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 module

DA16200MOD (US159-DA16200MEVZ)
DA16600MOD (US159-DA16600MEVZ)
In the following pages, the DA16XXXMOD is referred to as “the Wi-Fi module”.

Target Devices

RX65N Group

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

Target Compilers

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

Related Documents

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

Table of Contents

1. Overview

1.1. DA16XXX FIT Module

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

1.2. Overview of the DA16XXX Wi-Fi FIT Module

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

1.2.1. Connection with DA16XXX Wi-Fi

Examples of connection to the DA16200 Wi-Fi are shown below.

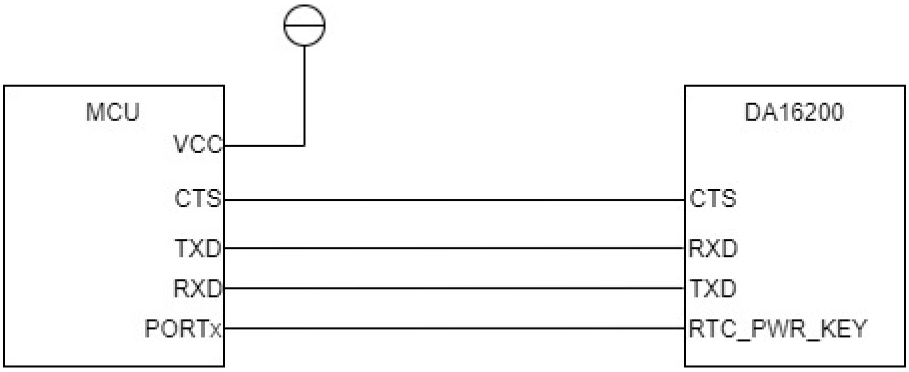

Figure 1-1 Example connection to the DA16200 Wi-Fi module.

1.2.2. Software configuration

Figure 1-2 shows the software configuration.

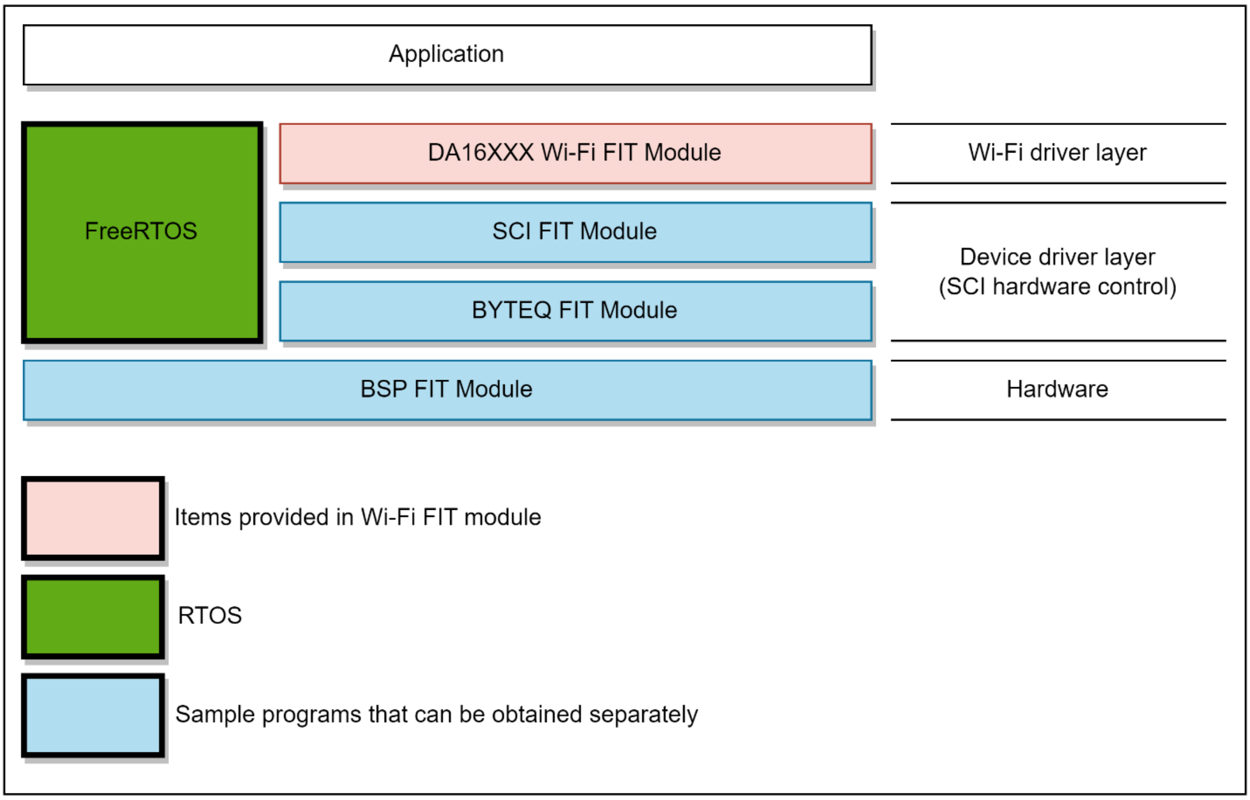

Figure 1-2 Software configuration diagram.

1. DA16XXX Wi-Fi FIT module
   The FIT module. This software is used to control the Wi-Fi module.
2. SCI FIT module
   Implements communication between the Wi-Fi module and the MCU. A sample program is available. Refer to “Related Documents” on page 1 and obtain the software.
3. Peripheral function modules
   This software implements timer control and buffer management. Sample programs are available. Refer to “Related Documents” on page 1 and obtain the software.
4. RTOS
   The RTOS manages the system overall. Operation of the FIT module has been verified using FreeRTOS.

1.3. API Overview

Table 1-1 lists the API functions included in the FIT module. The required memory sizes are lists in 2.8 Code Size.

Table 1-1 API Functions

Function	**Function Description**
R_WIFI_DA16XXX_Open()	Initializes the Wi-Fi module.
R_WIFI_DA16XXX_Close()	Closes 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 current local time based on current time zone in a string.
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_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.

1.4. Status Transitions

Figure 1-3 shows the status transitions of the FIT module up to communication status.

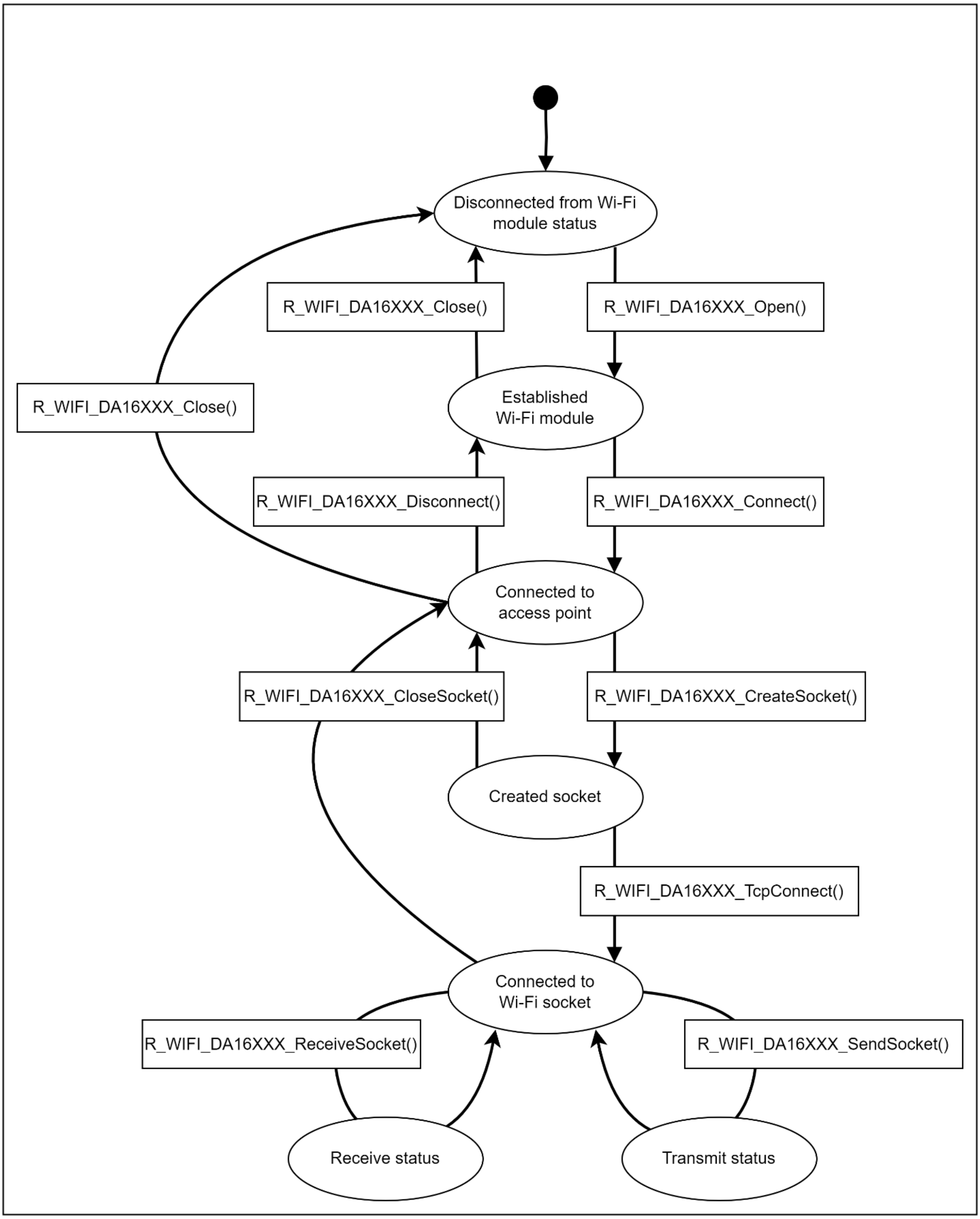
Disconnected from Wi-Fi module status
R_WIFI_DA16XXX_Close()
R_WIFI_DA16XXX_Open()
R_WIFI_DA16XXX_Close()
Established Wi-Fi module
R_WIFI_DA16XXX_Disconnect()
R_WIFI_DA16XXX_Connect()
Connected to access point
R_WIFI_DA16XXX_CloseSocket()
R_WIFI_DA16XXX_CreateSocket()
Created socket
R_WIFI_DA16XXX_TcpConnect()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

Figure 1-3 Status transitions.

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 module:

r_bsp
r_sci_rx
r_byteq_rx
FreeRTOS

2.3. Support Toolchain

The FIT module has been confirmed to work with the toolchain listed in 5.1 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	
WIFI_CFG_DA16600_SUPPORT Note: The default is 0	Use DA16600 module
WIFI_CFG_SCI_CHANNEL Note: The default is 0	SCI Channel number for DA16XXX Initial Command Port for AT command communication
WIFI_CFG_SCI_INTERRUPT_LEVEL Note: The default is 4	Interrupt Level for WIFI_CFG_SCI_CHANNEL
WIFI_CFG_SCI_PCLK_HZ Note: The default is 60000000	Peripheral clock speed for WIFI_CFG_SCI_CHANNEL
WIFI_CFG_SCI_BAUDRATE Note: The default is 115200	Communication baud rate for WIFI_CFG_SCI_CHANNEL
WIFI_CFG_CTS_SW_CTRL Note: The default is 1	UART hardware flow control
WIFI_CFG_CTS_PORT Note: The default is 2	Port of CTS
WIFI_CFG_CTS_PIN Note: The default is 3	Pin of CTS
WIFI_CFG_RTS_PORT Note: The default is 2	Port of RTS
WIFI_CFG_RTS_PIN Note: The default is 3	Pin of RTS
WIFI_CFG_PFS_SET_VALUE Note: The default is 0x0BU	Set value for PFS
WIFI_CFG_RESET_PORT Note: The default is 1	General-purpose port PDR register connected to the DA16XXX EN pin
WIFI_CFG_RESET_PIN Note: The default is 7	General-purpose port PODR register connected to the DA16XXX EN pin
WIFI_CFG_CREATABLE_SOCKETS Note: The default is 4	Creatable Sockets number
WIFI_CFG_SOCKETS_RECEIVE_BUFFER_SIZE Note: The default is 8192	Socket Receive buffer size
WIFI_CFG_AT_CMD_TX_BUFFER_SIZE Note: The default is 1500	AT command transfer buffer size
WIFI_CFG_AT_CMD_RX_BUFFER_SIZE Note: The default is 3000	AT command receive buffer size
WIFI_CFG_USE_CALLBACK_FUNCTION Note: The default is 0	Callback function use
WIFI_CFG_CALLBACK_FUNCTION_NAME Note: The default is NULL	Callback function name
WIFI_CFG_COUNTRY_CODE Note: The default is VN	Country code
WIFI_CFG_MAX_SSID_LEN Note: The default is 32	Max SSID Length
WIFI_CFG_MAX_BSSID_LEN Note: The default is 6	Max BSSID Length
WIFI_CFG_MAX_PASS_LEN	Max password Length
WIFI_CFG_SNTP_ENABLE Note: The default is 0	Use SNTP client service
WIFI_CFG_SNTP_SERVER_IP Note: The default is 0.0.0.0	The SNTP server IP address string
WIFI_CFG_SNTP_UTC_OFFSET Note: The default is 7	Time zone offset in hours (-12 ~ 12)
WIFI_CFG_USE_FREERTOS_LOGGING Note: The default is 0	Enable to use FreeRTOS logging
WIFI_CFG_DEBUG_LOG Note: The default is 4	Enable debug log

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 2048.
#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 2048.
#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_byteq_config.h)

Configuration Options in r_ byteq_config.h	
#define BYTEQ_CFG_MAX_CTRL_BLKS	Add the value specified by WIFI_CFG_CREATABLE_SOCKETS.

Table 2-4 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 realtime OS. When using this FIT module, set the following. FreeRTOS:1

2.8. Code Size

Typical code sizes associated with this module are listed below.
The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7 Compile Settings. The table lists reference values when the C compiler's compile options are set to their default values, as described in 2.3 Support 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.

ROM, RAM and Stack Code Sizes			
Device	Category	Memory usage	Remarks
		Renesas Compiler	
RX65N	ROM	27797 bytes	-
	RAM	7402 bytes	The size excluding the socket buffer (socket buffer size * number of sockets).
	Stack size	836 bytes	-

2.9. Return values

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

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

/* Error event for user callback */
typedef enum
{
    WIFI_EVENT_WIFI_REBOOT = 0,       // reboot WIFI
    WIFI_EVENT_WIFI_DISCONNECT,       // disconnected WIFI
    WIFI_EVENT_SERIAL_OVF_ERR,        // serial : overflow error
    WIFI_EVENT_SERIAL_FLM_ERR,        // serial : flaming error
    WIFI_EVENT_SERIAL_RXQ_OVF_ERR,    // serial : receiving queue overflow
    WIFI_EVENT_RCV_TASK_RXB_OVF_ERR,  // receiving task : receive buffer
overflow
    WIFI_EVENT_SOCKET_CLOSED,         // socket is closed
    WIFI_EVENT_SOCKET_RXQ_OVF_ERR     // socket : receiving queue overflow
} wifi_err_event_enum_t;
```

2.10. Parameter

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

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

/* Socket type */
typedef enum
{
    DA16XXX_SOCKET_TYPE_TCP_SERVER = 0, // TCP server
    DA16XXX_SOCKET_TYPE_TCP_CLIENT,     // TCP client
    DA16XXX_SOCKET_TYPE_UDP             // UDP
} da16xxx_socket_type_t;

/* Query current socket status */
typedef enum
{
    DA16XXX_SOCKET_STATUS_CLOSED = 0,   // "CLOSED"
    DA16XXX_SOCKET_STATUS_SOCKET,       // "SOCKET"
    DA16XXX_SOCKET_STATUS_BOUND,        // "BOUND"
    DA16XXX_SOCKET_STATUS_LISTEN,       // "LISTEN"
    DA16XXX_SOCKET_STATUS_CONNECTED     // "CONNECTED"
} da16xxx_socket_status_t;

/** Socket receive state */
typedef enum
{
    DA16XXX_RECV_PREFIX,         // +
    DA16XXX_RECV_CMD,            // command
    DA16XXX_RECV_SUFFIX,         // :
    DA16XXX_RECV_PARAM_CID,      // cid parameter
    DA16XXX_RECV_PARAM_IP,       // ip parameter
    DA16XXX_RECV_PARAM_PORT,     // port parameter
    DA16XXX_RECV_PARAM_LEN,      // length parameter
    DA16XXX_RECV_DATA
} da16xxx_recv_state_t;

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

```
typedef struct
{
    wifi_err_event_enum_t event;      // Error event
    uint8_t socket_number;            // Socket number
} wifi_err_event_t;

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

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

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) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) for RX devices that are not supported by the Smart Configurator.

1) Adding the FIT module to your project using the Smart Configurator in e2 studio. By using the Smart Configurator in e2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e2 studio (R20AN0451)” for details

2) Adding the FIT module to your project using the FIT Configurator in e2 studio. By using the FIT Configurator in e2 studio, the FIT module is automatically added to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

2.12. RTOS Usage Requirement

The FIT module utilizes RTOS functionality.

2.13. 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_TAKE_MUTEX	Failed to obtain mutex
WIFI_ERR_SERIAL_OPEN	Failed to initialize serial
WIFI_ERR_SOCKET_BYTEQ	BYTEQ allocation failure
WIFI_ERR_ALREADY_OPEN	Already open
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 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_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

3.3. R_WIFI_DA16XXX_Ping()

This function pings the specified IP address.

Format

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

Parameters

None

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex
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 pings the IP address specified by ip_address.

The parameter (count) specifies the number of transmissions.

Reentrant

No

Example

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

Special Notes:

None

3.4. R_WIFI_DA16XXX_Scan()

This function scans for access points.

Format

wifi_err_t R_WIFI_DA16XXX_Scan(
    wifi_scan_result_t * ap_results,
    uint32_t max_networks
)

Parameters

None

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 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.

In addition, the number of access points detected is reported in exist_ap_count.

Example

```
wifi_scan_result_t scan_rslt[5];
uint32_t max_networks = 5;
uint32_t exist_ap_count;
uint32_t max_ap;
R_WIFI_DA16XXX_Scan(scan_rslt, max_networks, &exist_ap_count);
printf(“Found access point(s) : %d\n”, exist_ap_count);
if (exist_ap_count >= max_networks)
{
    max_ap = max_networks;
}
else
{
    max_ap = exist_ap_count;
}
for (int i = 0; i < max_ap; i++)
{
    printf(“ --------------------\n”);
    printf(“ ssid : %s\n”, p[i].ssid);
    printf(“ rssi : %d\n”, p[i].rssi);
    printf(“ security : %d\n”, p[i].security);
    printf(“ encryption : %d\n”, p[i].encryption);
}
```

Special Notes:

None

3.5. 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_NOT_OPEN	Wi-Fi module not initialized
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_TAKE_MUTEX	Failed to obtain mutex
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module

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

3.6. 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_TAKE_MUTEX	Failed to obtain mutex

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

3.7. 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

3.8. R_WIFI_DA16XXX_DnsQuery()

This function performs a DNS query.

Format

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

Parameters

*domain_name	Domain name
*ip_address	IP address storage area

Return values

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

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

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

Reentrant

No

Example

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

Special Notes:

None

3.9. R_WIFI_DA16XXX_SntpServerIpAddressSet()

This function sets SNTP server IP address.

Format

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

Parameters

*ip_address	IP address storage area

Return values

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

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function sets SNTP server IP address.

Reentrant

No

Example

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

Special Notes:

None

3.10. R_WIFI_DA16XXX_SntpEnableSet()

This function enables or disables SNTP client service.

Format

wifi_err_t R_WIFI_DA16XXX_SntpEnableSet(

    wifi_sntp_enable * enable

)

Parameters

*ip_address	IP address storage area

Return values

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

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

3.11. R_WIFI_DA16XXX_SntpTimeZoneSet()

This function sets SNTP time zone.

Format

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

Parameters

Utc_offset_in_hour	time zone

Return values

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

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(7);  /* UTC+07:00 */
```

Special Notes:

None

3.12. R_WIFI_DA16XXX_LocalTimeGet()

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

Format

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

Parameters

local_time	time zone
size_string	size of string

Return values

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

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

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

Reentrant

No

Example

```
uint8_t time[20];
R_WIFI_DA16XXX_LocalTimeGet(time, 20);
printf(“It is %s\n”, time);
```

Special Notes:

None

3.13. 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_TAKE_MUTEX	Failed to obtain mutex
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module or domain does not exist

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

3.14. R_WIFI_DA16XXX_GetMacAddress()

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

Format

wifi_err_t R_WIFI_DA16XXX_GetMacAddress(
    uint8_t * mac_address
)

Parameters

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

Return values

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

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

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

Reentrant

No

Example

```
uint8_t mac[6];
R_WIFI_DA16XXX_Open();
R_WIFI_ DA16XXX_GetMacAddress(mac);
printf(“– MAC addr : %lx:%lx:%lx:%lx:%lx:%lx\r\n”,
mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
```

Special Notes:

None

3.15. 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_TAKE_MUTEX	Failed to obtain mutex
WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module or domain does not exist

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

3.16. R_WIFI_DA16XXX_GetAvailableSocket()

This function gets the next available socket ID.

Format

wifi_err_t R_WIFI_DA16XXX_GetAvailableSocket(
    uint32_t * socket_id
)

Parameters

* socket_id	Pointer to socket id storage area

Return values

WIFI_SUCCESS	Normal end
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Failed to count socket

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets the next available socket ID.

Reentrant

No

Example

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

Special Notes:

None

3.17. R_WIFI_DA16XXX_GetSocketStatus()

This function gets socket status.

Format

```
wifi_err_t R_WIFI_DA16XXX_GetSocketStatus(
        uint32_t socket_number,
        da16xxx_socket_status_t *socket_status
)
```

Parameters

socket_number	Pointer to socket number storage area
* socket_status	Pointer to socket status storage area

Return values

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

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

This function gets socket status.

Reentrant

No

Example

```
if (DA16XXX_SOCKET_STATUS_CLOSED == R_WIFI_DA16XXX_GetSocketStatus())
{
    printf(“Socket is available \n”);
}
else
{
    printf(“Socket is not available \n”);
}
```

Special Notes:

None

3.18. R_WIFI_DA16XXX_CreateSocket()

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

Format

wifi_err_t R_WIFI_DA16XXX_CreateSocket(

uint32_t socket_number,

da16xxx_socket_type_t *type,

uint8_t ip_version

)

Parameters

socket_number	Pointer to socket number storage area
* 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 returns the number of the created socket as an integer value.

Reentrant

No

Example

```
int32_t socket_no;
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);
```

Special Notes:

None

3.19. R_WIFI_DA16XXX_TcpConnect()

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

Format

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

Parameters

socket_number	Pointer to socket number storage area
* ip_address	IP address of TCP server
port	Port of TCP server

Return values

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

Properties

Prototype declarations are contained in r_wifi_da16xxx_if.h.

Description

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

Reentrant

No

Example

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

Special Notes:

None

3.20. R_WIFI_DA16XXX_SendSocket()

This function transmits data using the specified socket.

Format

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

Parameters

socket_number	Pointer to socket number storage area
* data	Pointer to transmit data storage area
length	Number of bytes of data to be transmitted
timeout_ms	Transmission timeout duration (ms)

Return values

WIFI_ERR_MODULE_COM	Failed to communicate with Wi-Fi module or domain does not exist
WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Failed to create socket
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

3.21. R_WIFI_DA16XXX_ReceiveSocket()

This function receives data from the specified socket.

Format

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

Parameters

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

Return values

WIFI_ERR_PARAMETER	Invalid argument
WIFI_ERR_NOT_CONNECT	Not connected to access point
WIFI_ERR_SOCKET_NUM	Failed to create socket
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_ReceiveSocket(sock, buffer, sizeof(buffer), 1000);
```

Special Notes:

None

3.22. R_WIFI_DA16XXX_CloseSocket()

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

Format

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

Parameters

socket_number	Socket number

Return values

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

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

4. Callback Function

4.1. callback()

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.

5. Appendix

5.1. Confirmed Operation Environment

This section describes confirmed operation environment for the FIT module.

Table 5.1 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 5.2 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)

5.2. Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got an 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 document:

For e2 studio, Application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)".

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

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

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.

(3) Q: The pin setting is supposed to be done, but it doesn’t look like that.

A: The pin setting may not be performed correctly. When using this FIT module, the pin setting must be performed. Refer to 2.7 Compile Settings for details.

6. 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. 21, 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
		10	Updated code size in 2.8. Code Size
		26	Updated example in 3.11. R_WIFI_DA16XXX_SntpTimeZoneSet()
		27	Updated format in 3.12. R_WIFI_DA16XXX_LocalTimeGet()
		40	Added table 5-2 Confirmed Operation Environment (Ver. 1.10)

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

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/.

Trademarks

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