RX Family

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-DA16600EVZ)

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

- RX140 Group
- RX261 Group
- RX65N Group
- RX66N Group
- RX671 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] RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
[2] RX Smart Configurator User's Guide: e² studio (R20AN0451)
[3] RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
[4] RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)
[5] FPB-RX140 v1 – User's Manual (R20UT5376)
[6] FPB-RX261 v1 – User's Manual (R20UT5363)
[7] EK-RX261 v1 – User's Manual (R20UT5351)
[8] CK-RX65N v1 – User's Manual (R20UT5100)
[9] CK-RX65N v2 – User's Manual (R20UT5366)
[10] RX66N Target Board – User's Manual (R20UT4895)
[11] EK-RX671 – User's Manual (R20UT5234)
[12] RX671 Target Board – User's Manual (R20UT4894)

Content

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 on-chip 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 DA16XXX on-chip OTA service for MCU firmware update.
- Supports 1 UART channel for interfacing with the DA16XXX module.
- Supports FreeRTOS-based user applications.
- Supports Bare metal-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.

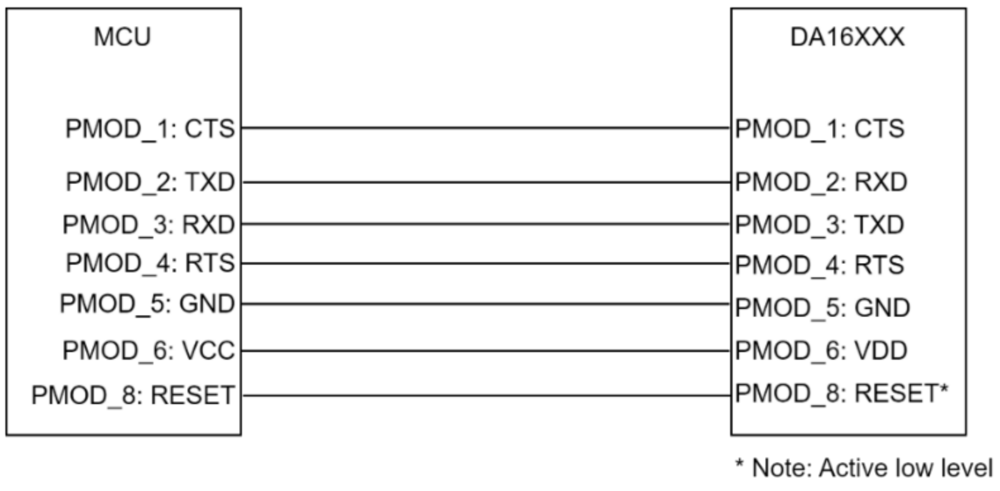


Figure 1.1 Example Connection to the DA16XXX Wi-Fi Module

1.2.2. Hardware Configuration

The hardware configuration for MCU host and the Wi-Fi Pmod module is shown below.

Table 1.1 MCU Host Hardware Configuration

Item	Content	Description
CK-RX65N v2 Cloud Kit	Target board for CK-RX65N v2 Part number: RTK5CK65N0S08001BE	Please see detail at: https://www.renesas.com/rx/ck-rx65n
EK-RX671 Evaluation Kit	Target board for EK-RX671 Part number: RTK5EK6710S00001BE	Please see detail at: https://www.renesas.com/rx/ek-rx671
RX671 Target Board *1	Target board for RX671 Part number: RTK5RX6710C00000BJ	Please see detail at: https://www.renesas.com/rtk5rx6710c00000bj
RX66N Target Board *1	Target board for RX66N Part number: RTK5RX66N0C00000BJ	Please see detail at: https://www.renesas.com/rtk5rx66n0c00000bj
RX140 Fast Prototyping Board	Target board for RX140 Fast Prototyping Board Part number: RTK5FP1400S00001BE	Please see detail at: https://www.renesas.com/rx/fpb-rx140
EK-RX261 Evaluation Kit *2	Target board for EK-RX261 Part number: RTK5EK2610S00001BE	Please see detail at: https://www.renesas.com/rx/ek-rx261
RX261 Fast Prototyping Board *2	Target board for RX261 Fast Prototyping Board Part number: RTK5FP2610S00001BE	Please see detail at: https://www.renesas.com/rx/fpb-rx261

Note 1: The PMOD on the RX66N and RX671 target boards is configured to Type 6A by default from the factory, which is not compatible with Wi-Fi module initialization. Therefore, changing the PMOD type to 2A or 3A is required. Please refer to **6.4 Limitations** for the instructions.

Note 2: The EK-RX261 Evaluation Kit and RX261 Fast Prototyping Board only support PMOD1 for the Wi-Fi Pmod module.

Table 1.2 Pmod Module Hardware Configuration

Item	Content	Description
DA16200 Wi-Fi Pmod module	Wi-Fi connection	This Pmod is used with MCU host for Wi-Fi connection. Please see detail at: https://www.renesas.com/us159-da16200mevz
DA16600 Wi-Fi Pmod module	Wi-Fi connection	This Pmod is used with MCU host for Wi-Fi connection. Please see detail at: US159-DA16600EVZ - Ultra-Low-Power Wi-Fi + Bluetooth Low Energy Combo Pmod Board

1.2.3. Software Configuration

Figure 1.2 shows the software configuration.

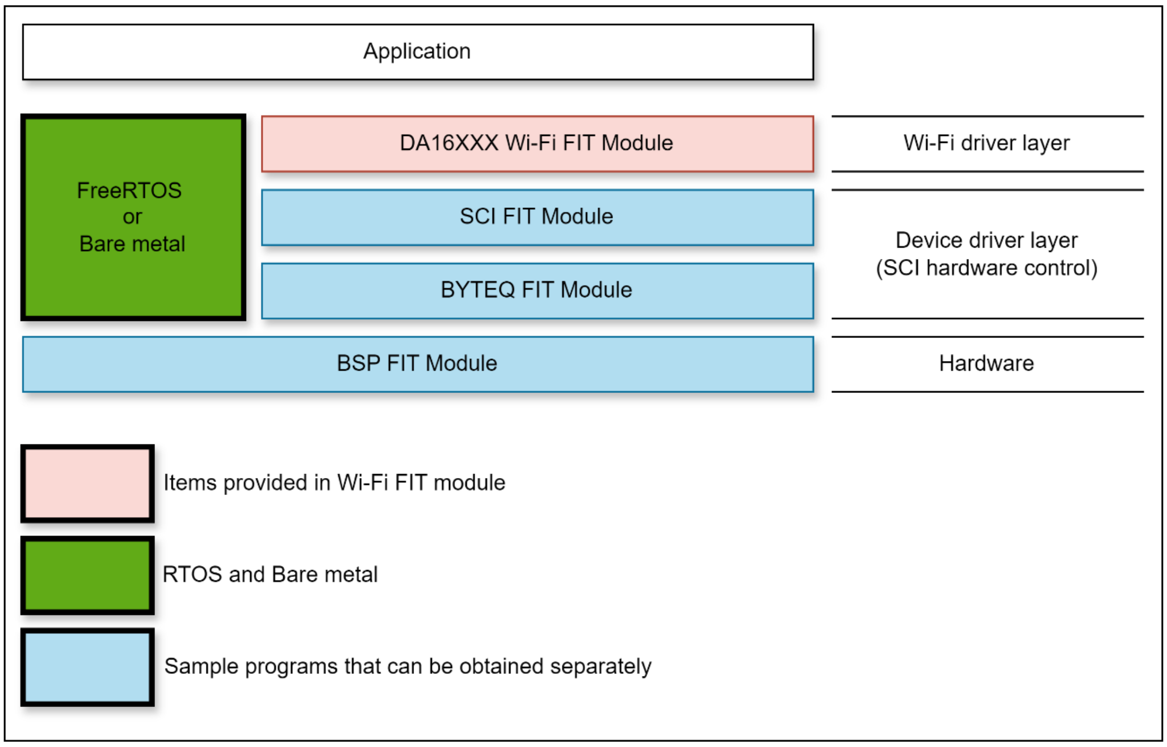


Figure 1.2 Software Configuration Diagram

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 or Bare metal.

1.3. API Overview

Table 1.3 lists the API functions included in the FIT module. The required memory sizes are lists in 2.8 Code Size.

Table 1.3 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 On-Chip 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_ConfigTlsSocket()	Configure SSL Connection on Wi-Fi module
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
Wi-Fi OTA On-chip Service API	
R_WIFI_DA16XXX_OtaOpen()	Initialize the DA16XXX on-chip OTA service
R_WIFI_DA16XXX_OtaClose()	Close the DA16XXX OTA service
R_WIFI_DA16XXX_OtaStart()	Start downloading firmware from an OTA server
R_WIFI_DA16XXX_OtaGetProgress()	Get progress status of firmware download.
R_WIFI_DA16XXX_OtaGetAddress()	Get the address where images is stored in DA16600/DA16200
R_WIFI_DA16XXX_OtaFirmwareSize()	Get a size in the header of the MCU firmware
R_WIFI_DA16XXX_OtaReadFirmwarebyBlock()	Read the MCU firmware as much as the size from the fw_addr and transmit it.
R_WIFI_DA16XXX_OtaEraseFirmware()	Erase the MCU firmware stored in a serial flash of the DA16200/DA16600
R_WIFI_DA16XXX_OtaGetState()	Get state of OTA transfer
R_WIFI_DA16XXX_OtaGetFirmware()	Get the data returned from the API OtaReadFirmwarbyBlock()

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.

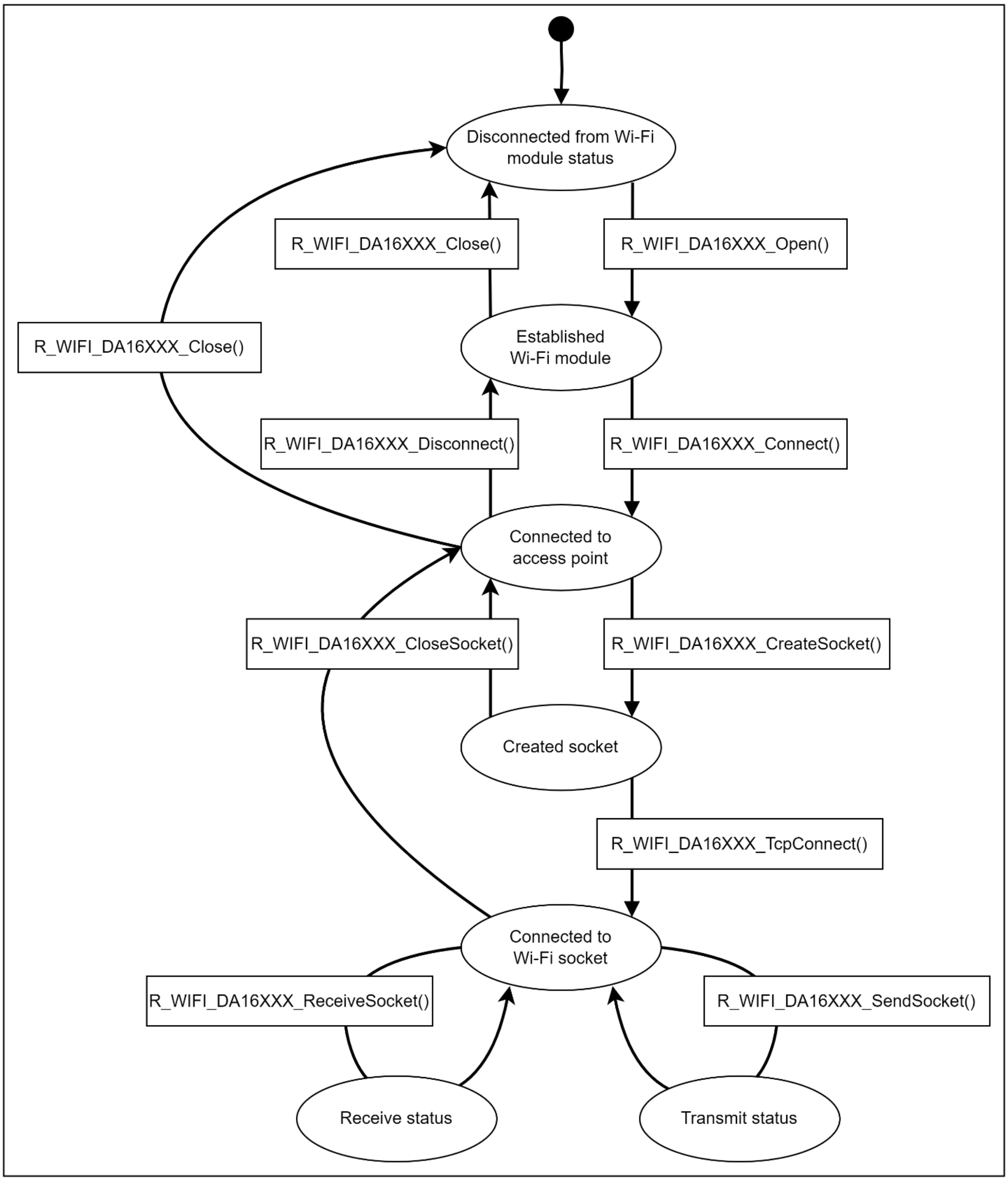

Disconnected from Wi-Fi module status
R_WIFI_DA16XXX_Close()
R_WIFI_DA16XXX_Open()
R_WIFI_DA16XXX_Close()
Established Wi-Fi module
R_WIFI_DA16XXX_Disconnect()
R_WIFI_DA16XXX_Connect()
Connected to access point
R_WIFI_DA16XXX_CloseSocket()
R_WIFI_DA16XXX_CreateSocket()
Created socket
R_WIFI_DA16XXX_TcpConnect()
Connected to Wi-Fi socket
R_WIFI_DA16XXX_ReceiveSocket()
R_WIFI_DA16XXX_SendSocket()
Receive status
Transmit status

Figure 1.3 Status Transitions When Using TCP Socket

1.4.2. Status Transitions of TLS On-Chip Client

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

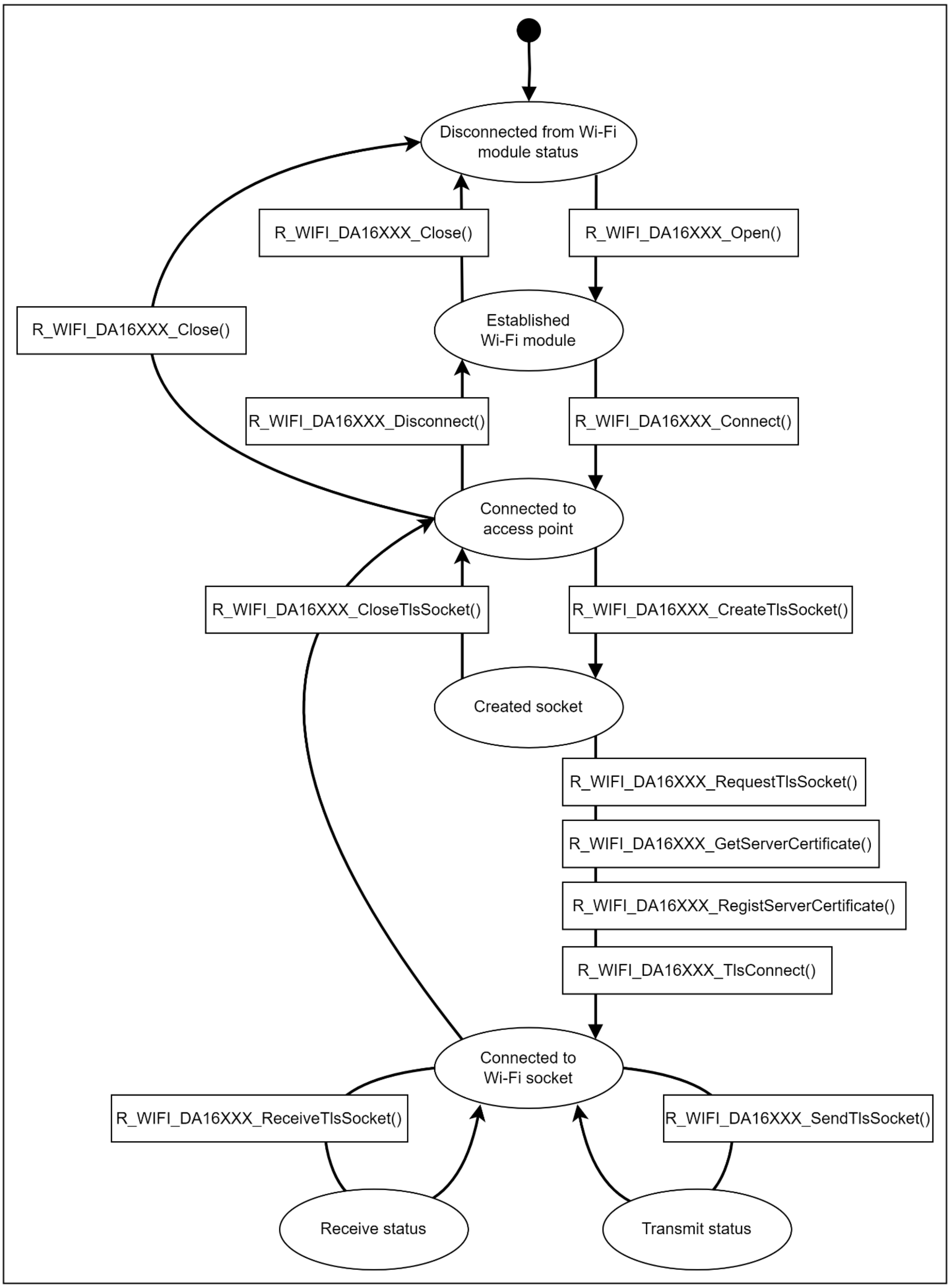

Disconnected from Wi-Fi module status
R_WIFI_DA16XXX_Close()
R_WIFI_DA16XXX_Open()
R_WIFI_DA16XXX_Close()
Established Wi-Fi module
R_WIFI_DA16XXX_Disconnect()
R_WIFI_DA16XXX_Connect()
Connected to access point
R_WIFI_DA16XXX_CloseTlsSocket()
R_WIFI_DA16XXX_CreateTlsSocket()
Created socket
R_WIFI_DA16XXX_RequestTlsSocket()
R_WIFI_DA16XXX_GetServerCertificate()
R_WIFI_DA16XXX_RegistServerCertificate()
R_WIFI_DA16XXX_TlsConnect()
Connected to Wi-Fi socket
R_WIFI_DA16XXX_ReceiveTlsSocket()
R_WIFI_DA16XXX_SendTlsSocket()
Receive status
Transmit status

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.

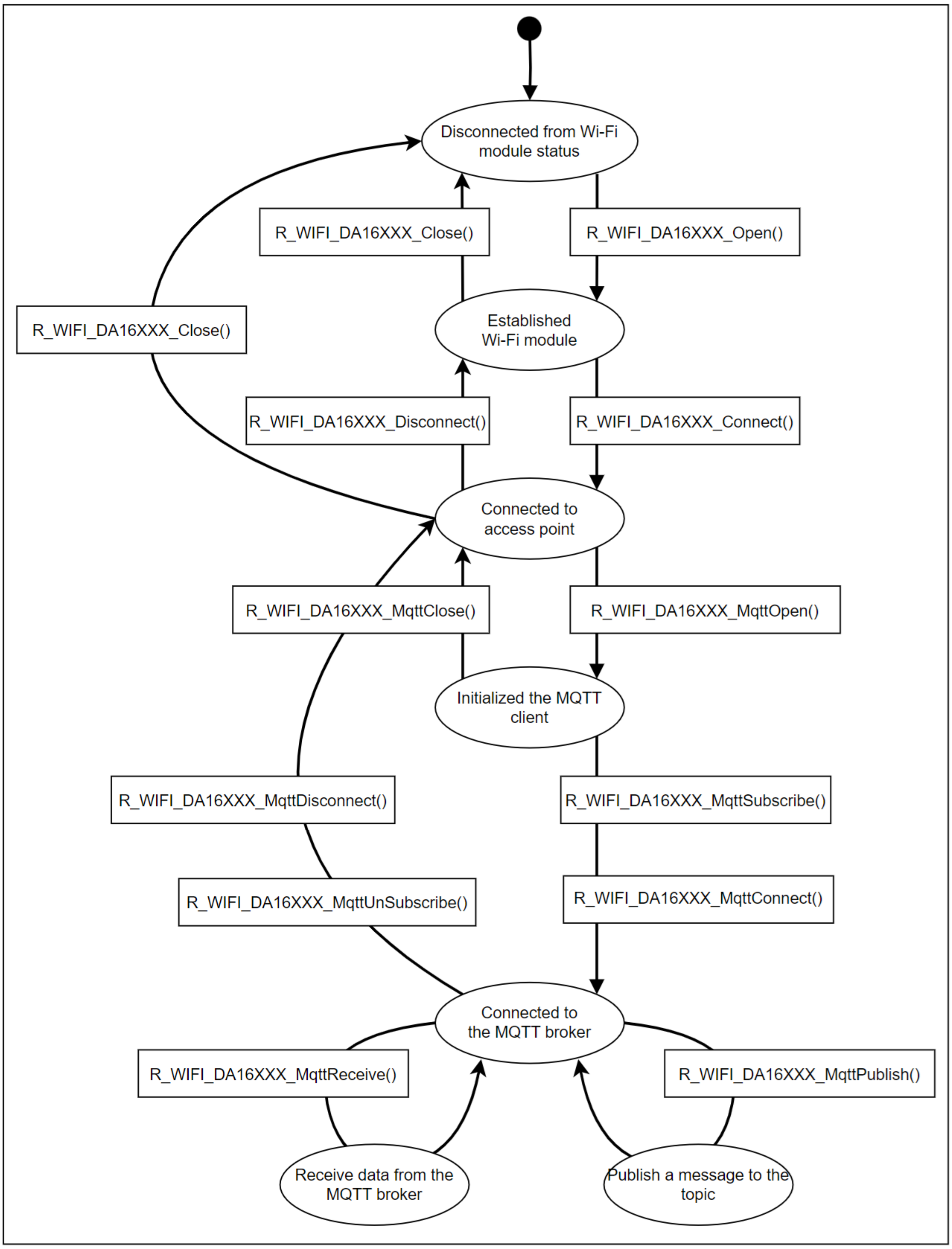

Disconnected from Wi-Fi module status
R_WIFI_DA16XXX_Close()
R_WIFI_DA16XXX_Open()
R_WIFI_DA16XXX_Close()
Established Wi-Fi module
R_WIFI_DA16XXX_Disconnect()
R_WIFI_DA16XXX_Connect()
Connected to access point
R_WIFI_DA16XXX_MqttClose()
R_WIFI_DA16XXX_MqttOpen()
Initialized the MQTT client
R_WIFI_DA16XXX_MqttDisconnect()
R_WIFI_DA16XXX_MqttSubscribe()
R_WIFI_DA16XXX_MqttUnSubscribe()
R_WIFI_DA16XXX_MqttConnect()
Connected to the MQTT broker
R_WIFI_DA16XXX_MqttReceive()
R_WIFI_DA16XXX_MqttPublish()
Receive data from the MQTT broker
Publish a message to the topic

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.

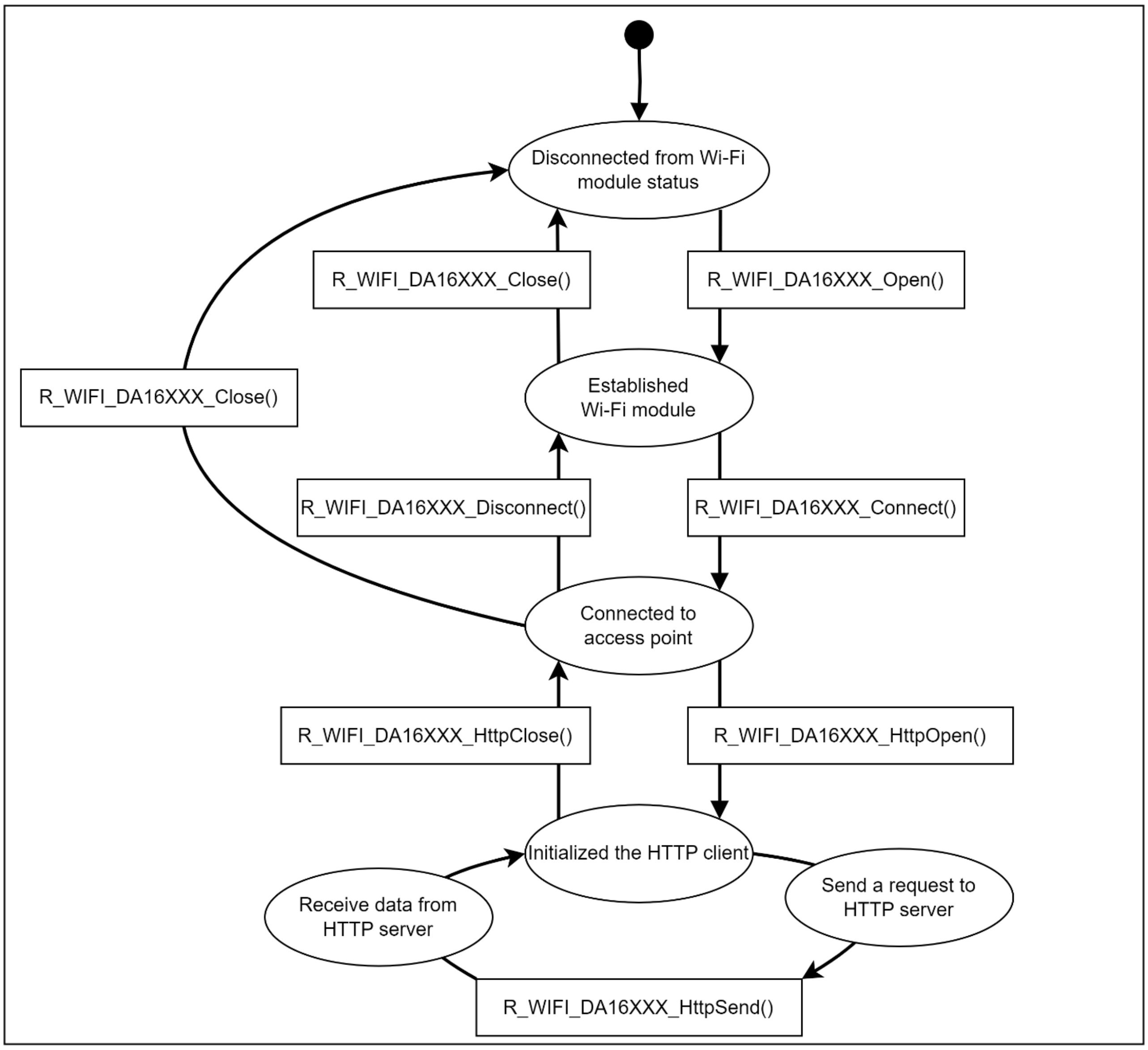

Disconnected from Wi-Fi module status
R_WIFI_DA16XXX_Close()
R_WIFI_DA16XXX_Open()
R_WIFI_DA16XXX_Close()
Established Wi-Fi module
R_WIFI_DA16XXX_Disconnect()
R_WIFI_DA16XXX_Connect()
Connected to access point
R_WIFI_DA16XXX_HttpClose()
R_WIFI_DA16XXX_HttpOpen()
Initialized the HTTP client
Receive data from HTTP server
Send a request to HTTP server
R_WIFI_DA16XXX_HttpSend()

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

1.4.5. Status Transitions of OTA On-Chip Service

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

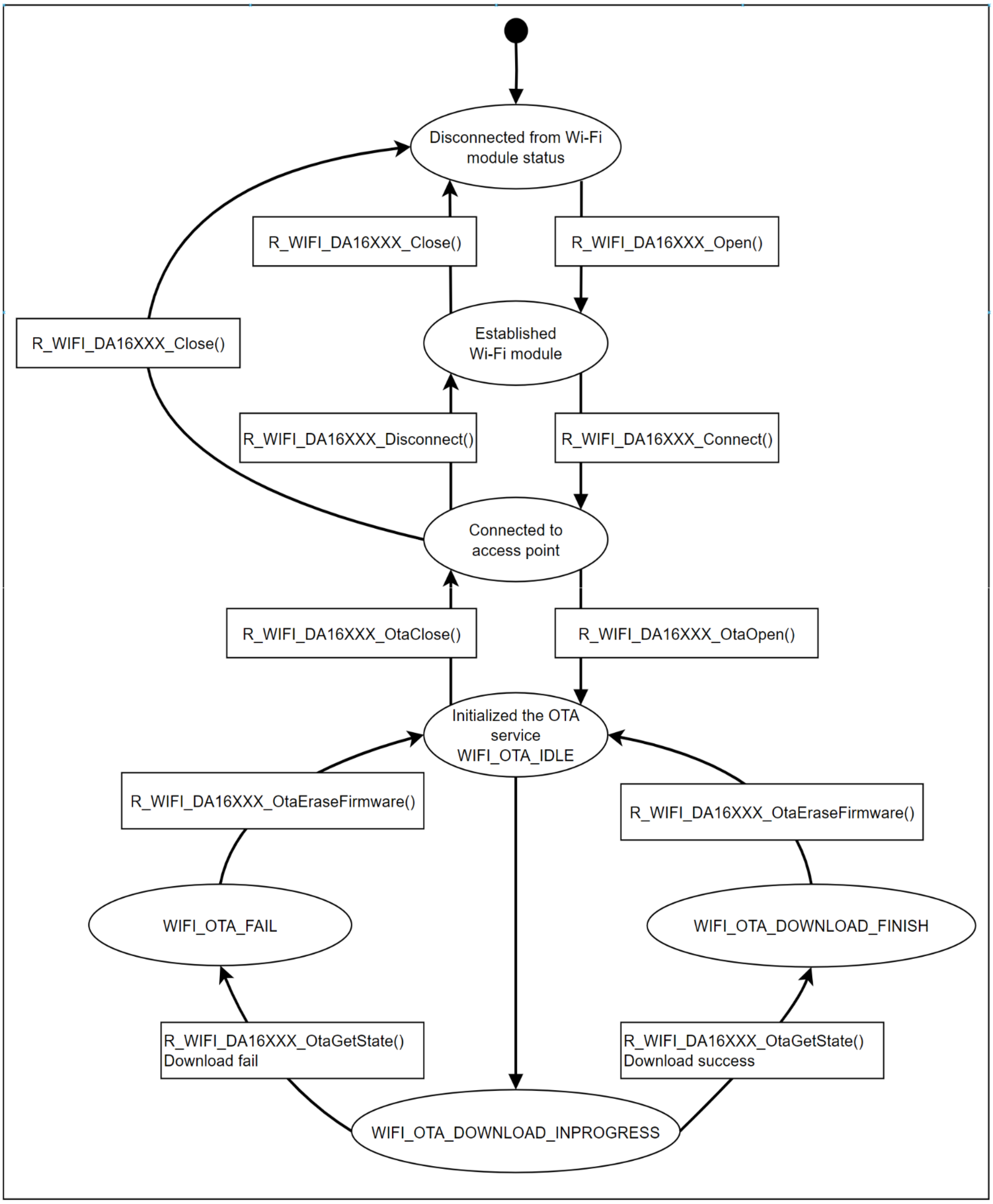

Disconnected from Wi-Fi module status
R_WIFI_DA16XXX_Close()
R_WIFI_DA16XXX_Open()
R_WIFI_DA16XXX_Close()
Established Wi-Fi module
R_WIFI_DA16XXX_Disconnect()
R_WIFI_DA16XXX_Connect()
Connected to access point
R_WIFI_DA16XXX_OtaClose()
R_WIFI_DA16XXX_OtaOpen()
Initialized the OTA service WIFI_OTA_IDLE
R_WIFI_DA16XXX_OtaEraseFirmware()
R_WIFI_DA16XXX_OtaEraseFirmware()
WIFI_OTA_FAIL
WIFI_OTA_DOWNLOAD_FINISH
R_WIFI_DA16XXX_OtaGetState() Download fail
R_WIFI_DA16XXX_OtaGetState() Download success
WIFI_OTA_DOWNLOAD_INPROGRESS

Figure 1.7 Status Transitions When Using the OTA On-Chip Service

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. This option takes effect when WIFI_CFG_USE_CALLBACK_FUNCTION is set to 1.
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. This option takes effect when WIFI_CFG_SNTP_ENABLE is set to 1.
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. E.g. "VN", "JP", "US".
WIFI_CFG_LOGGING_OPTION Default: "0"	Configures logging option. 0 = None, 1 = FreeRTOS logging, 2 = Serial port logging, 3 = Virtual console logging.
WIFI_CFG_LOG_TERM_CHANNEL Default: "5"	SCI Channel for DA16XXX logging function. Set this option to match the SCI port to be controlled. This option takes effect when WIFI_CFG_LOGGING_OPTION is set to 2.
WIFI_CFG_SCI_UART_TERMINAL_BAUDRATE Default: "115200"	Communication baud rate for serial port logging. This option takes effect when WIFI_CFG_LOGGING_OPTION is set to 2.
WIFI_CFG_SCI_UART_INTERRUPT_PRIORITY Default: "1"	Interrupt priority of serial port logging. Set this option to a value of 1 to 15 to match the system priority. This option takes effect when WIFI_CFG_LOGGING_OPTION is set to 2.
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 or Serial port logging. 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. This option takes effect when WIFI_CFG_LOGGING_OPTION is set to value other than 0.

Wi-Fi TCP On-Chip 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 On-Chip 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). Users must create a 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.
Wi-Fi OTA On-Chip Configuration	
WIFI_CFG_OTA_SUPPORT Default: “0”	Enables or disables OTA on-chip services. 1 = enabled, 0 = disabled.
WIFI_CFG_OTA_BLK_SIZE Default: “128”	Defines the size (in bytes) of each data block written to flash memory during the OTA (Over-the-Air) update process. Set this value in range from 128 to 1024
WIFI_CFG_OTA_TLS_AUTH Default: “0”	Configures OTA 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_OTA_CERTS_HEADER	Name of header file that will contain certificates (macros).

Default: “NULL”	Users must create a header file. Example: "cert_storage.h"
WIFI_CFG_OTA_ROOT_CA Default: “NULL”	Links to user-defined macro of the same name for Root CA 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 Bare metal:0

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 rev1.40.
Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.07.00
(The option of "-lang=c99" is added to the default settings of the integrated development environment.)
GCC for Renesas RX 14.2.0.202505
(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 for RX65N

Device	Protocol	Kernel	Category	Memory usage	
				Renesas Compiler	GCC
RX65N	TCP only (Default settings)	FreeRTOS	ROM	21796 bytes	46724 bytes
			RAM	11294 bytes	14768 bytes
		Bare metal	ROM	21591 bytes	46216 bytes
			RAM	11290 bytes	14768 bytes
	TLS on-chip only	FreeRTOS	ROM	23294 bytes	48628 bytes
			RAM	11292 bytes	14768 bytes
		Bare metal	ROM	23089 bytes	48120 bytes
			RAM	11288 bytes	14768 bytes
	MQTT on-chip only	FreeRTOS	ROM	23973 bytes	48100 bytes
			RAM	8872 bytes	12332 bytes
		Bare metal	ROM	23772 bytes	47608 bytes
			RAM	8868 bytes	12332 bytes
	HTTP on-chip only	FreeRTOS	ROM	21250 bytes	45428 bytes
			RAM	7177 bytes	10668 bytes
		Bare metal	ROM	21050 bytes	44936 bytes
			RAM	7173 bytes	10540 bytes
	All protocols	FreeRTOS	ROM	30047 bytes	57932 bytes
			RAM	17230 bytes	20656 bytes
		Bare metal	ROM	29836 bytes	57400 bytes
			RAM	17226 bytes	20656 bytes
	OTA on chip only	FreeRTOS	ROM	22994 bytes	48600 bytes
			RAM	11462 bytes	14896 bytes
		Bare metal	ROM	22789 bytes	48076 bytes
			RAM	11458 bytes	14896 bytes

Table 2.5 Memory Sizes for RX66N

Device	Protocol	Kernel	Category	Memory usage	
				Renesas Compiler	GCC
RX66N	TCP only (Default settings)	FreeRTOS	ROM	24586 bytes	46588 bytes
			RAM	11414 bytes	12928 bytes
		Bare metal	ROM	24381 bytes	46064 bytes
			RAM	11410 bytes	12928 bytes
	TLS on-chip only	FreeRTOS	ROM	26083 bytes	48492 bytes
			RAM	11412 bytes	12928 bytes
		Bare metal	ROM	25878 bytes	47968 bytes
			RAM	11408 bytes	12928 bytes
	MQTT on-chip only	FreeRTOS	ROM	26762 bytes	47956 bytes
			RAM	8992 bytes	10496 bytes
		Bare metal	ROM	26561 bytes	47464 bytes
			RAM	8988 bytes	10496 bytes
	HTTP on-chip only	FreeRTOS	ROM	24039 bytes	45292 bytes
			RAM	7297 bytes	8832 bytes
		Bare metal	ROM	23839 bytes	44784 bytes
			RAM	7293 bytes	8832 bytes
	All protocols	FreeRTOS	ROM	32837 bytes	57788 bytes
			RAM	17350 bytes	18944 bytes
		Bare metal	ROM	32626 bytes	57256 bytes
			RAM	17346 bytes	18944 bytes
	OTA on chip only	FreeRTOS	ROM	25784 bytes	48464 bytes
			RAM	11582 bytes	13184 bytes
		Bare metal	ROM	25579 bytes	47924 bytes
			RAM	11578 bytes	13184 bytes

Table 2.6 Memory Sizes for RX671

Device	Protocol	Kernel	Category	Memory usage	
				Renesas Compiler	GCC
RX671	TCP only (Default settings)	FreeRTOS	ROM	24586 bytes	46580 bytes
			RAM	11414 bytes	13056 bytes
		Bare metal	ROM	24381 bytes	46064 bytes
			RAM	11410 bytes	13056 bytes
	TLS on-chip only	FreeRTOS	ROM	26083 bytes	48484 bytes
			RAM	11412 bytes	13056 bytes
		Bare metal	ROM	25878 bytes	47968 bytes
			RAM	11408 bytes	13056 bytes
	MQTT on-chip only	FreeRTOS	ROM	26762 bytes	47948 bytes
			RAM	8992 bytes	10624 bytes
		Bare metal	ROM	26561 bytes	47464 bytes
			RAM	8988 bytes	10624 bytes
	HTTP on-chip only	FreeRTOS	ROM	24039 bytes	45284 bytes
			RAM	7297 bytes	8960 bytes
		Bare metal	ROM	23839 bytes	44784 bytes
			RAM	7293 bytes	8960 bytes
	All protocols	FreeRTOS	ROM	32837 bytes	57780 bytes
			RAM	17350 bytes	18944 bytes
		Bare metal	ROM	32626 bytes	57256 bytes
			RAM	17346 bytes	18944 bytes
	OTA on chip only	FreeRTOS	ROM	25784 bytes	48456 bytes
			RAM	11582 bytes	13184 bytes
		Bare metal	ROM	25579 bytes	47924 bytes
			RAM	11578 bytes	13184 bytes

Table 2.7 Memory Sizes for RX261

Device	Protocol	Kernel	Category	Memory usage	
				Renesas Compiler	GCC
RX261	TCP only (Default settings)	FreeRTOS	ROM	21772 bytes	46692 bytes
			RAM	11294 bytes	13056 bytes
		Bare metal	ROM	21567 bytes	46184 bytes
			RAM	11290 bytes	13056 bytes
	TLS on-chip only	FreeRTOS	ROM	23270 bytes	48596 bytes
			RAM	11292 bytes	13056 bytes
		Bare metal	ROM	23065 bytes	48088 bytes
			RAM	11288 bytes	13056 bytes
	MQTT on-chip only	FreeRTOS	ROM	23949 bytes	48068 bytes
			RAM	8872 bytes	10624 bytes
		Bare metal	ROM	23748 bytes	47576 bytes
			RAM	8868 bytes	10624 bytes
	HTTP on-chip only	FreeRTOS	ROM	21226 bytes	45396 bytes
			RAM	7177 bytes	8960 bytes
		Bare metal	ROM	21026 bytes	44904 bytes
			RAM	7173 bytes	8832 bytes
	All protocols	FreeRTOS	ROM	30023 bytes	57900 bytes
			RAM	17230 bytes	18944 bytes
		Bare metal	ROM	29812 bytes	57368 bytes
			RAM	17226 bytes	18944 bytes
	OTA on chip only	FreeRTOS	ROM	22970 bytes	48568 bytes
			RAM	11462 bytes	13184 bytes
		Bare metal	ROM	22765 bytes	48044 bytes
			RAM	11458 bytes	13184 bytes

Table 2.8 Memory Sizes for RX140

Device	Protocol	Kernel	Category	Memory usage	
				Renesas Compiler	GCC
RX140	TCP only (Default settings)	FreeRTOS	ROM	21772 bytes	46716 bytes
			RAM	11294 bytes	12928 bytes
		Bare metal	ROM	21567 bytes	46200 bytes
			RAM	11290 bytes	13056 bytes
	TLS on-chip only	FreeRTOS	ROM	23270 bytes	48620 bytes
			RAM	11292 bytes	12928 bytes
		Bare metal	ROM	23065 bytes	48104 bytes
			RAM	11288 bytes	13056 bytes
	MQTT on-chip only	FreeRTOS	ROM	23949 bytes	48092 bytes
			RAM	8872 bytes	10496 bytes
		Bare metal	ROM	23748 bytes	47600 bytes
			RAM	8868 bytes	10624 bytes
	HTTP on-chip only	FreeRTOS	ROM	21226 bytes	45420 bytes
			RAM	7177 bytes	8832 bytes
		Bare metal	ROM	21026 bytes	44920 bytes
			RAM	7173 bytes	8960 bytes
	All protocols	FreeRTOS	ROM	30023 bytes	57924 bytes
			RAM	17230 bytes	18944 bytes
		Bare metal	ROM	29812 bytes	57392 bytes
			RAM	17226 bytes	19072 bytes
	OTA on chip only	FreeRTOS	ROM	22970 bytes	48592 bytes
			RAM	11462 bytes	13184 bytes
		Bare metal	ROM	22765 bytes	48060 bytes
			RAM	11458 bytes	13184 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.

Table 2.9 API Error Codes (wifi_err_t)

Value	Error code	Description
0	WIFI_SUCCESS	OK, no error
-1	WIFI_ERR_PARAMETER	Invalid parameter
-2	WIFI_ERR_ALREADY_OPEN	Wi-Fi module already opens
-3	WIFI_ERR_NOT_OPEN	Wi-Fi module has not been opened
-4	WIFI_ERR_SERIAL_OPEN	Failed to open serial port
-5	WIFI_ERR_MODULE_COM	Failed communicating with Wi-Fi module
-6	WIFI_ERR_MODULE_TIMEOUT	Timed out communicating with Wi-Fi module
-7	WIFI_ERR_NOT_CONNECT	Not connected to AP
-8	WIFI_ERR_SOCKET_NUM	There are no available TCP/TLS sockets
-9	WIFI_ERR_SOCKET_CREATE	Failed creating TCP/TLS socket
-10	WIFI_ERR_CHANGE_SOCKET	Failed to change TCP/TLS socket number
-11	WIFI_ERR_SOCKET_CONNECT	Failed connecting a TCP/TLS socket
-12	WIFI_ERR_BYTEQ_OPEN	Failed to open BYTEQ module
-13	WIFI_ERR_SOCKET_TIMEOUT	TCP/TLS socket timeout
-14	WIFI_ERR_TAKE_MUTEX	Failed to take mutex
-15	WIFI_ERR_MQTT_ALREADY_OPEN	MQTT module already opens
-16	WIFI_ERR_MQTT_NOT_OPEN	MQTT module has not been opened
-17	WIFI_ERR_MQTT_NOT_CONNECT	Not connected to a MQTT broker
-18	WIFI_ERR_MQTT_CONNECTED	MQTT module is already connected
-19	WIFI_ERR_MQTT_INVALID_DATA	Invalid send/receive MQTT data
-20	WIFI_ERR_MQTT_OUT_OF_MEMORY	Out of memory for MQTT communication
-21	WIFI_ERR_HTTP_ALREADY_OPEN	HTTP module is already opened
-22	WIFI_ERR_HTTP_NOT_OPEN	HTTP module has not been opened
-23	WIFI_ERR_HTTP_INVALID_DATA	Invalid send/receive HTTP data
-24	WIFI_ERR_OTA_FAIL	OTA common error
-25	WIFI_ERR_OTA_NOT_OPEN	WIFI OTA service is not opened
-26	WIFI_ERR_OTA_ALREADY_OPEN	Already WIFI OTA opened
-27	WIFI_ERR_OTA_NOT_IDLE	OTA service state is not IDLE

Table 2.10 Error Event for User Callback (wifi_err_event_enum_t)

Value	Error code	Description
0	WIFI_EVENT_WIFI_REBOOT	Reboot Wi-Fi module
1	WIFI_EVENT_WIFI_DISCONNECT	Disconnected to Wi-Fi module
2	WIFI_EVENT_SERIAL_OVF_ERR	Serial overflow error
3	WIFI_EVENT_SERIAL_FLM_ERR	Serial flaming error
4	WIFI_EVENT_SERIAL_RXQ_OVF_ERR	Serial receive queue overflow
5	WIFI_EVENT_RCV_TASK_RXB_OVF_ERR	Received buffer overflow
6	WIFI_EVENT_SOCKET_CLOSED	Socket is closed
7	WIFI_EVENT_SOCKET_RXQ_OVF_ERR	Socket receive queue overflow

2.10. Parameters

This section describes the parameter structures used by the API functions in this module. The structures are defined in r_wifi_da16xxx_if.h.

Table 2.11 Definition of Security Type (wifi_security_t)

Value	Type	Description
0	WIFI_SECURITY_OPEN	Open – No security
1	WIFI_SECURITY_WEP	WEP security
2	WIFI_SECURITY_WPA	WPA security
3	WIFI_SECURITY_WPA2	WPA2 security
4	WIFI_SECURITY_WPA2_ENT	WPA2 enterprise security
5	WIFI_SECURITY_WPA3	WPA3 security
6	WIFI_SECURITY_UNDEFINED	Unknown security

Table 2.7 Definition of Encryption Type (wifi_encryption_t)

Value	Type	Description
0	WIFI_ENCRYPTION_TKIP	TKIP encryption
1	WIFI_ENCRYPTION_AES	AES encryption
2	WIFI_ENCRYPTION_TKIP_AES	TKIP+AES encryption
3	WIFI_ENCRYPTION_UNDEFINED	Unknown encryption

Table 2.8 Definition of Socket Type (wifi_socket_type_t)

Value	Type	Description
0	WIFI_SOCKET_TYPE_TCP_SERVER	TCP server
1	WIFI_SOCKET_TYPE_TCP_CLIENT	TCP client
2	WIFI_SOCKET_TYPE_UDP	UDP
3	WIFI_SOCKET_TYPE_TLS	TLS client

Table 2.9 Definition of Certificate Type (wifi_tls_key_type_t)

Value	Type	Description
0	WIFI_TLS_TYPE_CA_CERT	CA Certificate
1	WIFI_TLS_TYPE_CLIENT_CERT	Client Certificate
2	WIFI_TLS_TYPE_CLIENT_PRIVATE_KEY	Client Private Key
3	WIFI_TLS_TYPE_UNDEFINED	Unknown Encryption

Table 2.10 Definition of Socket Status (wifi_socket_status_t)

Value	Type	Description
0	WIFI_SOCKET_STATUS_CLOSED	Socket is closed
1	WIFI_SOCKET_STATUS_SOCKET	Socket is created
2	WIFI_SOCKET_STATUS_BOUND	Bounding
3	WIFI_SOCKET_STATUS_LISTEN	Listening socket
4	WIFI_SOCKET_STATUS_CONNECTED	Socket is connected

Table 2.11 MQTT Quality-of-Service (QoS) Levels (wifi_mqtt_qos_t)

Value	Type	Description
0	WIFI_MQTT_QOS_0	Delivery at most once
1	WIFI_MQTT_QOS_1	Delivery at least once
2	WIFI_MQTT_QOS_2	Delivery exactly once

Table 2.12 Cipher Suites Support for MQTT TLS (wifi_tls_cipher_suites_t)

Value	Type	Description
0xC011	WIFI_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
0xC014	WIFI_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
0xC027	WIFI_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
0xC028	WIFI_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
0xC02F	WIFI_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
0xC030	WIFI_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
0xC009	WIFI_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
0xC00A	WIFI_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
0xC023	WIFI_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
0xC024	WIFI_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
0xC02B	WIFI_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
0xC02C	WIFI_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

Table 2.13 SNTP Options (wifi_sntp_enable_t)

Value	Type	Description
0	WIFI_SNTP_DISABLE	Disable SNTP
1	WIFI_SNTP_ENABLE	Enable SNTP

Table 2.14 Member in Structure for Obtaining the Result of AP Scan (wifi_scan_result_t)

Type	Name	Description
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

Table 2.15 Member in Structure for IP Configurations (wifi_ip_configuration_t)

Type	Name	Description
uint32_t	ipaddress[4]	IP address
uint32_t	subnetmask[4]	Subnet mask
uint32_t	gateway[4]	Gateway

Table 2.20 Member in Structure for MQTT Subscription (wifi_mqtt_sub_info_t)

Type	Name	Description
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

Table 2.16 Member in Structure for MQTT Publish (wifi_mqtt_pub_info_t)

Type	Name	Description
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

Table 2.17 Member in Structure to be Passed to MQTT User Callback (wifi_mqtt_call_args_t)

Type	Name	Description
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

Table 2.18 Member in Structure for TLS Client on Chip Certificate Information (wifi_tls_cert_info_t)

Type	Name	Description
uint8_t	cert_ca[WIFI_CFG_TLS_CERT_MAX_NAME]	CA certificate name
uint8_t	cert_name[WIFI_CFG_TLS_CERT_MAX_NAME]	Client certificate name

Table 2.19 Definition of HTTP Methods (wifi_http_method_t)

Value	Type	Description
0	WIFI_HTTP_GET	GET method
1	WIFI_HTTP_POST	POST method
2	WIFI_HTTP_PUT	PUT method

Table 2.20 Definition of HTTP TLS Authentication (wifi_http_tls_auth_t)

Value	Type	Description
0	WIFI_HTTP_TLS_VERIFY_NONE	No needed verify client certification
1	WIFI_HTTP_TLS_VERIFY_OPTIONAL	Request client certification but not mandatory
2	WIFI_HTTP_TLS_VERIFY_REQUIRED	Require client certification

Table 2.21 Member in Structure for HTTP request (wifi_http_request_t)

Type	Name	Description
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

Table 2.22 Member in Structure for HTTP response (wifi_http_buffer_t)

Type	Name	Description
const char *	response_buffer	HTTP response buffer
uint32_t	resp_length	HTTP response length

Table 2.23 Member in Structure for OTA state (wifi_ota_state_t)

Type	Name	Description
0	WIFI_OTA_IDLE	OTA idle state
1	WIFI_OTA_DOWNLOAD_INPROGRESS	OTA download in progress state
2	WIFI_OTA_DOWNLOAD_FINISH	OTA download image finish state
3	WIFI_OTA_FAIL	OTA fail state

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. Limitations

2.13.1 Wi-Fi Security Type Limitations

Wi-Fi AP connections do not currently support WEP security.

2.13.2 Wi-Fi SDK Limitations

The default UART baud rate supported by v3.2.1 Wi-Fi SDK is 115200 and v3.2.4 Wi-Fi SDK is 230400. User needs to explicitly configure the default UART baud settings in the UART driver configurator properties based on the version of Wi-Fi SDK used in their testing.

2.13.3 The Daylight Savings Time Setting Limitations

In v3.2.1 Wi-Fi SDK, the daylight savings time setting is disabled by default. The user needs to mandatorily set the following parameters such as minutes = 0, daylight savings to disable when calling **R_WIFI_DA16XXX_SntpTimeZoneSet()** API.

2.13.4 Wi-Fi Network Connection Limitations

Network connection parameters SSID and Passphrase for the Access Point cannot contain any commas. This is a current limitation of the da16xxx module firmware. The **R_WIFI_DA16XXX_Connect()** function will return an error if a comma is detected.

2.13.5 Wi-Fi Access Point Scanning Limitations

Wi-Fi AP Scanning is currently limited to max of 10 Access Points.

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(
      uint32_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

```
uint32_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,
      uint8_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];
uint8_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,
      uint32_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

```
  uint32_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(
      uint32_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

```
uint32_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

`enable`	Enable/disable for SNTP

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(
      int8_t 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,
      uint8_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(
      uint32_t * mac_address
)
```

Parameters

mac_address	Pointer to storage area for MAC address

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

```
uint32_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 function 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(
      uint8_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

```
uint8_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(
      uint8_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(
      uint8_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

```
uint8_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(
      uint8_t socket_number,
      uint32_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

```
uint8_t socket_no;
uint32_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(
      uint8_t socket_number,
      uint8_t * data,
      uint16_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(
      uint8_t socket_number,
      uint8_t * data,
      uint16_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(
      uint8_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(
      uint8_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_CreateTlsSocket(
      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

```
uint32_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,
      uint32_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

```
uint32_t socket_no;
uint32_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,
      uint16_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,
      uint16_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_ TlsReconnect(
      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_ConfigTlsSocket()

This function configures SSL connection on Wi-Fi module.

Format

```
wifi_err_t R_WIFI_DA16XXX_ConfigTlsSocket(
      uint32_t * socket_num,
      wifi_tls_cert_info_t * cert_info,
      uint8_t WIFI_FAR * sni_name,
      uint8_t ser_valid,
      uint16_t trans_buf_size,
      uint16_t recv_buf_size,
      uint32_t timeout
)
```

Parameters

socket_num	Socket number
cert_info	Pointer to certificate information storage area
sni_name	Server Name Indication (SNI)
ser_valid	server validation
trans_buf_size	Incoming buffer length for TLS socket
recv_buf_size	Outgoing buffer length for TLS socket
timeout	SSL connection timeout

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 SNI (supported only for TLS client).
- Enable server validation.
- Set the Incoming buffer length.
- Set the Outgoing buffer length.
- Set the DA TLS connection timeout (ms).

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

Reentrant

No

Example

```
  char[] HostName = “awu6os-ats.iot.ap-northeast-1.amazonaws.com“;
  R_WIFI_DA16XXX_ConfigTlsSocket(&socketId, &cert_info, (uint8_t *)pHostName, 1,
8192, 8192, 1000);
```

Special Notes:

None

3.36. R_WIFI_DA16XXX_RegistServerCertificate()

This function registers server certificates on the Wi-Fi module (Deprecated, using R_WIFI_DA16XXX_ConfigTlsSocket() instead).

Format

```
wifi_err_t R_WIFI_DA16XXX_RegistServerCertificate(
        uint32_t socket_num,
        wifi_tls_cert_info_t * cert_info,
        uint8_t WIFI_FAR * sni_name,
        uint8_t ser_valid,
        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

3.37. R_WIFI_DA16XXX_RequestTlsSocket()

This function allocates the created TLS socket for SSL connection.

Format

```
wifi_err_t R_WIFI_DA16XXX_RequestTlsSocket (
      uint32_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

3.38. 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

3.39. 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,
      uint16_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

3.40. 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

3.41. 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

3.42. 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

3.43. 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

3.44. 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

3.45. 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

3.46. 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

3.47. 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

3.48. 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

3.49. 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

3.50. 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

3.51. 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

3.52. R_WIFI_DA16XXX_OtaOpen()

This function initializes DA16XXX OTA service.

Format

```
wifi_err_t R_WIFI_DA16XXX_OtaOpen (
      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_OTA_ALREADY_OPEN	Already WIFI OTA opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

Initialize the DA16XXX on-chip OTA service.

Reentrant

No

Example

```
R_WIFI_DA16XXX_OtaOpen();
```

Special Notes:

None

3.53. R_WIFI_DA16XXX_OtaClose()

This function closes the DA16XXX OTA service.

Format

```
wifi_err_t R_WIFI_DA16XXX_OtaClose (
      void
)
```

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_MODULE_COM	Cannot communicate WIFI module
WIFI_ERR_OTA_NOT_OPEN	WIFI OTA service is not opened
WIFI_ERR_OTA_FAIL	WIFI OTA common error

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function closes the DA16XXX OTA service.

Reentrant

No

Example

```
R_WIFI_DA16XXX_OtaOpen();
R_WIFI_DA16XXX_OtaClose();
```

Special Notes:

None

3.54. R_WIFI_DA16XXX_OtaStart()

Start downloading firmware from an OTA server.

Format

```
wifi_err_t R_WIFI_DA16XXX_OtaStart (
      const uint8_t * fw_url
)
```

Parameters

fw_url	Server URL where firmware exists

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_OTA_NOT_OPEN	WIFI OTA service is not opened
WIFI_ERR_OTA_NOT_IDLE	WIFI OTA state is not IDLE

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function starts downloading firmware from an OTA server with firmware URL.

Reentrant

No

Example

```
R_WIFI_DA16XXX_OtaStart(“firmware_url”);
```

Special Notes:

None

3.55. R_WIFI_DA16XXX_OtaEraseFirmware()

Erase the MCU firmware stored in a serial flash of the DA16200/DA16600.

Format

```
wifi_err_t R_WIFI_DA16XXX_OtaEraseFirmware (
      void
)
```

Parameters

fw_url	Server URL where firmware exists

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_NOT_CONNECT	Not connect to access point
WIFI_ERR_OTA_NOT_OPEN	WIFI OTA service is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function erases firmware stored in a serial flash of DA16200/DA16600.

Reentrant

No

Example

```
 R_WIFI_DA16XXX_OtaEraseFirmware();
```

Special Notes:

None

3.56. R_WIFI_DA16XXX_OtaGetProgress()

Get progress status of firmware download.

Format

```
wifi_err_t R_WIFI_DA16XXX_OtaGetProgress (
      uint8_t * progress
)
```

Parameters

progress	Pointer to a variable where the OTA update progress (in percentage) will be stored

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_NOT_CONNECT	Not connect to access point
WIFI_ERR_OTA_NOT_OPEN	WIFI OTA service is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets progress status of firmware download.

Reentrant

No

Example

```
uint8_t progress;
R_WIFI_DA16XXX_OtaGetProgress(&progress);
```

Special Notes:

None

3.57. R_WIFI_DA16XXX_OtaReadFirmwarebyBlock()

Read the MCU firmware as much as the size from the fw_addr and transmit it.

Format

```
wifi_err_t R_WIFI_DA16XXX_OtaReadFirmwarebyBlock (
      uint32_t fw_addr,
      uint32_t size
)
```

Parameters

fw_addr	Starting address from which the firmware block will be read.
size	Size (in bytes) of the firmware block to read.

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_NOT_CONNECT	Not connect to access point
WIFI_ERR_OTA_NOT_OPEN	WIFI OTA service is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function reads the MCU firmware as much as the size from the fw_addr and transmit it.

Reentrant

No

Example

```
uint32_t fw_addr = 0x3AD000;
uint32_t size = 1024;
R_WIFI_DA16XXX_OtaReadFirmwarebyBlock(fw_addr, size);
```

Special Notes:

None

3.58. R_WIFI_DA16XXX_OtaGetAddress()

Get the address where images is stored in DA16600/DA16200.

Format

```
wifi_err_t R_WIFI_DA16XXX_OtaGetAddress (
      uint32_t * fw_addr
)
```

Parameters

fw_addr	Pointer to a variable that will store the firmware address in the DA device.

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_NOT_CONNECT	Not connect to access point
WIFI_ERR_OTA_NOT_OPEN	WIFI OTA service is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets the address where images is stored in DA16600/DA16200.

Reentrant

No

Example

```
uint32_t fw_addr = 0;
R_WIFI_DA16XXX_OtaGetAddress(&fw_addr);
```

Special Notes:

None

3.59. R_WIFI_DA16XXX_OtaFirmwareSize()

Get a size in the header of the MCU firmware.

Format

```
wifi_err_t R_WIFI_DA16XXX_OtaFirmwareSize (
      uint32_t * fw_size
)
```

Parameters

fw_size	Pointer to a variable that will store the size of the image in the DA device.

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_NOT_CONNECT	Not connect to access point
WIFI_ERR_OTA_NOT_OPEN	WIFI OTA service is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets size of image in the header of the MCU firmware.

Reentrant

No

Example

```
uint32_t size = 0;
R_WIFI_DA16XXX_OtaFirmwareSize(&fw_size);
```

Special Notes:

None

3.60. R_WIFI_DA16XXX_OtaGetState()

Get state of OTA transfer.

Format

```
wifi_err_t R_WIFI_DA16XXX_OtaGetState (
      wifi_ota_state_t * state
)
```

Parameters

state	Pointer to a variable that will store the current state of OTA download

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_OTA_NOT_OPEN	WIFI OTA service is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets state of OTA transfer.

Reentrant

No

Example

```
 wifi_ota_state_t status;
 R_WIFI_DA16XXX_OtaGetState(&status);
```

Special Notes:

None

3.61. R_WIFI_DA16XXX_OtaGetFirmware()

Get the data returned from the API OtaReadFirmwarbyBlock().

Format

```
wifi_err_t R_WIFI_DA16XXX_OtaGetFirmware (
      uint8_t * buffer,
      uint16_t offset,
      uint16_t size
)
```

Parameters

buffer	Pointer to the buffer will receive the data from the API
offset	Offset (in bytes) from the start.
size	Size of the data to retrieve.

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_OTA_NOT_OPEN	WIFI OTA service is not opened

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets the data returned from the API OtaReadFirmwarbyBlock.

Reentrant

No

Example

```
uint8_t user_buf[1024] = {0};
uint16_t offset = 0;
uint16_t size = 0;
R_WIFI_DA16XXX_OtaGetFirmware(user_buf, offset, size);
```

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 */
    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 Wi-Fi DA16600 Multiple Protocols Demo Project

5.1.1 Prerequisites

- Hardware requirements:
  - CK-RX65N: Renesas CK-RX65N Cloud Kit v2 (Product no.: RTK5CK65N0S08001BE).
  - DA16600: US159-DA16600MEVZ as Wi-Fi module (included in CK-RX65N kit)
  - PC running Windows® 11.
  - Micro-USB cables for Power supply and the Wi-Fi module logging output (included as part of the kit. See CK-RX65N v2 – User's Manual at “Related Documents” on page 1).
- Software requirements for Windows 10 PC:
  - IDE: e^2 studio 2025-07 or later.
  - Compiler: Renesas Electronics C/C++ Compiler for RX Family V3.07.00.
  - Tera Term v4.99 or later.
  - Socket Test (for TCP Client demo): http://sockettest.sourceforge.net/.
  - Java Virtual Machine (JVM)1.3 or above (for Socket Test): http://www.java.com/.

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 5.1.5) or by downloading the demo project (see section 5.4).

- Import “ck_rx65n_wifi_da16xxx_baremetal_multiple_protocol” for Bare metal application.
- Import “ck_rx65n_wifi_da16xxx_freertos_multiple_protocol” for FreeRTOS application.

5.1.3 Hardware Setup

- Connect the Wi-Fi DA16600 Pmod module to the CK-RX65N v2 PMOD1 connector.
- Connect the micro-USB cable from PC to CK-RX65N micro-USB connector (J14) for Power supply.
- Connect the USB-Type C cable from PC to CK-RX65N micro-USB connector (J10) for logging output.
- Set the jumper of J16 (short 1-2) to “Debug”.

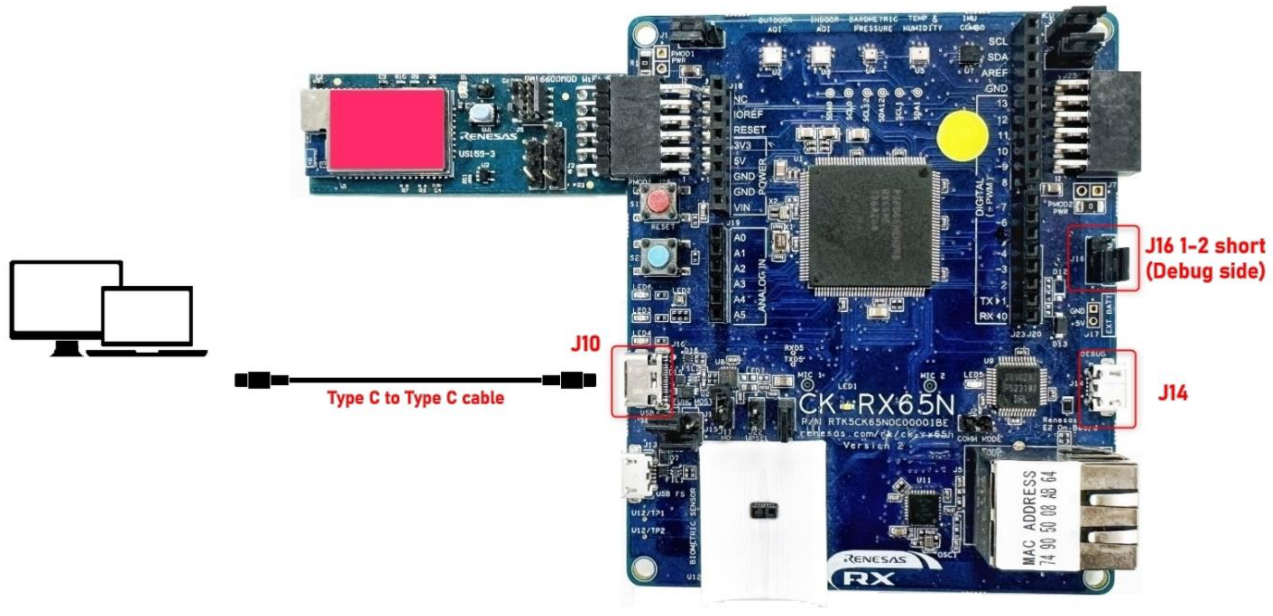
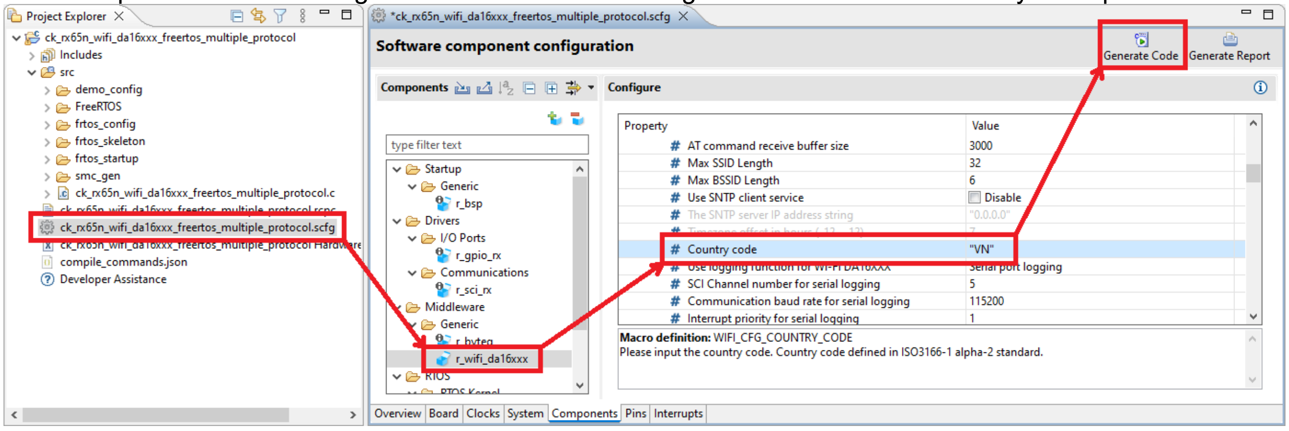


Figure 5.1 Hardware Setup

5.1.4 How to Run the Demo

a) Country Code Setting
Use the Smart Configurator to configure the country code.
Open the Smart Configurator as shown in the image below and set the Country code parameter.



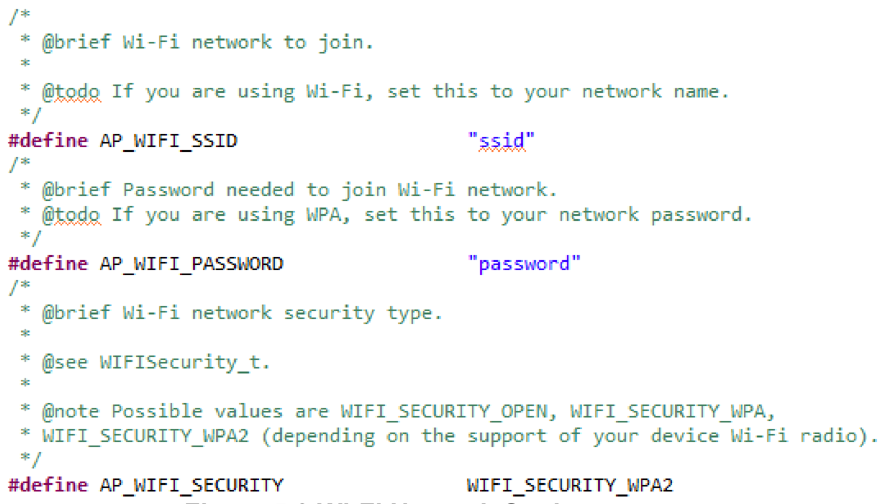
Property	Value
AT command receive buffer size	3000
Max SSID Length	32
Max BSSID Length	6
Use SNTP client service	Disable
The SNTP server IP address string	"0.0.0.0"
Timezone offset in hours (-12 ... 12)	7
Country code	"VN"
Use logging function for Wi-Fi DA16XXX	Serial port logging
SCI Channel number for serial logging	5
Communication baud rate for serial logging	115200
Interrupt priority for serial logging	1

Macro definition: WIFI_CFG_COUNTRY_CODE
Please input the country code. Country code defined in ISO3166-1 alpha-2 standard.

Figure 5.2 Country Code Setting

- "WIFI_CFG_COUNTRY_CODE": Country code defined in ISO 3166-1 alpha-2 standard. Such as KR, US, JP, and CH.

b) Wi-Fi Network Settings
Configure Wi-Fi network settings for the Wi-Fi module. Configure the following macro in "src/demo_config/demo_config.h".
Note: Ensure that the PC running Socket Test app and the Wi-Fi module are connected to the same Wi-Fi network.



```
/*
 * @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 5.3 Wi-Fi Network Settings

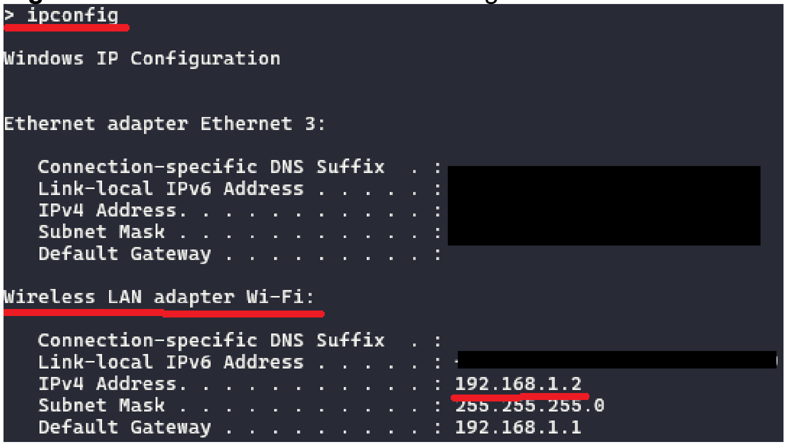
- AP_WIFI_SSID: Set the access point name of the Wi-Fi network.
- AP_WIFI_PASSWORD: Set the Wi-Fi network password.
- AP_WIFI_SECURITY: Set the Wi-Fi network security type (WIFI_SECURITY_OPEN, WIFI_SECURITY_WPA, WIFI_SECURITY_WPA2).

c) TCP Server Demo Settings

Follow the steps below to obtain the IP address in Windows OS.

- Select **Start** > **Settings** > **Network & internet** > **Wi-Fi** and then select the Wi-Fi network you're connected to.
- Under **Properties**, look for your IP address listed next to **IPv4 address**.

Or running "**ipconfig**" command in CMD or PowerShell to get the IP address.

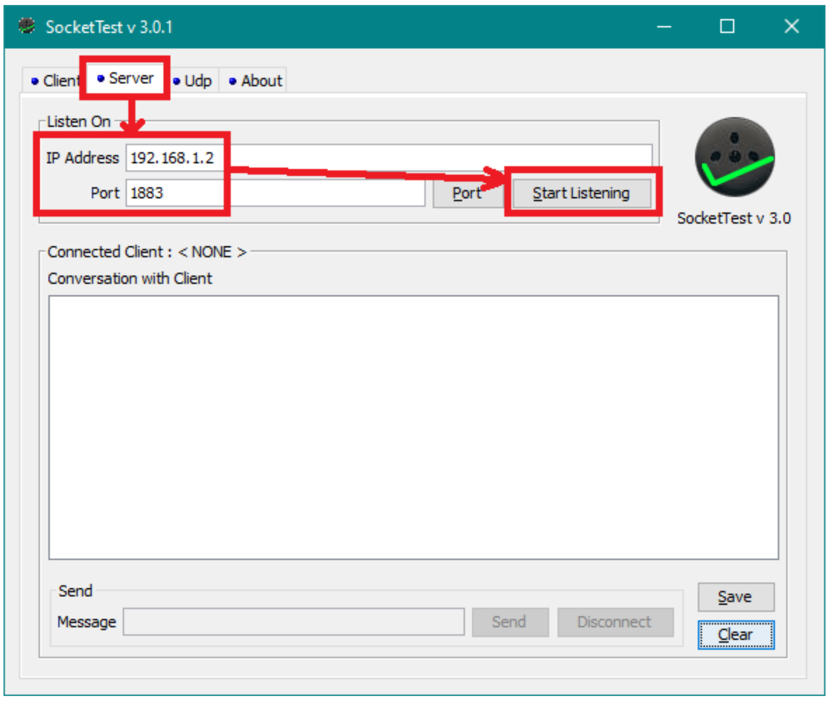


```
> ipconfig

Windows IP Configuration


Ethernet adapter Ethernet 3:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : [redacted]
   IPv4 Address. . . . . . . . . . . :
   Subnet Mask . . . . . . . . . . . :
   Default Gateway . . . . . . . . . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : [redacted]
   IPv4 Address. . . . . . . . . . . : 192.168.1.2
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
```

Figure 5.4 Get Server IP Address

Run Socket Master: "SocketTest-master\dist\SocketTest.jar" on PC.

Input the IP address in the designated text box, the port number is user-defined. In this demo, we use 1883.



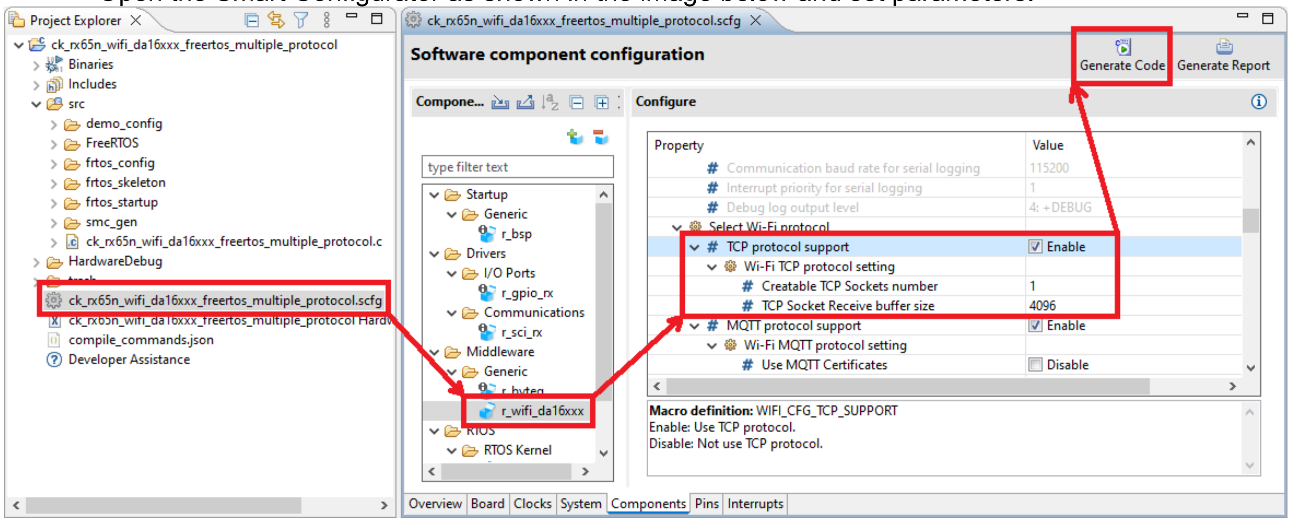
SocketTest v 3.0.1

Listen On
IP Address: 192.168.1.2
Port: 1883

Connected Client : < NONE >
Conversation with Client

Figure 5.5 Start TCP Server

d) TCP Client Demo Settings
Use the Smart Configurator to configure TCP protocol support.
Open the Smart Configurator as shown in the image below and set parameters.



Property	Value
Communication baud rate for serial logging	115200
Interrupt priority for serial logging	1
Debug log output level	4: +DEBUG
Select Wi-Fi protocol	
TCP protocol support	Enable
Wi-Fi TCP protocol setting	
Creatable TCP Sockets number	1
TCP Socket Receive buffer size	4096
MQTT protocol support	Enable
Wi-Fi MQTT protocol setting	
Use MQTT Certificates	Disable

Macro definition: WIFI_CFG_TCP_SUPPORT
Enable: Use TCP protocol.
Disable: Not use TCP protocol.

Figure 5.6 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.

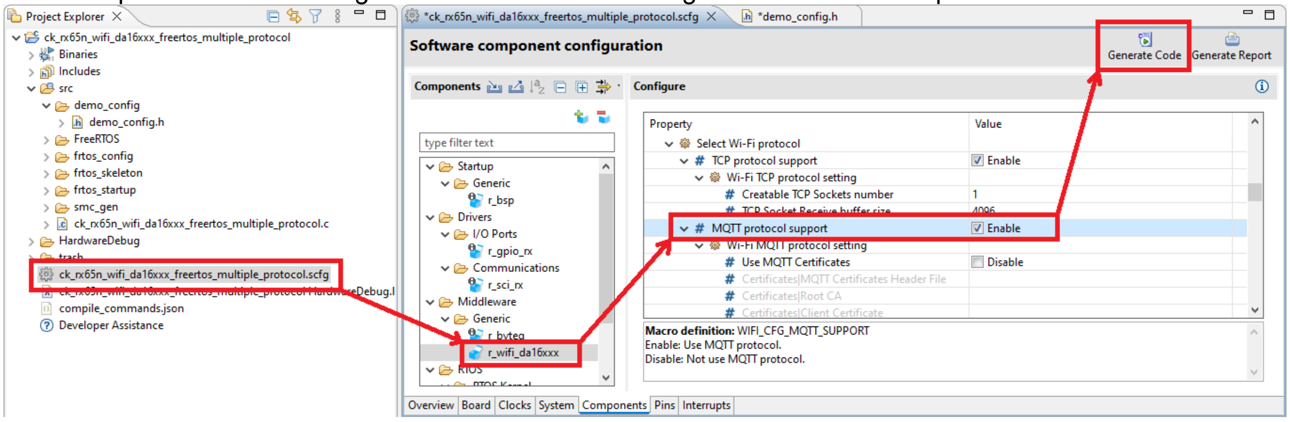
Configure TCP server settings from **c) TCP Server Demo Settings** to the following macro in “src/demo_config/demo_config.h”.

```
/*
 * @brief TCP server host name.
 *
 * @note Set this to your TCP host name server.
 */
#define TCP_SERVER_HOSTNAME           "192.168.1.2"
/*
 * @brief TCP server port.
 *
 * @note Set this to your TCP port server.
 */
#define TCP_SERVER_PORT                1883
```

Figure 5.7 TCP Server Settings

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

e) MQTT On-Chip Client Demo Settings
Use the Smart Configurator to configure the MQTT protocol.
Open the Smart Configurator as shown in the image below and set the parameter.



Property	Value
Select Wi-Fi protocol	
TCP protocol support	☑ Enable
Wi-Fi TCP protocol setting	
Creatable TCP Sockets number	1
TCP Socket Receive buffer size	4096
MQTT protocol support	☑ Enable
Wi-Fi MQTT protocol setting	
Use MQTT Certificates	☐ Disable
Certificates|MQTT Certificates Header File	
Certificates|Root CA	
Certificates|Client Certificate	

Macro definition: WIFI_CFG_MQTT_SUPPORT
Enable: Use MQTT protocol.
Disable: Not use MQTT protocol.

Figure 5.8 MQTT On-Chip Client Settings

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

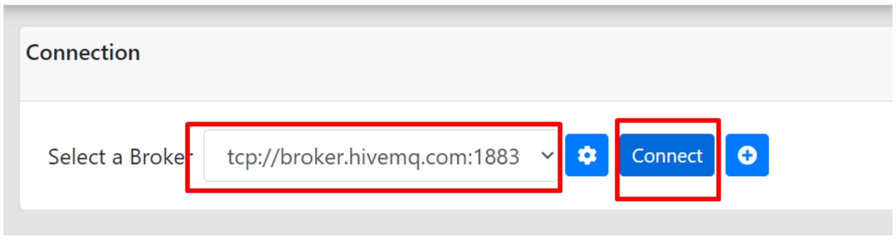
Configure the MQTT Publish/Subscribe topics. Configure the following macro in "src/demo_config/demo_config.h"

```
/*
 * @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 5.9 MQTT Topics Settings

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

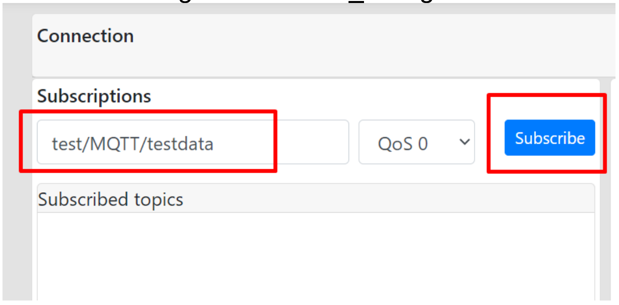
f) MQTT Broker Settings
Open URL: https://testclient-cloud.mqtt.cool/ and select a Broker below.



Select a Broker
tcp://broker.hivemq.com:1883

Figure 5.10 Start MQTT Broker

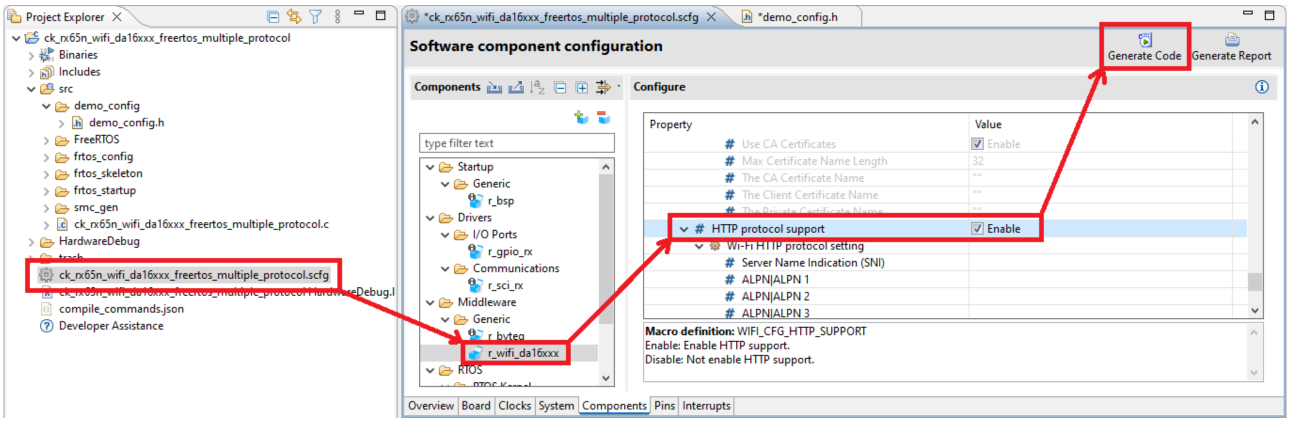
Enter the subscribe topic that was configured in demo_config.h.



test/MQTT/testdata
QoS 0

Figure 5.11 Subscribe Topic

g) HTTP On-Chip Client Demo Settings
Use the Smart Configurator to configure the HTTP protocol.
Open the Smart Configurator as shown in the image below and set the parameter.



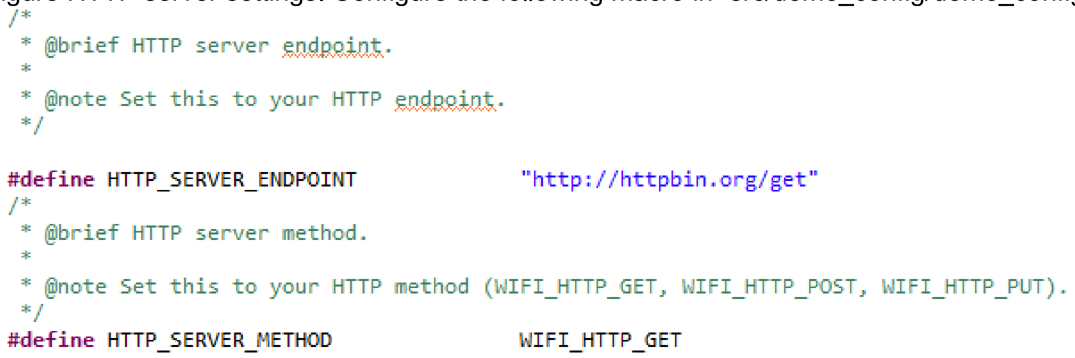
Property	Value
Use CA Certificates	Enable
Max Certificate Name Length	32
The CA Certificate Name	""
The Client Certificate Name	""
The Private Certificate Name	""
HTTP protocol support	☑ Enable
Wi-Fi HTTP protocol setting	
Server Name Indication (SNI)	
ALPN|ALPN 1	
ALPN|ALPN 2	
ALPN|ALPN 3	

Macro definition: WIFI_CFG_HTTP_SUPPORT
Enable: Enable HTTP support.
Disable: Not enable HTTP support.

Figure 5.12 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”



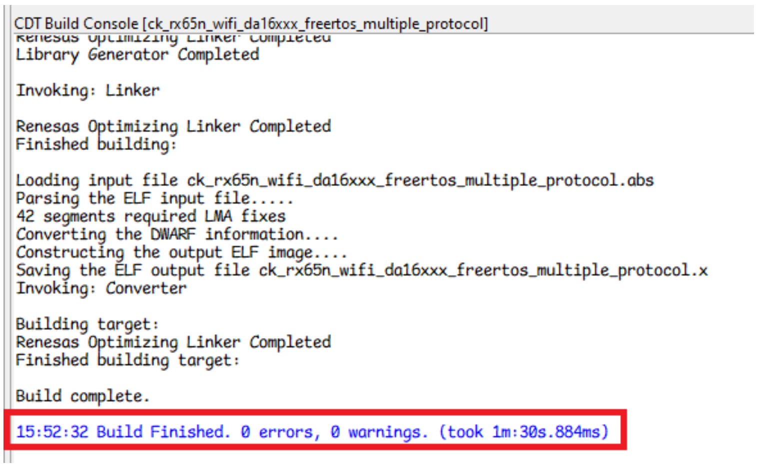
```
/*
 * @brief HTTP server endpoint.
 *
 * @note Set this to your HTTP endpoint.
 */

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

Figure 5.13 HTTP Server Settings

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

The HTTP demo only checks the data in the debug log on Tera Term. Please refer to **6.2.2 Debug with Serial Port Logging** for instructions on using the logging function.

h) Building the Demo Project
Build the project and confirm no build errors occur.



CDT Build Console [ck_rx65n_wifi_da16xxx_freertos_multiple_protocol]

```
Renesas Optimizing Linker Completed
Library Generator Completed

Invoking: Linker

Renesas Optimizing Linker Completed
Finished building:

Loading input file ck_rx65n_wifi_da16xxx_freertos_multiple_protocol.abs
Parsing the ELF input file.....
42 segments required LMA fixes
Converting the DWARF information....
Constructing the output ELF image....
Saving the ELF output file ck_rx65n_wifi_da16xxx_freertos_multiple_protocol.x
Invoking: Converter

Building target:
Renesas Optimizing Linker Completed
Finished building target:

Build complete.

15:52:32 Build Finished. 0 errors, 0 warnings. (took 1m:30s.884ms)
```

Figure 5.14 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**.

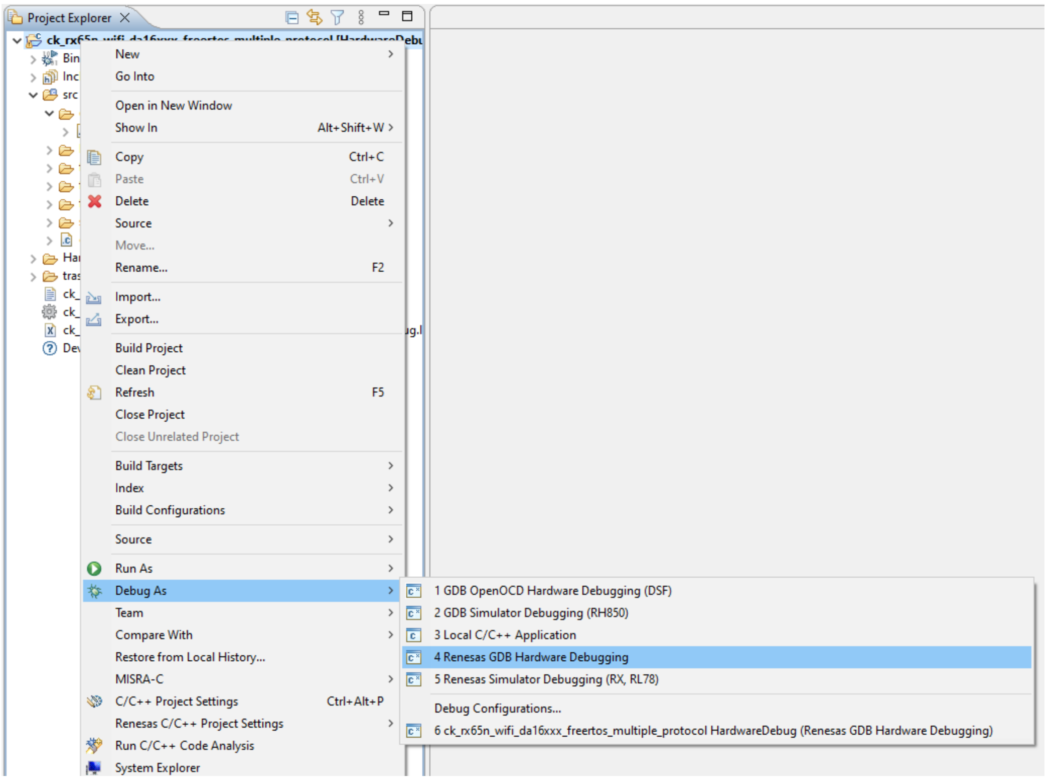
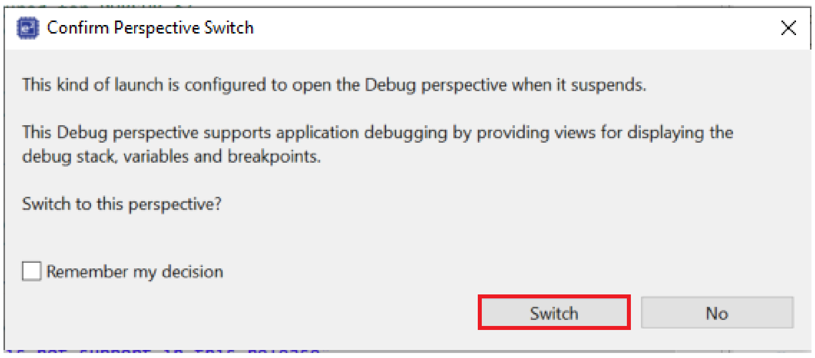

Project Explorer
ck_rx65n_wifi_da16xxx_freertos_multiple_protocol [HardwareDebu
New
Go Into
Open in New Window
Show In Alt+Shift+W
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Source
Move...
Rename... F2
Import...
Export...
Build Project
Clean Project
Refresh F5
Close Project
Close Unrelated Project
Build Targets
Index
Build Configurations
Source
Run As
Debug As
Team
Compare With
Restore from Local History...
MISRA-C
C/C++ Project Settings Ctrl+Alt+P
Renesas C/C++ Project Settings
Run C/C++ Code Analysis
System Explorer
1 GDB OpenOCD Hardware Debugging (DSF)
2 GDB Simulator Debugging (RH850)
3 Local C/C++ Application
4 Renesas GDB Hardware Debugging
5 Renesas Simulator Debugging (RX, RL78)
Debug Configurations...
6 ck_rx65n_wifi_da16xxx_freertos_multiple_protocol HardwareDebug (Renesas GDB Hardware Debugging)

Figure 5.15 Flashing Demo Project

If the window below appears, press "Switch".


Confirm Perspective Switch

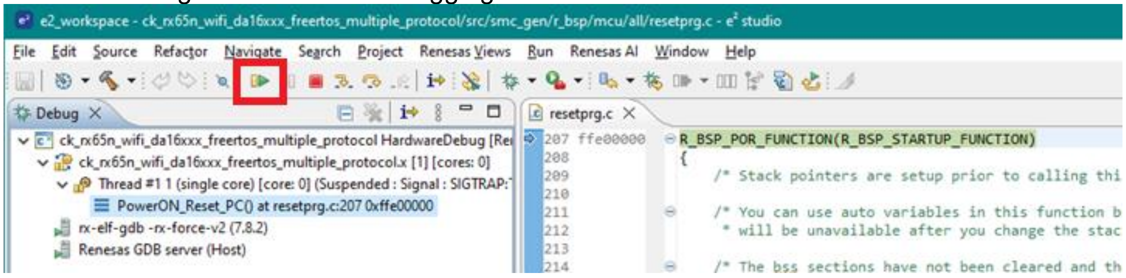
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective supports application debugging by providing views for displaying the debug stack, variables and breakpoints.

Switch to this perspective?

Figure 5.16 Confirm Perspective Switch

Press the following button to start debugging.


e2_workspace - ck_rx65n_wifi_da16xxx_freertos_multiple_protocol/src/smc_gen/r_bsp/mcu/all/resetprg.c - e² studio

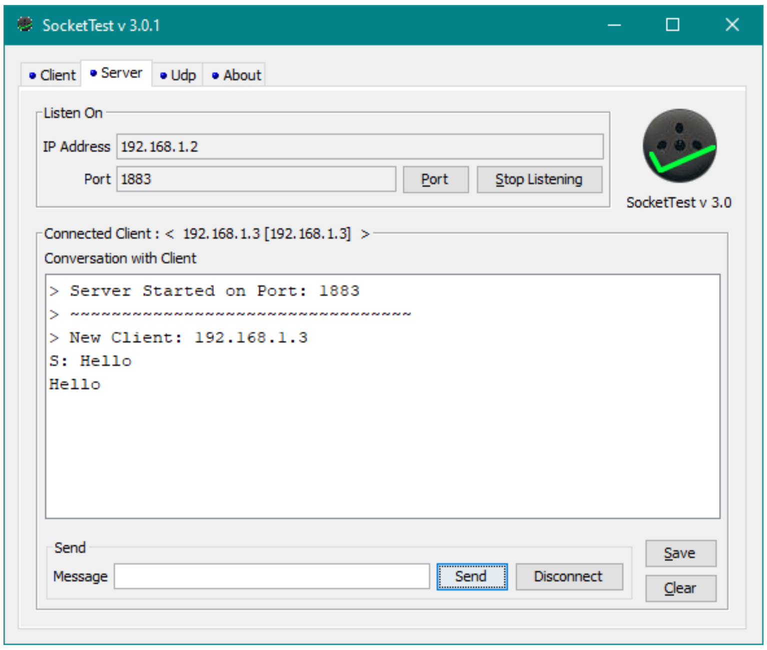
File Edit Source Refactor Navigate Search Project Renesas Views Run Renesas AI Window Help
Debug
resetprg.c

```
ck_rx65n_wifi_da16xxx_freertos_multiple_protocol HardwareDebug [Re
  ck_rx65n_wifi_da16xxx_freertos_multiple_protocol.x [1] [cores: 0]
    Thread #1 1 (single core) [core: 0] (Suspended : Signal : SIGTRAP:
      PowerON_Reset_PC() at resetprg.c:207 0xffe00000
  rx-elf-gdb -rx-force-v2 (7.8.2)
  Renesas GDB server (Host)
```

```
ffe00000  R_BSP_POR_FUNCTION(R_BSP_STARTUP_FUNCTION)
          {
              /* Stack pointers are setup prior to calling thi

              /* You can use auto variables in this function b
               * will be unavailable after you change the stac

              /* The bss sections have not been cleared and th
```

Figure 5.17 Start Debugging

i) Starting the TCP Demo
Wait for the SocketTest to display "**New Client:...**" to confirm that the Wi-Fi module is ready to run the TCP protocol.
After that, send a message and check if the sent data matches the received data in the message box.

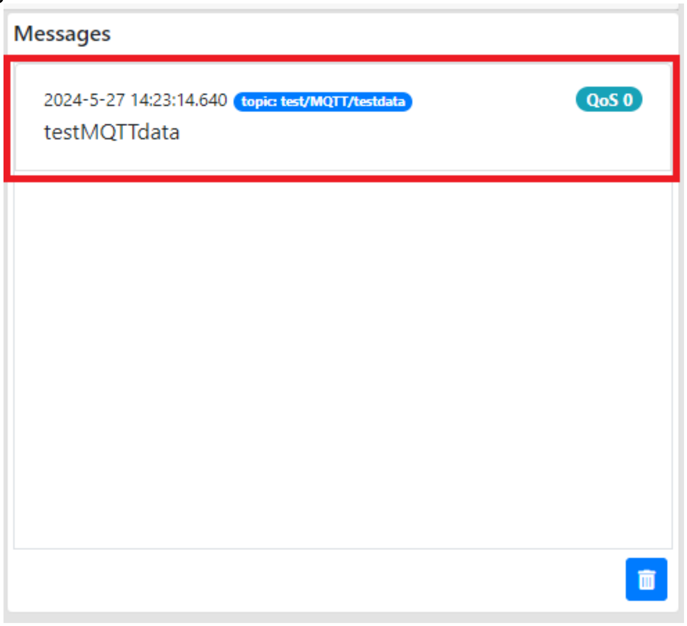

SocketTest v 3.0.1

IP Address 192.168.1.2
Port 1883
Connected Client : < 192.168.1.3 [192.168.1.3] >

```
> Server Started on Port: 1883
> ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> New Client: 192.168.1.3
S: Hello
Hello
```

Figure 5.18 Demo with TCP Client

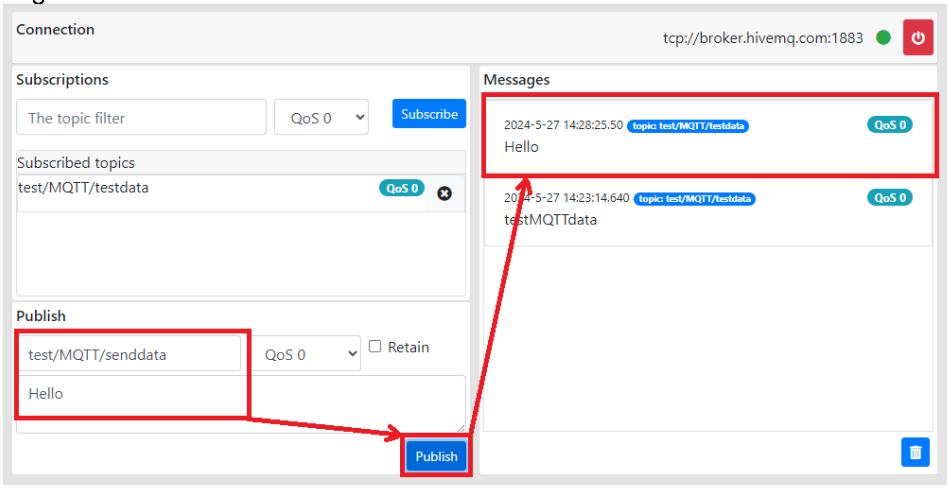
j) Starting the MQTT On-Chip Client Demo
Wait for the Wi-Fi module sends a topic to a subcribe topic that was configured in demo_config.h. It will display in Messages box.


Messages

2024-5-27 14:23:14.640 topic: test/MQTT/testdata QoS 0
testMQTTdata

Figure 5.19 MQTT Message From the Wi-Fi Module

Send a data from topic “test/MQTT/senddata”, and check if the sent data matches the received data in the Messages box.


Connection tcp://broker.hivemq.com:1883

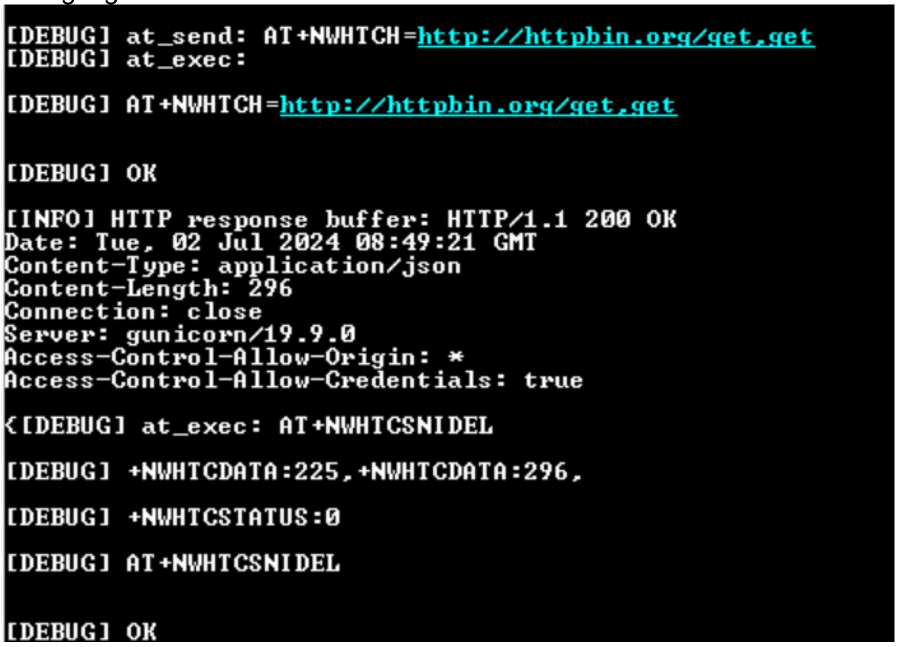
Subscriptions
The topic filter QoS 0 Subscribe
Subscribed topics
test/MQTT/testdata QoS 0

Publish
test/MQTT/senddata QoS 0 Retain
Hello
Publish

Messages
2024-5-27 14:28:25.50 topic: test/MQTT/testdata QoS 0
Hello
2024-5-27 14:23:14.640 topic: test/MQTT/testdata QoS 0
testMQTTdata

Figure 5.20 Demo with MQTT On-Chip Client

k) Starting the HTTP On-Chip Client Demo
Confirm debug log on Tera Term.



```
[DEBUG] at_send: AT+NWHTCH=http://httpbin.org/get,get
[DEBUG] at_exec:

[DEBUG] AT+NWHTCH=http://httpbin.org/get,get


[DEBUG] OK

[INFO] HTTP response buffer: HTTP/1.1 200 OK
Date: Tue, 02 Jul 2024 08:49:21 GMT
Content-Type: application/json
Content-Length: 296
Connection: close
Server: gunicorn/19.9.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

{[DEBUG] at_exec: AT+NWHTCSNIDEL

[DEBUG] +NWHTCDATA:225,+NWHTCDATA:296,

[DEBUG] +NWHTCSTATUS:0

[DEBUG] AT+NWHTCSNIDEL


[DEBUG] OK
```

Figure 5.21 HTTP On-Chip Client Debug Log

Note: The log output setting in this demo is enabled as follows:

- WIFI_CFG_LOGGING_OPTION: "**Serial port**" to print debug log on Tera Term via USB interface (J20).
- WIFI_CFG_DEBUG_LOG: debug log **level 4** to display all log information of the Wi-Fi module.

Please refer to **6.2.2 Debug with Serial Port Logging** for instructions on how to debug with serial port.

5.1.5 Porting the Demo Project to Another Device

How to create the demo project on the other MCUs other than RX65N.

1. Create demo project specifying CK-RX65N as **Target Board.**
2. Double-click .scfg in Project Explorer, and then click Board tab.
3. Change **Target Board** or **Target Device.**
4. Click ***Next*** -> ***Finish***
5. Confirm that the board or device are changed as you specified in previous step and click ***Generate Code*** button.
6. Change the following settings as needed.
    - Wi-Fi DA16XXX Module FIT: .scfg -> Components tab -> r_wifi_da16xxx(*)
    - Communications Setting：.scfg -> Components tab -> r_sci_rx(**)

(*)(**): Configure these macros appropriately for each target board or device by following the board-dependent settings specified in *r_wifi_da16xxx_config.h*. See table below for the list of supported board series for porting.

Table 5.1 Verified Boards Supporting Porting

Verified Supported Boards (Portable)	PMOD Supported
RX65N Cloud Kit	PMOD
RX65N Envision Kit	PMOD
RX65N RSK (2MB)	PMOD1 and PMOD2
Cloud Kit for RX65N v1	PMOD1 and PMOD2
Cloud Kit for RX65N v2	PMOD1 and PMOD2
RX671 Target Board	PMOD
RX66N Target Board	PMOD
FPB-RX261	Only PMOD1
EK-RX261	Only PMOD1
EK-RX671	PMOD1 and PMOD2
FPB-RX140	PMOD1 and PMOD2

Note: Refer to section ***1.2.2 Hardware Configuration*** for comprehensive details.

5.2 Wi-Fi DA16600 OTA on-chip Demo Project

The demo project can be accessed through the link below. This demonstration offers a comprehensive overview of the implemented features and functionalities. Please refer to the provided link for a detailed exploration of the project.
[Link] https://www.renesas.com/document/apn/rx-family-aws-cloud-connectivity-mcu-firmware-update-over-air-ck-rx65n-v2-wi-fi-da16600

5.3 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”.

5.4 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)

Table 6.4 Confirmed Operation Environment (Ver. 1.30)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2024.04
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.1.30
Board used	Renesas CK-RX65N Cloud Kit (Product no.: RTK5CK65N0S04000BE)

Table 6.5 Confirmed Operation Environment (Ver. 1.31)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2024.10
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.1.31
Board used	Renesas CK-RX65N Cloud Kit (Product no.: RTK5CK65N0S04000BE)

Table 6.6 Confirmed Operation Environment (Ver. 1.32)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2025.01
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.1.32
Board used	Renesas CK-RX65N Cloud Kit (Product no.: RTK5CK65N0S04000BE)

Table 6.7 Confirmed Operation Environment (Ver. 1.33)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2025.01
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 8.3.0.202405 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.1.33
Board used	Renesas CK-RX65N v1 Cloud Kit (Product no.: RTK5CK65N0S04000BE)
	Renesas CK-RX65N v2 Cloud Kit (Product no.: RTK5CK65N0S08001BE)
	Renesas EK-RX671 Evaluation Kit (Product no.: RTK5EK6710S00001BE)
	Renesas RX66N Target Board (Product no.: RTK5RX66N0C00000BJ)
	Renesas RX671 Target Board (Product no.: RTK5RX6710C00000BJ)
	Renesas RX140 Fast Prototyping Board (Product no.: RTK5FP1400S00001BE)
	Renesas EK-RX261 Evaluation Kit (Product no.: RTK5EK2610S00001BE)
	Renesas RX261 Fast Prototyping Board (Product no.: RTK5FP2610S00001BE)

Table 6.8 Confirmed Operation Environment (Ver. 1.40)

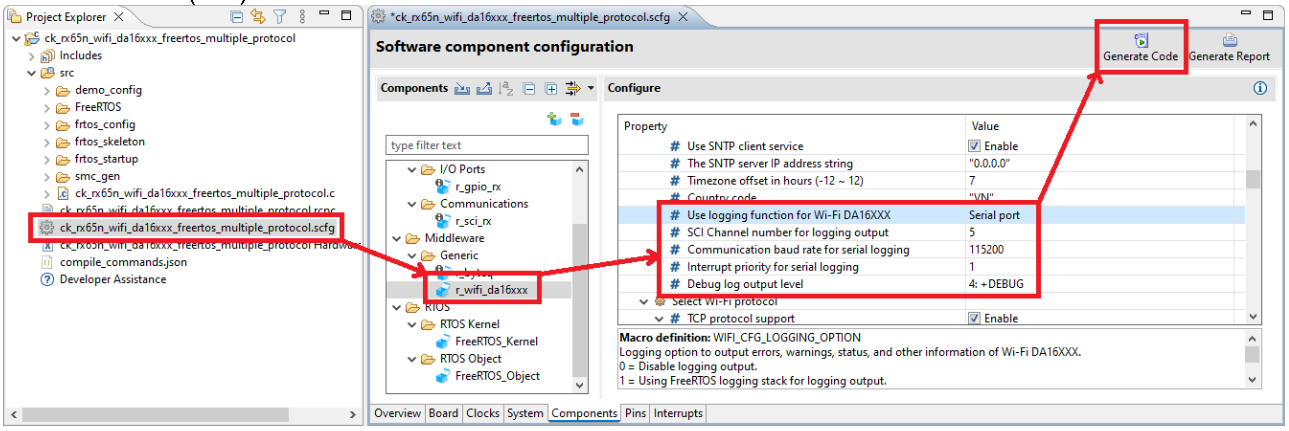
Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2025.07
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.07.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 14.2.0.202505 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.1.40
Board used	Renesas CK-RX65N v2 Cloud Kit (Product no.: RTK5CK65N0S08001BE)
	Renesas RX140 Fast Prototyping Board (Product no.: RTK5FP1400S00001BE)

6.2 Support Logging Function

6.2.2 Debug with Serial Port Logging

Configures the logging function for Wi-Fi module to print the debug log via SCI channel have been selected.

For the CK-RX65N Cloud Kit v2 (RTK5CK65N0S08001BE), the log will be output to the USB interface (J10) as follows:



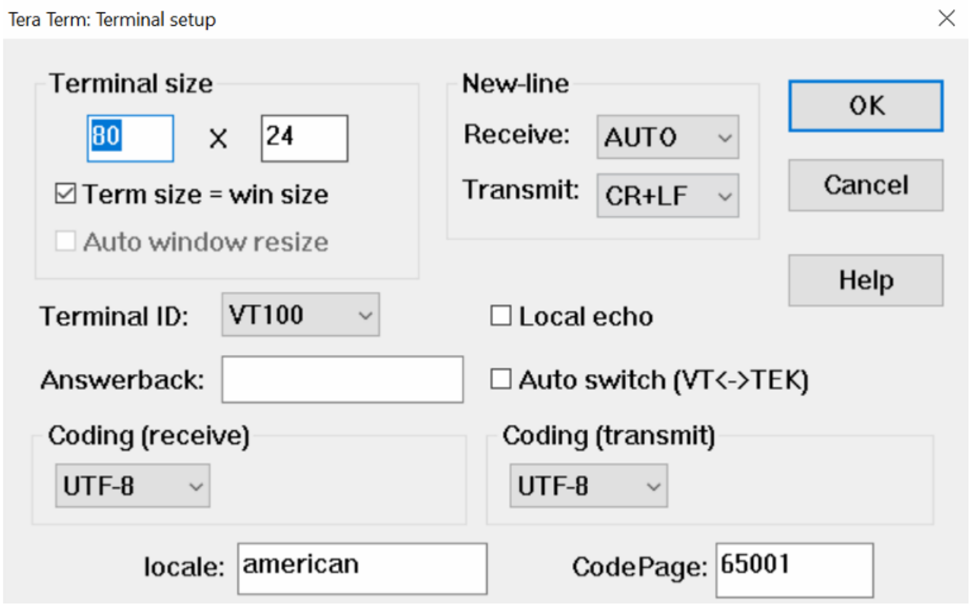
Property	Value
Use SNTP client service	✓ Enable
The SNTP server IP address string	"0.0.0.0"
Timezone offset in hours (-12 ~ 12)	7
Country code	"VN"
Use logging function for Wi-Fi DA16XXX	Serial port
SCI Channel number for logging output	5
Communication baud rate for serial logging	115200
Interrupt priority for serial logging	1
Debug log output level	4: +DEBUG
Select Wi-Fi protocol	
TCP protocol support	✓ Enable

Macro definition: WIFI_CFG_LOGGING_OPTION
Logging option to output errors, warnings, status, and other information of Wi-Fi DA16XXX.
0 = Disable logging output.
1 = Using FreeRTOS logging stack for logging output.

Figure 6.1 Logging Output Settings for Serial Port Logging

- WIFI_CFG_LOGGING_OPTION: Choose "Serial port".
- WIFI_CFG_LOG_TERM_CHANNEL: SCI channel for logging function.
- WIFI_CFG_SCI_UART_TERMINAL_BAUDRATE: Baud rate for serial logging (unit in bps).
- WIFI_CFG_SCI_UART_INTERRUPT_PRIORITY: Interrupt priority (default is 1).
- WIFI_CFG_DEBUG_LOG: Debug log level.

Configures Tera Term terminal, please select **Setup** -> **Terminal…**



Tera Term: Terminal setup

Terminal size: 80 x 24

☑ Term size = win size

☐ Auto window resize

New-line

Receive: AUTO

Transmit: CR+LF

Terminal ID: VT100

☐ Local echo

Answerback:

☐ Auto switch (VT<->TEK)

Coding (receive): UTF-8

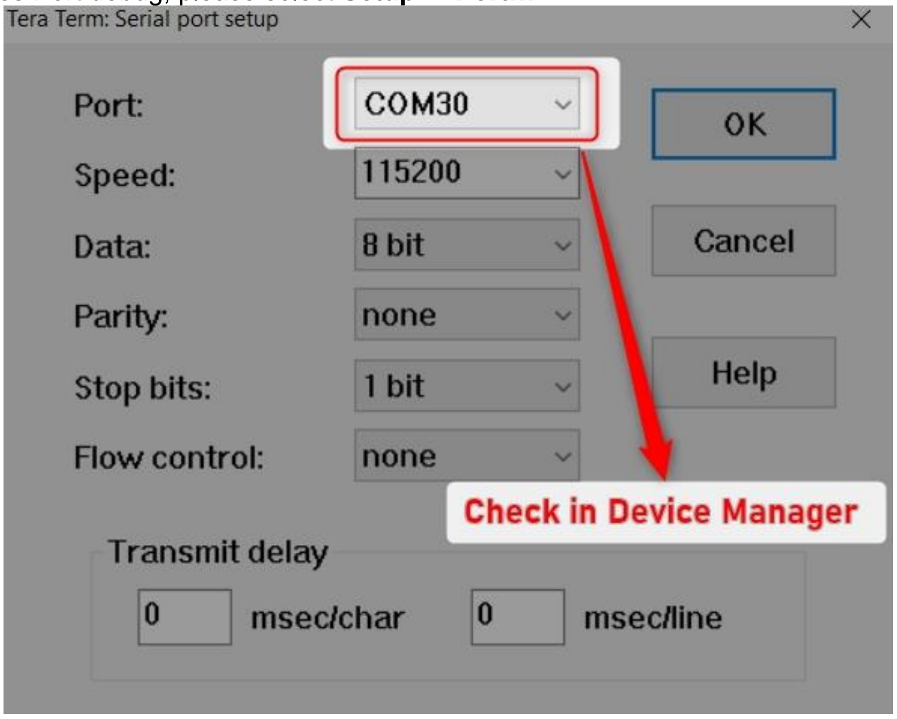
Coding (transmit): UTF-8

locale: american

CodePage: 65001

Figure 6.2 Tera Term Settings

Configures Port debug, please select **Setup** -> **Port…**



Tera Term: Serial port setup

Port: COM30

Speed: 115200

Data: 8 bit

Parity: none

Stop bits: 1 bit

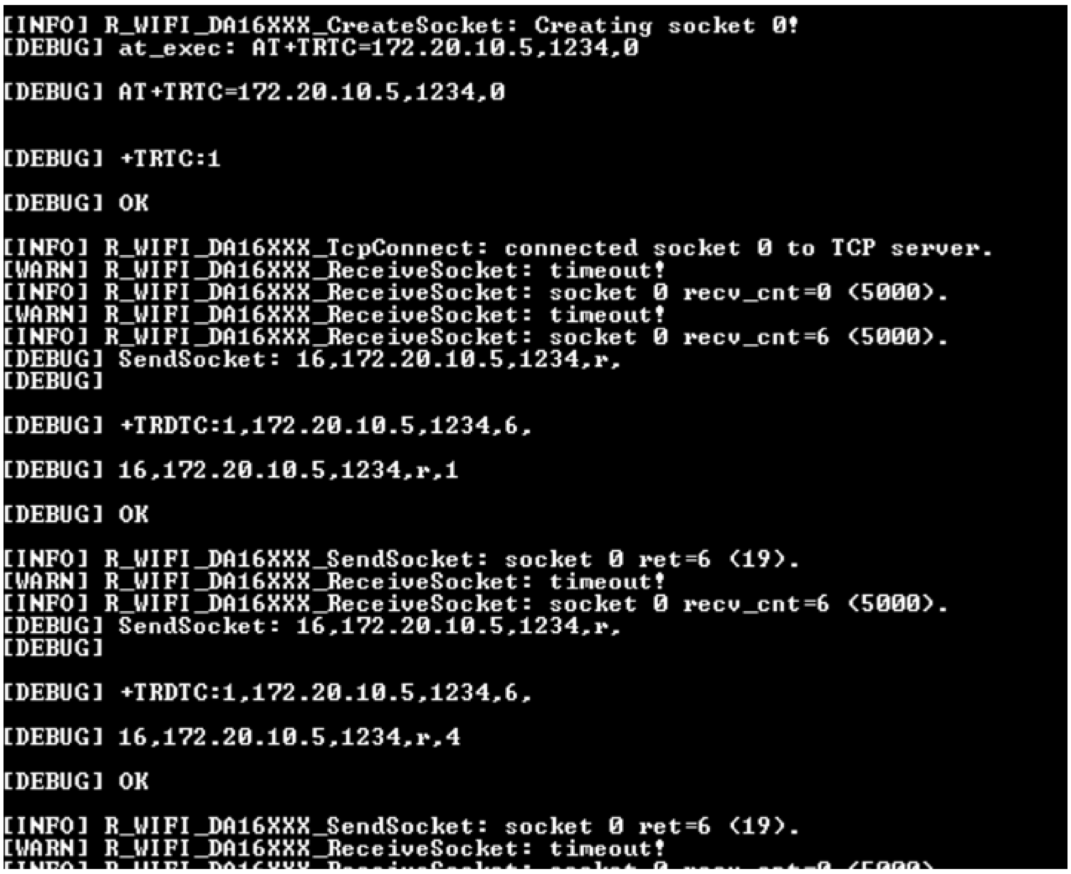
Flow control: none

Transmit delay

0 msec/char 0 msec/line

Figure 6.3 Tera Term Serial Port Settings

With this setting, the log will be output to the serial port. The user can debug the project and see the result on Tera Term as follows.

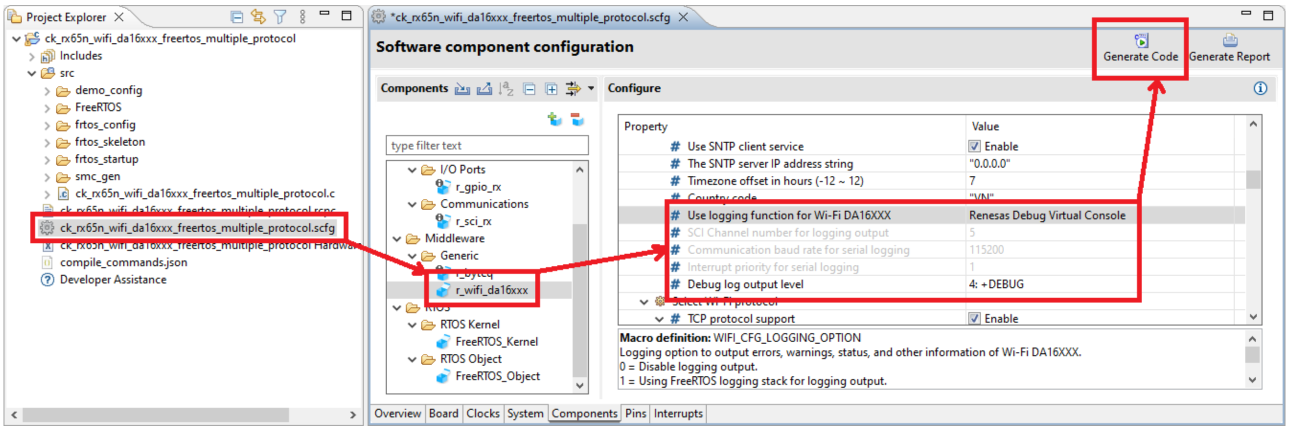


```
[INFO] R_WIFI_DA16XXX_CreateSocket: Creating socket 0!
[DEBUG] at_exec: AT+TRTC=172.20.10.5,1234,0

[DEBUG] AT+TRTC=172.20.10.5,1234,0


[DEBUG] +TRTC:1

[DEBUG] OK

[INFO] R_WIFI_DA16XXX_TcpConnect: connected socket 0 to TCP server.
[WARN] R_WIFI_DA16XXX_ReceiveSocket: timeout!
[INFO] R_WIFI_DA16XXX_ReceiveSocket: socket 0 recv_cnt=0 (5000).
[WARN] R_WIFI_DA16XXX_ReceiveSocket: timeout!
[INFO] R_WIFI_DA16XXX_ReceiveSocket: socket 0 recv_cnt=6 (5000).
[DEBUG] SendSocket: 16,172.20.10.5,1234,r,
[DEBUG]

[DEBUG] +TRDTC:1,172.20.10.5,1234,6,

[DEBUG] 16,172.20.10.5,1234,r,1

[DEBUG] OK

[INFO] R_WIFI_DA16XXX_SendSocket: socket 0 ret=6 (19).
[WARN] R_WIFI_DA16XXX_ReceiveSocket: timeout!
[INFO] R_WIFI_DA16XXX_ReceiveSocket: socket 0 recv_cnt=6 (5000).
[DEBUG] SendSocket: 16,172.20.10.5,1234,r,
[DEBUG]

[DEBUG] +TRDTC:1,172.20.10.5,1234,6,

[DEBUG] 16,172.20.10.5,1234,r,4

[DEBUG] OK

[INFO] R_WIFI_DA16XXX_SendSocket: socket 0 ret=6 (19).
[WARN] R_WIFI_DA16XXX_ReceiveSocket: timeout!
```

Figure 6.4 Wi-F Logging on Tera Term

6.2.3 Debug with Renesas Debug Virtual Console

Configures the logging function for Wi-Fi module to print the debug log using Renesas Debug Virtual Console as follows:


Project Explorer
ck_rx65n_wifi_da16xxx_freertos_multiple_protocol.scfg
r_wifi_da16xxx
Software component configuration
Generate Code
Generate Report
Components
Configure

Property	Value
Use SNTP client service	✓ Enable
The SNTP server IP address string	"0.0.0.0"
Timezone offset in hours (-12 ~ 12)	7
Country code	"VN"
Use logging function for Wi-Fi DA16XXX	Renesas Debug Virtual Console
SCI Channel number for logging output	5
Communication baud rate for serial logging	115200
Interrupt priority for serial logging	1
Debug log output level	4: +DEBUG
Select Wi-Fi protocol	
TCP protocol support	✓ Enable

Macro definition: WIFI_CFG_LOGGING_OPTION
Logging option to output errors, warnings, status, and other information of Wi-Fi DA16XXX.
0 = Disable logging output.
1 = Using FreeRTOS logging stack for logging output.

Figure 6.5 Logging Output Settings for Virtual Console Logging

- WIFI_CFG_LOGGING_OPTION: Choose "Renesas Debug Virtual Console".
- WIFI_CFG_DEBUG_LOG: Debug log level.

Open Renesas Debug Virtual Console.

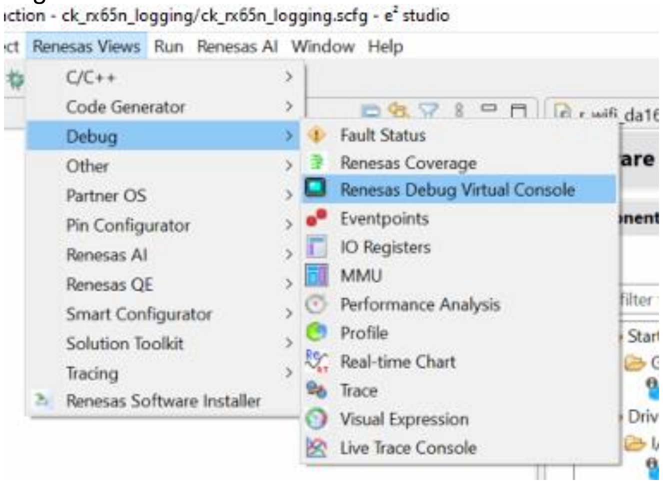
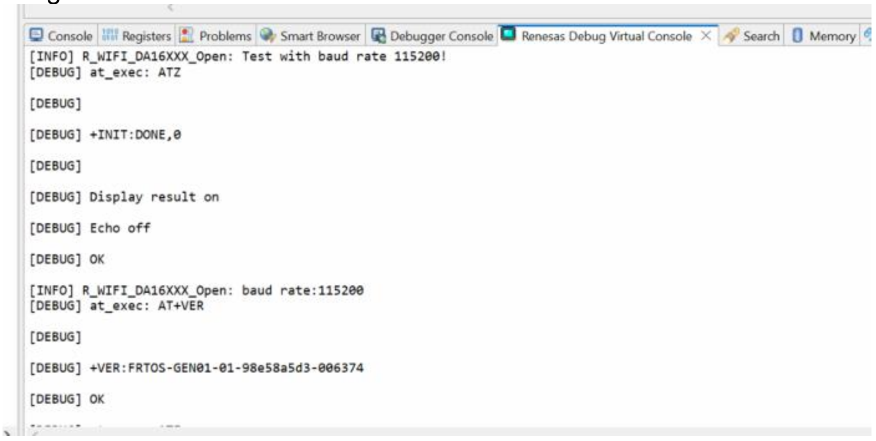


Figure 6.6 Choose Renesas Debug Virtual Console

When the setup is prepared completely, the user can debug the project and see the result on Renesas Debug Virtual Console as follows:



```
[INFO] R_WIFI_DA16XXX_Open: Test with baud rate 115200!
[DEBUG] at_exec: ATZ

[DEBUG]

[DEBUG] +INIT:DONE,0

[DEBUG]

[DEBUG] Display result on

[DEBUG] Echo off

[DEBUG] OK

[INFO] R_WIFI_DA16XXX_Open: baud rate:115200
[DEBUG] at_exec: AT+VER

[DEBUG]

[DEBUG] +VER:FRTOS-GEN01-01-98e58a5d3-006374

[DEBUG] OK
```

Figure 6.7 Wi-Fi Logging on Renesas Debug Virtual Console

6.3 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.

6.4 Limitations

The PMOD on the RX66N and RX671 target boards is configured to Type 6A by default from the factory, which is not compatible with Wi-Fi module initialization. Therefore, changing the PMOD type to 2A or 3A is required. Please refer to the instructions below:

- Remove ***SS13, SS14*** and short circuit ***SC1, SC2***.

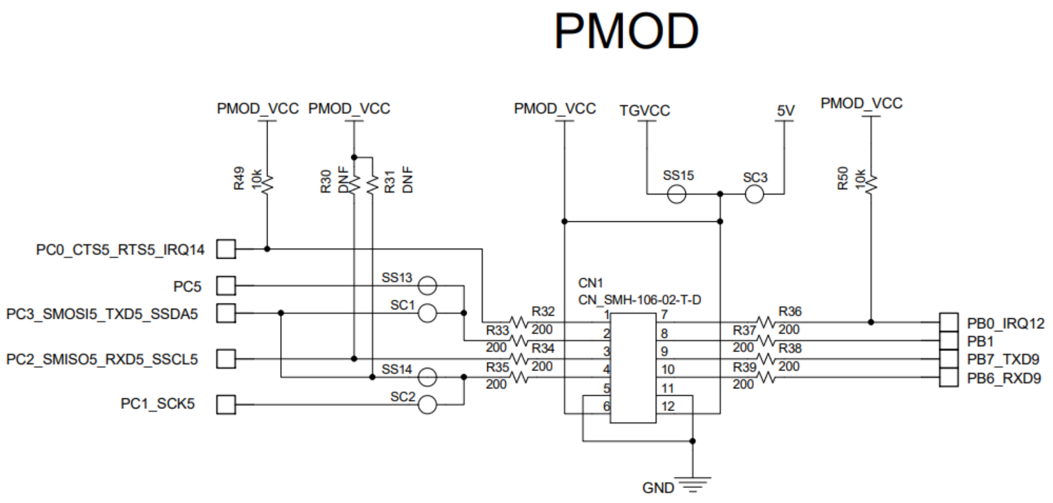


Figure 6.8 Circuit Schematic for the PMOD Connector of RX66N and RX671 Target boards

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
		21	Updated Return values
		22-24	Updated Parameters
		26	Added section 2.12. “for”, “while” and “do while” statements
		28	Added new API: R_WIFI_DA16XXX_IsOpened()
		43	Added new API: R_WIFI_DA16XXX_HardwareReset()
		44	Added new API: R_WIFI_DA16XXX_GetVersion()
		52	Added new API: R_WIFI_DA16XXX_TcpReconnect()
		52-65	Added new APIs for TLS socket
		66-73	Added new APIs for MQTT on-chip
		74-76	Added new APIs for HTTP on-chip
		79	Added callback function for MQTT on-chip
		80-88	Added Section 5. Demo Projects
		89	Added Table 6-3 Confirmed Operation Environment (Ver. 1.20)
1.30	July. 16, 2024	5	Updated section 1.2 to add Bare metal feature
		6	Updated Figure 1.2 to add Bare metal
		14	Updated Table 2.1 to add Logging output function
		18	Updated Table 2.3 to add Bare metal option
		20	Updated Table 2.4 with memory sizes for r_wifi_da16xxx rev 1.30
		21	Updated Section 2.9 with API error code tables
		22-24	Updated Section 2.10 with Parameter structure tables
		27	Removed Section 2.13 RTOS Usage Requirement and added section 2.13 Limitations
		82-90	Updated Section 5.1.4 How to Run the Demo
		93	Added Table 6.4 Confirmed Operation Environment (Ver. 1.30)
		94-97	Added Section 6.2 Support Logging Function
1.31	Jan. 13, 2025	8	Updated section 1.3 API Overview to add R_WIFI_DA16XXX_ConfigTlsSocket()
		20-21	Updated Code Size for r_wifi_da16xxx rev.1.31
		62-63	Added new API: R_WIFI_DA16XXX_ConfigTlsSocket()
		95	Added Table 6-5 Confirmed Operation Environment (Ver. 1.31)
1.32	Mar. 18, 2025	20-21	Updated Code Size for r_wifi_da16xxx rev.1.32
		95	Added Table 6-6 Confirmed Operation Environment (Ver. 1.32)

1.33	Mar. 27, 2025	1	Updated Top page
		6	Added section 1.2.2. Hardware Configuration
		21-25	Updated Code Size for r_wifi_da16xxx rev.1.33
		29	Updated struct wifi_ip_configuration_t
		37	Section 3: Updated format and example 3.4. R_WIFI_DA16XXX_Ping()
		42	Section 3: Updated format and example 3.9. R_WIFI_DA16XXX_DnsQuery()

		69-68	Section 3: Updated format and example
			3.27.R_WIFI_DA16XXX_GetAvailableTlsSocket()
			3.28.R_WIFI_DA16XXX_GetTlsSocketStatus()
			3.29.R_WIFI_DA16XXX_CreateTlsSocket()
			3.30.R_WIFI_DA16XXX_TlsConnect()
			3.31.R_WIFI_DA16XXX_SendTlsSocket()
			3.32.R_WIFI_DA16XXX_ReceiveTlsSocket()
			3.33.R_WIFI_DA16XXX_CloseTlsSocket()
			3.34.R_WIFI_DA16XXX_TlsReconnect()
			3.35.R_WIFI_DA16XXX_ConfigTlsSocket()

		102	Added Table 6-7 Confirmed Operation Environment (Ver. 1.33)
		108	Added Section 6.4 Limitations
1.40	Oct. 15, 2025	-	Update format layout
		10	Section 1.3 API Overview - Add new API for OTA On-Chip
		15	Section 1.4.5 Status Transitions - Add new Transitions of OTA On-Chip
		21 - 22	Section 2.7 Compile Settings - Add new table for Wi-Fi OTA On-Chip Configuration
		23 - 24	Section 2.8 Code Size - Modfiy codesize for Multiple boards - Adding new code size table: OTA On-Chip only
		29, 33	Section 2.9 Return Values - Add new API Error Codes for OTA On-Chip - Add new Member in Structure for OTA state (wifi_ota_state_t)
		88 - 97	Section 3 API Functions - Add new API functions for OTA On-Chip services
		111	Added new Section 5.1.5 Porting the Demo Project to Another Device
		112	Section 5.2 Adding path link to OTA demo project
		115	Section 6.1 Add new table for Wi-Fi v1.40

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Corporate Headquarters

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

Trademarks

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/.