (SNI).
WIFI_CFG_HTTP_ALPN1 Default: “NULL”	Select 1st Application Layer Protocol Negotiation (ALPN).
WIFI_CFG_HTTP_ALPN2 Default: “NULL”	Select 2nd ALPN.
WIFI_CFG_HTTP_ALPN3 Default: “NULL”	Select 3rd ALPN.
WIFI_CFG_HTTP_TLS_AUTH Default: “0”	Configures HTTP TLS Authentication levels. 0: None - No authentication required; accept connections without any form of authentication. 1: Optional - Allow both authenticated and unauthenticated connections. 2: Require - Demand authentication for connections.

WIFI_CFG_HTTP_CERTS_HEADER Default: “NULL”	Name of header file that will contain certificates (macros). User must create header file. Example: "cert_storage.h"
WIFI_CFG_HTTP_ROOT_CA Default: “NULL”	Links to user-defined macro of the same name for Root CA which user must define in application header.
WIFI_CFG_HTTP_CLIENT_CERT Default: “NULL”	Links to user-defined macro of the same name for client certificate which user must define in application header.
WIFI_CFG_HTTP_PRIVATE_KEY Default: “NULL”	Links to user-defined macro of the same name for private key which user must define in application header.

Table 2.2 Configuration Options (r_sci_rx_config.h)

Configuration Options in r_ sci_rx_config.h	
#define SCI_CFG_CHx_INCLUDED Notes: 1. CHx = CH0 to CH12 2. The default values are as follows: CH0 CH2 to CH12: 0, CH1: 1	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.
#define 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 WIFI_CFG_SCI_CHANNEL should be set to 2180.
#define 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 WIFI_CFG_SCI_CHANNEL should be set to 8192.
#define SCI_CFG_TEI_INCLUDED Note: The default is 0.	Enables the transmit end interrupt for serial transmissions. This option should be set to 1.

Table 2.3 Configuration Options (r_bsp_config.h)

Configuration Options in r_ bsp_config.h	
#define BSP_CFG_RTOS_USED Note: The default is 0.	Specifies the type of real-time OS. When using this FIT module, set the following. FreeRTOS:1

2.8. Code Size

Typical code sizes associated with this module are listed below.
The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7 Compile Settings. The table lists reference values when the C compiler's compile options are set to their default values, as described in 2.3 Supported Toolchain. The compile option default values are optimization level: 2, optimization type: for size, and data endianness: little-endian. The code size varies depending on the C compiler version and compile options.

The values in the table below are confirmed under the following conditions.

Module Revision: r_wifi_da16xxx rev1.20.
Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.06.00
(The option of "-lang=c99" is added to the default settings of the integrated development environment.)
GCC for Renesas RX 8.3.0.202311
(The option of "-std=gnu99" is added to the default settings of the integrated development environment.)
Configuration Options: Default settings.

Table 2.4 Memory Sizes

Device	Category		Memory usage	
			Renesas Compiler	GCC
RX65N	TCP only	ROM	23245 bytes	48440 bytes
		RAM	12013 bytes	14206 bytes
	TLS only	ROM	24430 bytes	49532 bytes
		RAM	12011 bytes	14206 bytes
	MQTT only	ROM	24832 bytes	49652 bytes
		RAM	9344 bytes	11518 bytes
	HTTP only	ROM	22826 bytes	47488 bytes
		RAM	7909 bytes	10110 bytes
	All protocols	ROM	30348 bytes	55864 bytes
		RAM	18456 bytes	20606 bytes

2.9. Return Values

The error codes returned by the API functions are listed below. The enumerated types of the return values and API function declarations are contained in r_wifi_da16xxx_if.h.

```
typedef enum
{
    WIFI_SUCCESS                = 0,     // success
    WIFI_ERR_PARAMETER          = -1,    // invalid parameter
    WIFI_ERR_ALREADY_OPEN       = -2,    // already WIFI module opened
    WIFI_ERR_NOT_OPEN           = -3,    // WIFI module is not opened
    WIFI_ERR_SERIAL_OPEN        = -4,    // serial open failed
    WIFI_ERR_MODULE_COM         = -5,    // communicate with module failed
    WIFI_ERR_MODULE_TIMEOUT     = -6,    // communicate with module timed out
    WIFI_ERR_NOT_CONNECT        = -7,    // not connect to access point
    WIFI_ERR_SOCKET_NUM         = -8,    // no available sockets
    WIFI_ERR_SOCKET_CREATE      = -9,    // create socket failed
    WIFI_ERR_CHANGE_SOCKET      = -10,   // cannot change socket
    WIFI_ERR_SOCKET_CONNECT     = -11,   // cannot connect socket
    WIFI_ERR_BYTEQ_OPEN         = -12,   // cannot assigned BYTEQ
    WIFI_ERR_SOCKET_TIMEOUT     = -13,   // socket timeout
    WIFI_ERR_TAKE_MUTEX         = -14,   // cannot take mutex

    /* For MQTT */
    WIFI_ERR_MQTT_ALREADY_OPEN  = -15,   // already WIFI MQTT opened
    WIFI_ERR_MQTT_NOT_OPEN      = -16,   // WIFI MQTT module is not opened
    WIFI_ERR_MQTT_NOT_CONNECT   = -17,   // not connect to MQTT channel
    WIFI_ERR_MQTT_CONNECTED     = -18,   // already connected to MQTT channel
    WIFI_ERR_MQTT_INVALID_DATA  = -19,   // invalid data to send/receive
    WIFI_ERR_MQTT_OUT_OF_MEMORY = -20,   // out of memory for MQTT communication

    /* For HTTP */
    WIFI_ERR_HTTP_ALREADY_OPEN  = -21,   // already WIFI HTTP opened
    WIFI_ERR_HTTP_NOT_OPEN      = -22,   // WIFI HTTP module is not opened
    WIFI_ERR_HTTP_INVALID_DATA  = -23,   // invalid data to send/receive
} wifi_err_t;
/* Error event for user callback */
typedef enum
{
    WIFI_EVENT_WIFI_REBOOT = 0,       // reboot WIFI
    WIFI_EVENT_WIFI_DISCONNECT,       // disconnected WIFI
    WIFI_EVENT_SERIAL_OVF_ERR,        // serial : overflow error
    WIFI_EVENT_SERIAL_FLM_ERR,        // serial : flaming error
    WIFI_EVENT_SERIAL_RXQ_OVF_ERR,    // serial : receiving queue overflow
    WIFI_EVENT_RCV_TASK_RXB_OVF_ERR,  // receiving task : receive buffer
overflow
    WIFI_EVENT_SOCKET_CLOSED,         // socket is closed
    WIFI_EVENT_SOCKET_RXQ_OVF_ERR     // socket : receiving queue overflow
} wifi_err_event_enum_t;
```

2.10. Parameters

```
/* Security type */
typedef enum
{
    WIFI_SECURITY_OPEN = 0,           // Open - No Security
    WIFI_SECURITY_WEP,                // WEP Security
    WIFI_SECURITY_WPA,                // WPA Security
    WIFI_SECURITY_WPA2,               // WPA2 Security
    WIFI_SECURITY_WPA2_ENT,           // WPA2 enterprise Security
    WIFI_SECURITY_WPA3,               // WPA3 Security
    WIFI_SECURITY_UNDEFINED           // Unknown Security
} wifi_security_t;

/* Encryption type */
typedef enum
{
    WIFI_ENCRYPTION_TKIP = 0,         // TKIP
    WIFI_ENCRYPTION_AES,              // AES
    WIFI_ENCRYPTION_TKIP_AES,         // TKIP+AES
    WIFI_ENCRYPTION_UNDEFINED,        // Unknow Encryption
} wifi_encryption_t;

/* Socket type */
typedef enum
{
    WIFI_SOCKET_TYPE_TCP_SERVER = 0, // TCP server
    WIFI_SOCKET_TYPE_TCP_CLIENT,     // TCP client
    WIFI_SOCKET_TYPE_UDP,            // UDP
    WIFI_SOCKET_TYPE_TLS             // TLS client
} wifi_socket_type_t;

/* Certificate type */
typedef enum
{
    WIFI_TLS_TYPE_CA_CERT = 0,           // CA Certificate
    WIFI_TLS_TYPE_CLIENT_CERT,           // Client Certificate
    WIFI_TLS_TYPE_CLIENT_PRIVATE_KEY,    // Client Private Key
    WIFI_TLS_TYPE_UNDEFINED              // Unknown Encryption
} wifi_tls_key_type_t;

/* Query current socket status */
typedef enum
{
    WIFI_SOCKET_STATUS_CLOSED = 0,   // "CLOSED"
    WIFI_SOCKET_STATUS_SOCKET,       // "SOCKET"
    WIFI_SOCKET_STATUS_BOUND,        // "BOUND"
    WIFI_SOCKET_STATUS_LISTEN,       // "LISTEN"
    WIFI_SOCKET_STATUS_CONNECTED     // "CONNECTED"
} wifi_socket_status_t;

/* MQTT Quality-of-service (QoS) levels */
typedef enum
{
    WIFI_MQTT_QOS_0 = 0,     // Delivery at most once.
    WIFI_MQTT_QOS_1 = 1,     // Delivery at least once.
    WIFI_MQTT_QOS_2 = 2      // Delivery exactly once.
} wifi_mqtt_qos_t;

/* MQTT TLS Cipher Suites */
```

```
typedef enum
{
    WIFI_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA      = 0xC011, //
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA protocol.
    WIFI_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA      = 0xC014, //
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA protocol.
    WIFI_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256   = 0xC027, //
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 protocol.
    WIFI_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384   = 0xC028, //
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 protocol.
    WIFI_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256   = 0xC02F, //
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 protocol.
    WIFI_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384   = 0xC030, //
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 protocol.
    WIFI_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA    = 0xC009, //
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA protocol.
    WIFI_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA    = 0xC00A, //
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA protocol.
    WIFI_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 = 0xC023, //
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 protocol.
    WIFI_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 = 0xC024, //
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 protocol.
    WIFI_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 = 0xC02B, //
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 protocol.
    WIFI_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 = 0xC02C, //
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 protocol.
} wifi_tls_cipher_suites_t;

/* Enable/disable for SNTP */
typedef enum
{
    WIFI_SNTP_DISABLE = 0,
    WIFI_SNTP_ENABLE  = 1
} wifi_sntp_enable_t;

/* AP scan result */
typedef struct
{
    uint8_t             ssid[WIFI_CFG_MAX_SSID_LEN];    // SSID
    uint8_t             bssid[WIFI_CFG_MAX_BSSID_LEN];  // BSSID
    wifi_security_t     security;                       // security type
    wifi_encryption_t   encryption;                     // encryption type
    int8_t              rssi;                           // RSSI
    uint8_t             hidden;                         // Hidden channel
} wifi_scan_result_t;

/* IP configurations */
typedef struct
{
    uint8_t ipaddress[4];             // IP address
    uint8_t subnetmask[4];            // subnet mask
    uint8_t gateway[4];               // gateway
} wifi_ip_configuration_t;

/* MQTT SUBSCRIBE packet parameters */
typedef struct st_wifi_mqtt_sub_info
{
    wifi_mqtt_qos_t    qos;                  // Quality of Service for
subscription.
    const char       * p_topic_filter;       // Topic filter to subscribe to.
```

```
    uint16_t           topic_filter_length;  // Length of subscription topic
filter.
} wifi_mqtt_sub_info_t;

/* MQTT PUBLISH packet parameters */
typedef struct st_wifi_mqtt_pub_info
{
    wifi_mqtt_qos_t    qos;                // Quality of Service for
subscription.
    const char       * p_topic_name;       // Topic name on which the message is
published.
    uint16_t           topic_name_Length;  // Length of topic name.
    const char *       p_payload;          // Message payload.
    uint32_t           payload_length;     // Message payload length.
} wifi_mqtt_pub_info_t;

/* MQTT Packet info structure to be passed to user callback */
typedef struct st_wifi_mqtt_callback_args
{
    uint8_t    * p_data;               // Payload received from subscribed MQTT
topic.
    const char * p_topic;              // Topic to which the message payload
belongs to.
    uint32_t     data_length;          // Length of the MQTT payload.
    void const * p_context;            // Placeholder for user data.
} wifi_mqtt_callback_args_t;

/* TCP TLS certificate information */
typedef struct {
    uint8_t cert_ca[WIFI_CFG_TLS_CERT_MAX_NAME];
    uint8_t cert_name[WIFI_CFG_TLS_CERT_MAX_NAME];
} wifi_tls_cert_info_t;

/* HTTP methods */
typedef enum
{
    WIFI_HTTP_GET  = 0,     // GET method
    WIFI_HTTP_POST = 1,     // POST method
    WIFI_HTTP_PUT  = 2      // PUT method
} wifi_http_method_t;

/* HTTP TLS authentication */
typedef enum
{
    WIFI_HTTP_TLS_VERIFY_NONE     = 0,     // No needed verify client
certification
    WIFI_HTTP_TLS_VERIFY_OPTIONAL = 1,     // Request client certification but
not mandatory
    WIFI_HTTP_TLS_VERIFY_REQUIRED = 2      // Require client certification
} wifi_http_tls_auth_t;

/* HTTP request packet parameters */
typedef struct st_wifi_http_request
{
    const char          * http_endpoint;      // HTTP endpoint
    wifi_http_method_t    method;             // HTTP request method
    const char          * request_body;       // HTTP request header
    uint32_t              length;             // HTTP request length
} wifi_http_request_t;

typedef struct st_wifi_http_buffer
```

```
{
    char   * response_buffer;
    uint32_t resp_length;
} wifi_http_buffer_t;
```

2.11. Adding the FIT Module to Your Project

The FIT 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’s 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’s Guide: IAREW (R20AN0535)” for details.

2.12. “for”, “while” and “do while” Statements

In FIT module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

This FIT module does not have any WAIT_LOOP. But others might have. Please take care for this WAIT_LOOP.

2.13. RTOS Usage Requirement

The FIT module utilizes RTOS functionality.

2.14. Restriction

The FIT module is subject to the following restrictions.
If WIFI_ERR_SERIAL_OPEN occurs, use R_WIFI_DA16XXX_Close() to close the Wi-Fi FIT module.

3. API Functions

3.1. R_WIFI_DA16XXX_Open()

This function initializes the FIT module and Wi-Fi module.

Format

```
wifi_err_t R_WIFI_DA16XXX_Open(
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_ALREADY_OPEN	Already open
WIFI_ERR_SERIAL_OPEN	Failed to initialize serial
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_BYTEQ_OPEN	BYTEQ allocation failure
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function initializes the FIT module and Wi-Fi module.

Reentrant

No

Example

```
R_WIFI_DA16XXX_Open();
```

Special Notes:

If WIFI_ERR_SERIAL_OPEN occurs, execute R_WIFI_DA16XXX_Close().

3.2. R_WIFI_DA16XXX_IsOpened()

This function checks Wi-Fi is opened.

Format

```
int32_t R_WIFI_DA16XXX_IsOpened(
      void
)
```

Parameters

None

Return values

0	Wi-Fi is opened
-1	Wi-Fi is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function checks Wi-Fi is opened.

Reentrant

No

Example

```
if (0 != R_WIFI_DA16XXX_IsOpened())
{
    return WIFI_SUCCESS;
}
```

Special Notes:

None

3.3. R_WIFI_DA16XXX_Close()

This function initializes the FIT module and Wi-Fi module.

Format

```
wifi_err_t R_WIFI_DA16XXX_Close(
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function closes the Wi-Fi module.

If this function is executed while the access point is connected, the access point will be disconnected, and the Wi-Fi module will be closed.

Reentrant

No

Example

```
R_WIFI_DA16XXX_Open();
R_WIFI_DA16XXX_Close();
```

Special Notes:

None

3.4. R_WIFI_DA16XXX_Ping()

This function pings the specified IP address.

Format

```
wifi_err_t R_WIFI_DA16XXX_Ping(
      uint8_t * ip_address,
      uint16_t count
)
```

Parameters

ip_address	IP address
count	Number of ping transmissions

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function pings the IP address specified by ip_address.

The parameter (count) specifies the number of transmissions.

Reentrant

No

Example

```
uint8_t ip_addr[4] = {192, 168, 5, 13};
R_WIFI_DA16XXX_Ping(ip_addr, 4);
```

Special Notes:

None

3.5. R_WIFI_DA16XXX_Scan()

This function scans for access points.

Format

```
wifi_err_t R_WIFI_DA16XXX_Scan(
      wifi_scan_result_t * ap_results,
      uint32_t max_networks
)
```

Parameters

ap_results	Pointer to the structure that stores the scan results
max_networks	Maximum number of access points to store in ap_results

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function scans for access points in the periphery of the Wi-Fi module.

The results of the scan are stored in the area specified by the ap_results argument, up to the maximum number of values specified by the max_networks argument.

Example

```
wifi_scan_result_t scan_rslt[5];
uint32_t max_networks = 5;
R_WIFI_DA16XXX_Scan(scan_rslt, max_networks);
for (int i = 0; i < 5; i++)
{
    printf(" --------------------\n");
    printf(" ssid : %s\n", scan_rslt[i].ssid);
    printf(" rssi : %d\n", scan_rslt[i].rssi);
    printf(" security : %d\n", scan_rslt[i].security);
    printf(" encryption : %d\n", scan_rslt[i].encryption);
}
```

Special Notes:

None

3.6. R_WIFI_DA16XXX_Connect()

This function connects to the specified access point.

Format

```
wifi_err_t R_WIFI_DA16XXX_Connect(
      const uint8_t * ssid,
      const uint8_t * pass,
      wifi_security_t security,
      wifi_encryption_t enc_type
)
```

Parameters

ssid	Pointer to SSID of access point
pass	Pointer to password of access point
security	Security type information
enc_type	Encryption type information

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

Connects to the access point specified by “ssid”.

Reentrant

No

Example

```
uint8_t ssid[] = “ssid”;
uint8_t pass[] = “passwd”;
wifi_security_t security = WIFI_SECURITY_WPA2;
wifi_encryption_t encryption = WIFI_ENCRYPTION_AES;

R_WIFI_DA16XXX_Open();
R_WIFI_DA16XXX_Connect(ssid, passwd, security, encryption);
```

Special Notes:

None

3.7. R_WIFI_DA16XXX_Disconnect()

This function disconnects the connecting access point.

Format

```
wifi_err_t R_WIFI_DA16XXX_Disconnect(
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function disconnects the connecting access point.

Reentrant

No

Example

```
uint8_t ssid[] = “ssid”;
uint8_t pass[] = “passwd”;
wifi_security_t security = WIFI_SECURITY_WPA2;
wifi_encryption_t encryption = WIFI_ENCRYPTION_AES;

R_WIFI_DA16XXX_Open();
R_WIFI_DA16XXX_Connect(ssid, passwd, security, encryption);
R_WIFI_DA16XXX_Disconnect();
```

Special Notes:

None

3.8. R_WIFI_DA16XXX_IsConnected()

This function obtains the connection status of the Wi-Fi module and access point.

Format

```
wifi_err_t R_WIFI_DA16XXX_IsConnected(
      void
)
```

Parameters

None

Return values

0	Connecting to the access point
-1	Not connected to access point

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

Returns the connection status of the Wi-Fi module and access point.

Reentrant

No

Example

```
if (0 == R_WIFI_DA16XXX_IsConnected())
{
    printf("connected \n");
}
else
{
    printf("not connect \n");
}
```

Special Notes:

None

3.9. R_WIFI_DA16XXX_DnsQuery()

This function performs a DNS query.

Format

```
wifi_err_t R_WIFI_DA16XXX_DnsQuery(
      uint8_t * domain_name,
      uint8_t * ip_address
)
```

Parameters

domain_name	Domain name
ip_address	IP address storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module or domain does not exist
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function performs a DNS query to obtain the IP address of the specified domain.

Reentrant

No

Example

```
Uint8_t ipaddr[4];
R_WIFI_DA16XXX_DnsQuery("hostname", ipaddr);
```

Special Notes:

None

3.10. R_WIFI_DA16XXX_SntpServerIpAddressSet()

This function sets SNTP server IP address.

Format

```
wifi_err_t R_WIFI_DA16XXX_SntpServerIpAddressSet(
      uint8_t * ip_address
)
```

Parameters

ip_address	IP address storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function sets SNTP server IP address.

Reentrant

No

Example

```
uint8_t ip_address_sntp_server[4] = {0, 0, 0, 0};
R_WIFI_DA16XXX_SntpServerIpAddressSet(ip_address_sntp_server);
```

Special Notes:

None

3.11. R_WIFI_DA16XXX_SntpEnableSet()

This function enables or disables SNTP client service.

Format

```
wifi_err_t R_WIFI_DA16XXX_SntpEnableSet(
      wifi_sntp_enable_t * enable
)
```

Parameters

ip_address	IP address storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function enables or disables SNTP client service.

Reentrant

No

Example

```
uint8_t ip_address_sntp_server[4] = {0, 0, 0, 0};
R_WIFI_DA16XXX_SntpServerIpAddressSet(ip_address_sntp_server);
R_WIFI_DA16XXX_SntpEnableSet(WIFI_SNTP_ENABLE);
```

Special Notes:

None

3.12. R_WIFI_DA16XXX_SntpTimeZoneSet()

This function sets SNTP time zone.

Format

```
wifi_err_t R_WIFI_DA16XXX_SntpTimeZoneSet(
      int utc_offset_in_hour
)
```

Parameters

utc_offset_in_hour	Time zone in UTC offset in hours

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function sets SNTP time zone.

Reentrant

No

Example

```
uint8_t ip_address_sntp_server[4] = {0, 0, 0, 0};
R_WIFI_DA16XXX_SntpServerIpAddressSet(ip_address_sntp_server;
R_WIFI_DA16XXX_SntpEnableSet(WIFI_SNTP_ENABLE);
R_WIFI_DA16XXX_SntpTimeZoneSet(25200);  /* UTC+07:00 */
```

Special Notes:

None

3.13. R_WIFI_DA16XXX_LocalTimeGet()

This function gets the current local time based on current time zone in a string.

Format

```
wifi_err_t R_WIFI_DA16XXX_LocalTimeGet(
      uint8_t * local_time,
      uint32_t size_string
)
```

Parameters

local_time	Pointer to local time in string format
size_string	size of string. The size of this string needs to be at least 25 bytes

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets the current local time based on the current time zone in a string.

Example: YYYY-MM-DD,HOUR:MIN:SECS.

Reentrant

No

Example

```
uint8_t time[25];
R_WIFI_DA16XXX_LocalTimeGet(time, 25);
printf("It is %s\n", time);
```

Special Notes:

None

3.14. R_WIFI_DA16XXX_SetDnsServerAddress()

This function sets DNS Server Address.

Format

```
wifi_err_t R_WIFI_DA16XXX_SetDnsServerAddress(
      uint8_t * dns_address
)
```

Parameters

dns_address	Pointed to DNS address storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function sets DNS Server Address.

Reentrant

No

Example

```
uint8_t dns[4] = {0, 0, 0, 0};
R_WIFI_DA16XXX_SetDnsServerAddress(dns);
```

Special Notes:

None

3.15. R_WIFI_DA16XXX_GetMacAddress()

This function obtains the MAC address value of the Wi-Fi module.

Format

```
wifi_err_t R_WIFI_DA16XXX_GetMacAddress(
      uint8_t * mac_address
)
```

Parameters

mac_address	Pointer to storage area for MAC address (6 bytes)

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

Obtains the MAC address value of the Wi-Fi module. The MAC address is stored as binary data in mac_address.

Reentrant

No

Example

```
uint8_t mac[6];
R_WIFI_DA16XXX_Open();
R_WIFI_DA16XXX_GetMacAddress(mac);
printf("— MAC addr : %lx:%lx:%lx:%lx:%lx:%lx\r\n",
mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
```

Special Notes:

None

3.16. R_WIFI_DA16XXX_GetIpAddress()

This function obtains the IP address assigned to the Wi-Fi module.

Format

```
wifi_err_t R_WIFI_DA16XXX_GetIpAddress(
      wifi_ip_configuration_t * ip_config
)
```

Parameters

ip_config	Pointer to IP address storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function obtains the IP address, subnet mask and gateway assigned to the Wi-Fi module and stores them in ip_config.

Reentrant

No

Example

```
wifi_ip_configuration_t ip_cfg;
R_WIFI_DA16XXX_GetIpAddress(&ip_cfg);
```

Special Notes:

None

3.17. R_WIFI_DA16XXX_HardwareReset()

This function resets the Wi-Fi module.

Format

```
wifi_err_t R_WIFI_DA16XXX_HardwareReset (
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_SERIAL_OPEN	Failed to initialize serial
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_BYTEQ_OPEN	BYTEQ allocation failure
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex
WIFI_ERR_SOCKET_CREATE	Failed to create socket

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function resets the Wi-Fi module with the RESET pin.

Reentrant

No

Example

```
R_WIFI_DA16XXX_HardwareReset();
```

Special Notes:

None

3.18. R_WIFI_DA16XXX_GetVersion()

This fuction obtains version information for the FIT module.

Format

```
uint32_t R_WIFI_DA16XXX_GetVersion (
      void
)
```

Parameters

None

Return values

Upper 2 bytes:	Major version (decimal notation)
Lower 2 bytes:	Minor version (decimal notation)

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function returns the version number of the FIT module.

The upper 2 bytes indicate the major version and the lower 2 bytes indicate the minor version.

Reentrant

No

Example

```
uint32_t ver;
ver = R_WIFI_DA16XXX_GetVersion();
printf("Version V%d.%2d\n", ((ver >> 16) & 0x0000FFFF), (ver & 0x0000FFFF));
```

Special Notes:

None

3.19. R_WIFI_DA16XXX_GetAvailableSocket()

This function gets the next available socket ID.

Format

```
wifi_err_t R_WIFI_DA16XXX_GetAvailableSocket(
      uint32_t * socket_id
)
```

Parameters

socket_id	Pointer to socket id storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	No socket available for connection socket

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets the next available socket ID.

Reentrant

No

Example

```
uint32_t socket_no;
R_WIFI_DA16XXX_GetAvailableSocket(&socket_no);
```

Special Notes:

None

3.20. R_WIFI_DA16XXX_GetSocketStatus()

This function gets the socket status.

Format

```
wifi_err_t R_WIFI_DA16XXX_GetSocketStatus(
      uint32_t socket_number,
      wifi_socket_status_t * socket_status
)
```

Parameters

socket_number	Socket number
socket_status	Pointer to socket status storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_SOCKET_NUM	Socket number is invalid

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets socket status.

Reentrant

No

Example

```
  if (WIFI_SOCKET_STATUS_CLOSED == R_WIFI_DA16XXX_GetSocketStatus(socket_no,
&socket_status))
  {
      printf("Socket is available \n");
  }
  else
  {
      printf("Socket is not available \n");
  }
```

Special Notes:

None

3.21. R_WIFI_DA16XXX_CreateSocket()

This function creates a socket by specifying the socket type and IP type.

Format

```
wifi_err_t R_WIFI_DA16XXX_CreateSocket(
      uint32_t socket_number,
      wifi_socket_type_t type,
      uint8_t ip_version
)
```

Parameters

socket_number	Socket number
type	Socket type
ip_version	IP version

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_CREATE	Failed to create socket

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function creates a TCP socket by specifying the socket type (WIFI_SOCKET_TYPE_TCP_CLIENT) and IP type.

Reentrant

No

Example

```
int32_t socket_no;
wifi_socket_type_t type = WIFI_SOCKET_TYPE_TCP_CLIENT;
R_WIFI_DA16XXX_GetAvailableSocket(&socket_no);
Sock_tcp = R_WIFI_DA16XXX_CreateSocket(socket_no, type, 4);
```

Special Notes:

None

3.22. R_WIFI_DA16XXX_TcpConnect()

This function connects to a specific IP and Port using socket.

Format

```
wifi_err_t R_WIFI_DA16XXX_TcpConnect(
      uint32_t socket_number,
      uint8_t * ip_address,
      uint16_t port
)
```

Parameters

socket_number	Socket number
ip_address	Pointer to IP address of TCP server in byte array format
port	Port of TCP server

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Socket numbet is invalid
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function connects to a specific IP and Port using socket.

Reentrant

No

Example

```
int32_t socket_no;
uint8_t ip_addr[4] = {192, 168, 1, 10};
uint16_t port = 1234;
da16xxx_socket_type_t type = DA16XXX_SOCKET_TYPE_TCP_CLIENT;
R_WIFI_DA16XXX_GetAvailableSocket(&socket_no);
Sock_tcp = R_WIFI_DA16XXX_CreateSocket(socket_no, type, 4);
R_WIFI_DA16XXX_TcpConnect(socket_no, ip_addr, port);
```

Special Notes:

None

3.23. R_WIFI_DA16XXX_SendSocket()

This function transmits data using the specified socket.

Format

```
wifi_err_t R_WIFI_DA16XXX_SendSocket(
      uint32_t socket_number,
      uint8_t * data,
      uint32_t length,
      uint32_t timeout_ms
)
```

Parameters

socket_number	Socket number
data	Pointer to transmit data in byte array format
length	Number of bytes of data to be transmitted
timeout_ms	Transmission timeout duration (millisecond)

Return values

Number of sent data	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_MODULE_TIMEOUT	Communicate with module timed out
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Socket number is invalid
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function sends the data stored in the data from the specified socket the number of bytes specified by length.

Reentrant

No

Example

```
int32_t recv_num;
uint8_t buffer[50];
recv_num = R_WIFI_DA16XXX_SendSocket(sock, buffer, sizeof(buffer), 1000);
```

Special Notes:

None

3.24. R_WIFI_DA16XXX_ReceiveSocket()

This function receives data from the specified socket.

Format

```
wifi_err_t R_WIFI_DA16XXX_ReceiveSocket(
      uint32_t socket_number,
      uint8_t * data,
      uint32_t length,
      uint32_t timeout_ms
)
```

Parameters

socket_number	Socket number
data	Pointer to receive data storage area
length	Number of bytes of data to be received
timeout_ms	Transmission timeout duration (millisecond)

Return values

Number of received data	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Socket number is invalid

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function sends the data stored in the data from the specified socket the number of bytes specified by length.

Reentrant

No

Example

```
int32_t recv_num;
uint8_t buffer[50];
recv_num = R_WIFI_DA16XXX_ReceiveSocket(sock, buffer, sizeof(buffer), 1000);
```

Special Notes:

None

3.25. R_WIFI_DA16XXX_CloseSocket()

This function disconnects communication with the specified socket and deletes the socket.

Format

```
wifi_err_t R_WIFI_DA16XXX_CloseSocket(
      uint32_t socket_number
)
```

Parameters

socket_number	Socket number

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_MODULE_TIMEOUT	Communicate with module timed out
WIFI_ERR_SOCKET_NUM	Socket number is invalid

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function disconnects communication with the specified socket and deletes the socket.

Reentrant

No

Example

```
R_WIFI_DA16XXX_TcpConnect(sock, ipaddr, port);
R_WIFI_DA16XXX_CloseSocket(sock);
```

Special Notes:

None

3.26. R_WIFI_DA16XXX_TcpReconnect()

This function reconnects to the existing socket.

Format

```
wifi_err_t R_WIFI_DA16XXX_TcpReconnect(
      uint32_t socket_number
)
```

Parameters

socket_number	Socket number

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Socket number is invalid
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function reconnects to the existing socket.

If sock_number is UINT8_MAX, this function will reconnect all disconnected sockets.

Reentrant

No

Example

```
R_WIFI_DA16XXX_TcpReconnect(socket_no);
```

Special Notes:

None

3.27. R_WIFI_DA16XXX_GetAvailableTlsSocket()

This function gets the next available TLS socket ID.

Format

```
wifi_err_t R_WIFI_DA16XXX_GetAvailableTlsSocket(
      uint32_t * socket_id
)
```

Parameters

socket_id	Pointer to socket id storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	No socket available for connection socket

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets the next available TLS socket ID.

Reentrant

No

Example

```
uint32_t socket_no;
R_WIFI_DA16XXX_GetAvailableTlsSocket(&socket_no);
```

Special Notes:

None

3.28. R_WIFI_DA16XXX_GetTlsSocketStatus()

This function gets the TLS socket status.

Format

```
wifi_err_t R_WIFI_DA16XXX_GetTlsSocketStatus(
      uint32_t socket_number,
      wifi_socket_status_t * socket_status
)
```

Parameters

socket_number	Socket number
socket_status	Pointer to socket status storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_SOCKET_NUM	Socket number is invalid

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets TLS Client socket status.

Reentrant

No

Example

```
 if(WIFI_SOCKET_STATUS_CLOSED == R_WIFI_DA16XXX_GetTlsSocketStatus(socket_no,
&socket_status))
 {
     printf("Socket is available \n");
 }
 else
 {
     printf("Socket is not available \n");
 }
```

Special Notes:

None

3.29. R_WIFI_DA16XXX_CreateTlsSocket()

This function creates a TLS socket by specifying the socket type and IP type.

Format

```
wifi_err_t R_WIFI_DA16XXX_CreateSocket(
      uint32_t socket_number,
      wifi_socket_type_t type,
      uint8_t ip_version
)
```

Parameters

socket_number	Socket number
type	Socket type
ip_version	IP version

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_CREATE	Failed to create socket

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function creates a TLS socket by specifying the socket type (WIFI_SOCKET_TYPE_TLS) and IP type.

Reentrant

No

Example

```
int32_t socket_no;
wifi_socket_type_t type = WIFI_SOCKET_TYPE_TLS;
R_WIFI_DA16XXX_GetAvailableTlsSocket(&socket_no);
Sock_tcp = R_WIFI_DA16XXX_CreateTlsSocket(socket_no, type, 4);
```

Special Notes:

None

3.30. R_WIFI_DA16XXX_TlsConnect()

This function connects to a specific IP and Port using TLS socket.

Format

```
wifi_err_t R_WIFI_DA16XXX_TlsConnect(
      uint32_t socket_number,
      uint8_t * ip_address,
      uint16_t port
)
```

Parameters

socket_number	Socket number
ip_address	IP address of TLS server in byte array format
port	Port of TLS server

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Socket number is invalid
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function connects to a specific IP and Port using TLS socket.

Reentrant

No

Example

```
int32_t socket_no;
uint8_t ip_addr[4] = {192, 168, 1, 10};
uint16_t port = 1234;
da16xxx_socket_type_t type = DA16XXX_SOCKET_TYPE_TLS;
R_WIFI_DA16XXX_GetAvailableTlsSocket(&socket_no);
Sock_tcp = R_WIFI_DA16XXX_CreateTlsSocket(socket_no, type, 4);
R_WIFI_DA16XXX_TlsConnect(socket_no, ip_addr, port);
```

Special Notes:

None

3.31. R_WIFI_DA16XXX_SendTlsSocket()

This function transmits data using the specified socket.

Format

```
wifi_err_t R_WIFI_DA16XXX_SendTlsSocket(
      uint32_t socket_number,
      uint8_t * data,
      uint32_t length,
      uint32_t timeout_ms
)
```

Parameters

socket_number	Socket number
data	Pointer to transmit data in byte array format
length	Number of bytes of data to be transmitted
timeout_ms	Transmission timeout duration (millisecond)

Return values

Number of sent data	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_MODULE_TIMEOUT	Communicate with module timed out
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Socket number is invalid or disconnected
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function sends the data stored in the data from the specified socket the number of bytes specified by length.

Reentrant

No

Example

```
int32_t recv_num;
uint8_t buffer[50];
recv_num = R_WIFI_DA16XXX_SendTlsSocket(sock, buffer, sizeof(buffer), 1000);
```

Special Notes:

None

3.32. R_WIFI_DA16XXX_ReceiveTlsSocket()

This function receives data from the specified socket.

Format

```
wifi_err_t R_WIFI_DA16XXX_ReceiveTlsSocket(
      uint32_t socket_number,
      uint8_t * data,
      uint32_t length,
      uint32_t timeout_ms
)
```

Parameters

socket_number	Socket number
data	Pointer to receive data storage area
length	Number of bytes of data to be received
timeout_ms	Transmission timeout duration (millisecond)

Return values

Number of received data	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Socket number is invalid

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function sends the data stored in the data from the specified socket the number of bytes specified by length.

Reentrant

No

Example

```
  int32_t recv_num;
  uint8_t buffer[50];
  recv_num = R_WIFI_DA16XXX_ReceiveTlsSocket(sock, buffer, sizeof(buffer),
1000);
```

Special Notes:

None

3.33. R_WIFI_DA16XXX_CloseTlsSocket()

This function disconnects communication with the specified TLS socket and deletes the socket.

Format

```
wifi_err_t R_WIFI_DA16XXX_CloseTlsSocket(
      uint32_t  socket_number
)
```

Parameters

socket_number	Socket number

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_MODULE_TIMEOUT	Communicate with module timed out
WIFI_ERR_SOCKET_NUM	Socket number is invalid

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function disconnects communication with the specified socket and deletes the socket.

Reentrant

No

Example

```
R_WIFI_DA16XXX_TlsConnect(sock, ipaddr, port);
R_WIFI_DA16XXX_CloseTlsSocket(sock);
```

Special Notes:

None

3.34. R_WIFI_DA16XXX_TlsReconnect()

This function reconnects to the existing socket.

Format

```
wifi_err_t R_WIFI_DA16XXX_TcpReconnect(
      uint32_t  socket_number
)
```

Parameters

socket_number	Socket number

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Socket number is invalid
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function reconnects to the existing socket.

If sock_number is UINT8_MAX, this function will reconnect all disconnected sockets.

Reentrant

No

Example

```
R_WIFI_DA16XXX_TlsReconnect(socket_no);
```

Special Notes:

None

3.35. R_WIFI_DA16XXX_RegistServerCertificate()

This function registers server certificates on the Wi-Fi module.

Format

```
wifi_err_t R_WIFI_DA16XXX_RegistServerCertificate(
      uint8_t socket_num,
      wifi_tls_cert_info_t * cert_info,
      uint32_t trans_buf_size,
      uint32_t recv_buf_size
)
```

Parameters

socket_num	Socket number
cert_info	Pointer to certificate information storage area
trans_buf_size	Incoming buffer length for TLS socket
recv_buf_size	Outgoing buffer length for TLS socket

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function configures SSL connection for specifies socket number with below list of configurations:

- Set SSL CA Certificate.
- Set SSL Certificate.
- Set the Incoming buffer length.
- Set the Outgoing buffer length.

This function must be called before calling this function: R_WIFI_DA16XXX_TlsConnect().

Reentrant

No

Example

```
R_WIFI_DA16XXX_RegistServerCertificate(socketId, &cert_info, 8192, 8192);
```

Special Notes:

None

3.36. R_WIFI_DA16XXX_RequestTlsSocket()

This function allocates the created TLS socket for SSL connection.

Format

```
wifi_err_t R_WIFI_DA16XXX_RequestTlsSocket (
      uint8_t socket_number
)
```

Parameters

socket_number	Socket number

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_CREATE	Failed to create socket

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function allocates the created TLS socket for SSL connection:

R_WIFI_DA16XXX_CreateTlsSocket() must be called before calling this function.

Reentrant

No

Example

```
R_WIFI_DA16XXX_RequestTlsSocket(socketId);
```

Special Notes:

None

3.37. R_WIFI_DA16XXX_GetServerCertificate()

This function gets stored server certificates on the Wi-Fi module.

Format

```
wifi_err_t R_WIFI_DA16XXX_GetServerCertificate(
      wifi_tls_cert_info_t * cert_info
)
```

Parameters

cert_info	Pointer to certificate information storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function obtains certificate information stored in the Wi-Fi module and returns the certificate information in cert_info.

Reentrant

No

Example

```
R_WIFI_DA16XXX_GetServerCertificate(&cert_info);
```

Special Notes:

None

3.38. R_WIFI_DA16XXX_WriteCertificate()

This function stores certificates on the Wi-Fi module.

Format

```
wifi_err_t R_WIFI_DA16XXX_WriteCertificate(
      const uint8_t * name,
      wifi_tls_key_type_t type_key,
      const uint8_t * p_data,
      uint32_t len
)
```

Parameters

name	Name of the certificate
type_key	Certificate type
p_data	Pointer to certificate data stored area
len	Certificate data size

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function writes a certificate or secret key in the sflash memory of the Wi-Fi module.

For the certificate type, see da16xxx_tls_key_type_t in 2.10 Parameter.

Reentrant

No

Example

```
R_WIFI_DA16XXX_WriteCertificate(WIFI_CFG_TLS_CERT_CA_NAME,
                                WIFI_TLS_TYPE_CA_CERT,
                                DEVICE_CERTIFICATE_AUTHORITY_PEM,
                                strlen(DEVICE_CERTIFICATE_AUTHORITY_PEM));
```

Special Notes:

None

3.39. R_WIFI_DA16XXX_DeleteCertificate()

This function deletes certificates on the Wi-Fi module.

Format

```
wifi_err_t R_WIFI_DA16XXX_DeleteCertificate(
      wifi_tls_key_type_t type_key,
      wifi_tls_cert_info_t * cert_info
)
```

Parameters

type_key	Certificate type
cert_info	Pointer to certificate information storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function removes a certificate or secret key in the sflash memory of the Wi-Fi module.

For the certificate type, see wifi_tls_key_type_t in 2.10 Parameter.

Reentrant

No

Example

```
R_WIFI_DA16XXX_DeleteCertificate(WIFI_TLS_TYPE_CA_CERT, &cert_info);
```

Special Notes:

None

3.40. R_WIFI_DA16XXX_MqttOpen()

This function initializes DA16XXX MQTT Client module.

Format

```
wifi_err_t R_WIFI_DA16XXX_MqttOpen (
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid parameter
WIFI_ERR_NOT_CONNECT	Not connect to access point
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_MQTT_ALREADY_OPEN	Already WIFI MQTT opened
WIFI_ERR_MQTT_INVALID_DATA	Invalid data to send/receive
WIFI_ERR_MQTT_OUT_OF_MEMORY	Out of memory for MQTT communication

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

Initialize the DA16XXX on-chip MQTT Client service.

Reentrant

No

Example

```
R_WIFI_DA16XXX_MqttOpen();
```

Special Notes:

None

3.41. R_WIFI_DA16XXX_MqttDisconnect()

This function disconnects from the DA16XXX MQTT Client service.

Format

```
wifi_err_t R_WIFI_DA16XXX_MqttDisconnect (
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_MQTT_NOT_OPEN	Wi-Fi MQTT module is not opened
WIFI_ERR_MQTT_NOT_CONNECT	Not connect to MQTT channel

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function disconnects from the DA16XXX MQTT Client service.

Reentrant

No

Example

```
uint32_t timeout;

R_WIFI_DA16XXX_MqttOpen();
R_WIFI_DA16XXX_MqttConnect(timeout);
R_WIFI_DA16XXX_MqttDisconnect();
```

Special Notes:

None

3.42. R_WIFI_DA16XXX_MqttConnect()

This function configures and connects to the DA16XXX MQTT Client service.

Format

```
wifi_err_t R_WIFI_DA16XXX_MqttConnect (
      uint32_t  timeout_ms
)
```

Parameters

timeout_ms	Time out (ms)

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_MQTT_NOT_OPEN	Wi-Fi MQTT module is not opened
WIFI_ERR_MQTT_CONNECTED	Not connect to access point

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function configures and connects to the DA16XXX MQTT Client service.

Reentrant

No

Example

```
uint32_t timeout;

R_WIFI_DA16XXX_MqttOpen();
R_WIFI_DA16XXX_MqttConnect(timeout);
```

Special Notes:

None

3.43. R_WIFI_DA16XXX_MqttPublish()

This function publishes a message for a given MQTT topic.

Format

```
wifi_err_t R_WIFI_DA16XXX_MqttPublish (
      wifi_mqtt_pub_info_t * const  p_pub_info
)
```

Parameters

p_pub_info	MQTT publish package parameters

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid parameter
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_MQTT_NOT_CONNECT	Not connect to MQTT channel
WIFI_ERR_MQTT_INVALID_DATA	Invalid data to send/receive

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function publishes a message for a given MQTT topic.

For the MQTT publish package, see da16xxx_mqtt_pub_info_t in 2.10 Parameter.

Reentrant

No

Example

```
wifi_mqtt_pub_info_t * const p_pub_info;

R_WIFI_DA16XXX_MqttPublish(p_pub_info);
```

Special Notes:

None

3.44. R_WIFI_DA16XXX_MqttReceive()

This function receives data subscribed to DA16XXX MQTT Client service.

Format

```
wifi_err_t R_WIFI_DA16XXX_MqttReceive (
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_MQTT_INVALID_DATA	Invalid data to send/receive
WIFI_ERR_MQTT_NOT_CONNECT	Not connect to MQTT channel

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function receives data subscribed to DA16XXX MQTT Client service.

Reentrant

No

Example

```
R_WIFI_DA16XXX_MqttReceive();
```

Special Notes:

None

3.45. R_WIFI_DA16XXX_MqttSubscribe()

This function subscribes to DA16XXX MQTT topics.

Format

```
wifi_err_t R_WIFI_DA16XXX_MqttSubscribe (
      wifi_mqtt_sub_info_t * const        p_sub_info,
      size_t                              subscription_count
)
```

Parameters

p_sub_info	MQTT subscribe package parameters
subscription_count	Number of subscribe topic.

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid parameter
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_MQTT_NOT_OPEN	Wi-Fi MQTT module is not opened
WIFI_ERR_MQTT_INVALID_DATA	Invalid data to send/receive

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function subscribes to DA16XXX MQTT topics.

For the MQTT subscribe package, see da16xxx_mqtt_sub_info_t in 2.10 Parameter.

Reentrant

No

Example

```
wifi_mqtt_sub_info_t * const  p_sub_info;
size_t          subscription_count;

 R_WIFI_DA16XXX_MqttSubscribe(p_sub_info, subscription_count);
```

Special Notes:

None

3.46. R_WIFI_DA16XXX_MqttUnSubscribe()

This function unsubscribes from DA16XXX MQTT topics.

Format

```
wifi_err_t R_WIFI_DA16XXX_MqttUnSubscribe (
      wifi_mqtt_sub_info_t * const  p_sub_info
)
```

Parameters

p_sub_info	MQTT subscribe package parameters

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid parameter
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_MQTT_NOT_CONNECT	Not connect to MQTT channel
WIFI_ERR_MQTT_INVALID_DATA	Invalid data to send/receive

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function unsubscribes from DA16XXX MQTT topics.

For the MQTT subscribe package, see da16xxx_mqtt_sub_info_t in 2.10 Parameter.

Reentrant

No

Example

```
wifi_mqtt_sub_info_t * const  p_sub_info;

R_WIFI_DA16XXX_MqttUnSubscribe(p_sub_info);
```

Special Notes:

None

3.47. R_WIFI_DA16XXX_MqttClose()

This function closes the DA16XXX MQTT Client service.

Format

```
wifi_err_t R_WIFI_DA16XXX_MqttClose (
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_MODULE_COM	Cannot communicate WIFI module
WIFI_ERR_MQTT_NOT_OPEN	WIFI MQTT module is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function closes the DA16XXX MQTT Client service.

Reentrant

No

Example

```
R_WIFI_DA16XXX_MqttOpen();
R_WIFI_DA16XXX_MqttClose();
```

Special Notes:

None

3.48. R_WIFI_DA16XXX_HttpOpen()

This function initializes DA16XXX HTTP Client module.

Format

```
wifi_err_t R_WIFI_DA16XXX_HttpOpen (
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid parameter
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_NOT_CONNECT	Not connect to access point
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex
WIFI_ERR_HTTP_ALREADY_OPEN	Already WIFI HTTP opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

Initialize the DA16XXX on-chip HTTP Client service.

Reentrant

No

Example

```
R_WIFI_DA16XXX_HttpOpen();
```

Special Notes:

None

3.49. R_WIFI_DA16XXX_HttpClose()

This function closes the DA16XXX HTTP Client service.

Format

```
wifi_err_t R_WIFI_DA16XXX_HttpClose (
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_MODULE_COM	Cannot communicate WIFI module
WIFI_ERR_HTTP_NOT_OPEN	WIFI HTTP module is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function closes the DA16XXX HTTP Client service.

Reentrant

No

Example

```
R_WIFI_DA16XXX_HttpOpen();
R_WIFI_DA16XXX_HttpClose();
```

Special Notes:

None

3.50. R_WIFI_DA16XXX_HttpSend()

This function sends the HTTP request with the configured buffers.

Format

```
wifi_err_t R_WIFI_DA16XXX_HttpSend (
      wifi_http_request_t request,
      wifi_http_buffer_t *buffer
)
```

Parameters

request	Pointer to HTTP request control structure
buffer	Pointer to HTTP user buffer struct for request and response

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid parameter
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module
WIFI_ERR_NOT_CONNECT	Not connect to access point
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex
WIFI_ERR_HTTP_NOT_OPEN	WIFI HTTP module is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function sends the HTTP request with the configured buffers.

For the HTTP request and HTTP user buffer, see wifi_http_request_t and wifi_http_buffer_t in 2.10 Parameter.

Reentrant

No

Example

```
R_WIFI_DA16XXX_HttpSend(http_post_req, &resp_buffer);
```

Special Notes:

None

4. Callback Function

4.1. Wi-Fi callback function

This function notifies the user application of a Wi-Fi module the errors related to communication.

Format

```
void * callback(
      void * pevent
)
```

Parameters

pevent Pointer to error information area

Return Values

None

Properties

This function is implemented by the user.

Description

Enable this API with the following configuration. The function name does not have to be “callback”.

```
#define WIFI_CFG_USE_CALLBACK_FUNCTION         (1)

#if WIFI_CFG_USE_CALLBACK_FUNCTION == 1

#define WIFI_CFG_CALLBACK_FUNCTION_NAME        (wifi_callback)

#endif
```

Since the event is notified as a void pointer type, cast it to wifi_err_event_t type before referencing it.

```
void wifi_callback(void * p_args)
{
    wifi_err_event_t *pevent;
    pevent = (wifi_err_event_t *)p_args;

    switch(pevent->event)
    {
        case WIFI_EVENT_SERIAL_OVF_ERR:
            break;
        …
    }
}
```

Reentrant

No

The notification events are as follows.

• WIFI_EVENT_SERIAL_OVF_ERR
Reports that the SCI module has detected a receive overflow error.
• WIFI_EVENT_SERIAL_FLM_ERR
Reports that the SCI module has detected a receive framing error.
• WIFI_EVENT_SERIAL_RXQ_OVF_ERR
Reports that the SCI module has detected a receive queue (BYTEQ) overflow.
• WIFI_EVENT_RCV_TASK_RXB_OVF_ERR
Reports that the FIT module has detected the overflow of the AT command receive buffer.
• WIFI_EVENT_SOCKET_RXQ_OVF_ERR
Reports that the socket has detected a receive queue (BYTEQ) overflow.

Example

```
[r_wifi_da16xxx_config.h]
#define WIFI_CFG_USE_CALLBACK_FUNCTION  (1)
#define WIFI_CFG_CALLBACK_FUNCTION_NAME (wifi_callback)

[xxx.c]
void wifi_callback(void *p_args)
{
    wifi_err_event_t *pevent;
    pevent = (wifi_err_event_t *)p_args;

    switch(pevent->event)
    {
        case WIFI_EVENT_SERIAL_OVF_ERR:
            break;
        case WIFI_EVENT_SERIAL_FLM_ERR:
            break;
        case WIFI_EVENT_SERIAL_RXQ_OVF_ERR:
            break;
        case WIFI_EVENT_RCV_TASK_OVF_ERR:
            break;
        case WIFI_EVENT_SOCKET_RXQ_OVF_ERR:
            switch(pevent->socket_number)
            {
                case 0:
                    break;
                case 1:
                    break;
                case 2:
                    break;
                case 3:
                    break;
            }
            break;
        default:
            break;
    }
}
```

Special Notes:

Do not call any of the functions listed in section 3. API Functions from the callback function.

4.2. MQTT callback function

This function notifies the user application of a Wi-Fi module the errors related to communication.

Format

```
void (* p_mqtt_callback) (
      void * pevent
)
```

Parameters

pevent Pointer to callback information to handle

Return Values

None

Properties

This function is implemented by the user.

Description

Enable this API with the following configuration. The function name does not have to be “callback”.

```
#define WIFI_CFG_MQTT_P_CALLBACK        (1)

#if WIFI_CFG_MQTT_P_CALLBACK == 1

#define WIFI_CFG_MQTT_P_CALLBACK_FUNCTION_NAME    /* Call back function name */

#endif
```

Reentrant

No

Example

```
[r_wifi_da16xxx_config.h]
#define WIFI_CFG_MQTT_P_CALLBACK  (1)
#define WIFI_CFG_MQTT_P_CALLBACK_FUNCTION_NAME (mqtt_userCallback)

[xxx.c]
void mqtt_userCallback (void * pevent)
{
    wifi_mqtt_callback_args_t * p_args;
    p_args = (wifi_mqtt_callback_args_t *)pevent;

    /* Code to handle incoming data */
    wifi_mqtt_pub_info_t pubTopic;
    wifi_err_t xMQTTStatus = WIFI_ERR_PARAMETER;

    char * ptr = strstr(p_args->p_topic, "test/MQTT/senddata");
    if (ptr != NULL)
    {
        if (0 == strcmp((const char *)p_args->p_data, "closeMQTT"))
        {
            cb_flag = 1;
        }
    }
}
```

Special Notes:

The R_WIFI_DA16XXX_MqttReceive() API should be called to use this callback function.

5. Demo Projects

Demo projects include function main() that utilizes the FIT module and its dependent modules (e.g. r_bsp). This FIT module includes the following demo project.

5.1 FreeRTOS Wi-Fi DA16600 Demo Project

5.1.1 Prerequisites

- Hardware requirements:
  - CK-RX65N: Renesas CK-RX65N Cloud Kit (Product no.: RTK5CK65N0S04000BE).
  - DA16600: US159-DA16600MEVZ as Wi-Fi module (included in CK-RX65N kit)
- Software requirements:
  - IDE: e^2 studio 2024-1 or later.
  - Compiler: Renesas Electronics C/C++ Compiler for RX Family V3.06.00.
  - Socket Test (for TCP Client demo): http://sockettest.sourceforge.net/.

5.1.2 Import the Demo Project

Users can import the demo project by adding the demo to their e^2 studio workspace (see section 5.2) or by downloading the demo project (see section 5.3).

5.1.3 Hardware Setup

- Connect the Wi-Fi DA16600 Pmod module to the CK-RX65N PMOD1 connector.
- Set the jumper of J16 to “Debug”.

5.1.4 How to Run the Demo

a) Country code and GMT time zone settings
Use the RX Smart Configurator to configure the country code and GMT time zone.
Open the RX Smart Configurator as shown in the image below and set the 2 parameters.

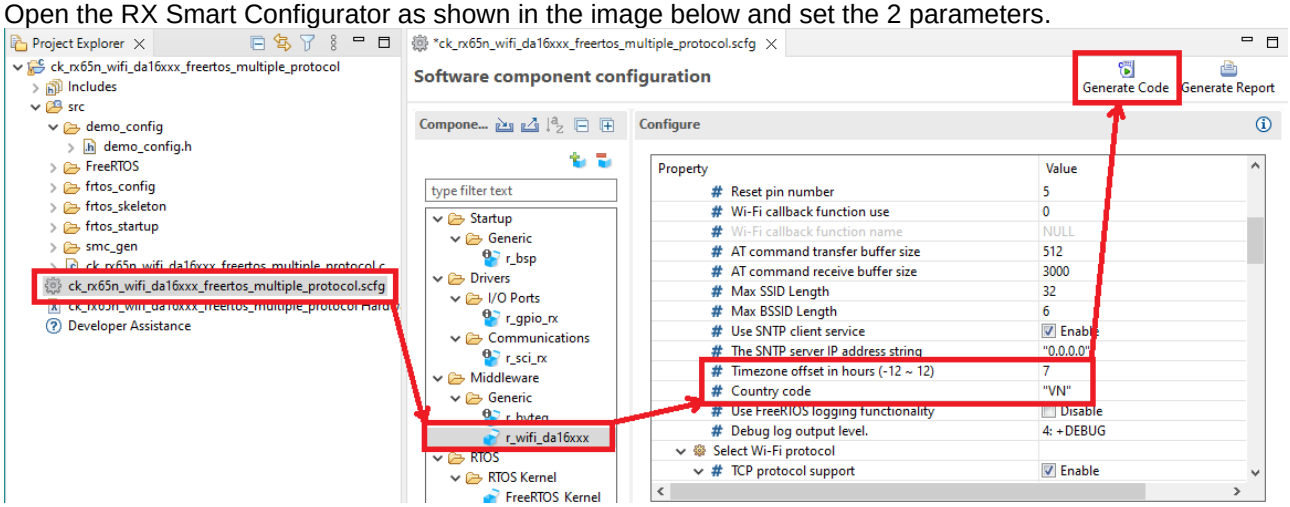

Figure 5.1 Country Code and GMT Time Zone Settings

- "WIFI_CFG_COUNTRY_CODE": Country code defined in ISO 3166-1 alpha-2 standard. Such as KR, US, JP, and CH.
- "WIFI_CFG_SNTP_UTC_OFFSET": GMT time zone offset in hours (-12 ~ 12).

b) Wi-Fi network settings
Configure Wi-Fi network settings for the Wi-Fi DA16600 module. Configure the following macro in "src/demo_config/demo_config.h"

```
/*
 * @brief Wi-Fi network to join.
 *
 * @todo If you are using Wi-Fi, set this to your network name.
 */
#define AP_WIFI_SSID                    "ssid"
/*
 * @brief Password needed to join Wi-Fi network.
 * @todo If you are using WPA, set this to your network password.
 */
#define AP_WIFI_PASSWORD                "password"
/*
 * @brief Wi-Fi network security type.
 *
 * @see WIFISecurity_t.
 *
 * @note Possible values are WIFI_SECURITY_OPEN, WIFI_SECURITY_WPA,
 * WIFI_SECURITY_WPA2 (depending on the support of your device Wi-Fi radio).
 */
#define AP_WIFI_SECURITY                WIFI_SECURITY_WPA2
```

Figure 5.2 Wi-Fi Network Settings

- AP_WIFI_SSID: Set the access point name of the Wi-Fi network.
- AP_WIFI_PASSWORD: Set the Wi-Fi network password.
- AP_WIFI_SECURITY: Set the Wi-Fi network security type (WIFI_SECURITY_OPEN, WIFI_SECURITY_WPA, WIFI_SECURITY_WPA2).

c) TCP client demo settings

Use the RX Smart Configurator to configure TCP protocol support.

Open the RX Smart Configurator as shown in the image below and set parameters.

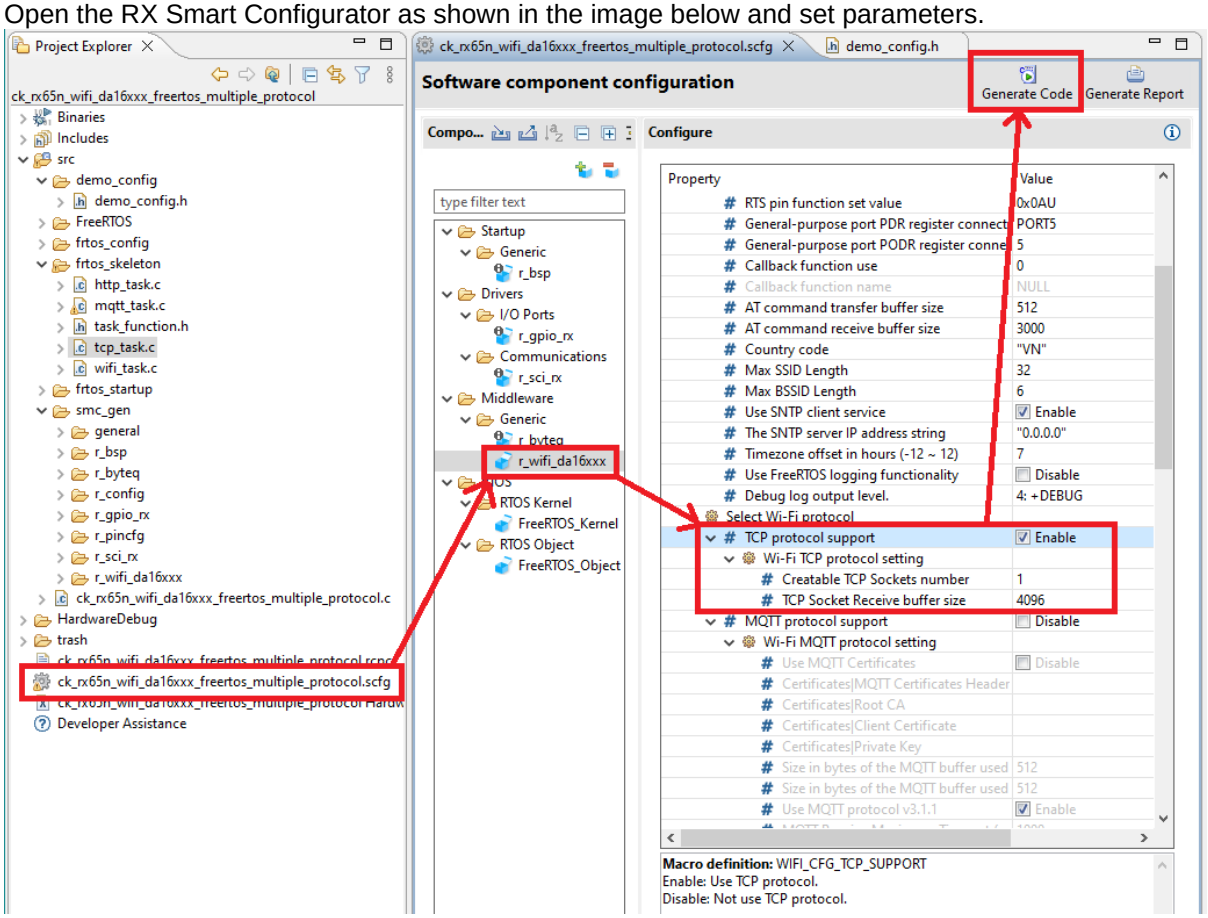

Figure 5.3 TCP Client Settings

- TCP protocol support: tick “Enable” to use the TCP demo or “Disable” to not use it.
- Creatable TCP Sockets number: This demo project only uses 1 socket number.
- Configures the TCP Receive buffer size: default is 4096.

Configure TCP server settings. Configure the following macro in “src/demo_config/demo_config.h”

```
/*
 * @brief TCP server host name.
 *
 * @note Set this to your TCP host name server.
 */
#define TCP_SERVER_HOSTNAME ..... "172.20.10.5"
/*
 * @brief TCP server port.
 *
 * @note Set this to your TCP port server.
 */
#define TCP_SERVER_PORT ..... 8883
```

Figure 5.4 TCP Server Settings

- TCP_SERVER_HOSTNAME: TCP server hostname or IP.
- TCP_SERVER_PORT: TCP server port.

d) MQTT on-chip client demo settings
Use the RX Smart Configurator to configure the MQTT protocol.
Open the RX Smart Configurator as shown in the image below and set the parameter.

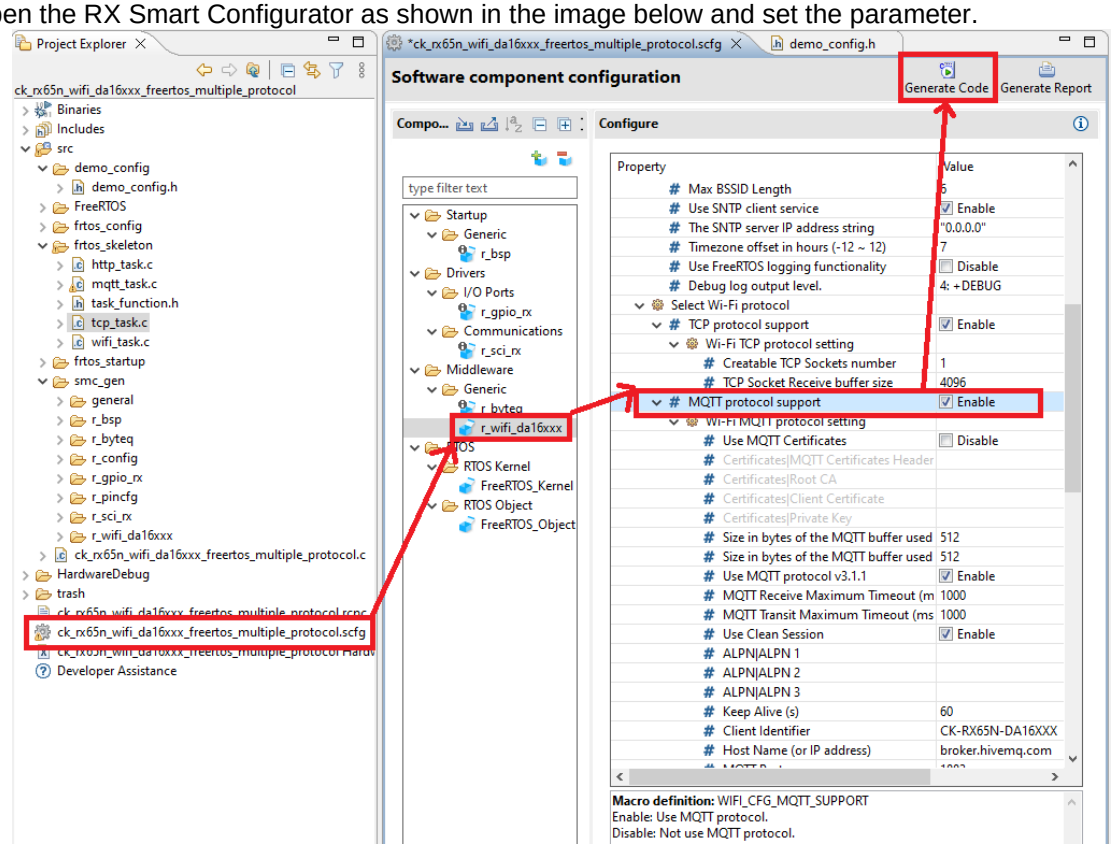

Figure 5.5 MQTT On-Chip Client Settings

- MQTT protocol support: tick “Enable” to use the MQTT on-chip client demo or “Disable” to not use it.

Configure the MQTT Publish/Subscribe topics. Configure the following macro in “src/demo_config/demo_config.h”

```
/*
 * @brief MQTT subscribe topic.
 *
 * @note Set subscribe topic for MQTT.
 */
#define MQTT_SUBSCRIBE_TOPIC            "test/MQTT/senddata"
/*
 * @brief MQTT publish topic.
 *
 * @note Set publish topic for MQTT.
 */
#define MQTT_PUBLISH_TOPIC              "test/MQTT/testdata"
```

Figure 5.6 MQTT Topics Settings

- MQTT_SUBSCRIBE_TOPIC: MQTT subscribe topic.
- MQTT_PUBLISH_TOPIC: MQTT publish topic.

e) HTTP on-chip client demo settings
Use the RX Smart Configurator to configure the HTTP protocol.
Open the RX Smart Configurator as shown in the image below and set the parameter.

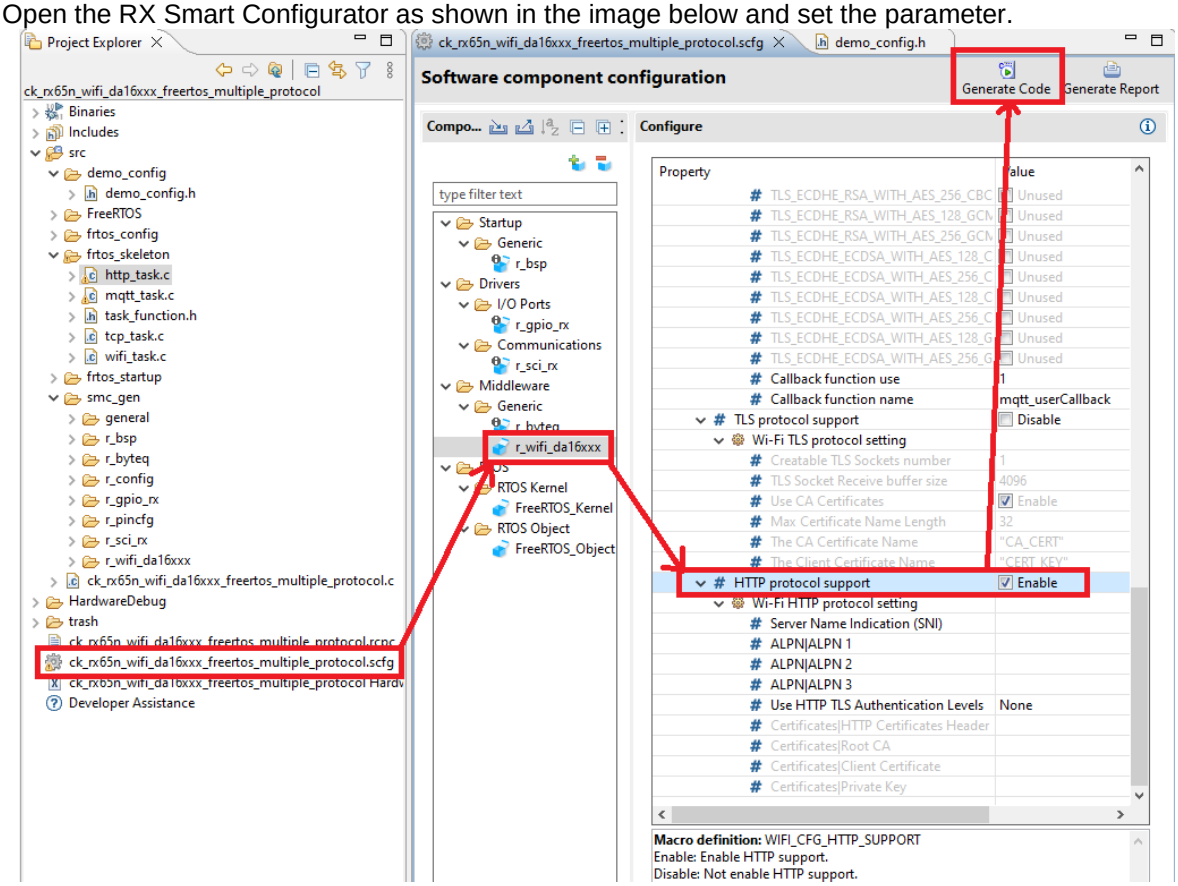

Figure 5.7 HTTP On-Chip Client Settings

- HTTP protocol support: tick “Enable” to use the HTTP on-chip client demo or “Disable” to not use it.

Configure HTTP server settings. Configure the following macro in “src/demo_config/demo_config.h”

```
/*
 * @brief HTTP server endpoint.
 *
 * @note Set this to your HTTP endpoint.
 */

#define HTTP_SERVER_ENDPOINT            "http://httpbin.org/get"
/*
 * @brief HTTP server method.
 *
 * @note Set this to your HTTP method (DA16XXX_HTTP_GET, DA16XXX_HTTP_POST, DA16XXX_HTTP_PUT).
 */
#define HTTP_SERVER_METHOD              DA16XXX_HTTP_GET
```

Figure 5.8 HTTP Server Settings

- HTTP_SERVER_ENDPOINT: Defines the URL to send HTTP requests to.
- HTTP_SERVER_METHOD: Request method to be used.

f) Building the demo project
Build the project and confirm no build errors occur.

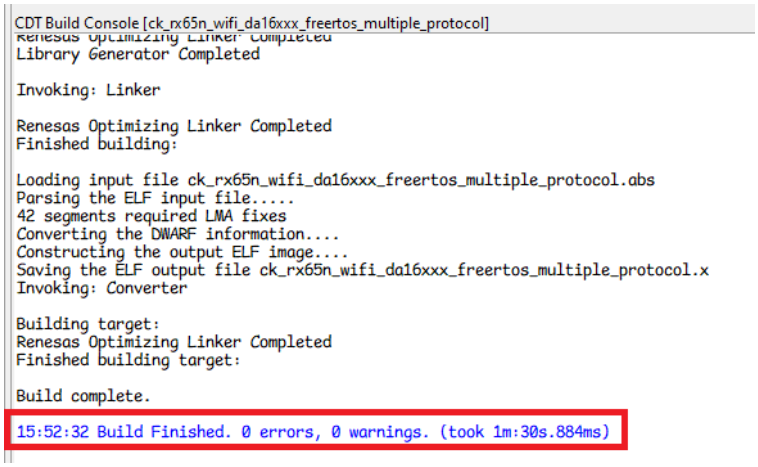

Figure 5.9 Confirm the Demo Project Build

In the **Project Explorer** panel of e^2 studio, right click on the project and select **Debug As** --> **Renesas GDB Hardware Debugging**.

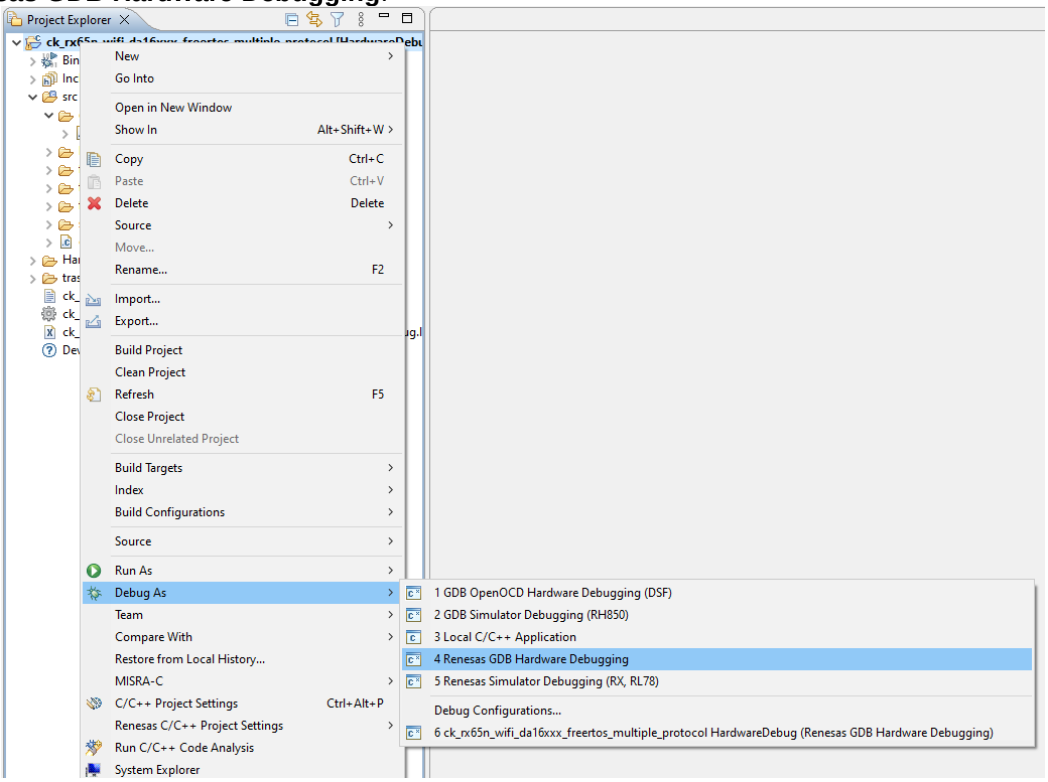

Figure 5.10 Flashing Demo Project

If the window below appears, press "Switch".

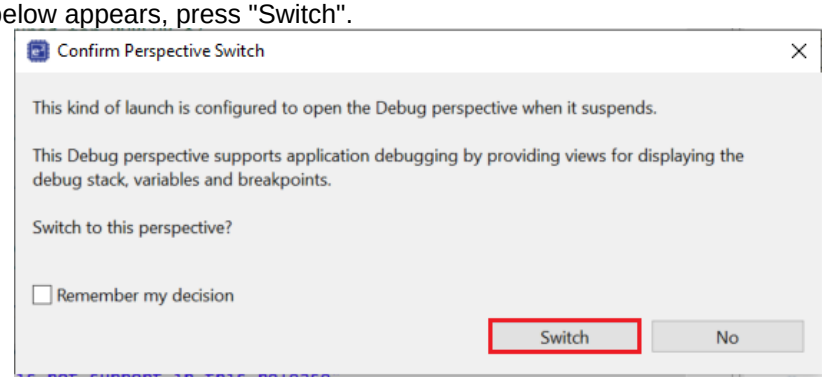

Figure 5.11 Confirm Perspective Switch

Press the following button to start debugging.

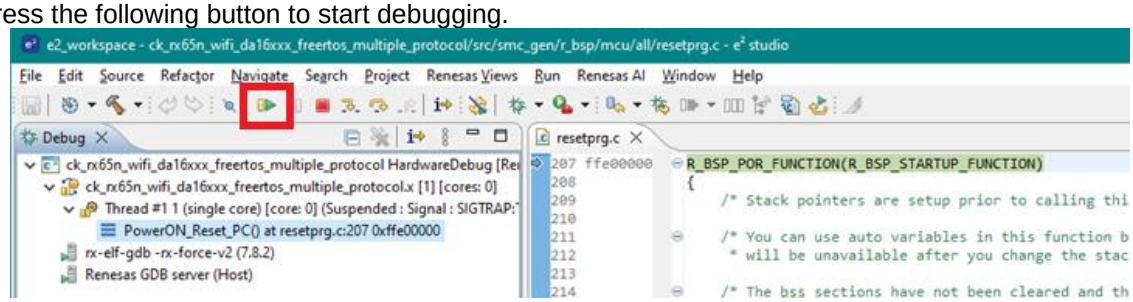

Figure 5.12 Start Debugging

g) Starting the demo using the TCP client
Run Socket Master: “SocketTest-master\dist\SocketTest.jar”
Note: Please start the TCP server before starting the debug.

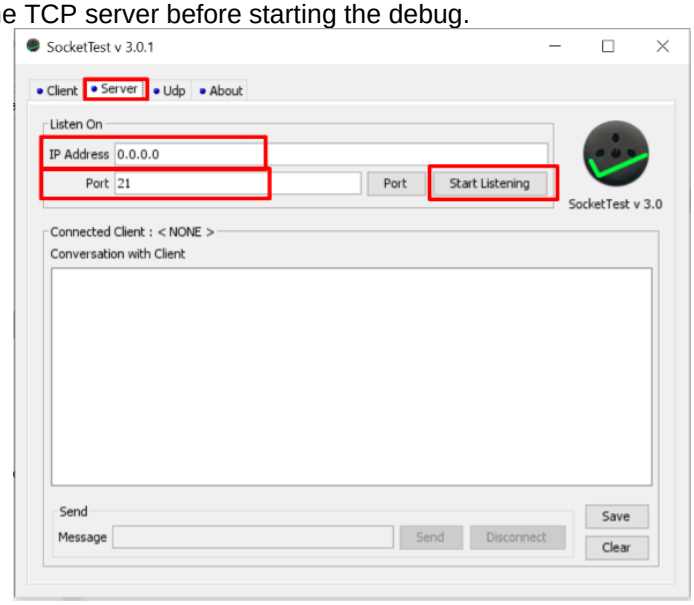

Figure 5.13 Start TCP Server

After connecting to the TCP socket, send a message and check if the sent data matches the received data in the message box.

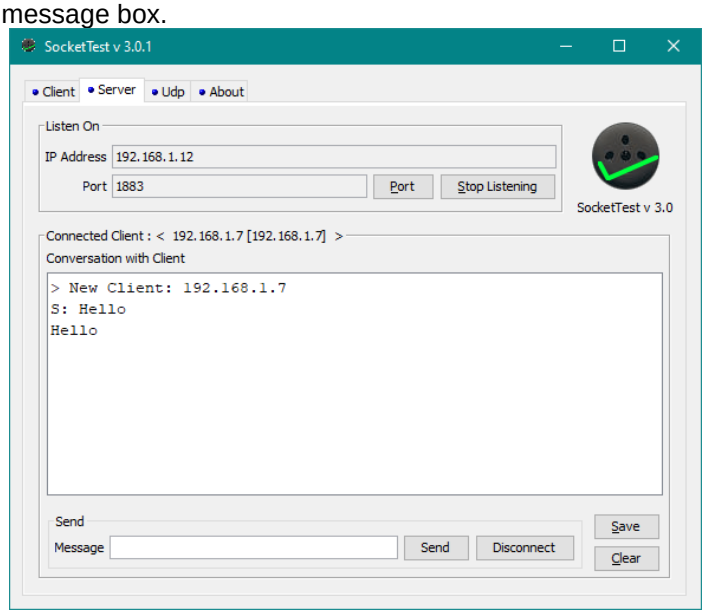

Figure 5.14 Demo with TCP Client

h) Starting the demo with the MQTT on-chip client
Open URL: https://testclient-cloud.mqtt.cool/ and select a Broker below.
Note: Please start the MQTT broker before starting the debug.

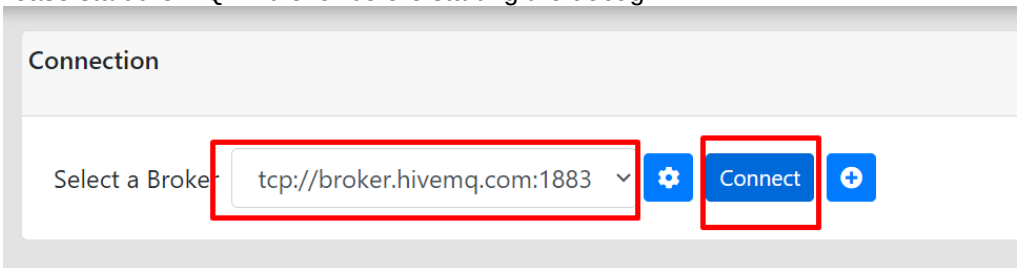

Figure 5.15 Start MQTT Broker

Enter the subscribe topic that was configured in demo_config.h.

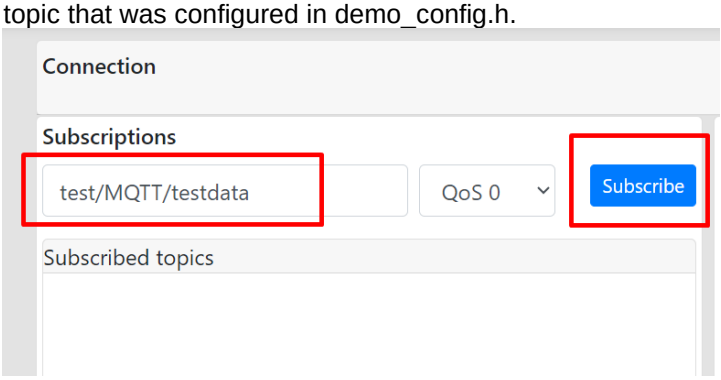

Figure 5.16 Subscribe Topic

In the messages box, confirm the receive data from topic "test/MQTT/testdata", and the publish data from topic "test/MQTT/senddata".

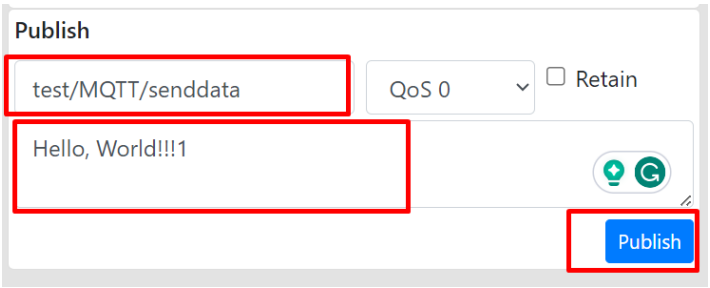

Figure 5.17 Publish Topic

i) Starting the demo with the HTTP on-chip client
Confirm debug log in **Renesas Debug Virtual Console**.

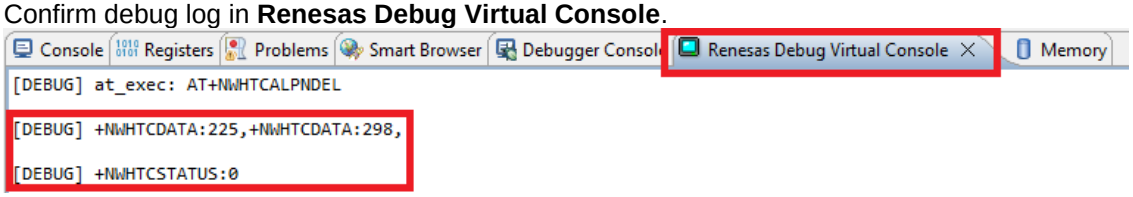

Figure 5.18 HTTP On-Chip Debug Log

5.2 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. To add a demo project to a workspace, select File >> Import >> General >> Existing Projects into Workspace, then click “Next”. From the Import Projects dialog, choose the “Select archive file” radio button. “Browse” to the FITDemos subdirectory, select the desired demo zip file, then click “Finish”.

5.3 Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on this application note and select “Sample Code (download)” from the context menu in the Smart Brower >> Application Notes tab.

6. Appendices

6.1 Confirmed Operation Environment

This section describes the confirmed operation environment for the FIT module.

Table 6.1 Confirmed Operation Environment (Ver. 1.00)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2022.04
C compiler	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
Endian order	Big endian / little endian
Revision of the module	Rev.1.00
Board used	Renesas CK-RX65N Cloud Kit (Product no.: RTK5CK65N0S04000BE)

Table 6.2 Confirmed Operation Environment (Ver. 1.10)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2023.04
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.05.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian order	Big endian / little endian
Revision of the module	Rev.1.10
Board used	Renesas CK-RX65N Cloud Kit (Product no.: RTK5CK65N0S04000BE)

Table 6.3 Confirmed Operation Environment (Ver. 1.20)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2024.01
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.06.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202311 Compiler option: The following option is added to the default settings of the integrated development environment. -std = gnu99
Endian order	Big endian / little endian
Revision of the module	Rev.1.20
Board used	Renesas CK-RX65N Cloud Kit (Product no.: RTK5CK65N0S04000BE)

6.2 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_wifi_da16xxx_config.h” may be wrong. Check the file "r_wifi_da16xxx_config.h". If there is a wrong setting, set the correct value for that. Refer to 2.7 Compile Settings for details.

7. Reference Documents

User's Manual: Hardware
(The latest versions can be downloaded from the Renesas Electronics website.)

Technical Update/Technical News
(The latest information can be downloaded from the Renesas Electronics website.)

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

Revision History

Rev.	Date	Revision History	
		Page	Summary
1.00	Mar. 10, 2023	-	First edition issued
1.10	Dec. 04, 2023	-	Rename DA16200 to DA16XXX
		9	Updated table 2-1 to add these configuration options below: • WIFI_CFG_CTS_SW_CTRL • WIFI_CFG_CTS_PORT • WIFI_CFG_CTS_PIN • WIFI_CFG_RTS_PORT • WIFI_CFG_RTS_PIN • WIFI_CFG_PFS_SET_VALUE • WIFI_CFG_USE_FREERTOS_LOGGING • WIFI_CFG_DEBUG_LOG
		40	Added table 5-2 Confirmed Operation Environment (Ver. 1.10)
1.20	Mar. 22, 2024	1	Added GCC for Renesas RX in Target Compilers
		5	Added Wi-Fi module features in section 1.2
		6	Updated Figure 1-1
		7-8	Added new APIs for TLS, MQTT on-chip, HTTP on-chip in Table 1-1
		10-12	Added Status transitions of TLS Client, MQTT on-chip, HTTP on-chip
		14-18	Added configuration option for TLS, MQTT, HTTP in table 2-1
		19	Updated Code Size for r_wifi_da16xxx rev.1.20
		20	Updated Return values
		21-23	Updated Parameters
		24	Added section 2.12. “for”, “while” and “do while” statements
		26	Added new API: R_WIFI_DA16XXX_IsOpened()
		41	Added new API: R_WIFI_DA16XXX_HardwareReset()
		42	Added new API: R_WIFI_DA16XXX_GetVersion()
		50	Added new API: R_WIFI_DA16XXX_TcpReconnect()
		51-63	Added new APIs for TLS socket
		64-71	Added new APIs for MQTT on-chip
		72-74	Added new APIs for HTTP on-chip
		77	Added callback function for MQTT on-chip
		78-86	Added Section 5. Demo Projects
		87	Added Table 5-3 Confirmed Operation Environment (Ver. 1.20)

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/.