RS9116W SAPI Programming Reference Manual

Version 2.0

10/21/2020

Table of Contents

About this Document

This document provides complete details on the RS9116 WiSeConnect SAPI (Simple API) library, including:

- briefs about the RS9116 WiSeConnect SAPI Architecture.
- details of the APIs and configurations available in the SAPI library.
- details of the new changes in the SAPI library in conjunction with previous releases.

> **Note:**
> This information should be considered by users/developers when upgrading to latest release.

1 Overview of Wireless SAPI and Features

The RS9116W SAPI library is a comprehensive collection of Wireless, Network Applications, BSD Socket APIs, and RS9116 driver code along with different HALs for mapping it to platform interface on which this library will be ported.

The SAPI is intended to run on host machine, it can be an MCU with/without RTOS or any Linux machine. Users are recommended to use the given APIs without any modification in order to facilitate upgrading to the future releases. Users are also recommended to upgrade SAPI along with the firmware.

The RS9116 WiSeConnect module includes Wi-Fi, TCP/IP Networking stack with SSL 3.0/TLS 1.2, HTTP/HTTPS, Websockets, DHCP, MQTT client, and BT 5 stacks embedded. This module requires a separate application processor, which acts as a host.

The host can communicate with the RS9116 module using one of the interfaces listed below.

- SPI
- UART
- USB
- SDIO

Note:

USB and SDIO interfaces are currently not supported.

SAPI enables easy migration onto any platform with its uniform APIs. This library simplifies application development on host. Users can develop application firmware without having to learn the underlying peripheral register interface and other details.

Note:

RS9116W release consists of 2 main components

- Firmware
- SAPI Library

Both components will have same revision number.

Latest releases might have bug-fixes, enhancements, and new features in SAPIs and Firmware together. Most of the new features will have associated APIs, which are available in the latest SAPIs only. Hence, it is recommended to upgrade SAPI and Firmware together.

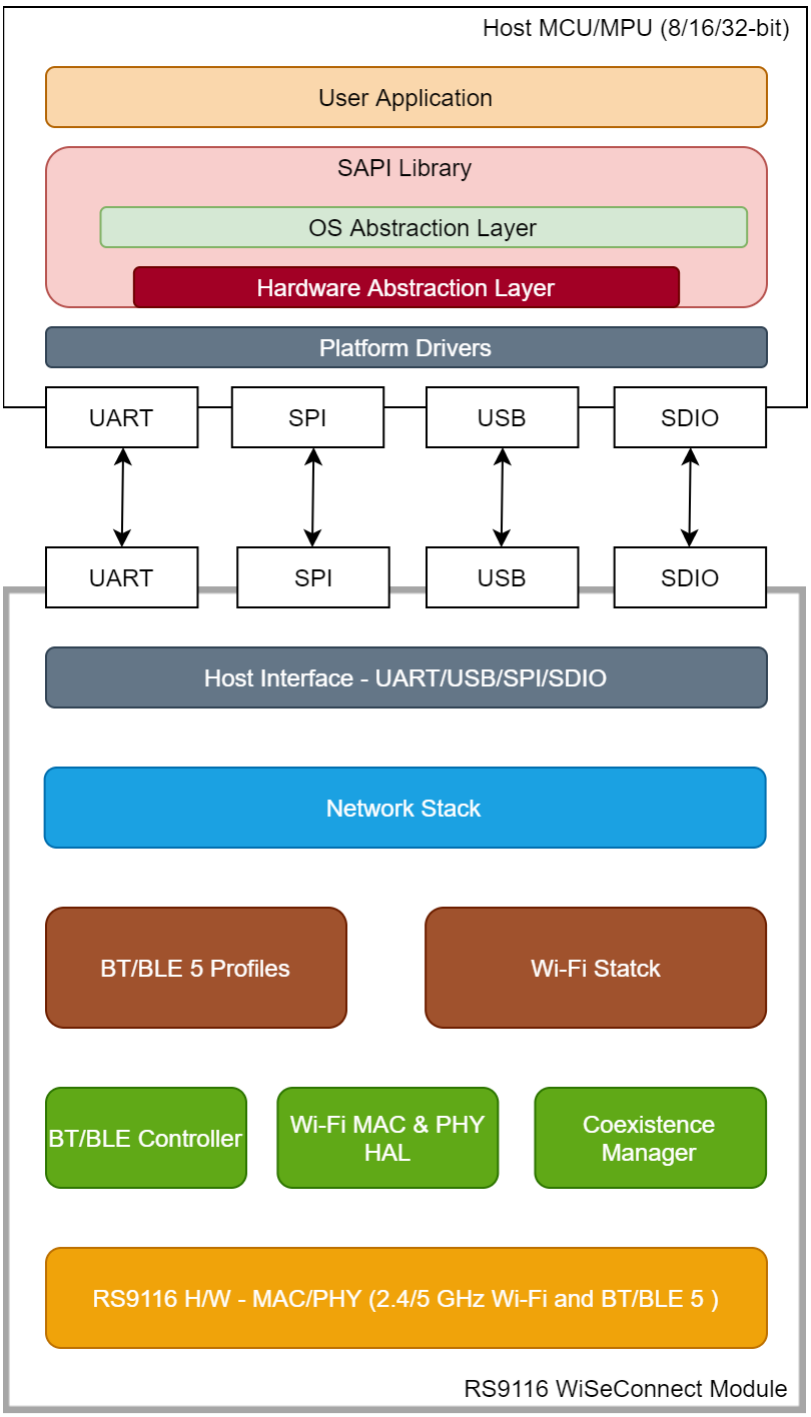


Figure 1: RS9116W Block Diagram with SAPIs

SAPI Features

- Platform-independent, interrupt-driven drivers written in C.
- Drivers provide a simpler, functional interface and eliminate the need to manage the low-level host interface protocol.
- Common APIs for four host interfaces (SPI, USB, UART, SDIO, USB-CDC), which enables easy migration to different host interfaces.
- Supports bare-metal and FreeRTOS out-of-box. Other RTOS can be supported through OS Abstraction changes.
- Supports Keil µvision, WiSeStudio and IAR IDEs – can be ported to other IDEs.

2 RS9116W Features

Wi-Fi

- Supports Wi-Fi Station, Access Point features and Concurrent mode (STA + AP) features.

Wi-Fi Station

- **Securities**: Open mode, WPA2, Mixed mode, EAP, TLS1.2, TLS1.0, TTLS, PEAP, WPS 2.0 push button.
- Supports auto rate, dual band, re-join, 802.11j, 802.11d, Selective Scan, RSSI support.
- Supports BSD socket interface.
- APIS for crypto hardware ECDH, DH, SHA, AES.

Access Point

- Supports WPA2 mode and hidden SSID mode.

Firmware Upgrade Options

- via bootloader, Host/MCU and Wireless (OTA).

Network Stack

- IPV4,ICMP,ARP,UDP,TCP,IGMP,DHCP Client, DHCP Server, HTTP Client, HTTPs Client, HTTP Server, ping from host, TCP/IP bypass, FTP Client, SSL3.0/TLS1.2(Client and server),SNTP,SNMP,MQTT Client, Websockets Client.
- Dynamic feature Enable/Disable, through opermode.

BT Features

- Master, Slave, BT Dual mode and Role switch support.
- BDR1, BDR2, BDR3 etc.
- ACL Data, Encryption, Slot offset, Adaptive frequency hopping, AFH channel classification, Enhanced data rate ACL, simple pairing, Host Controller Interface (HCI).
- Role switch (Master/Slave), sniff mode, RSSI
- SSP support?
- Profiles: GAP, SDP, L2CAP, SPP, RFCOMM, HID (only in AT commands).

BLE Features

- Master, Slave, BT Dual mode support.
- BT smart (Peripheral), BT Smart ready (Master), Multiple slave support.
- Advertising, Connection interval.
- Host controller interface
- 32-bit UUID support in LE, LE secure connections, Data length extensions.
- LE 2Mbps, LE long range, LE channel classification.
- Profiles: GATT, GAP, BPM (Blood pressure monitor), PXP (proximity), HOGP HID over GATT
- BLP Blood pressure profile & service, HTS/HTP health thermometer profile & service, HRP/HRS Heart rate profile& service, CGMP/CGMS continuous glucose monitoring profile & service

Operating Modes

- WLAN STATION
- WLAN AP
- WLAN+ BLE
- WLAN + BT
- WLAN + BT + BLE

Power Save

- GPIO based power save, Message based power save
- Host based wakeup, WoWLAN, packet pending interrupt for UART.

3 Wireless SAPI Related Resources

The following table provides links for obtaining RS9116W SAPI Porting Guide and SAPI Application Examples documents.

Name	Silicon Lab's Document Portal links
RS9116W SAPI Porting Guide	https://docs.silabs.com/rs9116
AN1282: RS9116W Guide for SAPI Application Examples	https://docs.silabs.com/rs9116

4 SAPI Architecture

Architecture

SAPI APIs are designed in layers, where each Layer is independent and use service of underlying layers.

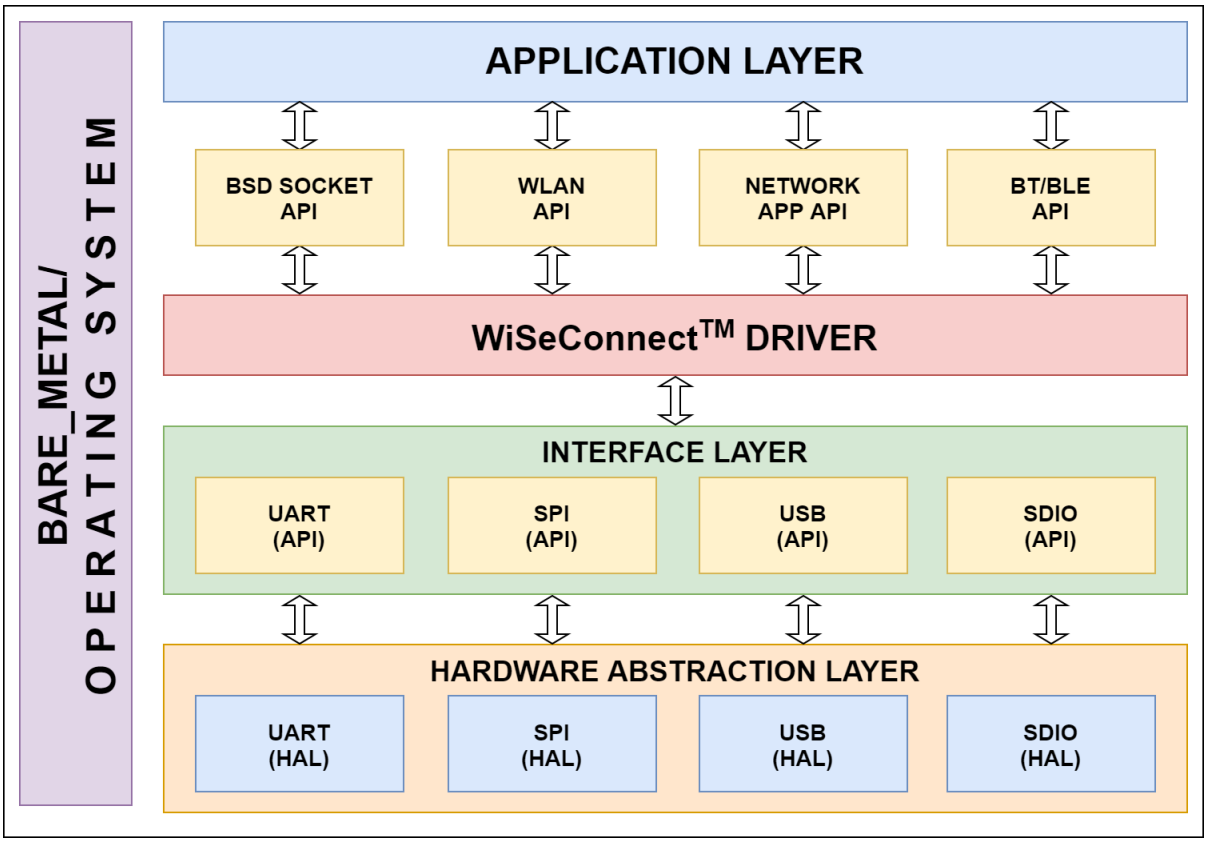


Figure 2: WiSeConnect API Architecture Diagram

Application Layer

Application layer contains application specific functionality. Application Layer call WiSeConnect Driver APIs to configure and operate the module.

BSD Socket API

This layer contains BSD Socket API compliant wrapper and supports some of the basic BSD Socket API calls. This APIs can be called from application to initialize and configure the embedded TCP/IP stack and perform data transfers.

WLAN API

This layer contains set of APIs called from application to initialize and configure Wi-Fi Module. User is recommended to use the given APIs without any modification in order to facilitate upgrading to future releases.

NETWORK API

This layer contains APIs related to network applications like HTTP/HTTPS, SSL 3.0/TLS 1.2, DHCP, MQTT, DNS, SNMP and WebSockets.

BT/BLE API

This layer contains set of APIs called from application to initialize and configure BT/BLE. User is recommended to use the given APIs without any modification in order to facilitate upgrading to future releases.

WiSeConnect Driver

WiSeConnect Driver software framework contains core functions for state machine maintenance, command preparation and command response parsing.

Interface Layer

The module supports 4 different host interfaces (SPI, USB, UART, SDIO, USB-CDC). These APIs are collection of functions specific to a particular interface. The interface functions between the Driver API Layer and the Interface

Specific API Layer are independent of the Host interface used. This allows the user to migrate to different interfaces without any change in the application layer.

HAL

Hardware Abstraction Layer APIs are platform specific APIs. The user needs to implement or modify these APIs to their platforms.

Reference Applications

WiSeConnect packages contain reference applications that operate the module in different modes. The user can use these applications as a reference or customize these applications as per their requirements.

SAPI Sample Flow

Below flow diagram shows sample API flow for Wi-Fi using SAPI library.

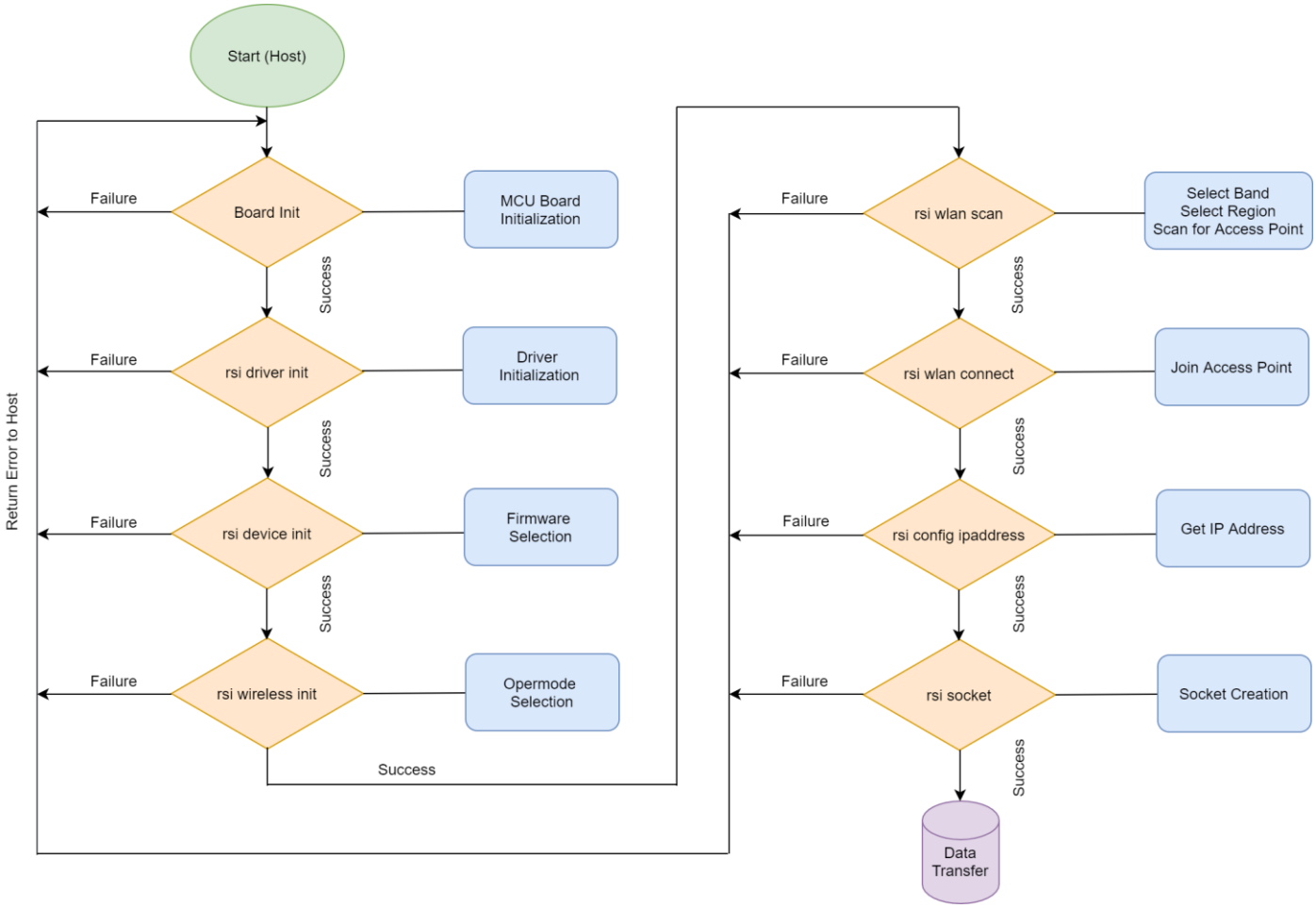


Figure 3: Wi-Fi SAPI Flow

5 SAPI Host Interfaces

RS9116 WiSeConnect Module supports SPI, USB, UART and SDIO for interfacing to host. This section describes each interface in detail including the supported features, protocols and commands.

Note:

USB and SDIO interfaces are currently not supported.

5.1 SPI Interface

This section describes RS9116-WiSeConnect SPI interface, including commands and processes to operate the module via SPI.

Features

The features are as follows:

- Supports 8-bit and 32-bit data mode.
- Supports flow control.

Hardware Interface

The SPI interface on the RS9116-WiSeConnect works in slave mode. It is a 4-wire interface. In addition to the SPI interface, the module provides an additional interrupt pin to signal events to the host.
The interrupt is raised by the module in SPI mode for the following condition.

- When the module has data in its output buffer, it indicates host by raising the active high signal on the interrupt pin.

The interrupt from the module is active high and the host has to configure the interrupt in level trigger mode.
The RS9116W also supports edge-triggered interrupts, provided the host is configured to handle these appropriately (This is generic and does not entail any specific implementations). The host must check for pending interrupts before clearing/acknowledging an interrupt to avoid missing interrupts and data.

Note:
By default, the interrupt from the module is active high but, it can be configured as active low as well (this can be configured from bootloader menu).

SPI Signals

SPI Interface is a full duplex serial Host interface, which supports 8-bit and 32-bit data mode. The SPI interface of the module consists of the following signals:

- SPI_MOSI (Input) - Serial data input for the module.
- SPI_MISO (Output) - Serial data output for the module.
- SPI_CS (Input) - Active low slave select signal. This should be low when SPI transactions are to be carried out.
- SPI_CLK (Input) - SPI clock. The maximum value allowed is 80 MHz.
- INTR (Output) - Active high (Default), Active low, level interrupt output from the module.

The module acts as an SPI slave only, while the Host is the SPI master. Following parameters should be in the host SPI interface.

- CPOL (clock polarity) = 0,
- CPHA (clock phase) = 0.

Interrupt

The module's INTERRUPT output signal should be connected to the interrupt input of the Host MCU. The INTERRUPT signal is an active high and level triggered signal. It is raised by the module in the following cases:

1. When the module needs to indicate to the Host that it has received data from the remote terminal and the data needs to be read by the Host.
2. When the module needs to indicate to the Host that a response to a command sent by the Host is ready to be read from the module.
3. To indicate to the Host that it should read a CARD READY message from the module. This operation is described in the subsequent sections.
4. An interrupt will be raised for asynchronous RX packets also.

SPI Interface Initialization

This section explains the initialization of the SPI slave interface in the module. Following is the series of steps for SPI initialization:

1. A host sends 0x00124A5C to the module.
2. On successful initialization, the module sends 0x58 (SPI success) to the host or else, the module sends 0x54 (SPI busy) or 0x52 (SPI failure).

Operations through SPI

RS9116-WiSeConnect can be configured and operated from the Host by sending commands through the SPI interface.
The SPI interface is programmed to perform a certain transfer using commands C1, C2, C3 and C4 and an optional 32-bit address. For all the Commands and Addresses, the Host is configured to transmit data with the 8-bit mode. At the end of all the Commands and Addresses, the Host is reconfigured to transmit data with 8-bit or 32-bit mode depending on the commands issued. The Module responds to all the commands with a certain response pattern. The four commands i.e. C1, C2, C3, and C4 indicate to the SPI interface for all the aspects of the transfer.

The command descriptions are as follows:

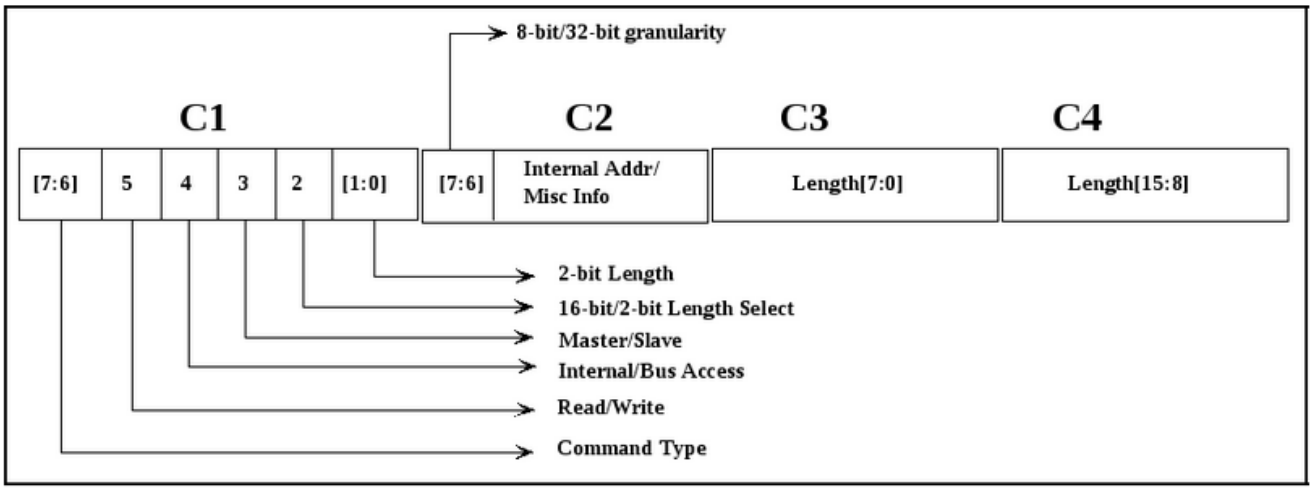


Figure 4: SPI Command Description

To all these commands, the SPI interface responds with a set of unique responses.

Module Response

RS9116 WiSeConnect gives responses to the host SPI command requests through the SPI interface. These are as follows:

- A success/failure response at the end of receiving the command. This response is driven with 8-bit mode during the Command and Address phase and is then switched to 8-bit or 32-bit mode during the Data phase as per the command issued.
    - Success: 0x58 or 0x00000058
    - Failure: 0x52 or 0x00000052
- An 8-bit or 32-bit start token is transmitted once the four commands (C1, C2, C3, C4) indicating a read request are received and the Module is ready to transmit data. The start token is immediately followed by the read data.
    - Start Token: 0x55 or 0x00000055

- An 8-bit or 32-bit busy response in case a new transaction is initiated while the previous transaction is still pending from the slave side.
    - Busy Response: 0x54 or 0x00000054

Module Bit Ordering of SPI Transmission / Reception

8-bit Mode:
If a sequence of bytes <B3 [7:0]> <B2 [7:0]> <B1[7:0]> <B0[7:0]> is to be sent, where B3 is interpreted as the most significant byte, then the sequence of transmission is as follows:

B0 [7] ..B0 [6] .. B0 [0] -> B1 [7] ..B1 [6] ..B1 [0] -> B2 [7] ..B2[6] .. B2[0] -> B3[7] ..B3[6] .. B3[0]

where, B0 is sent first, then B1, then B2 and so on.
In each of the bytes, the MSB is sent first. For example, when B0 is sent, B0 [7] is sent first, then B0 [6], then B0[5] and so on. Same is the case while receiving data. In this example, B0 [7] is expected first by the receiver, then B0 [6] and so on. Also, as can be seen, the data changes value at the falling edge and is latched at the rising edge.

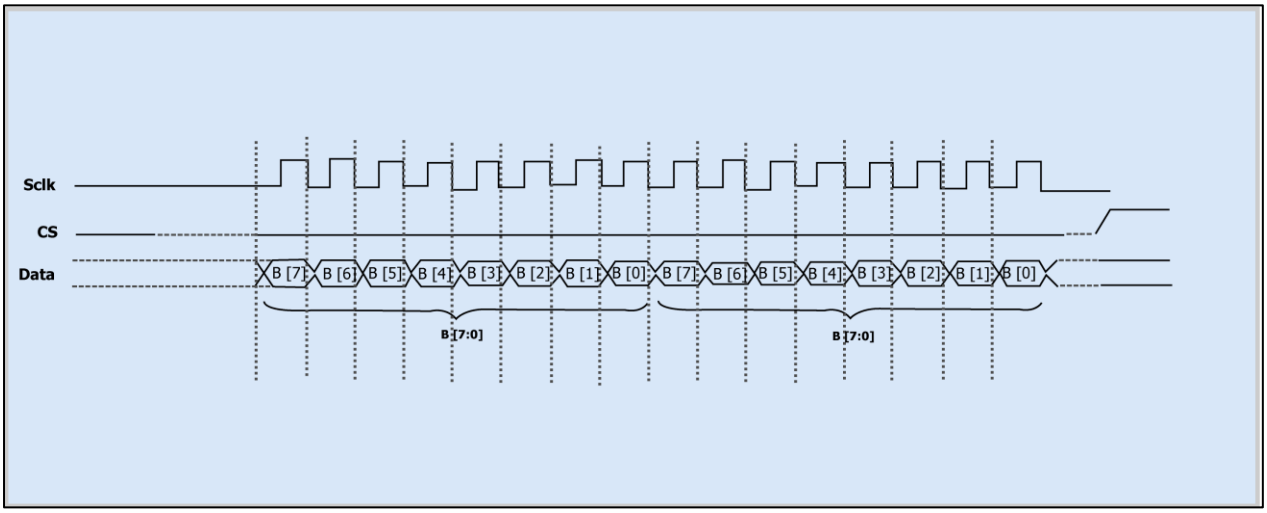


Figure 5 8-bit Mode

32-bit Mode:
If a sequence of 32-bit words is <W3[31:0]> <W2[31:0]> <W1[31:0]> <W0[31:0]> is to be sent, where W3 is interpreted as the most significant word, then the sequence of transmission is as follows:

W0[31] ..W0[30] ..W0[0] -> W1[31] ..W1[30] ..W1[0] -> W2[31] ..W2[30] ..W2[0] -> W3[31] ..W3[30] ..W3[0]

where, W0 is sent first, then W1, then W2 and so on.
In each of the 32-bit words, the MSB is sent first. For example, when W0 is sent, W0[31] is sent first, then W0[30], then W0[29] and so on. Same is the case when receiving data. In this example, W0 [31] is expected first by the receiver, then W0[30] and so on. Also, as can be seen, the data change value at the falling edge and is latched at the rising edge.

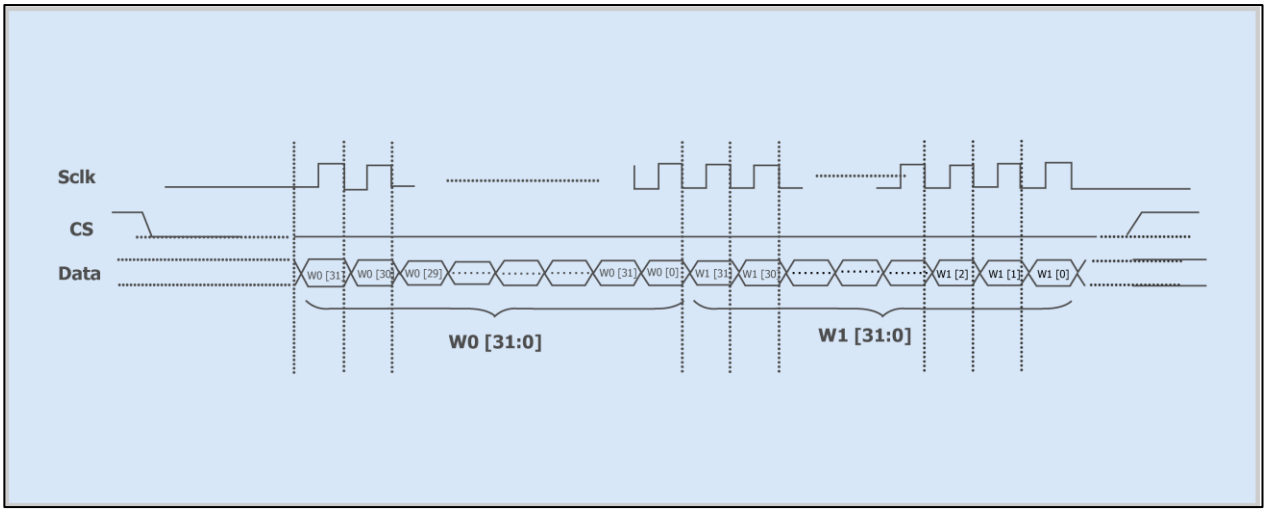


Figure 6 32-bit Mode

Bit Ordering of Module Response

The bit ordering is same as explained in **Module bit Ordering of SPI Transmission/Reception**. For example, a 0x58 response for 8-bit success is sent as
0->1->0-> 1->1-> 0 -> 0 -> 0. That is 0 is sent first, then 1, then 0, then 1, and so on.

Module SPI Interface Initialization

The Initialization command is given to the module to initialize the SPI interface. The SPI interface remains non-functional to all the commands before initialization and responds only after successful initialization. Initialization should be done only once after the power is on. The module treats the subsequent initialization of any command before it is reset as errors. SPI initialization is 24-bit command (0x124A5C) followed by an 8-bit dummy data. 24-bit SPI Initialization command should be sent in a sequence of 0x12, 0x4A, 0x5C bytes. Status response from the SPI Interface is driven during the transmission of the dummy data i.e. after the transfer of 8-bits of command C1.

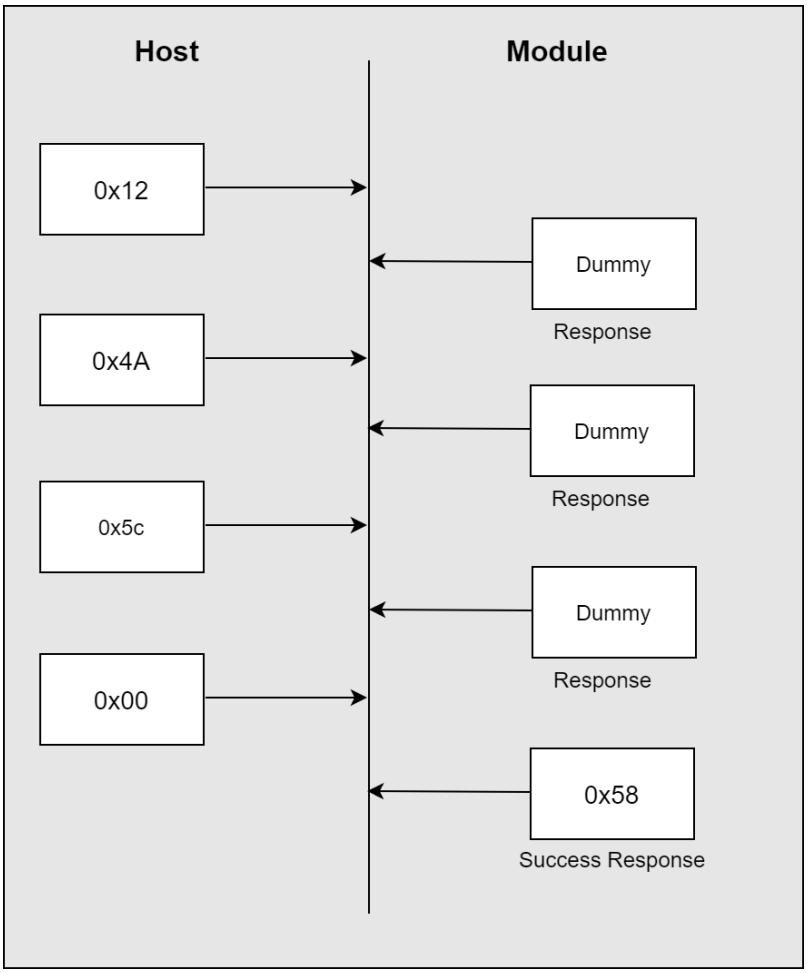


Figure 7: SPI Initialization exchanges between Host and Module

Host Interactions Using SPI Command

This section describes the procedures to be followed by the Host to interact with the RS9116 WiSeConnect using SPI commands. The Host interactions to the module can be categorized as below.

Table 1 SPI Command Description

Command	Bit Number	Description
C1	[7:6]	Command Type "00"- Initialization Command "01"- Read/Write Command "10", "11"- Reserved for future use
	5	Read/Write '0'- Read Command '1'- Write Command

Command	Bit Number	Description
	4	Register/(Memory & Frame) read/write Access '0'- register read/write '1'- Memory read/write or Frame read/write
	3	Memory/Frame Access '0'- Memory read/write '1'- Frame read/write
	2	2-bit or 16-bit length for the transfer '0'- 2-bit length for the transfer '1'- 16-bit length for the transfer
	1:0	2-bit length (in terms of bytes) for the transfer (valid only if bit 2 is cleared) "00"- 4 Bytes length "01"- 1 Byte length "10"- 2 Bytes length "11"- 3 Bytes length
C2	7:6	8-bit or 32-bit mode. Indicates the granularity of the write/read data. **Note:** The SPI (C1, C2, C3, C4) commands and addresses (A1, A2, A3, A4) will always be 8-bit irrespective of this value. "00"- 8-bit mode "01"- 32-bit mode "10", "11"- Reserved for future use
	5:0	This carries Register address, if bit 4 for Command C1 is cleared (i.e. register read/ write selected). Otherwise, reserved for future use.
C3	7:0	Length (7:0) LSB of the transfer's length (which is in terms of bytes) in case bit 2 of C1 is set. This command is skipped if bit 2 of C1 is cleared.
C4	7:0	MSB of the transfer Length (15:8) (Which is in terms of bytes) in case of bit 2 of C1 is set. This command is skipped if bit 2 of C1 is cleared i.e. if a 2-bit length is selected.

Memory Type:

The host needs to access the memory/registers of the RS9116 WiSeConnect for configuration and operation. To write data into a memory/register address, the **memory write** command must be framed as described in the figure below. If a "busy" or "failure" response is sent from the module, the host should either resend the command or reset the module.

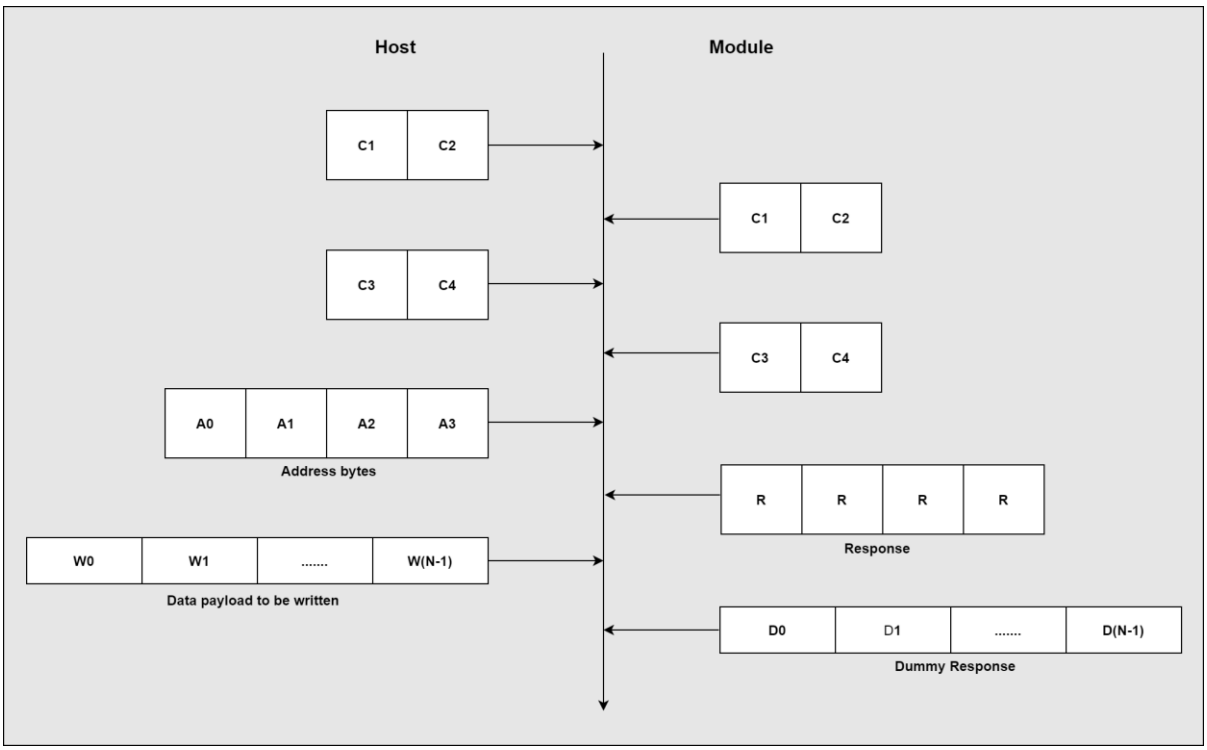


Figure 8: Memory Write

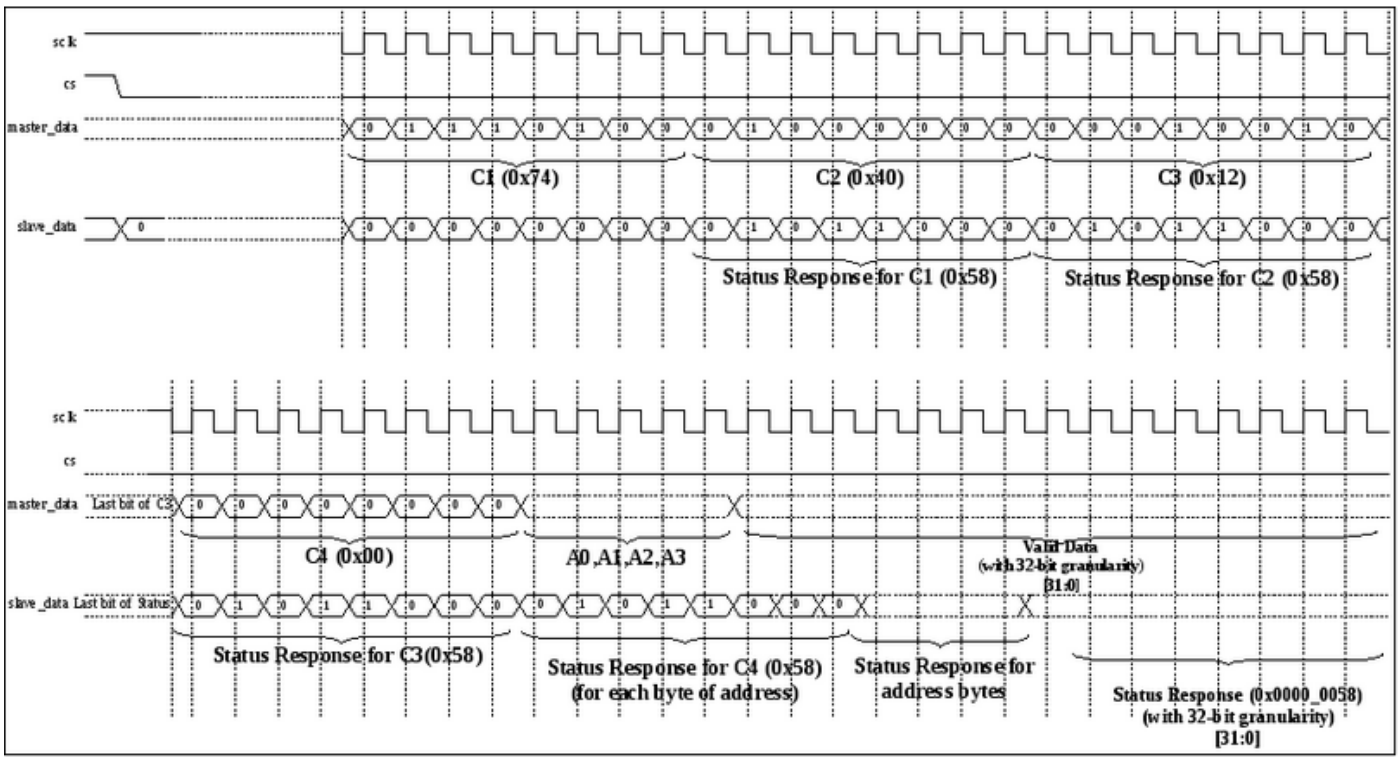


Figure 9: Interactions in the Physical Interface

Note:
This structure is similar for all following read/write operations.

The following procedure is to be followed by the Host.

1. Send the commands C1, C2.

2. Read the response (R) from the Module. The response will appear as described in **RS9116-WiSeConnect Module Response**. Status 0x58 indicates that the module is ready.
3. The host should send the commands C3 and C4, followed by the 4-byte address (corresponding to the memory/register address) and data. Status 0x54 indicates that the device is busy. The host has to retry. Status 0x52 indicates a failure response.

The commands C1, C2, C3 and C4 are sent in the following sequence (explained in Module bit Ordering of SPI Transmission / Reception) i.e. C1 first, then C2, C3 and finally C4.

The bit ordering is
C1[7] -> C1[6] … C1[0] -> C2[7] -> C2[6] … C2[0] -> C3[7] -> C3[6] … C3[0] -> C4[7] -> C4[6] … C4[0]. That is, C1[7] bit is sent first, then C1[6] and so on.

Total data payload size should be 1 byte, or 2 bytes or multiples of 4 bytes in this operation. For example, if 5 bytes need to be sent, then it should be padded with 3 more dummy bytes to make it 8 bytes (multiple of 4) before sending.

Original data **<W0[7:0]> <W1[7:0]> <W2[7:0]> <W3[7:0]> <W4[7:0]>**
Padded data **<W0[7:0]> <W1[7:0]> <W2[7:0]> <W3[7:0]> <W4[7:0]> <D5[7:0]><D6[7:0]><D7[7:0]>** , where D[7:0] is the dummy data.

The first byte sent is W0, the second byte sent is W1 and so on (refer **Module bit Ordering of SPI Transmission/Reception**).

Frame Write

The sequence of command transactions (with no failure from the Module) for a Frame Write is as described in the figure below. The operations are similar to memory write – except that bit 3 of C1 is set and the Address phase (A0, A1, A2, A3) is skipped.

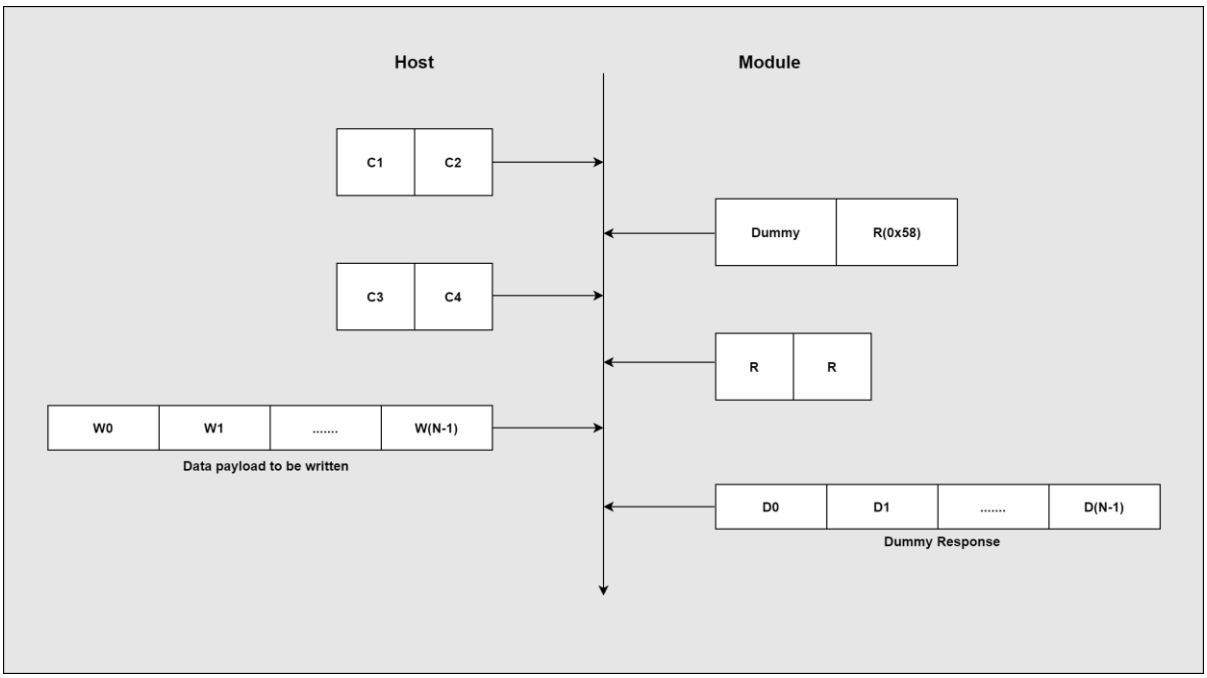


Figure 10: Frame Write

The following is the procedure to be followed by the Host.

1. Prepare and send the commands C1, C2 together as described for Frame write.
2. Read the response (R) from the Module (RS9116-WiSeConnect).
3. Status 0x58 indicates that the Module is ready. The Host can then send the commands C3 and C4, followed by the data. Status 0x54 indicates that the device is busy. The host must retry. Status 0x52 indicates a failure response.

Data payload size should be in multiples of 4 bytes in this operation. For example, if 5 bytes of data is to be written, it should be padded with 3 more dummy bytes to make it 8 bytes (multiple of 4) before sending.

Original data **<W4[7:0]> <W3[7:0]> <W2[7:0]> <W1[7:0]> <W0[7:0]>**
Padded data **<D7[7:0]> <D6[7:0]> <D5[7:0]> <W4[7:0]> <W3[7:0]> <W2[7:0]> <W1[7:0]> <W0[7:0]>** , where D[7:0] is the dummy data.

The first byte sent is W0, the second byte sent is W1 and so on (refer **Module bit Ordering of SPI Transmission/Reception**).

Memory Read

To read data from a memory/register address, the host must form and send a Memory Read command. The following figure gives the flow for memory reads between the Host and the Module.

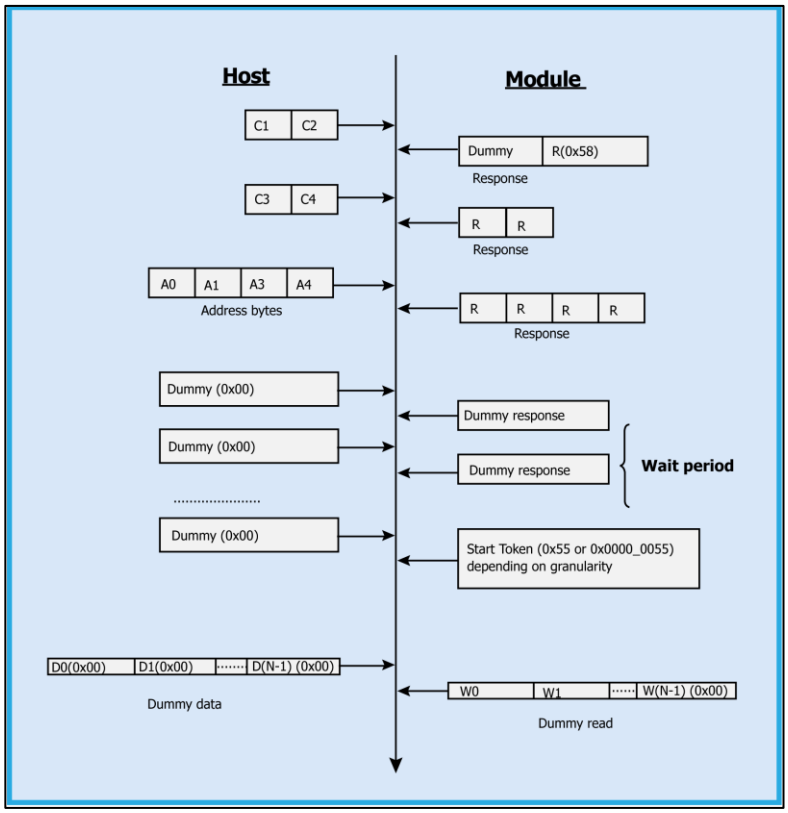


Figure 11: Memory Read

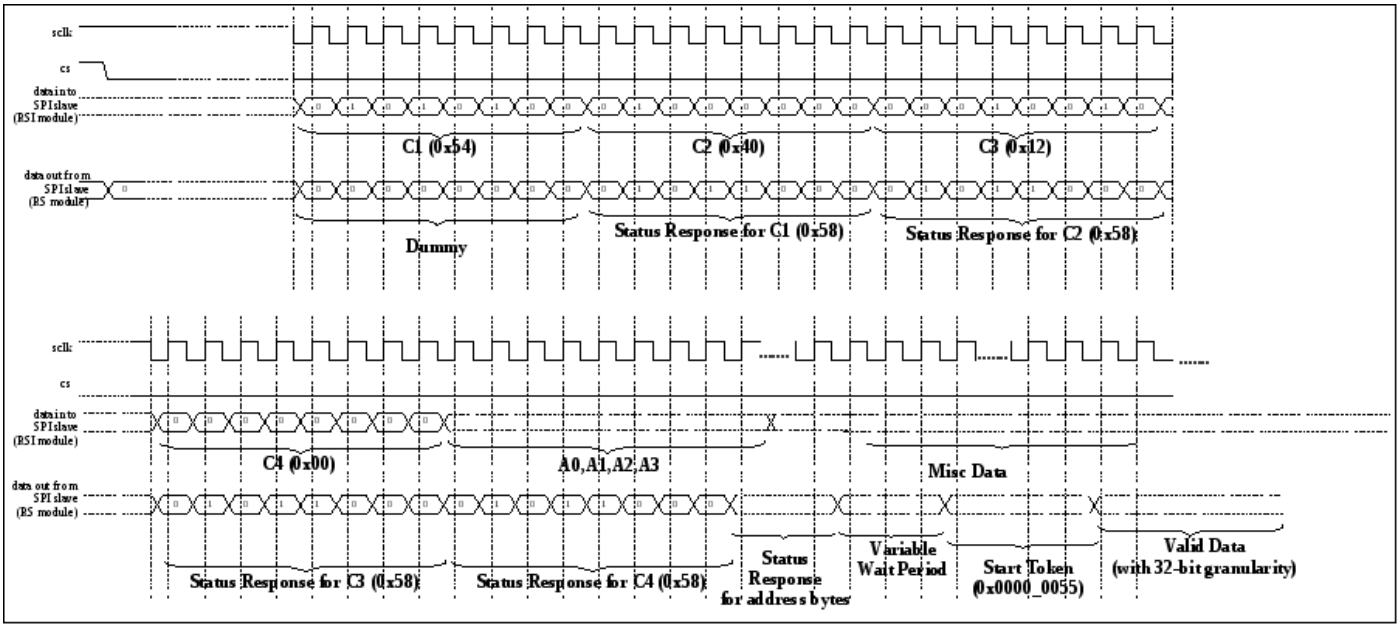


Figure 12: Memory Read at Physical Interface

To perform a memory read, the host must follow the procedure below.

1. Prepare and send the commands C1, C2 as described for Memory read.
2. Read the response from the RS9116-WiSeConnect.

3. Status 0x58 indicates that the slave is ready. The host should next send the commands C3, C4, which indicate the length of the data to be read, followed by the address of the memory location to be read.
4. After sending/receiving C3, C4 commands/response and the addresses, the host should wait for a start token (0x55). The host writes a stream of dummy bytes to read the data from the module each time. The Host should read this data until the start token is received. The data following the start token should be interpreted as the frame of a specified length that is read from the Module.
5. Status 0x54, after C1 and C2 bytes, indicates that the device is busy. The host has to retry.
6. Status 0x52, after C1 and C2 bytes, indicates a failure response from the Module.

There is a variable wait period, during which dummy data is sent. After this period, a start token is transmitted from the Module to the Host. The start token is followed by valid data. The start token indicates the beginning of valid data to the Host. To read out the valid data, the host must send dummy data i.e. [D0, D1, D2, …. D (N-1)]. Where, N is the number of 8bit or 32-bit dummy words (based on the mode selected) need to send in order to receive a specified length of valid data from the module.
The commands C1, C2, C3 and C4 are sent in the following sequence (explained in Module bit Ordering of SPI Transmission/Reception) : C1 first, then C2, C3 and finally C4.

The bit ordering is
C1[7] -> C1[6] … C1[0] -> C2[7] -> C2[6] … C2[0] -> C3[7] -> C3[6] … C3[0] -> C4[7] -> C4[6] … C4[0]. That is, C1[7] bit is sent first, then C1[6] and so on.

If <D3[7:0]> <D2[7:0]> <D1[7:0]> <D0[7:0]> is the data read, then D0 is sent from module first, then D1 and so on.

D0[7] -> D0[6] … D0[0] -> D1[7] -> D1[6] … D1[0] -> D2[7] -> D2[6] … D2[0] -> D3[7] -> D3[6] … D3[0]

Frame Read

This is same as Memory read, except that bit 3 of C1 is set and the Address phase is skipped. The sequence of command transactions (with no failure from the Module) for a Frame Read is as described in the figure below.

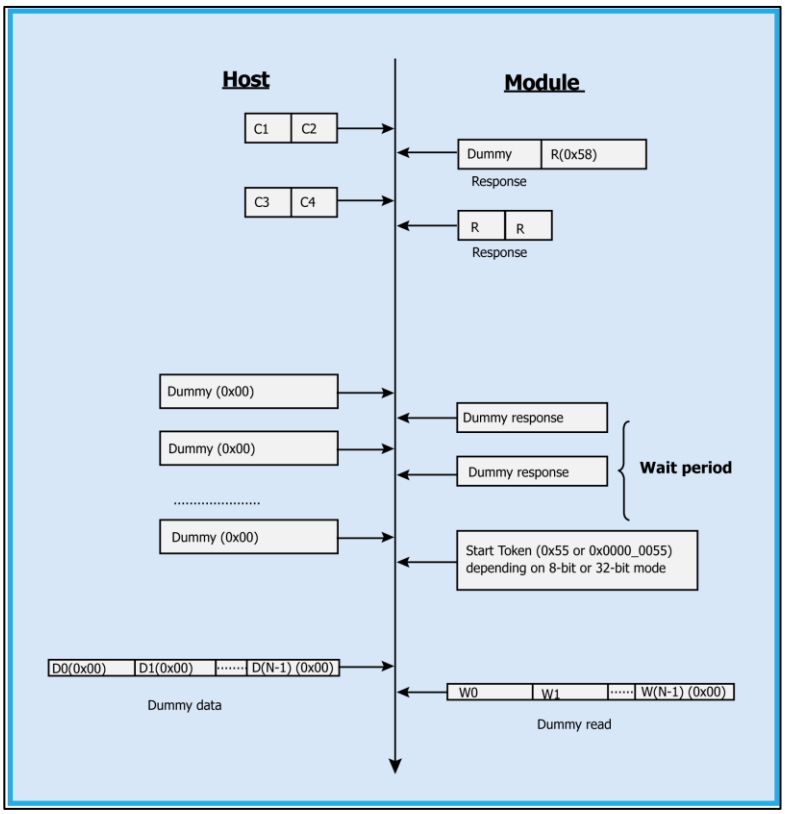


Figure 13: Frame Read

To perform a Frame Read, the host must follow the procedure below.

1. Prepare and send the commands C1, C2 as described for Frame Read.
2. Read the response from the RS9116-WiSeConnect.
3. Status 0x58 indicates that the Module is ready. The host should send the commands C3, C4 which indicate the length of the data to be read.

4. After sending/receiving C3, C4 commands/response, the host should wait for a start token (0x55). The data then follows the start token. The host writes a dummy byte to read the data from the module each time. The Host should read this data until the start token is received. The data following the start token should be interpreted as the frame of a specified length that is read from the Module. Status 0x54, after C1 and C2 bytes, indicates that the device is busy. The host must retry. Status 0x52, after C1 and C2 bytes, indicates a failure response from the module.

The commands C1, C2, C3 and C4 are sent in the following sequence (explained in Module bit Ordering of SPI Transmission/Reception): C1 first, then C2, C3 and finally C4.

The bit ordering is
C1[7] -> C1[6] … C1[0] -> C2[7] -> C2[6] … C2[0] -> C3[7] -> C3[6] … C3[0] -> C4[7] -> C4[6] … C4[0]. That is, C1[7] bit is sent first, then C1[6] and so on.
If <D3 [7:0]> <D2[7:0]> <D1[7:0]> <D0[7:0]> is the data read, then D0 is sent from module first, then D1 and so on.
D0[7] -> D0[6] … D0[0] -> D1[7] -> D1[6] … D1[0] -> D2[7] -> D2[6] … D2[0] -> D3[7] -> D3[6] … D3[0]

Register Read

Register Read commands are used to read the registers in the module using internal register address. Register read commands like C1 and C2 are only needed to send to the module (i.e., C3, C4, and address phases are skipped). The C2 command bits [5:0] should be the SPI internal address of the register.
For Register Reads following sequence needs to be followed:

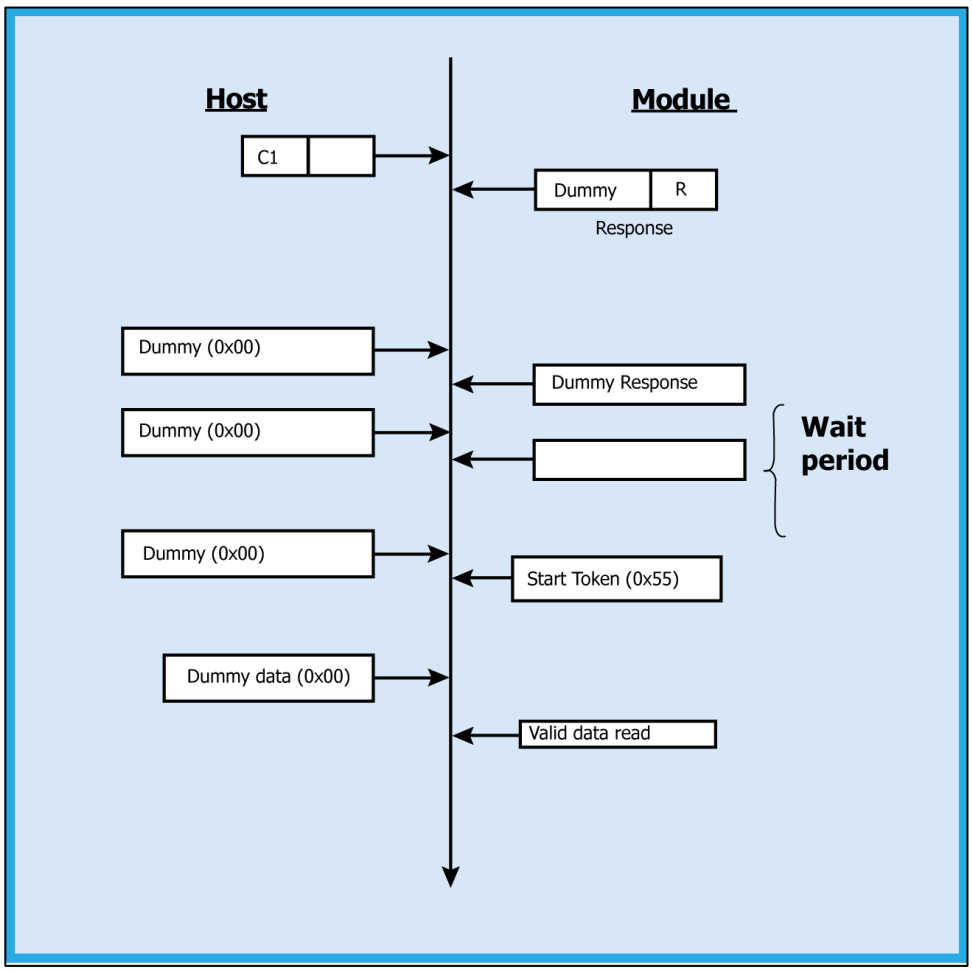


Figure 14: Register Read

Register Writes

Register write commands are used to write the data to the registers in the module by using internal register address. To Register write, the host needs to send C1 and C2 followed by the data to the module (i.e., C3, C4, and address phases are skipped). The C2 command bits [5:0] should be the SPI internal address of the register.
For Register writes, follow the below sequence as shown below:

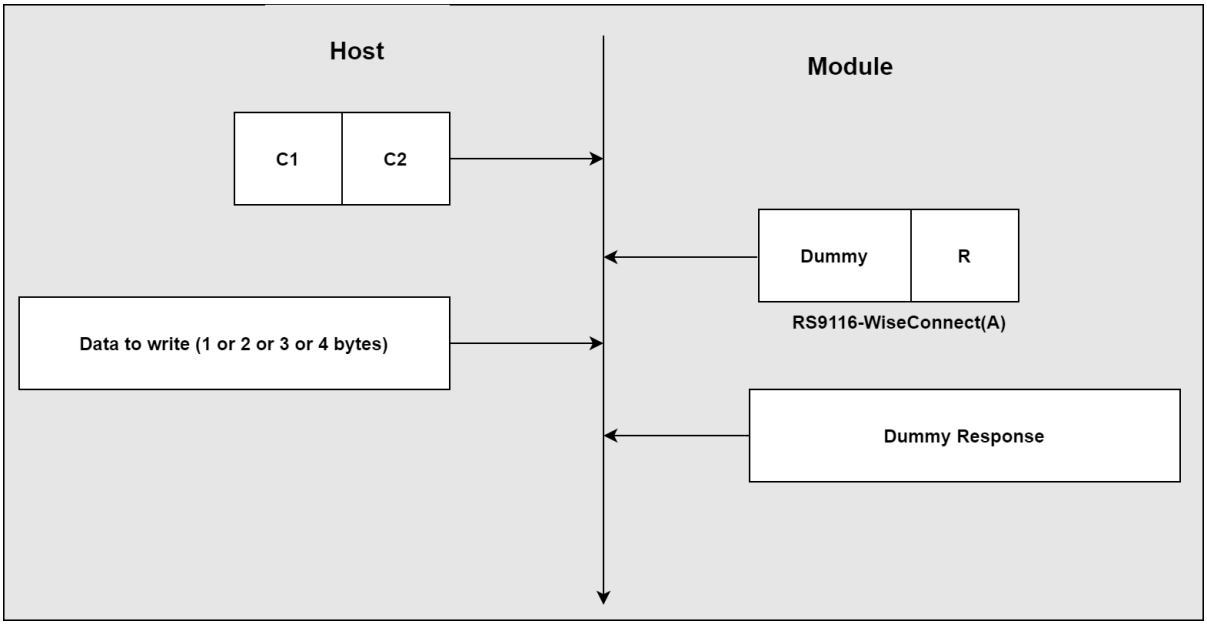


Figure 15: Register Write

Register	Address
SPI_HOST_INTR	0x00

Module registers can be accessed from the host by using register read / writes commands.

Table 2 SPI Host Interrupt Register

SPI_HOST_INTR				
Register Address: 0x00				
Bit	**Access**	**Function**	**Default Value**	**Description**
[7:0]	Read only	SPI_HOST_INTR	0x00	These bits indicate the interrupt status value • Bit 0: If '1', Buffer Full condition reached. When this bit is set to host shouldn't send data packets. This bit has to be polled before sending each packet. • Bit 1: Reserved • Bit 2: Reserved • Bit 3: If '1', indicates data packet or response to Management frames are pending. This is a self-clearing bit and is cleared after the packet is read by host. • Bit 4: Reserved • Bit 5: Reserved • Bit 6: Reserved

Software Protocol

This section explains the procedure that the host needs to follow in order to send Wi-Fi commands frames to the module and to receive responses from the module.

Tx Operation

The Host uses Tx operations:

1. To send management commands to the module from the Host.
2. To send actual data to the module which is to be transmitted onto the air.

The host should follow the steps below to send the command frames to the Module:

1. Host should check buffer full condition by reading interrupt status register using register read.
2. If buffer full bit is not set in interrupt status register, the host needs to send Command frame in two parts:
   First, it is required to send 16 byte Frame descriptor using Frame write.
   Second, it is required to send optional Frame body using Frame write.

Note:
Frame write is an API provided to the host which is present in the release package.

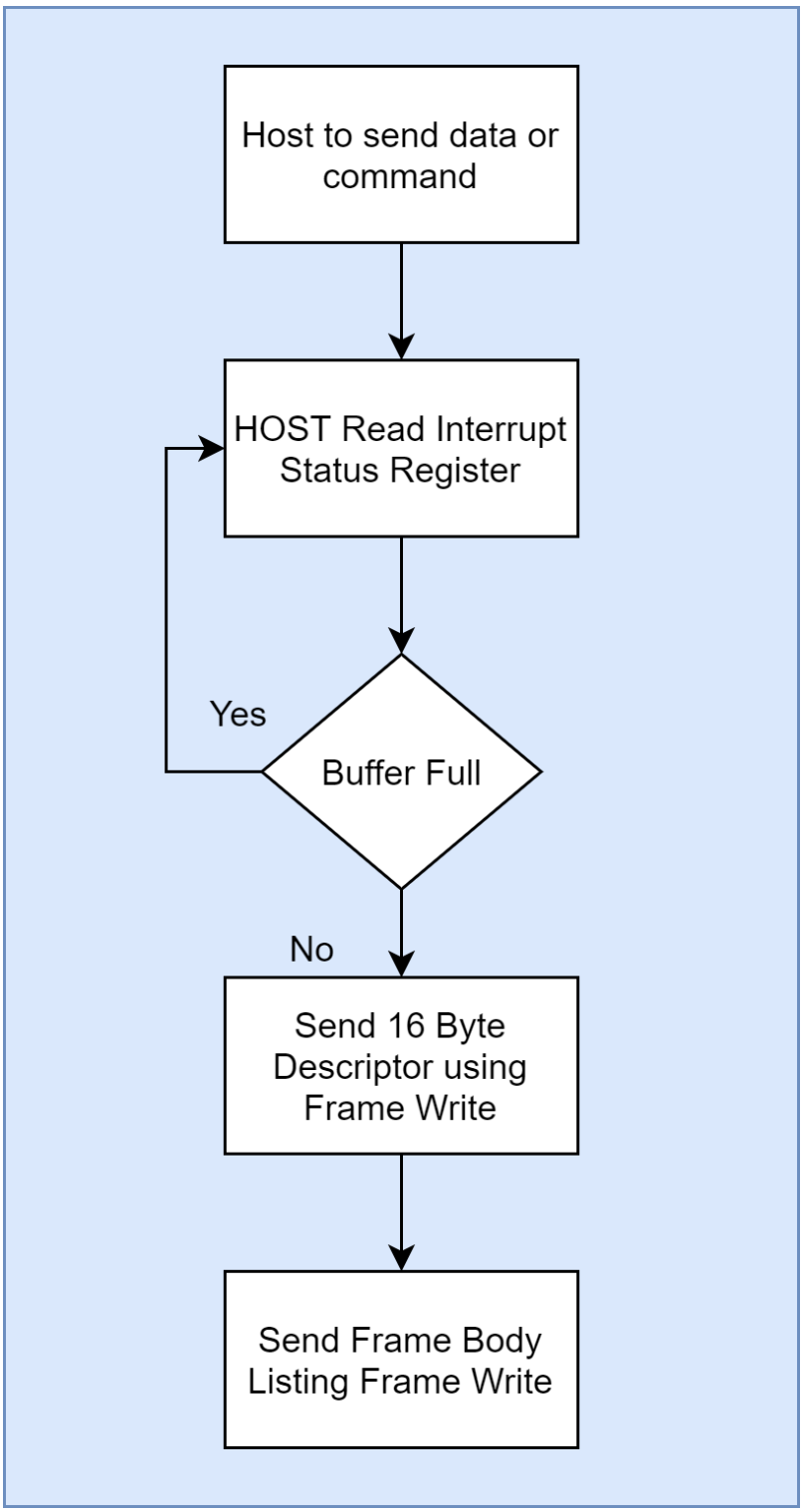


Figure 16: Tx From Host to Module

Management / Data Frame Descriptor and Frame body of command frames are sent to the module by using two separate frames write as shown below:

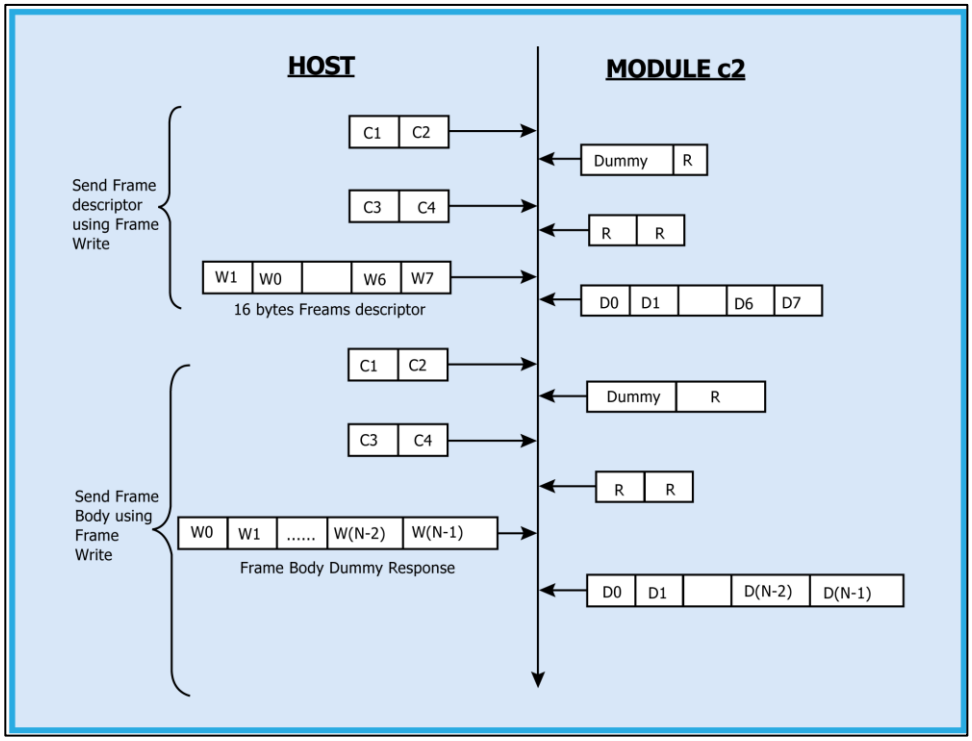


Figure 17: Exchanges between Host and Module for Tx Operation

Rx Operation

The Host uses this operation:

- Module's responses, for the commands.
- To read data received by the module from the remote peer.

The module sends the response/received data to Host in a format as shown below:

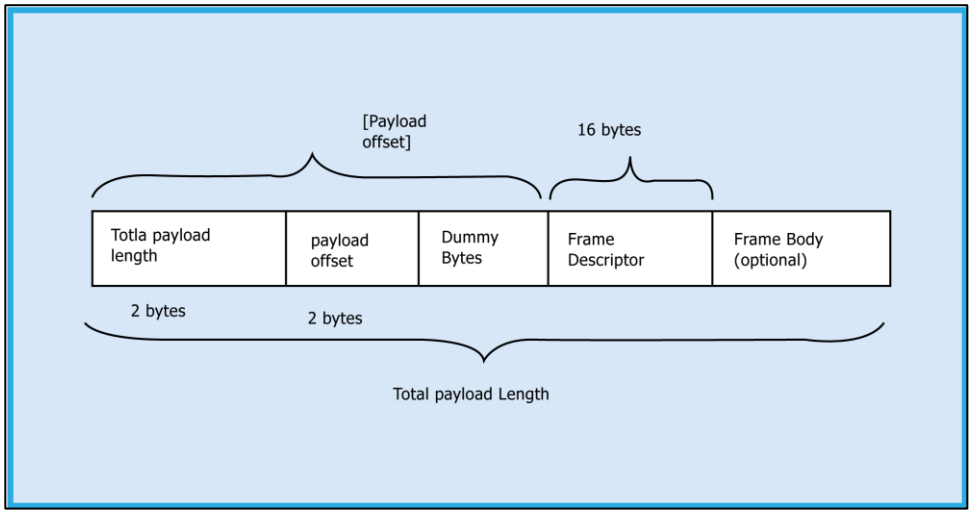


Figure 18: Rx Frame Format

Note:
If Payload offset is 'x', 'x-4' dummy bytes will be added before Frame Descriptor.

The host should follow the steps below to read the frame from the Module:

1. If any Data / Management packet is pending from the module, module raises an interrupt to HOST.
2. The host needs to check the reason for interrupt by reading interrupt status register using register read.
3. If data pending bit is set in interrupt status register, follow the steps below:
    a. Read 4 bytes using Frame read.
    b. Decode Total payload length and payload offset.

Read remaining payload by sending Frame to read with (total payload length – 4 bytes), discard Dummy bytes and then decode Frame descriptor and Frame Body.

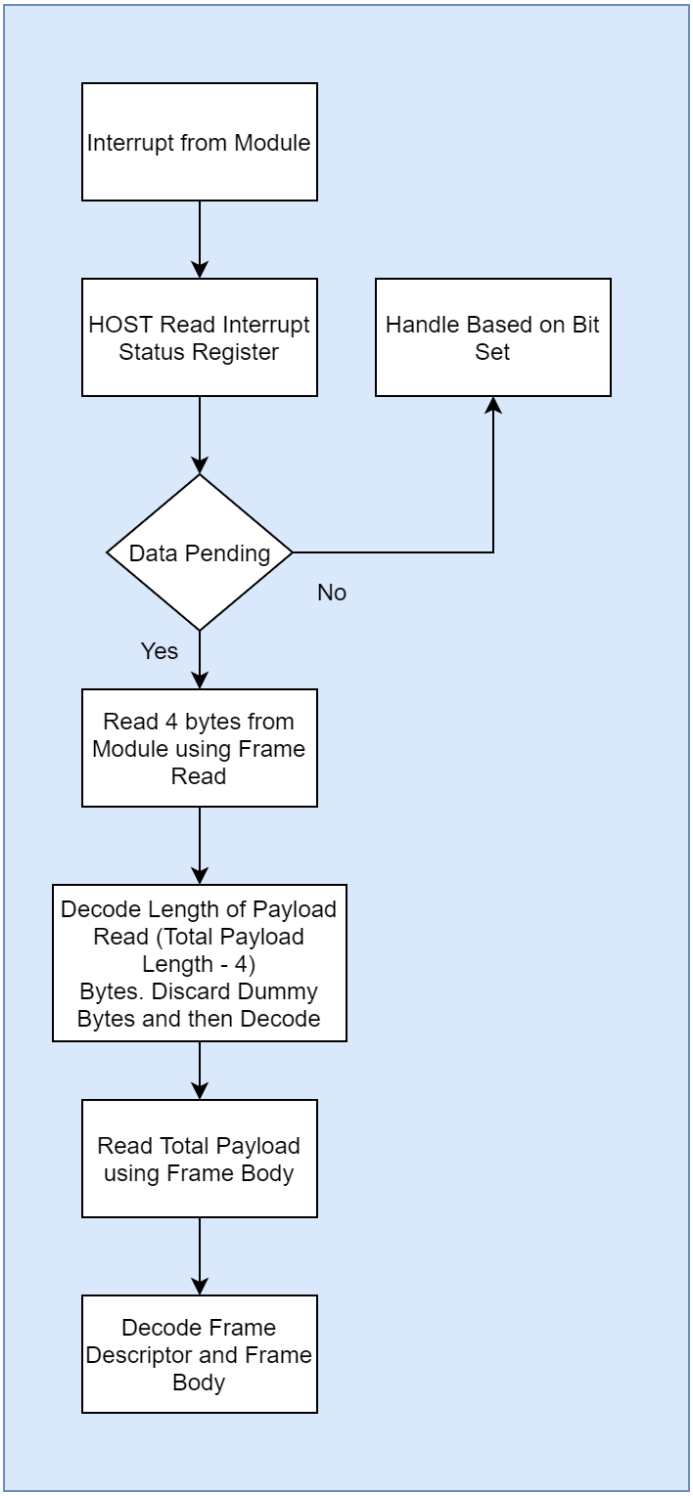


Figure 19: RX Operation from Module to Host

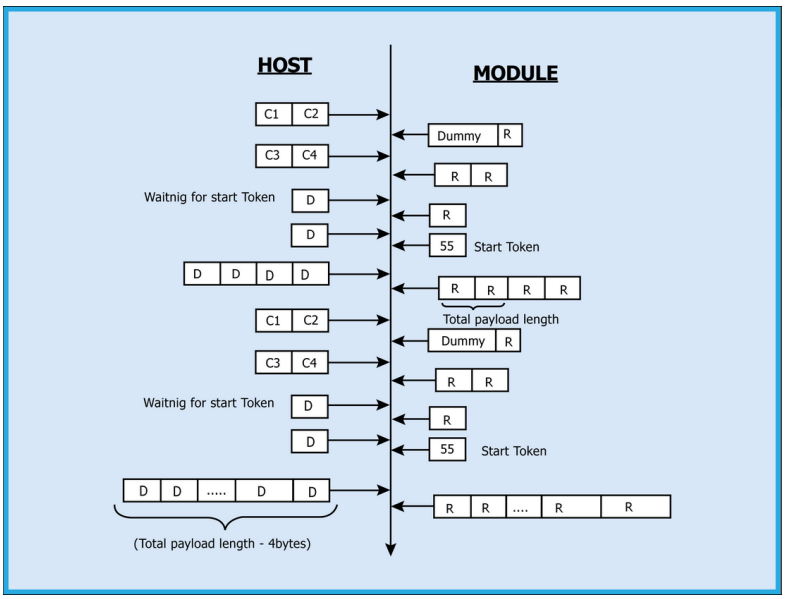


Figure 20: Message Exchanges between Host and Module for Rx Operation

5.2 UART Interface

This section describes RS9116-WiSeConnect UART interface, including the commands and processes to operate the module via UART.

UART on the RS9116-WiSeConnect is used as a host interface to configure the module to send data and also to receive data.

Features

- Supports hardware (RTS/CTS) flow control.
- Supports following baud rates
  - 9600 bps
  - 19200 bps
  - 38400 bps
  - 57600 bps
  - 115200 bps
  - 230400 bps
  - 460800 bps
  - 921600 bps

Note:
For baud rates greater than 115200, it is mandatory to enable UART hardware flow control.

Hardware Interface

RS9116W uses TTL serial UART at an operating voltage of 3.3V.
Host UART device must be configured with the following settings:

- Data bits - 8
- Stop bits - 1
- Parity - None
- Flow control - None

Software Protocol

This section explains the procedure that the host needs to follow in order to send commands frames to the module and to receive responses from the module.

TX Operation

The Host uses TX operations:

1. To send management commands to the module from the Host.
2. To send actual data to the module which is to be transmitted onto the air.
3. If the host receives error code indicating packet dropped, the host must wait for a while and send the next command /data.
4. The host should send next data packet only if it receives ACK frame for the previous one.

The host should follow the steps below to send the command frames to the Module:

1. First, it is required to send 16-byte Frame descriptor using Frame write.
   Second, it is required to send optional Frame body using Frame write.

RX Operation

RS9116W respond with frame of same frame type (with or without an error code in error code field).

The Host uses this operation:

- Module respond, for the commands.
- To read data received by module from remote peer.
- Receive asynchronous information from RS9116W module.

Module send/receive data to/from Host with below described format:

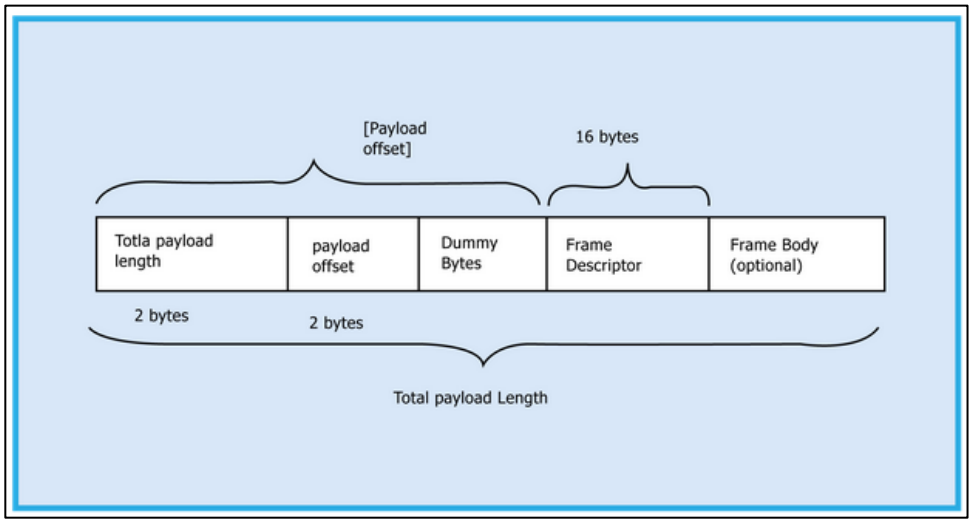


Figure 21: RX Frame Format

Note:
If Payload offset is 'x', 'x-4' dummy bytes get added before Frame Descriptor.

Host should follow the steps below to read the frame from the Module:

Read 4 bytes using Frame read.

1. Decode Total payload length and payload offset.
2. Read remaining payload by sending Frame to read with (total payload length – 4 bytes), discard Dummy bytes and then decode Frame descriptor and Frame Body.

6 Binary Command Mode

This section explains the Wi-Fi commands that are used to configure RS9116-WiSeConnect in Binary Mode.
Following are list of host interfaces supported in Binary Mode:

- UART
- SPI
- USB
- USB-CDC
- SDIO

The Wi-Fi configuration and operation commands are sent to the module and the responses are read from the module by using frame write/frame read (as mentioned in the preceding sections) so these configuration and operation commands are called as command frames.
The command frame is categorized as management or data frames. The management frames are used to configure the Wi-Fi module to access Wi-Fi connectivity, TCP/IP stack and operate the module. Data frames are used to send the data.
Management and data frames are exchanged between the host and module. Management frame is sent from Host to the module to configure the module and is sent from module to host to send responses to these commands.
The format of the command frames is divided into two parts:

1. Management/Data Frame descriptor
2. Management/Data Frame Body

Management/Data Frame Descriptor (16 bytes)	Management/Data Frame Body

Note:
Management/Data Frame Body (variable length) should be the multiples of 4 bytes in the case of SPI interface.

The following is the frame format for management and data frames in Binary Command Mode. The command frame format is shown below. This description is for a Little-Endian System.

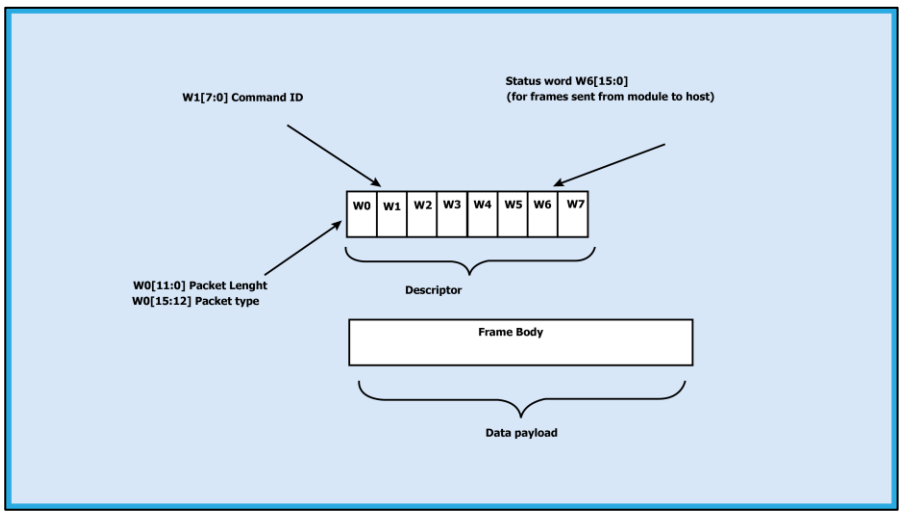


Figure 22: Command Frame Format

The following table provide general description of frame descriptor for management and data frames.

Table 3: Frame Descriptor for Management/Data Frames in Binary Mode

Word	Management Frame Descriptor	Management Frame Descriptor Data Frame Descriptor
Word0 W0[15:0]	Bits [11:0] – Length of the frame Bits [15:12] – 4 (indicate management packet).	Bits [11:0] – Length of the frame Bits [15:12] – 5 (indicate data packet)
Word1 W1[15:0]	Bits [7:0] - Command ID Bits [15:8] - Reserved	Bits [7:0] – 0x0 Datatype. **Note**: Anything other than 0x0 is a management packet. Bits[15:8]-Reserved
Word2 W2[15:0]	Reserved	Reserved
Word3 W3[15:0]	Reserved	Reserved
Word4 W4[15:0]	Reserved	Reserved
Word5 W5[15:0]	Reserved	Reserved
Word6 W6[15:0]	1. (0x0000) when sent from the host to module. 2. When sent from module to host (as response frame), it contains the status.	Reserved
Word7 W7[15:0]	Reserved	Reserved

The management frames represent the command frames that are sent from the Host to the RS9116-WiSeConnect to configure for Wi-Fi access. These are frame write commands. The following are the types of management requests and responses and the corresponding codes. The first table below is applicable when the Host sends the frames to the module; the second table below is applicable when the module sends the frames to the host. The corresponding code is to be filled in W1 [7:0] mentioned in the table above.

Table 4: Command IDs for Tx Data Operation

Command	Command ID
Send Data	0x00
Set Operating Mode	0x10
Band	0x11
Init	0x12
Scan	0x13
Join	0x14
Power save mode	0x15
Sleep Timer	0x16
Set Mac address	0x17
Query Network Parameters	0x18
Disconnect	0x19

Command	Command ID
Antenna Select	0x1B
Soft Reset	0x1C
Set region	0x1D
Config Save	0x20
Config Enable	0x21
Config Get	0x22
User Store configuration	0x23
AP config	0x24
Set WEP Keys	0x25
Debug Prints on UART2	0x26
Ping command	0x29
RSSI Query	0x3A
Multicast Address Filter	0x40
Set IP Parameters	0x41
Socket Create	0x42
Socket Close	0x43
DNS Resolution	0x44
Query LTCP Connection Status	0x46
User Gain table configuration	0x47
Query WLAN Connection Status	0x48
Query Firmware Version	0x49
Get MAC Address	0x4A
Configure P2P	0x4B
Configure EAP	0x4C
Set Certificate	0x4D
Query GO Param	0x4E
Load Web pages	0x50
HTTP GET	0x51
HTTP POST	0x52
HTTP CLIENT PUT	0x53
DNS Server	0x55
URL response	0x56
FWUP REQ OK	0x5A
BG scan	0x6A
HT Caps REQ	0x6D

Command	Command ID
Rejoin Params	0x6F
WPS method REQ	0x72
Select command	0x74
Roam Params REQ	0x7B
PER MODE REQ	0x7C
Webpage Clear REQ	0x7F
SNMP Get response	0x83
SNMP Get next response	0x84
SNMP ENABLE	0x85
SNMP TRAP	0x86
SNMP_GET_STATS	0x88
IPv6 Config	0x90
Trigger Autoconfiguration	0x91
WMM PS REQ	0x97
Firmware upgradation from host	0x99
Webpage Erase REQ	0x9A
JSON erase REQ	0x9B
JSON create REQ	0x9C
PER Stats REQ	0xA2
Transparent Mode REQ	0xA3
UART flow control	0xA4
Set PSK/PMK REQ	0xA5
Socket Configuration REQ	0xA7
RF current mode configuration	0xAD
Multicast request	0xB1
Sent Bytes on Socket	0xB2
HTTP Abort	0xB3
HTTP Credentials Request	0xB4
Load MFI i.e. in the beacon	0XB5
Read Authentication certificate	0xB6
IAP coprocessor init	0xB7
Generate MFI authentication signature	0XB8
certificate valid	0XBC
Set Region of Access point	0xBD
Request Config Command	0xBE

Command	Command ID
FEATURE_FRAME	0xC8
PUF_Intrinsic key	0xCE
PUF Enroll	0xD0
PUF disable enroll	0xD1
PUF start	0xD2
PUF set key	0xD3
PUF disable set key	0xD4
PUF get key	0xD5
PUF disable get key	0xD6
PUF load key	0xD7
AES Encrypt	0xD8
AES Decrypt	0xD9
AES MAC	0xDA
MDNSD START REQ	0xDB
Power save ACK	0xDE
FTP REQ	0xE2
FTP FILE WRITE REQ	0xE3
SNTP REQ	0xE4
SMTP REQ	0xE6
POP3 Client REQ	0xE7
Set RTC time	0xE9
DHCP USER CLASS REQ	0xEC

Table 5: Response IDs for Rx Operation

Command	Response ID
Set Operating Mode	0x10
Band	0x11
Init	0x12
Scan	0x13
Join	0x14
Power save mode	0x15
Sleep Timer	0x16
Set Mac address	0x17
Query Network Parameters	0x18
Disconnect	0x19

Command	Response ID
Antenna select	0x1B
Soft Reset	0x1C
Set region	0x1D
Config save	0x20
Config Enable	0x21
Config Get	0x22
User store configuration	0x23
AP Config	0x24
Set WEP Keys	0x25
Debug Prints on UART2	0x26
Ping command	0x29
Async connection accept the request from remote wfd device	0x30
RSSI Query	0x3A
Multicast Address filter	0x40
IP Parameters Configure	0x41
Socket Create	0x42
Socket Close	0x43
DNS Resolution	0x44
Query LTCP Connection Status	0x46
User Gain table configuration	0x47
Query WLAN Connection Status	0x48
Query Firmware Version	0x49
Get MAC Address	0x4A
Configure P2P	0x4B
Configure EAP	0x4C
Set Certificate	0x4D
Query GO Params	0x4E
Load webpage	0x50
HTTP GET	0x51
HTTP POST	0x52
HTTP PUT	0x53
Async WFD	0x54
DNS Server	0x55
URL response	0x56
FWUP RSP	0x59

Command	Response ID
Async TCP Socket Connection Established	0x61
Async Socket Remote Terminate	0x62
URL request	0x64
BG scan	0x6A
HT Caps RSP	0x6D
Rejoin params	0x6F
Module state	0x70
WPS method RSP	0x72
Roam Params RSP	0x7B
Select RSP	0x74
PER MODE RSP	0x7C
Webpage Clear RSP	0x7F
SNMP GET	0x80
SNMP GET NEXT	0x81
SNMP SET	0x82
SNMP GET RESPONSE RSP	0x83
SNMP GET NEXT RESPONSE RSP	0x84
SNMP ENABLE RSP	0x85
SNMP TRAP RSP	0x86
SNMP_GET_STATS	0x88
Card Ready	0x89
WMM PS RSP	0x97
Webpage Erase RSP	0x9A
JSON erase RSP	0x9B
JSON create RSP	0x9C
JSON Update	0x9D
Config IPv6	0xA1
PER Stats	0xA2
Transparent Mode RSP	0xA3
UART flow control RSP	0xA4
Set PSK/PMK RSP	0xA5
Socket Configuration RSP	0xA7
IP Change notify	0xAA
TCP ACK indication to host	0xAB
Delayed ACK for UART binary	0xAC

Command	Response ID
mode	
RF current mode configuration	0xAD
Multicast response	0xB1
HTTP Abort	0xB3
HTTP Credentials	0xB4
Load MFI i.e. in beacon	0XB5
Read Authentication certificate	0xB6
IAP coprocessor init	0xB7
Generate MFI authentication signature	0XB8
certificate valid	0xBC
Set Region of Access point	0xBD
rsi_config RSP	0xBE
Station connected Indication	0xC2
Station Disconnected Indication	0xC3
Feature Frame	0xC8
PUF intrinsic key	0xCE
PUF Enroll	0xD0
PUF disable enroll	0xD1
PUF start	0xD2
PUF set key	0xD3
PUF disable set key	0xD4
PUF get key	0xD5
PUF disable get key	0xD6
PUF load key	0xD7
AES Encrypt	0xD8
AES Decrypt	0xD9
AES MAC	0xDA
Wakeup indication	0xDD
Sleep indication	0xDE
Receive data	Ignore the response ID field while receiving data from a remote terminal
MDNSD START RSP	0xDB
FTP RSP	0xE2
FTP FILE WRITE RSP	0xE3
SNTP RSP	0xE4
SNTP SERVER RSP	0xE5

Command	Response ID
POP3 Client RSP	0xE7
POP3 Client terminate RSP	0xE8
RTC time response	0xE9
HTTP POST DATA	0xEB
DHCP USER CLASS RSP	0xEC
DNS Update	0xED
OTAF	0xEF

7 SAPI Library Directory Structure

This document explains the SAPI Library Directory Structure. It shows different APIs groups and how APIs are structured in directories.

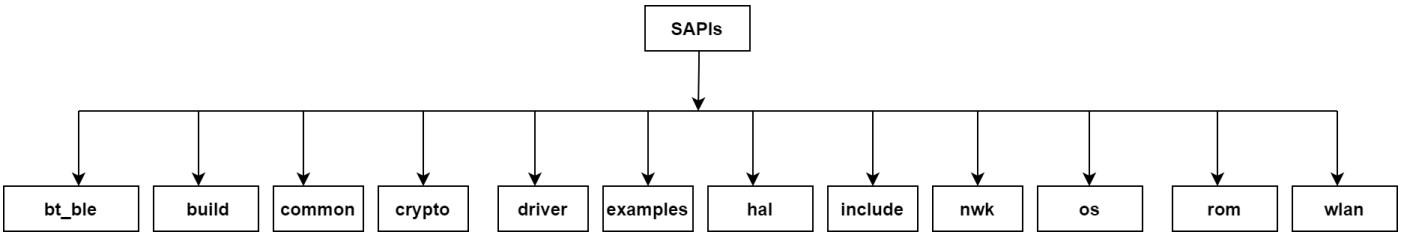


Figure 23: SAPI Library Directory Structure

SAPIs:

- **bt_ble**: This folder contains simplified APIs to use Bluetooth Classic and Bluetooth low energy wireless protocols
- **build**: This folder contains common Makefile to build all the applications present in "sapis" folder.
- **common**: This folder contains source files for common apis like device init, driver init, firmware query etc..
- **crypto**: This folder contains the apis related to cryptographic functions.
- **driver**: This folder contains driver source files for different host interfaces like spi, sdio and uart.
- **examples**: This folder contains reference examples for each wireless protocol.
- **hal**: This folder contains hardware abstraction layer for different host interfaces like uart, spi and sdio etc.. for MCUs.
- **include**: This folder contains all the dependent header files for the apis/applications.
- **nwk**: This folder contains network related applications (mqtt, http, dns etc..)
- **os**: This folder contains wrapper files if user wants to use Embedded OS.
- **rom**: This folder contains the rom related apis for host interfaces, network and MCU relates apis.
- **wlan**: This folder contains simplified APIs related to WLAN wireless protocol like scan, join, ipconfig etc.

8 Common APIs

This section explains the common APIs to initialize the driver and to handle common features which are independent of module configuration mode.

8.1 rsi_driver_init

Prototype

```
int32_t *rsi_driver_init(uint8_t *buffer, uint32_t length);
```

Description
This API initializes the WiSeConnect driver.

Precondition
NA

Parameters

Parameter	Description
buffer	This is the pointer to buffer from application. The driver uses this buffer to hold driver control for its operation.
length	This is the length of the buffer.

Return Values

Value	Description
<= length	Successful execution of the command. Returns the memory used, which is less than or equal to buffer length provided.
< 0 >length	Failure. -1: if UART initialization fails in SPI / UART mode -2: if maximum sockets is greater than 10 >length: returns total memory required by driver

Example

```
#define BUFFER_LENGTH 8000
uint8_t buffer[BUFFER_LENGTH];
rsi_driver_init(buffer, BUFFER_LENGTH);
```

8.2 rsi_set_intr_type

Prototype

```
int16_t rsi_set_intr_type(uint32_t interruptMaskVal)
```

Description
This API sets the interrupt type of the module.

Precondition
If interface is SPI, rsi_spi_iface_init() must be called before this API.

Parameters

Parameter	Description
interruptMaskVal	RSI_ACTIVE_HIGH_INTR - 0 RSI_ACTIVE_LOW_INTR - 2

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
// Configure the interrupt type of the module
 status = rsi_set_intr_type(RSI_ACTIVE_HIGH_INTR);
 if(status != RSI_SUCCESS)
 {
   return status;
 }
```

8.3 rsi_device_init

Prototype

```
int32_t rsi_device_init(uint8_t select_option);
```

Description
This API power cycles the module and sets the boot up option for WiSeConnect features. This API also initializes the module SPI.

Precondition
rsi_driver_init() must be called before this API.

Parameters

Parameter	Description
select_option	RSI_LOAD_IMAGE_I_FW : To load Firmware image RSI_LOAD_IMAGE_I_ACTIVE_LOW_FW : To load active low Firmware image. Active low firmware will generate active low interrupts to indicate that packets are pending on the module, instead of the default active high. RSI_UPGRADE_IMAGE_I_FW : To upgrade firmware file

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
// Initialize the module and tell it to load its default firmware.
 status = rsi_device_init(RSI_LOAD_IMAGE_I_FW);
 if(status != RSI_SUCCESS)
 {
   return status;
 }
```

8.4 rsi_bl_upgrade_firmware

Prototype

```
int16_t rsi_bl_upgrade_firmware(uint8_t *firmware_image,
                      uint32_t fw_image_size,
                      uint8_t flags);
```

Description
This API upgrades the firmware in the WiSeConnect device from the host. The firmware file is given in chunks to this API.
Each chunk must be a multiple of 4096 bytes unless it is the last chunk.

For the first chunk, set RSI_FW_START_OF_FILE in flags.
For the last chunk set RSI_FW_END_OF_FILE in flags.

Precondition

N/A

Parameters

Parameter	Description
firmware_image	This is a pointer to firmware image buffer
fw_image_size	This is the size of firmware image
Flags	1 - RSI_FW_START_OF_FILE 2 - RSI_FW_END_OF_FILE Set flags to 1 - if it is the first chunk 2 - if it is last chunk, 0 - for all other chunks

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
rsi_bl_upgrade_firmware(fw_image, FW_IMG_SIZE, 1);
```

8.5 rsi_cmd_uart_flow_ctrl

Prototype

```
int32_t rsi_cmd_uart_flow_ctrl(uint8_t uartflow_en);
```

Description

This function is used to enable or disable UART flow control.

parameter

Parameter	description
uartflow_en	Enable or Disable UART hardware flow control 1/2 - Enable and Pin set to be used to for RTS/CTS purpose. 0 - Disable If uart_hw_flowcontrol_enable parameter is 0, uart flow control is disabled. If the parameter is given as 1 or 2, then UART hardware flow control is enabled and Pin set to be used If parameter is given as 1: Pin set used for RTS/CTS functionality is: • UART_CTS: GPIO - 11 • UART_RTS: GPIO - 7 If parameter is given as 2: Pin set used for RTS/CTS functionality is: • UART_CTS: GPIO - 15 • UART_RTS: GPIO - 12
Output	none

Return Values

Value	Description
0	Success
<0	Failure

Example

```
status = rsi_cmd_uart_flow_ctrl(uartflow_en);
```

8.6 rsi_wireless_init

Prototype

```
int32_t rsi_wireless_init(uint16_t opermode, uint16_t coex_mode);
```

Description
This API initializes the WiSeConnect features.

Precondition
rsi_driver_init() API needs to be called before this API.

Parameters

Parameter	Description
opermode	Operating mode 0 - Client mode 2 - Enterprise security client mode 6 - Access point mode 8 – Transmit test mode 9 - Concurrent mode
coex_mode	Coexistence mode

Coex Mode	Description
0	WLAN only mode
1	WLAN
4	Bluetooth
5	WLAN + Bluetooth
8	Dual Mode (Bluetooth and BLE)
9	WLAN + Dual Mode
12	BLE mode
13	WLAN + BLE

Note: Coex modes are supported only in 384K memory configuration.

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non-Zero Value	Failure **If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x0025,0xFF73,0x002C,0xFF6E,0xFF6F, 0xFF70,0xFFC5

Note: Refer Error Codes section for above error codes description.

Example

```
int32_t result = rsi_wireless_init();
 if(result != RSI_SUCESS)
 {
   // handle error
 }
```

8.7 rsi_wireless_deinit

Prototype

```
int32_t rsi_wireless_deinit()
```

Description
This API resets WiSeConnect module, download the firmware and restarts the driver. This API should be called before rsi_wireless_init command, if user wants to change the previous configuration given through rsi_wireless_init.

Precondition
N/A

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command

Example

```
rsi_wireless_deinit();
```

Note:

For restarting RS9116W module from host, application needs to call rsi_driver_deinit() followed by rsi_driver_init() and rsi_device_init(). For OS cases, additionally needs to call rsi_task_destroy(driver_task_handle) to delete the driver task before calling rsi_driver_deinit() and should create again after rsi_device_init() using rsi_task_create().

8.8 rsi_wireless_driver_task

Prototype

```
void rsi_wireless_driver_task(void);
```

Description
This API handles the driver events. It should be called in application main loop for non-OS platforms.

Precondition
N/A

Parameters
None

Return Values
None

Example

```
// main loop (no OS)
 while(1)
 {
   // application logic goes here
    rsi_wireless_driver_task();
 }
```

8.9 rsi_wireless_antenna

Prototype

```
int32_t rsi_wireless_antenna(uint8_t type, uint8_t gain_2g, uint8_t gain_5g);
```

Description
This API configures the antenna.

Precondition
rsi_wireless_antenna command must be given after rsi_init command.

Parameters

Parameter	Description
type	0 : RF_OUT_2/Internal Antenna is selected 1 : RF_OUT_1/uFL connector is selected.
gain_2g	Currently not supported
gain_5g	Currently not supported

Note: Currently ignoring the gain_2g, gain_5g, antenna_path, antenna_type values.

Note: Currently wireless_antenna selection is not supported.

Return Values

Values	Description
0	Successful execution of the command
Non-Zero Value	Failure

Values	Description
	If return value is lesser than 0 -4: Buffer not available to serve the command **If return value is greater than 0** 0x0025, 0x002C

Note: Refer to Error Codes section for above error codes description.

Example

```
// select antenna and specify gain
 status = rsi_wireless_antenna(RF_OUT_2,gain_2g,gain_5g);
```

8.10 rsi_set_rtc_timer

Prototype

```
int32_t rsi_set_rtc_timer(module_rtc_time_t *timer);
```

Description
This API is used to set the host rtc timer.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
timer	pointer to fill rtc time

Response structure

```
typedef struct module_rtc_time_s{
    uint8_t    tm_sec[4];                    //@ seconds after the minute [0-60]
    uint8_t    tm_min[4];                    //@ minutes after the hour [0-59]
    uint8_t    tm_hour[4];                   //@ hours since midnight [0-23]
    uint8_t    tm_mday[4];                   //@ day of the month [1-31]
    uint8_t    tm_mon[4];                    //@ months since January [0-11]
    uint8_t    tm_year[4];                   //@ year since 0
}module_rtc_time_t;
```

Response parameters:

Parameter	Description
second	This is current real time clock seconds.
minute	This is current real time clock minute.
hour	This is current real time clock hour.
day	This is current real time clock day.
month	This is current real time clock month
year	This is current real time clock year.

Note: Hour is 24-hour format only (valid values are 0 to 23). Valid values for Month are 0 to 11 (January to December).

Return values

Value	Description
0	Success
Non-Zero Value	Failure 0x0021, 0x0025

Example

```
status = rsi_set_rtc_timer(module_rtc_time_t *timer);
```

8.11 rsi_get_rtc_timer

Prototype

```
int32_t rsi_get_rtc_timer(module_rtc_time_t *response);
```

Description
This API is used to get the host rtc timer.

Precondition

rsi_set_rtc_timer() API needs to be called before this API.

Parameters

Parameter	Description
response	This is the response of the requested command. This is an output parameter.

Response structure

```
typedef struct module_rtc_time_s{
    uint8_t    tm_sec[4];                    //@ seconds after the minute [0-60]
    uint8_t    tm_min[4];                    //@ minutes after the hour [0-59]
    uint8_t    tm_hour[4];                   //@ hours since midnight [0-23]
    uint8_t    tm_mday[4];                   //@ day of the month [1-31]
    uint8_t    tm_mon[4];                    //@ months since January [0-11]
    uint8_t    tm_year[4];                   //@ year since 0
}module_rtc_time_t;
```

Response parameters:

Parameter	Description
second	This is current real time clock seconds.
minute	This is current real time clock minute.
hour	This is current real time clock hour.
day	This is current real time clock day.
month	This is current real time clock month
year	This is current real time clock year.

Note: Hour is 24-hour format only (valid values are 0 to 23). Valid values for Month are 0 to 11 (January to December).

Return values

Value	Description
0	Success
Non-Zero Value	Failure 0x0021, 0x0025

Example

```
status = rsi_get_rtc_timer(module_rtc_time_t *response);
```

8.12 rsi_get_ram_log

Prototype

```
int32_t rsi_get_ram_log(uint32_t addr, uint32_t length);
```

Description
This API is used to get ram log on UART/UART2.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameters	Description
addr	Address in RS9116 module
length	Chunk length to read from RS9116 module.

Return values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state **If return value is greater than 0** 0x0021,0x003E

Note:

Refer Error Codes section for above error codes description.

Example

```
status = rsi_get_ram_log(addr, length);
```

8.13 rsi_get_ram_dump

Prototype

```
int32_t rsi_get_ram_dump(uint32_t addr,uint16_t length, uint8_t *buffer);
```

Description
This API is used to get ram dump through master reads.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameters	Description
addr	Address in RS9116 module
length	Chunk length to read from RS9116 module
buffer	ram content saved in buffer

Return values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is lesser than 0** -1: Invalid parameters -2: Command given in wrong state

Note:

Refer Error Codes section for above error codes description.

Example

```
status = rsi_get_ram_dump(READ_ADDRESS + offset,chunk_len, &ram_content[offset]);
```

8.14 rsi_get_fw_version

Prototype

```
int32_t rsi_get_fw_version(
 uint8_t *response,
 uint16_t length);
```

Description
This API gets the firmware version present in the device.

Parameters

Parameter	Description
response	This is the response of the requested command. This is an output parameter.
length	This is the length of the response buffer in bytes to hold result.

Return Value

Value	Description
0	Successful execution of the command.
Non-Zero Value	Failure

Value	Description
	If return value is lesser than 0 -3: Command given in wrong state -4: Buffer not available to serve the command -6: Insufficient input buffer given **If return value is greater than 0** 0x0021, 0x0025, 0x002c

Note:

Refer Error Codes section for above error codes description.

Example

```
uint8_t fw_version_array[20];

//! Get fw version
status = rsi_get_fw_version(fw_version_array, sizeof(fw_version_array));
```

8.15 rsi_common_debug_log

Prototype

```
int32_t rsi_common_debug_log(
uint32_t assertion_type,
uint32_t assertion_level);
```

Description
This command is used for debug prints on UART interfaces 1 and 2. Host can get 5 types of debug prints based on the assertion level and assertion type.

Precondition
None

Parameters

Parameter	Description
assertion_type	Assertion_type (Possible values 0 - 15), 0000 - LMAC core, 0001 - SME, 0010 - UMAC, 0100 - NETX, 1000 - Enables assertion indication and provides ram dump in critical assertion.
assertion_level	Assertion_level (Possible values 0 - 15). value 1 is least value/only specific prints, 15 is the highest level/Enable all prints. 0000 - Assertion required, 0010 - Recoverable, 0100 - Information

Return Values

Value	Description
0	Success
Non-Zero value	Failure
RSI_ERROR_INVALID_PARAM	Parameters invalid

Note:

1. To minimize the debug prints host is supposed to give the same command with assertion type and assertion level as 0.
2. Baud rate for UART 2 on host application side should be 460800.
3. Enable few Feature bit map for getting debug logs on UART:

```
//! To set custom feature select bit map
#define RSI_CUSTOM_FEATURE_BIT_MAP   FEAT_CUSTOM_FEAT_EXTENTION_VALID
//! To set Extended custom feature select bit map
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP    EXT_FEAT_UART_SEL_FOR_DEBUG_PRINTS
```

Example

```
//! enable debug log prints
  status = rsi_common_debug_log(0, 15);
  if(status != RSI_SUCCESS)
  {
    return status;
 }
```

8.16 rsi_driver_deinit

Prototype

```
int32_t rsi_driver_deinit();
```

Description

De-Initializes the driver components.

Clears all the memory given for driver operations in rsi_driver_init() API. In OS case, User need to take care of OS variables initialized in rsi_driver_init() .

This API has to be called by the thread/task/Master thread which is not dependent on OS variables allocated/initialized in rsi_driver_init() API.

Precondition
Need to call after the driver initialization.

Return Values

Value	Description
0	Successful execution of the command.
Non-Zero Value	Failure.

8.17 rsi_device_deinit

Prototype

```
int32_t rsi_device_deinit(void);
```

Description

- De-Initializes the module.
- Resets the module. Reset is driven to the module by asserting the RESET_PS pin for some duration and releasing it.
- RESET_PS pin is available on the EVK.

Precondition
NA

Parameters

No

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure.

Example

```
status = rsi_device_deinit();
```

8.18 rsi_gpio_pininit

Prototype

```
int32_t rsi_gpio_pininit(uint8_t pin_num, uint8_t configuration);
```

Description
This API is used to configure TA GPIO's using Command from host.

Precondition
NA

Parameters

Parameter	Description
pin_num	GPIO Number: Valid values 0 - 15
configuration	BIT[4] : 1 - Input mode, 0 - Output mode BIT[0 - 1] - Drive strength : 0 - 2mA , 1 - 4mA , 2 - 8mA, 3 - 12mA BIT[6 - 7] : 0 - Hi-Z, 1- Pull-up, 2- Pull-down

Return Values

Value	Description
0	Success
Non-Zero value	Failure

Example

```
status = rsi_gpio_pininit(pin_num, configuration);
```

8.19 rsi_gpio_writepin

Prototype

```
int32_t rsi_gpio_writepin(uint8_t pin_num, uint8_t value);
```

Description
This API is used to write the TA GPIO's high or low using command from host.

Precondition
NA

Parameters

Parameter	Description
pin_num	GPIO Number: Valid values 0 - 15
value	Value to be driven on GPIO 1 - Drive high 0 - Drive Low

Return Values

Value	Description
0	Success
Non-Zero value	Failure

Example

```
status = rsi_gpio_writepin(2, 1);
```

8.20 rsi_gpio_readpin

Prototype

```
int32_t rsi_gpio_readpin(uint8_t pin_num, uint8_t *gpio_value);
```

Description
This API is used to read the TA GPIO's using command from host.

Precondition
NA

Parameters

Parameter	Description
pin_num	GPIO Number: Valid values 0 - 15
gpio_value	Address of variable to store the value

Return Values

Value	Description
0	Success
Non-Zero value	Failure

Example

```
status = rsi_gpio_readpin(pin_num, gpio_value);
```

9 WLAN APIs

This Section explains Wi-Fi APIs in order to initialize and configure the module in Wi-Fi mode.

9.1 WLAN Core API

This section explains the core APIs required to configure the module in WLAN mode.

9.1.1 rsi_wlan_scan

Prototype

```
int32_t rsi_wlan_scan(uint8_t *ssid, uint8_t chno, rsi_rsp_scan_t *result,uint32_t length)
```

Description
This API scans the surrounding Wi-Fi networks.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
ssid	• This is the SSID of the Access points which are to be scanned. • This parameter scans Wi-Fi network with a given ssid. • The SSID size should be less than or equal to 32 bytes. • The SSID should be NULL to scan all the Access points. **Note**: SSID is a null terminated string.
chno	This is the channel number to perform scan. If 0, then the module will scan all the channels.
result	• This is the scanned Wi-Fi networks information. • This is an output parameter. This output contains a maximum of 11 scan results. • The structure *rsi_rsp_scan_t* contains members *scan_count* which specifies the number of scan results followed by array of structure type *rsi_scan_info_t,* where each array element contains information about each network scanned.
length	This is the size that should be allocated to buffer which will store scan results. where scan result buffer is of **Scan Response structure format**. **Note**: Size of structure rsi_rsp_scan_t is 54 bytes. In which length for storing one scan result is sizeof() which is 46 bytes. Maximum of 11 scan results will be returned by the module, in this case length should be configured as 8 + 11*sizeof(rsi_scan_info_t).

Channels supported in 2.4 GHz Band

Table 6: Channels Supported in 2.4 GHz Band

Channel Number	chno
All channels	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14

Channels supported in 5GHz band:

Table 7: Channels Supported in 5GHz band

Channel Number	chno
All channels	0
36	36
40	40
44	44
48	48
100	100
104	104
108	108
112	112
116	116
132	132
136	136
140	140
149	149
153	153
157	157

Channel Number	chno
161	161
165	165

Note:

To set various timeouts, user should change the following macros in rsi_wlan_config.h

#define RSI_TIMEOUT_SUPPORT RSI_ENABLE

#define RSI_TIMEOUT_BIT_MAP 1

#define RSI_TIMEOUT_VALUE 300

timeout_bitmap[0]	sets timeout for association and authentication request. timeout_value : timeout value in ms(default 300ms).
timeout_bitmap[31-1]	reserved

Scan Response structure format

```
typedef struct rsi_rsp_scan_s
 {
   uint8_t scan_count[4];
   uint8_t reserved[4];
   rsi_scan_info_t scan_info[11];
 } rsi_rsp_scan_t;
```

Structure Fields	Description
scan_count	This is the number of access points scanned. scan_count[0] contains the scan count. scan_count[3-1] are reserved.
scan_info	This is the information about scanned Access Point in rsi_scan_info_t structure.

rsi_scan_info_t structure

```
typedef struct rsi_scan_info_s
 {
   uint8_t rf_channel;
   uint8_t security_mode;
   uint8_t rssi_val;
   uint8_t network_type;
   uint8_t ssid[32];
   uint8_t bssid[6];
   uint8_t reserved[2];
 } rsi_scan_info_t;
```

Structure Fields	Description
rf_channel	This is the access point channel number
security_mode	This is the security mode 0 : Open 1 : WPA 2 : WPA2 3 : WEP 4 : WPA Enterprise 5 : WPA2 Enterprise
rssi_val	This is the RSSI value of the Access Point
network_type	This is the type of the network 1 : Infrastructure mode

Structure Fields	Description
ssid	This is the SSID of an access point
bssid	This is the MAC address of an access point

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0002, 0x0003, 0x0005, 0x000A, 0x0014, 0x0015, 0x001A, 0x0021,0x0024,0x0025,0x0026,0x002C,0x003c

Note: Refer Error Codes section for above error codes description.

Example

```
//! WC initialization
status = rsi_wireless_init(0, 0);
//! Scan for Access points
status = rsi_wlan_scan((int8_t *)SSID, (uint8_t)CHANNEL_NO, NULL, 0);
```

9.1.2 rsi_wlan_scan_async

Prototype

```
int32_t rsi_wlan_scan_async(uint8_t *ssid,
uint8_t chno, void(*scan_response_handler)(uint16_t status, const uint8_t *buffer, const uint16_t
length))
```

Note: If status is success, the argument buffer passed to scan_response_handler holds scan results in rsi_rsp_scan_t structure format.

Description
This API scans the surrounding Wi-Fi networks. A scan response handler is registered with it to get the response for scan.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
ssid	This is the SSID of the Access points which are to be scanned. This parameter scans the Wi-Fi network with a given SSID. The SSID size should be less than or equal to 32 bytes. The SSID should be NULL to scan all the Access points. **Note**: SSID is a null terminated string.

Parameter	Description
chno	Channel number to perform scan, if 0 then module will scan in all channels.
Scan_response_handler	This callback is called when the response for scan has been received from the module. The parameters involved are status, buffer & length. **Status:** This is the response status. If status is zero, the scan response is stated as success **Buffer:** This is the response buffer **Length:** This is the length of the resulted buffer measured in bytes to hold scan results. Size of structure rsi_rsp_scan_t is 54 bytes i n which length for storing one scan result is sizeof(rsi_scan_info_t) which is 46 bytes. Maximum of 11 scan results will be returned by the module, in this case length should be configured as 8 + 11*sizeof(rsi_scan_info_t).

Channels supported in 2.4GHz band:

Table 8: 2.4GHz Band Channel Mapping

Channel Number	ch no
All channels	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14

Channels supported in 5GHz band:

Table 9: 5GHz Band Channel Mapping

Channel Number	ch no
All channels	0
36	36
40	40
44	44

Channel Number	ch no
48	48
100	100
104	104
108	108
112	112
116	116
132	132
136	136
140	140
149	149
153	153
157	157
161	161
165	165

Note:

To set various timeouts, user should change the following macros in rsi_wlan_config.h

#define RSI_TIMEOUT_SUPPORT RSI_ENABLE

#define RSI_TIMEOUT_BIT_MAP 1

#define RSI_TIMEOUT_VALUE 300

timeout_bitmap[0]	sets timeout for association and authentication request. timeout_value: timeout value in ms (default 300ms).
timeout_bitmap[31-1]	reserved

Scan Response structure format

```
typedef struct rsi_rsp_scan_s
 {
 uint8_t scan_count[4];
 uint8_t reserved[4];
 rsi_scan_info_t scan_info[11];
 } rsi_rsp_scan_t;
```

Structure Fields	Description
scan_count	This is the number of Access points scanned scan_count[0] contains the scan count. scan_count[3-1] are reserved.
scan_info	This is the information about scanned Access points in rsi_scan_info_t structure

rsi_scan_info_t structure

```
typedef struct rsi_scan_info_s
{
 uint8_t rf_channel;
 uint8_t security_mode;
 uint8_t rssi_val;
```

```
uint8_t network_type;
uint8_t ssid[32];
uint8_t bssid[6];
uint8_t reserved[2];
}rsi_scan_info_t;
```

Structure Fields	Description
rf_channel	This is the access point channel number
security_mode	This is the security mode 0: Open 1: WPA 2: WPA2 3: WEP 4: WPA Enterprise 5: WPA2 Enterprise
rssi_val	This is the RSSI value of Access Point
network_type	This is the type of network 1: Infrastructure mode
ssid	This is the SSID of an access point
bssid	This is the MAC address of an access point

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0002,0x0003,0x0005,0x000A,0x0014,0x0015,0x001A, 0x0021, 0x0024, 0x0025, 0x0026, 0x002C, 0x003c

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
void rsi_scan_handler(uint16_t status,const uint8_t *buffer, const uint16_t length)
{
  // your code here
}
//! WC initialization
status = rsi_wireless_init(0, 0);
//! Scan for Access points
status = rsi_wlan_scan_async((int8_t *)SSID, 0,rsi_scan_handler);
if(status == RSI_SUCCESS)
{
  while(1)
  {
     //! check for scan done state
     if(state == SCAN_DONE)
        break;
```

```
    }
  }
```

9.1.3 rsi_wlan_scan_with_bitmap_options

Prototype

```
int32_t rsi_wlan_scan_with_bitmap_options(int8_t *ssid, uint8_t chno, rsi_rsp_scan_t *result,uint32_t
length,uint32_t scan_bitmap)
```

Description

This function is used to scan the access points available and posts the scan response to application. Application should call this api to get the scan results.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
ssid	This is the SSID of an access point to connect. The SSID should be less than or equal to 32 bytes. **Note**: SSID is a null terminated string.
chno	Channel number of access point.
result	buffer address provided by the application to fill the scan response.
length	This is the length of the resulted buffer measured in bytes to hold scan results.Size of structure rsi_rsp_scan_t is 54 bytes. In which length for storing one scan result is sizeof(rsi_scan_info_t) which is 46 bytes. Maximum of 11 scan results will be returned by the module, in this case length should be configured as 8 + 11*sizeof(rsi_scan_info_t).
scan_bitmap	scan bitmap options

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure

Example

```
status= rsi_wlan_scan_with_bitmap_options(int8_t *ssid, uint8_t chno, rsi_rsp_scan_t *result,uint32_t
length,uint32_t scan_bitmap)
```

9.1.4 rsi_wlan_scan_async_with_bitmap_options

Prototype

```
int32_t rsi_wlan_scan_async_with_bitmap_options(int8_t *ssid, uint8_t chno,uint32_t bitmap,
                                                void (*scan_response_handler)(uint16_t status, const
uint8_t *buffer, const uint16_t length))
```

Description

This function is used to scan the access points available and posts the scan response to application. Application should call this api to get the scan results.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
ssid	This is the SSID of an access point to connect. The SSID should be less than or equal to 32 bytes.
chno	Channel number of access point.
result	buffer address provided by the application to fill the scan response.
length	This is the length of the resulted buffer measured in bytes to hold scan results.Size of structure rsi_rsp_scan_t is 54 bytes. In which length for storing one scan result is sizeof(rsi_scan_info_t) which is 46 bytes. Maximum of 11 scan results will be returned by the module, in this case length should be configured as 8 + 11*sizeof(rsi_scan_info_t).
scan_bitmap	scan feature bitmap

Return Values

Value	Description
0	Successful execution of the command.
Non-Zero Value	Failure

Example

```
status =  rsi_wlan_scan_async_with_bitmap_options(int8_t *ssid, uint8_t chno,uint32_t bitmap,
                                                  void (*scan_response_handler)(uint16_t status, const
uint8_t *buffer, const uint16_t length))
```

9.1.5 rsi_wlan_connect

Prototype

```
int32_t rsi_wlan_connect(int8_t *ssid,
                 rsi_security_mode_t sec_type,
                 void *secret_key)
```

Description
This API connects to the specified Wi-Fi network.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
ssid	This is the SSID of an access point to connect. The SSID should be less than or equal to 32 bytes.
sec_type	This is the security type of the access point to connect. 0: RSI_OPEN, 1: RSI_WPA, 2: RSI_WPA2, 3: RSI_WEP, 4: RSI_WPA_EAP, 5: RSI_WPA2_EAP, 6: RSI_WPA_WPA2_MIXED, 7: RSI_WPA_PMK, 8: RSI_WPA2_PMK, 9: RSI_WPS_PIN,

Parameter	Description
	10: RSI_USE_GENERATED_WPSPIN, 11: RSI_WPS_PUSH_BUTTON,
secret_key	This is the pointer to a buffer that contains security information based on sec_type.

Security type (sec_type)	secret key structure format (secret_key)
RSI_OPEN	No secret key in open security mode.
RSI_WPA	PSK string terminated with NULL. Length of PSK should be at least 8 and less than 64 bytes.
RSI_WPA2	PSK string terminated with NULL. Length of PSK should be at least 8 and less than 64 bytes.
RSI_WEP	WEP keys should be in the following format typedef struct rsi_wep_keys_s { uint8_t index[2]; uint8_t key[4][32]; } rsi_wep_keys_t; **index:** WEP key index to use for TX packet encryption. **key:** 4 WEP keys, last three WEP keys are optional. If only first WEP key is valid then index should be 0.
RSI_WPA_EAP	Enterprise credentials should be in the following format typedef struct rsi_eap_credentials_s { uint8_t username[64]; uint8_t password[128]; } rsi_eap_credentials_t; **username:** username to be used in enterprise **password:** password for the given username
RSI_WPA2_EAP	Enterprise credentials should be in the following format typedef struct rsi_eap_credentials_s { uint8_t username[64]; uint8_t password[128]; } rsi_eap_credentials_t; **username:** username to be used in enterprise **password:** password for a given username.
RSI_WPA_WPA2_MIXED	PSK string terminated with NULL. Length of PSK should be at least 8 and less than 64 bytes.
RSI_WPA_PMK	PMK string, should be 32 bytes in length
RSI_WPA2_PMK	PMK string, should be 32 bytes in length
RSI_WPS_PIN	8 bytes WPS PIN
RSI_USE_GENERATED_WPSPIN	NULL string indicates to use PIN generated using rsi_wps_generate_pin API.
RSI_WPS_PUSH_BUTTON	NULL string indicates to generate push button event.

Note:

To set various timeouts, user should change the following macros in rsi_wlan_config.h

```
#define RSI_TIMEOUT_SUPPORT       RSI_ENABLE
#define RSI_TIMEOUT_BIT_MAP       1
#define RSI_TIMEOUT_VALUE         300
```

timeout_bitmap[0]	sets timeout for association and authentication request. timeout_value: timeout value in ms(default 300ms).
timeout_bitmap[31-1]	reserved

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0002,0x0003,0x0005,0x0008,0x0009,0x000A,0x000E,0x0014, 0x0015,0x0016,0x0019,0x001A,0x001E,0x0020,0x0021,0x0024, 0x0025,0x0026,0x0028,0x0039,0x003C,0x0044,0x0045,0x0046, 0x0047,0x0048,0x0049,0xFFF8

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! WC initialization
status = rsi_wireless_init(9, 0);
//! Connect to an Access point
status = rsi_wlan_connect((int8_t *)SSID, STA_SECURITY_TYPE, STA_PSK);
if(status != RSI_SUCCESS)
{
   return status;
}
```

9.1.6 rsi_wlan_connect_async

Prototype

```
int32_t rsi_wlan_connect_async(int8_t *ssid, rsi_security_mode_t sec_type, void *secret_key, void
(*join_response_handler)(uint16_t status,const uint8_t *buffer,const uint16_t length))
```

Description
This API connects to the specified Wi-Fi network. A join response handler is registered to get the response for join.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
ssid	This is the SSID of an access point to connect. The SSID should be less than or equal to 32 bytes.

Parameter	Description
sec_type	This is the security type of the Access point to connect. 0: RSI_OPEN, 1: RSI_WPA, 2: RSI_WPA2, 3: RSI_WEP, 4: RSI_WPA_EAP, 5: RSI_WPA2_EAP, 6: RSI_WPA_WPA2_MIXED, 7: RSI_WPA_PMK, 8: RSI_WPA2_PMK, 9: RSI_WPS_PIN, 10: RSI_USE_GENERATED_WPSPIN, 11: RSI_WPS_PUSH_BUTTON,
secret_key	This is the pointer to a buffer that contains security information based on sec_type.
join_response_handler	This callback is called when the response for join has been received from the module The parameters involved are status, buffer & length **Status:** This is the response status. If status is zero, then the join response is stated as success **Buffer:** This is the response buffer. On successful execution of the command. GO_Status (1 byte, hex): 0x47 (ASCII "G") – If the module becomes a Group Owner (GO) after the GO negotiation stage or becomes an Access Point. 0x43 (ASCII "C") – If the module does not become a GO after the GO negotiation stage or becomes a client (or station). **Length:** This is the response buffer length.

Note: The module gets a default IP of 192.168.100.76 if it becomes a Group Owner or Access Point in case of IPv4. and gets a default IP of 2001:db8:0:1:0:0:0:120 in case of IPv6.

Security type (sec_type)	secret key structure format (secret_key)
RSI_OPEN	No secret key in open security mode.
RSI_WPA	PSK string terminated with NULL. Length of PSK should be at least 8 and less than 64 bytes.
RSI_WPA2	PSK string terminated with NULL. Length of PSK should be at least 8 and less than 64 bytes.
RSI_WEP	WEP keys should be in the following format typedef struct rsi_wep_keys_s { uint8_t index[2]; uint8_t key[4][32]; }rsi_wep_keys_t; **index:** WEP key index to use for TX packet encryption. **key:** 4 WEP keys. The last three WEP keys are optional. If only first WEP key is valid then the index should be 0.
RSI_WPA_EAP	Enterprise credentials should be in the following format typedef struct rsi_eap_credentials_s { uint8_t username[64]; uint8_t password[128]; }rsi_eap_credentials_t;

Security type (sec_type)	secret key structure format (secret_key)
	username: Value username to be used in enterprise. **password:** password for a given username.
RSI_WPA2_EAP	Enterprise credentials should be in the following format typedef struct rsi_eap_credentials_s { uint8_t username[64]; uint8_t password[128]; }rsi_eap_credentials_t; **username:** username to be used in enterprise. **password:** password for a given username.
RSI_WPA_WPA2_MIXED	PSK string terminated with NULL. The Length of PSK should be at least 8 and less than 64 bytes.
RSI_WPA_PMK	PMK string, should be 32 bytes in length.
RSI_WPA2_PMK	PMK string, should be 32 bytes in length.
RSI_WPS_PIN	8 bytes WPS PIN
RSI_USE_GENERATED_WPSPIN	NULL string indicates to use PIN generated using rsi_wps_generate_pin API.
RSI_WPS_PUSH_BUTTON	NULL string indicates to generate push button event.

Return Values

Value	Description
0	Successful execution of the command.
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0002,0x0003,0x0005,0x0008,0x0009,0x000A,0x000E,0x0014, 0x0015,0x0016,0x0019,0x001A,0x001E,0x0020,0x0021,0x0024, 0x0025,0x0026,0x0028,0x0039,0x003C,0x0044,0x0045,0x0046, 0x0047,0x0048,0x0049,0xFFF8

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
// user-implemented callback
void rsi_join_handler(uint16_t status,const uint8_t *buffer, const uint16_t length)
{
if(status == 0)
{
state = JOIN_DONE;
}
}
status = rsi_wlan_connect_async((int8_t *)SSID, SECURITY_TYPE, PSK,rsi_join_handler);
```

9.1.7 rsi_wlan_bgscan_profile

Prototype

```
int32_t rsi_wlan_bgscan_profile(uint8_t cmd,rsi_rsp_scan_t *result,uint32_t length );
```

Description
This API is used to get background scan results or stop bgscan.

Precondition
rsi_wlan_connect() API needs to be called before this API.

Parameters

parameter	description
input	result
output	length

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is less than 0** -3: Command given in wrong state -4: Buffer not available to serve the command

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_wlan_bgscan_profile(cmd,(rsi_rsp_scan_t *)result,length);
```

9.1.8 rsi_wlan_execute_post_connect_cmds

Prototype

```
int32_t rsi_wlan_execute_post_connect_cmds(void)
```

Description
This API enables Bgscan and roaming after connecting to the Access Point.

Precondition
rsi_wlan_connect() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0006,0x0021,0x002C,0x004A,0x0025,0x0026

Note:

Please refer to Error Codes section for the description of the above error codes.

9.1.9 rsi_wlan_disconnect

Prototype

```
int32_t rsi_wlan_disconnect(void);
```

Description
This API disconnects the module from the connected Access point.

Precondition
rsi_wlan_connect() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	Failure **If return value is less than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0006,0x0021,0x002C,0x004A,0x0025,0x0026

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_wlan_disconnect(void);
```

9.1.10 rsi_wlan_enable_auto_config

Prototype

```
int32_t rsi_wlan_enable_auto_config(uint8_t enable, uint32_t type);
```

Description

This function is used to send raw data packet to module.

Precondition

None

Parameters

Parameter	Description
enable	to enable/disable the profile configuration, first parameter value would be 1(CFG_ENABLE)- means This command is used to enable or disable the feature of auto-join , auto-create on power up and 2 (enable_auto_config) - for profile based.

Parameter	Description
type	Profile type **Note**: Possible profile types. //! Client profile #define RSI_WLAN_PROFILE_CLIENT 0 //! P2P profile #define RSI_WLAN_PROFILE_P2P 1 //! EAP profile #define RSI_WLAN_PROFILE_EAP 2 //! AP profile #define RSI_WLAN_PROFILE_AP 6 //! All profiles #define RSI_WLAN_PROFILE_ALL 0xF
output	returns success/failure as a response.

Return Values

Value	Description
0	Successful execution of the command.
Non-Zero Value	Failure **If return value is less than 0** -4: Buffer not available to serve the command.

Example

```
status = rsi_wlan_enable_auto_config(enable,type);
```

9.1.11 rsi_wlan_pmk_generate

Prototype

```
int32_t rsi_wlan_pmk_generate(int8_t type,int8_t *psk,int8_t *ssid, uint8_t *pmk, uint16_t length);
```

Description

This API generates PMK using PSK and SSID are provided.

Precondition

rsi_wlan_connect() API needs to be called before this API.

Parametersrs

Parameter	Description
type	possible values of this field are 1, 2 and 3, but we only pass 3 for generation of PMK.
psk	In this field expected parameters are pre shared key(psk) of the access point to which module wants to associate.
ssid	This field contains the SSID of the access point, this field will be valid only if TYPE value is 3.
pmk	this is an array
length	length of pmk array

Return Values

Value	Description
0	Successful execution of the command. If TYPE value is '3'.
Non-Zero Value	Failure **If return value is greater than 0** 0x0021, 0x0025,0x0026,0x0028,0x002C,0x0039,0x003a, 0x003b

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_wlan_pmk_generate(PMK_TYPE, (int8_t *)PSK, (int8_t *)SSID, pmk, PMK_SIZE)
```

9.1.12 rsi_wlan_set_certificate_index

Prototype

```
int32_t rsi_wlan_set_certificate_index(uint8_t certificate_type, uint8_t cert_inx, uint8_t *buffer,
uint32_t certificate_length)
```

Description
This function is used to write or erase certificate into module.

Precondition

rsi_wireless_init() API must be called before this API.

Parameters

Parameter	Description
Certificate_type	This the type of certificate
cert_inx	Index of certificate Possible values: 0 and 1 for loading onto RAM Possible values: 0, 1 and 2 for loading onto FLASH
buffer	Certificate content
Certificate_length	This is the certificate length

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0015,0x0021,0x0025,0x0026,0x002C

Example

```
status = rsi_wlan_set_certificate_index(certificate_type, 0, buffer, certificate_length);
```

Note:

1. Index based certificate loading is valid only for storing certificates on to ram or flash but not both at a time.
2. Enable BIT(27) in tcp_ip_feature_bit_map to load SSl certificate into RAM.
3. Enable BIT(31) in tcp_ip_feature_bit_map and BIT (29) in ext_tcp_ip_feature_bit_map to open 3 SSL client sockets.
4. Three SSL client socket feature supported only in WLAN mode.

9.1.13 rsi_wlan_set_certificate

Prototype

```
int16_t rsi_wlan_set_certificate(uint8_t certificate_type,  uint8_t *buffer,  uint32_t
certificate_length);
```

Description
This API loads SSL / EAP certificate on WiSeConnect module.

Precondition
rsi_wireless_init() API must be called before this API.

Parameters

Parameter	Description
Certificate_type	This the type of certificate 1: TLS client certificate 2: FAST PAC file 3: SSL Client Certificate 4: SSL Client Private Key 5: SSL CA Certificate 6: SSL Server Certificate 7: SSL Server Private Key
buffer	This is the pointer to a buffer which contain certificate
certificate_length	This is the certificate length

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0015,0x0021,0x0025,0x0026,0x002C

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
// set EAP client certificate
status = rsi_wlan_set_certificate(RSI_EAP_CLIENT, Wi-Fiuser, (sizeof(Wi-Fiuser)-1));
```

9.1.14 rsi_wlan_get_status

Prototype

```
int32_t rsi_wlan_get_status(void);
```

Description
This API checks the status (specific error code) of the errors encountered during a call to a WLAN API or BSD sockets functions. User can call this API to check the error code (refer error code table for description of the errors).

Precondition
None

Parameters
None

Return Values
Returns the error code that previously occurred. If no error occurred, then it returns 0.

Example

```
// query the status
int32_t status = rsi_wlan_get_status();
```

9.1.15 rsi_wlan_update_gain_table

Prototype

```
int32_t rsi_wlan_update_gain_table(uint8_t band, uint8_t bandwidth,  uint8_t *payload, uint16_t
payload_len);
```

Description
This API assigns the user configurable channel gain values in different regions to the module from user. This method is used for overwriting default gain tables that are present in firmware. Customer can load all the three gain tables (i.e., 2.4GHz-20Mhz, 5GHz-20Mhz) one after other by changing band and bandwidth values.

Note:

This frame must be used by customers who has done FCC/ETSI/KCC certification with their own antenna. All other customers should not use this. Inappropriate use of this frame may result in violation of FCC/ETSI/KCC or any certifications and Silicon labs is not liable for that.

Precondition
rsi_radio_init() API needs to be called before this API.

Parameters

Parameter	Description
Band	0 → 2.4GHz 1 → 5GHz 2 → Dual band (2.4 GHz and 5 GHz)
Bandwidth	0 → 20 MHz 1 → 40 MHz
payload	Pass channel gain values for different regions in a given array format.
payload_len	Max payload length (table size) in 2.4GHz is 128 bytes Max payload length (table size) in 5GHz is 64 bytes

Gain Table Payload Format:

1. Gain table Format for 2.4G Band: (Each entry of the table is 1 byte)

```
<TABLE NAME>[] = {
                        <NO.of Regions>,
                        <REGION NAME 1>, <CHANNEL_CODE_2G>,
                        <CHANNEL NUMBER 1>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER
FOR n RATE>,
                        <CHANNEL NUMBER 2>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER
FOR n RATE>,
                         .
                         .
                         .
                         .
                         .
                        <CHANNEL NUMBER m-1>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER
FOR n RATE>,
                        <CHANNEL NUMBER m>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER
FOR n RATE>,
                        <REGION NAME 2>, <CHANNEL_CODE_2G>,
                        <CHANNEL NUMBER 1>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER
FOR n RATE>,
                        <CHANNEL NUMBER 2>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER
FOR n RATE>,
                        .
                        .
                        .
                        .
                        <CHANNEL NUMBER m-1>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER
FOR n RATE>,
                        <CHANNEL NUMBER m>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER
FOR n RATE>,
                        <REGION NAME 3>, <CHANNEL_CODE_2G>,
                        .
                        .
                        .
                        .
                        <REGION NAME y>, <CHANNEL_CODE_2G>,
                        };
```

2. Gain table Format for 5G Band: (Each entry of the table is 1 byte)

```
   <TABLE NAME>[] = {
                        <NO. of Regions>,
                        <REGION NAME 1>, <CHANNEL_CODE_5G>,
                        <CHANNEL NUMBER IN BAND 1 IF ANY>, <MAX POWER FOR g RATE>, <MAX POWER FOR n
RATE>,
                         <BAND_NUMBER 1>, <MAX POWER FOR g RATE>, <MAX POWER FOR n RATE>,
                        <CHANNEL NUMBER IN BAND 2 IF ANY>, <MAX POWER FOR g RATE>, <MAX POWER FOR n
RATE>,
                        <BAND_NUMBER 2>, <MAX POWER FOR g RATE>, <MAX POWER FOR n RATE>,
                        <CHANNEL NUMBER IN BAND 3 IF ANY>, <MAX POWER FOR g RATE>, <MAX POWER FOR n
RATE>,
                        <BAND_NUMBER 3>, <MAX POWER FOR g RATE>, <MAX POWER FOR n RATE>,
                        <CHANNEL NUMBER IN BAND 4 IF ANY>, <MAX POWER FOR g RATE>, <MAX POWER FOR n
RATE>,
                        <BAND_NUMBER 4>, <MAX POWER FOR g RATE>, <MAX POWER FOR n RATE>,
                        .
                        .
                        .
                        .
                        .
                        <REGION NAME y>, <CHANNEL_CODE_5G>,
                        };
```

3. Supported Region names: FCC, ETSI, TELEC, KCC

The following are the regions and their values to be passed instead of macros in the example.

Region	Macro Value
FCC	0
ETSI	1
TELEC	2
KCC	4

4. <CHANNEL_CODE_2G> is an 8-bit value which is encoded as:
   If Tx powers of all the channels are same, then use CHANNEL_CODE_2G as 17. In this case, mention channel number as 255.
   If Tx power is not same for all channels, then indicate CHANNEL_CODE_2G as no-of channels. And specify Tx power values for all the channels indicated.

5. <CHANNEL_CODE_5G> is a 8 bit value encoded as number of rows in a region for 5G band.
   a. 5G is divided into 4 sub bands:
      band 1: channel number <= 48
      band 2: channel number > 48 and channel number <= 64
      band 3: channel number > 64 and channel number <= 144
      band 4: channel number > 144
   b. If any channel in a band has different set of power values, specify the channel number followed by power values.
   c. If all the channels in a band 1 has same power values, specify the band number as 1 followed by power value.
   d. If all the channels in a band 2 has same power values, specify the band number as 2 followed by power value.
   e. If all the channels in a band 3 has same power values, specify the band number as 3 followed by power value.
   f. If all the channels in a band 4 has same power values, specify the band number as 4 followed by power value.

Example payload formats:

Examples:
For 2.4GHz Band in 20MHz bandwidth:

```
{3, //NUM_OF_REGIONS
    FCC, 13, //NUM_OF_CHANNELS
//   rate,  11b, 11g, 11n
        1,  17,  10,  10,
        2,  17,  14,  14,
        3,  17,  16,  16,
        4,  17,  18,  18,
        5,  17,  19,  19,
        6,  17,  20,  20,
        7,  17,  19,  19,
        8,  17,  18,  18,
        9,  17,  16,  16,
        10, 17,  16,  16,
        11, 17,  12,  12,
        12, 17,   8,  12,
        13, 17,   6,   6,
    TELEC, 17,
         255, 10,  8, 8,
    KCC, 17,
         255, 13,  10, 10,
}; //}}}
```

For 5GHz band in 20MHz bandwidth:

```
{2,
FCC, 6,
    1,  9, 10, //band 1
    2,  8,  9, //band 2
  100,  4,  4, //band 3
    3,  6,  8, //band 3
  149,  3,  3, //band 4
    4,  6,  7, //band 4
TELEC, 4,
  1, 9, 10, //band 1
  2, 8, 10, //band 2
  3, 6,  8, //band 3
  4, 6,  7, //band 4
};
```

For 5GHz band in 40MHz bandwidth:

```
{2,
FCC, 8,
    1,  9, 10, //band 1
   62,  8,  9, //band 2
    2,  8,  9, //band 2
  102,  4,  4, //band 3
  134,  6,  8, //band 3
    3,  6,  8, //band 3
  151,  3,  3, //band 4
    4,  6,  7, //band 4
TELEC, 4,
   1, 9, 10, //band 1
   2, 8, 10, //band 2
   3, 6,  8, //band 3
   4, 6,  7, //band 4
};
```

Customers using Certified MARS antenna should use the below mentioned gain table structures
/******2GHz-20MHz*****/

```
{3,//NUM_OF_REGIONS
    FCC, 0xD,//NUM_OF_CHANNELS
//   rate,  11b, 11g, 11n
        1,  14,  16,  15,
        2,  14,  16,  15,
        3,  14,  16,  15,
        4,  15,  14,  17,
        5,  15,  14,  17,
        6,  15,  14,  17,
        7,  15,  14,  17,
        8,  15,  14,  17,
        9,  14,  15,  15,
        10, 14,  15,  15,
        11, 14,  15,  15,
        12, 14,  15,  15,
        13, 14,  15,  15,
    TELEC,0x11, //NA
         255, 10,  8, 8,
    KCC, 0x11   //NA,
         255, 13,  10, 10
};
```

/******5GHz-20MHz*****/

```
{2,
FCC, 0x6,
    1, 12, 12, //band 1
    2, 11, 11, //band 2
  100, 10, 12, //band 3
    3, 13, 13, //band 3
  140, 10, 11, //band 4
    4, 13, 13, //band 4
TELEC, 0x4, //NA
  1, 9, 10, //band 1
  2, 8, 10, //band 2
  3, 6,  8, //band 3
  4, 6,  7, //band 4
};
```

/******5GHz-40MHz*****/

```
{2,
FCC, 0x8,
    1,  9,  9, //band 1
   62,  8,  8, //band 2
    2,  9,  9, //band 2
  102,  9,  9, //band 3
  134, 12, 12, //band 3
    3, 10, 10, //band 3
  151, 11, 11, //band 4
    4, 11, 11, //band 4
TELEC, 0x4, //NA
   1, 9, 10, //band 1
   2, 8, 10, //band 2
   3, 6,  8, //band 3
   4, 6,  7, //band 4
};
```

Note:

1. Length of the payload should match with payload_len parameter value.
2. 40Mhz is not supported.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state **If return value is greater than 0** 0x0021,0x003E

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_wlan_update_gain_table(BAND, BANDWIDTH, payload,PAYLOAD_LEN);
```

9.1.16 rsi_wlan_ap_start

Prototype

```
int32_t rsi_wlan_ap_start(int8_t *ssid, uint8_t channel, rsi_security_mode_t security_type,
rsi_encryption_mode_t encryption_mode,  uint8_t *password,
                          uint16_t beacon_interval,  uint8_t dtim_period)
```

Description
This API starts the module in Access point mode with the given configuration.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
ssid	This is the SSID of the Access Point. The length of the SSID should be less than or equal to 32 bytes.
channel	This is the channel number. Refer the following channels for the valid channel numbers supported. 2.4GHz Band Channel Mapping 5GHz Band Channel Mapping channel number 0 is to enable ACS feature
security_type	This is the type of the security modes on which an access point needs to be operated. 0: RSI_OPEN 1: RSI_WPA 2: RSI_WPA2 6: RSI_WPA_WPA2_MIXED 11: RSI_WPS_PUSH_BUTTON
encryption_mode	This is the type of the encryption mode 0: RSI_NONE 1: RSI_TKIP 2: RSI_CCMP
password	This is the PSK to be used in security mode. Minimum and maximum length of PSK is 8 bytes and 63 bytes respectively.
beacon_interval	This is the beacon interval
dtim_period	This is the DTIM period

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x0025,0x002C,0x0026,0x004C,0x0028,0x001A,0x000A,0x001D

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
 //! WC initialization
 status = rsi_wireless_init(6, 0);
 if(status != RSI_SUCCESS)
 {
 return status;
 }
 //! Configure IP

 status = rsi_config_ipaddress(RSI_IP_VERSION_4, RSI_STATIC, (uint8_t*)&ip_addr, (uint8_t
*)&network_mask, (uint8_t *)&gateway,
                               NULL, 0,0);
 if(status != RSI_SUCCESS)
 {
 return status;
 }
 //! Start Access point
 status = rsi_wlan_ap_start((int8_t *)SSID, CHANNEL_NO, SECURITY_TYPE, ENCRYPTION_TYPE, PSK,
BEACON_INTERVAL, DTIM_INTERVAL);
 if(status != RSI_SUCCESS)
 {
 return status;
 }
```

9.1.17 rsi_wlan_wps_push_button_event

Prototype

```
int32_t rsi_wlan_wps_push_button_event(int8_t *ssid);
```

Description
This API starts the WPS Push button in AP mode. It should be called after rsi_wlan_ap_start API once it has returned success.

Precondition
rsi_wlan_ap_start() API needs to be called before this API.

Parameters

Parameter	Description
ssid	This is the SSID of the Access Point. The SSID should be same as that of given in AP start API. The length of the SSID should be less than or equal to 32 bytes.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
int32_t rsi_wlan_wps_push_button_event(int8_t *ssid);
```

9.1.18 rsi_wlan_wps_generate_pin

Prototype

```
int32_t rsi_wlan_wps_generate_pin( uint8_t *wps_pin, uint16_t length);
```

Description
This API generates WPS pin.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
wps_pin	This is the 8-byte WPS pin generated by the device. This is the output parameter.
length	This is the length of the resulted buffer measured in bytes to hold WPS pin.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002C, 0x0037, 0x0038

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
int32_t rsi_wlan_wps_push_button_event(int8_t *ssid);
```

9.1.19 rsi_wlan_wps_enter_pin

Prototype

```
int32_t rsi_wlan_wps_enter_pin(int8_t *wps_pin)
```

Description
This function is used to validate wps pin entered.

Precondition
rsi_wlan_init() API needs to be called before this API.

Parameter

parameter	description
input	wps_pin , 8 byte valid wps pin
output	same wps pin if command success

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure

Example

```
int32_t rsi_wlan_wps_enter_pin(wps_pin);
```

9.1.20 rsi_get_random_bytes

Prototype

```
int32_t rsi_get_random_bytes(uint8_t *result,uint32_t length)
```

Description
This function is used to get random bytes from the device.

Parameters

Parameter	Description
input	result, pointer to the buffer in which random data which is to be copied.
output	length, the length of the random data required.

Return Values

Value	Description
0	Successful execution of the command
<0	Failure

Example

```
int32_t rsi_get_random_bytes(result, length);
```

9.1.21 rsi_wlan_disconnect_stations

Prototype

```
int32_t rsi_wlan_disconnect_stations( uint8_t *mac_address);
```

Description
This API disconnects the connected stations in AP mode.

Precondition
rsi_wlan_ap_start() API needs to be called before this API.

Parameters

Parameter	Description
mac_address	Mac address (6 bytes) of the station to be disconnected.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0013, 0x0021, 0x002C, 0x0015

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
int32_t rsi_wlan_disconnect_stations(station_mac_address);
```

9.1.22 rsi_wlan_get

Prototype

```
int32_t rsi_wlan_get( rsi_wlan_query_cmd_t cmd_type, uint8_t *response, uint16_t length);
```

Description
This API gets the required information based on the type of command.

Precondition
rsi_wireless_init () API needs to be called before this API.

Parameters

Parameter	Description
cmd_type	This is the Query command type: 1: RSI_FW_VERSION 2: RSI_MAC_ADDRESS 3: RSI_RSSI 4: RSI_WLAN_INFO 5: RSI_CONNECTION_STATUS 6: RSI_STATIONS_INFO 7: RSI_SOCKETS_INFO 8:RSI_CFG_GET 9:RSI_GET_WLAN_STATS
response	This is the the response of the requested command. This is an output parameter.
length	This is the length of the response buffer in bytes to hold result.

cmd_type	Response structure
RSI_FW_VERSION	uint8_t response[20] **Note**: RSI_FW_VERSION should be used after successful rsi_wireless_init().
RSI_MAC_ADDRESS	uint8_t response[6]

cmd_type	Response structure
	Note: RSI_MAC_ADDRESS should be used after successful rsi_wlan_scan() API.
RSI_RSSI	uint8_t response[2] **Note**: RSI_RSSI should be used after successful WLAN connection.

RSI_WLAN_INFO	`typedef struct rsi_rsp_wireless_info_s`
	`{`
	`uint16_t wlan_state;`
	`uint16_t channel_number;`
	`uint8_t ssid[32];`
	`uint8_t mac_address[6];`
	`uint8_t sec_type;`
	`uint8_t psk[64];`
	`uint8_t ipv4_address[4];`
	`uint8_t ipv6_address[16];`
	`uint8_t reserved1[2];`
	`uint8_t reserved2[2];`
	`} rsi_rsp_wireless_info_t;`
	
	wlan_state:
	In station mode - Connected / Unconnected state
	In AP mode - No of stations connected information
	channel_number :
	In station mode - Channel in which station is associated
	In AP mode - Channel in which device is acting as AP
	ssid :
	In station mode - SSID of AP associated to.
	In AP mode - Device SSID
	mac_address: Mac address of the device
	sec_type :
	In station mode – security type of AP
	In AP mode – NA
	psk: 63 bytes of PSK
	ipv4_address : IPv4 address of the device
	ipv6_address : IPv6 address of the device
	reserved1 : reserved field
	reserved2 : reserved field
	Note:
	RSI_WLAN_INFO should be used after successful WLAN connection.

cmd_type	Response structure

RSI_STATIONS_INFO	typedef struct rsi_rsp_stations_info_s
	{
	uint8_t sta_count[2];
	rsi_go_sta_info_t sta_info[8];
	}
	sta_count : No of stations connected
	sta_info : structure holding stations information
	typedef struct rsi_go_sta_info_s
	{
	uint8_t ip_version[2];
	uint8_t mac[6];
	union
	{
	uint8_t ipv4_address[4];
	uint8_t ipv6_address[16];
	}ip_address;
	}rsi_go_sta_info_t;
	ip_version : IP version
	4 - IPv4
	6 - IPv6
	mac : Mac address of connected station
	ipv4_address[4] &
	ipv6_address[16]: Union of IPv4 and IPv6 address of connected stations.
	
	Note:
	This command type is used to get connected WLAN stations information AP mode. This should be used after successful WLAN connection.
	If no stations are connected to AP, then the response will be zeros.

RSI_SOCKETS_INFO	typedef struct rsi_rsp_sockets_info_s
	{
	uint8_t num_open_socks[2];
	rsi_sock_info_query_t socket_info[10];
	} rsi_rsp_sockets_info_t;
	num_open_socks : Number of sockets opened
	socket_info : Each Socket information in below structure format.
	typedef struct rsi_sock_info_query_s
	{
	uint8_t sock_id[2];
	uint8_t sock_type[2];
	uint8_t source_port[2];
	uint8_t dest_port[2];
	union{
	uint8_t ipv4_address[4];
	uint8_t ipv6_address[16];
	}dest_ip_address;
	}rsi_sock_info_query_t;
	sock_id : Socket descriptor
	sock_type : Socket type
	0 - TCP/SSL client
	2 - TCP/SSL server
	4 - Listening UDP
	source_port : Port number of socket in the module
	dest_port : Destination port of remote peer
	ipv4_address[4] &
	ipv6_address[16] : Union of IPv4 and IPv6 address. In case of IPv4 address , only 4 bytes are filled , remaining are zeroes
	
	Note:

cmd_type	Response structure
	This command type can be used to get the socket information. 1. For TCP client, this should be called after successful socket creation. 2. For TCP server, this should be called after listen() API. 3. For UDP this should be called after below API's. i)sendto() API ii)bind() API. If no sockets are created, then the response will be zeros.

RSI_GET_WLAN_STATS	`typedef struct rsi_rsp_wlan_stats_s{`
	`uint8_t operating_mode;`
	`uint8_t dtim_period;`
	`uint8_t ideal_beacon_info[2];`
	`uint8_t busy_beacon_info[2];`
	`uint8_t beacon_interval[2];`
	`} rsi_rsp_wlan_stats_t;`
	`operating_mode: Wifi Client mode`
	`dtim_period: DTIM stands for Delivery traffic indication map or message. The DTIM interval (1-255) means the peroid of time to wake up wireless clients from Sleep Mode.`
	`ideal_beacon_info: The number of beacons without multicast and broadcast indication in TIM field.`
	`busy_beacon_info: The number of beacons with multicast and broadcast indication in TIM field.`
	`beacon_interval: Beacon Broadcast interval is the time lag between each of the beacons sent by your router or access points.`
	Note:
	RSI_GET_WLAN_STATS should be used after successful rsi_wireless_init().

RSI_CONNECTION_STATUS	It returns 0x0001 if the WLAN is connected else 0x0000 **Note:** RSI_CONNECTION_STATUS should be used after successful rsi_wireless_init().
RSI_CFG_GET	This command type is used to get the stored configurations from flash.

cmd_type	Response structure

```
//! structure for cfg_get
typedef struct rsi_cfgGetFrameRcv{
uint8_t cfg_enable;
uint8_t opermode[4];
uint8_t feature_bit_map[4];
uint8_t tcp_ip_feature_bit_map[4];
uint8_t custom_feature_bit_map[4];
uint8_t band;
uint8_t scan_feature_bitmap;
uint8_t join_ssid[RSI_SSID_LEN];
uint8_t uRate;
uint8_t uTxPower;
uint8_t join_feature_bitmap;
uint8_t reserved_1;
uint8_t scan_ssid_len;
uint8_t reserved_2;
uint8_t csec_mode;
uint8_t psk[RSI_PSK_LEN];
uint8_t scan_ssid[RSI_SSID_LEN];
uint8_t scan_cnum;
uint8_t dhcp_enable;
uint8_t ip[IP_ADDRESS_SZ];
uint8_t sn_mask[IP_ADDRESS_SZ];
uint8_t dgw[IP_ADDRESS_SZ];
uint8_t eap_method[32];
uint8_t inner_method[32];
uint8_t user_identity[64];
uint8_t passwd[128];
uint8_t go_intent[2];
uint8_t device_name[64];
uint8_t operating_channel[2];
uint8_t ssid_postfix[64];
uint8_t psk_key[64];
uint8_t pmk[WISE_PMK_LEN];
rsi_apconfig apconfig;
uint8_t module_mac[6];
uint8_t antenna_select[2];
uint8_t reserved_3[2];
rsi_wepkey wep_key;
uint8_t dhcp6_enable[2];
uint8_t prefix_length[2];
uint8_t ip6[16];
uint8_t dgw6[16];
uint8_t tcp_stack_used;
uint8_t bgscan_magic_code[2];
uint8_t bgscan_enable[2];
uint8_t bgscan_threshold[2];
uint8_t rssi_tolerance_threshold[2];
uint8_t bgscan_periodicity[2];
uint8_t active_scan_duration[2];
```

cmd_type	Response structure

```
uint8_t passive_scan_duration[2];
uint8_t multi_probe;
uint8_t chan_bitmap_magic_code[2];
uint8_t scan_chan_bitmap_stored_2_4_GHz[4];
uint8_t scan_chan_bitmap_stored_5_GHz[4];
uint8_t roam_magic_code[2];
rsi_uRoamParams roam_params_stored;
uint8_t rejoin_magic_code[2];
rsi_rejoin_params_t rejoin_param_stored;
uint8_t region_request_from_host;
uint8_t rsi_region_code_from_host;
uint8_t region_code;
uint8_t reserved_4[43];
uint8_t multicast_magic_code[2];
uint8_t multicast_bitmap[2];
uint8_t powermode_magic_code[2];
uint8_t powermode;
uint8_t ulp_mode;
uint8_t wmm_ps_magic_code[2];
uint8_t wmm_ps_enable;
uint8_t wmm_ps_type;
uint8_t wmm_ps_wakeup_interval[4];
uint8_t wmm_ps_uapsd_bitmap;
uint8_t listen_interval[4];
uint8_t listen_interval_dtim;
uint8_t ext_custom_feature_bit_map[4];
uint8_t private_key_password[82];
uint8_t join_bssid[6];
uint8_t fast_psp_enable;
uint8_t monitor_interval[2];
uint8_t timeout_value[2];
uint8_t timeout_bitmap[4];
uint8_t request_timeout_magic_word[2];
rsi_uHtCaps ht_caps;
uint8_t ht_caps_magic_word[2];
//! AP IP parameters in Concurrent mode
uint8_t dhcp_ap_enable;
uint8_t ap_ip[4];
uint8_t ap_sn_mask[4];
uint8_t ap_dgw[4];
uint8_t dhcp6_ap_enable[2];
uint8_t ap_prefix_length[2];
uint8_t ap_ip6[16];
uint8_t ap_dgw6[16];
uint8_t ext_tcp_ip_feature_bit_map[4];
uint8_t http_credentials_avail;
uint8_t http_username[MAX_HTTP_SERVER_USERNAME];
uint8_t http_password[MAX_HTTP_SERVER_PASSWORD];
} rsi_cfgGetFrameRcv_t ,rsi_user_store_config_t;
```

Note:

This RSI_CFG_GET should be called after successful WLAN connection.

Once the CFG parameters are saved in flash memory of module we can give RSI_CFG_GET after wireless_init() API. It will print previously stored configurations.

Note:

RSI_WLAN_INFO is relevant in both station and AP mode.
RSI_SOCKETS_INFO is relevant in both station mode and AP mode.
RSI_STATIONS_INFO is relevant in AP mode.

RSI_GET_WLAN_STATS is relevant in AP and Station mode.

Return Value

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -3: Command given in wrong state -4: Buffer not available to serve the command -6: Insufficient input buffer given **If return value is greater than 0** 0x0021, 0x0025, 0x002c

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Get mac address
status = rsi_wlan_get(RSI_MAC_ADDRESS, &glbl_mac[0], 6);
```

9.1.23 rsi_wlan_set

Prototype

```
int32_t rsi_wlan_set( rsi_wlan_set_cmd_t cmd_type, uint8_t *request, uint16_t length);
```

Description
This API sets the requested configuration based on the command type.

Precondition
rsi_wireless_init() API needs to be called before this API. For setting MAC address, call this api immediately after rsi_wireless_init() and before calling any other api.

Parameters

Parameter	Description
cmd_type	Set command type: 1: RSI_SET_MAC_ADDRESS 2: RSI_MULTICAST_FILTER 3: RSI_JOIN_BSSID
request	Request buffer
length	Length of the request buffer in bytes

cmd_type	Request Structure
RSI_SET_MAC_ADDRESS	uint8_t mac_address[6]
RSI_MULTICAST_FILTER	typedef struct rsi_req_multicast_filter_info_s { uint8_t cmd_type; uint8_t mac_address[6];

cmd_type	Request Structure
	}rsi_req_multicast_filter_info_t; cmd_type : 1. RSI_ MULTICAST_MAC_ADD_BIT (To set particular bit in multicast bitmap) 2. RSI_ MULTICAST_MAC_CLEAR_BIT (To reset particular bit in multicast bitmap) 3. RSI_ MULTICAST_MAC_CLEAR_ALL (To clear all the bits in multicast bitmap) 4. RSI_ MULTICAST_MAC_SET_ALL (To set all the bits in multicast bitmap) mac_address : MAC address to which filter has to be applied
RSI_JOIN_BSSID	uint8_t join_bssid[6]

Return Value

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002c

Note:

Please refer to Error Codes section for the description of the above error codes.

9.1.24 rsi_wlan_ ping_async

Prototype

```
int32_t rsi_wlan_ping_async(uint8_t flags, uint8_t* ip_address, uint32_t size, void
(*wlan_ping_response_handler) (uint16_t status, const uint8_t *buffer, const uint16_t length))
```

Description
This API sends the ping request to the target IP address.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
flags	To select IP version and security BIT(0) – RSI_IPV6 Set this bit to enable IPv6; by default it is configured to IPv4
ip_address	target IP address IPv4 address – 4 Bytes hexa-decimal, IPv6 address – 16 Bytes hexa-decimal
size	ping data size to send. Maximum supported is 300 bytes.
wlan_ping_response_handler	This callback is called when ping response has been received from the module. The parameters involved are status, buffer & length. **Status**: This is the response status

Parameter	Description
	If status is zero, ping response is stated as success. **Buffer:** This is the response buffer. **Length:** This is the response buffer length.

Response Structure:

```
typedef struct rsi_rsp_ping_t
 {
 uint8_t ip_version[2];
 uint8_t ping_size[2];
 union
 {
 uint8_t ipv4_address[4];
 uint32_t ipv6_address[4];
 }ping_address;
 }rsi_rsp_ping_t;
```

Return Value

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2 : Invalid parameters -4 : Buffer not available to serve the command **If return value is greater than 0** 0x0015,0xBB21,0xBB4B,0xBB55

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
// ping remote device
status = rsi_wlan_ping_async(0, (uint8_t *)&remote_ip_addr, size, rsi_ping_response_handler);
```

9.1.25 rsi_register_auto_config_rsp_handler

Prototype

```
void rsi_register_auto_config_rsp_handler(void (*rsi_auto_config_rsp_handler )(uint16_t status, uint8_t state));
```

Description

This function is used to register Auto configuration response handler.

Precondition

None

Parameter

Parameter	Description
rsi_auto_config_rsp_handler	pointer to rsi_auto_config_rsp_handler
output	none

Return Value

None

Example

```
rsi_register_auto_config_rsp_handler(rsi_auto_config_response_handler);
```

9.1.26 rsi_wlan_add_profile

Prototype

```
int32_t rsi_wlan_add_profile(uint32_t type, uint8_t *profile);
```

Description

This function is used to add profile for auto configuration.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameter

Parameter	Description
type	profile type. Supported profile types are 1.RSI_WLAN_PROFILE_AP, 2. RSI_WLAN_PROFILE_CLIENT, 3.RSI_WLAN_PROFILE_EAP, 4. RSI_WLAN_PROFILE_P2P, 5.RSI_WLAN_PROFILE_ALL
profile	pointer to config profile. Config profile structure ap_profile, eap_client_profile_t, client_profile_t, p2p_profile_t, rsi_config_profile_t
output	returns success/failure as response

Return Value

Value	Description
0	Success
Non-Zero Value	Failure **If return value is less than 0** -4: Buffer not available to serve the command.

Example

```
status = rsi_wlan_add_profile(profile_type, wlan_profile_ptr);
```

9.1.27 rsi_wlan_get_state

Prototype

```
uint8_t rsi_wlan_get_state(void);
```

Description

This function is used to get the current wlan state.

Precondition

None

Parameter

Parameter	Description
input	none
output	returns the current wlan state. Wlan states are mentioned below RSI_WLAN_STATE_NONE = 0, RSI_WLAN_STATE_OPERMODE_DONE, RSI_WLAN_STATE_BAND_DONE, RSI_WLAN_STATE_INIT_DONE, RSI_WLAN_STATE_SCAN_DONE, RSI_WLAN_STATE_CONNECTED, RSI_WLAN_STATE_IP_CONFIG_DONE, RSI_WLAN_STATE_IPV6_CONFIG_DONE, RSI_WLAN_STATE_AUTO_CONFIG_GOING_ON, RSI_WLAN_STATE_AUTO_CONFIG_DONE, RSI_WLAN_STATE_AUTO_CONFIG_FAILED

Return Value

This function returns current wlan state.

Example

```
status = rsi_wlan_get_state();
```

9.1.28 rsi_wlan_get_profile

Prototype

```
int32_t rsi_wlan_get_profile(uint32_t type, rsi_config_profile_t *profile_rsp, uint16_t length);
```

Description

This function is used to get the stored config profile.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameter

Parameter	Description
type	config profile type. Supported profile types are RSI_WLAN_PROFILE_AP, RSI_WLAN_PROFILE_CLIENT, RSI_WLAN_PROFILE_EAP, RSI_WLAN_PROFILE_P2P,

Parameter	Description
	RSI_WLAN_PROFILE_ALL
profile-rsp	Config profile response in the form of below structure ap_profile, eap_client_profile_t, client_profile_t, p2p_profile_t, rsi_config_profile_t
length	Length of the config profile response
output	returns success/failure as response

Return Value

Value	Description
0	Success
Non-Zero Value	Failure **If return value is less than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_wlan_get_profile(type,(rsi_config_profile_t *)profile_rsp, length);
```

9.1.29 rsi_fill_config_profile

Prototype

```
uint8_t* rsi_fill_config_profile(uint32_t type, uint8_t *profile_buffer);
```

Description

This function is used to fill the config profile based on the profile type.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameter

Parameter	Description
type	profile type
profile-buffer	pointer to profile buffer
output	returns profile buffer

Return Value

This function returns profile_buffer.

Example

```
wlan_profile_ptr = rsi_fill_config_profile(type,profile_buffer);
```

9.1.30 rsi_wlan_delete_profile

Prototype

```
int32_t rsi_wlan_delete_profile(uint32_t type);
```

Description
This function is used to send raw data packet.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameter

Parameter	Description
type	profile type
output	returns success/failure as response

Return Value

Value	Description
0	Success
Non-Zero Value	Failure **If return value is less than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_wlan_delete_profile(type);
```

9.1.31 rsi_wlan_ power_save_profile

Prototype

```
int32_t rsi_wlan_power_save_profile(uint8_t psp_mode, uint8_t psp_type);
```

Description

This API sets the power save profile in wlan mode.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
psp_mode	Follwing psp_mode is defined. RSI_ACTIVE (0): In this mode module is active and power save is disabled. RSI_SLEEP_MODE_1 (1): This is the connected sleep mode. In this sleep mode, SoC will never turn off, therefore no handshake is required before sending data to the module. RSI_SLEEP_MODE_2 (2): This is the connected sleep mode. In this sleep mode, SoC will go to sleep based on GPIO or Message. Therefore handshake is required before sending data to the module. RSI_SLEEP_MODE_8 (8): This is disconnected sleep mode. In this sleep mode, module will turn off the SoC. Because SoC is turned off, a handshake is required before sending data to the module.
psp_type	Follwing psp_type is defined.

Parameter	Description
	RSI_MAX_PSP (0): This psp_type will be used for max power saving. RSI_FAST_PSP (1): This psp_type allows module to disable power save for any Tx / Rx packet for monitor interval of time (monitor interval can be set through configuration file, default value is 50 ms). If there is no data for monitor interval of time then module will again enable power save. RSI_UAPSD (2): This psp_type is used to enable WMM power save.

Note:

1. psp_type is only valid in psp_mode 1 and 2.
2. psp_type UAPSD is applicable only if WMM_PS is enabled in rsi_wlan_config.h file.
3. In RSI_MAX_PSP mode, Few Access points won't aggregate the packets, when power save is enabled from STA. This may cause the drop in throughputs.

Note:

For the power mode 3/9 select the RSI_HAND_SHAKE_TYPE as MSG_BASED.

Note:

Powersave handshake option:

- When sleep clock source is configured to '32KHz bypass clock on UULP_VBAT_GPIO_3',
  - use UULP_VBAT_GPIO_0 for SLEEP_IND_FROM_DEV
  - set RS9116_SILICON_CHIP_VER in 'RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\include\rsi_user.h' to 'CHIP_VER_1P4_AND_ABOVE'
- If not using external clock on UULP_VBAT_GPIO_3' as sleep clock source,
  - use UULP_VBAT_GPIO_3 for SLEEP_IND_FROM_DEV
  - set RS9116_SILICON_CHIP_VER in 'RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\include\rsi_user.h' to 'CHIP_VER_1P3'

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: **Command given in wrong state** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002C, 0xFFF8,0x0015,0x0026,0x0052

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
// set device in sleep mode 2, max PSP
// Note: device must be associated with AP before this call.
status = rsi_wlan_power_save_profile(PSP_MODE, PSP_TYPE);
if(status != RSI_SUCCESS)
{
return status;
}
```

9.1.32 rsi_wlan_power_save_with_listen_interval

Prototype

```
int32_t rsi_wlan_power_save_with_listen_interval(uint8_t psp_mode, uint8_t psp_type,uint16_t
listen_interval)
```

Description

This API sets the power save profile in wlan mode with listen interval-based wakeup.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
psp_mode	Following psp_mode is defined. RSI_ACTIVE (0): In this mode module is active and power save is disabled. RSI_SLEEP_MODE_1 (1): This is the connected sleep mode. In this sleep mode, SoC will never turn off, therefore no handshake is required before sending data to the module. RSI_SLEEP_MODE_2 (2): This is the connected sleep mode. In this sleep mode, SoC will go to sleep based on GPIO or Message. Therefore, handshake is required before sending data to the module. RSI_SLEEP_MODE_8 (8): This is disconnected sleep mode. In this sleep mode, module will turn off the SoC. Because SoC is turned off, a handshake is required before sending data to the module.
psp_type	Following psp_type is defined. RSI_MAX_PSP (0): This psp_type will be used for max power saving. RSI_FAST_PSP (1): This psp_type allows module to disable power save for any Tx / Rx packet for monitor interval of time (monitor interval can be set through configuration file, default value is 50 ms). If there is no data for monitor interval of time, then module will again enable power save. RSI_UAPSD (2):

Parameter	Description
	This psp_type is used to enable WMM power save.
listen_interval	This parameter is used to configure maximum sleep duration in power save. **Note**: This is valid only if BIT (7) in join_feature_bit_map is set. This value is given in time units (1024 microsecond). This parameter is used to configure. maximum sleep duration in power save. and should be less than the listen interval configured in join command.

Note:

1. psp_type is only valid in psp_mode 1 and 2.
2. psp_type UAPSD is applicable only if WMM_PS is enabled in rsi_wlan_config.h file.
3. In RSI_MAX_PSP mode, Few Access points won't aggregate the packets, when power save is enabled from STA. This may cause the drop in throughputs.

Note:

For the power mode 3/9 select the RSI_HAND_SHAKE_TYPE as MSG_BASED.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid parameters -3: **Command given in wrong state** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002C, 0xFFF8,0x0015,0x0026,0x0052

Example

```
status = rsi_wlan_power_save_with_listen_interval(PSP_mode, PSP_TYPE, LISTEN_INTERVAL);
if(status != RSI_SUCCESS)
{
  return status;
}
```

9.1.33 rsi_wlan_set_sleep_timer

Prototype

```
int32_t rsi_wlan_set_sleep_timer( uint16_t sleep_time )
```

Description

This API is used to configures the sleep timer mode of the module to go into sleep during power save operation.

Precondition

This command can be issued any time in case of power save mode 9 (MSG_BASED).

Parameters

Parameters	Description
sleep_time	Sleep Time value in seconds Minimum value is 1, and maximum value is 2100

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	**If return value is less than 0** **If return value is greater than 0** 0x0025, 0x002C

Example

```
rsi_wlan_set_sleep_timer(sleep_time);
```

9.1.34 rsi_wlan_filter_broadcast

Prototype

```
int32_t rsi_wlan_filter_broadcast(uint16_t beacon_drop_threshold,uint8_t filter_bcast_in_tim,uint8_t
filter_bcast_tim_till_next_cmd)
```

Description

This API is used to program the ignoring broad cast packet threshold levels when station is in powersave mode and is used to achieve low currents in standby associated mode.

Precondition

rsi_wlan_filter_broadcast() API needs to be called after opermode command only.

Parameters

Parameter	Description
beacon_drop_threshold	LMAC beacon drop threshold(ms): The amount of time that FW waits to receive full beacon. Default value is 5000ms.
filter_bcast_in_tim	If this bit is set, then from the next dtim any broadcast data pending bit in TIM indicated will be ignored valid values: 0 - 1 **Note**: Validity of this bit is dependent on the filter_bcast_tim_till_next_cmd.
filter_bcast_tim_till_next_cmd	0 -filter_bcast_in_tim is valid till disconnect of the STA 1 - filter_bcast_in_tim is valid till next update by giving the same command

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	**If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command

Value	Description
	If return value is greater than 0 0x0021

Note:

Please refer to Error Codes section for the description of the above error codes.

9.1.35 rsi_wlan_register_callbacks

Prototype

```
int32_t rsi_wlan_register_callbacks(uint32_t callback_id, void(*callback_handler_ptr)(uint16_t *status,
const uint8_t *buffer, const uint32_t length));
```

Description

This API registers the WLAN call back functions.

Precondition

None

Parameters

Parameter	Description
callback_id	This is the Id of the call back function Following ids are supported:
void(*callback_handler_ptr)(uint16_t *status,	This is the Call back handler **status:** status of the asynchronous response

Value	Callback id
1	RSI_JOIN_FAIL_CB
2	RSI_IP_FAIL_CB
3	RSI_REMOTE_SOCKET_TERMINATE_CB
4	RSI_IP_CHANGE_NOTIFY_CB
5	RSI_STATIONS_DISCONNECT_NOTIFY_CB
6	RSI_STATIONS_CONNECT_NOTIFY_CB
7	RSI_WLAN_DATA_RECEIVE_NOTIFY_CB
8	RSI_WLAN_RECEIVE_STATS_RESPONSE_CB
9	RSI_WLAN_SCAN_RESPONSE_HANDLER
10	RSI_WLAN_JOIN_RESPONSE_HANDLER
11	RSI_WLAN_PING_RESPONSE_HANDLER
12	RSI_WLAN_RAW_DATA_RECEIVE_HANDLER
13	RSI_AUTO_CONFIG_RESPONSE_HANDLER
14	RSI_WLAN_SOCKET_CONNECT_NOTIFY_CB
15	RSI_WLAN_SERVER_CERT_RECEIVE_NOTIFY_CB
16	RSI_WLAN_ASYNC_MODULE_STATE
17	RSI_FLASH_RESPONSE_HANDLER

Parameter	Description
const uint8_t *buffer, const uint16_t length)	**buffer:** payload of the asynchronous response **length:** length of the payload

Prototypes of the call back functions with given call back id

Call back id	Function Description
RSI_JOIN_FAIL_CB	This callback is called when asynchronous rejoin failure is received from the module.
RSI_IP_FAIL_CB	This callback is called when asynchronous DHCP renewal failure is received from the module.
RSI_REMOTE_SOCKET_TERMINATE_CB	This callback is called when asynchronous remote TCP socket closed is received from the module.
RSI_IP_CHANGE_NOTIFY_CB	This callback is called when asynchronous IP change notification is received from the module.
RSI_STATIONS_DISCONNECT_NOTIFY_CB	This callback is called when asynchronous station disconnect notification is received from the module in AP mode.
RSI_STATIONS_CONNECT_NOTIFY_CB	This callback is called when asynchronous station connect notification is received from the module in AP mode.
RSI_WLAN_DATA_RECEIVE_NOTIFY_CB	This callback is called when asynchronous data is received from the module in TCP/IP bypass mode.
RSI_WLAN_RECEIVE_STATS_RESPONSE_CB	This callback is called when asynchronous receive statistics from the module in per or end to end mode.
RSI_WLAN_SCAN_RESPONSE_HANDLER	This callback is called when a response for scan request is received from the module.
RSI_WLAN_JOIN_RESPONSE_HANDLER	This callback is called when a response for join request is received from the module
RSI_WLAN_PING_RESPONSE_HANDLER	This callback is called when a response come from the remote server and it has to be notified to the host.
RSI_WLAN_RAW_DATA_RECEIVE_HANDLER	This callback is called when raw data packets are received from the module.
RSI_AUTO_CONFIG_RESPONSE_HANDLER	This callback is called when response come from the module to notify to the host.
RSI_WLAN_SOCKET_CONNECT_NOTIFY_CB	This callback is called when a socket connection response comes to the host
RSI_WLAN_SERVER_CERT_RECEIVE_NOTIFY_CB	This callback is called when certificate response comes from the module
RSI_WLAN_ASYNC_MODULE_STATE	This callback is called when async response come from the module to the host.
RSI_FLASH_RESPONSE_HANDLER	This callback is called when flash response come from the module to notify to the host.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	3: Failure If call_back_id is greater than the maximum callbacks to register, returns 1.

Note:

In callbacks, application should not initiate any TX operation to the module.

RESPONSE STRUCTURES OF CALLBACK HANDLERS:

Response structure for RSI_STATIONS_CONNECT_NOTIFY_CB:

```
typedef struct rsi_connect_rsp_s{
  uint8 mac_address[6];
}rsi_connect_rsp_t;
```

Structure member	Description
mac_address	Holds the mac address of the station connected.

Response structure for RSI_STATIONS_DISCONNECT_NOTIFY_CB:

typedef struct rsi_disconnect_rsp_s{
uint8 mac_address[6];
}rsi_disconnect_rsp_t;

Structure member	Description
mac_address	Holds the mac address of the station disconnected.

Response structure for Socket terminate call back:

```
typedef struct rsi_socket_close_rsp_s {
uint8 socketDsc[2];
uint8 bytesSent[4];
}rsi_socket_close_rsp_t;
```

Structure member	Description
socketDsc	Socket descriptor of the socket closed.
bytesSent	Number of bytes sent successfully on that socket.

Response structure for IP Change

```
typedef struct rsi_recvIpchange_s {
uint8_t macAddr[6];
uint8_t ipaddr[4];
uint8_t netmask[4];
uint8_t gateway[4];
} rsi_recvIpChange_t;
```

Structure members	Description
macAddr	Holds MAC Address
Ipaddr	Holds Assigned IP address
Netmask	Holds Assigned subnet address
Gateway	Holds Assigned gateway address

Examples

```
//! Initialze station connect notify call back
rsi_wlan_register_callbacks(RSI_STATIONS_CONNECT_NOTIFY_CB, rsi_stations_connect_notify_handler);
```

Response structure for RSI_WLAN_SOCKET_CONNECT_NOTIFY_CB:

```
//! socket create command response structure
typedef struct rsi_rsp_socket_create_s
{
  //! ip version 4 or 6
  uint8_t   ip_version[2];
  //! 2 bytes, type of socket created
  uint8_t   socket_type[2];
  //! 2 bytes socket descriptor, like a file handle, usually 0x00
  uint8_t   socket_id[2];
  //! 2 bytes, Port number of our local socket
  uint8_t   module_port[2];
  union{
    //! 4 bytes, Our (module) IPv4 Address
    uint8_t   ipv4_addr[4];
    //! 4 bytes, Our (module) IPv6 Address
    uint8_t   ipv6_addr[16];
  }module_ip_addr;
  //! 2 bytes, Remote peer MSS size
  uint8_t   mss[2];
  //! 4 bytes, Remote peer Window size
  uint8_t   window_size[4];
} rsi_rsp_socket_create_t;
```

Examples

```
 //! Register socket connetion notify handler
    rsi_wlan_register_callbacks(RSI_WLAN_SOCKET_CONNECT_NOTIFY_CB, socket_connect_response_handler);
```

Response structure for RSI_WLAN_SERVER_CERT_RECEIVE_NOTIFY_CB:

```
typedef struct rsi_cert_recv_s
{
    //! Ip version
    uint8_t ip_version[2];
    //! Socket descriptor
    uint8_t sock_desc[2];
    //! local port
    uint8_t src_port[2];
    //! Destination port
    uint8_t dst_port[2];
    union
    {
        //! destination ipv4 address
        uint8_t ipv4_address[4];
        //! destination ipv6 address
        uint8_t ipv6_address[16];
    }ip_address;
    //! sequence number;
    uint8_t sequence_no[2];
    //! total length of the certificate
    uint8_t total_len[2];
    //!length of the current segment
    uint8_t current_len[2];
    //! more chunks flag
    uint8_t more_chunks;
    uint8_t reserved[5];
}rsi_cert_recv_t;
```

Examples

```
     //! Register certificate receive notify handler
    rsi_wlan_register_callbacks(RSI_WLAN_SERVER_CERT_RECEIVE_NOTIFY_CB, certificate_response_handler);
```

9.1.36 rsi_nwk_register_callbacks

Prototype

```
int32_t rsi_nwk_register_callbacks(rsi_nwk_callback_id_t callback_id, void
(*callback_handler_ptr)(uint8_t command_type , uint32_t status, const uint8_t *buffer, const uint32_t
length));
```

Parameters

Parameter	Description
callback_id	This is the Id of the call back function Following ids are supported: 0-RSI_NWK_ERROR_CB 1-RSI_WLAN_NWK_URL_REQ_CB 2-RSI_WLAN_NWK_JSON_UPDATE_CB 3-RSI_WLAN_NWK_FW_UPGRADE_CB 4-RSI_WLAN_NWK_JSON_EVENT_CB

Prototypes of the call back functions with given call back id

Call back id	Function prototype	Parameters	Function Description
RSI_NWK_ERROR_CB	void (*nwk_error_call_back_handler)(uint8_t command_type , uint32_t status, const uint8_t *buffer, const uint32_t length);	command_type: command_type of the response status: status of the response buffer: payload of the response length: length of the payload	This callback is used to Register join fail
RSI_WLAN_NWK_URL_REQ_CB	void (*rsi_webpage_request_handler)(uint8_t type, uint8_t *url_name, uint8_t *post_content_buffer, uint32_t post_content_length, uint32_t status);	type: type of the handler. url_name:url name post_content_buffer: Webpage content. post_content_lenght: length of webpage content. status: status of the response	This callback is used to Register webpage request
RSI_WLAN_NWK_JSON_UPDATE_CB	void (*rsi_json_object_update_handler)(uint8_t *file_name, uint8_t *json_object, uint32_t length, uint32_t status);	file_name: File name json_object: Json object length: length of the json object. status: status of the response.	This callback is used to Register json update
RSI_WLAN_NWK_FW_UPGRADE_CB	void (*rsi_wireless_fw_upgrade_handler)(uint8_t type, uint32_t status);	type: type of the handler. status: status of the response.	This callback is used to Register wireless firmware upgrade
RSI_WLAN_NWK_JSON_EVENT_CB	void (*rsi_json_object_event_handler)(uint32_t status, uint8_t *json_object_str, uint32_t length);	status: status of the response. json_object_str: json object string. length: length of the string.	This callback is used to Register json update

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	3: Failure If call_back_id is greater than the maximum callbacks to register, returns 1

Note:

In callbacks, application should not initiate any TX operation to the module.

9.1.37 rsi_wlan_buffer_config

Prototype

```
int32_t  rsi_wlan_buffer_config(void)
```

Description
This API is used to configure the tx ,rx global buffers ratio.

Precondition

rsi_wlan_buffer_config() API needs to be called after opermode command only.

Parameters

Parameter	Description
dynamic_tx_pool	To configure the dynamic tx ratio
dynamic_rx_pool	To configure the dynamic rx ratio
dynamic_global_pool	To configure the dynamic global ratio

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	**If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
int32_t rsi_wlan_buffer_config();
```

9.1.38 rsi_wlan_receive_stats_start

Prototype

```
int32_t rsi_wlan_receive_stats_start(uint16_t channel);
```

Description
This API gets the Transmit (TX) & Receive (RX) packets statistics. When this API is called by the host with valid channel number, the module gives the statistics to the host for every 1 second asynchronously.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
Channel	Valid channel number 2.4GHz or 5GHz **2.4GHz Band Channel Mapping** **5GHz Band Channel Mapping**

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002c, 0x000A

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_wlan_receive_stats_start(channel);
```

9.1.39 rsi_wlan_receive_stats_stop

Prototype

```
int32_t rsi_wlan_receive_stats_stop(void);
```

Description
This API stops the Transmit (TX) & Receive(RX) packets statistics.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure

Value	Description
	If return value is less than 0 -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002c

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_wlan_receive_stats_stop();
```

9.1.40 rsi_wlan_send_data

Prototype

```
int32_t rsi_wlan_send_data(uint8_t *buffer, uint32_t length);
```

Description
This API sends the raw data in TCP / IP bypass mode.

Precondition
None

Parameters

Parameter	Description
Buffer	This is the pointer to the buffer to send
Length	This is the length of the buffer to send

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -2: Invalid Parameters -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002c

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
uint8_t data[] = "data";
rsi_wlan_send_data(data, 5);
```

9.1.41 rsi_transmit_test_start

Prototype

```
int32_t rsi_transmit_test_start(uint16_t power,  uint32_t rate, uint16_t length,  uint16_t mode,
uint16_t channel);
```

Description
This API starts the transmit test.

Note:

This API is relevant in PER mode.

Precondition
rsi_wlan_radio_init() API needs to be called before this API.

Parameters

Parameter	Description
power	To set TX power in dbm. The valid values are from 2dbm to 18dbm for WiSeConnect module.
rate	To set transmit data rate.
length	To configure length of the TX packet. The valid values are in the range of 24 to 1500 bytes in the burst mode and range of 24 to 260 bytes in the continuous mode.
mode	0- Burst Mode 1- Continuous Mode 2- Continuous wave Mode (non modulation) in DC mode 3- Continuous wave Mode (non modulation) in single tone mode (center frequency -2.5MHz) 4- Continuous wave Mode (non modulation) in single tone mode (center frequency +5MHz)
channel	For setting the channel number in 2.4 GHz / 5GHz.

Note:

Before starting Continuous Wave mode, it is required to start Burst mode with power and channel values which is intended to be used in Continuous Wave mode i.e.

1. Start Burst mode with intended power value and channel values
2. Pass any valid values for rate and length
3. Stop Burst mode
4. Start Continuous wave mode

Data Rate (Mbps)	Value of rate
1	0
2	2
5.5	4
11	6
6	139
9	143
12	138
18	142

Data Rate (Mbps)	Value of rate
24	137
36	141
48	136
54	140
MCS0	256
MCS1	257
MCS2	258
MCS3	259
MCS4	260
MCS5	261
MCS6	262
MCS7	263

Channel Numbers (2.4GHz)	Center frequencies for 20MHz channel width (MHz)
1	2412
2	2417
3	2422
4	2427
5	2432
6	2437
7	2442
8	2447
9	2452
10	2457
11	2462
12	2467
13	2472
14	2484

The following tables map the channel number to the actual radio frequency in the 2.4 GHz spectrum.

Note: To start transmit test in 12,13,14 channels, configure set region parameters in rsi_wlan_config.h.

The following table maps the channel number to the actual radio frequency in the 5 GHz spectrum for 20MHz channel bandwidth. The channel numbers in 5 GHz range is from 36 to 165.

Channel Numbers (5GHz)	Center frequencies for 20MHz channel width (MHz)
36	5180
40	5200
44	5220
48	5240
52	5260
56	5280
60	5300
64	5320
149	5745
153	5765
157	5785
161	5805
165	5825

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x000A, 0x0021, 0x0025, 0x002C

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_transmit_test_start(power, rate,length, mode, channel);
```

9.1.42 rsi_transmit_test_stop

Prototype

```
int32_t rsi_transmit_test_stop(void);
```

Description

This API stops the transmit test.

Note:

This API is relevant in PER mode.

Precondition
rsi_wlan_radio_init() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is less than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002C

Note:
Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_transmit_test_stop();
```

9.1.43 rsi_req_wireless_fwup

Prototype

```
int32_t rsi_req_wireless_fwup(void);
```

Description
This API sends the wireless firmware upgrade request.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is less than 0** -2: Invalid Parameters -4: Buffer not available to serve the command

Note:
Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_req_wireless_fwup();
```

9.1.44 rsi_load_bootup_params

Prototype

```
static int rsi_load_bootup_params(struct rsi_common *common);
```

Description

This API is used to send Boot_up parameters to the Firmware.

Parameters

Parameter	Description
common	Pointer to the driver private structure

Return Values

Value	Description
Zero	Success
Corresponding error code	Failure

9.1.45 rsi_efuse_write

Prototype

```
int32_t rsi_efuse_write(rsi_efuse_write_t *efuse_write_p)
```

Description

This Function is used to write content to efuse.

Parameters

Parameter	Description
efuse_write_p	write efuse content

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_efuse_write_t *efuse_write_p;
status =  rsi_efuse_write(efuse_write_p);
```

9.1.46 rsi_efuse_read

Prototype

```
int32_t rsi_efuse_read(efuse_read_p, buf, length);
```

Description

This Function is used to get efuse content.

Parameters

Parameter	Description
efuse_read_p	Read efuse content

Parameter	Description
buffer	Pointer to buffer to store efuse content
length	length of the efuse content

Return Values

Value	Description
0	Success
Non-Zero Value	Failure

Example

```
status = rsi_efuse_read(efuse_read_p, buf, length);
```

9.1.47 rsi_flash_mem_wr

Prototype

```
int32_t rsi_flash_mem_wr(rsi_flash_write_t *write_req);
```

Description

This API is used to write content to flash memory.

Parameters

Parameter	Description
write_req	Content to be written to flash memory

Structure Prototype

```
typedef  struct rsi_flashwrite_s{
uint32_t flash_offset;
uint16_t length;
uint16_t flash_init_req:1;
uint16_t flash_erase_req:1;
uint16_t flash_write_req:1;
uint16_t reserved1:1;
uint16_t flash_type:1;
uint16_t flash_pinset:4;
uint16_t reserved:4;
uint32_t  src_buf_addr;
}rsi_flash_write_t;
```

Return Values

Value	Description
0	Success
Non-Zero Value	Failure

Example

```
rsi_flash_write_t *write_req;
status = rsi_flash_mem_wr(write_req);
```

9.1.48 rsi_reset_mac

Prototype

```
int32_t rsi_reset_mac();
```

Description

This API is used to reset mac.

Parameters

None

Return Values

Value	Description
0	Success
Non-Zero Value	Failure

Example

```
status = rsi_reset_mac();
```

9.1.49 translate_channel

Prototype

```
uint8_t translate_channel(uint8_t channelNum);
```

Description

This API is used to translate channel.

Parameters

Parameter	Description
channelNum	Channel number to translate

Return Values

Value	Description
channelNum	Success

Example

```
uint8_t channelNum;

status = translate_channel(channelNum);
```

9.1.50 rsi_radio_caps

Prototype

```
int32_t rsi_radio_caps(uint8_t operating_channel);
```

Description

Radio capabilities of the operating channel.

Parameters

Parameter	Description
channelNum	Operating channel number

Return Values

Value	Description
0	Success
Non-Zero Value	Failure

Example

```
status = rsi_radio_caps(1);
```

9.1.51 rsi_certificate_valid

Prototype

```
void  rsi_certificate_valid(uint16_t valid, uint16_t socket_id)
```

Description

After validating the server certificate received from module, host can indicate the status to module using this API.

Parameters

Parameter	Description
valid	This field indicates whether the server certificate is valid or not **1** - indicates that the server certificate is valid **0** - indicates that the server certificate is not valid
Socket_id	Socket identifier

Return Values

Value	Description
0	Success
<0	Failure

Example

```
status = rsi_certificate_valid(valid, socket_id)
```

9.2 Configuration parameters

This section explains the configuration of macros that may needed to be changed based on application requirement. These macros, with default values, are placed in "**rsi_wlan_config.h**".

9.2.1 Configure opermode parameters

Define	Meaning
RSI_FEATURE_BIT_MAP	To select WiSeConnect module feature bit map FEAT_SECURITY_OPEN : open mode (No security) FEAT_SECURITY_PSK : PSK security FEAT_AGGREGATION : WLAN Aggregation FEAT_LP_GPIO_BASED_HANDSHAKE : LP mode GPIO handshake FEAT_ULP_GPIO_BASED_HANDSHAKE : ULP mode GPIO based handshake FEAT_DEV_TO_HOST_ULP_GPIO_1 : To select ULP GPIO 1 for wake up indication

Define	Meaning
	FEAT_RF_SUPPY_VOL_3_3_VOLT : To supply 3.3 volt supply FEAT_WPS_DISABLE : Disable WPS in AP mode
RSI_TCP_IP_BYPASS	To select TCP IP Bypass mode in WiSeConnect module
RSI_TCP_IP_FEATURE_BIT_MAP	TCP_IP_FEAT_BYPASS – Select to use TCP/IP bypass TCP_IP_FEAT_HTTP_SERVER – Select to use HTTP SERVER TCP_IP_FEAT_DHCPV4_CLIENT – Select to use DHCPv4 client TCP_IP_FEAT_DHCPV6_CLIENT – Select to use DHCPv6 client TCP_IP_FEAT_DHCPV4_SERVER – Select to use DHCPv4 Server TCP_IP_FEAT_DHCPV6_SERVER – Select to use DHCPv6 server TCP_IP_FEAT_JSON_OBJECTS – Select to use JSON objects TCP_IP_FEAT_HTTP_CLIENT -Select to use HTTP CLIENT TCP_IP_FEAT_DNS_CLIENT – Select to use DNS CLIENT TCP_IP_FEAT_SNMP_AGENT – Select to use SNMP AGENT TCP_IP_FEAT_SSL - Select to enable SSL TCP_IP_FEAT_ICMP – Select to use PING from host TCP_IP_FEAT_HTTPS_SERVER -Select to HTTPS server TCP_IP_FEAT_FTP_CLIENT – Select to use FTP client feature TCP_IP_FEAT_IPV6 – Select to enable IPv6 feature TCP_IP_FEAT_MDNSD – Select to use MDNSD feature TCP_IP_FEAT_SMTP_CLIENT – Select to use SMTP client TCP_IP_FEAT_SINGLE_SSL_SOCKET – Select to use single SSL socket TCP_IP_FEAT_LOAD_PUBLIC_PRIVATE_CERTS – Select to load public and private keys for TLS or SSL hand shake User can configure number of sockets by using the below macros: TCP_IP_TOTAL_SOCKETS_1 TCP_IP_TOTAL_SOCKETS_2 TCP_IP_TOTAL_SOCKETS_3 TCP_IP_TOTAL_SOCKETS_4 TCP_IP_TOTAL_SOCKETS_5 TCP_IP_TOTAL_SOCKETS_6 TCP_IP_TOTAL_SOCKETS_7 TCP_IP_TOTAL_SOCKETS_8 TCP_IP_TOTAL_SOCKETS_9 TCP_IP_TOTAL_SOCKETS_10
RSI_CUSTOM_FEATURE_BIT_MAP	CUSTOM_FEAT_AP_IN_HIDDEN_MODE – Used to create AP in hidden mode CUSTOM_FEAT_DFS_CHANNEL_SUPPORT – Used to scan DFS channels in Wi-Fi client mode CUSTOM_FEAT_LED_FEATURE – LED blink feature after module initialization CUSTOM_FEAT_ASYNC_CONNECTION_STATUS – Enables asynchronous indication of WLAN connection state in Wi-Fi client mode CUSTOM_FEAT_WAKE_ON_WIRELESS – To enable wake on wireless CUSTOM_FEAT_BT_IAP - To enable IAP support in Bluetooth Classic
RSI_EXT_CUSTOM_FEATURE_BIT_MAP	EXT_FEAT_RSA_KEY_WITH_4096_SUPPORT – Used to support 4096 size RSA KEY certificate EXT_FEAT_TELEC_SUPPORT – Used to support TELEC EXT_FEAT_SSL_CERT_WITH_4096_KEY_SUPPORT – Used to support 4096 size KEY SSL certificate EXT_FEAT_AP_BROADCAST_PKT_SND_B4_DTIM – Used to enable wake on wireless for AP Broadcast customization EXT_FEAT_FCC_LOW_PWR – Used t to support FCC EXT_FEAT_PUF – Used to enable PUF EXT_FEAT_SPECTRAL_MASK_NOKIA – Used to Nokia spectral mask extended custom bitmap

Define	Meaning
	EXT_HTTP_SKIP_DEFAULT_LEADING_CHARACTER – Used to skip default leading character '\' in HTTP header EXT_FEAT_PUF_PRIVATE_KEY – Used to enable PUF private key

Note:

Enabling Aggregation bit (feature_bit_map[2]) and Low power mode bit (ext_custom_feature_bit_map[19]) in Opermode will result in Wi-Fi Data not working. So, they cannot be enabled at the same time.

Note:

1. Set BIT(31) in tcp_ip_feature_bit_map & BIT(29) in ext_tcp_ip_feature_bit_map to open 3 SSL sockets.
2. It supports in WLAN mode only.

9.2.2 Configure scan parameters

Define	Meaning
RSI_SCAN_CHANNEL_BIT_MAP_2_4	To select channels in 2.4GHz band to do selective channel scan. This macro is valid only if channel 0 is selected in rsi_wlan_scan API.
RSI_SCAN_CHANNEL_BIT_MAP_5	To select channels in 5GHz band to do selective channel scan. This macro is valid only if channel 0 is selected in rsi_wlan_scan API.
RSI_SCAN_FEAT_BITMAP	RSI_ENABLE_QUICK_SCAN - If enabled, module scans for the AP given in scan API and posts the scan results immediately to the host after finding the access point. This bit is valid only if specific channel and ssid to scan is given. RSI_SCAN_RESULTS_TO_HOST - if enabled additional scan results are given to host, but after getting scan results host has to issue another scan request with this bit disabled before join

9.2.3 Configure AP Mode parameters

Define	Meaning
RSI_AP_KEEP_ALIVE_ENABLE	To Enable keep alive functionality in AP mode
RSI_AP_KEEP_ALIVE_TYPE	RSI_NULL_BASED_KEEP_ALIVE – To perform keep alive by sending Null data packet to stations RSI_DEAUTH_BASED_KEEP_ALIVE – To perform keep alive based on packets received from stations within time out
RSI_AP_KEEP_ALIVE_PERIOD	To configure keep alive period
RSI_MAX_STATIONS_SUPPORT	To configure maximum stations supported

9.2.4 Configure Set Region parameters

Define	Meaning
RSI_SET_REGION_SUPPORT	Enable to send set region command during Wi-Fi client connection
RSI_SET_REGION_FROM_USER_OR_BEACON	1 - region configurations taken from user 0 - region configurations taken from beacon
RSI_REGION_CODE	0 - Default Region domain 1 - US 2 - EUROPE 3 - JAPAN

9.2.5 Configure Set Region AP parameters

Define	Meaning
RSI_SET_REGION_AP_SUPPORT	Enable to send set region AP command during AP start
RSI_SET_REGION_AP_FROM_USER	1 - region configurations taken from user 0 - region configurations taken from firmware
RSI_COUNTRY_CODE	Country code which is supposed to be in Upper case. If the first parameter is 1, the second parameter should be one of the these 'US','EU','JP' country codes **Note:** If the country code is of 2 characters,3rd character should be <space>

9.2.6 Configure Rejoin parameters

Define	Meaning
RSI_REJOIN_PARAMS_SUPPORT	Enable to send rejoin parameters command during Wi-Fi client connection
RSI_REJOIN_MAX_RETRY	Number of retries **Note**: If Max retries is 0, retries infinity times.
RSI_REJOIN_SCAN_INTERVAL	Periodicity of rejoin attempt
RSI_REJOIN_BEACON_MISSED_COUNT	Beacon missed count
RSI_REJOIN_FIRST_TIME_RETRY	ENABLE or DISABLE retry for the first time join failure

9.2.7 Configure BG scan parameters

Define	Meaning
RSI_BG_SCAN_SUPPORT	Enable to send BG scan command after Wi-Fi client connection
RSI_BG_SCAN_ENABLE	To enable or disable BG Scan.

Define	Meaning
RSI_INSTANT_BG	Instant BG scan or normal BG scan
RSI_BG_SCAN_THRESHOLD	This is the threshold in dBm to trigger the BG scan
RSI_RSSI_TOLERANCE_THRESHOLD	This is the difference of last RSSI of connected AP and current RSSI of connected AP. Here, last RSSI is the RSSI calculated at the last beacon received and current RSSI is the RSSI calculated at current beacon received. If this difference is more than RSI_RSSI_TOLERANCE_THRESHOLD then BG scan will be triggered irrespective of periodicity.
RSI_BG_SCAN_PERIODICITY	This is the time period in seconds to trigger BG scan
RSI_ACTIVE_SCAN_DURATION	This is the active scan duration per channel in milli seconds
RSI_PASSIVE_SCAN_DURATION	This is the passive scan duration per DFS channel in 5GHz in milli seconds
RSI_MULTIPROBE	If it is set to one, then module will send two probe request - one with specific SSID provided during join command and other with NULL SSID (to scan all the access points)

9.2.8 Configure Roaming parameters

Define	Meaning
RSI_ROAMING_SUPPORT	Enable to send roaming command after Wi-Fi client connection
RSI_ROAMING_THRESHOLD	If connected AP, RSSI falls below this then module will search for new AP from background scanned list
RSI_ROAMING_HYSTERISIS	If module found new AP with same configuration (SSID, Security etc) and if (connected_AP_RSSI – Selected_AP_RSSI) is greater than RSI_ROAMING_HYSTERISIS then it will try to roam to the new selected AP

Note:

WLAN - RS9116 supports Wi-Fi L2 roaming.

9.2.9 Configure HT capabilities

Define	Meaning
RSI_MODE_11N_ENABLE	Enable to send Ht capabilities command during AP start
RSI_HT_CAPS_BIT_MAP	Bit map corresponding to high throughput capabilities. ht_caps_bit_map[10:15]: All set to '0' ht_caps_bit_map[8:9]:Rx STBC support 00- Rx STBC support disabled 01- Rx STBC support enabled ht_caps_bit_map[6:7]: Set to '0' ht_caps_bit_map[5]: short GI for 20Mhz support 0- short GI for 20Mhz support disabled 1- short GI for 20Mhz support enabled ht_caps_bit_map[4]: Green field support 0 -Green field support disabled 1 -Green field support enabled ht_caps_bit_map[0:3]:Set to '0'

9.2.10 Configure Enterprise mode parameters

Define	Meaning
RSI_EAP_METHOD	Should be one of among TLS, TTLS, FAST or PEAP. It should be ASCII Character string.
RSI_EAP_INNER_METHOD	This field is valid only in TTLS/PEAP. In case of TTLS/PEAP, the supported inner methods are MSCHAP/MSCHAPV2. In case of TLS/FAST, it should be fixed to MSCHAPV2.

9.2.11 Configure Join parameters

Define	Meaning
RSI_POWER_LEVEL	This fixes the Transmit Power level of the module. This value can be set as follows: At 2.4GHz 0– Low power (7+/-1) dBm 1– Medium power (10 +/-1) dBm 2– High power (18 + /- 2) dBm At 5 GHz 0– Low power (5+/-1) dBm 1– Medium power (7 +/-1) dBm 2– High power (12 +/- 2) dBm
RSI_JOIN_FEAT_BIT_MAP	BIT[0]: To enable b/g only mode in station mode, host has to set this bit. 0 – b/g/n mode enabled in station mode 1 – b/g only mode enabled in station mode BIT[1]: To take listen interval from join command. 0 – Listen interval invalid 1 – Listen interval valid BIT[2]:To enable/disable quick join feature. 1 - To enable quick join feature. 0 - To disable quick join feature. BIT[3]-BIT[7]: Reserved.
RSI_LISTEN_INTERVAL	This is valid only if BIT (1) in join_feature_bit_map is set. This value is given in Time units (1024 microsecond). This parameter is used to configure maximum sleep duration in power save.
RSI_DATA_RATE	To select Auto or Fixed data rate. Recommended to use Auto rate. RSI_DATA_RATE_AUTO: Auto rate

9.2.12 Configure SSL parameters

Define	Meaning
RSI_SSL_VERSION	To select ssl version. By default, it supports 1.2 version RSI_SSL_V_1: To support TLS 1.0 version RSI_SSL_V_2: To support TLS 1.2 version
RSI_SSL_CIPHERS	To select SSL ciphers. Below Macros are used to select type of cipher. SSL_ALL_CIPHERS TLS_RSA_WITH_AES_256_CBC_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_CBC_SHA TLS_RSA_WITH_AES_128_CCM_8 TLS_RSA_WITH_AES_256_CCM_8

Define	Meaning
	For example: To select two ciphers, define RSI_SSL_CIPHERS with (TLS_RSA_WITH_AES_256_CCM_8 TLS_RSA_WITH_AES_128_CCM_8)

9.2.13 Curve IDs Supported

Curve Id	Description	Support
15	secp160k1	Yes
16	secp160r1	Yes
17	secp160r2	Yes
18	secp192k1	Yes
19	secp192r1	Yes
20	secp224k1	Yes
21	secp224r1	Yes
22	secp256k1	Yes
23	secp256r1	Yes
24	secp384r1	Yes
25	secp521r1	Yes
26	brainpoolP256r1	Yes
27	brainpoolP384r1	Yes
28	brainpoolP512r1	Yes

9.2.14 Configure Power Save parameters

Define	Meaning
RSI_HAND_SHAKE_TYPE	To set handshake type of power mode MSG_BASED/GPIO_BASED
RSI_SELECT_LP_OR_ULP_MODE	0 - LP, 1- ULP mode with RAM retention and 2 - ULP without RAM retention
RSI_DTIM_ALIGNED_TYPE	set DTIM alignment required 0 - module wakes up at beacon which is just before or equal to listen_interval. 1 - module wakes up at DTIM beacon which is just before or equal to listen_interval.
RSI_MONITOR_INTERVAL	Monitor interval for the FAST PSP mode. Default is 50 ms, and this parameter is valid for FAST PSP only
RSI_WMM_PS_ENABLE	To set wmm enable or disable
RSI_WMM_PS_TYPE	To set wmm type

Define	Meaning
	0 - TX BASED 1 - PERIODIC
RSI_WMM_PS_WAKE_INTERVAL	To set wmm wake up interval
RSI_WMM_PS_UAPSD_BITMAP	To set wmm UAPSD bitmap
RSI_NUM_OF_DTIM_SKIP	If this macro is n then module will wake up at (n+1)th DTIM.

9.2.15 Configure 10 TCP Sockets when Device is in AP mode

Define	Meaning
HIGH_PERFORMANCE_ENABLE	To make RSI_ENABLE or RSI_DISABLE high performance sockets
TOTAL_SOCKETS	To Set total number of Sockets - TCP_TX + TCP_RX + UDP_TX + UDP_RX 10 - To Open 10 Sockets in total
TOTAL_TCP_SOCKETS	To Set total TCP Sockets - TCP_TX + TCP_RX 10 - To open 10 TCP Sockets
TOTAL_UDP_SOCKETS	To Set total UDP Sockets 0 - Disable 1 - Enable
TCP_TX_ONLY_SOCKETS	To Set total TCP TX only Sockets - TCP TX
TCP_RX_ONLY_SOCKETS	To Set TCP RX only Sockets - TCP RX 10 - To Open 10 TCP_RX sockets
UDP_TX_ONLY_SOCKETS	To Set UDP TX only Sockets - UDP TX 0 - Disable 1 - Enable
UDP_RX_ONLY_SOCKETS	To Set UDP RX only Sockets - UDP RX 0 - Disable 1 - Enable
TCP_RX_HIGH_PERFORMANCE_SOCKETS	To Set TCP_RX high performance Sockets 10 - To open 10 TCP_RX_High performance Sockets
TCP_RX_WINDOW_SIZE_CAP	To Set TCP RX window size 10 - To make the TCP RX window size to 10
TCP_RX_WINDOW_DIV_FACTOR	To Set TCP_RX window division factor 10 - To make TCP RX window division factor to 10
RSI_NUMBER_OF_LTCP_SOCKETS	To Set the number of LTCP sockets 10 - To open 10 LTCP sockets
RSI_NUMBER_OF_SOCKETS	Default number of Sockets supported. Set this to, 10 + RSI_NUMBER_OF_LTCP_SOCKETS

10 NWK APIs

This section explains the Network APIs MQTT, FTP, HTTP, SMTP, SNTP, DHCP, POP3 and Websockets.

10.1 DHCP User class (Option-77) API

10.1.1 rsi_dhcp_user_class

Prototype

```
int32_t  rsi_dhcp_user_class(uint8_t mode, uint8_t count,
user_class_data_t *user_class_data,
void(*dhcp_usr_cls_rsp_handler)(uint16_t status));
```

Description

This API is used to enable DHCP user class.

Parameters

Parameter	Description
mode	This is the DHCP User Class mode 1- RFC Compatible mode 2- Windows Compatible mode
count	This is the DHCP User Class count
user_class_data	Length – User class data length Data – User class data count
Status	This is the status of the DHCP user class 0 = success <0 = failure

It is used to get status for socket API, as in new driver is storing errors for each socket id, it will take sock id as input and returns error codes.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x002C, 0x0015, 0xFF74, 0x003E

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_dhcp_user_class((uint8_t) mode, (uint8_t) count, (user_class_data_t *)&user_class_data[0],
rsi_dhcp_usr_cls_response_handler);
```

10.2 EMB_MQTT_client API

10.2.1 rsi_emb_mqtt_client_init

Prototype

```
int32_t rsi_emb_mqtt_client_init(int8_t *server_ip, uint32_t server_port, uint32_t client_port,
uint16_t flags, uint16_t keep_alive_interval,int8_t *clientid, int8_t *username, int8_t *password)
```

Description

This API is used to create MQTT objects. TCP level connection will happen in this API.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
server_ip	This is the MQTT broker IP address to connect
server_port	This is the port number of MQTT broker
client_port	This is the port number of MQTT client (local port)
flags	These are the network flags. Each bit in the flag has its own significance BIT(0) – Clean Session for clearing the historic data if set 1 - Enable Clean Session BIT(1) - SSL Enable Bit 0 - SSL Disable 1 - SSL Enable BIT(2) - IP version of the Server IP 0 - IPV4 1 - IPV6
keep_alive_interval	This is the MQTT client keep alive interval If there are no transactions between MQTT client and broker within this time period, then MQTT Broker shall disconnects the MQTT client.
clientid	This is the client_id of MQTT_client which should be unique for different clients.
username	This is the user_name of the MQTT_client which is a credential for logging to MQTT_server as an authentication
password	This is the Password of the MQTT_client which is also credential for MQTT_server as an authentication

Note: Maximum supported length for username is 60 bytes.

Maximum supported length for password is 60 bytes.

Return Values

Value	Description
0	Successful execution
Non-Zero Value	Failure -5: command not supported -3: command given in wrong state -2: invalid parameters

Value	Description
	-4: packet allocation failure -44: parameter length exceeds maximum value -32: network command in progress

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
//! MQTT client initialisation
status = rsi_emb_mqtt_client_init((int8_t *)&server_address, SERVER_PORT , CLIENT_PORT ,
RSI_EMB_MQTT_CLEAN_SESSION , RSI_KEEP_ALIVE_PERIOD , (int8_t *)clientID , (int8_t *)username , (int8_t
*)password);
```

10.2.2 rsi_emb_mqtt_connect

Prototype

```
int32_t rsi_emb_mqtt_connect (uint8_t mqtt_flags, int8_t *will_topic, uint16_t will_message_len, int8_t
*will_message)
```

Description

This API is used to Connect to MQTT Server/Broker. MQTT level connection will happen in this API.

Precondition

rsi_emb_mqtt_client_init()API needs to be called before this API.

Parameters

Parameter	Description
flags	These are the network flags. Each bit in the flag has its own significance BIT(7) - usrFlag 0 - Disable 1 - Enable BIT(6) - pwdFlag 0 - Disable 1 - Enable
will topic	This is the will topic that the MQTT client wants the MQTT Server to publish when disconnected unexpectedly.
will_message_len	Length of will_message
will_message	This is the will message issued by the MQTT_Server for a broken client.

Note: will topic and will_message are not supported and should be NULL.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure -5: command not supported

Value	Description
	-3: command given in wrong state -4: packet allocation failure -32: network command in progress

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_emb_mqtt_connect((RSI_EMB_MQTT_USER_FLAG | RSI_EMB_MQTT_PWD_FLAG), NULL, 0, NULL);
```

10.2.3 rsi_emb_mqtt_subscribe

Prototype

```
int32_t rsi_emb_mqtt_subscribe(uint8_t qos, int8_t *topic)
```

Description

This API subscribes to the topic specified. Thus, MQTT client will receive any data which is published on this topic.

Precondition

rsi_emb_mqtt_connect()API needs to be called before this API.

Parameters

Parameter	Description
qos	This is the quality of service of message at MQTT protocol level. The valid values are 0,1,2.
topic	This the topic string on which MQTT client wants to subscribe.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure -5: command not supported -3: command given in wrong state -2: invalid parameters -4: packet allocation failure -44: parameter length exceeds maximum value -32: network command in progress

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Subscribe to the topic given
rsi_emb_mqtt_subscribe(QOS,RSI_MQTT_TOPIC);
```

10.2.4 rsi_emb_mqtt_publish

Prototype

```
int32_t rsi_emb_mqtt_publish(int8_t *topic, rsi_mqtt_pubmsg_t  *publish_msg)
```

Description
This API publishes the given message on the topic specified.

Precondition

rsi_emb_mqtt_connect()API needs to be called before this API.

Parameters

Parameter	Description
topic	This is the topic string on which MQTT client wants to publish data
publish_msg	This is the publish message

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure -5: command not supported -3: command given in wrong state -2: invalid parameters -4: packet allocation failure -44: parameter length exceeds maximum value -32: network command in progress

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_emb_mqtt_publish(RSI_MQTT_TOPIC,&publish_msg);
```

10.2.5 rsi_emb_mqtt_unsubscribe

Prototype

```
int32_t rsi_emb_mqtt_unsubscribe(int8_t *topic)
```

Description
This API unsubscribes to the topic specified. Thus, MQTT client will not receive any data published on this topic.

Precondition

rsi_emb_mqtt_connect()API needs to be called before this API.

Parameters

Parameter	Description
topic	This is the topic string to which MQTT client wants to unsubscribe.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure -5: command not supported -3: command given in wrong state -2: invalid parameters -4: packet allocation failure -44: parameter length exceeds maximum value -32: network command in progress

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! UnSubscribe to the topic given
rsi_emb_mqtt_unsubscribe(RSI_MQTT_TOPIC);
```

10.2.6 rsi_emb_mqtt_disconnect

Prototype

```
int32_t rsi_emb_mqtt_disconnect()
```

Description

This API disconnects the client from MQTT Server/Broker. TCP and MQTT level disconnection take place here.

Precondition

rsi_emb_mqtt_connect()API needs to be called before this API.

Parameters

Parameter	Description
input	void

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure -5: command not supported -3: command given in wrong state -4: packet allocation failure -32: network command in progress

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Disconnect to the MQTT broker
rsi_emb_mqtt_disconnect();
```

10.2.7 rsi_emb_mqtt_destroy

Prototype

```
int32_t rsi_emb_mqtt_destroy()
```

Description
This API is used to delete MQTT client's profile configuration and TCP level disconnection happens here, if required.

Precondition

rsi_emb_mqtt_client_init()API needs to be called before this API based on requirement.

This API can also be issued after rsi_emb_mqtt_disconnect() API.

Parameters

Parameter	Description
input	void

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure -5: command not supported -3: command given in wrong state -4: packet allocation failure -32: network command in progress

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Disconnect to the MQTT broker
rsi_emb_mqtt_destroy();
```

10.3 Multicast

10.3.1 rsi_multicast

Prototype

```
static int32_t rsi_multicast(uint8_t flags, int8_t *ip_address, uint8_t command_type)
```

Description
This is a helper function for actual APIs.

Parameters

Parameter	Description
flags	to select version and security, BIT(0) : 0 - IPv4 , 1 - IPv6
ip_address	multicast IP address
command_type	type of the command JOIN/LEAVE

Return Values

Value	Description
0	success
<0	failure

Example

```
rsi_multicast(flags, ip_address, RSI_MULTICAST_JOIN);
```

10.3.2 rsi_send_raw_data

Prototype

```
int32_t rsi_send_raw_data(uint8_t* buffer, uint32_t length)
```

Description
This API is used to send RAW data to Module.

Parameters

Parameter	Description
buffer	This is the pointer to the buffer to send
length	This is the length of the buffer to send

Return Values

Value	Description
0	success
<0=	failure

Example

```
rsi_send_raw_data(buffer,length);
```

10.3.3 rsi_multicast_join

Prototype

```
int32_t rsi_multicast_join(uint8_t flags, int8_t *ip_address);
```

Description
This API joins to a multicast group.

> **Note**:
>
> RS916W module supports only one Multicast group. It should leave the previous group, if it wants to join a new Multicast group.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
flags	To select the IP version. BIT(0) – RSI_IPV6 Set this bit to enable IPv6 , by default it is configured to IPv4.
ip_address	IPv4/IPv6 address of multicast group.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015,0xBB16,0xBB17

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Join to multicast group
status = rsi_multicast_join(FLAGS, (int8_t *)&multicast_ip);
```

10.3.4 rsi_multicast_leave

Prototype

```
int32_t rsi_multicast_leave(uint8_t flags, int8_t *ip_address);
```

Description
This API is used to leave the multicast group.

Note:

RS916W module supports only one Multicast group. It should leave the previous group, if it wants to join a new Multicast group.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
flags	To select the IP version. BIT(0) – RSI_IPV6 Set this bit to enable IPv6 , by default it is configured to IPv4.
ip_address	IPv4/IPv6 address of multicast group.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015,0xBB16,0xBB17

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Leave multicast group
status = rsi_multicast_leave(FLAGS, (int8_t *)&multicast_ip);
```

10.4 MDNS

10.4.1 rsi_mdns_txt_rec_create

Prototype

```
void rsi_mdns_txt_rec_create( rsi_mdns_txt_rec_t *txtRecord, uint16_t bufferLen, void *buffer);
```

Description

This API is used to create mdns text record.

Parameters

Parameters	Description
txtRecord	Pointer to text record
bufferLen	Length of buffer which is given by buffer
buffer	Pointer to buffer, which is to be used to for text record

Return Values

None

Example

```
rsi_mdns_txt_rec_t txtRecord;
uint16_t bufferlen;
char *buff;
rsi_mdns_txt_rec_create(&txtRecord,bufferlen,buff);
```

10.4.2 rsi_mdns_txt_rec_setvalue

Prototype

```
int8_t rsi_mdns_txt_rec_setvalue( rsi_mdns_txt_rec_t *txtRecord, const char *key, uint8_t valueSize,
const void *value);
```

Description

Function to set value in mdns text record.

Parameters

Parameters	Description
txtRecord	Pointer to text record
key	Pointer to Key
valueSize	Size of value
value	Pointer to value

Return Values

Value	Description
0	True
Non-Zero	Error

Example

```
rsi_mdns_txt_rec_setvalue(&txtRecord,&key,valuesize,&value);
```

10.4.3 rsi_mdns_txt_rec_search

Prototype

```
uint8_t *rsi_mdns_txt_rec_search(uint16_t txtLen, const void *txtRecord, const char *key, uint32_t
*keylen);
```

Description
This function is used to search in current mdns text record.

Parameters

Parameter	Description
txtLen	Text length
txtRecord	Pointer to Text Record
key	pointer to Key
keylen	pointer to Key Length

Return Values

Value	Description
0	success
<0	failure

Example

```
rsi_mdns_txt_rec_search(txtLen, &txtRecord, &key ,&keylen);
```

10.4.4 rsi_mdns_txt_rec_removevalue

Prototype

```
int8_t  rsi_mdns_txt_rec_removevalue(rsi_mdns_txt_rec_t *txtRecord, const char *key);
```

Description
This function is used to remove a value from MDNS text record.

Parameters

Parameter	Description
txtRecord	Pointer to Text Record
Key	Pointer to Key

Return Values

Value	Description
0	success
<0	failure

Example

```
rsi_mdns_txt_rec_removevalue( &txtRecord , &key );
```

10.4.5 rsi_mdns_get_txt_rec_buffer

Prototype

```
const void * rsi_mdns_get_txt_rec_buffer(rsi_mdns_txt_rec_t *txtRecord);
```

Description
This function is used to return MDNS text record buffer pointer.

Parameters

Parameter	Description
txtRecord	Pointer to Text Record

Return Values

Pointer to text record Buffer

Example

```
rsi_mdns_txt_rec_removevalue( &txtRecord  );
```

10.5 BSD Socket API

This section explains the network stack APIs required to configure the embedded TCP / IP stack and data transfer over the network.

10.5.1 rsi_config_ipaddress

Prototype

```
int32_t rsi_config_ipaddress(
rsi_ip_version_t version,
rsi_ip_config_mode_t mode,
uint8_t *ip_addr,
uint8_t *mask,
uint8_t *gw,
uint8_t *ipconfig_rsp,
uint16_t length,
uint8_t vap_id);
```

Description
This API configures the IP address to the module.

Precondition
rsi_wlan_connect() API needs to be called before this API.

Parameters

Parameter	Description
version	This is the IP version RSI_IP_VERSION_4 (4) – to select IPv4 RSI_IP_VERSION_6 (6) – to select IPv6
mode	This is the IP configuration mode RSI_STATIC (0)- to give static address RSI_DHCP (1)- to use DHCP RSI_DHCP_HOSTNAME (3) - to enable DHCP and to send DHCP host name in DHCP discover RSI_DHCP_OPTION_81 (4) - to enable FQDN option with static ip(Option 81 with static ip) RSI_DHCP_OPTION_77 (5) - to enable 71 with static ip RSI_DHCP_OPTION_81 (7) - to enable DHCP, hostname, DHCP client FQDN option (Option 81 with Dynamic RSI_DHCP_UNICAST (9) - to support DHCP unicast Offer from server
ip_addr	This is the pointer to IP address
mask	This is the pointer to network mask
gw	This is the pointer to gateway address
ipconfig_rsp	To hold the IP configuration received using DHCP. On successful DHCP, this buffer holds <Module MAC address> <Module IP address> <network mask> <gateway> in sequence
length	This is the length of ipconfig_rsp buffer
vap_id	This is the vap id to differentiate between AP and station in concurrent mode 0 – for station 1 – for Access point

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid Parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002C, 0xFFFC, 0xFF74, 0xFF9C, 0xFF9D

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
 //! Configure IP

 status = rsi_config_ipaddress(RSI_IP_VERSION_4, RSI_STATIC, (uint8_t
*)&ip_addr, (uint8_t *)&network_mask, (uint8_t *)&gateway,
NULL, 0,0);
```

10.5.2 rsi_socket

Prototype

```
int32_t rsi_socket(int32_t protocolFamily,
int32_t type,
uint32_t protocol);
```

Description
This API creates the socket.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
protocolFamily	Protocol family to select IPv4 or IPv6 AF_INET (2): to select IPv4 AF_INET6 (3): to select IPv6
type	Select socket type UDP or TCP SOCK_STREAM (1): to select TCP SOCK_DGRM (2): to select UDP
protocol	0: non SSL sockets 1: SSL sockets BIT(5) should be enabled for selecting the certificate index 0<<12 for index 0 1<<12 for index 1 Note: For selecting the certificate index feature, certificates should be loaded into ram.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
// create TCP Socket
client_socket = rsi_socket(AF_INET, SOCK_STREAM, 0);
```

Note:

To configure No of sockets Need to configure the bits[21:24] in tcp_ip_feature_bit_map in rsi_wlan_config.h.

To configure the No.of sockets , user need to configure the define RSI_NUMBER_OF_SOCKETS and define RSI_NUMBER_OF_LTCP_SOCKETS parameters in rsi_user.h.

By default, two sockets are enabled.

Note:

For higher throughput, bi-directional data traffic, multiple sockets applications use 256k Memory configuration.

10.5.3 rsi_bind

Prototype

```
int32_t rsi_bind(uint32_t sockID,
 struct sockaddr *localAddress,
 int32_t addressLength);
```

Description
This API assigns an address to a socket.

> **Note**:
>
> Bind command is mandatory to call after socket create command.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This the socket descriptor ID
localAddress	This is the address assigned to the socket. The format is compatible with BSD socket
addressLength	This is the length of the address measured in bytes

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
 struct sockaddr_in client_addr;
 //! Memset client structrue
 memset(&client_addr, 0, sizeof(client_addr));
 //! Set family type
 client_addr.sin_family= AF_INET;
 //! Set local port number
 client_addr.sin_port = htons(DEVICE_PORT);
 //! Bind socket
 status = rsi_bind(client_socket, (struct sockaddr *) &client_addr, sizeof(client_addr)
```

10.5.4 rsi_connect

Prototype

```
int32_t rsi_connect(uint32_t sockID,
 struct sockaddr *remoteAddress,
 int32_t addressLength);
```

Description
This API connects the socket to the specified remote address.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This the socket descriptor ID
remoteAddress	This is the remote peer address. The format is compatible with BSD socket
addressLength	This is the length of the address in bytes

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
status = rsi_connect(client_socket, (struct sockaddr *) &server_addr, sizeof(server_addr));
```

Note: For asynchronous behaviour, user needs to register RSI_STATIONS_CONNECT_NOTIFY_CB callback id.

Example:

rsi_wlan_register_callbacks(RSI_STATIONS_CONNECT_NOTIFY_CB, stations_connect_notify_handler);

10.5.5 rsi_listen

Prototype

```
int32_t rsi_listen(uint32_t sockID,
 int32_t backlog);
```

Description
This API makes the socket to listen for a remote connection request in passive mode.

Precondition
rsi_socket() / rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This is the socket descriptor ID
Backlog	The maximum length to which the queue of pending connections can be held

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
status = rsi_listen(server_socket, 1);
```

10.5.6 rsi_accept

Prototype

```
int32_t rsi_accept(uint32_t sockID,
 struct sockaddr *ClientAddress,
 int32_t *addressLength);
```

Description
This API accepts the connection request from the remote peer. This API extracts the connection request from the queue of pending connections on listening socket and accepts it.

Precondition
rsi_socket()/rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This is the socket descriptor ID
ClientAddress	This is the remote peer address. This parameter is an out parameter filled by driver on successful connection acceptance. The format is compatible with BSD socket.
addressLength	This is the length of the address measured in bytes

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
new_socket = rsi_accept(server_socket, (struct sockaddr *)&client_addr, &addr_size);
```

10.5.7 rsi_accept_async

Prototype

```
int32_t  rsi_accept_async(uint32_t sockID, void (*callback)(int32_t sock_id, int16_t dest_port,int8_t
*ip_addr, int16_t ip_version));
```

Description
This API accepts the connection request from the remote peer and register a callback which will be used by the driver to forward the received connection request asynchronously to the application.

Precondition
rsi_socket()/rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This is the socket descriptor ID
Callback	callback function to indicate the socket parameters connected sock_id: This is the socket descriptor ID dest_port: Port number of remote peer ip_addr: This is the remote peer address ip_version: 4 if it is IPV4 connection 6 if it is IPV6 connection

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
new_socket = rsi_accept_async(sockID, accept_async_handler)
```

10.5.8 rsi_select

Prototype

```
int32_t  rsi_select(int32_t nfds,
                    rsi_fd_set *readfds,
                    rsi_fd_set *writefds,
                    rsi_fd_set *exceptfds,
```

```
                    struct rsi_timeval *timeout,
                     void(*callback)(rsi_fd_set *fd_read,rsi_fd_set *fd_write,rsi_fd_set
*fd_except,int32_t status));
```

Description
This API retrieves the data pending from the remote peer on a given socket descriptor.

Precondition
rsi_socket()/rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
Nfds	Maximum number of socket descriptors
Readfds	This is the pointer to hold the bitmap for the select on read operation
Writefds	This is the pointer to hold the bitmap for the select on write operation
Exceptfds	This is the pointer to hold the bitmap for exceptional cases for select request
Timeout	This is the timeout value to break the wait for select.
Callback	This is the callback when asynchronous response comes for the select request. fd_read: This is the pointer which holds the bitmap for the select on read operation fd_write: This is the pointer which holds the bitmap for the select on write operation fd_except: This is the pointer which holds the bitmap for the select on exceptional operation status: This is the status code

Return Values

Value	Description
>0	Indicates the total number of bits that are ready across all the descriptor sets.
0	Time out error
<0	Failure

Note:

To configure No of selects Need to configure the bits[12:15] in ext_tcp_ip_feature_bit_map in rsi_wlan_config.h.

Need to configure the define RSI_NUMBER_OF_SELECTS to select no of selects. By default, two selects are enabled.

10.5.9 rsi_recvfrom

Prototype

```
int32_t rsi_recvfrom(uint32_t sockID,
 int8_t *buffer,
 int32_t buffersize,
 int32_t flags,
 struct sockaddr *fromAddr,
 int32_t *fromAddrLen);
```

Description
This API retrieves the received data from the remote peer on a given socket descriptor.

Precondition
rsi_socket()/rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This is the socket descriptor

Parameter	Description
Buffer	This is the pointer to buffer to hold receive data. This is an out parameter.
Buffersize	This is the size of the buffer supplied
flags	Reserved
fromAddr	This is the address of remote peer, from where current packet was received. It is an out parameter.
fromAddrLen	This is the pointer which contains remote peer address(fromAddr) length

Return Values

Value	Description
0	Number of bytes received successfully
Non-Zero Value	-1: Failure

Example

```
 status = rsi_recvfrom(new_socket, recv_buffer, recv_size, 0, (struct sockaddr *)&client_addr,
&addr_size);
```

10.5.10 rsi_recv

Prototype

```
int32_t rsi_recv(uint32_t sockID,
 VOID *rcvBuffer,
 int32_t bufferLength,
 int32_t flags);
```

Description
This API retrieves the data from the remote peer on a specified socket.

Precondition
rsi_socket()/rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This is the socket descriptor ID
rcvBuffer	This is the pointer to the buffer to hold the data received from the remote peer
bufferLength	This is the length of the buffer
flags	Reserved

Return Values

Value	Description
0	Number of bytes received successfully
Non-Zero Value	-1: Failure

Example

```
 status = rsi_recv(client_socket, (recv_buffer + recv_offset), recv_size, 0);
```

10.5.11 rsi_sendto

Prototype

```
int32_t rsi_sendto(uint32_t sockID,
 int8_t *msg,
 int32_t msgLength,
```

```
  int32_t flags,
  struct sockaddr *destAddr,
  int32_t destAddrLen);
```

Description
This API sends the data to a specified remote peer on a given socket.

Precondition
rsi_socket()/rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This is the Socket Descriptor ID
msg	This is the pointer to data buffer containing data to send to remote peer
msgLength	This is the length of the buffer
flags	Reserved
destAddr	This is the address of the remote peer to send data
destAddrLen	This is the length of the address in bytes

Return Values

Value	Description
0	Number of bytes sent successfully
Non-Zero Value	-1: Failure

Example

```
  status = rsi_sendto(client_socket, (int8_t *)"Hello from UDP client!!!",
(sizeof("Hello from UDP client!!!") - 1), 0, (struct sockaddr
*)&server_addr, sizeof(server_addr));
```

10.5.12 rsi_sendto_async

Prototype

```
int32_t rsi_sendto_async(int32_t sockID, int8_t *msg, int32_t msgLength, int32_t flags, struct
rsi_sockaddr *destAddr, int32_t destAddrLen,
                      void (*data_transfer_complete_handler)(int32_t sockID, uint16_t length))
```

Description
This function is used to send data to specific remote peer on a given socket asynchronously.

Parameters

Parameter	Description
sockID	This is the Socket Descriptor ID
msg	This is the pointer to data buffer containing data to send to remote peer
msgLength	This is the length of the buffer
flags	Reserved
destAddr	This is the address of the remote peer to send data
destAddrLen	This is the length of the address in bytes
data_transfer_complete_handler	This is pointer to the callback function which will be called after data transfer completion **Parameters** **sockID:** Socket Descriptor ID

Parameter	Description
	length: length of the buffer.

Return Values

Value	Description
0	If fails
>0	If success

Example

```
status = rsi_sendto_async(sockID, msg, msgLength, flags, destAddr, destAddrLen,
data_transfer_complete_handler);
```

10.5.13 rsi_send

Prototype

```
int32_t rsi_send(uint32_t sockID,
 const int8_t *msg,
 int32_t msgLength,
 int32_t flags);
```

Description
This API sends the data to remote peer on a given socket.

Precondition
rsi_socket()/rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This is the socket descriptor ID
msg	This is the pointer to the buffer containing data to send to the remote peer
msgLength	This is the length of the buffer
flags	Reserved

Return Values

Value	Description
0	Number of bytes sent successfully
Non-Zero Value	-1: Failure

Example

```
 status = rsi_send(client_socket,(int8_t *)"Hello from TCP client!!!", (sizeof("Hello from TCP
client!!!") - 1), 0);
```

10.5.14 rsi_send_async

Prototype

```
int32_t rsi_send_async(int32_t sockID, const int8_t *msg, int32_t msgLength, int32_t flags,
    void (*data_transfer_complete_handler)(int32_t sockID, uint16_t length))
```

Description
This API is used to send data on a given socket asynchronously.

Precondition
rsi_socket()/rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This is the socket descriptor ID
Msg	This is the pointer to the buffer containing data to send to the remote peer
msgLength	This is the length of the buffer
Flags	Reserved
data_transfer_complete_handler	pointer to the callback function which will be called after data transfer completion **Parameters** sockID: This is the socket descriptor ID length:

Return Values

Value	Description
0	Number of bytes sent successfully
Non-Zero Value	-1: Failure

Example

```
  rsi_send_async(client_socket,(int8_t *)"Hello from TCP client!!!", (sizeof("Hello from TCP
client!!!") - 1), 0,data_transfer_complete_handler);
```

10.5.15 rsi_shutdown

Prototype

```
int32_t rsi_shutdown(uint32_t sockID, int32_t how);
```

Description
This API closes the socket specified in a socket descriptor.

Precondition
rsi_socket()/rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
sockID	This is the socket descriptor ID
how	0: close the specified socket 1: close all the sockets open on specified socket's source port number. **Note:** Valid for passively open sockets (listen) with more than one backlogs specified.

Note:

This API in turn calls the rsi_socket_shutdown() API.

Note:

1. If EXT_TCP_IP_WAIT_FOR_SOCKET_CLOSE (BIT(16)) in RSI_EXT_TCPIP_FEATURE_BITMAP is enabled, then to close LTCP socket, argument "how" should be 1.

2. If EXT_TCP_IP_WAIT_FOR_SOCKET_CLOSE (BIT(16)) in RSI_EXT_TCPIP_FEATURE_BITMAP is enabled,Though remote socket has terminated connection**,** socket is disconnected and deleted only if host

issues rsi_shutdown().

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
status = rsi_shutdown(server_socket, 0);
```

10.5.16 rsi_getsockopt

Prototype

```
int rsi_getsockopt(int sockID, int level, int option_name,const void *option_value, rsi_socklen_t option_len)
```

Description
This API is used to get the socket options.

Parameters

parameter	Description
SockID	Socket descriptor
level	To set the socket option take the socket level
opt_name	provided the name of the ID SO_CHECK_CONNECTED_STATE - to check the socket connected state
opt_value	value of the parameter
opt_len	length of the parameter

Precondition

rsi_getsockopt() API needs to be called after rsi_socket command only.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	**If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021

Example

```
status = rsi_getsockopt(sockID,level,option_name,option_value,option_len);
```

10.5.17 rsi_setsockopt

Prototype

```
int rsi_setsockopt(int sockID, int level, int option_name,const void *option_value, rsi_socklen_t option_len)
```

Description

This API is used to set the socket options.

Parameters

Parameter	Description
sockID	Socket descriptor
level	To set the socket option take the socket level
option_name	provide the name of the ID 1.SO_MAX_RETRY - To select tcp max retry count 2.SO_SOCK_VAP_ID - To select the vap id 3.SO_TCP_KEEP_ALIVE-To configure the tcp keep alive initial time 4.SO_RCVBUF - To configure the application buffer for receiving server certificate 5.SO_SSL_ENABLE-To configure ssl socket 6.SO_HIGH_PERFORMANCE_SOCKET-To configure high performance socket
option_value	value of the parameter
option_len	length of the parameter

Default Valueslength of the parameter

Option_name	Default values
SO_MAX_RETRY	10
SO_SOCK_VAP_ID	0
SO_TCP_KEEP_ALIVE	1200 (in ms)
SO_RCVBUF	depends on size of the receive buffer mentioned in the application.
SO_SSL_ENABLE	0
SO_HIGH_PERFORMANCE_SOCKET	0

Precondition

rsi_setsockopt() API needs to be called after rsi_socket command only.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	**If return value is less than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021

Note: Please refer to Error Codes section for the description of the above error codes.

Example

NA

10.5.18 rsi_socket_async

Prototype

```
int32_t rsi_socket_async(int32_t protocolFamily,
int32_t type,
int32_t protocol,
void (callback*)(uint32_t sock_no,
uin8_t *buffer,
uint32_t length));
```

Description

This API creates a socket and register a callback which will be used by the driver to forward the received packets asynchronously to the application (on packet reception) without waiting for recv API call.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
protocolFamily	Protocol family to select IPv4 or IPv6 AF_INET (2) : to select IPv4 AF_INET6 (3) : to select IPv6
type	Select socket type UDP or TCP SOCK_STREAM (1) : to select TCP SOCK_DGRM (2) : to select UDP
protocol	0: non SSL sockets 1: SSL sockets BIT(5) should be enabled for selecting the certificate index 0<<12 for index 0 1<<12 for index 1 Note: For selecting the certificate index feature, certificates should be loaded in to ram.
callback	This is the callback function to read data asynchronously from socket sock_no: Application socket number buffer: Pointer to buffer to hold the data length: length of the buffer.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

10.5.19 rsi_fd_isset

Prototype

```
int rsi_fd_isset(uint32_t fd, struct rsi_fd_set_s *fds_p)
```

Description

This API checks for the bit set in the bitmap.

Parameters

Parameter	Description
fd	socket id
fds_p	pointer to the rsi_fd_set_structure

Return Values

Value	Description
0	If fails
>0	If success

Example

```
rsi_fd_isset(fd,fds_p);
```

10.5.20 rsi_set_fd

Prototype

```
void rsi_set_fd(uint32_t fd, struct rsi_fd_set_s *fds_p)
```

Description
This API sets the bit for the file descriptor fd in the file descriptor set rsi_fd_set.

Parameters

Parameter	Description
fd	socket id
fds_p	pointer to the rsi_fd_set_structure

Example
rsi_set_fd(fd,fds_p);

10.5.21 rsi_fd_clr

Prototype

void rsi_fd_clr(uint32_t fd, struct rsi_fd_set_s *fds_p)

Description
This API clears the bit in the bitmap.

Parameters

Parameter	Description
fd	socket id
fds_p	pointer to the rsi_fd_set_structure

Example

```
rsi_fd_clr(fd,fds_p);
```

10.5.22 rsi_select_set_status

Prototype

```
void rsi_select_set_status(int32_t status, int32_t selectid);
```

Description
This API is used to set the status of the socket specified in the select ID.

Precondition
rsi_socket()/rsi_socket_async() API needs to be called before this API.

Parameters

Parameter	Description
status	Status value to be set
selectid	This is the socket ID on which the status is set

Return Values

None

Example

```
status = rsi_select_set_status(status, id);
```

10.5.23 rsi_select_get_status

Prototype

```
int32_t rsi_select_get_status(int32_t selectid);
```

Description

This API is used to get the status of the socket specified in the select ID.

Precondition

None

Parameter

Parameter	Description
selectid	This is the socket ID to get the status

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Example

```
status = rsi_select_get_status(selectid);
```

10.6 DNS

10.6.1 rsi_dns_req

Prototype

```
int32_t rsi_dns_req(
uint8_t ip_version,
uint8_t *url_name,
uint8_t *primary_server_address,
uint8_t *secondary_server_address,
rsi_rsp_dns_query_t *dns_query_resp,
uint16_t length)
```

Description
This API queries the IP address of a given domain name.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
ip_version	IP version 4: IPv4 6: IPv6
url_name	This is the pointer to the domain name to resolve IP address
primary_server_address	This is the IP address of the DNS server. This parameter is optional if module get DNS server address using DHCP.
secondry_server_address	This is IP address of the secondary DNS server. In case of no secondary dns server ip give it NULL
dns_query_resp	This is the pointer to hold DNS query results. This is an out parameter.
length	This is the length of the resultant buffer.

DNS results response format

```
typedef struct rsi_rsp_dns_query_s
 {
 uint8_t ip_version[2];
 uint8_t ip_count[2];
 union
 {
 uint8_t ipv4_address[4];
 uint8_t ipv6_address[16];
 }ip_address[10];
 } rsi_rsp_dns_query_t;
```

Structure Field	Description
ip_version	This is the IP version 4: IPv4 6: IPv6
ip_count	This is the number of IP addresses resolved for a given domain name
ip_address	This is the IP address of a given domain name.This field is union of IPv4 and IPv6 address (16 bytes in size). The number of bytes filled depends on the IP version.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_dns_req(RSI_IP_VERSION_4,(uint8_t *)RSI_DNS_URL_NAME,(uint8_t *)&server_address, NULL,
&dns_query_resp,sizeof(rsi_rsp_dns_query_t));
```

10.6.2 rsi_dns_update

Prototype

```
int32_t rsi_dns_update(
uint8_t ip_version,
uint8_t *zone_name,
uint8_t *host_name,
uint8_t *server_address,
uint16_t *ttl,
void(*dns_update_rsp_handler)(uint16_t status) );
```

Description
This API updates the host name for a given host and zone name.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
ip_version	This is the IP version 4: IPv4 6: IPv6
zone_name	This is the pointer to a zone name and to update host name
host_name	This is a pointer to a host name to update a host name
server_address	This is the IP address of the DNS server. This parameter is optional if module get DNS server address using DHCP.
ttl	This is the time to live value of the host name.
dns_update_rsp_handler	This is the call back function called by driver on reception of dns update response.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	-1: Failure

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_dns_update(RSI_IP_VERSION_4, (uint8_t *)RSI_DNS_ZONE_NAME, (uint8_t *)RSI_DNS_HOST_NAME,
(uint8_t *)&server_address, (uint16_t)RSI_DNS_TTL, rsi_dns_response_handler);
```

10.7 FWUP

10.7.1 rsi_fwup

Prototype

```
static int32_t rsi_fwup(uint8_t type, uint8_t *content, uint16_t length)
```

Description
This API is a helper function for actual APIs

Parameters

Parameter	Description
type	firmware upgrade chunk type
content	firmware content
length	length of the content

Return Values

Value	Description
0	success
3	Firmware up-gradation completed successfully
<0 =	failure

Example

```
rsi_fwup(type,content,length) ;
```

10.7.2 rsi_fwup_start

Prototype

```
int32_t rsi_fwup_start(
 uint8_t *rps_header);
```

Description
This API sends the RPS header content of firmware file.

Precondition
rsi_wlan_radio_init() API needs to be called before this API.

Parameters

Parameter	Description
rps_header	This is the pointer to the rps header content

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	Failure **If return value is lesser than 0** -2: Invalid Parameters -4: Buffer not available to serve the command

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
uint8_t recv_buffer[1000];
rsi_fwup_start(recv_buffer);
```

10.7.3 rsi_fwup_load

Prototype

```
int32_t rsi_fwup_load(
 uint8_t *content,
 uint16_t length);
```

Description
This API sends the firmware file content.

Precondition
rsi_wlan_radio_init() API needs to be called before this API

Parameters

Parameter	Description
content	This is the pointer to the firmware file content
length	This is the length of the content

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	3: Firmware upgradation completed successfully **If return value is lesser than 0** -2: Invalid Parameters -4: Buffer not available to serve the command

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
// see rsi_firmware_upgradation_app.c for a detailed example
fwup_chunk_length = rsi_bytes2R_to_uint16(&recv_buffer[1]);
rsi_fwup_load(recv_buffer, fwup_chunk_length);
```

10.8 FTP Client API

10.8.1 rsi_ftp_connect

Prototype

```
int32_t rsi_ftp_connect(uint16_t flags, int8_t *server_ip, int8_t *username, int8_t *password, uint32_t
server_port)
```

Description
This API creates FTP objects and connects to the FTP server on the given server port. This should be the first command for accessing FTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
flags	This are the network flags. Each bit in the flag has its own significance BIT(0) – RSI_IPV6 Set this bit to enable IPv6. By default it is configured to IPv4. BIT(1) to BIT(15) are reserved for future use
server_ip	This is the FTP server IP address to connect
username	This is the username for server authentication
password	This is the password for server authentication
server_port	This is the port number of FTP server **Note:** FTP server port is configurable on nonstandard port also

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
 //! Connect to FTP Server

 retval = rsi_ftp_connect(RSI_IP_VERSION_4,(uint8_t
*)&server_ip,FTP_SERVER_LOGIN_USERNAME,FTP_SERVER_LOGIN_PASSWORD,FTP_SERVER_PORT);
 if(retval != RSI_SUCCESS)
 {
 return retval;
 }
```

10.8.2 rsi_ftp_disconnect

Prototype

```
int32_t rsi_ftp_disconnect(void)
```

Description
This function disconnects from the FTP server and also destroys the FTP objects. Once the FTP objects are destroyed, FTP server cannot be accessed. For further accessing, FTP objects should be created again.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Disconnect from FTP server
retval = rsi_ftp_disconnect();
if(retval != RSI_SUCCESS)
{
return retval;
 }
```

10.8.3 rsi_ftp_file_write

Prototype

```
int32_t rsi_ftp_file_write(int8_t *file_name)
```

Description
This function opens a file in the specified path on the FTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
file_name	This is the file name or filename including path can be given. e.g "example.txt" or "/test/ftp/example.txt"

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! File Write
retval = rsi_ftp_file_write(FTP_FILE_TO_WRITE);
```

10.8.4 rsi_ftp_file_write_content

Prototype

```
int32_t rsi_ftp_file_write_content(uint16_t flags, int8_t *file_content,int16_t content_length,uint8_t end_of_file)
```

Description
This function writes the content into the file which is opened using rsi_ftp_file_write() API.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
flags	These are the network flags. Each bit in the flag has its own significance BIT(0) – RSI_IPV6 Set this bit to enable IPv6. By default, it is configured to IPv4 BIT(1) to BIT(15) are reserved for future use
file_content	This is the data stream to be written into the file
content_length	This is the file content length
end_of_file	This flag indicates the end of file 1 – This chunk represents the end of content to be written into the file 0 – This are the extra data which is pending to write into the file **Note:** This API can be called multiple times to append data into the same file and at the last chunk ,this flag should be 1.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Note:

File content length should not exceed 1344 bytes in case of IPV4 and 1324 bytes in case of IPV6.If exceeds, this API will break the file content and send it in multiple packets.

Example

```
//! File Write Content
retval = rsi_ftp_file_write_content(0, file_data_read,file_data_length,1);
```

10.8.5 rsi_ftp_file_read_async

Prototype

```
int32_t rsi_ftp_file_read_aysnc(int8_t *file_name, void
(*call_back_handler_ptr)(uint16_t status, int8_t *file_content, uint16_t
 content_length, uint8_t end_of_file))
```

Description
This function reads the content from the specified file on the FTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
file_name	This is the file name or filename including path can be given. e.g "example.txt" or "/test/ftp/example.txt"
call_back_handler_ptr	This is the callback function when asynchronous response comes for the file read request. The parameters involved are :status , file_content, content_length & end_of_file **status:** This is the status code. The other parameters are valid only if status is 0 **file_content:** file content **content_length:** This is the length of file content **end_of_file:** This indicates the end of file 1 – No more data 0 – more data present

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: **Invalid parameter,expects call back handler** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Read the file in FTP server
retval = rsi_ftp_file_read_aysnc(FTP_FILE_TO_READ,rsi_file_read_cb);
```

10.8.6 rsi_ftp_file_delete

Prototype

```
int32_t rsi_ftp_file_delete(int8_t *file_name)
```

Description
This API deletes the file which is present in the specified path on the FTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
file_name	This is the file name or filename including path can be given to delete e.g "example.txt" or "/test/ftp/example.txt"

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_ftp_file_delete(file_name);
```

10.8.7 rsi_ftp_file_rename

Prototype

```
int32_t rsi_ftp_file_rename(int8_t *old_file_name, int8_t *new_file_name)
```

Description
This AP rename the file with a new name on the FTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API

Parameters

Parameter	Description
old_file_name	This is the filename/file name which has to be renamed
new_file_name	This is the new file name

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_ftp_file_rename(old_file_name , new_file_name);
```

10.8.8 rsi_ftp_directory_create

Prototype

```
int32_t rsi_ftp_directory_create(int8_t *directory_name)
```

Description
This API creates a directory on the FTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
directory_name	This is the directory name (with path if required) to create e.g. "example" or "/test/ftp/example"

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_ftp_directory_create(directory_name);
```

10.8.9 rsi_ftp_directory_delete

Prototype

```
int32_t rsi_ftp_directory_delete(int8_t *directory_name)
```

Description
This API deletes the directory on the FTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API

Parameters

Parameter	Description
directory_name	directory name (with path if required) to delete e.g. "example" or "/test/ftp/example"

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_ftp_directory_delete(directory_name);
```

10.8.10 rsi_ftp_directory_set

Prototype

```
int32_t rsi_ftp_directory_set(int8_t *directory_name)
```

Description
This function changes the current working directory to the specified directory path on the FTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
directory_name	This is the directory name (with path if required) to create

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_ftp_directory_set(directory_name);
```

10.8.11 rsi_ftp_directory_list_async

Prototype

```
int32_t rsi_ftp_directory_list_async(int8_t *directory_path,void
(*call_back_handler_ptr)(uint16_t status, int8_t *directory_list,
uint16_t length , uint8_t end_of_list))
```

Description
This function gets the list of directories present in the specified directory on the FTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
file_name	This is the file name or filename including path can be given. e.g "example.txt" or "/test/ftp/example.txt"
call_back_handler_ptr	This is the callback function when asynchronous response comes for the directory list request The parameters involved are: status , directory_list, length and end_of_list status: status code.Other parameters are valid only if status is 0 **directory_list:** This is the stream of data with directory list as content length: This is the length of content end_of_list: This indicates end of list 1 – No more data 0 – more data present

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid parameter, expects call back handler -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_ftp_directory_list_async(/test/ftp/example.txt,call_back_handler_ptr);
```

10.8.12 rsi_ftp_mode_set

Prototype

```
int32_t rsi_ftp_mode_set(uint8_t mode)
```

Description
This function sets the FTP client mode - either in Passive mode or Active Mode.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
Mode	Used to select the mode of FTP client if FTP is enabled 0-Active Mode 1-Passive Mode.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid parameter, expects call back handler -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
// set to Active Mode
rsi_ftp_mode_set(0);
```

10.9 HTTP Client API

10.9.1 rsi_http_client_get_async

Prototype

```
int32_t rsi_http_client_get_async(uint8_t flags,
uint8_t *ip_address,
uint16_t port,
uint8_t *resource,
uint8_t *host_name,
uint8_t *extended_header,
uint8_t *user_name,
uint8_t *password,
void(*http_client_get_response_handler)
(uint16_t status,
const uint8_t *buffer,
const uint16_t length)
const uint32_t more_data);
```

The above API is deprecated, the new form of API is described below and is available when the flag RSI_HTTP_STATUS_INDICATION_EN is enabled.

```
int32_t rsi_http_client_get_async(uint8_t flags,
uint8_t *ip_address,
uint16_t port,
uint8_t *resource,
uint8_t *host_name,
uint8_t *extended_header,
uint8_t *user_name,
uint8_t *password,
void(*http_client_get_response_handler)
(uint16_t status,
```

```
const uint8_t *buffer,
const uint16_t length)
const uint32_t more_data,
const uint16_t status_code);
```

Description
This API sends http get request to remote HTTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
flags	To select IP version and security BIT(0) – RSI_IPV6 Set this bit to enable IPv6 ,by default it is configured to IPv4 BIT(1) – RSI_SSL_ENABLE Set this bit to enable SSL feature, if this is enabled then by default BIT(2) - RSI_SSL_V_1_0 Set this bit to support SSL TLS Version 1.0 if HTTPS is enabled. BIT(3) - RSI_SSL_V_1_2 Set this bit to support SSL_TLS Version 1.2 if HTTPS is enabled BIT(4) - RSI_SSL_V_1_1 Set this bit to support SSL_TLS Version 1.1 if HTTPS is enabled BIT(5)_ HTTP_POST_DATA Set this bit to enable http_post data feature BIT(6)_ HTTP_V_1_1 Set this bit to use http version 1.1 BIT(7)_HTTP_USER_DEFINED_CONTENT_TYPE Set this bit to enable user defined http_content_type
ip_address	This is the server IP address
port	This is the port number of HTTP server **Note:** HTTP server port is configurable on non-standard port also
resource	This is the URL string for requested resource
hostname	This is the host name
extended_header	This is the user defined extended header
username	This is the username for server Authentication
password	This is the password for server Authentication
http_client_get_response_handler	This is the callback when asynchronous response comes for the request The parameters involved are: status, buffer, length & more_data **status:** This is the status of response from module. This will return failure upon an internal error only. **buffer**: This is the buffer pointer **length**: This is the length of data **more_data**: 1 – No more data 0 – more data present **status_code:** This is HTTP response code as returned by server in HTTP header. e.g. 200, 201, 404 etc. This field is valid only when status field (first argument) is

Parameter	Description
	success indicating a response is received from HTTP server. A status_code equal to 0 indicates that there was no HTTP header in the received packet, probably a continuation of the frame body received in the previous chunk. This field is available, if the feature RSI_HTTP_STATUS_INDICATION_EN is enabled in rsi_wlan_config.h

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters, call back not registered -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! send http get request for the given url

 status = rsi_http_client_get_async((uint8_t)flags, (uint8_t
*)HTTP_SERVER_IP_ADDRESS, (uint16_t)HTTP_PORT, (uint8_t *)HTTP_URL,
(uint8_t *)HTTP_HOSTNAME,(uint8_t *)HTTP_EXTENDED_HEADER,(uint8_t
*)USERNAME,(uint8_t *)PASSWORD,
 rsi_http_get_response_handler);
```

10.9.2 rsi_http_client_post_async

Prototype

```
int32_t rsi_http_client_post_async(uint8_t flags,
uint8_t *ip_address,
uint16_t port,
uint8_t *resource,
uint8_t *host_name,
uint8_t *extended_header,
uint8_t *user_name,
uint8_t *password,
uint8_t *post_data,
uint32_t post_data_length,
void(*http_client_post_response_handler)
(uint16_t status,
const uint8_t *buffer,
const uint16_t length)
const uint32_t more_data);
```

The above API is deprecated, the new form of API is described below and is available when the flag RSI_HTTP_STATUS_INDICATION_EN is enabled

```
int32_t rsi_http_client_post_async(uint8_t flags,
uint8_t *ip_address,
uint16_t port,
uint8_t *resource,
uint8_t *host_name,
uint8_t *extended_header,
uint8_t *user_name,
uint8_t *password,
uint8_t *post_data,
uint32_t post_data_length,
void(*http_client_post_response_handler)
(uint16_t status,
const uint8_t *buffer,
const uint16_t length)
const uint32_t more_data,
const uint16_t status_code
);
```

Description

This API sends http post request to remote HTTP server.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
Flags	To select IP version and security BIT(0) – RSI_IPV6 Set this bit to enable IPv6 , by default it is configured to IPv4 BIT(1) – RSI_SSL_ENABLE Set this bit to enable SSL feature BIT(2) - RSI_SSL_V_1_0 Set this bit to support SSL TLS Version 1.0 if HTTPS is enabled. BIT(3) - RSI_SSL_V_1_2 Set this bit to support SSL_TLS Version 1.2 if HTTPS is enabled BIT(4) - RSI_SSL_V_1_1 Set this bit to support SSL_TLS Version 1.1 if HTTPS is enabled BIT(5)_ HTTP_POST_DATA Set this bit to enable HTTP_POST large data feature BIT(6)_ HTTP_V_1_1 Set this bit to use HTTP version 1.1 BIT(7)_HTTP_USER_DEFINED_CONTENT_TYPE Set this bit to enable user defined http_content_type
ip_address	This is the server IP address
port_no	This is the port number of HTTP server
resource	This is the URL string for requested resource
hostname	This is the host name
extended_header	This is the user defined extended header
username	This is the username for server Authentication

Parameter	Description
password	This is the password for server Authentication
post_data	The is the HTTP data to be posted to server
post_data_length	This is the post data length
http_client_post_response_handler	This is the callback when asynchronous response comes for the request The parameters involved are: status , buffer, length, more_data, and status_code as returned in HTTP header by server **status:** This is the status of response from module. This will return failure upon an internal error only. **buffer:** This is the buffer pointer **length:** This is the length of data **more_data:** 1 – No more data 0 – more data present 2 – HTTP post success response **status_code:** This is HTTP response code as returned by server in HTTP header. e.g. 200, 201, 404 etc. This field is valid only when status field (first argument) is success indicating a response is received from HTTP server. A status_code equal to 0 indicates that there was no HTTP header in the received packet, probably a continuation of the frame body received in the previous chunk. This field is available, if the feature RSI_HTTP_STATUS_INDICATION_EN is enabled in rsi_wlan_config.h

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid parameters, call back not registered -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! send http post request for the given url with given http data

 status = rsi_http_client_post_async((uint8_t)FLAGS, (uint8_t
*)HTTP_SERVER_IP_ADDRESS, (uint16_t)HTTP_PORT, (uint8_t *)HTTP_URL,
(uint8_t *)HTTP_HOSTNAME, (uint8_t *)HTTP_EXTENDED_HEADER, (uint8_t
*)USERNAME, (uint8_t *)PASSWORD,(uint8_t *)HTTP_DATA, strlen(HTTP_DATA),
 rsi_http_post_response_handler);
```

10.9.3 rsi_http_client_async

Prototype

```
int32_t rsi_http_client_async(uint8_t type,
uint8_t flags,
uint8_t *ip_address,
uint16_t port,
uint8_t *resource,
uint8_t *host_name,
```

```
uint8_t *extended_header,
uint8_t *user_name,
uint8_t *password,
uint8_t *post_data,
uint32_t post_data_length,
void(*callback)(uint16_t status,
const uint8_t *buffer,
const uint16_t length,
uint32_t moredata));
```

The above API is deprecated, the new form of API is described below and is available when the flag RSI_HTTP_STATUS_INDICATION_EN is enabled.

```
int32_t rsi_http_client_async(uint8_t type,
uint8_t flags,
uint8_t *ip_address,
uint16_t port,
uint8_t *resource,
uint8_t *host_name,
uint8_t *extended_header,
uint8_t *user_name,
uint8_t *password,
uint8_t *post_data,
uint32_t post_data_length,
void(*callback)(uint16_t status,
const uint8_t *buffer,
const uint16_t length,
uint32_t moredata,
uint16_t status_code));
```

Description
This API sends http get request/http post request to remote HTTP server based on the type selected.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
type	0- RSI_HTTP_GET 1- RSI_HTTP_POST
flags	To select IP version and security BIT(0) – RSI_IPV6 Set this bit to enable IPv6 , by default it is configured to IPv4 BIT(1) – RSI_SSL_ENABLE Set this bit to enable SSL feature BIT(2) - RSI_SSL_V_1_0 Set this bit to support SSL TLS Version 1.0 if HTTPS is enabled. BIT(3) - RSI_SSL_V_1_2 Set this bit to support SSL_TLS Version 1.2 if HTTPS is enabled BIT(4) - RSI_SSL_V_1_1 Set this bit to support SSL_TLS Version 1.1 if HTTPS is enabled BIT(5)_ HTTP_POST_DATA Set this bit to enable HTTP_POST large data feature BIT(6)_ HTTP_V_1_1 Set this bit to use HTTP version 1.1

Parameter	Description
	BIT(7)_HTTP_USER_DEFINED_CONTENT_TYPE Set this bit to enable user defined http_content type.
ip_address	This is the server IP address
port	This is the port number of HTTP server **Note:** HTTP server port is configurable on non-standard port also
resource	This is the URL string for requested resource
hostname	This is the host name
extended_header	This is the user defined extended header
username	This is the username for server Authentication
password	This is the password for server Authentication
post_data	The is the HTTP data to be posted to server
post_data_length	This is the post data length
http_client_get_response_handler	This is the callback when asynchronous response comes for the request The parameters involved are: status, buffer, length & more_data **status:** This is the status of response from module. This will return failure upon an internal error only. **buffer**: This is the buffer pointer **length**: This is the length of data **more_data**: 1 – No more data 0 – more data present **status_code:** This is HTTP response code as returned by server in HTTP header. e.g. 200, 201, 404 etc. This field is valid only when status field (first argument) is success indicating a response is received from HTTP server. A status_code equal to 0 indicates that there was no HTTP header in the received packet, probably a continuation of the frame body received in the previous chunk. This field is available, if the feature RSI_HTTP_STATUS_INDICATION_EN is enabled in rsi_wlan_config.h
http_client_post_response_handler	This is the callback when asynchronous response comes for the request The parameters involved are : status , buffer, length and more_data **status:** This is the status code **buffer:** This is the buffer pointer **length:** This is the length of data **more_data:** 1 – No more data 0 – more data present 2 – HTTP post success response **status_code:** This is HTTP response code as returned by server in HTTP header. e.g. 200, 201, 404 etc. This field is valid only when status field (first argument) is success indicating a response is received from HTTP server. A status_code equal to 0 indicates that there was no HTTP header in the received packet, probably a continuation of the frame body received in the previous chunk. This field is available, if the feature RSI_HTTP_STATUS_INDICATION_EN is enabled in rsi_wlan_config.h

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid parameters, call back not registered -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! send http get request for the given url
status = rsi_http_client_async(RSI_HTTP_GET, flags, ip_address, port, resource, host_name,
extended_header, user_name, password, NULL, 0, http_client_get_response_handler);
 status = rsi_http_client_async(RSI_HTTP_POST, flags, ip_address, port, resource, host_name,
extended_header, user_name, password, post_data, post_data_length, http_client_post_response_handler);
```

10.9.4 rsi_http_client_post_data

Prototype

```
int32_t rsi_http_client_post_data(uint8_t *file_content,
uint16_t current_chunk_length,
void(*http_client_post_data_response_handler)
(uint16_t status,
const uint8_t *buffer,
const uint16_t length)
const uint32_t more_data);
```

The above API is deprecated, the new form of API is described below and is available when the flag RSI_HTTP_STATUS_INDICATION_EN is enabled.

```
int32_t rsi_http_client_post_data(uint8_t *file_content,
uint16_t current_chunk_length,
void(*http_client_post_data_response_handler)
(uint16_t status,
const uint8_t *buffer,
const uint16_t length)
const uint32_t more_data,
const uint16_t status_code);
```

Description
This API sends the http post data packet to remote HTTP server.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
file_content	This is the user given http file content
current_chunk_length	This is the length of the current http data

Parameter	Description
http_client_post_data_response_handler	This is the callback when asynchronous response comes for the request. The parameters involved are: status, buffer, length and more_data **status:** This is the status of response from module. This will return failure upon an internal error only. **buffer**: This is the buffer pointer **length**:This is the length of data **more_data**: 1 – No more data 0 – more data 4 – HTTP post data response 8 – HTTP post data receive response **status_code:** This is HTTP response code as returned by server in HTTP header. e.g. 200, 201, 404 etc. This field is valid only when status field (first argument) is success indicating a response is received from HTTP server. A status_code equal to 0 indicates that there was no HTTP header in the received packet, probably a continuation of the frame body received in the previous chunk. This field is available, if the feature RSI_HTTP_STATUS_INDICATION_EN is enabled in rsi_wlan_config.h

Return Value

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters, call back not registered -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0, 0xBB38, 0xBB3E, 0xBBEF

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
 //! send http post data request for the given url with given http data
 status = rsi_http_client_post_data((uint8_t *)file_content, (uint16_t)chunk_length,
rsi_http_post_data_response_handler);
```

10.9.5 rsi_http_client_put_create

Prototype

```
int32_t rsi_http_client_put_create(void);
```

Description
This API creates the http put client

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0015, 0x0025, 0xBB38.

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Create HTTP client for PUT method

status = rsi_http_client_put_create();
```

10.9.6 rsi_http_client_put_start

Prototype

```
int32_t rsi_http_client_put_start(uint8_t flags,
uint8_t *ip_address,
uint32_t port_number,
uint8_t *resource,
uint8_t *host_name,
uint8_t *extended_header,
uint8_t *user_name,
uint8_t *password,
uint32_t content_length,
void(*callback)
(uint16_t status,
uint8_t type,
const uint8_t *buffer,
uint16_t length,
const uint8_t end_of_put_pkt));
```

Description

This API starts the http client put process

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
Flags	To select IP version and security BIT(0) – RSI_IPV6 Set this bit to enable IPv6 , by default it is configured to IPv4 BIT(1) – RSI_SSL_ENABLE Set this bit to enable SSL feature BIT(2) - RSI_SSL_V_1_0 Set this bit to support SSL TLS Version 1.0 if HTTPS is enabled. BIT(3) - RSI_SSL_V_1_2 Set this bit to support SSL_TLS Version 1.2 if HTTPS is enabled BIT(4) - RSI_SSL_V_1_1 Set this bit to support SSL_TLS Version 1.1 if HTTPS is enabled

Parameter	Description
	BIT(5)_POST_DATA Set this bit to enable Http_post large data feature BIT(6)_ HTTP_V_1_1 Set this bit to use HTTP version 1.1 BIT(7)_HTTP_USER_DEFINED_CONTENT_TYPE Set this bit to enable user defined http_content type.
ip_address	This is the server IP address
port_no	This is the port number of HTTP server
resource	THis is the URL string for requested resource
hostname	This is the host name
extended_header	This is the user defined extended header
username	This is the username for server Authentication
password	This is the password for server Authentication
content lenght	This is the total length of http data
http_client_put_response_handler	This is the callback when asynchronous response comes for the request The parameters involved are : status , type, buffer, length, end_of_put_pkt **status:** This is the status code **type:** This is the HTTP Client PUT command type **buffer:** This is the buffer pointer **length:** This is the length of data **end_of_put_pkt:** This is the End of file or HTTP resource content 0 - More data is pending from host 1 - End of HTTP file/resource content

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid parameters, call back not registered -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0015, 0x0025, 0xBB38.

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Start HTTP client PUT method for the given URL

status = rsi_http_client_put_start((uint8_t)flags,(uint8_t
```

```
*)HTTP_SERVER_IP_ADDRESS,(uint32_t)HTTP_PORT,(uint8_t *)HTTP_URL,(uint8_t *)HTTP_HOSTNAME,(uint8_t
*)HTTP_EXTENDED_HEADER,(uint8_t *)USERNAME,(uint8_t *)PASSWORD,(uint32_t)(sizeof(rsi_index)-1),
rsi_http_client_put_response_handler);
```

10.9.7 rsi_http_client_put_pkt

Prototype

```
int32_t rsi_http_client_put_pkt(uint8_t *file_content, uint16_t current_chunk_length);
```

Description
This API uses to put http data onto the http server for the created URL resource.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
file_content	This is the HTTP buffer contains the put data content
current_chunk_length	This is the length of the current http put content chunk

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0015, 0x0025, 0xBB38.

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Send resource content to the HTTP server for the given URL and total content length
status = rsi_http_client_put_pkt((uint8_t *)file_content,(uint16_t)chunk_length);
```

HTTP_client_put_pkt server response structure

```
//! HTTP Client PUT pkt server response structure
typedef struct http_Put_Data_s
{
    uint32_t command_type;
    uint32_t more;
    uint32_t offset;
    uint32_t data_len;
}http_Put_Data_t;
```

10.9.8 rsi_http_client_put_delete

Prototype

```
int32_t rsi_http_client_put_delete(void);
```

Description
This API deletes the created http put client

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0015, 0x0025, 0xBB38.

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Delete HTTP Client

status = rsi_http_client_put_delete();
```

10.9.9 rsi_http_client_abort

Prototype

```
int32_t rsi_http_client_abort(void)
```

Description
This API aborts any ongoing HTTP request from the client.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
http_client_abort();
```

10.9.10 rsi_http_credentials

Prototype

```
int32_t rsi_http_credentials(int8_t *username , int8_t *password)
```

Description
This API creates a http server credentials request. This API set username and password for http server.

Precondition
We can call this API after rsi_wireless_init API.

Parameters

Parameter	Description
username	This is the user given username
password	This is the user given password

Note-

1. Its not mandatory to call this api.
2. Default username and password are respectively redpine and admin

Return Value

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state **If return value is greater than 0** 0x0021,0x0025, 0x00F1,0x001

Note:
Please refer to Error Codes section for the description of the above error codes.

Example

```
//! send http server credentials request for a given username and password
status = rsi_http_credentials((int8_t *)HTTP_SERVER_USERNAME ,(int8_t *)HTTP_SERVER_PASSWORD );
  if(status != RSI_SUCCESS)
  {
    return status;
  }
```

10.10 Network Application Protocol

10.10.1 SMTP client API

10.10.1.1 rsi_smtp_client_create

Prototype

```
int32_t rsi_smtp_client_create(uint8_t flags,
uint8_t *username,
uint8_t *password,
uint8_t *from_address,
uint8_t *client_domain,
uint8_t auth_type,
uint8_t *server_ip,
uint32_t port);
```

Description

This API creates a smtp client. This initializes the client with a given configuration.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
Flags	To select IPv6 version, a bit in flags is set. By default IP version is set to IPV4. RSI_IPV6 – BIT(0) To select IPv6 version
username	This is the username for authentication It should be a NULL terminated string
password	This is the password for authentication It should be a NULL terminated string
from_address	This is the sender's address It should be a NULL terminated string
client_domain	This is the domain name of the client It should be a NULL terminated string
auth_type	This is the client authentication type 1 - SMTP_CLIENT_AUTH_LOGIN 3 - SMTP_CLIENT_AUTH_PLAIN
server_ip	This is the SMTP server IP address IPv4 address – 4 Bytes hexa-decimal, IPv6 address – 16 Bytes hexa-decimal
Port	This is the SMTP server TCP port **Note:** SMTP server port is configurable on non-standard port also

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3 : Command given in wrong state -4 : Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015,0xBBA5,0xBB21,0x003E,0xBBB2

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
 //! create smtp client

 status = rsi_smtp_client_create((uint8_t )FLAGS, (uint8_t *)USERNAME,
(uint8_t *)PASSWORD,(uint8_t *)FROM_ADDRESS, (uint8_t *)CLIENT_DOMAIN,
```

```
(uint8_t)AUTH_TYPE, (uint8_t *)&server_ip, (uint16_t)SMTP_PORT);
 if(status != RSI_SUCCESS)
 {
 return status;
 }
```

10.10.1.2 rsi_smtp_client_mail_send_async

Prototype

```
int32_t rsi_smtp_client_mail_send_async(
uint8_t *mail_recipient_address,
uint8_t priority,
uint8_t *mail_subject,
uint8_t *main_body,
uint16_t mail_body_length,
void(*smtp_client_mail_response_handler)
(uint16_t status,
const uint8_t cmd));
```

Description

This API sends mail to the recipient from the smtp client.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
mail_recipient_address	This is the mail recipient address
priority	This is the priority level at which mail is delivered 1 - RSI_SMTP_MAIL_PRIORITY_LOW 2- RSI_SMTP_MAIL_PRIORITY_NORMAL 4 - RSI_SMTP_MAIL_PRIORITY_HIGH
mail_subject	This is the Subject line text It is a null terminated string.
mail_body	This is the mail message
mail_body_length	This is the length of the mail body The total maximum length of mail_recipient_address, mail_subject & mail_body is 1024 bytes
smtp_client_mail_response_handler	This is the callback when asynchronous response comes from the sent mail The parameters involved are: status and cmd status: status code cmd: sub command type

Note:

If the status in callback is nonzero, then the sub command type will be in 6th byte of the descriptor.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015,0x003E,0xBBA5,0xBBA3,0xBBA0,0xBBA1,0xBBA2,OxBBA4,OxBBA6,0xB

Value	Description
	BA7,0xBBA8,0xBBA9,0xBBAA,0xBBAB,0xBBAC,0xBBAD,0xBBAE,0xBBAF,0xBBB0,0xBBB1,0xBBB2

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
 //! send mail to a SMTP server from our client
 status = rsi_smtp_client_mail_send_async((uint8_t )MAIL_RECIPIENT_ADDRESS, (uint8_t)PRIORITY, (uint8_t *)MAIL_SUBJECT,(uint8_t *)MAIL_BODY, strlen((const char)MAIL_BODY),rsi_smtp_client_mail_send_response_handler);
 if(status != RSI_SUCCESS)
 {
 return status;
 }
```

10.10.1.3 rsi_smtp_client_delete_async

Prototype

```
int32_t rsi_smtp_client_delete_async(
void(*smtp_client_mail_response_handler)
(uint16_t status,
const uint8_t cmd));
```

Description

This API deletes the smtp client.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
smtp_client_delete_response_handler	This is the callback when asynchronous response comes for the delete request The parameters involved are: status & cmd status:This is the status code cmd: This is the sub command type

Note:

If the status in callback is nonzero, then the sub command type will be in 6th byte of the descriptor.

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! send smtp client delete request
status = rsi_smtp_client_delete_async(rsi_smtp_client_delete_response_handler);
```

10.10.2 SNTP Client API

10.10.2.1 rsi_ sntp_client_create_async

Prototype

```
int32_t rsi_sntp_client_create_async(uint8_t flags, uint8_t *server_ip,
uint8_t sntp_method, uint16_t
sntp_timeout,void(*rsi_sntp_client_create_response_handler)(uint16_t
status,const uint8_t cmd_typr, const uint8_t *buffer));
```

Description
This API creates the sntp client.

Parameters

Parameter	Description
flags	To select IP version and security BIT(0) – RSI_IPV6 Set this bit to enable IPv6, by default it is configured to IPv4 BIT(1) – RSI_SSL_ENABLE Set this bit to enable SSL feature
Server_ip	This is the server IP address
sntp_method	These are the SNTP methods to use 1-For Broadcast Method 2-For Unicast Method
sntp_timeout	This is the SNTP timeout value
rsi_sntp_client_create_response_handler	This is the callback function when asynchronous response comes for the request. The parameters involved are: status, cmd_type and buffer status: This is the status code cmd_type: This is the command type buffer: This is the buffer pointer

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid parameters, call back not registered -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x002C, 0x0015, 0x0025, 0xFF74, 0xBBF0

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_sntp_client_create_async((uint8_t) FLAGS, (uint8_t
*)&server_ip, (uint8_t)SNTP_METHOD,(uint16_t)SNTP_TIMEOUT,
rsi_sntp_client_create_response_handler);
```

10.10.2.2 rsi_sntp_client_gettime

Prototype

```
int32_t rsi_sntp_client_gettime(uint16_t length,uint8_t*sntp_time_rsp);
```

Description

This API gets the current time parameters.

Parameters

Parameter	Description
Length	This is the length of the buffer
sntp_time_rsp	This is the current time response

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters, call back not registered -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x002C, 0x0015, 0x0025, 0xFF74, 0xBBF0

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_sntp_client_gettime((uint16_t)length, (uint8_t *)sntp_time);
```

10.10.2.3 rsi_sntp_ client_gettime_date

Prototype

```
int32_t rsi_sntp_client_gettime_date(uint16_t length, uint8_t *sntp_time_date_rsp)
```

Description

This API gets the current time in time date format parameters.

Parameters

Parameter	Description
Length	This is the length of the buffer
sntp_time_date_rsp	This is the current time and date response

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid parameters, call back not registered -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x002C, 0x0015, 0x0025, 0xFF74, 0xBBF0

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_sntp_client_gettime_date((uint16_t)length, (uint8_t *)sntp_date_time);
```

10.10.2.4 rsi_ sntp_client_server_info

Prototype

```
int33_t rsi_sntp_client_info(uint16_t length, uint8_t *sntp_server_response);
```

Description

This API gets the parameters for SNTP server details.

Parameters

Parameter	Description
Length	This is the length of the buffer
sntp_server_response	This is the function to get the server details

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid parameters, call back not registered -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x002C, 0x0015, 0x0025, 0xFF74, 0xBBF0

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_sntp_client_server_info((uint16_t)length, (uint8_t *)&sntp_server_info_rsp);
```

10.10.2.5 rsi_ sntp_client_delete_async

Prototype

```
int32_t rsi_sntp_client_delete_async(void);
```

Description

This API deletes the SNTP client.

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid parameters, call back not registered -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x002C, 0x0015, 0x0025, 0xFF74, 0xBBF0

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_sntp_client_delete_async();
```

10.10.3 MQTT Client API

10.10.3.1 rsi_mqtt_client_init

Prototype

```
rsi_mqtt_client_info_t * rsi_mqtt_client_init( int8_t *buffer, uint32_t
length, int8_t *server_ip, uint32_t server_port, uint32_t client_port,
uint16_t flags, uint16_t keep_alive_interval)
```

Description

This API initializes the MQTT client structure memory with the linear buffer pointed. This memory is used by MQTT client for further MQTT operations.

Precondition

rsi_config_ipaddress() API needs to be called before this API

Parameters

Parameter	Description
Buffer	This is the linear buffer required to initialize MQTT client structure
Length	This is the length of the linear buffer pointed
server_ip	This is the MQTT broker IP address to connect
server_port	This is the port number of MQTT broker
client_port	This is the port number of MQTT client (local port)

Parameter	Description
Flags	These are the network flags. Each bit in the flag has its own significance BIT(0) – RSI_IPV6 Set this bit to enable IPv6 , By default it is configured to IPv4 BIT(1) to BIT(15) are reserved for future use
keep_alive_interval	This is the MQTT client keep alive interval If there are no transactions between MQTT client and broker within this time period, then MQTT Broker shall disconnects the MQTT client

Return Values

Value	Description
0	Returns MQTT client info structure pointer
Non Zero Value	NULL- Failure

Example

```
//! MQTT client initialisation
rsi_mqtt_client = rsi_mqtt_client_init(mqqt_client_buffer,
MQTT_CLIENT_INIT_BUFF_LEN,(int8_t
*)&server_address,SERVER_PORT,CLIENT_PORT,0,RSI_KEEP_ALIVE_PERIOD);
```

10.10.3.2 rsi_mqtt_connect

Prototype

```
int32_t rsi_mqtt_connect (rsi_mqtt_client_info_t *rsi_mqtt_client,
uint16_t flags, int8_t *client_id,int8_t *username,int8_t *password)
```

Description

This API establishes TCP connection with the given MQTT client port and establishes MQTT protocol level connection.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
rsi_mqtt_client	This is the MQTT client info structure pointer
Flags	To select IP version and security BIT(0) – RSI_IPV6 Set this bit to enable IPv6 , by default it is configured to IPv4 BIT(1) – RSI_SSL_ENABLE Set this bit to enable SSL feature
client_id	This is the clientID string of the MQTT Client and should be unique for each device.
username	This is the username for server Authentication
password	This is the password for server Authentication

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid parameter, expects call back handler **If return value is greater than 0** 0x0021, 0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_mqtt_connect(rsi_mqtt_client,0,clientID,NULL,NULL);
```

10.10.3.3 rsi_mqtt_disconnect

Prototype

```
int32_t rsi_mqtt_disconnect(rsi_mqtt_client_info_t *rsi_mqtt_client)
```

Description

This API disconnects the client from MQTT broker.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
rsi_mqtt_client	This is the MQTT client info structure pointer

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid Parameter **If return value is greater than 0** 0x0021, 0x002C,0x0015

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Disconnect to the MQTT broker
rsi_mqtt_disconnect(rsi_mqtt_client);
```

10.10.3.4 rsi_mqtt_publish

Prototype

```
int32_t rsi_mqtt_publish(rsi_mqtt_client_info_t *rsi_mqtt_client, int8_t *topic, MQTTMessage
*publish_msg)
```

Description

This API publishes the message on the topic specified.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
rsi_mqtt_client	This is the MQTT client info structure pointer

Parameter	Description
Topic	This is the Topic string on which MQTT client wants to publish data
publish_msg	This is the publish message structure

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid Parameter

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_mqtt_publish(rsi_mqtt_client,RSI_MQTT_TOPIC,&publish_msg);
```

10.10.3.5 rsi_mqtt_subscribe

Prototype

```
int32_t rsi_mqtt_subscribe(rsi_mqtt_client_info_t rsi_mqtt_client,uint8_t qos, int8_t *topic,void
(*call_back_handler_ptr)(MessageData md))
```

Description

This API subscribes to the topic specified. Thus, MQTT client will receive any data which is published on this topic further and callback registered will be called.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
rsi_mqtt_client	This is the MQTT client info structure pointer
Qos	This is the quality of service of message at MQTT protocol level. The valid values are 0,1,2
Topic	This the topic string on which MQTT client wants to subscribe
call_back_handler_ptr	This is the callback function when asynchronous data comes on the subscribed data. MessageData* md Message data pointer received

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid Parameter

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Subscribe to the topic given
rsi_mqtt_subscribe(rsi_mqtt_client,QOS,RSI_MQTT_TOPIC,rsi_message_received);
```

10.10.3.6 rsi_mqtt_unsubscribe

Prototype

```
int32_t rsi_mqtt_unsubscribe( rsi_mqtt_client_info_t *rsi_mqtt_client, int8_t *topic)
```

Description

This API unsubscribes to the topic specified. Thus, MQTT client will not receive any data published on this topic further.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
rsi_mqtt_client	This is the MQTT client info structure pointer
topic	This is the topic string on which MQTT client wants to unsubscribe to

Return Values

Value	Description
0	Successful execution of the command
Non-Zero Value	**If return value is lesser than 0** -2: Invalid Parameter

Note:

Please refer to Error Codes section for the description of the above error codes.

Example

//! UnSubscribe to the topic given

```
rsi_mqtt_unsubscribe(rsi_mqtt_client,RSI_MQTT_TOPIC);
```

10.10.3.7 rsi_mqtt_poll_for_recv_data

Prototype

```
int32_t rsi_mqtt_poll_for_recv_data(rsi_mqtt_client_info_t *rsi_mqtt_client, uint16_t time_out)
```

Description

This API waits for the messages to receive on the subscribed topics.

Precondition

rsi_mqtt_subscribe API should be success before this API called.

Parameters

Parameter	Description
rsi_mqtt_client	This is the MQTT client info structure pointer
time_out	This is the time out in milli seconds for which MQTT client has to wait for the messages to receive on the subscribed topic

Return Values

Value	Description
0	Successful execution of the command
Non Zero	-2: Invalid Parameters

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Recv data published on the subscribed topic
status = rsi_mqtt_poll_for_recv_data(rsi_mqtt_client,10);
```

10.10.4 HTTP Server API

10.10.4.1 rsi_webpage_load

Prototype

```
int32_t rsi_webpage_load(uint8_t flags, uint8_t *file_name, uint8_t
*webpage, uint32_t length);
```

Description

This API loads the webpage to the HTTP Server's file system which is present in the WiSeConnect module.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
Flags	BIT (2) is used to set webpage which is associated with json object
file_name	This is the file name of the html webpage.
webpage	This is the pointer to the html webpage which contains the html webpage content
Length	This is the webpage length

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0015,0x0021,0x0025,0x00C1,0x00C2,0x00C3,0x00C5, 0x00C6,0x00C8

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
// "provisioning" is an HTML page stored as an array of uint8_t
status = rsi_webpage_load(FLAGS, FILE_NAME, provisioning, (sizeof(provisioning)-1));
```

If root webpage is to be loaded, clearing the existing root webpage is mandatory.

10.10.4.2 rsi_webpage_send

Prototype

```
int32_t rsi_webpage_send(uint8_t flags, uint8_t *webpage, uint32_t length);
```

Description
This API is used for webpage bypass.

Precondition
rsi_wlan_connect() API needs to be called before this API.

Parameters

Parameter	Description
Flags	This is used to set webpage
webpage	This is the pointer to the html webpage which contains the html webpage content
Length	This is the webpage length

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0015,0x0021,0x0025,0x00C1,0x00C2,0x00C3,0x00C5, 0x00C6,0x00C8

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
// "provisioning" is an HTML page stored as an array of uint8_t
int32_t rsi_webpage_send(flags, provisioning, length);
```

10.10.4.3 rsi_json_object_create

Prototype

```
int32_t rsi_json_object_create(uint8_t flags, uint8_t *file_name, uint8_t *json_object, uint32_t
length);
```

Description
This API creates the json object to the webpage which is already present in the WiSeConnect module's HTTP server file system.

Precondition
rsi_wireless_init() API needs to be called before this API

Parameters

Parameter	Description
flags	Rserved
file_name	This is the file name of the json object data
json_object	This is the pointer to the json object data
length	This is the length of the json object data

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0015, 0x0021,0x0025,0x002C,0x00B1,0x00B2,0x00B3,0x00B4, 0x00B5,0x00B6.

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
// associate json_object_str with the page FILE_NAME
// future requests to FILE_NAME will have the json object appended to the HTML as a script.
status = rsi_json_object_create(0, FILE_NAME, json_object_str, strlen(json_object_str));
```

10.10.4.4 rsi_webpage_erase

Prototype

```
int32_t rsi_webpage_erase(uint8_t *file_name);
```

Description

This API erases the webpage from HTTP server's file system which is present in the WiSeConnect module.

Precondition

rsi_wireless_init() API needs to be called before this API

Parameters

Parameter	Description
file_name	To erase particular/All loaded webpage files from the HTTP server's file system file_name: To erase the particular webpage file NULL: To erase all loaded webpage files

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025,0x002C, 0x00C4

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
rsi_webpage_erase(FILE_NAME);
```

10.10.4.5 rsi_json_object_delete

Prototype

```
int32_t rsi_json_object_delete(uint8_t *file_name);
```

Description
This API deletes the json object of the HTTP server's file system which is already present in the WiSeConnect module.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
file_name	To delete the particular json object which is already created in the HTTP server's file system

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002C,0x00B4

Note: Please refer to Error Codes section for the description of the above error codes.

10.10.5 MDNSD API

10.10.5.1 rsi_mdnsd_init

Prototype

```
int32_t rsi_mdnsd_init(uint8_t ip_version, uint16_t ttl, uint8_t *host_name);
```

Description
This API initializes the MDNSD service in WiSeConnect Device. It creates MDNS daemon.

Note:

1. Currently registering only one service is supported.
2. IPv4 is only supported for MDNS/DNS-SD service.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
ip_version	To select the IP version. 4 – To select IPv4 6 – To select IPv6
ttl	This is the time to live. THis is the time in seconds for which service should be active
host_name	This is the host name which is used as host name in Type A record.

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	**If return value is lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0015, 0x0074

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Initialize MDNSD service
status = rsi_mdnsd_init((uint8_t)MDNSD_IP_VERSION, (uint16_t)MDNSD_INIT_TTL, (uint8_t
*)MDNSD_HOST_NAME);
```

10.10.5.2 rsi_mdnsd_register_service

Prototype

```
int32_t rsi_mdnsd_register_service(uint16_t port,
uint16_t ttl,
uint8_t more,
uint8_t *service_ptr_name,
uint8_t *service_name,
uint8_t *service_text);
```

Description

This API is used to add a service / start service discovery.

Note: Currently registering only one service is supported.

Precondition

rsi_config_ipaddress() API needs to be called before this API

Parameters

Parameter	Description
port	This is the port number on which service should be added.
ttl	This is the time to live. This is the time in seconds for which service should be active
more	This byte should be set to '1' when there are more services to add. 0 – This is last service, starts MDNS service. 1 – Still more services will be added.
service_ptr_name	This is the name to be added in Type-PTR record
service_name	This is the name to be added in Type-SRV record(Service name)
service_text	This is the text field to be added in Type-TXT record

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is lesser than 0** -4: Buffer not available to serve the command -6: Data size exceeded **If return value is greater than 0** 0x0021, 0x0015, 0x0074

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
//! Add required services
status = rsi_mdnsd_register_service(MDNSD_SERVICE_PORT,
MDNSD_SERVICE_TTL, MDNSD_SERVICE_MORE, MDNSD_POINTER_NAME,
MDNSD_SERVICE_NAME, MDNSD_SERVICE_TEXT);
```

10.10.5.3 rsi_mdnsd_deinit

Prototype

```
int32_t rsi_mdnsd_deinit(void);
```

Description
This API deletes the mdnsd service.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0015, 0x0074,0xFF2B

Note: Please refer to Error Codes section for the description of the above error codes.

10.10.6 Socket configuration API

10.10.6.1 rsi_socket_config

Prototype

```
int32_t rsi_socket_config()
```

Description
This command is used to set the socket configuration parameters. User is recommended to use this command(optional). Based on the socket configuration, module will use available buffers effectively. This command should be given after IP configuration command and before any socket creation.

Parameters

Parameter	Description
total socket	Desired total number of sockets to open.
total_tcp_sockets	Desired total number of TCP sockets to open
total_udp_sockets	Desired total number of UDP sockets to open.
tcp_tx_only_sockets	Desired total number of TCP sockets to open which are used only for data transmission.

Parameter	Description
tcp_rx_only_sockets	Desired total number of TCP sockets to open which are used only for data reception.
udp_tx_only_sockets	Desired total number of UDP sockets to open which are used only for data transmission.
udp_rx_only_sockets	Desired total number of UDP sockets to open which are used only for data reception.
tcp_rx_high_performance_sockets	Desired total number of high performance TCP sockets to open. High performance sockets can be allocated with more buffers based on the buffers availability. This option is valid only for TCP data receive sockets. Socket can be opened as high performance by setting high performance bit in socket create command.
tcp_rx_window_size_cap	Desired to increase the tcp rx window size
tcp_ack_window_div_factor	In case of high latency networks to configure the TCP ACK division factor with respective to the window size **Note:** Default value is 2. Possible values: 2 - tcp_rx_window_size_cap

Following conditions has to be met:

1. total_sockets <= Maximum allowed sockets(10)
2. (total_tcp_sockets + total_udp_sockets) <= total_sockets
3. (total_tcp_tx_only_sockets + total_tcp_rx_only_sockets) <= total_tcp_sockets
4. (total_udp_tx_only_sockets + total_udp_rx_only_sockets) <= total_udp_sockets
5. total_tcp_rx_high_performance_sockets <= total_tcp_rx_only_sockets

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is greater than 0** 0x0021,0x0025,0x002C,0xFF6D.

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_socket_config();
```

10.10.7 rsi_wlan_req_radio

Prototype

```
int32_t rsi_wlan_req_radio(uint8_t enable)
```

Description

This function is used to register and de-register wlan radio.

Parameters

Value	Description
Enable	To register or de-register wlan radio

Return values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_wlan_req_radio(uint8_t enable)
```

10.10.8 rsi_wlan_add_mfi_ie

Prototype

```
int32_t rsi_wlan_add_mfi_ie(int8_t *mfi_ie, uint32_t ie_len);
```

Description

This function will add the Apple defined IE elements to the in Beacon command request structure.

Parameters

Value	Description
mfie_ie	pointer to the IE element
ie_len	length of the IE element

Return Values

Value	Description
0	Successful execution of the command
<0	-4: Buffer not available to serve the command

Example

```
status = rsi_wlan_add_mfi_ie(mfi_ie,ie_len);
```

10.10.9 Web socket API

10.10.9.1 rsi_web_socket_create

Prototype

```
int32_t rsi_web_socket_create(int8_t flags,
uint8_t *server_ip_addr,
uint16_t server_port,
uint16_t device_port,
uint8_t *webs_resource_name,
uint8_t *webs_host_name,
int32_t *socket_id,
void (*web_socket_data_receive_notify_callback)
(uint32_t sock_no,
uint8_t *buffer,
uint32_t length));
```

Description

This API creates a web socket client .

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
flags	To select IP version and security BIT(0) – RSI_IPV6 Set this bit to enable IPv6 , by default it is configured to IPv4 BIT(1) – RSI_SSL_ENABLE Set this bit to enable SSL feature
server_ip_addr	This is the web server ip address
server_port	This is the web server socket port.
device_port	This is the local port
webs_resource_name	This is the web resource name **Note:** string of 50 characters maximum.
webs_host_name	This is the web host name **Note:** string of 50 characters maximum.
web_socket_data_receive_notify_callback)	This is the callback when data packet is received on the created socket. The parameters involved are: sock_no , buffer, length and more_data **sock_no:** This is the Application socket ID **buffer:** This is the buffer pointer **length:** This is the length of data

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is lesser than 0** -2: Invalid parameter -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0015, 0x0074

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
//! create web socket client and connect to server
status = rsi_web_socket_create(flags,(uint8_t *)&server_ip_addr,(uint16_t)SERVER_PORT,
(uint16_t)DEVICE_PORT,
WEB_SOCKET_RESOURCE_NAME, WEB_SOCKET_HOST_NAME, &sockID,
rsi_websocket_data_receive_handler);
```

10.10.9.2 rsi_web_socket_send_async

Prototype

```
int32_t rsi_web_socket_send_async(uint32_t sockID,
uint8_t opcode,
```

```
int8_t *msg,
int32_t msg_length);
```

Description
This API sends data from the web socket client.

Precondition
rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
sockID	Application socket ID
Opcode	opcode (type of the packet to be included in web socket header). OPCODE should be as follows(Refer RFC 6455): 0 - Continuation frame 1 – Text frame 2 – Binary frame [3-7] – Reserved for further non-control frames 8 - Connection close frame 9 - Ping frame 10 - Pong frame [B-F] - Reserved for further control frames FIN Bit should be as follows: 0: More web socket frames to be followed. 1: Final frame web socket message.
Msg	data
msg_length	Data length

Return Values

Value	Description
0	Successful execution of the command
Non Zero	-1: Failure

Example

```
status = rsi_web_socket_send_async(sockID, opcode, MESSAGE, strlen((const char *)MESSAGE));
```

10.10.9.3 rsi_web_socket_close

Prototype

```
int32_t rsi_web_socket_close(int32_t sockID);
```

Description
This API closes the web socket client.

Precondition
rsi_config_ipaddress() API needs to be called before this API

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	-1: Failure

Example

```
//! close client websocket
status = rsi_web_socket_close(sockID);
```

10.10.10 OTAF client API

10.10.10.1 rsi_ota_firmware_upgradation

Prototype

```
int32_t rsi_ota_firmware_upgradation(uint8_t flags,
uint8_t *server_ip,
uint32_t server_port,
uint16_t chunk_number,
uint16_t timeout,
uint16_t tcp_retry_count,
void(*ota_fw_up_response_handler)(uint16_t
status, uint16_t chunk_number));
```

Description

This API creates an otaf client.This initializes the client with given configuration.

Precondition

rsi_config_ipaddress() API needs to be called before this API.

Parameters

Parameter	Description
flags	To select IPv6 version, a bit in flags is set. By default IP version is set to IPV4. RSI_IPV6 – BIT(0) To select IPv6 version
server_ip	This is the OTAF server IP address
server_port	This is the OTAF server port number
chunk_number	This is the firmware content request chunk number
timeout	This is the TCP receive packet timeout
tcp_retry_count	This is the TCP retransmissions count
ota_fw_up_response_handler	This is the callback when asynchronous response comes for the firmware upgrade request. The parameters involved are : status and chunk_number **status:** This is the status code **chunk_number:** This is the chunk number of the firmware content

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command

Note: Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_ota_firmware_upgradation(RSI_IP_VERSION_4, (uint8_t
*)&otaf_server_addr, (uint32_t *)OTAF_SERVER_PORT, (uint16_t
*)chunk_number, (uint16_t *)OTAF_RX_TIMEOUT, (uint16_t
*)OTAF_TCP_RETRY_COUNT, rsi_ota_fw_up_response_handler);
 if(status != RSI_SUCCESS)
 {
```

```
return status;
}
```

Note: It is recommended to finish pending transactions (like data transfers and disconnect sockets if any) and disable power save before starting firmware upgrade process.

11 BT/BLE Common APIs

This section explains the BT/BLE Common APIs to initialize and configure the module.

11.1 rsi_bt_set_local_name

Prototype

```
int32_t rsi_bt_set_local_name(int8_t *local_name);
```

Description

This API sets the given name to local device.

Precondition

rsi_wireless_init() API need to be called before this API.

Parameters

Parameter	Description
local_name	This is the name to be set to the local device

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define RSI_BT_LOCAL_NAME "BT_SAMPLE_APP"
int32_t status = 0;
status = rsi_bt_set_local_name(RSI_BT_LOCAL_NAME);
```

11.2 rsi_bt_get_local_name

Prototype

```
int32_t rsi_bt_get_local_name (rsi_bt_resp_get_local_name_t *bt_resp_get_local_name);
```

Description

This API requests the local device name.

Precondition

rsi_wireless_init() API need to be called before this API.

Parameters

Parameter	Description
bt_resp_get_local_name	This parameter is the response buffer to hold the response of this API. This is a structure variable of rsi_bt_resp_get_local_name_s

rsi_bt_resp_get_local_name_t

Prototype

```
#define RSI_DEV_NAME_LEN  50
typedef struct rsi_bt_get_local_name_s
{
   uint8_t name_len;
   int8_t name[RSI_DEV_NAME_LEN];
} rsi_bt_resp_get_local_name_t;
```

Description

This structure describes the format of the get local name response structure.

Variables

Variables	Description
name_len	This is the name length
name	This is an array which consists name of the local device. The maximum size of this array is 50

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
rsi_bt_resp_get_local_name_t local_name = {0};
status = rsi_bt_get_local_name(&local_name);
```

11.3 rsi_bt_get_rssi

Prototype

```
int32_t rsi_bt_get_rssi(int8_t *dev_addr, uint8_t *resp);
```

Description

This API requests the RSSI of the remote device.

Precondition

rsi_bt_connect() API need to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
resp	This parameter is to hold the response of this API

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
uint8_t rsi_app_resp_rssi = 0;
status = rsi_bt_get_rssi(remote_dev_addr, &rsi_app_resp_rssi);
```

11.4 rsi_bt_get_local_device_address

Prototype

```
int32_t rsi_bt_get_local_device_address(uint8_t *resp);
```

Description

This API requests the local device address.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
resp	This parameter is to hold the response of this API

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
uint8_t rsi_app_resp_get_dev_addr[RSI_DEV_ADDR_LEN] = {0};
status = rsi_bt_get_local_device_address(rsi_app_resp_get_dev_addr);
```

11.5 rsi_bt_init

Prototype

```
int32_t rsi_bt_init(void);
```

Description

This API initializes the BT device.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_init();
```

11.6 rsi_bt_deinit

Prototype

```
int32_t rsi_bt_deinit(void);
```

Description

This API deinitializes the BT device.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_deinit();
```

11.7 rsi_bt_set_antenna

Prototype

```
int32_t rsi_bt_set_antenna(uint8_t antenna_value);
```

Description

This API selects either internal / external antenna on the chip.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
antenna_value	This parameter is used to select either internal or external antenna Possible values : **Value** / **Description** 0x00 / RSI_SEL_INTERNAL_ANTENNA 0x01 / RSI_SEL_EXTERNAL_ANTENNA

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	Failure

Example

```
#define RSI_SEL_ANTENNA RSI_SEL_INTERNAL_ANTENNA
int32_t status = 0;
//! select/set the antenna type(internal/external)
status = rsi_bt_set_antenna(RSI_SEL_ANTENNA);
```

11.8 rsi_bt_set_feature_bitmap

Prototype

```
int32_t rsi_bt_set_feature_bitmap(uint32_t feature_bit_map);
```

Description

This API enables /disables the mentioned features.

Precondition

rsi_sspmode_init() API needs to be called before this API.

Parameters

Parameter	Description	
feature_bit_map	**Bits**	**Description**
	0	This parameter is used for security purposes. If this bit is set pairing process occurs, else does not occur.
	1 to 31	Reserved for future use.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_set_feature_bitmap(1);
```

11.9 rsi_bt_set_antenna_tx_power_level

Prototype

```
int32_t rsi_bt_set_antenna_tx_power_level(uint8_t protocol_mode, int8_t tx_power);
```

Description

This API enables / disables the mentioned features.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
protocol_mode	1 – BT classic 2 – BT Low Energy
tx_power	Antenna transmit power level Default tx power index for BT low energy is 30 Default tx power index for BT classic is 14 **Note:** The default value will vary based on country region and board.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_set_antenna_tx_power_level(1,19);
```

11.10 rsi_bt_power_save_profile

Prototype

```
int32_t rsi_bt_power_save_profile(uint8_t psp_mode, uint8_t psp_type);
```

Description

This API selects the power save profile mode for BT / BLE.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
psp_mode	Following psp_mode is defined. RSI_ACTIVE (0): In this mode module is active and power save is disabled. RSI_SLEEP_MODE_1 (1): This is on mode. In this sleep mode, SoC will never turn off, therefore no handshake is required before sending data to the module. BT/BLE does not support this mode. RSI_SLEEP_MODE_2 (2): This is connected sleep mode. In this sleep mode, SoC will go to sleep based on GPIO or Message, therefore handshake is required before sending data to the module. RSI_SLEEP_MODE_8 (8): This is disconnected sleep mode. In this sleep mode, module will turn off the SoC. Since SoC is turn off, therefore handshake is required before sending data to the module.
psp_type	Following psp_type is defined. RSI_MAX_PSP (0): This psp_type will be used for max power saving. BT/BLE supports only RSI_MAX_PSP mode. Remaining modes are not supported.

Note:
psp_type is only valid in psp_mode 2.
BT/BLE doesnot support in RSI_SLEEP_MODE_1.

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	Failure

Example

```
//! initiating power save in BT mode
status = rsi_bt_power_save_profile(RSI_SLEEP_MODE_2, RSI_MAX_PSP);
```

12 Bluetooth Classic APIs

This section explains the BT APIs required to initialize and configure the module in BT mode.

Note:

A new BT API must be called only after getting the response for the previous API.

12.1 Test Mode (RF Regulatory Mode) API

This section describes the APIs that are being used for testing purposes.

12.1.1 rsi_bt_enable_device_under_testmode

Prototype

```
int32_t rsi_bt_enable_device_under_testmode(void);
```

Description

This API is used to keep the device in the LMP_TEST_MODE.

Precondition

rsi_wireless_init() API need to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_enable_device_under_testmode();
```

12.1.2 rsi_bt_per_rx

Prototype

```
int32_t rsi_bt_per_rx(uint32_t *bt_rx_per);
```

Description

This API is used to configure the PER receive parameters in the controller and start/stop the PER.

Precondition

rsi_wireless_init() API need to be called before this API.

Parameters

Parameter	Description
bt_rx_per	This parameter is the buffer to hold the structure values This is a structure variable of rsi_bt_rx_per_params_t.

rsi_bt_rx_per_params_t

Prototype

```
typedef struct rsi_bt_rx_per_params_s {
   uint8_t cmd_id;
   uint8_t receive_enable;
   uint8_t device_Addr[6];
   uint8_t pkt_len[2];
   uint8_t pkt_type;
   uint8_t br_edr_mode;
   uint8_t rx_chnl_in;
   uint8_t tx_chnl_in;
   uint8_t link_type;
   uint8_t scrambler_seed;
   uint8_t hopping_type;
   uint8_t ant_sel;
   uint8_t pll_mode;
   uint8_t rf_type;
   uint8_t rf_chain;
   uint8_t loop_back_mode;
} rsi_bt_rx_per_params_t;
```

Description

This structure describes the format for accepting bt per rx parameters to be sent to local device.

Variables

Variables	Description
cmd_id	This parameter takes per BT_RECEIVE_CMD_ID of value 0x16
receive_enable	This parameter enables/disables the bt per receive mode 1 → PER Receive Enable 0 → PER Receive Disable
device_addr	This is the device BD address from which packets to be received eg: 000012345678 BD addr will be used while Testing with PKT generator
pkt_len	length of the packet to be received
pkt_type	This field corresponds to packet types proposed by BT_SIG. Value takes from 0 to 15
br_edr_mode	This field corresponds to BR/EDR Mode 1 → BR_MODE 2 → EDR_MODE
rx_chnl_in	This field corresponds to rx channel number to be used for receive (0 to 78)
tx_chnl_in	This field corresponds to tx channel number to be used (0 to 78)
link_type	This field corresponds to link type to be setup for receiving per packets 0 → SCO_LINK 1 → ACL_LINK 2 → ESCO_LINK
scrambler_seed	Initial seed to be used for whitening. It should be set to '0' in order to disable whitening. In order to enable, one should give the scrambler seed value which is used on the transmit side
hopping_type	This field defines the frequency hopping type to be used 0 → NO_HOPPING 1 → FIXED_HOPPING 2 → RANDOM_HOPPING (rx_chnl_in, tx_chnl_in parameters are unused in this mode)
ant_sel	This field defines the antenna selection (onboard/external) to be used for reception

Variables	Description
	2 → ONBOARD_ANT_SEL 3 → EXT_ANT_SEL
pll_mode	This field define the pll_mode type to be used 0 → PLL_MODE0 (to be used by Default) 1 → PLL_MODE1
rf_type	This field defines the selection of RF type (internal/external) 0 → BT_EXTERNAL_RF 1 → BT_INTERNAL_RF (to be used by Default)
rf_chain	This field corresponds to the selection of RF chain (HP/LP) to be used 2 → BT_HP_CHAIN 3 → BT_LP_CHAIN (Only for BR_MODE)
loop_back_mode	This field defines the loopback to be enable or disable 0 → LOOP_BACK_MODE_DISABLE 1 → LOOP_BACK_MODE_ENABLE

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
rsi_bt_rx_per_params_t bt_rx_per;
int32_t status = 0;

/* Example rx_per parameters */
bt_rx_per.cmd_id = BT_RECEIVE_CMD_ID;
bt_rx_per.receive_enable=ENABLE;
/*................*/

status = rsi_bt_per_rx(&bt_rx_per);
```

12.1.3 rsi_bt_per_tx

Prototype

```
int32_t rsi_bt_per_tx(uint32_t *bt_tx_per);
```

Description

This API is used to configure the PER transmit parameters in the controller and start/stop the PER.

Precondition

rsi_wireless_init() API need to be called before this API.

Parameters

Parameter	Description
bt_tx_per	This parameter is the buffer to hold the structure values This is a structure variable of rsi_bt_tx_per_params_t.

rsi_bt_tx_per_params_t

Prototype

```
typedef struct rsi_bt_tx_per_params_s {
   uint8_t cmd_id;
   uint8_t transmit_enable;
   uint8_t device_Addr[6];
   uint8_t pkt_len[2];
   uint8_t pkt_type;
   uint8_t br_edr_mode;
   uint8_t rx_chnl_in;
   uint8_t tx_chnl_in;
   uint8_t link_type;
   uint8_t scrambler_seed;
   uint8_t hopping_type;
   uint8_t ant_sel;
   uint8_t pll_mode;
   uint8_t rf_type;
   uint8_t rf_chain;
   uint8_t payload_type;
   uint8_t tx_power;
   uint8_t transmit_mode;
   uint8_t inter_pkt_gap;
   uint8_t no_of_packets[4];
} rsi_bt_tx_per_params_t;
```

Description

This structure describes the format for accepting bt per rx parameters to be sent to local device.

Variables

Variables	Description
cmd_id	This parameter takes per BT_TRANSMIT_CMD_ID of value 0x15
transmit_enable	This parameter enables/disables the bt per transmit mode 1 → PER Transmit Enable 0 → PER Transmit Disable
device_addr	This is the device BD address to which packets are to be transmitted. This variable is unused while Testing with analyzer
pkt_len	length of the packet to be transmitted
pkt_type	This field corresponds to packet types proposed by BT_SIG. Takes the values from 0 to 15
br_edr_mode	This field corresponds to BR/EDR Mode 1 → BR_MODE 2 → EDR_MODE
rx_chnl_in	This field corresponds to rx channel number to be used (0 to 78)
tx_chnl_in	This field corresponds to tx channel number to be used (0 to 78)
link_type	This field corresponds to link type to be setup for transmitting per packets 0 → SCO_LINK 1 → ACL_LINK 2 → ESCO_LINK
scrambler_seed	Initial seed to be used for whitening. It should be set to '0' in order to disable whitening. In order to enable, one should give the scrambler seed value which is used on the receive side
hopping_type	This field defines the frequency hopping type to be used 0 → NO_HOPPING 1 → FIXED_HOPPING 2 → RANDOM_HOPPING (rx_chnl_in, tx_chnl_in parameters are unused in this mode)

Variables	Description
ant_sel	This field defines the antenna selection (onboard/external) to be used for transmission 2 → ONBOARD_ANT_SEL 3 → EXT_ANT_SEL
pll_mode	This field defines the pll_mode type to be used 0 → PLL_MODE0 (to be used by Default) 1 → PLL_MODE1
rf_type	This field defines the selection of RF type (internal/external) 0 → BT_EXTERNAL_RF 1 → BT_INTERNAL_RF (to be used by Default)
rf_chain	This field corresponds to the selection of RF chain (HP/LP) to be used 2 → BT_HP_CHAIN 3 → BT_LP_CHAIN (Only for BR_MODE)
payload_type	This field corresponds to the payload sequence of data to be transmitted. The default payload type is '4'. 0 → SEQUENCE_0 1 → SEQUENCE_1 2 → SEQUENCE_2 3 → SEQUENCE_F0 4 → SEQUENCE_PRBS
tx_power	This field corresponds to the transmit power. Range is different based on rf_chain BT_LP_CHAIN - 1 - 31 power index - 0 dBm Mode 33 - 63 power index - 8 dBm Mode Following are the equations to be used for deriving power output in dBm (0 - 31) o/p power equation is -2 + 10log10(power_index/31) (32-63) o/p power equation is -2 + 8+10log10((power_index -32)/31) BT_HP_CHAIN - 1 to 22 dBm (based on Country Regulations and Chip capability Max dBm varies. Configure 127 to select Max permitted value)
transmit_mode	This field corresponds to the transmit mode to be used either Burst/Continuous 0 → BURST_MODE (BT stats are observed only in this mode) 1 → CONTINUOUS_MODE (no_of_packets variable is unused when this mode is selected)
inter_pkt_gap	This field takes the value of inter packet gap. Number of slots to be skipped between two packets. Each slot will be 625usec
no_of_packets	This field defines the number of packets to be transmitted, default to zero for continuous transmission

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	Failure

Example

```
rsi_bt_tx_per_params_t bt_tx_per;
int32_t status = 0;

/* Example tx_per parameters */
```

```
bt_tx_per.cmd_id = BT_TRANSMIT_CMD_ID;
bt_tx_per.transmit_enable=ENABLE;
/*................*/

status = rsi_bt_per_tx(&bt_tx_per);
```

12.1.4 rsi_bt_per_stats

Prototype

```
int32_t rsi_bt_per_stats(uint8_t cmd_type, rsi_bt_per_stats_t *per_stats);
```

Description

This API requests the local device for BT PER operation.

Precondition

rsi_wireless_init() API need to be called before this API.

Parameters

Parameter	Description
cmd_type	This parameter defines the command id type for the PER operation BT_PER_STATS_CMD_ID (0x08) - This command id enables PER statistics BT_TRANSMIT_CMD_ID (0x15) - This command id enables PER transmit BT_RECEIVE_CMD_ID (0x16) - This command id enables PER receive
per_stats	This parameter is the response buffer to hold the response of this API. This is a structure variable of rsi_bt_per_stats_t

rsi_bt_per_stats_t

Prototype

```
typedef struct rsi_bt_per_stats_s
{
  uint16_t crc_fail_cnt;
  uint16_t crc_pass_cnt;
  uint16_t tx_abort_cnt;
  uint16_t rx_drop_cnt;
  uint16_t rx_cca_idle_cnt;
  uint16_t rx_start_idle_cnt;
  uint16_t rx_abrt_cnt;
  uint16_t tx_dones;
  uint16_t rssi;
  uint16_t id_pkts_rcvd;
  uint16_t dummy[5];
} rsi_bt_per_stats_t
```

Description

This structure describes the format of PER statistics.

Variables

Variables	Description
crc_fail_cnt	Packet count of CRC fails (Cyclic Redundancy Check (CRC))
crc_pass_cnt	Packet count of CRC pass
tx_abort_cnt	Packet count of aborted Tx
rx_drop_cnt	Packet count of dropped Rx

Variables	Description
rx_cca_idle_cnt	Packet count of CCA Idle (Clear Channel Assessment (CCA))
rx_start_idle_cnt	Packet count of Rx start
rx_abrt_cnt	Packet count of aborted Rx
tx_dones	Packet count of successful transmissions
rssi	Received Signal Strength Indicator of the packet
id_pkts_rcvd	Packet count of ID packets received
dummy	Dummy array of length 5

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
rsi_bt_per_stats_t per_stats = {0};
status = rsi_bt_per_stats(BT_PER_STATS_CMD_ID,  &per_stats);
```

12.2 GAP API

This section explains the BT GAP APIs.

12.2.1 rsi_bt_set_local_class_of_device

Prototype

```
int32_t rsi_bt_set_local_class_of_device(uint32_t class_of_device);
```

Description

This API requests the local Class Of Device (COD) name.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
class_of_device	This is the class of device

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
uint32_t class_of_device = 284000;
int32_t status = 0;
status = rsi_bt_set_local_class_of_device(class_of_device);
```

12.2.2 rsi_bt_get_local_class_of_device

Prototype

```
int32_t rsi_bt_get_local_class_of_device(uint8_t *resp);
```

Description

This API requests the local COD name.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
resp	This parameter is to hold the response of this API

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_get_local_class_of_device(&class_of_device);
```

12.2.3 rsi_bt_start_discoverable

Prototype

```
int32_t rsi_bt_start_discoverable(void);
```

Description

This API requests the local device to enter discovery mode.

Precondition

rsi_wireless_init API needs to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_start_discoverable();
```

12.2.4 rsi_bt_start_limited_discoverable

Prototype

```
int32_t rsi_bt_start_limited_discoverable(int32_t time_out_ms);
```

Description

This API requests the local device to enter limited discovery mode. The device comes out of discovery mode after the time out.

If the 'time_out_ms' is set to '0', the device will be in continuous discoverable mode.

Precondition

rsi_wireless_init API needs to be called before this API.

Parameters

Parameter	Description
time_out_ms	Limited discovery mode time_out in ms.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_start_limited_discoverable(100);
```

12.2.5 rsi_bt_stop_discoverable

Prototype

```
int32_t rsi_bt_stop_discoverable(void);
```

Description

This API request the local device to exit the discovery mode.

Parameters

None

Precondition

rsi_wireless_init API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_stop_discoverable();
```

12.2.6 rsi_bt_get_discoverable_status

Prototype

```
int32_t rsi_bt_get_discoverable_status(uint8_t *resp)
```

Description

This API is used to request the local device discovery mode status.

Precondition

rsi_bt_start_discoverable() API need to be called before this API.

Parameters

Parameter	Description
Resp	This parameter is to hold the response of this API

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
uint8_t device_state = 0;
int32_t status = 0;
status = rsi_bt_get_discoverable_status(&device_state);
```

12.2.7 rsi_bt_set_connectable

Prototype

```
int32_t rsi_bt_set_connectable(void);
```

Description

This API is used to set the BT Module in connectable mode.

Precondition

rsi_wireless_init API needs to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_set_connectable();
```

12.2.8 rsi_bt_set_non_connectable

Prototype

```
int32_t rsi_bt_set_non_connectable(void);
```

Description

This API sets the BT Module in non-connectable mode.

Precondition

rsi_wireless_init API need to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_set_non_connectable();
```

12.2.9 rsi_bt_get_connectable_status

Prototype

```
int32_t rsi_bt_get_connectable_status(uint8_t *resp);
```

Description

This API gets the BT Module connectivity status.

Precondition

rsi_bt_set_connectable() API needs to be called before this API

Parameters

Parameter	Description
Resp	This parameter is to hold the response of this API

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
uint8_t device_state = 0;
int32_t status = 0;
status = rsi_bt_get_connectable_status(&device_state);
```

12.2.10 rsi_bt_set_afh_host_channel_classification

Prototype

```
int32_t rsi_bt_set_afh_host_channel_classification (uint8_t enable, uint8_t *channel_map)
```

Description

This API is used to set the number of channels to be used.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
Enable	This parameter is used to enable or disable afh host channel classification. This parameter supports the following values. 0 - Disable 1 - Enable
channel map	Array of channel mapping values.

Parameter	Description
	Channel map range: This parameter contains 80 1-bit fields. The nth such field (in the range 0 to 78 bits) contains the value for channel n: 0: channel n is bad 1: channel n is unknown

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
uint8_t channel_map[CHANNEL_MAP_LEN] = {0xff, 0xff, 0xff, 0x00, 0xff, 0x00, 0x00, 0xff, 0x00, 0x0f};
int32_t status = 0;
status = rsi_bt_set_afh_host_channel_classification(1, &channel_map);
```

12.2.11 rsi_bt_get_afh_host_channel_classification

Prototype

```
int32_t rsi_bt_get_afh_host_channel_classification (uint8_t *mode)
```

Description

This API is used to get the afh channel assessment mode.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
Mode	This parameter is to hold the response of this API

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
uint8_t mode = 0;
status = rsi_bt_get_afh_host_channel_classification(&mode);
```

12.2.12 rsi_bt_remote_name_request_async

Prototype

```
int32_t rsi_bt_remote_name_request_async(int8_t *remote_dev_addr, rsi_bt_event_remote_device_name_t
*bt_event_remote_device_name);
```

Description

This API is used to know the remote device name.

Precondition

rsi_wireless_init() API need to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address
bt_event_remote_device_name	This parameter is a response buffer to hold the name of the remote device. This is a structure variable of rsi_bt_event_remote_device_name_s

rsi_bt_event_remote_device_name_t

Prototype

```
typedef struct rsi_bt_event_remote_device_name_s
{
   uint8_t dev_addr[RSI_DEV_ADDR_LEN];
   uint8_t name_length;
   uint8_t remote_device_name[RSI_DEV_NAME_LEN];
} rsi_bt_event_remote_device_name_t;
```

Description

This structure describes the format of the remote device name event structure.

Variables

Variables	Description
dev_addr	This is the BD address of remote device. This is an array of max length 6 bytes.
name_length	This is the length of remote device name.
remote_device_name	This is the name of remote device. This is an array of max length 50 bytes

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
rsi_bt_event_remote_device_name_t remote_device_name = {0};
int32_t status = 0;
status = rsi_bt_remote_name_request_async(remote_dev_addr , &remote_dev_name)
```

12.2.13 rsi_bt_remote_name_request_cancel

Prototype

```
int32_t rsi_bt_remote_name_request_cancel(int8_t *remote_dev_addr);
```

Description

This API is used to cancel the remote device name request.

Precondition

rsi_bt_remote_name_request_async() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_remote_name_request_cancel(remote_dev_addr);
```

12.2.14 rsi_bt_inquiry

Prototype

```
int32_t rsi_bt_inquiry(uint8_t inquiry_type, uint32_t inquiry_duration, uint8_t max_devices);
```

Description

This API starts the inquiry.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
inquiry_type	This is the Inquiry type. InquiryType – 0 -Standard Inquiry 1 - Inquiry with RSSI 2 - Extended Inquiry
inquiry_duration	This is the duration of inquiry
max_devices	This is the maximum number of devices allowed to inquiry from 1 to 10

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_inquiry(0, 500, 0x02);
```

12.2.15 rsi_bt_cancel_inquiry

Prototype

```
int32_t rsi_bt_cancel_inquiry(void);
```

Description

This API cancels the inquiry.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_cancel_inquiry();
```

12.2.16 rsi_bt_set_eir_data

Prototype

```
int32_t rsi_bt_set_eir_data(int8_t *eir_data, uint16_t data_len);
```

Description

This API sets the extended Inquiry Response data.

Precondition

rsi_wireless_init() API needs to be called before this API

Parameters

Parameter	Description
eir_data	Pointer to the EIR data buffer which is an array that can store data upto 200 Bytes.
data_len	This is the length of the eir data

Note:
Currently EIR data supports upto 200 bytes.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define RSI_BT_LOCAL_NAME "SPP_SLAVE"
int32_t status = 0;
uint8_t eir_data[200] = {2,1,0};
//! prepare Extended Response Data
eir_data[3] = strlen (RSI_BT_LOCAL_NAME) + 1;
eir_data[4] = 9;
strcpy (&eir_data[5], RSI_BT_LOCAL_NAME);
//! set eir data
status = rsi_bt_set_eir_data (eir_data, strlen (RSI_BT_LOCAL_NAME) + 5);
```

12.2.17 rsi_bt_connect

Prototype

```
int32_t rsi_bt_connect(int8_t *remote_dev_addr);
```

Description

This API initiates the connection request.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define REMOTE_BD_ADDR "00:1B:DC:07:2C:F0"
int32_t status = 0;
//! start the discover mode
status = rsi_bt_start_discoverable();
//! start the connectability mode
status = rsi_bt_set_connectable();
status = rsi_bt_connect(REMOTE_BD_ADDR);
```

12.2.18 rsi_bt_cancel_connect

Prototype

```
int32_t rsi_bt_cancel_connect(int8_t *remote_dev_address);
```

Description

This API cancels the connection request.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
remote_dev_address	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_cancel_connect(remote_dev_addr);
```

12.2.19 rsi_bt_disconnect

Prototype

```
int32_t rsi_bt_disconnect(int8_t *remote_dev_address);
```

Description

This API is used to disconnect the physical connection.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
remote_dev_address	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_disconnect(remote_dev_addr);
```

12.2.20 rsi_bt_set_ssp_mode

Prototype

```
int32_t rsi_bt_set_ssp_mode (uint8_t pair_mode, uint8_t IOcapability);
```

Description

This API enables / disables Simple Secure Profile (SSP) mode.

Precondition

rsi_bt_set_connectable() API needs to be called before this API.

Parameters

Parameter	Description
pair_mode	This parameter is used to enable or disable SSP mode. This parameters supports the following values. 0 - Disable 1 – Enable
IOcapability	This is the IO capability request for SSP mode. This parameter supports the following values. 0 – DisplayOnly 1 – DisplayYesNo 2 – KeyboardOnly 3 – NoInputNoOutput

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
//! start the discover mode
status = rsi_bt_start_discoverable();
//! start the connectability mode
status = rsi_bt_set_connectable();
//! start the ssp mode
status = rsi_bt_set_ssp_mode(1, 1);
```

12.2.21 rsi_bt_accept_ssp_confirm

Prototype

```
int32_t rsi_bt_accept_ssp_confirm(int8_t *remote_dev_address);
```

Description

This API gives the confirmation for the passkey sent by local BT device at the time of pairing.

Precondition

rsi_bt_set_ssp_mode() API need to be called before this API

Parameters

Parameter	Description
remote_dev_address	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_accept_ssp_confirm(str_conn_bd_addr);
```

12.2.22 rsi_bt_reject_ssp_confirm

Prototype

```
int32_t rsi_bt_reject_ssp_confirm(int8_t *remote_dev_address)
```

Description

This API rejects the confirmation for the passkey sent by the local BT device at the time of pairing.

Precondition

rsi_bt_set_ssp_mode() API need to be called before this API

Parameters

Parameter	Description
remote_dev_address	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_reject_ssp_confirm(remote_dev_addr);
```

12.2.23 rsi_bt_passkey

Prototype

```
int32_t rsi_bt_passkey(int8_t *remote_dev_addr, uint32_t passkey, uint8_t reply_type);
```

Description

This API sends the passkey or rejects the incoming pass key request.

Precondition

rsi_bt_spp_connect() and rsi_bt_on_passkey_request() APIs need to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address
passkey	This is the passkey input
reply_type	This is the positive or negative reply 0 – negative reply 1 – positive reply

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_passkey(remote_dev_addr, 123456, 1);
```

12.2.24 rsi_bt_pincode_request_reply

Prototype

```
int32_t rsi_bt_pincode_request_reply(int8_t *remote_dev_addr, uint8_t *pin_code, uint8_t reply_type);
```

Description

This API sends the pincode or rejects the incoming pincode request.

Precondition

rsi_bt_ssp_mode() and rsi_bt_app_on_pincode_req() APIs need to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address
pin_code	This is the pincode input
reply_type	This is the positive or negative reply 0 – negative reply 1 – positive reply

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define PIN_CODE "4321"
int32_t status = 0;
status = rsi_bt_pincode_request_reply(str_conn_bd_addr, PIN_CODE, 1);
```

12.2.25 rsi_bt_linkkey_request_reply

Prototype

```
int32_t rsi_bt_linkkey_request_reply (int8_t *remote_dev_addr, uint8_t *linkkey, uint8_t reply_type);
```

Description

This API sends either positive (along with the link key) or negative reply to the incoming link key request.

Precondition

rsi_wireless_init() API need to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	Remote device address
linkkey	Linkkey input
reply_type	Positive or negative reply 0 – negative reply 1 – positive reply

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
//! sending the linkkey request negative reply
status = rsi_bt_linkkey_request_reply (str_conn_bd_addr, NULL, 0);
```

12.2.26 rsi_bt_get_local_device_role

Prototype

```
int32_t rsi_bt_get_local_device_role(int8_t *remote_dev_addr, uint8_t *resp);
```

Description

This API requests the role of a local device.

Precondition

rsi_bt_set_ssp_mode() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address
resp	This parameter is to hold the response of this API

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_get_local_device_role(&remote_dev_addr, resp);
```

12.2.27 rsi_bt_set_local_device_role

Prototype

```
int32_t rsi_bt_set_local_device_role(uint8_t  *remote_dev_addr,uint8_t set_role, uint8_t *resp);
```

Description

This API sets the role of a local device.

Precondition

rsi_bt_set_ssp_mode() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address
set_role	This paramets sets either Master/Slave Role 0 → Master Role 1 → Slave Role
resp	This parameter is to hold the response of this API

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_set_local_device_role(&remote_dev_addr, set_role, resp);
```

12.2.28 rsi_bt_sniff_mode

Prototype

```
int32_t rsi_bt_sniff_mode(uint8_t *remote_dev_addr, uint16_t sniff_max_intv, uint16_t sniff_min_intv,
uint16_t sniff_attempt, uint16_t sniff_tout);
```

Description

This API requests the local device to enter into sniff mode.

Precondition

rsi_bt_spp_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address

Parameter	Description
sniff_max_intv	This is the sniff maximum interval
sniff_min_intv	This is the sniff minimum interval
sniff_attempt	This is the sniff attempt
sniff_tout	This is the sniff timeout

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_sniff_mode(&remote_dev_addr, sniff_max_intv, sniff_min_intv, sniff_attempt,
sniff_tout);
```

12.2.29 rsi_bt_sniff_exit_mode

Prototype

```
int32_t rsi_bt_sniff_exit_mode(uint8_t *remote_dev_addr);
```

Description

This API is used to request the local device to exit from sniff/sniff subrating mode.

Precondition

rsi_bt_sniff_mode() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_sniff_exit_mode(&remote_dev_addr);
```

12.2.30 rsi_bt_sniff_subrating_mode

Prototype

```
int32_t rsi_bt_sniff_subrating_mode(uint8_t *remote_dev_addr, uint16_t max_latency, uint16_t
min_remote_tout, uint16_t min_local_tout);
```

Description

This API requests the device entered into the sniff sub rating mode.

Precondition

rsi_bt_sniff_mode() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	Remote device address
max_latency	Maximum latency
min_remote_tout	Minimum remote timeout
min_local_tout	Minimum local timeout

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_sniff_subrating_mode(&remote_dev_addr, max_latency, min_remote_tout, min_local_tout);
```

12.2.31 rsi_bt_add_device_id

Prototype

```
int32_t rsi_bt_add_device_id(rsi_bt_add_device_id(uint16_t spec_id, uint16_t vendor_id, uint16_t
product_id, uint16_t version, int primary_rec, uint16_t vendor_id_source);
```

Description

This API adds device Identification in SDP protocol.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
SpecificationID	Version number of the Bluetooth Device ID Profile specification supported by the device.
VendorID	Uniquely identify the vendor of the device.
ProductID	To distinguish between different products made by the vendor
Version	A numeric expression identifying the device release number in Binary-Coded Decimal
PrimaryRecord	Set to TRUE in the case single Device ID Service Record exists in the device. If multiple Device ID Service Records exist, and no primary record has been defined, set to FALSE.
VendorIDSource	This attribute designates which organization assigned the VendorID attribute, 0x201.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_add_device_id(spec_id, vendor_id, product_id, version, primary_rec, vendor_id_source);
```

12.2.32 rsi_bt_enable_authentication

Prototype

```
int32_t rsi_bt_enable_authentication(void)
```

Description

This function is used to request the local device to enable authentication.

Precondition

rsi_wireless_init() API needs to be called before this API.

Return Values

0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_enable_authentication();
```

12.2.33 rsi_bt_disable_authentication

Prototype

```
int32_t rsi_bt_disable_authentication(void);
```

Description

This function is used to request the local device to disable authentication.

Precondition

rsi_wireless_init() API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_disable_authentication();
```

12.2.34 rsi_bt_get_authentication

Prototype

```
int32_t rsi_bt_get_authentication(void);
```

Description

This function is used to request authentication.

Precondition

rsi_wireless_init() API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_get_authentication();
```

12.2.35 rsi_bt_ptt_req

Prototype

```
int32_t rsi_bt_ptt_req(uint8_t mode);
```

Description

This function is used to configure the LP HP transitions in the controller.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
Mode	BR/EDR mode

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_ptt_req(mode);
```

12.2.36 rsi_bt_write_current_iac_lap

Prototype

```
int32_t rsi_bt_write_current_iac_lap(uint8_t no_of_iaps, uint8_t *iap_lap_list);
```

Description

This function is used to write the current iac lap to the controller.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
no_of_iaps	number of iap lap count
iap_lap_list	pointer to the iap laps list

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_write_current_iac_lap(no_of_iaps, iap_lap_list);
```

12.2.37 rsi_bt_get_services_async

Prototype

```
int32_t rsi_bt_get_services_async(uint8_t *remote_dev_addr, rsi_bt_resp_query_services_t
*bt_resp_query_services);
```

Description

This function is to query the services of local devices.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	Remote device address
service_uuid	uuid of the service for search

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_get_services_async(remote_dev_addr, &bt_resp_query_services);
```

12.2.38 rsi_bt_search_service_async

Prototype

```
int32_t rsi_bt_search_service_async(uint8_t *remote_dev_addr, uint32_t service_uuid);
```

Description

This function is used to search service of the given uuid.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	Remote device address
service_uuid	uuid of the service for search

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_search_service_async(remote_dev_addr, service_uuid);
```

12.3 SPP API

12.3.1 rsi_bt_spp_init

Prototype

```
int32_t rsi_bt_spp_init(void);
```

Description

This API sets the SPP profile mode.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
//! start the discover mode
status = rsi_bt_start_discoverable();
//! start the connectability mode
status = rsi_bt_set_connectable();
//! initilize the SPP profile
status = rsi_bt_spp_init();
```

12.3.2 rsi_bt_spp_connect

Prototype

```
int32_t rsi_bt_spp_connect(uint8_t *remote_dev_addr);
```

Description

This API initiates the SPP profile level connection.

Precondition

rsi_bt_spp_init() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define REMOTE_BD_ADDR        "00:1A:57:45:32:BA"
int32_t status = 0;
status = rsi_bt_spp_connect(REMOTE_BD_ADDR);
```

12.3.3 rsi_bt_spp_disconnect

Prototype

```
int32_t rsi_bt_spp_disconnect(uint8_t *remote_dev_addr);
```

Description

This API initiates the SPP service level disconnection.

Precondition

rsi_bt_spp_connect() API need to be called before this API

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_spp_disconnect(&remote_dev_addr);
```

12.3.4 rsi_bt_spp_transfer

Prototype

```
int32_t rsi_bt_spp_transfer(uint8_t *remote_dev_addr, uint8_t *data, uint16_t length);
```

Description

This API transfers the data through SPP profile.

Precondition

rsi_bt_spp_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address
Data	This is the data for transmission
Length	This is the data length for transfer, Max length supported upto 1000 bytes

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
uint8_t data[32];
uint16_t data_len = 0;
strcpy((char *)data, "SAMPLE_TEST");
data_len = strlen((char *)data);
```

```
//! SPP data transfer (loop back)
status = rsi_bt_spp_transfer (str_conn_bd_addr, data, data_len);
```

12.4 SDP API

12.4.1 rsi_bt_set_sdp_attr_id

Prototype

```
uint32_t  rsi_bt_set_sdp_attr_id(rsi_sdp_att_record_t *att_rec,uint16_t attr_id, uint16_t
pattr_buf_idx,uint16_t  attr_buf_len);
```

Description

This function is used to sets the Attribute ID.

Parameters

Parameter	Description
att_rec	sdp attribute record pointer
attr_id	sdp attribute id
pattr_buf_idx	pattern buffer idx
attr_buf_len	attribute buffer length

Structure Prototype

```
typedef struct rsi_sdp_att_record_s
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;


typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx **attr_buf_len :** attribute buffer pointer
attr_array_cnt	attribute array count
buf_len	buffer length
buf_array	buffer array

Return values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_set_sdp_attr_id(&att_rec, attr_id, pattr_buf_idx, attr_buf_len);
```

12.4.2 rsi_bt_add_sdp_attribute

Prototype

```
uint32_t  rsi_bt_add_sdp_attribute(rsi_sdp_att_record_t *att_rec, uint16_t  attr_id, uint8_t
att_data_8, uint16_t att_data_16,uint8_t is_boolean, uint8_t param_len);
```

Description

This function is used to add the attribute data to the records pointer.

Parameters

Parameter	Description
att_rec	sdp attribute record pointer
attr_id	sdp attribute id
att_data_8	attribute data 8
att_data_16	attribute data 16
is_boolean	boolean data type or unsigned Integer type
param_len	param lengh

Structure Prototype

```
typedef struct rsi_sdp_att_record_s
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;


typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx **attr_buf_len :** attribute buffer pointer
attr_array_cnt	attribute array count

Variables	Description
buf_len	buffer length
buf_array	buffer array

Return values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t
rsi_bt_add_sdp_attribute(&att_rec,attr_id, att_data_8, att_data_16,is_boolean, param_len);
```

12.4.3 rsi_bt_add_sdp_hid_language_attribute

Prototype

```
uint32_t rsi_bt_add_sdp_hid_language_attribute(rsi_sdp_att_record_t  *att_record_data, uint16_t
lang_id, uint16_t lang_attr_base)
```

Description

This function is used to add the sdp hid language attribute to the sdp service record.

Parameters

Parameter	Description
att_record_data	sdp attribute record pointer
lang_id	language ID
lang_attr_base	language attribute base

Structure Prototype

```
typedef struct rsi_sdp_att_record_s
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;


typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx **attr_buf_len :** attribute buffer pointer

Variables	Description
attr_array_cnt	attribute array count
buf_len	buffer length
buf_array	buffer array

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_add_sdp_hid_language_attribute(&att_record_data, lang_id, lang_attr_base);
```

12.4.4 rsi_bt_add_sdp_hid_descriptor_list

Prototype

```
uint32_t rsi_bt_add_sdp_hid_descriptor_list(rsi_sdp_att_record_t  *att_record_data, uint8_t *buff_ptr,
uint8_t buff_len)
```

Description

This function is used to add the attribute data to the records pointer.

Parameters

Parameter	Description
att_record_data	sdp attribute record pointer
buff_ptr	buffer pointer
buff_len	buffer length

Structure Prototype

```
typedef struct rsi_sdp_att_record_s
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;


typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx

Variables	Description
	attr_buf_len : attribute buffer pointer
attr_array_cnt	attribute array count
buf_len	buffer length
buf_array	buffer array

Return values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_add_sdp_hid_descriptor_list(&att_record_data, buff_ptr, buff_len);
```

12.4.5 rsi_bt_add_sdp_service_attribute

Prototype

```
uint32_t rsi_bt_add_sdp_service_attribute(rsi_sdp_att_record_t *att_rec, char *service_name, uint8_t
name_len, uint16_t attr_id);
```

Description

This function is used to add the service name attribute to the service record.

Parameters

Parameter	Description
att_record_data	sdp attribute record pointer
service_name	Service name
name_len	name length
attr_id	attribute ID

Structure Prototype

typedef struct rsi_sdp_att_record_s

```
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;


typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx **attr_buf_len :** attribute buffer pointer
attr_array_cnt	attribute array count
buf_len	buffer length
buf_array	buffer array

Return values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_add_sdp_service_attribute(&att_rec, service_name, name_len, attr_id);
```

12.4.6 rsi_bt_add_sdp_service_classid

Prototype

```
uint32_t  rsi_bt_add_sdp_service_classid(rsi_sdp_att_record_t *att_rec, uint16_t serv_class_uuid);
```

Description

This function is used to add the service name attribute to the service record.

Parameters

Value	Description
att_rec	sdp_attribute_record_pointer
serv_class_uuid	service class uuid

Structure Prototype

```
typedef struct rsi_sdp_att_record_s
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;


typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx **attr_buf_len :** attribute buffer pointer
attr_array_cnt	attribute array count
buf_len	buffer length
buf_array	buffer array

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_add_sdp_service_classid(&att_rec, serv_class_uuid);
```

12.4.7 rsi_bt_add_sdp_service_handle

Prototype

uint32_t rsi_bt_add_sdp_service_handle(rsi_sdp_att_record_t *att_rec, uint32_t serv_hndl);

Description

This function is used to add the service name attribute to the service record.

Parameters

Value	Description
att_rec	sdp_attribute_record_pointer
serv_class_uuid	service class uuid

Structure Prototype

```
typedef struct rsi_sdp_att_record_s
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;


typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx **attr_buf_len :** attribute buffer pointer
attr_array_cnt	attribute array count
buf_len	buffer length
buf_array	buffer array

Return values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_add_sdp_service_handle(&att_rec, serv_hndl);
```

12.4.8 rsi_bt_add_sdp_protocol_list

Prototype

```
uint32_t rsi_bt_add_sdp_protocol_list(rsi_sdp_att_record_t *att_rec, bt_sdp_proto_desc_list_elem_t
*list, uint8_t list_cnt, uint16_t attr_id);
```

Description

This function is used to add the sdp protocol attribute list to the service record

Parameters

Value	Description
att_rec	sdp_attribute_record_pointer
list	SDP protocol list
attr_id	attribute id

Structure Prototype

typedef struct rsi_sdp_att_record_s

```
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;


typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx **attr_buf_len :** attribute buffer pointer
attr_array_cnt	attribute array count
buf_len	buffer length
buf_array	buffer array

Return values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_add_sdp_protocol_list(&att_rec, &list, list_cnt, attr_id);
```

12.4.9 rsi_bt_add_sdp_language_base_attributeid_list

Prototype

```
uint32_t rsi_bt_add_sdp_language_base_attributeid_list(rsi_sdp_att_record_t *att_rec,
bt_sdp_lang_attr_id_elem_t *list, uint8_t list_cnt);
```

Description

This function is used to add the sdp language base attribute list to the service record.

Parameter values

Value	Description
att_rec	sdp_attribute_record_pointer
list	bt sdp language attribute element list
list_cnt	list count

Structure Prototype

```
typedef struct rsi_sdp_att_record_s
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;


typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
```

```
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx **attr_buf_len :** attribute buffer pointer
attr_array_cnt	attribute array count
buf_len	buffer length
buf_array	buffer array

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_add_sdp_language_base_attributeid_list(&att_rec, &list, list_cnt);
```

12.4.10 rsi_bt_add_sdp_profile_descriptor_list

Prototype

```
uint32_t  rsi_bt_add_sdp_profile_descriptor_list(rsi_sdp_att_record_t *att_rec, uint16_t profile_uuid,
uint16_t profile_version);
```

Description

This function is used to add the sdp profile descriptor to the sdp service record.

Parameters

Value	Description
att_rec	sdp_attribute_record_pointer
profile_uuid	profile uuid
profile_version	Profile version

Structure Prototype

```
typedef struct rsi_sdp_att_record_s
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;
```

```
typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx **attr_buf_len :** attribute buffer pointer
attr_array_cnt	attribute array count
buf_len	buffer length
buf_array	buffer array

Return values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_add_sdp_profile_descriptor_list(&att_rec, profile_uuid, profile_version);
```

12.4.11 rsi_bt_add_sdp_service_record_handle

Prototype

```
uint32_t  rsi_bt_add_sdp_service_record_handle(rsi_sdp_att_record_t *att_rec, uint32_t serv_hndl);
```

Description

This function is used to add the sdp service handle attribute list to the service record.

Parameters

Value	Description
att_rec	sdp_attribute_record_pointer
serv_hndl	Service Handle

Structure Prototype

```
typedef struct rsi_sdp_att_record_s
{
#define MAX_SDP_ATTR 27
 attr_id_array_t attr_array[MAX_SDP_ATTR];
 uint16_t attr_array_cnt;
#define MAX_SDP_BUFF_LEN 500
 uint16_t buf_len;
 uint8_t  buf_array[MAX_SDP_BUFF_LEN];
}rsi_sdp_att_record_t;
```

```
typedef struct attr_id_array_s
{
 uint16_t attr_id;
 uint16_t attr_buf_ptr_idx;
 uint16_t attr_buf_len;
}attr_id_array_t;
```

Structure Variables

Variables	Description
attr_array	sdp attribute record array **attr_id :** attribute record **attr_buf_ptr_idx :** attribute buffer pointer idx **attr_buf_len :** attribute buffer pointer
attr_array_cnt	attribute array count
buf_len	buffer length
buf_array	buffer array

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_add_sdp_service_record_handle(&att_rec, serv_hndl);
```

12.5 HID API

12.5.1 rsi_bt_hid_service_initialize

Prototype

```
int32_t rsi_bt_hid_service_initialize(char *service_name, char *service_description,
                                      char *service_provider, uint8_t *desc_buf, uint16_t
desc_buf_len);
```

Description

This API is used to initialize the device with hid services.

Precondition

rsi_driver_init(), rsi_device_init() and rsi_wireless_init() are to be done.

Parameters

Parameter	Description
service_name	Name of the Service in SDP Attribute
service_description	Description of the Service in SDP Attribute
service_provider	Name of the Service Provider in SDP Attribute
desc_buf	Pointer to the buffer with Report Descriptor Data field
desc_buf_len	Length of Descriptor buffer

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_hid_service_initialize("Keyboard2", "BT Widget", "Silicon Labs", hid_desc_160_data,
160);
```

12.5.2 rsi_bt_hid_init

Prototype

```
int32_t rsi_bt_hid_init (void)
```

Description

This API is used to initialize the device with hid profile.

Precondition

rsi_bt_hid_service_initialize() is to be done.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_hid_init(void);
```

12.5.3 rsi_bt_hid_connect

Prototype

```
int32_t rsi_bt_hid_connect(uint8_t *remote_dev_addr)
```

Description

This function is used to request HID connection.

Precondition

This API is used to initiate HID connection.

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
uint8_t  str_conn_bd_addr =
status = rsi_bt_hid_connect(str_conn_bd_addr);
```

12.5.4 rsi_bt_hid_disconnect

Prototype

```
int32_t rsi_bt_hid_disconnect(uint8_t *remote_dev_addr);
```

Description

This API is used to disconnect the device hid protocol.

Precondition

HID connection is established and active.

Parameters

Parameter	Description
remote_dev_addr	remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = int32_t rsi_bt_hid_disconnect(remote_connected_bd_addr);
```

12.5.5 rsi_bt_hid_register_callbacks

Prototype

```
void rsi_bt_hid_register_callbacks (
    rsi_bt_on_hid_connect_t            bt_on_hid_connect_event,
    rsi_bt_on_hid_rx_data_t            bt_on_hid_rx_data_event,
    rsi_bt_on_hid_handshake_t          bt_on_hid_handshake_event,
    rsi_bt_on_hid_control_t            bt_on_hid_control_event,
    rsi_bt_on_hid_get_report_t         bt_on_hid_get_report,
    rsi_bt_on_hid_set_report_t         bt_on_hid_set_report,
    rsi_bt_on_hid_get_protocol_t       bt_on_hid_get_proto,
    rsi_bt_on_hid_set_protocol_t       bt_on_hid_set_proto
    );
```

Description

This API is used to register the hid event callbacks for received HID messages.

Precondition

rsi_device_init() is to be done.

Parameters

Parameter	Description
bt_on_hid_connect_event	callback on HID connection event is received
bt_on_hid_rx_data_event	callback on HID RX data event is received for both control and interrupt channels
bt_on_hid_handshake_event	callback on HID profile handshake msg is received

Parameter	Description
bt_on_hid_control_event	callback on HID profile control msg is received
bt_on_hid_get_report	callback on HID profile get report msg is received
bt_on_hid_set_report	callback on HID profile set report msg is received
bt_on_hid_get_proto	callback on HID profile get protocol msg is received
bt_on_hid_set_proto	callback on HID profile set protocol msg is received

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;

status =   rsi_bt_hid_register_callbacks (rsi_bt_app_on_hid_connect,
                                rsi_bt_app_on_hid_data_rx,
                                NULL,
                                rsi_bt_app_on_hid_control_msg,
                                rsi_bt_app_on_hid_get_report_msg,
                                rsi_bt_app_on_hid_set_report_msg,
                                rsi_bt_app_on_hid_get_protocol_msg,
                                rsi_bt_app_on_hid_set_protocol_msg
                                );
```

12.5.6 rsi_bt_hid_send_interrupt_data

Prototype

```
int32_t rsi_bt_hid_send_interrupt_data(uint8_t *remote_dev_addr, uint8_t *data, uint16_t data_len)
```

Description

This API is used to send interrupt data to host using hid protocol.

Precondition

HID connection is established and active.

Parameters

Parameter	Description
remote_dev_addr	Address of remote device
Data	pointer to data to send
data_len	length of data to send

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_hid_send_interrupt_data(str_conn_bd_addr, hid_int_data_kb_down, hid_int_data_len);
```

12.5.7 rsi_bt_hid_profile_data

Prototype

```
int32_t rsi_bt_hid_profile_data(uint8_t *remote_dev_addr, uint8_t *data, uint16_t len, uint8_t cid)
```

Precondition

None

Parameters

Parameter	Description
remote_dev_addr	Address of remote device
Data	pointer to data to send
Len	length of data to send

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_hid_profile_data(uint8_t *remote_dev_addr, uint8_t *data, uint16_t len, uint8_t cid)
```

12.5.8 rsi_bt_hid_send_handshake

Prototype

```
int32_t rsi_bt_hid_send_handshake(uint8_t *remote_dev_addr,uint8_t status)
```

Precondition

None

Parameters

Parameter	Description
remote_dev_addr	Address of remote device
status	Result Code received in Handshake message for previous control message

Return Values

Value	Description
0	Successful execution of the command
<0	Failure
1	NOT_READY
2	ERR_INVALID_REPORT_ID
3	ERR_UNSUPPORTED_REQUEST
4	ERR_INVALID_PARAMETER
14	ERR_UNKNOWN

Value	Description
15	ERR_FATAL

Example

```
int32_t status = 0;
status = rsi_bt_hid_send_handshake(uint8_t *remote_dev_addr,uint8_t status)
```

12.5.9 rsi_bt_hid_send_control

Prototype

```
int32_t rsi_bt_hid_send_control(uint8_t *remote_dev_addr,uint8_t control)
```

Precondition

None

Parameters

Parameter	Description
remote_dev_addr	Address of remote device
control	Control Operation received in Control message

Return Values

Value	Description
0	Success on execution of command
<0	Failure

Example

```
int32_t status = 0;
status = rsi_bt_hid_send_control(uint8_t *remote_dev_addr,uint8_t control)
```

12.5.10 rsi_bt_hid_get_report

Prototype

```
int32_t rsi_bt_hid_get_report(uint8_t *remote_dev_addr, uint8_t report_id,uint8_t report_type)
```

Precondition

None

Parameters

Parameter	Description
remote_dev_addr	Address of remote device
report_type	Report Type of the data received 1 - Input 2 - Output 3 - Feature
report_id	Report Id requested by the host

Return Values

Value	Description
0	Success on execution of command
<0	Failure

Example

```
int32_t status = 0;
status = rsi_bt_hid_send_control(uint8_t *remote_dev_addr,uint8_t control)
```

12.5.11 rsi_bt_hid_set_report

Prototype

```
int32_t rsi_bt_hid_set_report(uint8_t *remote_dev_addr, uint8_t report_id,uint8_t report_type, uint8_t
*data, uint16_t data_len)
```

Precondition

None

Parameters

Parameter	Description
remote_dev_addr	Address of remote device
report_type	Report Type of the data received 1 - Input 2 - Output 3 - Feature
report_id	Report Id requested by the host
Data	HEX format of data separated by comma sent by the Host
Len	Length of data from the host

Return Values

Value	Description
0	Success on execution of command
<0	Failure

Example

```
int32_t status = 0;
status = rsi_bt_hid_set_report(uint8_t *remote_dev_addr, uint8_t report_id,uint8_t report_type, uint8_t
*data, uint16_t data_len)
```

12.5.12 rsi_bt_hid_get_protocol

Prototype

```
int32_t rsi_bt_hid_get_protocol(uint8_t *remote_dev_addr)
```

Precondition

None

Parameters

Parameter	Description
remote_dev_addr	Address of remote device

Return Values

Value	Description
0	Success on execution of command
<0	Failure

Example

```
int32_t status = 0;
status = rsi_bt_hid_get_protocol(uint8_t *remote_dev_addr)
```

12.5.13 rsi_bt_hid_set_protocol

Prototype

```
int32_t rsi_bt_hid_set_protocol(uint8_t *remote_dev_addr, uint8_t protocol)
```

Precondition

None

Parameters

Parameter	Description
remote_dev_addr	Address of remote device
protocol	Protocol Mode the Device needs to set

Return Values

Value	Description
0	Success on execution of command
<0	Failure

Example

```
int32_t status = 0;
status =rsi_bt_hid_set_protocol(uint8_t *remote_dev_addr, uint8_t protocol)
```

12.5.14 rsi_bt_hid_set_protocol

Prototype

```
int32_t rsi_bt_hid_set_protocol(uint8_t *remote_dev_addr, uint8_t protocol)
```

Precondition

None

Parameters

Parameter	Description
remote_dev_addr	Address of remote device
protocol	Protocol Mode the Device needs to set

Return Values

Value	Description
0	Success on execution of command
<0	Failure

Example

```
int32_t status = 0;
status =rsi_bt_hid_set_protocol(uint8_t *remote_dev_addr, uint8_t protocol)
```

12.6 GAP Register Callbacks

This function register GAP callbacks.

12.6.1 rsi_bt_gap_register_callbacks

prototype

```
void rsi_bt_gap_register_callbacks (
    rsi_bt_on_role_change_t               bt_on_role_change_status_event,
    rsi_bt_on_connect_t                   bt_on_conn_status_event,
    rsi_bt_on_unbond_t                    bt_on_unbond_status ,
    rsi_bt_on_disconnect_t                bt_on_disconnect_event,
    rsi_bt_on_scan_resp_t                 bt_on_scan_resp_event,
    rsi_bt_on_remote_name_resp_t          bt_on_remote_name_resp_event,
    rsi_bt_on_passkey_display_t           bt_on_passkey_display_event,
    rsi_bt_on_remote_name_request_cancel_t          bt_on_remote_name_request_cancel_event,
    rsi_bt_on_confirm_request_t           bt_on_confirm_request_event,
    rsi_bt_on_pincode_request_t           bt_on_pincode_request_event,
    rsi_bt_on_passkey_request_t           bt_on_passkey_request_event,
    rsi_bt_on_inquiry_complete_t          bt_on_inquiry_complete_event,
    rsi_bt_on_auth_complete_t             bt_on_auth_complete_event,
    rsi_bt_on_linkkey_request_t           bt_on_linkkey_request_event,
    rsi_bt_on_ssp_complete_t              bt_on_ssp_complete_event,
    rsi_bt_on_linkkey_save_t              bt_on_linkkey_save_event,
    rsi_bt_on_get_services_t              bt_on_get_services_event,
    rsi_bt_on_search_service_t            bt_on_search_service_event,
    rsi_bt_on_mode_chnage_t               bt_on_mode_change_event,
    rsi_bt_on_sniff_subrating_t           bt_on_sniff_subrating_event
    );
```

Description

This function registers the GAP callbacks.

Parameters

bt_on_role_change_status_event	Role change status event callback
bt_on_conn_status_event	Connection status event callback
bt_on_unbond_status	Unbond status callback
bt_on_disconnect_event	Disconnection status callback
bt_on_scan_resp_event	Scan report callback
bt_on_remote_name_resp_event	Remote name report callback
bt_on_passkey_display_event	Passkey display report callback
bt_on_remote_name_request_cancel_event	Remote name request cancel status callback
bt_on_confirm_request_event	Authentication status callback
bt_on_pincode_request_event	Pincode request status callback
bt_on_passkey_request_event	Passkey request status callback

bt_on_inquiry_complete_event	Inquiry report callback
bt_on_auth_complete_event	Authentication status callback
bt_on_linkkey_request_event	Linkkey request report callback
bt_on_ssp_complete_event	SSP status callback
bt_on_linkkey_save_event	Linkkey save status callback
bt_on_get_services_event	Get services report callback
bt_on_search_service_event	Search service status callback
bt_on_mode_change_event	mode change callback
bt_on_sniff_subrating_event	sniff subrating callback

Return Values

None

12.6.2 rsi_bt_on_role_change_t

Prototype

typedef void (*rsi_bt_on_role_change_t) (uint16_t resp_status, rsi_bt_event_role_change_t *role_change_status);

Description

This callback function will be called if the role change status event is received from the module.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
role_change_status	Structure pointer holding the response payload for the role change status event This is the structure pointer for rsi_bt_event_role_change_t structure

Structure Prototype

```
typedef struct rsi_bt_event_role_change_s
{
   uint8_t dev_addr[RSI_DEV_ADDR_LEN];
   uint8_t role;
} rsi_bt_event_role_change_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
role	Currenty role of the local device

12.6.3 rsi_bt_on_connect_t

Prototype

```
typedef void (*rsi_bt_on_connect_t) (uint16_t resp_status, rsi_bt_event_bond_t *bond_response);
```

Description

This callback function is called if a new connection completion is received from the module. This event will be given by the module in the following two scenarios:

- In case of slave mode, when the connection is initiated from the remote device.
- In case of Master mode, when the connect command is issued to connect to a remote device.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
bond_response	Structure pointer holding the response payload for the connect response event This is the structure pointer for rsi_bt_event_bond_t structure

Structure Prototype

```
typedef struct rsi_bt_event_bond_response_s
{
   uint8_t dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_bond_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

12.6.4 rsi_bt_on_unbond_t

Prototype

```
typedef void (*rsi_bt_on_unbond_t) (uint16_t resp_status, rsi_bt_event_unbond_t *unbond_status);
```

Description

This callback is called when unbond event is raised from the module. This event will be given by the module when either slave or master device issues unbond command to the other.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
unbond_status	Structure pointer holding the response payload for the unbond response event This is the structure pointer for rsi_bt_event_unbond_t structure

Structure Prototype

```
typedef struct rsi_bt_event_unbond_s
{
   uint8_t dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_unbond_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

12.6.5 rsi_bt_on_disconnect_t

Prototype

```
typedef void (*rsi_bt_on_disconnect_t) (uint16_t resp_status, rsi_bt_event_disconnect_t
*bt_disconnect);
```

Description

This callback is called when disconnection event is raised from the module. This event will be given by the module when either slave or master device issues disconnect command to the other.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
bt_disconnect	Structure pointer holding the response payload for the disconnect response event This is the structure pointer for rsi_bt_event_disconnect_t structure

Structure Prototype

```
typedef struct rsi_bt_event_disconnect_s
{
   uint8_t dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_disconnect_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

12.6.6 rsi_bt_on_scan_resp_t

Prototype

```
typedef void (*rsi_bt_on_scan_resp_t) (uint16_t resp_status, rsi_bt_event_inquiry_response_t
*single_scan_resp);
```

Description

This callback function will be called if the single scan response is received from the module in response to inquiry command.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
single_scan_resp	Structure pointer holding the response payload for the inquiry response event This is the structure pointer for rsi_bt_event_inquiry_response_t structure

Structure Prototype

```
typedef struct rsi_bt_inquiry_response_s
{
   uint8_t  inquiry_type;
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
   uint8_t  name_length;
   uint8_t  remote_device_name[RSI_DEV_NAME_LEN];
   uint8_t  cod[3];
   uint8_t  rssi;
} rsi_bt_event_inquiry_response_t;
```

Structure Variables

Variables	Description
inquiry_type	Inquiry scan type
dev_addr	This array holds the device address of length 6 bytes
name_length	Name length of the remote device
remote_device_name	This is the name of remote device. This is an array of max length 50 bytes
cod	Class of Device
rssi	RSSI of the remote device

12.6.7 rsi_bt_on_remote_name_resp_t

Prototype

```
typedef void (*rsi_bt_on_remote_name_resp_t) (uint16_t resp_status, rsi_bt_event_remote_device_name_t
*name_resp);
```

Description

This callback is called if the remote name request command response is received from the module.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
name_resp	Structure pointer holding the response payload for the name request command response event This is the structure pointer for rsi_bt_event_remote_device_name_t structure

Structure Prototype

```
typedef struct rsi_bt_event_remote_device_name_s
{
   uint8_t dev_addr[RSI_DEV_ADDR_LEN];
   uint8_t name_length;
   uint8_t remote_device_name[RSI_BT_DEVICE_NAME_LEN];
} rsi_bt_event_remote_device_name_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
name_length	Name length of the remote device
remote_device_name	This is the name of remote device. This is an array of max length 50 bytes

12.6.8 rsi_bt_on_passkey_display_t

Prototype

```
typedef void (*rsi_bt_on_passkey_display_t) (uint16_t resp_status, rsi_bt_event_user_passkey_display_t
*bt_event_user_passkey_display);
```

Description

This callback function is called if the passkey display request is received from the module.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
bt_event_user_passkey_display	Structure pointer holding the response payload for the passkey display request command response event This is the structure pointer for rsi_bt_event_user_passkey_display_t structure

Structure Prototype

```
typedef struct rsi_bt_event_user_passkey_display_s
{
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
   uint8_t  reserved[2];
   uint8_t  passkey[4];
} rsi_bt_event_user_passkey_display_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
reserved	Reserved field for future use
passkey	Passkey value

12.6.9 rsi_bt_on_remote_name_request_cancel_t

Prototype

```
typedef void (*rsi_bt_on_remote_name_request_cancel_t) (uint16_t resp_status,
rsi_bt_event_remote_name_request_cancel_t *remote_name_request_cancel);
```

Description

This callback is called if the remote name request cancels the command response received from the module.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
remote_name_request_cancel	Structure pointer holding the response payload for the name request cancel command response event

Variables	Description
	This is the structure pointer for rsi_bt_event_remote_name_request_cancel_t structure

Structure Prototype

```
typedef struct rsi_bt_event_remote_name_request_cancel_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_remote_name_request_cancel_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

12.6.10 rsi_bt_on_confirm_request_t

Prototype

```
typedef void (*rsi_bt_on_confirm_request_t) (uint16_t resp_status,
rsi_bt_event_user_confirmation_request_t *user_confirmation_request);
```

Description

This callback function is called if the user confirmation request is received from the module. The user has to give rsi_bt_accept_ssp_confirm or rsi_bt_reject_ssp_confirm command upon reception of this event.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
user_confirmation_request	Structure pointer holding the response payload for the user confirmation request command response event This is the structure pointer for rsi_bt_event_user_confirmation_request_t structure

Structure Prototype

```
typedef struct rsi_bt_event_user_confirmation_request_s
{
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
   uint8_t  reserved[2];
   uint8_t  confirmation_value[4];
} rsi_bt_event_user_confirmation_request_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
reserved	Reserved field for future use
confirmation_value	This is the passkey to be confirmed

12.6.11 rsi_bt_on_pincode_request_t

Prototype

```
typedef void (*rsi_bt_on_pincode_request_t) (uint16_t resp_status, rsi_bt_event_user_pincode_request_t
*user_pincode_request);
```

Description

This callback function is called if pincode request is received from the module. User has to give rsi_bt_accept_pincode_request command upon reception of this event.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
user_pincode_request	Structure pointer holding the response payload for the pincode request response event. This is the structure pointer for rsi_bt_event_user_pincode_request_cancel_t structure

Structure Prototype

```
typedef struct rsi_bt_event_user_pincode_request_s
{
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_user_pincode_request_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

12.6.12 rsi_bt_on_passkey_request_t

Prototype

```
typedef void (*rsi_bt_on_passkey_request_t) (uint16_t resp_status, rsi_bt_event_user_passkey_request_t
*user_passkey_request);
```

Description

This callback function is called if the passkey request is received from the module. User has to give rsi_bt_passkey command upon reception of this event.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
user_passkey_request	Structure pointer holding the response payload for the passkey request response event This is the structure pointer for rsi_bt_event_user_passkey_request_t structure

Structure Prototype

```
typedef struct rsi_bt_event_user_passkey_request_s
{
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_user_passkey_request_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

12.6.13 rsi_bt_on_inquiry_complete_t

Prototype

```
typedef void (*rsi_bt_on_inquiry_complete_t) (uint16_t resp_status);
```

Description

This callback function is called if inquiry complete status is received from the module. This event will be given by the module when inquiry command is completely executed.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure

12.6.14 rsi_bt_on_auth_complete_t

Prototype

```
typedef void (*rsi_bt_on_auth_complete_t) (uint16_t resp_status, rsi_bt_event_auth_complete_t
*auth_complete);
```

Description

This callback function is called if authentication complete indication is received from the module.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
auth_complete	Structure pointer holding the response payload for the authentication complete indication event This is the structure pointer for rsi_bt_event_auth_complete_t structure

Structure Prototype

```
typedef struct rsi_bt_event_auth_complete_s
{
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_auth_complete_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

12.6.15 rsi_bt_on_linkkey_request_t

Prototype

```
typedef void (*rsi_bt_on_linkkey_request_t) (uint16_t resp_status, rsi_bt_event_user_linkkey_request_t
*user_linkkey_request);
```

Description

This callback is called if linkkey request is received from the module. User has to give linkkey reply command upon reception of this event.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
user_linkkey_request	Structure pointer holding the response payload for the user linkkey request event This is the structure pointer for rsi_bt_event_user_linkkey_request_t structure

Structure Prototype

```
typedef struct rsi_bt_event_user_linkkey_request_s
{
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_user_linkkey_request_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

12.6.16 rsi_bt_on_ssp_complete_t

Prototype

```
typedef void (*rsi_bt_on_ssp_complete_t) (uint16_t resp_status, rsi_bt_event_ssp_complete_t
*ssp_complete);
```

Description

This callback function is called if SSP complete status is received from the module.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
ssp_complete	Structure pointer holding the response payload for the ssp complete event This is the structure pointer for rsi_bt_event_ssp_complete_t structure

Structure Prototype

```
typedef struct rsi_bt_event_ssp_complete_s
{
   uint8_t   dev_addr[RSI_DEV_ADDR_LEN];
```

```
   uint8_t   status;
} rsi_bt_event_ssp_complete_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
status	This is the SSP mode connection status with remote device

12.6.17 rsi_bt_on_linkkey_save_t

Prototype

```
typedef void (*rsi_bt_on_linkkey_save_t) (uint16_t resp_status, rsi_bt_event_user_linkkey_save_t
*user_linkkey_save);
```

Description

This callback function is called if linkkey save is received from the module.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
user_linkkey_save	Structure pointer holding the response payload for the linkkey save event This is the structure pointer for rsi_bt_event_user_linkkey_save_t structure

Structure Prototype

```
typedef struct rsi_bt_event_user_linkkey_save_s {
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
   uint8_t  linkKey[RSI_LINK_KEY_LEN];
} rsi_bt_event_user_linkkey_save_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
linkKey	Link Key to be saved

12.6.18 rsi_bt_on_get_services_t

Prototype

```
typedef void (*rsi_bt_on_get_services_t) (uint16_t resp_status, rsi_bt_resp_query_services_t
*service_list);
```

Description

This callback function is called if the get services command response is received from the module.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
service_list	Structure pointer holding the response payload for the get services command response This is the structure pointer for rsi_bt_resp_query_services_t structure

Structure Prototype

```
typedef struct rsi_bt_resp_query_services_s
{
   uint8_t  num_of_services;
   uint8_t  reserved[3];
   uint32_t  uuid[32];
} rsi_bt_resp_query_services_t;
```

Structure Variables

Variables	Description
num_of_services	Number of services fetched
reserved	Reserved field for future use
uuid	Service UUID

12.6.19 rsi_bt_on_search_service_t

Prototype

```
typedef void (*rsi_bt_on_search_service_t) (uint16_t resp_status, uint8_t *remote_dev_addr, uint8_t
status);
```

Description

This callback function is called if the search service command response is received from the module.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
remote_dev_addr	This is the remote device address
status	This is the search service status

12.6.20 rsi_bt_on_mode_change_t

Prototype

```
typedef void (*rsi_bt_on_mode_chnage_t) (uint16_t resp_status, rsi_bt_event_mode_change_t
*mode_change);
```

Description

This callback function is called when the local device enters / exits the Sniff mode.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
mode_change	Structure pointer holding the response payload for the mode change response This is the structure pointer for rsi_bt_event_mode_change_t structure

Structure Prototype

```
typedef struct rsi_bt_event_mode_change_s
{
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
   uint8_t  current_mode;
   uint8_t  reserved;
   uint16_t  mode_interval;
} rsi_bt_event_mode_change_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
current_mode	This indicates the current state of connection between the local device and the remote device i.e., Active mode / Hold mode/ Sniff mode 0 → Active Mode 1 → Hold Mode 2 → Sniff Mode
reserved	Reserved field for future use
mode_interval	This specifies the time interval to each mode

12.6.21 rsi_bt_on_sniff_subrating_t

Prototype

```
typedef void (*rsi_bt_on_sniff_subrating_t) (uint16_t resp_status, rsi_bt_event_sniff_subrating_t
*sniff_subrating);
```

Description

This callback function is called when Sniff subrating is enabled or the parameters are negotiated with the remote device.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
sniff_subrating	Structure pointer holding the response payload for the Sniff subrating response event This is the structure pointer for rsi_bt_event_sniff_subrating_t structure

Structure Prototype

```
typedef struct rsi_bt_event_sniff_subrating_s
{
   uint8_t   dev_addr[RSI_DEV_ADDR_LEN];
   uint16_t  max_tx_latency;
   uint16_t  min_remote_timeout;
   uint16_t  min_local_timeout;
} rsi_bt_event_sniff_subrating_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
max_tx_latency	This is the maximum latency for data being transmitted from the local device to the remote device
min_remote_timeout	This is the base sniff subrate timeout in baseband slots that the remote device use
min_local_timeout	This is the base sniff subrate timeout in baseband slots that the local device use

12.7 SPP event callbacks

This function registers the SPP callbacks.

12.7.1 rsi_bt_spp_register_callbacks

Prototype

```
void rsi_bt_spp_register_callbacks (
    rsi_bt_on_spp_connect_t         bt_on_spp_connect_event,
    rsi_bt_on_spp_disconnect_t      bt_on_spp_disconnect_event,
    rsi_bt_on_spp_rx_data_t         bt_on_spp_rx_data_event
    )
```

Description

This function registers the spp callbacks.

Parameters

Parameter	Description
bt_on_spp_connect_event	spp connection status callback
bt_on_spp_disconnect_event	spp connection status callback
bt_on_spp_rx_data_event	spp data transfer status callback

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

12.7.2 rsi_bt_on_spp_connect_t

Prototype

```
typedef void (*rsi_bt_on_spp_connect_t) (uint16_t resp_status, rsi_bt_event_spp_connect_t
*spp_connect);
```

Description

This callback is called when SPP connected event is raised from the module. This event will be given by the module when spp profile level connection happens from either side.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
spp_connect	Structure pointer holding the response payload for the spp connection event This is the structure pointer for rsi_bt_event_spp_connect_t structure

Structure Prototype

```
typedef struct rsi_bt_event_spp_connect_s
{
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
   uint16_t  rem_mtu_size;
} rsi_bt_event_spp_connect_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
rem_mtu_size	MTU size supported by the remote device

12.7.3 rsi_bt_on_spp_disconnect_t

Prototype

```
typedef void (*rsi_bt_on_spp_disconnect_t) (uint16_t resp_status, rsi_bt_event_spp_disconnect_t
*spp_disconnect);
```

Description

This callback is called when SPP disconnected event is raised from the module. This event will be given by the module when spp profile level disconnection happens from either side.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
spp_disconnect	Structure pointer holding the response payload for the spp disconnection event This is the structure pointer for rsi_bt_event_spp_disconnect_t structure

Structure Prototype

```
typedef struct rsi_bt_event_spp_disconnect_s
{
   uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_spp_disconnect_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

12.7.4 rsi_bt_on_spp_rx_data_t

Prototype

```
typedef void (*rsi_bt_on_spp_rx_data_t) (uint16_t resp_status, rsi_bt_event_spp_receive_t
*bt_event_spp_receive);
```

Description

This callback is called when SPP receive event is raised from the module. This event will be given by the local device when it receives data from the remote device.

Precondition

None

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
bt_event_spp_receive	Structure pointer holding the response payload for the spp data receive event This is the structure pointer for rsi_bt_event_spp_receive_t structure

Structure Prototype

```
typedef struct rsi_bt_event_spp_receive_s
{
   uint16_t data_len;
   uint8_t  data[RSI_BT_MAX_PAYLOAD_SIZE];
} rsi_bt_event_spp_receive_t;
```

Structure Variables

Variables	Description
data_len	Length of the data received
data	Buffer holding the received SPP data. It holds the max payload length of 1k bytes

12.7.5 rsi_bt_change_pkt_type

Prototype

```
int32_t rsi_bt_change_pkt_type(uint8_t *remote_dev_addr, uint16_t pkt_type)
```

Description

This function is used to change the packet types in the controller after connection only.

Precondition

None

Parameters

Parameter	Description
remote_dev_addr	Remote device address
pkt_type	change the packet types in the controller

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status =  rsi_bt_change_pkt_type(uint8_t *remote_dev_addr, uint16_t pkt_type)
```

12.8 HCI packet change callbacks

This function registers the HCI callbacks.

12.8.1 rsi_bt_pkt_change_events_register_callbacks

Prototype

```
void rsi_bt_pkt_change_events_register_callbacks (
    rsi_bt_pkt_change_stats_t                bt_pkt_change_event
    )
```

Description

This function registers the HCI callbacks.

Parameters

Variables	Description
bt_pkt_change_event	This a callback function and will be called when ever packet change event received from the module.

12.8.2 rsi_bt_pkt_change_stats_t

Prototype

```
typedef void (*rsi_bt_pkt_change_stats_t) (uint16_t resp_status, rsi_bt_event_pkt_change_t
*bt_pkt_change_stats);
```

Description

This callback is called when HCI packet change event is raised from the module.

Precondition

None. (Default this will be enable in the module)

Parameters

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
bt_pkt_change_stats	Structure pointer holding the response payload for the HCI Pkt Change event .This is the structure pointer for rsi_bt_event_pkt_change_t structure.

Structure Prototype

```
typedef struct rsi_bt_event_pkt_change_s
{
   uint8_t  bd_addr[6];
```

```
    uint16_t pkt_type;
} rsi_bt_event_pkt_change_t;
```

Structure Variables

Variables	Description
bd_addr	This array holds the remote address of length 6 bytes
pkt_type	This is a packet_type bitmap.

Packet Bitmap

Value	Description
0x0001	Reserved for future use
0x0002	2-DH1 should not be used.
0x0004	3-DH1 should not be used
0x0008	DM1 may be used.
0x0010	DH1 may be used.
0x0020	Reserved for future use.
0x0040	Reserved for future use.
0x0080	Reserved for future use.
0x0100	2-DH3 should not be used.
0x0200	3-DH3 should not be used.
0x0400	DM3 may be used.
0x0800	DH3 may be used.
0x1000	2-DH5 should not be used.
0x2000	3-DH5 should not be used.
0x4000	DM5 may be used.
0x8000	DH5 may be used.

12.9 Configuration Parameters

This section explains the configuration of macros which may be needed to be changed based on the application requirement. These macros with default values are placed in "**rsi_bt_config.h**".

Define	Meaning
CONFIG_MAC_ADDRESS	This is used to set the MAC address. Size of MAC address is 6 bytes. If all 6 bytes are 0's, then it will be ignored.
RSI_OPERMODE_WLAN_BT_CLASSIC	This is a BT Classic Opermode. Opermode can be 5, 9, etc.
RSI_BT_REMOTE_BD_ADDR	This is a remote BD address to which EVK connects.
RSI_BT_LOCAL_NAME	This is used to set the local device name
PIN_CODE	Four or six digit pin code used for BT Classic (PHY level) authentication process
BT_BDR_MODE	Basic Data Rate, chain selection. 0 - Disabled, 1 - HP chain, 2 - LP chain

Define	Meaning
RSI_RF_TYPE	This is used to select the RF type. 0 - Internal RF, 1 - External RF

13 Bluetooth Low Energy APIs

This section contains a description of BLE APIs to initialize and configure the module in BLE mode.

Note:

A new BLE API must be called only after getting the response for the previous API.

13.1 Test Mode (RF Regulatory Mode) API

This section describes the APIs that are being used for testing purposes.

13.1.1 rsi_ble_rx_test_mode

Prototype

```
int32_t rsi_ble_rx_test_mode (uint8_t rx_channel, uint8_t phy, uint8_t modulation);
```

Description

This API is used to start the ble rx test mode in controller.

Parameters

Variables	Description
rx_channel	Channel in which packet have to be received (0 - 39)
phy	0x00 → Reserved for future use 0x01 → Receiver set to use the LE 1M PHY 0x02 → Receiver set to use the LE 2M PHY 0x03 → Receiver set to use the LE Coded PHY (0x04 – 0xFF) → Reserved for future use.
modulation	0x00 → Assume transmitter will have a standard standard modulation index 0x01 → Assume transmitter will have a stable modulation index (0x02 – 0xFF) → Reserved for future use

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define RSI_BLE_RX_CHANNEL              0x10
#define RSI_BLE_1MBPS                   0x1
#define RSI_BLE_2MBPS                   0x2
#define RSI_BLE_CODED_PHY               0x3
#define RSI_BLE_RX_PHY                  RSI_BLE_1MBPS

int32_t status = 0;
status = rsi_ble_rx_test_mode ( RSI_BLE_RX_CHANNEL,    /* Channel Number */
                        RSI_BLE_RX_PHY,     /* Phy 1Mbps Selected */
                        0x00);              /* standard modulation */
```

13.1.2 rsi_ble_tx_test_mode

Prototype

```
int32_t rsi_ble_tx_test_mode (uint8_t tx_channel, uint8_t phy, uint8_t tx_len, uint8_t mode);
```

Description

This API is used to start the ble tx test mode in controller.

Parameters

Variables	Description
tx_channel	RF Channel (0-39).
Phy	0x00 → Reserved for future use 0x01 → Transmitter set to use the LE 1M PHY 0x02 → Transmitter set to use the LE 2M PHY 0x03 → Transmitter set to use the LE Coded PHY with S=8 data coding 0x04 → Transmitter set to use the LE Coded PHY with S=2 data coding (0x05 – 0xFF) → Reserved for future use.
tx_len	Length in bytes of payload data in each packet (1 - 251 bytes).
Mode	0x00 → PRBS9 sequence ‘11111111100000111101... 0x01 → Repeated ‘11110000’ 0x02 → Repeated ‘10101010’ 0x03 → PRBS15 0x04 → Repeated ‘11111111’ 0x05 → Repeated ‘00000000’

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define RSI_BLE_TX_CHANNEL              0x10
#define RSI_BLE_1MBPS                   0x1
#define RSI_BLE_2MBPS                   0x2
#define RSI_BLE_TX_PHY                  RSI_BLE_1MBPS

#define RSI_BLE_TX_PAYLOAD_LEN          0x20
#define RSI_BLE_TX_PAYLOAD_TYPE         0x00


int32_t status = 0;
status = rsi_ble_tx_test_mode ( RSI_BLE_TX_CHANNEL,       /* channel number*/
                          RSI_BLE_TX_PHY,         /* phy - 1Mbps selected */
                        RSI_BLE_TX_PAYLOAD_LEN,       /* data_length */
                        RSI_BLE_TX_PAYLOAD_TYPE); /* packet payload sequence */
```

13.1.3 rsi_ble_end_test_mode

Prototype

```
int32_t rsi_ble_end_test_mode (uint16_t *num_of_pkts);
```

Description

This API is used to stop the ble tx / rx test mode in controller.

Parameters

Variables	Description
num_of_pkts	No.of rx packets received are displayed

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_ble_end_test_mode (100);
```

13.1.4 rsi_ble_per_transmit

Prototype

```
int32_t rsi_ble_per_transmit (struct rsi_ble_per_transmit_s *rsi_ble_per_tx);
```

Description

This API initiates the ble transmit PER mode in the controller.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
rsi_ble_per_tx	This parameter is the buffer to hold the structure values This is a structure variable of struct rsi_ble_per_transmit_s

rsi_ble_per_transmit_t

Structure Prototype

```
typedef struct rsi_ble_per_transmit_s {
   uint8_t cmd_ix;
   uint8_t transmit_enable;
   uint8_t access_addr[4];
   uint8_t phy_rate;
   uint8_t rx_chnl_num;
   uint8_t tx_chnl_num;
   uint8_t scrambler_seed;
   uint8_t le_chnl_type;
   uint8_t freq_hop_en;
   uint8_t ant_sel;
   uint8_t pll_mode;
   uint8_t rf_type;
   uint8_t rf_chain;
   uint8_t pkt_len[2];
   uint8_t payload_type;
   uint8_t tx_power;
   uint8_t transmit_mode;
   uint8_t inter_pkt_gap;
   uint8_t num_pkts[4];
} rsi_ble_per_transmit_t;
```

Structure Description

This structure contains the parameters required to per transmission.

Structure Variables

Variables	Description
cmd_ix	This parameter takes per BLE_TRANSMIT_CMD_ID of value 0x13
transmit_enable	This parameter enables/disables the ble per transmit mode 1 → PER Transmit Enable 0 → PER Transmit Disable
access_addr	This is the access address with which packets are transmitted
phy_rate	This is the Phy rate at which packets are transmitted 1 → 1Mbps 2 → 2 Mbps 4 → 125 Kbps Coded 8 → 500 Kbps Coded
rx_chnl_num	Rx channel number (0 - 39)
tx_chnl_num	Tx channel number (0 - 39)
scrambler_seed	Initial seed to be used for whitening. It should be set to '0' in order to disable whitening. In order to enable, one should give the scrambler seed value which is used on the receive side
le_chnl_type	This is the LE channel type (data or advertise channel) 0x00 → Advertise Channel 0x01 → Data Channel (to be used by Default)
freq_hop_en	This field defines the frequency hopping type to be used 0 → No Hopping 1 → Fixed Hopping 2 → Random Hopping (rx_chnl_num, tx_chnl_num parameters are unused in this mode)
ant_sel	This field defines the antenna selection (onboard/external) to be used for reception 2 → ONBOARD_ANT_SEL 3 → EXT_ANT_SEL
pll_mode	This field define the pll_mode type to be used 0 → PLL_MODE0 (to be used by Default) 1 → PLL_MODE1
rf_type	This field defines the selection of RF type (internal/external) 0 → BT_EXTERNAL_RF 1 → BT_INTERNAL_RF (to be used by Default) **Note**: The above macros are applicable for both BT and BLE
rf_chain	This field corresponds to the selection of RF chain (HP/LP) to be used 2 → BT_HP_CHAIN 3 → BT_LP_CHAIN **Note**: The above macros are applicable for both BT and BLE
pkt_len	length of the packet to be transmitted
payload_type	Type of payload data sequence

Variables	Description
	0x00 → PRBS9 sequence '11111111100000111101... 0x01 → Repeated '11110000' 0x02 → Repeated '10101010' 0x03 → PRBS15 0x04 → Repeated '11111111' 0x05 → Repeated '00000000' 0x06 → Repeated '00001111' 0x07 → Repeated '01010101'
tx_power	This field corresponds to the transmit power • Transmit power value for the rf_chain parameter set to the HP chain the values for the tx power indexes are 0 - 20. • Transmit power value for the rf chain parameter set to LP chain and values are: • 0 -31 o/p power equation is -2+10log10(power_index/31) • 32-63 o/p power equation is 6 + 10log10((power_index -32)/31) • TX power for the BLE LP Chain :1 to 31 (0dBm Mode), 33 to 63 (10dBm Mode) • TX power for the BLE HP chain : 64 to 79
transmit_mode	This field corresponds to the transmit mode to be used either Burst/Continuous 0 → BURST_MODE 1 → CONTINUOUS_MODE
inter_pkt_gap	This field takes the value of inter packet gap. Number of slots to be skipped between two packets - Each slot will be 1250usec
num_pkts	This field defines the number of packets to be transmitted, default to zero for continuous transmission

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure If return value is lesser than 0 -4 : Buffer not available to serve the command

Example

```
rsi_ble_per_transmit_t rsi_ble_per_tx;
int32_t status = 0;

/* Example rsi_ble_per_tx parameters */
rsi_ble_per_tx.cmd_ix = BLE_TRANSMIT_CMD_ID;
rsi_ble_per_tx.transmit_enable = ENABLE;
/*.................*/

status = rsi_ble_per_transmit(&rsi_ble_per_tx);
```

13.1.5 rsi_ble_per_receive

Prototype

```
int32_t rsi_ble_per_receive (struct rsi_ble_per_receive_s *rsi_ble_per_rx);
```

Description

This API initiates the BLE receive PER mode in the controller.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
rsi_ble_per_rx	This parameter is the buffer to hold the structure values This is a structure variable of struct rsi_ble_per_receive_s

rsi_ble_per_receive_t

Structure Prototype

```
typedef struct rsi_ble_per_receive_s {
   uint8_t cmd_ix;
   uint8_t receive_enable;
   uint8_t access_addr[4];
   uint8_t phy_rate;
   uint8_t rx_chnl_num;
   uint8_t tx_chnl_num;
   uint8_t scrambler_seed;
   uint8_t le_chnl_type;
   uint8_t freq_hop_en;
   uint8_t ant_sel;
   uint8_t pll_mode;
   uint8_t rf_type;
   uint8_t rf_chain;
   uint8_t ext_data_len_indication;
   uint8_t loop_back_mode;
   uint8_t duty_cycling_en;
} rsi_ble_per_receive_t;
```

Description

This structure contains the parameters required to per receive.

Structure Variables

Variables	Description
cmd_ix	This parameter takes per BLE_RECEIVE_CMD_ID of value 0x14
receive_enable	This parameter enables/disables the ble per receive mode 1 → PER Receive Enable 0 → PER Receive Disable
access_addr	This is the access address with which packets are transmitted
phy_rate	This is the Phy rate at which packets are received 1 → 1Mbps 2 → 2 Mbps 4 → 125 Kbps Coded 8 → 500 Kbps Coded
rx_chnl_num	Rx channel number (0 - 39)
tx_chnl_num	Tx channel number (0 - 39)
scrambler_seed	Initial seed to be used for whitening. It should be set to ‘0’ in order to disable whitening.

Variables	Description
	In order to enable, one should give the scrambler seed value which is used on the transmit side
le_chnl_type	This is the LE channel type (data or advertise channel) 0x00 → Advertise Channel 0x01 → Data Channel (to be used by Default)
freq_hop_en	This field defines the frequency hopping type to be used 0 → No Hopping 1 → Fixed Hopping 2 → Random Hopping (rx_chnl_num, tx_chnl_num parameters are unused in this mode)
ant_sel	This field defines the antenna selection (onboard/external) to be used for reception 2 → ONBOARD_ANT_SEL 3 → EXT_ANT_SEL
pll_mode	This field define the pll_mode type to be used 0 → PLL_MODE0 (to be used by Default) 1 → PLL_MODE1
rf_type	This field defines the selection of RF type (internal/external) 0 → BT_EXTERNAL_RF 1 → BT_INTERNAL_RF (to be used by Default)
rf_chain	This field corresponds to the selection of RF chain (HP/LP) to be used 2 → BT_HP_CHAIN 3 → BT_LP_CHAIN
ext_data_len_indication	This field enables/disables the extended data length 0 → Extended Data length disabled 1 → Extended Data length enabled
loop_back_mode	This field defines the loopback to be enable or disable 0 → LOOP_BACK_MODE_DISABLE 1 → LOOP_BACK_MODE_ENABLE
duty_cycling_en	This field enables/disables the duty cycling 0 → Duty Cycling Disabled (to be used by Default) 1 → Duty Cycling Enabled

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure If return value is lesser than 0 -4 : Buffer not available to serve the command

Example

```
rsi_ble_per_receive_t rsi_ble_per_rx;
int32_t status = 0;
```

```
/* Example rsi_ble_per_rx parameters */
rsi_ble_per_rx.cmd_ix = BLE_RECEIVE_CMD_ID;
rsi_ble_per_rx.receive_enable = ENABLE;
/*.................*/

status = rsi_ble_per_receive (&rsi_ble_per_rx);
```

13.2 GAP API

This section describes the GAP APIs.

13.2.1 rsi_ble_set_random_address

Prototype

```
int32_t rsi_ble_set_random_address(void);
```

Description

This API request the local device to set a random address.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure If return value is lesser than 0 -4 : Buffer not available to serve the command

Example

```
int32_t status = 0;
status = rsi_ble_set_random_address();
```

13.2.2 rsi_ble_set_random_address_with_value

Prototype

```
int32_t rsi_ble_set_random_address_with_value(uint8_t *random_addr);
```

Description

This API request the local device to set a given random address.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
random_addr	random address of the device to be set

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure If return value is lesser than 0 -4 : Buffer not available to serve the command

Example

```
int32_t status = 0;
status = rsi_ble_set_random_address_with_value(random_addr);
```

13.2.3 rsi_ble_update_directed_address

Prototype

```
void rsi_ble_update_directed_address(int8_t *remote_dev_addr);
```

Description

This API updates the direct address with remote device address.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr	remote device bd address

Return Values

None

Example

```
rsi_ble_update_directed_address(remote_dev_bd_addr);
```

13.2.4 rsi_ble_start_advertising

Prototype

```
int32_t rsi_ble_start_advertising(void);
```

Description

This API requests the local device to start advertising.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_start_advertising();
```

13.2.5 rsi_ble_start_advertising_with_values

Prototype

```
int32_t rsi_ble_start_advertising_with_values(void *rsi_ble_adv);
```

Description

This API request the local device to start advertising with specified values.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
rsi_ble_adv	This structure pointer holds the information of advertising values This variable is pointer of rsi_ble_req_adv_s structure

rsi_ble_req_adv_t

Structure Prototype

```
typedef struct rsi_ble_req_adv_s
{
   uint8_t status;
   uint8_t adv_type;
   uint8_t filter_type;
   uint8_t direct_addr_type;
   uint8_t direct_addr[RSI_DEV_ADDR_LEN];
   uint16_t adv_int_min;
   uint16_t adv_int_max;
   uint8_t own_addr_type;
   uint8_t adv_channel_map;
} rsi_ble_req_adv_t;
```

Structure Description

This structure defines the format for the advertising values.

Structure Variables

Variables	Description
status	Advertising status - disable/enable 0 - disable, 1 - enable
adv_type	Advertising type used during advertising. 1. Advertising will be visible(discoverable) to all the devices. Scanning/Connection is also accepted from all devices. #define UNDIR_CONN 0x80 2. Advertising will be visible(discoverable) to the particular device mentioned in RSI_BLE_ADV_DIR_ADDR only. Scanning and Connection will be accepted from that device only. #define DIR_CONN 0x81 3. Advertising will be visible(discoverable) to all the devices. Scanning will be accepted from all the devices. Connection will be not be accepted from any device.

Variables	Description
	#define UNDIR_SCAN 0x82 4. Advertising will be visible(discoverable) to all the devices. Scanning and Connection will not be accepted from any device. #define UNDIR_NON_CONN 0x83 5. Advertising will be visible(discoverable) to the particular device mentioned in RSI_BLE_ADV_DIR_ADDR only. Scanning and Connection will be accepted from that device only. #define DIR_CONN_LOW_DUTY_CYCLE 0x84
filter_type	advertising filter type. 1. #define ALLOW_SCAN_REQ_ANY_CONN_REQ_ANY 0x00 2. #define ALLOW_SCAN_REQ_WHITE_LIST_CONN_REQ_ANY 0x01 3. #define ALLOW_SCAN_REQ_ANY_CONN_REQ_WHITE_LIST 0x02 4. #define ALLOW_SCAN_REQ_WHITE_LIST_CONN_REQ_WHITE_LIST 0x03
direct_addr_type	address type of the device to which directed advertising has to be done 1. #define LE_PUBLIC_ADDRESS 0x00 2. #define LE_RANDOM_ADDRESS 0x01 3. #define LE_RESOLVABLE_PUBLIC_ADDRESS 0x02 4. #define LE_RESOLVABLE_RANDOM_ADDRESS 0x03
direct_addr	address of the device to which directed advertising has to be done
adv_int_min	advertising interval min 0x0020 to 0x4000
adv_int_max	advertising interval max 0x0020 to 0x4000
own_addr_type	address of the local device. 1. #define LE_PUBLIC_ADDRESS 0x00 2. #define LE_RANDOM_ADDRESS 0x01 3. #define LE_RESOLVABLE_PUBLIC_ADDRESS 0x02 4. #define LE_RESOLVABLE_RANDOM_ADDRESS 0x03
adv_channel_map	advertising channel map. #define RSI_BLE_ADV_CHANNEL_MAP 0x01 or 0x03 or 0x07

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure If return value is lesser than 0 -4 : Buffer not available to serve the command

Example

```
int32_t status = 0;
rsi_ble_req_adv_t rsi_ble_adv;
status = rsi_ble_start_advertising_with_values(&rsi_ble_adv);
```

13.2.6 rsi_ble_encrypt

Prototype

```
int32_t rsi_ble_encrypt(uint8_t *key,uint8_t *data,uint8_t *resp);
```

Description

This API encrypts the data.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
Key	16 Bytes key for Encryption of data.
Data	16 Bytes of Data request to encrypt.
Resp	Encrypted data

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_encrypt( &key,&data, &resp);
```

13.2.7 rsi_ble_stop_advertising

Prototype

```
int32_t rsi_ble_stop_advertising(void)
```

Description

This API stops advertising.

Precondition

rsi_ble_start_advertising() API needs to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_stop_advertising();
```

13.2.8 rsi_ble_start_scanning

Prototype

```
int32_t rsi_ble_start_scanning(void)
```

Description
This API starts scanning.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_start_scanning(void);
```

13.2.9 rsi_ble_start_scanning_with_values

Prototype

```
int32_t rsi_ble_start_scanning_with_values(void *rsi_ble_scan_params)
```

Description
This API starts scanning.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameters	Description
status	scanning status - disable/enable 0 - disable, 1 - enable
scan_type	scan type - passive/active 1. #define SCAN_TYPE_ACTIVE 0x01 2. #define SCAN_TYPE_PASSIVE 0x00
filter_type	FILTERING_DISABLED = 0 (default) WHITELIST_FILTERING = 1 **Note**: In order to allow only whitelisted devices, need to add bd_addr into whitelist by calling rsi_ble_addto_whitelist() API
own_addr_type	address of the local device 1. #define LE_PUBLIC_ADDRESS 0x00 2. #define LE_RANDOM_ADDRESS 0x01 3. #define LE_RESOLVABLE_PUBLIC_ADDRESS 0x02 4. #define LE_RESOLVABLE_RANDOM_ADDRESS 0x03

Parameters	Description
scan_int	Scan interval This is defined as the time interval from when the Controller started its last LE scan until it begins the subsequent LE scan. Range: 0x0004 to 0x4000
scan_win	Scan window The duration of the LE scan. LE_Scan_Window shall be less than or equal to LE_Scan_Interval Range: 0x0004 to 0x4000

```
typedef struct rsi_ble_req_scan_s
{
 //!uint8, scanning status - disable/enable
 uint8_t status;
 //!uint8, scan type - passive/active
 uint8_t scan_type;
 //!uint8, to filter incoming advertising reports
 uint8_t filter_type;
//!uint8, address of the local device
 uint8_t own_addr_type;
 //!uint8, scan interval
 uint16_t scan_int;

 //!uint8, scan window
 uint16_t scan_win;

}rsi_ble_req_scan_t;
```

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_ble_start_scanning_with_values(void *rsi_ble_scan_params)
```

13.2.10 rsi_ble_stop_scanning

Prototype

```
int32_t rsi_ble_stop_scanning(void)
```

Description
This API stops scanning.

Precondition
rsi_ble_start_scanning() API needs to be called before this API.

Parameters
None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_stop_scanning();
```

13.2.11 rsi_ble_connect

Prototype

```
int32_t rsi_ble_connect(uint8_t remote_dev_addr_type,
 int8_t *remote_dev_addr)
```

Description
This API connects to the remote BLE device.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr_type	This parameter describes address type of remote device
remote_dev_addr	This parameter describes the device address of remote device

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_connect(RSI_BLE_DEV_ADDR_TYPE, RSI_BLE_DEV_ADDR);
```

13.2.12 rsi_ble_connect_with_params

Prototype

```
int32_t rsi_ble_connect_with_params(uint8_t remote_dev_addr_type, int8_t *remote_dev_addr, uint16_t
scan_interval, uint16_t scan_window, uint16_t conn_interval_max, uint16_t conn_interval_min, uint16_t
conn_latency, uint16_t supervision_tout);
```

Description
This API connects to the remote BLE device with the user configured parameters.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_addr_type	AddressType – Specifies the type of the address mentioned in BD Address 0 – Public Address 1 – Random Address
remote_dev_addr	This parameter describes the device address of remote device
scan_interval	LE Scan Interval : N=0xXXXX

Parameter	Description
	This is defined as the time interval from when the Controller started its last LE scan until it begins the subsequent LE scan. Range: 0x0004 to 0x4000 Time = N * 0.625 msec Time Range: 2.5 msec to 10 . 24 seconds
scan_window	LE Scan Window : N=0xXXXX Amount of time for the duration of the LE scan. LE_Scan_Window shall be less than or equal to LE_Scan_Interval Range: 0x0004 to 0x4000 Time = N * 0.625 msec Time Range: 2.5 msec to 10 . 24 seconds
conn_interval_max	Max Connection Interval : N=0xXXXX Minimum value for the connection event interval. This shall be greater than or equal to Conn_Interval_Min. Range: 0x0006 to 0x0C80 Time = N * 1.25 msec Time Range: 7.5 msec to 4 seconds. 0x0000 – 0x0005 and 0x0C81 – 0xFFFF - Reserved for future use
conn_interval_min	Min Connection Interval : N=0xXXXX Minimum value for the connection event interval. This shall be less than or equal to Conn_Interval_Max. Range: 0x0006 to 0x0C80 Time = N * 1.25 msec Time Range: 7.5 msec to 4 seconds 0x0000 – 0x0005 and 0x0C81 – 0xFFFF - Reserved for future use
conn_latency	Connection Latency : N = 0xXXXX Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F4
supervision_tout	Supervision Timeout : N = 0xXXXX Supervision timeout for the LE Link. Range: 0x000A to 0x0C80 Time = N * 10 msec Time Range: 100 msec to 32 seconds 0x0000 – 0x0009 and 0x0C81 – 0xFFFF - Reserved for future use

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_connect_with_params(RSI_BLE_DEV_ADDR_TYPE, RSI_BLE_DEV_ADDR, LE_SCAN_INTERVAL,
LE_SCAN_WINDOW, CONNECTION_INTERVAL_MIN, CONNECTION_INTERVAL_MAX, CONNECTION_LATENCY,
SUPERVISION_TOUT);
```

13.2.13 rsi_ble_connect_cancel

Prototype

```
int32_t rsi_ble_connect_cancel(int8_t *remote_dev_address)
```

Description
This API cancels the connection to the remote BLE device.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_address	This parameter describes the device address of the remote device

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_connect_cancel(RSI_BLE_DEV_ADDR);
```

13.2.14 rsi_ble_disconnect

Prototype

```
int32_t rsi_ble_disconnect(int8_t *remote_dev_address)
```

Description
This API disconnects with the remote BLE device.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_address	This parameter describes the device address of the remote device

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_disconnect(RSI_BLE_DEV_ADDR);
```

13.2.15 rsi_ble_get_device_state

Prototype

```
int32_t rsi_ble_get_device_state(uint8_t *resp)
```

Description
This API gets the local device state. The state value is filled in "resp".

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
resp	This is an output parameter which consists of local device state. This is a 1-byte value. The possible states are described in the below table

Bit filed	Description
BIT(0)	Advertising state
BIT(1)	Scanning state
BIT(2)	Initiating state
BIT(3)	Connected state
BIT(4)–BIT(7)	Reserved

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
static uint8_t rsi_app_resp_device_state = 0;
 status = rsi_ble_get_device_state(&rsi_app_resp_device_state);
```

13.2.16 rsi_ble_set_smp_pairing_cap_data

Prototype

```
int32_t rsi_ble_set_smp_pairing_cap_data (rsi_ble_set_smp_pairing_capabilty_data_t *smp_pair_cap_data);
```

Description
This API sets the SMP Pairing Capability of local device.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
smp_pair_cap_data	This structure pointer holds the information of smp capability data values

Parameter	Description
	This variable is pointer of rsi_ble_set_smp_pairing_capabilty_data_t structure

rsi_ble_set_smp_pairing_capabilty_data_t

Structure Prototype

```
typedef struct rsi_ble_set_smp_pairing_capabilty_data {
  //!uint8, device IO capabilities
  uint8_t io_capability;
  //!uint8, oob_data_flag (present or not)
  uint8_t oob_data_flag;
  //!uint8, auth_req contaions bonding type, mitm req, sc, keypress, CT2
  uint8_t auth_req;
  //!uint8, supported encryption key size 7 to 16 bytes
  uint8_t enc_key_size;
  //!uint8, initiator generates/requires the no .of keys after sucessful paring
  uint8_t ini_key_distribution;
  //!uint8, responder generates/responder the no .of keys after sucessful pairing
  uint8_t rsp_key_distribution;
} rsi_ble_set_smp_pairing_capabilty_data_t;
```

Structure Description

This structure defines the format for the smp pairing capability data.

Structure Variables

Variables	Description
io_capability	Device IO Capabilities
oob_data_flag	OOB data flag (Enabled/Disabled)
auth_req	Authentication Request contains bonding type, MITM Request, Secure Connections, Keypress, CT2
enc_key_size	Encryption Key Size (7 to 16 bytes)
ini_key_distribution	Initiator generates/requires the no of keys after successfull pairing
rsp_key_distribution	Responder generates/resonpds the no of keys after successfull pairing

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure If return value is lesser than 0 -4 : Buffer not available to serve the command

Example

```
int32_t status = 0;
rsi_ble_set_smp_pairing_capabilty_data_t smp_pair_cap_data;
status = rsi_ble_set_smp_pairing_cap_data(&smp_pair_cap_data);
```

13.2.17 rsi_ble_set_local_irk_value

Prototype

```
int32_t rsi_ble_set_local_irk_value (uint8_t *l_irk)
```

Description

setting irk value to the local device.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
l_irk	l_irk Pointer to local_irk

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_ble_set_local_irk_value (uint8_t *l_irk)
```

13.2.18 rsi_ble_set_advertise_data

Prototype

```
int32_t rsi_ble_set_advertise_data(uint8_t *data, uint16_t data_len)
```

Description
This API sets the advertising data.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
data	This is the advertise data.
data_len	This is the total length of advertising data

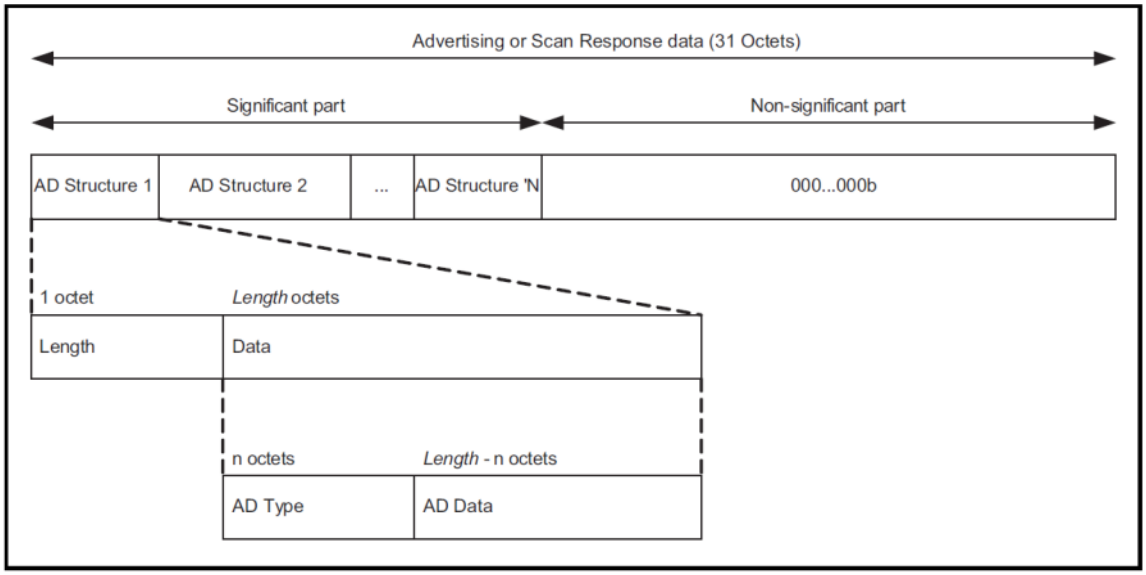
Note:

1. The maximum length of advertising data payload is 31 bytes.
2. The basic format of advertise payload record contains length and data as below:

1 octet 1 octet (length 1) octets

Length	AD type	AD data

The details regarding AD type field is specified in Volume 3, Part C, Chapter 18, Appendix C in Bluetooth 4.0 specification.



For more information on advertising data with types, please go through the following link:
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
uint8_t adv[31] = {2,1,6};
 //! prepare advertise data
 adv[3] = strlen (RSI_BLE_LOCAL_NAME) + 1;
 adv[4] = 9;
 strcpy (&adv[5], RSI_BLE_LOCAL_NAME);
 //! set advertise data
 rsi_ble_set_advertise_data (adv, strlen (RSI_BLE_LOCAL_NAME) + 5);
```

13.2.19 rsi_ble_smp_pair_request

Prototype

```
int32_t rsi_ble_smp_pair_request (uint8_t *remote_dev_address, uint8_t io_capability, mitm_req)
```

Description
This API requests the SMP pairing process with the remote device.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_address	This is the remote device address
io_capability	This is the device input output capability 0x00 - Display Only

Parameter	Description
	0x01 - Display Yes/No 0x02 - Keyboard Only 0x03 - No Input No Output
mitm_req	MITM enable/disable 0 - Disable 1 - Enable

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_smp_pair_request (str_remote_address, RSI_BLE_SMP_IO_CAPABILITY, MITM_REQ);
```

13.2.20 rsi_ble_smp_pair_response

Prototype

```
int32_t rsi_ble_smp_pair_response(uint8_t *remote_dev_address, uint8_t io_capability, uint8_t
mitm_req);
```

Description
This API sends SMP pairing response during the process of pairing with the remote device.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_address	This is the remote device address
io_capability	This is the device input output capability 0x00 - Display Only 0x01 - Display Yes/No 0x02 - Keyboard Only 0x03 - No Input No Output
mitm_req	MITM Request info 0 - Disable 1 - Enable

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Value	Description
	If return value is lesser than 0 -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_smp_pair_response (str_remote_address, RSI_BLE_SMP_IO_CAPABILITY, MITM_REQ);
```

13.2.21 rsi_ble_smp_passkey

Prototype

```
int32_t rsi_ble_smp_passkey (uint8_t *remote_dev_address, uint32_t passkey)
```

Description
This API is sends SMP passkey during SMP pairing process with the remote device.

Precondition
rsi_ble_smp_pair_request and rsi_ble_smp_pair_response API need to be called before this API.

Parameters

Parameter	Description
remote_dev_address	This is the remote device address
passkey	This is the key required in pairing process

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_smp_passkey (str_remote_address, RSI_BLE_SMP_PASSKEY);
```

13.2.22 rsi_ble_get_le_ping_timeout

Note:

Currently Get ping is not supported.

Prototype

```
int32_t rsi_ble_get_le_ping_timeout(uint8_t *remote_dev_address, uint16_t *time_out);
```

Description
This API gets the timeout value of the LE ping.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_address	This is the remote device address
time_out	This a response parameter which holds timeout value for authentication payload command.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_le_ping_timeout (remote_dev_address, &time_out);
```

13.2.23 rsi_ble_set_le_ping_timeout

Prototype

```
int32_t rsi_ble_set_le_ping_timeout(uint8_t *remote_dev_address uint16_t time_out)
```

Description
This API sets the timeout value of the LE ping.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_address	This is the remote device address
time_out	This input parameter sets timeout value for authentication payload command.(in milli seconds)

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_set_le_ping_timeout (remote_dev_address,  time_out);
```

13.2.24 rsi_ble_set_scan_response_data

Prototype

```
int32_t rsi_ble_set_scan_response_data(uint8_t *data, uint16_t data_len);
```

Description

This API request the local device to set the scan response data.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
data	data to be sent is filled in this
data_len	length of data to be sent

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_set_scan_response_data (adv_arr, strlen (adv));
```

13.2.25 rsi_ble_clear_whitelist

Prototype

```
int32_t rsi_ble_clear_whitelist(void);
```

Description

This API clears all the BD address present in white list.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_clear_whitelist();
```

13.2.26 rsi_ble_addto_whitelist

Prototype

```
int32_t rsi_ble_addto_whitelist( int8_t *dev_address,uint8_t dev_addr_type);
```

Description

This API Adds BD address to white list.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
dev_address	Address of the device which is going to add in white list
dev_addr_type	address type of bd address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_addto_whitelist(RSI_BLE_WHITELIST_DEV_ADDR1,RSI_BLE_WHITELIST_DEV_ADDR1_TYPE);
```

13.2.27 rsi_ble_deletefrom_whitelist

Prototype

```
int32_t rsi_ble_deletefrom_whitelist(int8_t *dev_address,uint8_t dev_addr_type)
```

Description

This API deletes particular BD address from white list.

Precondition

rsi_ble_addto_whitelist() API needs to be called before this API.

Parameters

Parameter	Description
dev_address	Address of the device which is going to delete from white list
dev_addr_type	address type of bd address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_deletefrom_whitelist(RSI_BLE_WHITELIST_DEV_ADDR2,RSI_BLE_WHITELIST_DEV_ADDR2_TYPE);
```

13.2.28 rsi_ble_vendor_rf_type

Prototype

```
int32_t rsi_ble_vendor_rf_type (uint8_t perf_mode,uint8_t duty_cycling);
```

Description

This API gives vendor specific command to the controller to set rf type.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
perf_mode	performance mode
duty_cycling	duty cycling

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_vendor_rf_type (perf_mode, duty_cycling
```

13.2.29 rsi_ble_white_list_using_adv_data

Prototype

```
int32_t rsi_ble_white_list_using_adv_data (uint8_t enable, uint8_t data_compare_index, uint8_t
len_for_compare_data, uint8_t *payload);
```

Description

This enable gives vendor specific command to set the whitelist feature based on the advertisers advertising payload.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
enable	enable/disable
data_compare_index	From which index onwards compare
len_for_compare_data	total length of data to compare
payload	Payload

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_white_list_using_adv_data (enable, data_compare_index, len_for_compdata, payload);
```

13.2.30 rsi_ble_start_encryption

Prototype

```
int32_t rsi_ble_start_encryption (uint8_t *remote_dev_address, uint16_t ediv, uint8_t *rand, uint8_t
*ltk);
```

Description

This API start the encryption process with remote device.

Precondition

Encryption enabled event should come before calling this api for second time smp connection.

Parameters

Parameter	Description
remote_dev_address	remote bd address in string format
Ediv	remote device ediv value
Rand	remote device rand value
Ltk	remote device ltk value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_start_encryption(remote_dev_address,ediv,rand_arr,ltk_arr);
```

13.3 Basic Structure defines

The below structure defines explains in detail the structure variables with description. These Structure defines are used in some of the APIs in this document.

13.3.1 uuid_t structure

```
typedef struct bt_uuid128 {
 UINT08 data1[4];
 UINT08 data2[2];
 UINT08 data3[2];
 UINT08 data4[8];
 } UUID128;
 typedef UINT16 UUID16;
 typedef UINT32 UUID32;
 /* Main UUID structure */
 typedef struct bt_uuid {
 UINT08 size; // Size of the UUID
 UINT08 reserved[3];
 union bt_uuid_t {
 UUID128 val128;
```

```
UUID32 val32;
UUID16 val16;
} val; // UUID value
} uuid_t;
```

Description

The size of a UUID (Universal Unique Identifier) can be 2byte (16bit), 4byte (32bit) or 16byte (128bit). This structure defines the size of uuid and uuid value.

Structure Variables

Variables	Description
size	This is the size of uuid
reserved	Reserved
val	This is the value of one of the 3 types (128 bit, 32 bit or 16 bit) of UUIDs.

13.3.2 inc_service_data_t

structure

```
typedef struct inc_serv_data_s
 {
 uint16_t start_handle;
 uint16_t end_handle;
 uuid_t uuid;
 } inc_serv_data_t;
```

Description

This structure describes the format of the include service attribute data.

Structure Variables

Variables	Description
start_handle	This include service start handle
end_handle	This include service end handle
Uuid	This is the UUID value of the included service.

13.3.3 inc_service_t

structure

```
typedef struct inc_serv_s
 {
 Uint16_t handle;
 uint8_t reserved[2];
 inc_serv_data_t inc_serv;
 } inc_serv_t;
```

Description

This structure defines the included service attribute record.

Structure Variables

Variables	Description
handle	This include service defined handle
reserved	Reserved
inc_service	This include service attribute data structure.

13.3.4 char_serv_data_t

structure

```
typedef struct char_serv_data_s
 {
 uint8_t char_property;
 uint8_t reserved;
 uint16_t char_handle;
 uuid_t char_uuid;
 } char_serv_data_t;
```

Description

This structure defines the service characteristic data format.

Structure Variables

Variables	Description
char_property	This is the characteristic value property.
reserved	Reserved
char_handle	This is the characteristic value handle
char_uuid	This is the characteristic value attributes uuid.

13.3.5 char_serv_t

structure

```
typedef struct char_serv_s
 {
 uint16_t handle;
 uint8_t reserved[2];
 char_serv_data_t char_data;
 } char_serv_t;
```

Description

This structure defines the service characteristic attribute record format.

Structure Variables

Variables	Description
handle	This is the characteristic service attribute handle.
reserved	Reserved
char_data	This is the characteristic data structure variable

13.3.6 att_desc_t

Structure Prototype

```
typedef struct att_desc_s
 {
 uint8_t handle[2];
 uint8_t reserved[2];
 uuid_t att_type_uuid;
 } att_desc_t;
```

Description

This structure defines attribute or characteristic descriptors format.

Structure Variables

Variables	Description
handle	This is the attribute handle
reserved	Reserved
att_type_uuid	This is the attribute uuid (attribute type) .

13.3.7 rsi_ble_resp_profiles_list_t

Structure

```
typedef struct rsi_ble_resp_profiles_list_s
 {
 uint8_t number_of_profiles;
 uint8_t reserved[3];
 profile_descriptors_t profile_desc[RSI_BLE_MAX_RESP_LIST];
 } rsi_ble_resp_profiles_list_t;
```

Description
This structure defines GATT profiles list response structure.

Structure Variables

Variables	Description
number_of_profiles	This is the number of profiles found
reserved	Reserved
profile_desc	This is the list of found profiles The maximum value is 5.

13.3.8 rsi_ble_resp_char_services_t

Structure

```
typedef struct rsi_ble_resp_char_serv_s
 {
 uint8_t num_of_services;
 uint8_t reserved[3];
 char_serv_t char_services[RSI_BLE_MAX_RESP_LIST];
 } rsi_ble_resp_char_services_t;
```

Description
This is a GATT characteristic query service response structure.

Structure Variables

Variables	Description
num_of_services	This is the number of profiles found
reserved	Reserved
char_services	This is the characteristic service array. The maximum value is 5.

13.3.9 rsi_ble_resp_inc_services_t

Structure

```
typedef struct rsi_ble_resp_inc_serv
 {
 uint8_t num_of_services;
 uint8_t reserved[3];
 inc_serv_t services[RSI_BLE_MAX_RESP_LIST];
 } rsi_ble_resp_inc_services_t;
```

Description
This is a GATT included service response structure.

Structure Variables

Variables	Description
num_of_services	This is the number of profiles found
reserved	Reserved

Variables	Description
services	This include service list. The maximum value is 5.

13.3.10 rsi_ble_resp_att_value_t

Structure

```
typedef struct rsi_ble_resp_att_value_s
 {
 uint8_t len;
 uint8_t att_value[100];
 } rsi_ble_resp_att_value_t;
```

Description
This is a GATT attribute value response structure.

Structure Variables

Variables	Description
len	This is the length of the attribute value. The maximum length is 30.
att_value	This is the attribute value.

13.3.11 rsi_ble_resp_att_descs_t

Structure

```
typedef struct rsi_ble_resp_att_descs_s
 {
 uint8_t num_of_att;
 uint8_t reserved[3];
 att_desc_t att_desc[RSI_BLE_MAX_RESP_LIST];
 } rsi_ble_resp_att_descs_t;
```

Description
This is a GATT attribute descriptors response structure.

Structure Variables

Variables	Description
num_of_att	This is the number of descriptors found
reserved	Reserved
att_desc	This is the attribute descriptors list. The maximum value is 5.

13.3.12 rsi_ble_req_add_att_t

Structure

```
typedef struct rsi_ble_req_add_att_s
 {
 void *serv_handler;
 uint16_t handle;
 uint16_t reserved;
 uuid_t att_uuid;
 uint8_t property;
 uint8_t data[31];
 uint16_t data_len;
 } rsi_ble_req_add_att_t;
```

Description
This structure generally adds new attributes to a service record.

Structure Variables

Variables	Description
serv_handler	This is the service handler
handle	This is the handle
reserved	If this variable is 1, Host has to maintain attributes and records in the application. If 0, Stack will maintain the attributes and records
att_uuid	This is the attribute type UUID
property	This is the attribute property
data	This is the attribute data. The maximum value is 31
data_len	This is the attribute data length

Note:

If application want to send more than 20 bytes of data, it has to handle attributes and records by setting **reserved** variable as 1.

If reserved variable is 0, then stack will store the data upto 20 bytes.

13.3.13 rsi_ble_resp_local_att_value_t

Structure

```
typedef struct rsi_ble_resp_local_att_value_s
 {
 uint16_t handle;
 uint16_t data_len;
 uint8_t data[31];
 } rsi_ble_resp_local_att_value_t;
```

Description
This is a read local attribute value response structure.

Structure Variables

Variables	Description
Handle	This is the attribute handle
data_len	This is the attribute value length
Data	This is the attribute value (data). The maximum value is 31.

13.3.14 rsi_bt_event_le_ltk_request_s

Structure

```
ttypedef struct rsi_bt_event_le_ltk_request_s {
  //!uint8_t, address of the remote device encrypted
  uint8_t   dev_addr[RSI_DEV_ADDR_LEN];
  uint16_t  localediv;
  uint8_t   localrand[8];
} rsi_bt_event_le_ltk_request_t;
```

Description
This is a read local attribute value response structure.

Structure Variables

Variables	Description
Handle	This is the attribute handle
data_len	This is the attribute value length
Data	This is the attribute value (data). The maximum value is 31.

13.3.15 rsi_ble_set_le_ltkreqreply

Structure

```
typedef struct rsi_ble_set_le_ltkreqreply_s{
  uint8_t dev_addr[RSI_DEV_ADDR_LEN];
    uint8_t    replytype;
    uint8_t    localltk[16];
}rsi_ble_set_le_ltkreqreply_t;
```

Description

This function is used to start the SMP pairing process.

Structure Variables

Variables	Description
dev_addr	Remote Device Address
replytype	positive or negative reply
localltk	local ltk value

13.4 Credit based flow APIs

13.4.1 rsi_ble_cbfc_connreq

Prototype

```
uint32_t rsi_ble_cbfc_connreq (uint8_t *dev_addr, uint16_t psm);
```

Description

This command is used to initiate new PSM connection with remote device.

Structure Variables

Variables	Description
dev_addr	Remote Device address
psm	Protocol/Service Multiplexer (PSM) This field helps to indicate the protocol.

Return values

Value	Description
0	command succeded
Non Zero	Error (Refer Error codes)

Example

```
RSI_BLE_PSM_VALUE = 0x23;

rsi_ble_cbfc_connreq (remote_dev_addr, RSI_BLE_PSM_VALUE);
```

13.4.2 rsi_ble_cbfc_connresp

Prototype

```
uint32_t rsi_ble_cbfc_connresp (uint8_t *dev_addr, uint16_t lcid, uint8_t  result);
```

Description

This command is used to Respond to a new PSM connection with remote device.

Structure Variables

Variables	Description
dev_addr	Remote Device address
lcid	local channel id, received from CBFC conn request event
result	accept/reject the connection request.

Return values

Value	Description
0	command succeeded
Non Zero	Error (Refer Error codes)

Example

```
result = 1 (for accept)
rsi_ble_cbfc_connresp (remote_dev_addr, lcid, result);
```

13.4.3 rsi_ble_cbfc_data_tx

Prototype

```
uint32_t rsi_ble_cbfc_data_tx (uint8_t  *dev_addr, uint16_t  lcid, uint16_t  len, uint8_t  *p_data);
```

Description

This command is used to initiate new PSM connection with remote device.

Structure Variables

Variables	Description
dev_addr	Remote Device address
lcid	local channel identifiervalue
len	length of the data to be sent to remote device.
data	Actual data that has to be sent

Return values

Value	Description
0	command succeded
Non Zero	Error (Refer Error codes)

Example

```
rsi_ble_cbfc_data_tx (remote_dev_addr, lcid, sizeof (cbfc_data), cbfc_data);
```

13.4.4 rsi_ble_cbfc_disconnect

Prototype

```
uint32_t rsi_ble_cbfc_disconnect (uint8_t  *dev_addr, uint16_t  lcid);
```

Description
This command is used to delete PSM connection with remote device.

Structure Variables

Variables	Description
dev_addr	Remote Device address
lcid	local channel identifiervalue

Return values

Value	Description
0	command succeded
Non Zero	Error (Refer Error codes)

Example

```
rsi_ble_cbfc_disconnect (remote_dev_addr, lcid);
```

GATT API

This section explains the GATT APIs. The response payload for all the aysnc APIs will be indicated to the application by using the corresponding callback functions as described in the below sections. The response payload structure format is described along with the callback functions.

13.5 GATT Client APIs

13.5.1 rsi_ble_get_profiles

Prototype

```
int32_t rsi_ble_get_profiles(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 rsi_ble_resp_profiles_list_t *p_prof_list)
```

Description

This API is used to get the supported profiles / services of the connected remote device asynchronously.

rsi_ble_on_profiles_list_resp_t callback function will be called after the profiles list response is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address in ASCII string format
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records
p_prof_list	The profiles/services information will be filled in this structure after retrieving from the remote device.

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_profiles (remote_dev_addr, 1, 0xffff, &ble_profiles);
```

13.5.2 rsi_ble_get_profile

Prototype

```
int32_t rsi_ble_get_profile(uint8_t *dev_addr,
 uuid_t profile_uuid,
 profile_descriptors_t *p_profile)
```

Description

This API gets the specific profile / service of the connected remote device.

rsi_ble_on_profile_resp_t callback function is called once the service characteristics response is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
profile_uuid	This is the services/profiles which are searched using profile_uuid
p_profile	This is the profile / service information filled in this structure after retrieving from the remote device.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status= rsi_ble_get_profile (remote_dev_addr, search_serv, &ble_profile);
```

13.5.3 rsi_ble_get_char_services

Prototype

```
int32_t rsi_ble_get_char_services(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 rsi_ble_resp_char_services_t *p_char_serv_list);
```

Description

This API gets the service characteristics of the connected / remote device.

rsi_ble_on_inc_services_resp_t callback function is called once the included service characteristics response is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records
p_char_service_list	This is the service characteristics details filled in this structure

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
//! get characteristics of the immediate alert service
 rsi_6byte_dev_address_to_ascii ((int8_t *)remote_dev_addr, (uint8_t *)conn_event_to_app.dev_addr);

 rsi_ble_get_char_services (remote_dev_addr, *(uint16_t
*)rsi_ble_service.start_handle, *(uint16_t *)rsi_ble_service.end_handle,
 &char_servs);
```

13.5.4 rsi_ble_get_inc_services

Prototype

```
int32_t rsi_ble_get_inc_services(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 rsi_ble_resp_inc_services_t p_inc_service_list);
```

Description

This API gets the supported include services of the connected / remote device.

rsi_ble_on_att_desc_resp_t callback function is called once the service characteristics response is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records
p_inc_service_list	This include service characteristics details filled in this structure

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Value	Description
	If return value is lesser than 0 -4: Buffer not available to serve the command

Example

```
status = rsi_ble_get_inc_services (remote_dev_addr, 1, 0xFFFF, &ble_inc_serv);
```

13.5.5 rsi_ble_get_char_value_by_uuid

Prototype

```
int32_t rsi_ble_get_char_value_by_uuid(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 uuid_t char_uuid,
 rsi_ble_resp_att_value_t *p_char_val)
```

Description

This API gets the characteristic value by UUID (char_uuid).

rsi_ble_on_read_resp_t callback function is called once the attribute value is received

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records
char_uuid	This is the UUID of the characteristic
p_char_val	This is the characteristic value entered in this structure

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_char_value_by_uuid (remote_dev_addr, 1, 0xFFFF, search_serv, &ble_read_val);
```

13.5.6 rsi_ble_get_att_descriptors

Prototype

```
int32_t rsi_ble_get_att_descriptors(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 rsi_ble_resp_att_descs_t *p_att_desc)
```

Description

This API gets the characteristic descriptors list from the remote device**.**

rsi_ble_on_att_desc_resp_t callback function is called once the attribute descriptors response is received

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records
p_att_desc	This is the characteristic descriptor entered in this structure

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_att_descriptors(remote_dev_addr, 1, 0xFFFF, &ble_desc_services);
```

13.5.7 rsi_ble_get_att_value

Prototype

```
int32_t rsi_ble_get_att_value(uint8_t *dev_addr,
 uint16_t handle,
 rsi_ble_resp_att_value_t *p_att_val)
```

Description

This API gets the attribute by handle.

rsi_ble_on_read_resp_t callback function is called upon receiving the attribute value

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
handle	This is the handle of attribute
p_att_val	This is the attribute value entered in this structure.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_att_value(remote_dev_addr, 3, &ble_read_val);
```

13.5.8 rsi_ble_get_multiple_att_values

Prototype

```
int32_t rsi_ble_get_multiple_att_values(uint8_t *dev_addr,
 uint8_t num_of_handlers,
 uint16_t *handles, rsi_ble_resp_att_value_t *p_att_vals)
```

Description

This API gets the multiple attribute values by using multiple handles.

rsi_ble_on_read_resp_t callback function is called once the attribute value is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
num_of_handlers	This is the number of handles in the list
handles	This is the list of attribute handles
p_att_vals	These are the attribute values entered in this structure

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_get_multiple_att_values (remote_dev_addr, 2, handles, &ble_read_val);
```

13.5.9 rsi_ble_get_long_att_value

Prototype

```
int32_t rsi_ble_get_long_att_value(uint8_t *dev_addr,
 uint16_t handle,
 uint16_t offset,
 rsi_ble_resp_att_value_t *p_att_vals)
```

Description

This API gets the long attribute value by using handle and offset.

rsi_ble_on_read_resp_t callback function is called once the attribute value is received

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address

Parameter	Description
handle	This is the attribute handle
offset	This is the offset within the attribute value
p_att_vals	This is the attribute value entered in this structure

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_get_long_att_value(remote_dev_addr, 3, 0, &ble_read_val);
```

13.5.10 rsi_ble_set_att_value

Prototype

```
int32_t rsi_ble_set_att_value(uint8_t *dev_addr,
 uint16_t handle,
 uint8_t data_len,
 uint8_t *p_data)
```

Description

This API sets the attribute value.

rsi_ble_on_write_resp_t callback function is called once the attribute set action is completed

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
handle	This is the attribute handle
data_len	This is the attribute value length
p_data	This is the attribute value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
rsi_ble_set_att_value (RSI_BLE_DEV_1_ADDR, 0x60, 2, (uint8_t *)&slave_val);
```

13.5.11 rsi_ble_set_att_cmd

Prototype

```
int32_t rsi_ble_set_att_cmd (uint8_t *dev_addr,
 uint16_t handle,
 uint8_t data_len,
 uint8_t *p_data)
```

Description

This API sets attribute value without waiting for any ack from remote device.

rsi_ble_on_write_resp_t callback function is called if the attribute set action is completed.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
handle	This is the attribute handle
data_len	This is the attribute value length
p_data	This is the attribute value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_set_att_cmd (remote_dev_addr, 3, 7, "Silicon Labs");
```

13.5.12 rsi_ble_set_long_att_value

Prototype

```
int32_t rsi_ble_set_long_att_value(uint8_t *dev_addr,
 uint16_t handle,
 uint16_t offset,
 uint8_t data_len,
 uint8_t *p_data)
```

Description

This API sets long attribute value.

rsi_ble_on_write_resp_t callback function is called once the attribute set action is completed.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
handle	attribute handle
Offset	attribute value offset

Parameter	Description
data_len	Attribute value length
p_data	Attribute value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
rsi_ble_set_long_att_value (dev_addr,handle,offset,data_len,&p_data);
```

13.5.13 rsi_ble_prepare_write

Prototype

```
int32_t rsi_ble_prepare_write(uint8_t *dev_addr,
 uint16_t handle,
 uint16_t offset,
 uint8_t data_len,
 uint8_t *p_data)
```

Description

This API prepares attribute value.

rsi_ble_on_write_resp_t callback function is called once the prepare attribute write action is completed.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
handle	This is the attribute handle
offset	This is the attribute value offset
data_len	This is the attribute value length
p_data	This is the attribute value

Return Values

Value

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_prepare_write(dev_addr, handle, offset, data_len, &p_data);
```

13.5.14 rsi_ble_execute_write

Prototype

```
int32_t rsi_ble_execute_write(uint8_t *dev_addr,
 uint8_t exe_flag)
```

Description

This API executes the prepared attribute values.

rsi_ble_on_write_resp_t callback function is called once the execute attribute write action is completed.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
exe_flag	This is the execute flag to write

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_execute_write (dev_addr,  exe_flag);
```

13.5.15 rsi_ble_notify_value

Prototype

```
int32_t rsi_ble_notify_value (uint8_t  *dev_addr,
                              uint16_t  handle,
                              uint16_t  data_len,
                              uint8_t  *p_data);
```

Description

This API Notify the local value to remote device.

Precondition
Connection must be established before calling this API.

Parameters

Parameter	Description
dev_addr	remote bd address in string format
handle	local attribute handle
data_len	attribute value length
p_data	attribute value

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non-Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status int32_t rsi_ble_notify_value (dev_addr,handle,data_len,p_data);
```

13.5.16 rsi_ble_indicate_value

Prototype

```
int32_t rsi_ble_indicate_value (uint8_t  *dev_addr,
                                uint16_t  handle,
                                uint16_t  data_len,
                                uint8_t  *p_data);
```

Description

This API Indicate the local value to remote device.

Precondition

rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	remote bd address in string format
handle	local attribute handle
data_len	attribute value length
p_data	attribute value

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_indicate_value (dev_addr,handle,data_len,p_data);
```

13.5.17 rsi_ble_remove_gatt_service

Prototype

```
int32_t rsi_ble_remove_gatt_service (uint32_t  service_handler);
```

Description

This API Remove the GATT service record.

Precondition

rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
service_handler	GATT service record handler

Return Values

Value	Description
0	Successful execution of the command
Non-Zero	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command

Example

```
status = rsi_ble_remove_gatt_service (service_handler);
```

13.5.18 rsi_ble_remove_gatt_attibute

Prototype

```
int32_t rsi_ble_remove_gatt_attibute (uint32_t  service_handler, uint16_t att_hndl);
```

Description

This API Remove the GATT service record.

Precondition

rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
service_handler	GATT service record handler
att_hndl	attribute handle

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_remove_gatt_attibute (service_handler,att_handle);
```

13.5.19 rsi_ble_att_error_response

Prototype

```
int32_t rsi_ble_att_error_response (uint8_t *dev_addr, uint16_t  handle, uint8_t opcode, uint8_t err);
```

Description

This API is used to send att error response for any of the att request.

Precondition

rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	Remote Device Bd Address
att_hndl	attribute handle
opcode	Error response opcode
err	Specific error related Gatt

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_ble_att_error_response (&dev_addr,handle,opcode,err);
```

13.5.20 rsi_ble_mtu_exchange_event

Prototype

```
int32_t rsi_ble_mtu_exchange_event (uint8_t *dev_addr, uint8_t mtu_size)
```

Description

This API is used for mtu exchange event.

Precondition

rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address in ASCII string format
mtu_size	requested mtu value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status= rsi_ble_mtu_exchange_event (uint8_t *dev_addr,
                                    uint8_t mtu_size)
```

13.5.21 rsi_ble_gatt_write_response

Prototype

```
int32_t rsi_ble_gatt_write_response(uint8_t *dev_addr, uint8_t type)
```

Description

This function is used to send the local attribute value to remote device.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address in ASCII string format
type	0 - write response, 1- execute write response

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status=rsi_ble_gatt_write_response(uint8_t  *dev_addr,uint8_t   type);
```

13.5.22 rsi_ble_gatt_prepare_write_response

Prototype

```
int32_t rsi_ble_gatt_prepare_write_response(uint8_t  *dev_addr,
                                           uint16_t  handle,
                                           uint16_t  offset,
                                           uint16_t  length,
                                           uint8_t   *data )
```

Description

This function is used to send the local attribute value to remote device.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address in ASCII string format
handle	att handle
offset	att offset
length	lentgh of the data to be sent in response
data	data to be send

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status=rsi_ble_gatt_prepare_write_response(uint8_t *dev_addr,uint16_t handle,uint16_t offset,uint16_t
length,uint8_t *data);
```

13.6 GATT Client APIs Asynchronous

13.6.1 rsi_ble_get_profiles_async

Prototype

```
int32_t rsi_ble_get_profiles_async(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle)
```

Description

This API is used to get the supported profiles / services of the connected remote device asynchronously.

rsi_ble_on_event_profiles_list_t callback function will be called after the profiles list event is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address in ASCII string format
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_profiles_async (remote_dev_addr, 1, 0xffff);
```

13.6.2 rsi_ble_get_profile_async

Prototype

```
int32_t rsi_ble_get_profile_async(uint8_t *dev_addr,
 uuid_t profile_uuid)
```

Description

This API gets the specific profile / service of the connected remote device.

rsi_ble_one_event_profile_by_uuid_t callback function is called once the service characteristics response is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address

Parameter	Description
profile_uuid	This is the services/profiles which are searched using profile_uuid

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status= rsi_ble_get_profile_async (remote_dev_addr, search_serv);
```

13.6.3 rsi_ble_get_char_services_async

Prototype

```
int32_t rsi_ble_get_char_services_async(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle);
```

Description

This API gets the service characteristics of the connected / remote device.

rsi_ble_on_event_read_by_char_services_t callback function is called once the included service characteristics response is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
//! get characteristics of the immediate alert service
 rsi_6byte_dev_address_to_ascii ((int8_t *)remote_dev_addr, (uint8_t *)conn_event_to_app.dev_addr);

 rsi_ble_get_char_services_async (remote_dev_addr, *(uint16_t
*)rsi_ble_service.start_handle, *(uint16_t *)rsi_ble_service.end_handle);
```

13.6.4 rsi_ble_get_inc_services_async

Prototype

```
int32_t rsi_ble_get_inc_services_async(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle);
```

Description

This API gets the supported include services of the connected / remote device.

rsi_ble_on_event_read_by_inc_services_t callback function is called once the service characteristics response is received

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_inc_services_async(remote_dev_addr, 1, 0xFFFF, &ble_inc_serv);
```

13.6.5 rsi_ble_get_char_value_by_uuid_async

Prototype

```
int32_t rsi_ble_get_char_value_by_uuid_async(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 uuid_t char_uuid)
```

Description

This API gets the characteristic value by UUID (char_uuid).

rsi_ble_on_event_read_att_value_t callback function is called once the attribute value is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records
char_uuid	This is the UUID of the characteristic

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_char_value_by_uuid_async(remote_dev_addr, 1, 0xFFFF, search_serv);
```

13.6.6 rsi_ble_get_att_descriptors_async

Prototype

```
int32_t rsi_ble_get_att_descriptors_async(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle)
```

Description

This API gets the characteristic descriptors list from the remote device.

rsi_ble_on_gatt_desc_val_event_t callback function is called once the attribute descriptors response is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_att_descriptors_async(remote_dev_addr, 1, 0xFFFF, &ble_desc_services);
```

13.6.7 rsi_ble_get_att_value_async

Prototype

```
int32_t rsi_ble_get_att_value_async(uint8_t *dev_addr,
 uint16_t handle)
```

Description

This API gets the attribute by handle.

rsi_ble_on_event_read_resp_t callback function is called upon receiving the attribute value.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
handle	This is the handle of attribute

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_att_value_async(remote_dev_addr, 3);
```

13.6.8 rsi_ble_get_multiple_att_values_async

Prototype

```
int32_t rsi_ble_get_multiple_att_values_asyc(uint8_t *dev_addr,
 uint8_t num_of_handlers,
 uint16_t *handles)
```

Description

This API gets the multiple attribute values by using multiple handles.

rsi_ble_on_event_read_resp_t callback function is called once the attribute value is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
num_of_handlers	This is the number of handles in the list
handles	This is the list of attribute handles

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_multiple_att_values_async (remote_dev_addr, 2, handles);
```

13.6.9 rsi_ble_get_long_att_value_async

Prototype

```
int32_t rsi_ble_get_long_att_value_async(uint8_t *dev_addr,
 uint16_t handle,
 uint16_t offset)
```

Description

This API gets the long attribute value by using handle and offset.

rsi_ble_on_event_read_resp_t callback function is called once the attribute value is received.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
handle	This is the attribute handle
offset	This is the offset within the attribute value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_long_att_value_async(remote_dev_addr, 3, 0);
```

13.6.10 rsi_ble_set_att_value_async

Prototype

```
int32_t rsi_ble_set_att_value_async(uint8_t *dev_addr,
 uint16_t handle,
 uint8_t data_len,
 uint8_t *p_data)
```

Description

This API sets the attribute value.

rsi_ble_on_event_write_resp_t callback function is called once the attribute set action is completed.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
handle	This is the attribute handle
data_len	This is the attribute value length
p_data	This is the attribute value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
rsi_ble_set_att_value (RSI_BLE_DEV_1_ADDR, 0x60, 2, (uint8_t *)&slave_val);
```

13.6.11 rsi_ble_prepare_write_async

Prototype

```
int32_t rsi_ble_prepare_write_async(uint8_t *dev_addr,
 uint16_t handle,
 uint16_t offset,
 uint8_t data_len,
 uint8_t *p_data)
```

Description

This API prepares attribute value.

rsi_ble_on_event_prepare_write_resp_t callback function is called once the prepare attribute write action is completed.

Precondition
rsi_ble_connect() API needs to be called before this API

Parameters

Parameter	Description
dev_addr	This is the remote device address
handle	This is the attribute handle
offset	This is the attribute value offset
data_len	This is the attribute value length
p_data	This is the attribute value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_prepare_write_async(dev_addr, handle, offset, data_len, &p_data);
```

13.6.12 rsi_ble_execute_write_async

Prototype

```
int32_t rsi_ble_execute_write_async(uint8_t *dev_addr,
 uint8_t exe_flag)
```

Description

This API executes the prepared attribute values.

rsi_ble_on_event_write_resp_t callback function is called once the execute attribute write action is completed.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device address
exe_flag	This is the execute flag to write

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_execute_write_async(dev_addr,  exe_flag);
```

13.7 GATT Server APIs

Note:

Please refer definitions for the following structures in "**Basic Structure defines**" section.

13.7.1 rsi_ble_add_service

Prototype

```
int32_t rsi_ble_add_service (uuid_t serv_uuid,
 rsi_ble_resp_add_serv_t *p_resp_serv)
```

Description
This API adds a new service to the local GATT Server.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
serv_uuid	This is the new service uuid value
p_resp_serv	This is the new service handler entered in this structure. This is the output parameter

The uuid_t structure details

```
typedef struct bt_uuid128 {
 UINT08 data1[4];
 UINT08 data2[2];
```

```
UINT08 data3[2];
UINT08 data4[8];
} UUID128;
typedef UINT16 UUID16;
typedef UINT32 UUID32;
/* Main UUID structure */
typedef struct bt_uuid {
UINT08 size; // Size of the UUID
UINT08 reserved[3];
union bt_uuid_t {
UUID128 val128;
UUID32 val32;
UUID16 val16;
} val; // UUID value
} uuid_t;
```

Description

The size of a UUID (Universal Unique Identifier) can be 2byte (16bit), 4byte (32bit) or 16byte (128bit). This structure defines the size of uuid and uuid value.

Structure Variables

Variables	Description
size	This is the size of uuid
reserved	Reserved
val	This is the value of one of the 3 types ((128 bit, 32 bit or 16 bit) of UUIDs.

The rsi_ble_resp_add_serv_t structure details

```
typedef struct rsi_ble_resp_add_serv_s
{
    void      *serv_handler;
    uint16_t   start_handle;
} rsi_ble_resp_add_serv_t;
```

Description

The above structure is output parameter structure in which contents like the service record address and the from which handle it starts will contain.

Variables	Description
serv_handler	Contains the address of the service record stored in the Silicon Labs stack.
start_handle	The handle from where the service starts.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Please refer to Error Codes section for the description of the above error codes.

Example

```
status = rsi_ble_add_service  (service_uuid, &p_resp_serv);
```

13.7.2 rsi_ble_add_attribute

Prototype

```
int32_t rsi_ble_add_attribute (rsi_ble_req_add_att_t
 *p_attribute)
```

Description
This API adds a new attribute to a specific service.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
p_attribute	This is used to add a new attribute to the service.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
uuid_t new_uuid = {0};
 rsi_ble_resp_add_serv_t new_serv_resp = {0};
 //! adding new service
 new_uuid.size = 2;
 new_uuid.val.val16 = RSI_BLE_NEW_SERVICE_UUID;
 rsi_ble_add_service (new_uuid, 10, 100, &new_serv_resp);
```

13.7.3 rsi_ble_set_local_att_value

Prototype

```
int32_t rsi_ble_set_local_att_value (uint16_t handle,
 uint16_t data_len,
 uint8_t *p_data)
```

Description
This API changes the local attribute value.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
handle	This is the local attribute handle
data_len	This is the attribute value length
p_data	This is the attribute value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0**

Value	Description
	-4 : Buffer not available to serve the command

Example

```
//! set the local attribute value.
 rsi_ble_set_local_att_value (rsi_ble_att2_val_hndl, RSI_BLE_MAX_DATA_LEN, rsi_ble_app_data);
```

13.7.4 rsi_ble_get_local_att_value

Prototype

```
int32_t rsi_ble_get_local_att_value (uint16_t handle, rsi_ble_resp_local_att_value_t
*p_resp_local_att_val)
```

Description
This API gets the local attribute value.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
handle	This is the local attribute handle
p_resp_local_att_val	This is the local attribute value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
status = rsi_ble_get_local_att_value (handle, &p_resp_local_att_val);
```

13.7.5 rsi_ble_set_wo_resp_notify_buf_info

Prototype

```
int32_t rsi_ble_set_wo_resp_notify_buf_info (uint8_t *dev_addr,
                                             uint8_t  buf_mode,
                                             uint8_t  buf_cnt)
```

Description
This function is used to configure the buf mode for Notify and WO resp commands.

Parameters

Parameter	Description
dev_addr	This is the remote device BD Address
buf_mode	buffer mode configuration
buf_count	no of buffers to be configured

Return Values

Value	Description
0	Successful execution of the command
Nonzero	Failure

Example

```
rsi_ble_get_local_att_value (rsi_ble_att1_val_hndl, &local_att_val);
```

13.7.6 rsi_ble_gatt_read_response

Prototype

```
int32_t rsi_ble_gatt_read_response(uint8_t *dev_addr,
 uint8_t read_type,
 uint16_t handle,
 uint16_t offset,
 uint16_t length,
 uint8_t *p_data)
```

Description
This API sends the local attribute value to the remote device.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
dev_addr	This is the remote device BD Address
read_type	0-Read Response 1- Read blob response
handle	This is the local attribute start handle
offset	This is the local attribute value start offset
length	This is the local attribute value length
data	This is the Local attribute value

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example

```
rsi_ble_get_local_att_value (rsi_ble_att1_val_hndl, &local_att_val);
```

13.7.7 rsi_ble_setphy

Prototype

```
int32_t rsi_ble_setphy(int8_t *remote_dev_address, uint8_t tx_phy, uint8_t rx_phy, uint16_t coded_phy)
```

Description

This function is used to set tx and rx phy.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_address	Address of remote device
tx_phy	0 The Host prefers to use the LE 1M transmitter PHY (possibly among others) 1 The Host prefers to use the LE 2M transmitter PHY (possibly among others) 2 The Host prefers to use the LE Coded transmitter PHY (possibly among others) 3 – 7 Reserved for future use
rx_phy	0 The Host prefers to use the LE 1M receiver PHY (possibly among others) 1 The Host prefers to use the LE 2M receiver PHY (possibly among others) 2 The Host prefers to use the LE Coded receiver PHY (possibly among others) 3 – 7 Reserved for future use
coded_phy	0 = the Host has no preferred coding when transmitting on the LE Coded PHY 1 = the Host prefers that S=2 coding be used when transmitting on the LE Coded PHY 2 = the Host prefers that S=8 coding be used when transmitting on the LE Coded PHY 3 = Reserved for future use

Return Values

Value	Description
0	command succeeded
Non Zero	Failure

Example

```
status = rsi_ble_setphy(str_addr, 1, 2, 0 );
```

13.7.8 rsi_ble_conn_params_update

Prototype

```
int32_t rsi_ble_conn_params_update(int8_t *remote_dev_address, uint16_t min_int, uint16_t max_int,
uint16_t latency, uint16_t timeout)
```

Description

This function is used to request the connection params with the remote device when RSI device is acting as Slave and update the connection params with the remote device when RSI device is acting as Master.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_address	Address of remote device
min_int	Minimum value for the connection interval. This shall be less than or equal to Conn_Interval_Max.

Parameter	Description
max_int	Conn_Interval_Max (in dec) – Maximum value for the connection interval. This shall be greater than or equal to Conn_Interval_Min.
latency	Conn_Latency (in dec) – Slave latency for the connection in number of connection events. Range: 0 to 499
timeout	Supervision_Timeout (in dec) – Supervision timeout for the LE Link. Range: 10 to 3200 (Time = N * 10 ms, Time Range: 100 ms to 32 s)

Note:

Min and max interval Range: 6 to 3200 (Time = N * 1.25 ms, Time Range: 7.5 ms to 4 s).

Return Values

Value	Description
0	Connection Params Update command succeeded
Non Zero	Failure

Example

```
status = rsi_ble_conn_params_update(str_addr, 160, 160, 0, 2000);
```

13.7.9 rsi_ble_conn_param_resp

Prototype

```
int32_t rsi_ble_conn_param_resp(uint8_t *remote_dev_address, uint8_t status)
```

Description

This function is used to give the response for the remote device connection params request.

Precondition

rsi_ble_connect() API needs to be called before this API.

Parameters

Parameters	Description
remote_dev_address	Address of remote device
status	accept or reject status (0-ACCEPT 1-REJECT)

Return Values

None

Example

```
status = rsi_ble_conn_param_resp(remote_device_address, 0 );
```

13.7.10 rsi_ble_set_data_len

Prototype

```
int32_t rsi_ble_set_data_len (uint8_t *remote_dev_address, uint16_t tx_octets ,uint16_t tx_time )
```

Description

This function is used to set tx octets and tx time.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameter	Description
remote_dev_address	Address of remote device
tx_octets	Preferred maximum number of payload octets that the local Controller should include in a single Link Layer packet on this connection.
tx_time	Preferred maximum number of microseconds that the local Controller should use to transmit a single Link Layer packet on this connection.

Return Values

Value	Description
0	LE_Set_Data_Length command succeeded.
Non Zero	Failure
Connection_Handle	Connection_HandleRange 0x0000-0x0EFF

Example

```
TX_LEN    = 0x001e
TX_TIME   = 0x01f4

status = rsi_ble_set_data_len(str_addr, TX_LEN , TX_TIME );
```

13.7.11 rsi_ble_read_max_data_len

Prototype

```
int32_t rsi_ble_read_max_data_len (rsi_ble_read_max_data_length_t *blereaddatalen)
typedef struct rsi_ble_resp_read_max_data_length_s{
    uint16_t  maxtxoctets;
    uint16_t  maxtxtime;
    uint16_t  maxrxoctets;
    uint16_t  maxrxtime;
}rsi_ble_read_max_data_length_t;
```

Description

This function is used to set tx octets and tx time.

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters:

structure variable of the type rsi_ble_read_max_data_length_t.

Return Values

Parameter	Description
tx_octets	Preferred maximum number of payload octets that the local Controller should include in a single Link Layer packet on this connection.
tx_time	Preferred maximum number of microseconds that the local Controller should use to transmit a single Link Layer packet on this connection
maxrxoctets	Maximum number of payload octets that the local Controller supports for reception of a single Link Layer packet on a data connection. .

Parameter	Description
maxrxtime	Maximum time, in microseconds, that the local Controller supports for reception of a single Link Layer packet on a data connection.

Example

```
status = rsi_ble_read_max_data_len(&blereadmaxdatalen);
```

13.7.12 rsi_ble_readphy

Prototype

```
int32_t rsi_ble_readphy(int8_t *remote_dev_address, rsi_ble_resp_read_phy_t *resp)

typedef struct rsi_ble_resp_read_phy_s {
   uint8_t   dev_addr[RSI_DEV_ADDR_LEN];
   uint8_t   tx_phy;
   uint8_t   rx_phy;
}rsi_ble_resp_read_phy_t;
```

Description

This function is used to read tx and rx phy.

Precondition
rsi_ble_connect() API needs to be called before this API.

Command Parameters

Parameter	Description
Connection_Handle	Connection_Handle Range: 0x0000-0x0EFF

Return values

Parameter	Description
Connection_Handle	Connection_Handle Range:0x0000-0x0EFF
TX_PHY:	0x01 The transmitter PHY for the connection is LE 1M 0x02 The transmitter PHY for the connection is LE 2M 0x03 The transmitter PHY for the connection is LE Coded All other values Reserved for future use
RX_PHY:	0x01 The receiver PHY for the connection is LE 1M 0x02 The receiver PHY for the connection is LE 2M 0x03 The receiver PHY for the connection is LE Coded All other values Reserved for future use

Example

```
status = rsi_ble_readphy( str_addr, &read_phy_resp);
```

13.7.13 rsi_ble_resolvlist

Prototype:

```
int32_t rsi_ble_resolvlist (uint8_t process_type, uint8_t remote_dev_addr_type, uint8_t
*remote_dev_address, uint8_t *peer_irk, uint8_t *local_irk)
```

Description
It will add a device to resolving list device address, address type and irk's.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters:

Parameter	Description
process_type	it will tell type of process means: 1 for add device to resolve list 2 for remove a device from resolve list 3 for clear the entire resolve list
remote_dev_addr_type	This is the remote device address
peer_irk	16-byte irk of the peer device
local_irk	16-byte irk of the local device

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example:

```
status = rsi_ble_resolvlist(RSI_BLE_PROCESS_TYPE,RSI_BLE_DEV_ADDR_TYPE, RSI_BLE_DEV_ADDR_1, peer_irk
,local_irk );
```

13.7.14 rsi_ble_get_resolving_list_size

Prototype

```
int32_t rsi_ble_get_resolving_list_size (uint8_t *resp)
```

Description
request to get resolving list size.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
resp	This is an output parameter which consists of resolving list size This is a 1 byte value.

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example:

```
status = rsi_ble_get_resolving_list_size(&rsi_app_resp_resolvlist_size);
```

13.7.15 BT_LE_ADPacketExtract

Prototype:

```
void BT_LE_ADPacketExtract (uint8_t *remote_name,uint8_t *pbuf,uint8_t buf_len)
```

Description

This function is used to extract the advertised packet.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
pbuf	advertise data packet buffer pointer
buf_len	buffer length
remote_name	device name

Return value

None

Example

```
status = BT_LE_ADPacketExtract (&remote_name,*pbuf,buf_len);
```

13.7.16 rsi_ble_set_addr_resolution_enable

Prototype:

```
int32_t rsi_ble_set_addr_resolution_enable (uint8_t enable,uint16_t tout)
```

Description

request to enable address resolution, and to set resolvable private address timeout.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters:

Parameter	Description
enable	1 for enable address resolution 0 for disable address resolution
tout	the period for changing address of our local device in seconds.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example:

```
status =
rsi_ble_set_addr_resolution_enable(RSI_BLE_DEV_ADDR_RESOLUTION_ENABLE,RSI_BLE_SET_RESOLVABLE_PRIV_ADDR_
TOUT);
```

13.7.17 rsi_ble_set_privacy_mode

Prototype

```
int32_t rsi_ble_set_privacy_mode (uint8_t remote_dev_addr_type, uint8_t *remote_dev_address, uint8_t
privacy_mode)
```

Description

request to set privacy mode for device.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters:

Parameter	Description
remote_dev_addr_type	This is the remote device address 0x00 Public Identity Address 0x01 Random (static) Identity Address
remote_dev_address	bd address of remte device
privacy_mode	0-Network privacy mode 1-Device privacy mode

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example:

```
status = rsi_ble_set_privacy_mode(RSI_BLE_DEV_ADDR_TYPE,RSI_BLE_DEV_ADDR_1,RSI_BLE_PRIVACY_MODE);
```

13.7.18 rsi_ble_ltk_req_reply

Prototype

```
int32_t rsi_ble_ltk_req_reply (uint8_t *remote_dev_address, uint8_t reply_type, uint8_t *ltk);
```

Description
This API is used to send the local long term key of its associated local EDIV and local Rand.

Parameters

Parameter	Description
remote_dev_address	bd address of remote device
reply_type	0 - Negative reply 1 - Positive Reply
ltk	Long Term Key 16bytes

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure **If return value is lesser than 0** -4 : Buffer not available to serve the command

Example:

```
status = rsi_ble_ltk_req_reply(remote_dev_addr, 1, localltk);
```

13.8 Callback functions

GAP Register Callbacks

This function is used to register the call-back functions for synchronous GAP events.

13.8.1 rsi_ble_gap_register_callbacks

Prototype

```
void rsi_ble_gap_register_callbacks (
 rsi_ble_on_adv_report_event_t               ble_on_adv_report_event,
 rsi_ble_on_connect_t                        ble_on_conn_status_event,
 rsi_ble_on_disconnect_t                     ble_on_disconnect_event,
 rsi_ble_on_le_ping_payload_timeout_t        timeout_expired_event,
 rsi_ble_on_phy_update_complete_t            ble_on_phy_update_complete_event,
 rsi_ble_on_data_length_update_t             rsi_ble_on_data_length_update_event,
 rsi_ble_on_enhance_connect_t                ble_on_enhance_conn_status_event,
 rsi_ble_on_directed_adv_report_event_t      ble_on_directed_adv_report_event,
 rsi_ble_on_conn_update_complete_t           ble_on_conn_update_complete_event);
```

Description
This API registers GAP callbacks.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
ble_on_adv_report_event	This is the advertise report callback
ble_on_conn_status_event	This is the connection status callback
ble_on_disconnect_event	This is the disconnection status callback
timeout_expired_event	This is the ping payload timeout callback
ble_on_phy_update_complete_event	This is the Phy update complete event callback
rsi_ble_on_data_length_update_event	This is the data length update event callback
ble_on_enhance_conn_status_event	This is the Enhanced connection status callback

Parameter	Description
ble_on_directed_adv_report_event	This is the Directed advertising report callback
ble_on_conn_update_complete_event	This is the Connection parameters updated event callback

Return Values
None

Note: For more information about each callback, please refer to GAP Callback Descriptions section.

GAP Event Callback Descriptions

13.8.2 rsi_ble_event_adv_report_t

Structure

```
typedef struct rsi_ble_event_adv_report_s
 {
 uint8_t dev_addr_type;
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint8_t adv_data_len;
 uint8_t adv_data[RSI_MAX_ADV_REPORT_SIZE];
 uint8_t rssi;
 }rsi_ble_event_adv_report_t;
```

Prototype

```
typedef void (*rsi_ble_on_adv_report_event_t) (rsi_ble_event_adv_report_t *rsi_ble_event_adv)
```

Description
This event occurs when rsi_ble_start_scanning command is issued. This callback is called when an advertising event is raised from the module and advertising report is filled in rsi_ble_event_adv structure.

Precondition
rsi_wireless_init() API needs to be called before this API.

Structure Variables

Variables	Description
dev_addr_type	This is the address type of the advertising device
dev_addr	This is the device address of the advertising device.
Rssi	This is the signal strength
Adv_data_len	This is the raw advertisement data length
Adv_data	This is the advertisement data

13.8.3 rsi_ble_event_conn_status_t

Structure

```
typedef struct rsi_ble_event_conn_status_s
 {
 uint8_t dev_addr_type;
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint8_t status;
 }rsi_ble_event_conn_status_t;
```

Prototype

```
typedef void (*rsi_ble_on_connect_t) (rsi_ble_event_conn_status_t *rsi_ble_event_conn)
```

Description
This call back is called when the module gives the connection status event. And the status will be filled in rsi_ble_event_conn structure. This event will be given by the module in the following two scenarios:

1. In case of slave mode, when the connection is initiated from the remote device.
2. In case of Master mode, when the connect command is issued to connect to a remote device.

Precondition
rsi_wireless_init() API needs to be called before this API.

Structure Variables

Variables	Description
dev_addr_type	This is the address type of the connected device
dev_addr	This is the device address of the connected device.
status	This is the status of the connection – Success or Failure

13.8.4 rsi_ble_event_disconnect_t

Structure

```
typedef struct rsi_ble_event_disconnect_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 }rsi_ble_event_disconnect_t;
```

Prototype

```
typedef void (*rsi_ble_on_disconnect_event_t) (rsi_ble_event_disconnect_t *rsi_ble_event_disconnect,
 uint16_t reason)
```

Description
This callback is called when the disconnect event is raised from the module. This callback will be called in the following two scenarios:

1. In case, disconnection is issued locally
2. In case, disconnection is initiated from a remote device

Precondition
rsi_wireless_init() API needs to be called before this API.

Structure Variables

Variables	Description
dev_addr	This is the device address of the disconnected device.
reason	This is the reason for disconnection

13.8.5 rsi_ble_on_le_ping_payload_timeout_t

Structure

```
typedef struct rsi_ble_event_le_ping_time_expired_s
 {
 //!uint8_t, address of the disconnected device
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 }rsi_ble_event_le_ping_time_expired_t;
```

Prototype

```
typedef void (*rsi_ble_on_le_ping_payload_timeout_t) (rsi_ble_event_le_ping_time_expired_t
*rsi_ble_event_timeout_expired);
```

Description
This callback is called when the LE ping payload timeout event is raised from the module i.e. when the timer exceeds threshold value, the module gets disconnected with the remote device and raises this event to the host.

Precondition
rsi_wireless_init() API needs to be called before this API.

Structure Variables

Variables	Description
dev_addr	Device address of the disconnected device.

13.8.6 rsi_ble_on_phy_update_complete_t

Structure

```
//! phy update complete event
typedef struct rsi_ble_event_phy_update_s {
    uint8_t  dev_addr[6];
    uint8_t  TxPhy;
    uint8_t  RxPhy;
} rsi_ble_event_phy_update_t;
```

Prototype

```
typedef void (*rsi_ble_on_phy_update_complete_t) (rsi_ble_event_phy_update_t
*rsi_ble_event_phy_update_complete);
```

Description
This callback is called when the phy update complete event is received.

Precondition
rsi_wireless_init() API needs to be called before this API.

Structure Variables

Variables	Description
dev_addr	Device address of the remote device.
TxPhy	Transmission phy rate
RxPhy	Reception phy rate

13.8.7 rsi_ble_on_data_length_update_t

Structure

```
typedef struct rsi_ble_event_data_length_update_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
  uint16_t  MaxTxOctets;
  uint16_t  MaxTxTime;
  uint16_t  MaxRxOctets;
  uint16_t  MaxRxTime;
} rsi_ble_event_data_length_update_t;
```

Prototype

```
typedef void (*rsi_ble_on_data_length_update_t) (rsi_ble_event_data_length_update_t
*remote_dev_address);
```

Description
This callback is called when the data length update complete event is received.

Precondition
rsi_wireless_init() API needs to be called before this API.

Structure Variables

Variables	Description
dev_addr	Device address of the remote device.
MaxTxOctets	Maximum Tx Octets to be transmitted

Variables	Description
MaxTxTime	Maximum Tx time to transmit the MaxTxOctets
MaxRxOctets	Maximum Rx Octets to be received
MaxRxTime	Maximum Rx tome to receive the MaxRxOctets

13.8.8 rsi_ble_on_enhance_connect_t

Structure

```
//!Enhance Connection status event structure
typedef struct rsi_ble_event_enhnace_conn_status_s
{
    //!uint8_t, address type of the connected device
    uint8_t dev_addr_type;
    //!uint8_t[6], address of the connected device
    uint8_t dev_addr[RSI_DEV_ADDR_LEN];
    //!uint8_t[6], resolvable address of the connected device
    uint8_t peer_resolvable_addr[RSI_DEV_ADDR_LEN];
    //!uint8_t[6], resolvable address of the local device
    uint8_t local_resolvable_addr[RSI_DEV_ADDR_LEN];
    //!uint8_t, status of the connection - success/failure
    uint8_t status;
}rsi_ble_event_enhance_conn_status_t;
```

Prototype

```
typedef void (*rsi_ble_on_enhance_connect_t) (rsi_ble_event_enhance_conn_status_t
*rsi_ble_event_enhance_conn);
```

Description
This callback is called when the BLE Enhanced connection status is received from the module.

Precondition
rsi_wireless_init() API needs to be called before this API.

Structure Variables

Variables	Description
dev_addr_type	Device address type of the Connected Remote Device
dev_addr	Device address of the remote device.
peer_resolvable_addr	Remote Device resolvable address
local_resolvable_addr	Local Device resolvable address
status	Status of the Connection

13.8.9 rsi_ble_on_directed_adv_report_event_t

Structure

```
typedef struct rsi_ble_event_directedadv_report_s
{
    uint16_t event_type;
    uint8_t dev_addr_type;
    uint8_t dev_addr[RSI_DEV_ADDR_LEN];
    uint8_t directed_addr_type;
    uint8_t directed_dev_addr[RSI_DEV_ADDR_LEN];
    int8_t rssi;
}rsi_ble_event_directedadv_report_t;
```

Prototype

```
typedef void (*rsi_ble_on_directed_adv_report_event_t) (rsi_ble_event_directedadv_report_t
*rsi_ble_event_directed);
```

Description
This callback is called when the advertise event report is received from the module.

Precondition
rsi_wireless_init() API needs to be called before this API.

Structure Variables

Variables	Description
event_type	Received Advertising Type
dev_addr_type	Remote device address type
dev_addr	Device address of the remote device.
directed_addr_type	Address type of the device to whom it is intended
directed_dev_addr	Bd Address of the device to whom it is intended
rssi	rssi value

13.8.10 rsi_ble_on_conn_update_complete_t

Structure

```
//! conn update complete event
typedef struct rsi_ble_event_conn_update_s {
    uint8_t   dev_addr[6];
    uint16_t  conn_interval;
    uint16_t  conn_latency;
    uint16_t  timeout;
} rsi_ble_event_conn_update_t;
```

Prototype

```
typedef void (*rsi_ble_on_conn_update_complete_t) (rsi_ble_event_conn_update_t
*rsi_ble_event_conn_update_complete, uint16_t resp_status);
```

Description
This callback is called when the connection parameters update complete event is received

Precondition
rsi_wireless_init() API needs to be called before this API.

Structure Variables

Variables	Description
dev_addr	Device address of the remote device.
conn_interval	Connection Interval
conn_latency	Connection Latency
timeout	Supervision Timeout

GAP Extended Register Callbacks

This function is used to register the callback functions for GAP Extended responses and events.

13.8.11 rsi_ble_gap_extended_register_callbacks

Prototype

```
rsi_ble_gap_extended_register_callbacks (
    rsi_ble_on_remote_features_t      ble_on_remote_features_event,
    rsi_ble_on_le_more_data_req_t     ble_on_le_more_data_req_event) ;
```

Description
This API registers GAP Extended responses/events callbacks.

Precondition
rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
ble_on_remote_features_event	Call back function for Remote feature request
ble_on_le_more_data_req_event	Call back function for LE More data request

Return Values

None

Note: For more information about each callback, please refer to GAP Extended callbacks description section.

GAP Extended Callbacks Description

13.8.12 ble_on_remote_features_event

Prototype

```
typedef void (*rsi_ble_on_remote_features_t) (rsi_ble_event_remote_features_t
*rsi_ble_event_remote_features)
```

Structure

```
typedef struct rsi_ble_event_remote_features_s {
    uint8_t   dev_addr[6];
    uint8_t   remote_features[8];
} rsi_ble_event_remote_features_t;
```

Description
This callback function is called if the remote features are received from the module . This callback has to be registered using rsi_ble_gap_extended_register_callbacks API.

resp_status, contains the response status (Success(0) or Error code).

Structure Variables

Variables	Data type	Description
dev_addr	uint8_t	Remote device address
remote_features	uint8_t	Remote device supported features

13.8.12.1 ble_on_le_more_data_req_event

Prototype

```
typedef void (*rsi_ble_on_le_more_data_req_t) ();
```

Structure

No variables, just an event without data.

Description
This callback function is called if the le more data request event received from the module. This callback has to be registered using rsi_ble_gap_extended_register_callbacks API.

resp_status, contains the response status (Success (0) or Error code).

GATT Register Callbacks

This function is used to register the callback functions for GATT responses and events.

13.8.13 rsi_ble_gatt_register_callbacks

Prototype

```
rsi_ble_gatt_register_callbacks (
    rsi_ble_on_profiles_list_resp_t          ble_on_profiles_list_resp,
    rsi_ble_on_profile_resp_t                ble_on_profile_resp,
    rsi_ble_on_char_services_resp_t          ble_on_char_services_resp,
    rsi_ble_on_inc_services_resp_t           ble_on_inc_services_resp,
    rsi_ble_on_att_desc_resp_t               ble_on_att_desc_resp,
    rsi_ble_on_read_resp_t                   ble_on_read_resp,
    rsi_ble_on_write_resp_t                  ble_on_write_resp,
    rsi_ble_on_gatt_write_event_t            ble_on_gatt_event,
    rsi_ble_on_gatt_prepare_write_event_t    ble_on_gatt_prepare_write_event,
    rsi_ble_on_execute_write_event_t         ble_on_execute_write_event,
    rsi_ble_on_read_req_event_t              ble_on_read_req_event,
    rsi_ble_on_mtu_event_t                   ble_on_mtu_event,
    rsi_ble_on_gatt_error_resp_t             ble_on_gatt_error_resp_event,
    rsi_ble_on_gatt_desc_val_event_t         ble_on_gatt_desc_val_resp_event,
    rsi_ble_on_event_profiles_list_t         ble_on_profiles_list_event,
    rsi_ble_on_event_profile_by_uuid_t       ble_on_profile_by_uuid_event,
    rsi_ble_on_event_read_by_char_services_t ble_on_read_by_char_services_event,
    rsi_ble_on_event_read_by_inc_services_t  ble_on_read_by_inc_services_event,
    rsi_ble_on_event_read_att_value_t        ble_on_read_att_value_event,
    rsi_ble_on_event_read_resp_t             ble_on_read_resp_event,
    rsi_ble_on_event_write_resp_t            ble_on_write_resp_event,
    rsi_ble_on_event_prepare_write_resp_t    ble_on_prepare_write_resp_event);
```

Description

This API registers GATT response callbacks.

Precondition

rsi_wireless_init() API needs to be called before this API.

Parameters

Parameter	Description
ble_on_profiles_list_resp	This is the callback for rsi_ble_req_profiles command
ble_on_profile_resp	This is the callback for rsi_ble_req_profile command
ble_on_char_services_resp	This is the callback for rsi_ble_req_char_services command
ble_on_inc_services_resp	This is the callback for rsi_ble_req_inc_services command
ble_on_att_desc_resp	This is the callback for rsi_ble_req_att_descriptors command
ble_on_read_resp	This is the callback for all read requests command
ble_on_write_resp	This is the callback for all write commands
ble_on_gatt_event	This is the callback for gatt write event
ble_on_gatt_prepare_write_event	This is the callback for gatt prepare write event
ble_on_execute_write_event	This is the callback for gatt execute write event
ble_on_read_req_event	This is the callback for gatt read request event
ble_on_mtu_event	This is the callback for MTU size
ble_on_gatt_error_resp_event	Callback for GATT error events
ble_on_gatt_desc_val_resp_event	Callback for GATT descriptor value event
ble_on_profiles_list_event	callback function for profiles list async event
ble_on_profile_by_uuid_event	callback function for on profile async event
ble_on_read_by_char_services_event	callback function for char services async event
ble_on_read_by_inc_services_event	callback function for inc services async event
ble_on_read_att_value_event	callback function for read att value async event

Parameter	Description
ble_on_read_resp_event	callback function for read att async event
ble_on_write_resp_event	callback function for write async event
ble_on_prepare_write_resp_event	callback function for prepare write async event

Return Values

None

> **Note**:
>
> For more information about each callback, please refer to GATT response callbacks description section.

GATT Response Callbacks Description

13.8.14 rsi_ble_on_profiles_list_resp_t

Prototype

```
typedef void (*rsi_ble_on_profiles_list_resp_t) (
 uint16_t resp_status,
 rsi_ble_resp_profiles_list_t *rsi_ble_resp_profiles);
```

Structure

```
typedef struct rsi_ble_resp_profiles_list_s
 {
 uint8_t number_of_profiles;
 uint8_t reserved[3];
 profile_descriptors_t profile_desc[RSI_BLE_MAX_RESP_LIST];
 } rsi_ble_resp_profiles_list_t;
```

Description

This callback function is called if the profiles list response is received from the module . This callback must be registered using rsi_ble_gatt_register_callbacks API.

resp_status, contains the response status (Success(0) or Error code).

Structure Variables

Variables	Data type	Description
number_of_profiles	uint8_t	This is the number of profiles found
reserved	uint8_t	Reserved
profile_desc	profile_descriptors_t	This contains the profiles list. The maximum of 5 profiles are filled.

13.8.15 rsi_ble_on_profile_resp_t

Prototype

```
typedef void (*rsi_ble_on_profile_resp_t) (uint16_t resp_status, profile_descriptors_t
*rsi_ble_resp_profile);
```

Structure

```
typedef struct profile_descriptor_s
 {
 uint8_t start_handle[2];
 uint8_t end_handle[2];
 uuid_t profile_uuid;
 } profile_descriptors_t ;
```

Description

This callback function is called if the profile response is received from the module. This callback must be registered using rsi_ble_gatt_register_callbacks API.

resp_status, contains the response status (Success (0) or Error code).

Structure Variables

Variables	Data type	Description
start_handle	uint8_t	This is the start handle.
End_handle	uint8_t	This is the end handle.
profile_uuid	uuid_t	This is the profile uuid.

13.8.16 rsi_ble_on_char_services_resp_t

Prototype

```
typedef void (*rsi_ble_on_char_services_resp_t) (
 uint16 resp_status,
 rsi_ble_resp_char_services_t *rsi_ble_resp_char_serv);
```

Structure

```
typedef struct rsi_ble_resp_char_service_s
 {
 uint8_t num_of_services;
 uint8_t reserved[3];
 char_service_t char_services[RSI_BLE_MAX_RESP_LIST];
 } rsi_ble_resp_char_services_t;
```

Description

This callback function will be called if the service characteristics response is received from the module.

This callback has to be registered using rsi_ble_gatt_register_callbacks API.

resp_status, contains the response status (Success(0) or Error code).

Structure Variables

Variables	Data type	Description
num_of_services	uint8_t	This is the number of characteristic services found
Reserved	uint8_t	Reserved
char_services	char_service_t	It contains the characteristic service list. The maximum value is 5.

13.8.17 rsi_ble_on_inc_services_resp_t

Prototype

```
typedef void (*rsi_ble_on_inc_services_resp_t) (
 uint16_t resp_status,
 rsi_ble_resp_inc_services_t *rsi_ble_resp_inc_serv);
```

Structure

```
typedef struct rsi_ble_resp_inc_service
 {
 uint8_t num_of_services;
 uint8_t reserved[3];
 inc_service_t services[RSI_BLE_MAX_RESP_LIST];
 } rsi_ble_resp_inc_services_t;
```

Description

This callback uses GATT include service response structure.

1. This callback function is called if the included service response is received from the module.
2. This callback has to be registered using rsi_ble_gatt_register_callbacks API.

3. resp_status, contains the response status (Success (0) or Error code).

Structure Variables

Variables	Data type	Description
num_of_services	uint8_t	This is the number of included services found
reserved	uint8_t	Reserved
services	inc_service_t	This is the list of included services. The maximum value is 5.

13.8.18 rsi_ble_on_att_desc_resp_t

Prototype

```
typedef void (*rsi_ble_on_att_desc_resp_t) (
 uint16_t resp_status,
 rsi_ble_resp_att_descs_t *rsi_ble_resp_att_desc);
```

Structure

```
typedef struct rsi_ble_resp_att_descs_s
 {
 uint8_t num_of_att;
 uint8_t reserved[3];
 att_desc_t att_desc[RSI_BLE_MAX_RESP_LIST];
 } rsi_ble_resp_att_descs_t;
```

Description

This callback function is called if the attribute descriptors response is received from the module.

This callback has to be registered using rsi_ble_gatt_register_callbacks API.

resp_status, contains the response status (Success (0) or Error code).

Structure Variables

Variables	Data type	Description
num_of_att	uint8_t	This is the number of descriptors found
reserved	uint8_t	Reserved
att_desc	att_desc_t	This is the attribute descriptors list. The maximum value is 5.

13.8.19 rsi_ble_on_read_resp_t

Prototype

```
typedef void (*rsi_ble_on_read_resp_t) (uint16_t resp_status,
 uint16_t resp_id,
 rsi_ble_resp_att_value_t *rsi_ble_resp_att_val);
```

Structure

```
typedef struct rsi_ble_resp_att_value_s
 {
 uint8_t len;
 uint8_t att_value[100];
 } rsi_ble_resp_att_value_t;
```

Description

This callback function is called upon receiving the attribute value

This callback has to be registered using rsi_ble_gatt_register_callbacks API.

resp_status, contains the response status (Success(0) or Error code).

Structure Variables

Variables	Data type	Description
len	uint8_t	This is the length of attribute value
att_value	uint8_t	This is the attribute values list. Each attribute value is maximum of size 30.

13.8.20 ble_on_write_resp

Prototype

```
typedef void (*rsi_ble_on_write_resp_t) (uint16_t resp_status,
 uint16_t resp_id);
```

Description

This callback function is called if the attribute set / prepare / execute action is completed.

This callback must be registered using rsi_ble_gatt_register_callbacks API.

resp_status, contains the response status (Success (0) or Error code).

Precondition
rsi_ble_connect() API needs to be called before this API.

Parameters

Parameters	Data type	Description
resp_id	uint16_t	This is the response ID for corresponding write command
status	uint16_t	This is the status of corresponding writes command.

GATT Event Callbacks

This callback function will be called if the GATT write / notify / indicate events are received.

This callback must be registered using rsi_ble_gatt_register_callbacks API.

13.8.21 GATT Write event

Prototype

```
typedef struct rsi_ble_event_write_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint8_t reserved;
 #define RSI_BLE_WRITE_CMD_EVENT      0x01
 #define RSI_BLE_WRITE_REQUEST_EVENT  0x02
 #define RSI_BLE_NOTIFICATION_EVENT   0x03
 #define RSI_BLE_INDICATION_EVENT     0x04
 uint8_t pkt_type;
 uint8_t handle[2];
 uint8_t length;
 uint8_t att_value[240];
 } rsi_ble_event_write_t;
 typedef void (*rsi_ble_on_gatt_write_event_t) (
 uint16_t event_id, rsi_ble_event_write_t *rsi_ble_write);
```

Description
This event callback uses GATT Write event structure.

Structure Variables

Variables	Data type	Description
dev_addr	uint8	This is the remote device address
reserved	uint8	This is reserved for future.
pkt_type	uint8	This is type of the event received from the remote device.

Variables	Data type	Description
handle	uint8	This is the attribute handle.
length	uint8	This is the length of attribute value
att_value	uint8	This contains the attribute value. The maximum value is 100.

13.8.22 GATT Prepare Write event

Prototype

```
typedef struct rsi_ble_event_prepare_write_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint8_t handle[2];
 uint8_t offset[2];
 uint16_t length;
 uint8_t att_value[100];
 } rsi_ble_event_prepare_write_t;
 typedef void (*rsi_ble_on_gatt_prepare_write_event_t) (uint16_t event_id,
rsi_ble_event_prepare_write_t *rsi_ble_write);
```

Description
This event callback uses GATT Prepare Write event structure.

Structure Variables

Variables	Data type	Description
dev_addr	uint8	This is the remote device address
handle	uint8	This is the attribute handle.
offset	uint8	This is the value offset
length	uint8	This is the length of attribute value
att_value	uint8	This contains the attribute value. The maximum value is 100.

13.8.23 GATT Execute Write event

Prototype

```
typedef struct rsi_ble_execute_write_s {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint8_t exeflag;
 } rsi_ble_execute_write_t;
 typedef void (*rsi_ble_on_execute_write_event_t) (uint16_t event_id, rsi_ble_execute_write_t
*rsi_ble_execute_write);
```

Description
This event callback uses GATT Execute Write event structure.

Structure Variables

Variables	Data type	Description
dev_addr	uint8	This is the remote device address
exeflag	uint8	This executes flag set after prepare writes are completely sent

13.8.24 GATT Read request event

Prototype

```
typedef struct rsi_ble_read_req_s {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint16_t handle;
 uint8_t type;
```

```
 uint8_t reserved;
 uint16_t offset;
 } rsi_ble_read_req_t;
 typedef void (*rsi_ble_on_read_req_event_t) (uint16_t event_id, rsi_ble_read_req_t
*rsi_ble_read_req);Description
```

This event callback uses GATT Read request event structure.

Structure Variables

Variables	Data type	Description
dev_addr	uint8	This is the remote device address
handle	uint16	This is the attribute handle.
type	uint8	0 – Read request 1 – Read Blob request
offset	uint16	This is the offset of attribute value to be read

13.8.25 MTU event

Prototype

```
typedef struct rsi_ble_event_mtu_s {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint16_t mtu_size;
 } rsi_ble_event_mtu_t
 typedef void (*rsi_ble_on_mtu_event_t) (rsi_ble_event_mtu_t *rsi_ble_event_mtu)
```

Description

This event callback uses MTU event structure.

Structure Variables

Variables	Data type	Description
dev_addr	uint8	This is the remote device address
mtu_size	uint16	This is the MTU size

Gatt Error Resp Structure

```
//RSI_BLE_EVENT_GATT_ERROR_RESP, event_id: 0x1500
typedef struct rsi_ble_event_error_resp_s {
   //!uint8_t[6], remote device address.
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
    //!uint8_t[2], attribute handle.
    uint8_t  handle[2];
    uint8_t  error[2];
} rsi_ble_event_error_resp_t;
typedef void (*rsi_ble_on_gatt_error_resp_t) (uint16_t event_status, rsi_ble_event_error_resp_t
*rsi_ble_gatt_error);
```

Description

This structure describes the format of Gatt Error Response.

Structure Variables

Variables	Description
dev_addr	Remote device address
handle	This is the Gatt Error handle
error	This error indicates the type of Gatt Error

GATT Descriptor value Event

```
//RSI_BLE_EVENT_GATT_CHAR_DESC - event_ix = 1501
typedef struct rsi_ble_event_gatt_desc_s {
   //!uint8_t[6], remote device address.
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
    //!uint8_t, number of descriptors found.
    uint8_t     num_of_att;
    uint8_t reserved;
    //!(uint8_t[24] * 5), attribute descriptors list.
    att_desc_t  att_desc[RSI_BLE_MAX_RESP_LIST];
} rsi_ble_event_gatt_desc_t ;
typedef void (*rsi_ble_on_gatt_desc_val_event_t) (uint16_t event_status, rsi_ble_event_gatt_desc_t
*rsi_ble_gatt_desc_val);
```

Description

- This callback function is called if the attribute descriptors event is received from the module.
- This callback must be registered using rsi_ble_gatt_register_callbacks API

Structure Variables

Variables	Description
dev_addr	Remote Device Address
num_of_att	This is the number of descriptors found
reserved	Reserved
att_desc	This is the attribute descriptors list. The maximum value is 5.

GATT Primary Services Event:

```
//RSI_BLE_EVENT_GATT_PRIMARY_SERVICE_LIST, event_id: 0x1509
typedef struct rsi_ble_event_profiles_list_s {
   //!uint8_t[6], remote device address.
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
   //!uint8_t, number of profiles found.
    uint8_t number_of_profiles;
    uint8_t reserved;
    //!(uint8_t[24] * 5), list of found profiles.
    profile_descriptors_t  profile_desc[RSI_BLE_MAX_RESP_LIST];
} rsi_ble_event_profiles_list_t;
typedef void (*rsi_ble_on_event_profiles_list_t) (uint16_t event_status, rsi_ble_event_profiles_list_t
*rsi_ble_event_profiles);
```

Description

This callback function is called if the profiles list event is received from the module.

This callback must be registered using rsi_ble_gatt_register_callbacks API.

Structure Variables

Variables	Description
dev_addr	Remote Device Address
number_of_profiles	This is the number of profiles
reserved	Reserved
profile_desc	This is the attribute descriptors list. The maximum value is 5.

Get Profile by UUID Event:

```
//RSI_BLE_EVENT_GATT_PRIMARY_SERVICE_BY_UUID, event_id = 0x1502
typedef struct rsi_ble_event_profile_by_uuid_s {
   //!uint8_t[6], remote device address.
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
    //!uint8_t[2], profile start handle
    uint8_t  start_handle[2];
```

```
    //!uint8_t[2], profile end handle
    uint8_t  end_handle[2];
} rsi_ble_event_profile_by_uuid_t;
typedef void (*rsi_ble_on_event_profile_by_uuid_t) (uint16_t event_status,
rsi_ble_event_profile_by_uuid_t *rsi_ble_event_profile);
```

Description

This callback function is called if the profile event is received from the module.

This callback must be registered using rsi_ble_gatt_register_callbacks API.

Structure Variables

Variables	Description
dev_addr	Remote Device Address
start_handle	This is the start handle.
end_handle	This is the end handle.

Get Char Services Event :

```
//RSI_BLE_EVENT_GATT_READ_CHAR_SERVS, event_id = 0x1503
typedef struct rsi_ble_event_read_by_type1_s {
    //!uint8_t[6], remote device address.
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
   //!uint8_t, number of characteristic services found.
    uint8_t      num_of_services;
    uint8_t reserved;
       //!(uint8_t[28] * 5), characteristic service list.
    char_serv_t  char_services[RSI_BLE_MAX_RESP_LIST];
}rsi_ble_event_read_by_type1_t;
typedef void (*rsi_ble_on_event_read_by_char_services_t) (uint16_t event_status,
rsi_ble_event_read_by_type1_t  *rsi_ble_event_read_type1);
```

Description

This callback function will be called if the service characteristics event is received from the module.

This callback has to be registered using rsi_ble_gatt_register_callbacks API.

Structure Variables

Variables	Description
dev_addr	Remote Device Address
num_of_services	This is the number of char services found
reserved	Reserved
char_services	It contains the characteristic service list. The maximum value is 5.

Get Include Services Event :

```
//RSI_BLE_EVENT_GATT_READ_INC_SERVS, event_id = 0x1504
typedef struct rsi_ble_event_read_by_type2_s {
   //!uint8_t[6], remote device address.
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
   //!uint8_t, number of characteristic services found.
    uint8_t      num_of_services;
    uint8_t reserved;
    //!(uint8_t[28] * 5), include service list.
    inc_serv_t  services[RSI_BLE_MAX_RESP_LIST];
} rsi_ble_event_read_by_type2_t;
typedef void (*rsi_ble_on_event_read_by_inc_services_t) (uint16_t event_status,
rsi_ble_event_read_by_type2_t *rsi_ble_event_read_type2);
```

Description

This callback uses GATT include service event structure.

1. This callback function is called if the included service event is received from the module.
2. This callback has to be registered using rsi_ble_gatt_register_callbacks API.

Structure Variables

Variables	Description
dev_addr	Remote Device Address
num_of_services	This is the number of include services found
reserved	Reserved
services	This is the list of included services. The maximum value is 5.

Read Attribute value by UUID Event :

```
//RSI_BLE_EVENT_GATT_READ_VAL_BY_UUID, event_id = 0x1505
typedef struct rsi_ble_event_read_by_type3_s {
    //!uint8_t[6], remote device address.
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
    //!uint8_t[2], attribute handle.
    uint8_t  handle[2];
    //!uint8_t, length of attribute value.
    uint16_t  length;
    //!uint8_t[50], attribute value.
    uint8_t  att_value[RSI_DEV_ATT_LEN];
} rsi_ble_event_read_by_type3_t;
typedef void (*rsi_ble_on_event_read_att_value_t) (uint16_t event_status, rsi_ble_event_read_by_type3_t
*rsi_ble_event_read_type3);
```

Description

This callback function is called upon receiving the attribute value.

This callback has to be registered using rsi_ble_gatt_register_callbacks API.

Structure Variables

Variables	Description
dev_addr	Remote Device Address
handle	This is the attribute handle
length	This is the length of attribute value
att_value	This contains the attribute value. The maximum value is 100

Read Respone Event:

```
//RSI_BLE_EVENT_GATT_READ_RESP , evet_id = 0x1506,0x1507,0x1508
typedef struct rsi_ble_event_att_value_s {
    //!uint8_t[6], remote device address.
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
    //!uint8_t, length of attribute value.
    uint16_t  length;
    //!uint8_t[50], attribute value.
    uint8_t  att_value[RSI_DEV_ATT_LEN];
} rsi_ble_event_att_value_t;
typedef void (*rsi_ble_on_event_read_resp_t) (uint16_t event_status, rsi_ble_event_att_value_t
*rsi_ble_event_att_val);
```

Description

This event callback uses GATT Read/Multiple/Blob request response event structure**.**

Structure Variables

Variables	Description
dev_addr	Remote Device Address

Variables	Description
length	This is the length of attribute value
att_val	This contains the attribute value. The maximum value is 100.

Write Respone Event:

```
//RSI_BLE_EVENT_GATT_WRITE_RESP, event_id: 0x150A,0x150C
typedef struct rsi_ble_set_att_resp_s {
    //!uint8_t[6], remote device address.
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_ble_set_att_resp_t;
typedef void (*rsi_ble_on_event_write_resp_t) (uint16_t event_status, rsi_ble_set_att_resp_t
*rsi_ble_event_set_att_rsp);
```

Description

This event callback uses GATT Write /Execute Write event structure.

Structure Variables

Variables	Description
dev_addr	Remote Device Address

Prepare Write Respone Event:

```
//RSI_BLE_EVENT_GATT_PREPARE_WRITE_RESP, event_id: 0x150B
typedef struct rsi_ble_prepare_write_resp_s {
    //!uint8_t[6], remote device address.
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
    //!uint8_t[2], attribute handle.
    uint8_t  handle[2];
    //!uint8_t[2], attribute value offset.
  uint8_t  offset[2];
    //!uint8_t, length of attribute value.
    uint16_t  length;
    //!uint8_t[50], attribute value.
    uint8_t  att_value[RSI_DEV_ATT_LEN];
} rsi_ble_prepare_write_resp_t;
typedef void (*rsi_ble_on_event_prepare_write_resp_t) (uint16_t event_status,
rsi_ble_prepare_write_resp_t  *rsi_ble_event_prepare_write);
```

Description

This structure describes the format of GATT Prepare Write event.

Structure Variables

Variables	Description
dev_addr	Remote device address
handle	This is the attribute handle
offset	This is the value offset
length	This is the length of attribute value
att_value	This contains the attribute value. The maximum value is 100.

13.9 l2cap cbsc register callbacks

This function is used to registers L2CAP CBSC callback functions.

13.9.1 rsi_ble_l2cap_cbsc_register_callbacks

Prototype

```
void rsi_ble_l2cap_cbsc_register_callbacks (
     rsi_ble_on_cbfc_conn_req_event_t   ble_on_cbsc_conn_req,
     rsi_ble_on_cbfc_conn_complete_event_t   ble_on_cbsc_conn_complete,
     rsi_ble_on_cbfc_rx_data_event_t    ble_on_cbsc_rx_data,
     rsi_ble_on_cbfc_disconn_event_t    ble_on_cbsc_disconn);
```

Description

This function registers the function pointers for GATT responses.

Parameters

ble_on_cbsc_conn_req	callback function for CBFC connection request event
ble_on_cbsc_conn_complete	callback function for CBFC connection complete status event
ble_on_cbsc_rx_data	callback function for CBFC data receive event
ble_on_cbsc_disconn	callback function for CBFC disconnect event

Return Values

None

rsi_ble_l2cap_cbsc Callbacks Declarations

13.9.2 rsi_ble_on_cbfc_conn_req_event_t

Prototype

```
typedef struct rsi_ble_event_cbfc_conn_req_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
  uint16_t psm;
  uint16_t lcid;
} rsi_ble_event_cbfc_conn_req_t;
typedef void (*rsi_ble_on_cbfc_conn_req_event_t) (rsi_ble_event_cbfc_conn_req_t
*rsi_ble_cbfc_conn_req);
```

Description

This callback function will be called when connected to indicate connection request.

Structure Variables

dev_addr	This is the device address of the disconnected device.
psm	protocol service multiplexer
lcid	l2cap channel ID

13.9.3 rsi_ble_on_cbfc_conn_complete_event_t

Prototype

```
typedef struct rsi_ble_event_cbfc_conn_complete_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
  uint16_t psm;
  uint16_t mtu;
  uint16_t mps;
  uint16_t lcid;
} rsi_ble_event_cbfc_conn_complete_t;
typedef void (*rsi_ble_on_cbfc_conn_complete_event_t) (rsi_ble_event_cbfc_conn_complete_t
*rsi_ble_cbfc_conn_complete, uint16_t status);
```

Description

This callback function will be called when connected to indicate connection complete status.

Structure Variables

dev_addr	This is the device address of the disconnected device.
psm	protocol service multiplexer
mtu	maximum transmit unit
mps	maximum payload size
lcid	l2cap channel ID

13.9.4 rsi_ble_on_cbfc_rx_data_event_t

Prototype

```
typedef struct rsi_ble_event_cbfc_rx_data_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
  uint16_t lcid;
  uint16_t len;
  uint8_t  data[RSI_DEV_ATT_LEN];
} rsi_ble_event_cbfc_rx_data_t;
typedef void (*rsi_ble_on_cbfc_rx_data_event_t) (rsi_ble_event_cbfc_rx_data_t *rsi_ble_cbfc_rx_data);
```

Description

This callback function will be called when connected to indicate received data.

Structure Variables

dev_addr	This is the device address of the disconnected device.
lcid	l2cap channel ID
len	data length
data	recieved data

13.9.5 rsi_ble_on_cbfc_disconn_event_t

Prototype

```
typedef struct rsi_ble_event_cbfc_disconn_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
  uint16_t lcid;
} rsi_ble_event_cbfc_disconn_t;
typedef void (*rsi_ble_on_cbfc_disconn_event_t) (rsi_ble_event_cbfc_disconn_t *rsi_ble_cbfc_disconn)
```

Description

This callback function will be called when connected to indicate disconnect l2cap connection

Structure Variables

dev_addr	This is the device address of the disconnected device.
lcid	l2cap channel ID

SMP Register Callbacks

This function is used to register the call-back functions for an asynchronous SMP events.

13.9.6 rsi_ble_smp_register_callbacks

Prototype

```
void rsi_ble_smp_register_callbacks (
 rsi_ble_on_smp_request_t ble_on_smp_request_event,
```

```
  rsi_ble_on_smp_response_t ble_on_smp_response_event,
  rsi_ble_on_smp_passkey_t ble_on_smp_passkey_event,
  rsi_ble_on_smp_failed_t ble_on_smp_fail_event,
  rsi_ble_on_encrypt_started_t  rsi_ble_on_encrypt_started_event,
  rsi_ble_on_smp_passkey_display_t  ble_on_smp_passkey_display_event,
  rsi_ble_on_sc_passkey_t         ble_sc_passkey_event,
  rsi_ble_on_le_ltk_req_event_t  ble_on_le_ltk_req_event,
  rsi_ble_on_le_security_keys_t  ble_on_le_security_keys_event);
```

Description

This API registers SMP callbacks.

Parameters

Variables	Description
ble_on_smp_request_event	This is the SMP request callback
ble_on_smp_response_event	This is the SMP response callback
ble_on_smp_passkey_event	This is the SMP passkey callback
ble_on_smp_fail_event	This is the SMP failed callback
rsi_ble_on_encrypt_started_event	This is the SMP encryption callback
ble_on_smp_passkey_display_event	callback function for smp display (i/o capability diaply only) events
ble_sc_passkey_event	callback function for smp display (i/o capability diaply only) events
ble_on_le_ltk_req_event	This is the SMP ltk request callback
ble_on_le_security_keys_event	This is the SMP security keys callback

Return Values

None

> **Note:**
>
> For more information about each callback, please refer to SMP callbacks description section.

SMP event Callbacks Declarations

13.9.7 rsi_ble_on_smp_request_t

Prototype

```
typedef struct rsi_bt_event_smp_req_s {
 //!uint8_t, address of remote device
 uint8_t dev_addr[6];
 } rsi_bt_event_smp_req_t;
 typedef void (*rsi_ble_on_smp_request_t) (rsi_bt_event_smp_req_t *remote_dev_address);
```

Description

This callback is called when SMP request is received from the remote device.

Structure Variables

Variables	Description
dev_addr	This is the device address of the advertising device.

13.9.8 rsi_ble_on_smp_response_t

Prototype

```
typedef struct rsi_bt_event_smp_resp_s {
 //!uint8_t, address of remote device
 uint8_t dev_addr[6];
 } rsi_bt_event_smp_resp_t;
 typedef void (*rsi_ble_on_smp_response_t) (rsi_bt_event_smp_resp_t *remote_dev_address);
```

Description

This callback is called when SMP response is received from the remote device.

Structure Variables

Variables	Description
dev_addr	This is the device address of the connected device.

13.9.9 rsi_ble_on_smp_passkey_t

Prototype

```
typedef struct rsi_bt_event_smp_passkey_s {
 //!uint8_t, address of remote device
 uint8_t dev_addr[6];
 } rsi_bt_event_smp_passkey_t;
 typedef void (*rsi_ble_on_smp_passkey_t) (rsi_bt_event_smp_passkey_t *remote_dev_address);
```

Description

This callback is called when SMP passkey is received from the remote device.

Structure Variables

Variables	Description
dev_addr	This is the device address of the disconnected device.

13.9.10 rsi_ble_on_smp_failed_t

Prototype

```
typedef struct rsi_bt_event_smp_failed_s {
 //!uint8_t, address of remote device
 uint8_t dev_addr[6];
 } rsi_bt_event_smp_failed_t;
 typedef void (*rsi_ble_on_smp_failed_t) (uint16_t resp_status, rsi_bt_event_smp_failed_t
*remote_dev_address);
```

Description

This callback function is called if the SMP process is failed with remote device.

Structure Variables

Variables	Description
resp_status	This is the response status whether success or error
dev_addr	This is the device address of the disconnected device.

13.9.11 rsi_ble_on_encrypt_started_t

Prototype

```
typedef void (*rsi_ble_on_encrypt_started_t) (uint16_ resp_status,rsi_bt_event_encryption_enabled_t
*enc_enabled);


//!encryption enabled structure
typedef struct rsi_bt_event_encryption_enabled_s {
  //!uint8_t, address of the remote device encrypted
```

```
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
  uint8_t  enabled;
  uint8_t  sc_enable;
  uint16_t localediv;
  uint8_t  localrand[8];
  uint8_t  localltk[16];
  uint8_t  dev_addr_type;
} rsi_bt_event_encryption_enabled_t;
```

Description
This callback function is called if the encryption process is started with the remote device.

Structure Variables

Variables	Description
resp_status	This is the response status

13.9.12 rsi_ble_on_smp_passkey_display_t

Prototype

```
typedef void (*rsi_ble_on_smp_passkey_display_t ) (rsi_bt_event_smp_passkey_display_t
*smp_passkey_display);
//SMP passkey display event structure
typedef struct rsi_bt_event_smp_passkey_display_s {
        //!uint8_t, address of remote device
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
    uint8_t  passkey[6];
} rsi_bt_event_smp_passkey_display_t;
```

Description
This callback function is called if the passkey display process is started with the remote device.

Structure Variables

Variables	Description
smp_passkey_display	SMP passkey display response structure

13.9.13 rsi_ble_on_sc_passkey_t

Prototype

```
typedef void (*rsi_ble_on_sc_passkey_t ) (rsi_bt_event_sc_passkey_t *sc_passkey);
//SMP passkey display event structure
typedef struct rsi_bt_event_sc_passkey_s {
        //!uint8_t, address of remote device
    uint8_t   dev_addr[RSI_DEV_ADDR_LEN];
    uint8_t   reserved[2];
    uint32_t  passkey;
} rsi_bt_event_sc_passkey_t
```

Description
This callback function is called if the sc passkey process is started with the remote device.

Structure Variables

Variables	Description
sc_passkey	SMP sc passkey response structure

13.9.14 rsi_ble_on_le_ltk_req_event_t

Prototype

```
typedef void (*rsi_ble_on_le_ltk_req_event_t) (rsi_bt_event_le_ltk_request_t
*rsi_ble_event_le_ltk_request);
//le ltk request event Structure
typedef struct rsi_bt_event_le_ltk_request_s {
  //!uint8_t, address of the remote device encrypted
  uint8_t   dev_addr[RSI_DEV_ADDR_LEN];
  uint16_t  localediv;
  uint8_t   localrand[8];
  uint8_t   dev_addr_type;
} rsi_bt_event_le_ltk_request_t;
```

Description

This Callback function to be called if le ping payload timeout expired event is received.

Structure Variables

Variables	Description
rsi_ble_event_le_ltk_request	SMP le ltk response structure

13.9.15 rsi_ble_on_le_security_keys_t

Prototype

```
typedef void (*rsi_ble_on_le_security_keys_t) (rsi_bt_event_le_security_keys_t
*rsi_ble_event_le_security_keys);
//le security keys event Structure
typedef struct rsi_bt_event_le_security_keys_s {
    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
    uint8_t  local_irk[16];
    uint8_t  remote_irk[16];
    uint16_t remote_ediv;
    uint8_t  remote_rand[16];
    uint8_t  remote_ltk[16];
    uint8_t  Identity_addr_type;
    uint8_t  Identity_addr[6];
    uint8_t  dev_addr_type;
} rsi_bt_event_le_security_keys_t;
```

Description

This Callback function to be called if le security keys event is received.

Structure Variables

Variables	Description
rsi_ble_event_le_security_keys	SMP security keys response structure

13.9.16 rsi_ble_on_phy_update_complete_t

Prototype

```
typedef struct rsi_ble_event_phy_update_s {

    uint8_t  dev_addr[6];

    uint8_t  Status;

    uint8_t  TxPhy;

    uint8_t  RxPhy;

} rsi_ble_event_phy_update_t;

 typedef void (*rsi_ble_on_phy_update_complete_t) (rsi_ble_event_phy_update_t
*rsi_ble_event_phy_update_complete);
```

Description

This callback function will be called when phy update complete event is received.

Structure Variables

Variables	Description
dev_addr	address of the remote device
Status	Status of operation
TxPhy	Transmission PHY rate
TxPhy	Reception PHY rate

13.9.17 rsi_ble_on_data_length_update_t

Prototype

```
typedef struct rsi_ble_event_data_length_update_s

{

    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];

    uint16_t  MaxTxOctets;

  uint16_t  MaxTxTime;

  uint16_t  MaxRxOctets;

  uint16_t  MaxRxTime;

} rsi_ble_event_data_length_update_t;

typedef void (*rsi_ble_on_data_length_update_t) (rsi_ble_event_data_length_update_t
*remote_dev_address);
```

Description

This callback function will be called when data length update complete event is received.

Structure Variables

Variables	Description
dev_addr	Device address of the remote device.
MaxTxOctets	Maximum Tx Octets to be transmitted
MaxTxTime	Maximum Tx time to transmit the MaxTxOctets
MaxRxOctets	Maximum Rx Octets to be received
MaxRxTime	Maximum Rx tome to receive the MaxRxOctets

13.9.18 rsi_ble_on_directed_adv_report_event_t

Prototype

```
typedef struct rsi_ble_event_directedadv_report_s

{

    uint16_t event_type;

    uint8_t dev_addr_type;

    uint8_t dev_addr[RSI_DEV_ADDR_LEN];

    uint8_t directed_addr_type;
```

```
    uint8_t directed_dev_addr[RSI_DEV_ADDR_LEN];

    int8_t rssi;

}rsi_ble_event_directedadv_report_t;

typedef void (*rsi_ble_on_directed_adv_report_event_t) (rsi_ble_event_directedadv_report_t
*rsi_ble_event_directed);
```

Description

This callback function will be called if the advertise event report is received from the module

Structure Variables

Variables	Description
event_type	This is the device address of the disconnected device.
dev_addr_type	Address type of remote device
dev_addr	Address of the remote device
directed_addr_type	Directed address type
rssi	rssi value

13.9.19 rsi_ble_on_enhance_connect_t:

Prototype

```
typedef struct rsi_ble_event_enhnace_conn_status_s

{

uint8_t dev_addr_type;

uint8_t dev_addr[RSI_DEV_ADDR_LEN];

uint8_t peer_resolvlable_addr[RSI_DEV_ADDR_LEN];

 uint8_t local_resolvlable_addr[RSI_DEV_ADDR_LEN];

 uint8_t status;

}rsi_ble_event_enhance_conn_status_t;


typedef void (*rsi_ble_on_enhance_connect_t) (rsi_ble_event_enhance_conn_status_t
*rsi_ble_event_enhance_conn);
```

Description

This callback function will be called if the BLE enhanced connection status is received from the module.

Structure Variables

Variables	Description
dev_addr_type	Address type of remote device
dev_addr	Address of the remote device
peer_resolvlable_addr	resolvable address of the peer device
local_resolvlable_addr	resolvable address of the local device

13.9.20 rsi_ble_event_cbfc_conn_req_t

Prototype

```
typedef struct rsi_ble_event_cbfc_conn_req_s

{

  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];

  uint16_t psm;

  uint16_t lcid;

} rsi_ble_event_cbfc_conn_req_t;


typedef void (*rsi_ble_on_cbfc_conn_req_event_t) (rsi_ble_event_cbfc_conn_req_t
*rsi_ble_cbfc_conn_req);
```

Description

This event is used to notify the PSM connection request to host.

Structure Variables

Variables	Description
Remote-BDAddress	BD Address of the remote device
Status	0 - Success Non-zero – Failure
Psm	Protocol/Service Multiplexer (PSM) This field helps to indicate the protocol
Lcid	local device channel identifier

13.9.21 rsi_ble_event_cbfc_conn_complete_t

Prototype

```
typedef struct rsi_ble_event_cbfc_conn_complete_s

{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];

  uint16_t psm;

  uint16_t mtu;

  uint16_t mps;

  uint16_t lcid;

} rsi_ble_event_cbfc_conn_complete_t;


typedef void (*rsi_ble_on_cbfc_conn_complete_event_t) (rsi_ble_event_cbfc_conn_complete_t
*rsi_ble_cbfc_conn_complete, uint16_t status);
```

Description

This event is used to notify the PSM connection request to host.

Structure Variables

Variables	Description
Remote-BDAddress	BD Address of the remote device
Status	0 - Success

Variables	Description
	Non-zero – Failure
Psm	Protocol/Service Multiplexer (PSM) This field helps to indicate the protocol
MTU	Maximum transmission unit that device can transmit
MPS	Maximum payload size, this is the maximum size of l2cap packet that can be transmitted.
Lcid	local device channel identifier

13.9.22 rsi_ble_event_cbfc_rx_data_t

Prototype

```
typedef struct rsi_ble_event_cbfc_rx_data_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
  uint16_t lcid;
  uint16_t len;
  uint8_t  data[RSI_DEV_ATT_LEN];
} rsi_ble_event_cbfc_rx_data_t;
typedef void (*rsi_ble_on_cbfc_rx_data_event_t) (rsi_ble_event_cbfc_rx_data_t *rsi_ble_cbfc_rx_data);
```

Description

This event is used to notify the remote device specific PSM data has been transferred to Host.

Structure Variables

Variables	Description
Remote-BDAddress	BD Address of the remote device
Lcid	local device channel identifier
Data length	length of data transferred
Data	Actual data that got transfered

13.9.23 rsi_ble_event_cbfc_disconn_t

Prototype

```
typedef struct rsi_ble_event_cbfc_disconn_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
  uint16_t lcid;
} rsi_ble_event_cbfc_disconn_t;

typedef void (*rsi_ble_on_cbfc_disconn_event_t) (rsi_ble_event_cbfc_disconn_t *rsi_ble_cbfc_disconn);
```

Description

This function is used to notify the disconnect event to host.

Structure Variables

Variables	Description
dev_addr	BD Address of the remote device
Lcid	local device channel identifier

13.9.24 rsi_bt_event_le_ltk_request

Prototype

```
typedef struct rsi_bt_event_le_ltk_request_s {

  //!uint8_t, address of the remote device encrypted

  uint8_t   dev_addr[RSI_DEV_ADDR_LEN];

  uint16_t  localediv;

  uint8_t   localrand[8];

} rsi_bt_event_le_ltk_request_t;
```

Description

This functions is called if ltk request is received from remote device

Structure Variables

Variables	Description
dev_addr	BD Address of the remote device
localediv	ediv of local device
localrand	rand of local device

13.10 Configuration parameters

This section explains the configuration of macros which may be needed to be changed based on the application requirement. These macros with default values are placed in "**rsi_ble_config.h**".

13.10.1 Configure advertise parameters

Define	Meaning
RSI_BLE_ADV_TYPE	UNDIR_CONN: Advertising will be visible to all the devices. Scanning/Connection is also accepted from all devices. DIR_CONN: Advertising will be visible to the particular device mentioned in RSI_BLE_ADV_DIR_ADDR only. Scanning and Connection will be accepted from that device only. UNDIR_SCAN: Advertising will be visible to all the devices. Scanning will be accepted from all the devices. The connection will be not be accepted from any device. UNDIR_NON_CONN: Advertising will be visible to all the devices. Scanning and Connection will not be accepted from any device.
RSI_BLE_ADV_FILTER_TYPE	ALLOW_SCAN_REQ_ANY_CONN_REQ_ANY ALLOW_SCAN_REQ_WHITE_LIST_CONN_REQ_ANY ALLOW_SCAN_REQ_ANY_CONN_REQ_WHITE_LIST ALLOW_SCAN_REQ_WHITE_LIST_CONN_REQ_WHITE_LIST
RSI_BLE_ADV_DIR_ADDR_TYPE	LE_RANDOM_ADDRESS: For random address LE_PUBLIC_ADDRESS: For fixed address

13.10.2 Configure scan parameters

Define	Meaning
RSI_BLE_SCAN_TYPE	SCAN_TYPE_ACTIVE: Also receives scan response data SCAN_TYPE_PASSIVE: Don't receive scan response data
RSI_BLE_SCAN_FILTER_TYPE	SCAN_FILTER_TYPE_ALL: Accepts all advertisers SCAN_FILTER_TYPE_ONLY_WHITE_LIST: Accept white list advertisers

13.10.3 Configure connection parameters

Define	Meaning
LE_SCAN_INTERVAL	The interval between the start of two consecutive scan windows. It shall be a multiple of 0.625 ms Recommended value: 0x0100 (0x0100*0.625 = 160ms) Range: 0x0050 to 0x1000
LE_SCAN_WINDOW	During scanning, the Link Layer listens on an advertising channel index for the duration of the scan window. It shall be a multiple of 0.625 ms. Recommended value: 0x0050 (0x0050*0.625 = 50ms) Range: 0x0050 to 0x1000
CONNECTION_INTERVAL_MIN	The start of connection events ar e spaced regularly with an interval of connInterval. It shall be a multiple of 1.25 ms. Recommended value: 0x0050 (0x0050*0.625 = 50ms) Range: 0x0050 to 0x320
CONNECTION_INTERVAL_MAX	The start of connection events are spaced regularly with an interval of connInterval. It shall be a multiple of 1.25 ms. Recommended value: 0x0050 (0x0050*0.625 = 50ms) Range: 0x0050 to 0x320
CONNECTION_LATENCY	The conn_latency parameter defines the maximum allowed connection Latency. Recommended value: 0x0000
SUPERVISION_TIMEOUT	The supervision_tout parameter defines the link supervision timeout for the connection. Recommended value: 0x07D0 (*10) ms 0x07D0 (0x07D0 *10=20s) Range: 0x64 to 0xC80

13.10.4 Configuration for White list based on Advertising payload data

Define	Meaning
RSI_BLE_ENABLE_WHITELIST_BASEDON_ADV_PAYLOAD	To enable the feature
RSI_BLE_ADV_PAYLOAD_INDEX_FOR_COMPARE	Index ID in the advertising payload data that have the advertiser i.e. in a advertisers advertising data (in 32bytes), from which index need to compare
RSI_BLE_ADV_PAYLOAD_LENGTH_FROM_INDEX_TO_COMPARE	Length of the payload to compare i.e. from index to how much length to compare

14 Crypto APIs

This Section explains Crypto APIs to encrypt or decrypt the given data with the key provided.

14.1 rsi_aes

Prototype

```
int32_t rsi_aes(uint16_t aes_mode, uint16_t enc_dec, uint8_t *msg, uint16_t msg_length,
            uint8_t *key, uint16_t key_length, uint8_t *iv, uint8_t *output)
```

Description

This API encrypts or decrypts the input data with Key provided. This API provides provision for encryption with AES engine in three modes (ECB, CBC, CTR).

Parameters should be provided to API depending on mode of usage. API will return failure if there is an error in input.

Parameters

Parameter	Description
aes_mode	1 – For AES CBC mode 2 – For AES ECB mode 3 – For AES CTR mode
enc_dec	1 – For AES Encryption 2 – For AES Decryption
msg	Pointer to message to encrypt/decrypt
msg_length	Total message length in bytes
key	Pointer to AES key.
key_length	AES key length in bytes.
iv	Pointer to AES IV
output	Output parameter to hold encrypted/decrypted from AES

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0xFF15, 0xCC9C, 0xCC9B

Example

```
See <SW Package Path>/host/sapis/examples/crypto/AES/
status = rsi_aes(CBC_MODE, AES_ENCRYPTION, msg, sizeof(msg), key, AES_KEY_SIZE_256, iv,
aes_encry_data);
```

14.2 rsi_exponentiation

Prototype

```
int32_t rsi_exponentiation(uint8_t *prime, uint32_t prime_length, uint8_t *base, uint32_t base_length,
                    uint8_t *exponent, uint32_t exponent_length, uint8_t *exp_result)
```

Description

This API is used to calculate the DH key.

Parameters

Parameter	Description
prime	pointer to the prime
prime_length	Length of the prime in bytes
base	pointer to base
base_length	Length of the base in bytes
exponent	pointer to exponent
exponent_length	Length of the exponent in bytes
exp_result	Output exponentiation result

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0xFF15, 0xCC9C, 0xCC9B

Example

```
See <SW Package Path>/host/sapis/examples/crypto/DH/
status = rsi_exponentiation(prime, sizeof(prime), base, sizeof(base), exp, sizeof(exp), exp_result );
```

14.3 rsi_ecdh_point_multiplication

Prototype

```
int32_t rsi_ecdh_point_multiplication(uint8_t ecdh_mode, uint8_t *d, uint8_t *sx, uint8_t *sy,
                           uint8_t *sz, uint8_t *rx, uint8_t *ry, uint8_t *rz)
```

Description

This API is used to compute the ECDH point multiplication vector.

Parameters

Parameter	Description
ecdh_mode	1 – For ECDH 192 2 – For ECDH 224 3 – For ECDH 256
d	Pointer to scalar value that needs to be multiplied
sx, sy, sz	Pointers to x, y, z coordinates of the point to be multiplied with scalar ‘d’
rx, ry, rz	Pointers to x, y, z coordinates of the resultant vector

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0xFF15, 0xCC9C, 0xCC9B

Example

```
See <SW Package Path>/host/sapis/examples/crypto/ECDH/
status = rsi_ecdh_point_multiplication(ECDH_256, pm_d, pm_sx, pm_sy, pm_sz, rx, ry, rz);
```

14.4 rsi_ecdh_point_addition

Prototype

```
int32_t rsi_ecdh_point_addition(uint8_t ecdh_mode, uint8_t *sx, uint8_t *sy, uint8_t *sz, uint8_t *tx,
                      uint8_t *ty, uint8_t *tz, uint8_t *rx, uint8_t *ry, uint8_t *rz)
```

Description
This API is used to compute the ECDH point addition vector.

Parameters

Parameter	Description
ecdh_mode	1 – For ECDH 192 2 – For ECDH 224 3 – For ECDH 256
sx, sy, sz	Pointers to x, y, z coordinates of the point1 that needs to be added
tx, ty, tz	Pointers to x, y, z coordinates of the point2 that needs to be added
rx, ry, rz	Pointers to x, y, z coordinates of the resultant vector

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0xFF15, 0xCC9C, 0xCC9B

Example

```
See <SW Package Path>/host/sapis/examples/crypto/ECDH/
status = rsi_ecdh_point_addition(ECDH_256, pa_sx, pa_sy, pa_sz, pa_tx, pa_ty, pa_tz, rx, ry, rz);
```

14.5 rsi_ecdh_point_subtraction

Prototype

```
int32_t rsi_ecdh_point_subtraction(uint8_t ecdh_mode, uint8_t *sx, uint8_t *sy, uint8_t *sz, uint8_t
*tx,
                          uint8_t *ty, uint8_t *tz, uint8_t *rx, uint8_t *ry, uint8_t *rz)
```

Description

This API is used to compute the ECDH point subtraction vector.

Parameters

Parameter	Description
ecdh_mode	1 – For ECDH 192 2 – For ECDH 224 3 – For ECDH 256
sx, sy, sz	Pointers to x, y, z coordinates of the point1 that needs to be subtracted
tx, ty, tz	Pointers to x, y, z coordinates of the point2 that needs to be subtracted
rx, ry, rz	Pointers to x, y, z coordinates of the resultant vector

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0xFF15, 0xCC9C, 0xCC9B

Example

```
See <SW Package Path>/host/sapis/examples/crypto/ECDH/
status = rsi_ecdh_point_subtraction(ECDH_256, pa_sx, pa_sy, pa_sz, pa_tx, pa_ty, pa_tz, rx, ry, rz);
```

14.6 rsi_ecdh_point_double

Prototype

```
int32_t rsi_ecdh_point_double(uint8_t ecdh_mode, uint8_t *sx, uint8_t *sy, uint8_t *sz, uint8_t *rx,
uint8_t *ry, uint8_t *rz)
```

Description

This API is used to compute the ECDH point double vector.

Parameters

Parameter	Description
ecdh_mode	1 – For ECDH 192 2 – For ECDH 224 3 – For ECDH 256
sx, sy, sz	Pointers to x, y, z coordinates of the point that needs to be doubled
rx, ry, rz	Pointers to x, y, z coordinates of the resultant vector

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0xFF15, 0xCC9C, 0xCC9B

Example

```
See <SW Package Path>/host/sapis/examples/crypto/ECDH/
status = rsi_ecdh_point_double(ECDH_256, pd_sx, pd_sy, pd_sz, rx, ry, rz);
```

14.7 rsi_ecdh_point_affine

Prototype

```
int32_t rsi_ecdh_point_affine(uint8_t ecdh_mode, uint8_t *sx, uint8_t *sy, uint8_t *sz, uint8_t *rx,
uint8_t *ry, uint8_t *rz)
```

Description
This API is used to compute the ECDH point affinify vector.

Parameters

Parameter	Description
ecdh_mode	1 – For ECDH 192 2 – For ECDH 224 3 – For ECDH 256
sx, sy, sz	Pointers to x, y, z coordinates of the point that needs to perform affinify
rx, ry, rz	Pointers to x, y, z coordinates of the resultant vector

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0xFF15, 0xCC9C, 0xCC9B

Example

```
See <SW Package Path>/host/sapis/examples/crypto/ECDH/
status = rsi_ecdh_point_affine(ECDH_192, sx, sy, sz, rx, ry, rz);
```

14.8 rsi_hmac_sha

Prototype

```
int32_t rsi_hmac_sha(uint8_t hmac_sha_mode, uint8_t *msg, uint32_t msg_length, uint8_t *key,
                uint32_t key_length, uint8_t *digest, uint8_t *hmac_buffer)
```

Description
This API computes the HMAC-SHA digest for the given input message.

Parameters

Parameter	Description
hmac_sha_mode	1 – For HMAC-SHA1 2 – For HMAC-SHA256 3 – For HMAC-SHA384 4 – For HMAC-SHA512
msg	Pointer to message input
msg_length	Total message length in bytes
key	Pointer to key.
key_length	key length in bytes.
hmac_buffer	Buffer to hold the key and message together
digest	Output parameter to hold computed digest from HMAC-SHA

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0xFF15, 0xCC9C, 0xCC9B

Example

```
See <SW Package Path>/host/sapis/examples/crypto/HMAC/
status = rsi_hmac_sha(HMAC_SHA_512, msg, sizeof(msg), key, sizeof(key), digest, hmac_buf);
```

14.9 rsi_sha

Prototype

```
int32_t rsi_sha(uint8_t sha_mode, uint8_t *msg, uint16_t msg_length, uint8_t *digest)
```

Description
This API computes the SHA digest for the given input message.

Parameters

Parameter	Description
sha_mode	1 – For SHA1 2 – For SHA256 3 – For SHA384 4 – For SHA512
msg	Pointer to message input
msg_length	Total message length in bytes

Parameter	Description
key	Pointer to key.
key_length	key length in bytes.
digest	Output parameter to hold computed digest from HMAC-SHA

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid parameters -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0xFF15, 0xCC9C, 0xCC9B

Example

```
See <SW Package Path>/host/sapis/examples/crypto/SHA/
status = rsi_sha(SHA_512, SHA, strlen(SHA), digest);
```

15 Driver APIs

This section explains the Driver APIs to initialize the driver and also to handle the events.

15.1 Device Init

15.1.1 rsi_bl_module_power_cycle

Prototype

```
int16_t rsi_bl_module_power_cycle(void);
```

Description
This API power cycles the module. This API is valid only if there is a power gate, external to the module, which is controlling the power to the module using a GPIO signal of the MCU.

Precondition
rsi_driver_init() must be called before this API.

Parameters

None

Return Values

Value	Description
0	Success
Non Zero Value	-1: Failure

Example

```
status = rsi_bl_module_power_cycle();
```

15.2 Driver

15.2.1 rsi_interrupt_handler

Prototype

```
void rsi_interrupt_handler(void);
```

Description
This API is used to handle the interrupt coming from the module.

Precondition
N/A

Parameters
None

Return Values
None

Example

```
rsi_interrupt_handler();
```

15.2.2 rsi_mask_ta_interrupt

Prototype

```
void rsi_mask_ta_interrupt(void);
```

Description
This API masks the TA interrupt.

Precondition
N/A

Parameters
None

Return Values
None

Example

```
rsi_mask_ta_interrupt();
```

15.2.3 rsi_unmask_ta_interrupt

Prototype

void rsi_unmask_ta_interrupt(void);

Description
This API unmasks the TA interrupt.

Precondition
N/A

Parameters
None

Return Values
None

Example

```
rsi_unmask_ta_interrupt();
```

15.3 Events

15.3.1 rsi_set_event

Prototype

```
void rsi_set_event(uint32_t event_num)
```

Description
This API is used to set an event.

Precondition
N/A

Parameters

Parameter	Description
event_num	Event number to be set

Return Values
None

Example

```
rsi_set_event(event_num);
```

15.3.2 rsi_clear_event

Prototype

```
void rsi_clear_event(uint32_t event_num)
```

Description
This API is used to clear an event.

Precondition
N/A

Parameters

Parameter	Description
event_num	Event number to clear

Return Values
None

Example

```
rsi_clear_event(event_num);
```

15.3.3 rsi_mask_event

Prototype

```
void rsi_mask_event(uint32_t event_num)
```

Description
This API is used to mask an event specified using the event number.

Precondition
N/A

Parameters

Parameter	Description
event_num	Event number to mask

Return Values
None

Example

```
uint32_t event_num;
rsi_mask_event(event_num);
```

15.3.4 rsi_unmask_event

Prototype

```
void rsi_unmask_event(uint32_t event_num)
```

Description
This API is used to unmask an event.

Precondition
N/A

Parameters

Parameter	Description
event_num	Event number to unmask

Return Values
None

Example

```
uint32_t event_num;
rsi_unmask_event(event_num);
```

15.3.5 rsi_unmask_event_from_isr

Prototype

```
void rsi_unmask_event_from_isr(uint32_t event_num);
```

Description
This API is used to unmask an event from ISR context.

Precondition
N/A

Parameters

Parameter	Description
event_num	Event number to unmask

Return Values
None

Example

```
uint32_t event_num;
rsi_unmask_event_from_isr(event_num);
```

15.3.6 rsi_find_event

Prototype

```
uint32_t rsi_find_event(uint32_t event_map)
```

Description
This API is used to find the event which is set from the event map.

Precondition
N/A

Parameters

Parameter	Description
event_map	Event map

Return Values

None

Example

```
rsi_find_event(event_map);
```

15.3.7 rsi_register_event

Prototype

```
uint16_t rsi_register_event(uint32_t event_id, void (*event_handler_ptr)(void))
```

Description
This API is used to register the event.

Precondition
N/A

Parameters

Parameter	Description
event_id	event number which needs to be registered
event_handler_ptr	event handler which needs to be registered for a given event

Return Values

Parameter	Description
0	success
1	Error

Example

```
status = rsi_register_event(event_id,event_handler_ptr);
```

15.3.8 rsi_set_event_from_isr

Prototype

```
void rsi_set_event_from_isr(uint32_t event_num)
```

Description
This API is used to set the event from isr context.

Precondition
N/A

Parameters

Parameter	Description
event_num	Event number to mask

Return Values
None

Example

```
rsi_set_event_from_isr(event_num);
```

15.3.9 rsi_events_init

Prototype

```
void rsi_events_init(void)
```

Description
This API is used to initialize the events.

Precondition
N/A

Parameters

None

Return Values
None

Example

```
rsi_events_init();
```

15.4 Nwk

15.4.1 rsi_wlan_get_nwk_status

Prototype

```
int32_t rsi_wlan_get_nwk_status(void)
```

Description
This API returns wlan status.

Precondition
NA

Parameters

None

Return Values

wlan status

Example

```
status = rsi_wlan_get_nwk_status();
```

15.5 Set Region AP

15.5.1 extract_setregionap_country_info

Prototype

```
void extract_setregionap_country_info(rsi_req_set_region_ap_t *rsi_set_region_ap)
```

Description
This API is used to initialize the global parameter structure with parameters used in set region ap command based on the region.

Precondition
NA

Parameters

Parameter	Description
rsi_set_region_ap	Pointer to the global parameter structure

Return Values

None

Example

```
rsi_req_set_region_ap_t *rsi_set_region_ap;
extract_setregionap_country_info(rsi_set_region_ap);
```

15.6 Timer

15.6.1 rsi_timer_expiry_interrupt_handler

Prototype

```
void rsi_timer_expiry_interrupt_handler(void);
```

Description
This API is called for every millisecond and increments the timer counter.

Precondition
NA

Parameters

None

Return Values

None

Example

```
rsi_timer_expiry_interrupt_handler();
```

15.6.2 rsi_timer_read_counter

Prototype

```
uint32_t rsi_timer_read_counter(void);
```

Description
This API returns the timer counter value.

Precondition
NA

Parameters

None

Return Values

Timer counter value in milliseconds

Example

```
rsi_timer_read_counter();
```

15.6.3 rsi_init_timer

Prototype

```
void rsi_init_timer(rsi_timer_instance_t *rsi_timer, uint32_t duration);
```

Description
This API initializes the timer instance with the expiry time.

Precondition
NA

Parameters

Parameter	Description
rsi_timer	timer instance
duration	duration in milliseconds

Return Values

None

Example

```
rsi_timer_instance_t *rsi_timer;
uint32_t duration;
rsi_init_timer(rsi_timer, duration);
```

15.6.4 rsi_timer_expired

Prototype

```
int32_t rsi_timer_expired(rsi_timer_instance_t *timer);
```

Description
This API is used to check whether the timer instance is expired or not.

Precondition
NA

Parameters

Parameter	Description
rsi_timer	timer instance

Return Values

Value	Description
1	If timer is expired
0	If timer is not expired

Example

```
rsi_timer_instance_t *rsi_timer;
rsi_timer_expired(rsi_timer);
```

15.6.5 rsi_timer_left

Prototype

```
uint32_t rsi_timer_left(rsi_timer_instance_t *timer);
```

Description
This API is used to get the remaining time for timer expiry.

Precondition
NA

Parameters

Parameter	Description
timer	timer instance

Return Values

Value	Description
>0	Time left to expire

Value	Description
0	If timer is expired

Example

```
rsi_timer_instance_t *timer;
rsi_timer_left(timer);
```

15.7 Utils

15.7.1 rsi_uint16_to_2bytes

Prototype

```
void rsi_uint16_to_2bytes(uint8_t *dBuf, uint16_t val);
```

Description
This API is used to convert uint16 to two byte array.

Precondition
NA

Parameters

Parameter	Description
dBuf	Pointer to buffer to put the data in
val	Data to convert

Return Values

None

Example

```
rsi_uint16_to_2bytes(dBuf, val);
```

15.7.2 rsi_uint32_to_4bytes

Prototype

```
void rsi_uint32_to_4bytes(uint8_t *dBuf, uint16_t val);
```

Description
This API is used to convert uint32 to four byte array.

Precondition
NA

Parameters

Parameter	Description
dBuf	Pointer to buffer to put the data in
val	Data to convert

Return Values

None

Example

```
rsi_uint32_to_4bytes(dBuf, val);
```

15.7.3 rsi_bytes2R_to_uint16

Prototype

```
uint16_t rsi_bytes2R_to_uint16(uint8_t *dBuf);
```

Description
This API is used to convert a 2 byte array to uint16, first byte in array is LSB.

Precondition
NA

Parameters

Parameter	Description
dBuf	Pointer to a buffer to get the data from

Return Values
uint16 converted data

Example

```
uint8_t recv_buffer[RECV_BUFFER_SIZE];
rsi_bytes2R_to_uint16(&recv_buffer[1]);
```

15.7.4 rsi_bytes4R_to_uint32

Prototype

```
uint32_t rsi_bytes4R_to_uint32(uint8_t *dBuf);
```

Description
This API is used to convert a 4 byte array to uint32, first byte in array is LSB.

Precondition
NA

Parameters

Parameter	Description
dBuf	Pointer to a buffer to get the data from

Return Values

uint32 converted data

Example

```
uint8_t *buf;
rsi_bytes4R_to_uint32(buf);
```

15.7.5 rsi_ascii_hex2num

Prototype

```
int8_t rsi_ascii_hex2num(int8_t ascii_hex_in);
```

Description
This API is used for ASCII to hex conversion.

Precondition
NA

Parameters

Parameter	Description
ascii_hex_in	ASCII hex input

Return Values

hex number

Example

```
int8_t ascii_hex_in;
rsi_ascii_hex2num(ascii_hex_in);
```

15.7.6 rsi_char_hex2dec

Prototype

```
int8_t rsi_char_hex2dec (int8_t *cBuf);
```

Description
This API is used to convert given ASCII hex notation to decimal notation (used for mac address).

Precondition
NA

Parameters

Parameter	Description
cBuf	ASCII hex notation string

Return Value

Integer value

Example

```
int8_t cBuf;
rsi_char_hex2dec(cBuf);
```

15.7.7 rsi_ascii_dev_address_to_6bytes_rev

Prototype

```
uint8_t* rsi_ascii_dev_address_to_6bytes_rev(uint8_t *hex_addr, int8_t *ascii_mac_address);
```

Description
This API is used to convert notation MAC address to a 6-byte hex address.

Precondition
NA

Parameters

Parameter	Description
ascii_mac_address	Source address to convert, must be a null terminated string. This is input parameter.
hex_addr	Converted hex address is returned here. This is output parameter.

Return Value

Hex address

Example

```
#define REMOTE_BD_ADDR  "4c:4f:ee:4f:1f:b9"
int8_t *hex_addr;
rsi_ascii_dev_address_to_6bytes_rev(hex_addr, REMOTE_BD_ADDR));
```

15.7.8 rsi_6byte_dev_address_to_ascii

Prototype

```
uint8_t* rsi_6byte_dev_address_to_ascii(uint8_t *ascii_mac_address, uint8_t *hex_addr);
```

Description
This API is used to convert given 6-byte hex address to ASCII Mac address.

Precondition
NA

Parameters

Parameter	Description
hex_addr	Hex address input. This is input parameter.
ascii_mac_address	Converted ASCII mac address is returned here. This is output parameter.

Return Values

ASCII mac address

Example

```
int8_t *hex_addr;
rsi_6byte_dev_address_to_ascii(ascii_mac_address, hex_addr);
```

15.7.9 lmac_crc8_c

Prototype

```
uint8_t lmac_crc8_c(uint8_t crc8_din, uint8_t crc8_state, uint8_t end);
```

Description
This API is used to calculate crc for a given byte and accumulate crc.

Precondition
NA

Parameters

Parameter	Description
crc8_din	crc byte input
crc8_state	accumulated crc
end	last byte crc

Return Values

6bit crc

Example

```
lmac_crc8_c(crc8_din, crc8_state, end);
```

15.7.10 multicast_mac_hash

Prototype

```
uint8_t multicast_mac_hash(uint8_t *mac);
```

Description
This API is used to is to calculate 6-bit hash value for given mac address.

Precondition
NA

Parameters

Parameter	Description
mac	pointer to mac address

Return Values

6bit hash value

Example

```
uint8_t *mac;
multicast_mac_hash(mac);
```

15.7.11 convert_lower_case_to_upper_case

Prototype

```
uint8_t convert_lower_case_to_upper_case(uint8_t lwrcase);
```

Description
This API is used to convert the given lower-case character to upper case.

Precondition
NA

Parameters

Parameter	Description
lwrcase	Lower case character to convert

Return Values

Converted Upper case character

Example

```
uint8_t lwrcase;
```

convert_lower_case_to_upper_case(lwrcase);

15.7.12 string2array

Prototype

```
void string2array(uint8_t *dst, uint8_t *src, uint32_t length);
```

Description
This API is used to convert the given string to destination array.

Precondition
NA

Parameters

Parameter	Description
dst	Pointer to destination array
src	Pointer to source string
length	Length of the string

Return Values

None

Example

```
string2array(dst, src, length);
```

15.7.13 rsi_itoa

Prototype

```
uint8_t* rsi_itoa(uint32_t val, uint8_t *str);
```

Description
This API is used to convert integer value into null-terminated string and stores the result in the array given by str parameter.

Precondition
NA

Parameters

Parameter	Description
val	Value to be converted to a string
str	Array in memory where to store the resulting null-terminated string

Return Values

String

Example

```
uint32_t val;
uint8_t str[7] = {0};
rsi_itoa(val, str);
```

15.7.14 rsi_atoi

Prototype

```
int32_t rsi_atoi(const int8_t* str);
```

Description
This API is used to convert string to an integer.

Precondition
NA

Parameters

Parameter	Description
str	This is the string representation of an integral number

Return Values

Converted Integer

Example

```
uint8_t str[20];
rsi_atoi(str);
```

15.7.15 asciihex_2_num

Prototype

```
int8_t asciihex_2_num(int8_t ascii_hex_in);
```

Description
This API is used for ASCII to hex conversion.

Precondition
NA

Parameters

Parameter	Description
ascii_hex_in	ASCII hex input

Return Values

hex number

Example

```
int8_t ascii_hex_in;
asciihex_2_num(ascii_hex_in);
```

15.7.16 rsi_charhex_2_dec

Prototype

```
int8_t rsi_charhex_2_dec (int8_t *cBuf);
```

Description
This API is used to convert given ASCII hex notation to decimal notation (used for mac address).

Precondition
NA

Parameters

Parameter	Description
cBuf	ASCII hex notation string. This is input parameter.

Return Values

Integer value

Example

```
int8_t cBuf;
rsi_charhex_2_dec(cBuf);
```

15.7.17 rsi_ascii_mac_address_to_6bytes

Prototype

```
void rsi_ascii_mac_address_to_6bytes(uint8_t *hexAddr, int8_t *asciiMacAddress);
```

Description
This API is used to convert notation MAC address to a 6-byte hex address.

Precondition
NA

Parameters

Parameter	Description
asciiMacAddress	source address to convert, must be a null terminated string. This is input parameter
hexAddr	Converted hex address is returned here. This is output parameter.

Return Values

Hex address

Example

```
#define asciiMacAddress  "4c:4f:ee:4f:1f:b9"
int8_t *hex_addr;
rsi_ascii_dev_address_to_6bytes_rev(hex_addr, asciiMacAddress));
```

15.7.18 rsi_ascii_dot_address_to_4bytes

Prototype

```
void rsi_ascii_dot_address_to_4bytes(uint8_t *hexAddr, int8_t *asciiDotAddress);
```

Description
This API is used to convert notation network address to 4-byte hex address.

Precondition
NA

Parameters

Parameter	Description
asciiDotAddress	source address to convert, must be a null terminated string. This is input parameter
hexAddr	Output value is passed back in the 4-byte Hex Address. This is output parameter.

Return Values

None

Example

```
int8_t *USER_CFG_IPV4_ADDRESS;
int8_t *hex_addr;
rsi_ascii_dot_address_to_4bytes(hex_addr, USER_CFG_IPV4_ADDRESS);
```

15.7.19 ip_to_reverse_hex

Prototype

```
uint64_t ip_to_reverse_hex(char *ip);
```

Description
This API is used to convert IP address to reverse Hex format.

Precondition
NA

Parameters

Parameter	Description
ip	IP address to convert. This is input parameter

Return Values

IP address in reverse Hex format

Example

```
char *ip;
ip_to_reverse_hex(ip);
```

15.8 Wlan

15.8.1 rsi_wlan_cb_init

Prototype

```
int8_t rsi_wlan_cb_init(rsi_wlan_cb_t *wlan_cb);
```

Description

This function initializes the WLAN control block structure.

Precondition

NA

Parameters

Parameter	Description
wlan_cb	Pointer to WLAN cb structure

Structure Prototype

```
//! driver WLAN control block
typedef struct rsi_wlan_cb_s
{
  //! driver wlan control block state
  volatile rsi_wlan_state_t state;

  //! Auto config state
  volatile uint8_t auto_config_state;

  //! driver wlan control block status
  volatile int32_t status;

  //! driver wlan control block mutex
  rsi_mutex_handle_t wlan_mutex;

  //! driver wlan control block expected command response
  rsi_wlan_cmd_response_t expected_response;

  //! driver wlan control block semaphore
  rsi_semaphore_handle_t wlan_sem;

  //! driver wlan control block tx packet pool
  rsi_pkt_pool_t  wlan_tx_pool;

  //! socket id of socket which is waiting for socket create response
  uint8_t  waiting_socket_id;

  //! buffer pointer given by application to driver
  uint8_t *app_buffer;

  //! buffer length given by application to driver
  uint32_t app_buffer_length;

  //! driver wlan control block requested query command
  uint8_t query_cmd;

  //! validity
  uint16_t field_valid_bit_map;

  //! mac address
  uint8_t mac_address[6];

  //! opermode
  uint16_t opermode;
}rsi_wlan_cb_t;
```

Return Values

Value	Description
0	Success
Non Zero Value	Failure

Example

```
rsi_wlan_cb_t *wlan_cb;
status = rsi_wlan_cb_init(wlan_cb);
```

15.8.2 rsi_driver_wlan_send_cmd

Prototype

```
int32_t rsi_driver_wlan_send_cmd(rsi_wlan_cmd_request_t cmd, rsi_pkt_t *pkt);
```

Description

This function fills commands and places into wlan TX queue.

Precondition

NA

Parameters

Parameter	Description
cmd	Type of the command to send
pkt	Pointer of the packet to send

Structure Prototype

```
typedef enum rsi_wlan_cmd_request_e
{
#ifdef RSI_CHIP_MFG_EN
  RSI_RESET_MAC                     = 0x01,
  RSI_RADIO_CAPS                    = 0x02,
  RSI_SCAN_REQ                      = 0x06,
  RSI_BOOTUP_PARAMS                 = 0x0c,
  RSI_VAP_CAPS                      = 0x0d,
  RSI_FLASH_WRITE                   = 0x0f,
  RSI_BB_RF_INIT                    = 0x15,
  RSI_PER_CMD_PKT                   = 0x19,
  RSI_FEATURE_FRAME                 = 0x33,
#endif
  RSI_WLAN_REQ_BAND                 = 0x11,
  RSI_WLAN_REQ_INIT                 = 0x12,
  RSI_WLAN_REQ_SCAN                 = 0x13,
  RSI_WLAN_REQ_JOIN                 = 0x14,
  RSI_WLAN_REQ_CONFIG               = 0xBE,
  RSI_WLAN_REQ_SET_SLEEP_TIMER      = 0x16,
  RSI_WLAN_REQ_SET_MAC_ADDRESS      = 0x17,
  RSI_WLAN_REQ_QUERY_NETWORK_PARAMS = 0x18,
#ifndef RSI_CHIP_MFG_EN
  RSI_WLAN_REQ_DISCONNECT           = 0x19,
#endif
  RSI_WLAN_REQ_SET_REGION           = 0x1D,
  RSI_WLAN_REQ_CFG_SAVE             = 0x20,
  RSI_WLAN_REQ_AUTO_CONFIG_ENABLE   = 0x21,
  RSI_WLAN_REQ_GET_CFG              = 0x22,
  RSI_WLAN_REQ_USER_STORE_CONFIG    = 0x23,
  RSI_WLAN_REQ_AP_CONFIGURATION     = 0x24,
  RSI_WLAN_REQ_SET_WEP_KEYS         = 0x25,
  RSI_WLAN_REQ_PING_PACKET          = 0x29,
  RSI_WLAN_REQ_SET_PROFILE          = 0x31,
  RSI_WLAN_REQ_GET_PROFILE          = 0x32,
#ifndef RSI_CHIP_MFG_EN
  RSI_WLAN_REQ_DELETE_PROFILE       = 0x33,
```

```
#endif
  RSI_WLAN_REQ_RSSI                   = 0x3A,
  RSI_WLAN_REQ_IPCONFV4               = 0x41,
  RSI_WLAN_REQ_SOCKET_CREATE          = 0x42,
  RSI_WLAN_REQ_SOCKET_CLOSE           = 0x43,
  RSI_WLAN_REQ_EAP_CONFIG             = 0x4C,
  RSI_WLAN_REQ_FW_VERSION             = 0x49,
  RSI_WLAN_REQ_MAC_ADDRESS            = 0x4A,
  RSI_WLAN_REQ_QUERY_GO_PARAMS        = 0x4E,
  RSI_WLAN_REQ_SOCKET_READ_DATA       = 0x6B,
  RSI_WLAN_REQ_SOCKET_ACCEPT          = 0x6C,
  RSI_WLAN_REQ_IPCONFV6               = 0x90,
  RSI_WLAN_REQ_HOST_PSK               = 0xA5,
  RSI_WLAN_REQ_SET_MULTICAST_FILTER   = 0x40,
  RSI_WLAN_REQ_GAIN_TABLE             = 0x47,
  RSI_WLAN_REQ_SET_CERTIFICATE        = 0x4D,
  RSI_WLAN_REQ_DNS_QUERY              = 0x44,
  RSI_WLAN_REQ_DNS_UPDATE             = 0xED,
  RSI_WLAN_REQ_CONNECTION_STATUS      = 0x48,
  RSI_WLAN_REQ_CONFIGURE_P2P          = 0x4B,
  RSI_WLAN_REQ_HTTP_CLIENT_GET        = 0x51,
  RSI_WLAN_REQ_HTTP_CLIENT_POST       = 0x52,
  RSI_WLAN_REQ_HTTP_CLIENT_POST_DATA  = 0xEB,
  RSI_WLAN_REQ_HTTP_CLIENT_PUT        = 0x53,
  RSI_WLAN_REQ_DNS_SERVER_ADD         = 0x55,
  RSI_WLAN_REQ_WIRELESS_FWUP          = 0x59,
  RSI_WLAN_REQ_BG_SCAN                = 0x6A,
  RSI_WLAN_REQ_HT_CAPABILITIES        = 0x6D,
  RSI_WLAN_REQ_REJOIN_PARAMS          = 0x6F,
  RSI_WLAN_REQ_WPS_METHOD             = 0x72,
  RSI_WLAN_REQ_ROAM_PARAMS            = 0x7B,
  RSI_WLAN_REQ_TX_TEST_MODE           = 0x7C,
  RSI_WLAN_REQ_WMM_PS                 = 0x97,
  RSI_WLAN_REQ_FWUP                   = 0x99,
  RSI_WLAN_REQ_RX_STATS               = 0xA2,
  RSI_WLAN_REQ_SOCKET_CONFIG          = 0xA7,
  RSI_WLAN_REQ_MULTICAST              = 0xB1,
  RSI_WLAN_REQ_HTTP_ABORT             = 0xB3,
  RSI_WLAN_REQ_HTTP_CREDENTIALS       = 0xB4,
#ifdef RSI_WAC_MFI_ENABLE
  RSI_WLAN_REQ_ADD_MFI_IE             = 0xB5,
#endif
#ifdef RSI_CHIP_MFG_EN
  RSI_EFUSE_WRITE_REQ                 = 0xB6,
  RSI_EFUSE_READ_REQ                  = 0xB7,
#endif
#ifndef RSI_M4_INTERFACE
  RSI_WLAN_REQ_CERT_VALID             = 0xBC,
#endif
  RSI_WLAN_REQ_SET_REGION_AP          = 0xBD,
  RSI_WLAN_REQ_DYNAMIC_POOL           = 0xC7,
  RSI_WLAN_REQ_FILTER_BCAST_PACKETS   = 0xC9,
  RSI_WLAN_REQ_MDNSD                  = 0xDB,
  RSI_WLAN_REQ_FTP                    = 0xE2,
  RSI_WLAN_REQ_FTP_FILE_WRITE         = 0xE3,
  RSI_WLAN_REQ_SNTP_CLIENT            = 0xE4,
  RSI_WLAN_REQ_SMTP_CLIENT            = 0xE6,
  RSI_WLAN_REQ_OTA_FWUP               = 0xEF,
  RSI_WLAN_REQ_WEBPAGE_LOAD           = 0x50,
  RSI_WLAN_REQ_JSON_LOAD              = 0x9c,
  RSI_WLAN_REQ_WEBPAGE_ERASE          = 0x9A,
  RSI_WLAN_REQ_JSON_OBJECT_ERASE      = 0x9B,
  RSI_WLAN_REQ_WEBPAGE_CLEAR_ALL      = 0x7F,
  RSI_WLAN_REQ_HOST_WEBPAGE_SEND      = 0x56,
  RSI_WLAN_REQ_GET_RANDOM             = 0xF8,
  RSI_WLAN_REQ_POP3_CLIENT            = 0xE7,
  RSI_WLAN_REQ_DHCP_USER_CLASS        = 0xEC,
  RSI_WLAN_REQ_SELECT_REQUEST          = 0x74,
  RSI_WLAN_REQ_TIMEOUT                = 0xEA,
  RSI_WLAN_REQ_RADIO                  = 0x81,
  RSI_WLAN_REQ_GET_STATS              = 0xF1
```

```
}rsi_wlan_cmd_request_t;


typedef struct rsi_pkt_s
{
  //! next packet pointer
  struct rsi_pkt_s *next;

  //! host descriptor
  uint8_t desc[16];

  //! payload
  uint8_t data[1];
}rsi_pkt_t;
```

Return Values

Value	Description
0	Success
Non Zero Value	Failure

Example

```
status = rsi_driver_wlan_send_cmd(RSI_WLAN_REQ_FWUP, pkt);
```

15.8.3 rsi_extract_filename

Prototype

```
uint8_t* rsi_extract_filename(uint8_t* json, uint8_t* buffer);
```

Description

This function is used to extract filename out of the received json update data.

Precondition

NA

Parameters

Parameter	Description
json	json object data string
buffer	contains file name

Return Values

Returns file name extracted

Example

```
uint8_t filename[24] = {'\0'};
uint8_t *payload;
status = rsi_extract_filename(payload, filename);
```

15.8.4 rsi_driver_process_wlan_recv_cmd

Prototype

```
uint32_t rsi_driver_process_wlan_recv_cmd(rsi_pkt_t *pkt);
```

Description

This function processes WLAN receive commands.

Note: Memory allocation for the pointer is from receive handler, after process it will be freed .

Precondition

NA

Parameters

Parameter	Description
pkt	Pointer to received RX packet

Structure Prototype

```
typedef struct rsi_pkt_s
{
  //! next packet pointer
  struct rsi_pkt_s *next;

  //! host descriptor
  uint8_t desc[16];

  //! payload
  uint8_t data[1];
}rsi_pkt_t;
```

Return Values

Value	Description
0	Success
Non Zero Value	Failure

Example

```
rsi_pkt_t *rx_pkt;
status = rsi_driver_process_wlan_recv_cmd(rx_pkt);
```

15.8.5 rsi_wlan_check_waiting_socket_cmd

Prototype

```
int32_t rsi_wlan_check_waiting_socket_cmd(void);
```

Description

This function checks, if any socket command is in waiting state.

Precondition

NA

Parameters

None

Return Values

Value	Description
0	No socket command is in waiting state
1	socket command is in waiting state

Example

```
status = rsi_wlan_check_waiting_socket_cmd();
```

15.8.6 rsi_wlan_check_waiting_wlan_cmd

Prototype

```
int32_t rsi_wlan_check_waiting_wlan_cmd(void);
```

Description

This function checks if any WLAN command is in waiting state.

Precondition

NA

Parameters

None

Return Values

Value	Description
0	No WLAN command is in waiting state
1	No WLAN command is in waiting state

Example

```
status = rsi_wlan_check_waiting_wlan_cmd();
```

15.8.7 rsi_wlan_check_wlan_state

Prototype

```
uint32_t rsi_check_wlan_state(void);
```

Description

This function is used to get WLAN status.

Precondition

NA

Parameters

None

Return Values

Value	Description
0	RSI_WLAN_STATE_NONE
1	RSI_WLAN_STATE_OPERMODE_DONE
2	RSI_WLAN_STATE_BAND_DONE
3	RSI_WLAN_STATE_INIT_DONE
4	RSI_WLAN_STATE_SCAN_DONE
5	RSI_WLAN_STATE_CONNECTED
6	RSI_WLAN_STATE_IP_CONFIG_DONE
7	RSI_WLAN_STATE_IPV6_CONFIG_DONE
8	RSI_WLAN_STATE_AUTO_CONFIG_GOING_ON
9	RSI_WLAN_STATE_AUTO_CONFIG_DONE

Value	Description
10	RSI_WLAN_STATE_AUTO_CONFIG_FAILED

Example

```
status = rsi_check_wlan_state();
```

15.8.8 rsi_wlan_set_status

Prototype

```
void rsi_wlan_set_status(int32_t status);
```

Description

This API sets the WLAN status.

Precondition

NA

Parameters

Parameter	Description
status	Status value to be set

Return Values

None

Example

```
status = rsi_wlan_set_status(status);
```

15.8.9 rsi_post_waiting_semaphore

Prototype

```
int32_t rsi_post_waiting_semaphore(void);
```

Description

This API is used to post on a waiting semaphore.

Precondition

NA

Parameters

None

Return Values

Value	Description
0	Success
Non Zero Value	Failure

Example

```
status = rsi_post_waiting_semaphore();
```

15.8.10 rsi_wlan_process_raw_data

Prototype

```
void rsi_wlan_process_raw_data(rsi_pkt_t *pkt);
```

Description

This API is used to receive raw data packet from module.

Note:

Memory allocation for the pointer is from receive handler, after processing it will be freed.

Precondition

NA

Parameters

Parameter	Description
pkt	Pointer to rx packet

Structure Prototype

```
typedef struct rsi_pkt_s
{
  //! next packet pointer
  struct rsi_pkt_s *next;

  //! host descriptor
  uint8_t desc[16];

  //! payload
  uint8_t data[1];
}rsi_pkt_t;
```

Return Values

None

Example

```
rsi_pkt_t *pkt;
 rsi_wlan_process_raw_data(pkt);
```

15.8.11 rsi_wlan_packet_transfer_done

Prototype

```
void rsi_wlan_packet_transfer_done(rsi_pkt_t *pkt);
```

Description

This API is used to handle packet transfer completion.

Note:

Memory allocation for the pointer is from transfer handler, after processing it will be freed.

Precondition

NA

Parameters

Parameter	Description
pkt	Pointer to packet

Structure Prototype

```
typedef struct rsi_pkt_s
{
  //! next packet pointer
  struct rsi_pkt_s *next;

  //! host descriptor
  uint8_t desc[16];

  //! payload
  uint8_t data[1];
}rsi_pkt_t;
```

Return Values

None

Example

```
 rsi_wlan_packet_transfer_done(pkt);
```

15.8.12 rsi_check_wlan_buffer_full

Prototype

```
void rsi_check_wlan_buffer_full(rsi_pkt_t *pkt);
```

Description

This API will clear TX packet, if TX buffer is full.

> **Note**:
>
> BUFFER_FULL_HANDLING macro should be enabled.

Precondition

NA

Parameters

Parameter	Description
pkt	Buffer pointer

Structure Prototype

```
typedef struct rsi_pkt_s
{
  //! next packet pointer
  struct rsi_pkt_s *next;

  //! host descriptor
  uint8_t desc[16];

  //! payload
  uint8_t data[1];
```

```
}rsi_pkt_t;
```

Return Values

None

Example

```
rsi_check_wlan_buffer_full(pkt);
```

15.8.13 rsi_check_common_buffer_full

Prototype

```
void rsi_check_common_buffer_full(rsi_pkt_t *pkt);
```

Description

This API is used to check if the common buffer is full.

Precondition

NA

Parameters

Parameter	Description
pkt	Buffer pointer

Structure Prototype

```
typedef struct rsi_pkt_s
{
  //! next packet pointer
  struct rsi_pkt_s *next;

  //! host descriptor
  uint8_t desc[16];

  //! payload
  uint8_t data[1];
}rsi_pkt_t;
```

Return Values

None

Example

```
 rsi_check_common_buffer_full(pkt);
```

15.8.14 rsi_wait_on_wlan_semaphore

Prototype

```
rsi_error_t rsi_wait_on_wlan_semaphore(rsi_semaphore_handle_t *semaphore, uint32_t timeout_ms);
```

Description

This API is used by wireless library to acquire or wait for nwk semaphore.

Precondition

NA

Parameters

Parameter	Description
semaphore	Semaphore handle pointer
timeout_ms	Maximum time to wait to acquire semaphore. If timeout_ms is 0, then wait till semaphore is acquired. **Note**: Units of time is in milli second.

Return Values

Value	Description
0	Success
<0	Failure

Example

```
status = rsi_wait_on_wlan_semaphore(&rsi_driver_cb_non_rom->wlan_cmd_sem,
RSI_TIMEOUT_RESPONSE_WAIT_TIME);
```

15.8.15 rsi_update_wlan_cmd_state_to_free_state

Prototype

```
void rsi_update_wlan_cmd_state_to_free_state(void);
```

Description
This API is used by wireless library to update the WLAN command state to free state.

Precondition
NA

Parameters

None

Return Values

None

Example

```
rsi_update_wlan_cmd_state_to_free_state();
```

15.8.16 sort_index_based_on_rssi

Prototype

```
void sort_index_based_on_rssi(struct wpa_scan_results_arr *scan_results_array);
```

Description
This API is used to sort the scan list based on rssi value.

Note:

The sorted array will replace the original pointer to the results.

Precondition
NA

Parameters

Parameter	Description
scan_results_array	Pointer to scan results array

Structure Prototype

```
struct wpa_scan_res {
   int freq;
   int level;
        uint8_t channel_no;
        uint8_t security_mode;
        uint8_t rssiVal;
        uint8_t uNetworkType;
        uint8_t ssid[RSI_SSID_LEN];
        uint8_t ssid_len;
   uint8_t bssid[ETH_ALEN];
};


struct wpa_scan_results_arr {

        uint16_t num;
   uint16_t sort_index[MAX_SCAN_COUNT];
        struct wpa_scan_res res[MAX_SCAN_COUNT];
};
```

Return Values

None

Example

```
extern struct wpa_scan_results_arr *scan_results_array;
sort_index_based_on_rssi(scan_results_array);
```

15.8.17 process_scan_results

Prototype

```
int process_scan_results(uint8_t *buf, uint16_t len, int8_t rssi, uint8_t channel, uint16_t freq);
```

Description
This API is used to process received beacons and probe responses.

Precondition
NA

Parameters

Parameter	Description
buf	Received frame
len	Length of the buffer
rssi	RSSI value
channel	Channel in which the frame is received
freq	Frequency of the channel

Return Values

Value	Description
0	Success

Value	Description
Non Zero Value	Failure

Example

```
status = process_scan_results(buf, len, rssi, channel, freq);
```

15.9 SPI

15.9.1 rsi_nlink_pkt_rd

Prototype

```
int16_t rsi_nlink_pkt_rd(uint8_t *buf, uint16_t total_len)
```

Description

This API performs frame descriptor and payload read.

Precondition
NA

Parameters

Parameter	Description
buf	pointer to the buffer into which descriptor and payload has to be read
total_len	number of bytes to be read

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_nlink_pkt_rd(buf,total_len);
```

15.9.2 rsi_frame_read

Prototype

```
int16_t rsi_frame_read(uint8_t *pkt_buffer)
```

Description

This is a common function to read response for all the command and data from Wi-Fi module.

Precondition
NA

Parameters

Parameter	Description
pkt_buffer	pointer to the buffer to which packet has to be read

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_frame_read(pkt_buffer);
```

15.9.3 rsi_frame_write

Prototype

```
int16_t rsi_frame_write(rsi_frame_desc_t *uFrameDscFrame,uint8_t *payloadparam,uint16_t size_param)
```

Description

This is a common function used to process a command to the Wi-Fi module.

Precondition
NA

Parameters

Parameter	Description
uFrameDscFrame	frame descriptor
payloadparam	pointer to the command payload parameter structure
size_param	size of the payload for the command

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_frame_write(uFrameDscFrame,payloadparam,size_param);
```

15.9.4 rsi_pre_dsc_rd

Prototype

```
int16_t rsi_pre_dsc_rd(uint8_t *dbuf)
```

Description

This function performs pre frame descriptor read.

Precondition
NA

Parameters

Parameter	Description
dbuf	pointer to the buffer into which pre descriptor has to be read

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_pre_dsc_rd(dbuf);
```

15.9.5 rsi_pkt_rd

Prototype

```
int16_t rsi_pkt_rd(uint8_t *buf, uint16_t dummy_len, uint16_t total_len)
```

Description

This function performs frame descriptor and payload read.

Precondition
NA

Parameters

Parameter	Description
dbuf	pointer to the buffer into which descriptor and payload has to be read
dummy_len	number of dummy bytes which can be discarded
total_len	number of bytes to be read

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_pkt_rd(buf,dummy_len,total_len);
```

15.9.6 rsi_spi_frame_dsc_wr

Prototype

```
int16_t rsi_spi_frame_dsc_wr(rsi_frame_desc_t *uFrameDscFrame)
```

Description

This API is used to write a Frame descriptor.

Precondition
NA

Parameters

Parameter	Description
uFrameDscFrame	frame descriptor

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_spi_frame_dsc_wr(uFrameDscFrame);
```

15.9.7 rsi_spi_frame_data_wr

Prototype

```
int16_t rsi_spi_frame_data_wr(uint16_t bufLen, uint8_t *dBuf,uint16_t tbufLen,uint8_t *tBuf)
```

Description

This API is used to perform Data Frame Write.

Precondition
NA

Parameters

Parameter	Description
bufLen	length of the data buffer to write
dBuf	pointer to the buffer of data to write
tbufLen	length of the data fragment to write
tBuf	pointer to the buffer of data fragment to write

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_spi_frame_data_wr(bufLen,dBuf,tbufLen,tBuf);
```

15.9.8 rsi_send_c1c2

Prototype

```
int16_t rsi_send_c1c2(uint8_t c1, uint8_t c2)
```

Description

This API is used to send the C1 & C2 command bytes, should check response for C1 command. In case of busy should retry.

Precondition
NA

Parameters

Parameter	Description
c1	SPI c1 command bytes to be sent
c2	SPI c2 command bytes to be sent

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_send_c1c2(c1, c2);
```

15.9.9 rsi_send_c3c4

Prototype

```
int16_t rsi_send_c3c4(uint8_t c3, uint8_t c4)
```

Description

This API is used to send the C3 & C4 command bytes.

Precondition
rsi_send_c1c2 API execution should be successful before using this function.

Parameters

Parameter	Description
c3	SPI c3 command bytes to be sent
c4	SPI c3 command bytes to be sent

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_send_c3c4(c3,c4);
```

15.9.10 rsi_spi_wait_start_token

Prototype

```
int16_t rsi_spi_wait_start_token(uint32_t timeout,uint8_t mode)
```

Description

This API loops reading the SPI until a start token, 0x55, is received.

Precondition
Should issue read commands before using this function.

Parameters

Parameter	Description
timeout	Timeout for start token
mode	To indicate 8bit/32bit mode

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_spi_wait_start_token(timeout,mode);
```

15.9.11 rsi_set_intr_mask

Prototype

```
int16_t rsi_set_intr_mask(uint8_t interruptMask)
```

Description

This API is used to sets the INTERRUPT MASK REGISTER of the Wi-Fi module.

Precondition
NA

Parameters

Parameter	Description
interruptMask	the value to set the mask register to

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_set_intr_mask(interruptMask);
```

15.9.12 rsi_set_intr_type

Prototype

```
int16_t rsi_set_intr_type(uint32_t interruptMaskVal)
```

Description

This API sets the INTERRUPT TYPE of the Wi-Fi module based on selected LOAD_IMAGE.

Precondition
NA

Parameters

Parameter	Description
interruptMaskVal	the value to set the mask register

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_set_intr_type(interruptMaskVal);
```

15.9.13 rsi_clear_interrupt

Prototype

```
int16_t rsi_clear_interrupt(uint8_t interruptClear)
```

Description

This API is used to clear the interrupt register.

Precondition
NA

Parameters

Parameter	Description
interruptClear	the value to set the interrupt clear register to

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_clear_interrupt(interruptClear);
```

15.9.14 rsi_device_interrupt_status

Prototype

```
int16_t rsi_device_interrupt_status(uint8_t *int_status)
```

Description

This returns the value of the Interrupt register.

Precondition
NA

Parameters

Parameter	Description
int_status	pointer to the buffer of data to be read, assumed to be at least a byte

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_device_interrupt_status(int_status);
```

15.9.15 rsi_spi_pkt_len

Prototype

```
int16_t rsi_spi_pkt_len(uint16_t *length)
```

Description

This API returns the length of the register.

Precondition
NA

Parameters

Parameter	Description
length	pointer to the buffer of data to write, assumed to be at least 2 bytes long

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_spi_pkt_len(length);
```

15.9.16 rsi_spi_high_speed_enable

Prototype

```
int16_t rsi_spi_high_speed_enable(void)
```

Description

This API is used to program the module SPI to high speed mode.

Precondition
NA

Parameters

None

Return Values

Value	Description
0	Success

Example

```
status = rsi_spi_high_speed_enable();
```

15.9.17 rsi_spi_iface_init

Prototype

```
int16_t rsi_spi_iface_init(void)
```

Description

This API initializes the Wi-Fi module's Slave SPI interface.

Precondition
NA

Parameters

None

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_spi_iface_init();
```

15.9.18 rsi_ulp_wakeup_init

Prototype

```
void rsi_ulp_wakeup_init(void)
```

Description

This API initializes the Wi-Fi module's Slave SPI interface on ulp wakeup.

Precondition
NA

Parameters

None

Return Values

None

Example

```
rsi_ulp_wakeup_init();
```

15.9.19 rsi_mem_wr

Prototype

```
int16_t rsi_mem_wr(uint32_t addr, uint16_t len, uint8_t *dBuf)
```

Description

This API is used to perform a memory write to the Wi-Fi module.

Precondition
NA

Parameters

Parameter	Description
addr	address to write in the module
len	number of bytes to write
dBuf	pointer to the buffer of data to write

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_mem_wr(addr,len, dBuf);
```

15.9.20 rsi_mem_rd

Prototype

```
int16_t rsi_mem_rd(uint32_t addr, uint16_t len, uint8_t *dBuf)
```

Description

This API Performs a memory read from the Wi-Fi module.

Precondition
NA

Parameters

Parameter	Description
addr	address to read from
len	number of bytes to read
dBuf	pointer to the buffer to receive the data into

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_mem_rd(addr, len, dBuf);
```

15.9.21 rsi_reg_rd

Prototype

```
int16_t rsi_reg_rd(uint8_t regAddr, uint8_t *dBuf)
```

Description

This API reads from a register in the Wi-Fi module from the address specified.

Precondition
NA

Parameters

Parameter	Description
regAddr	address of the register from which data is to be read, address is 6-bits, upper 2 bits must be cleared
dBuf	pointer to the buffer of data to write, assumed to be at least 2 bytes long

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_reg_rd(regAddr, dBuf);
```

15.9.22 rsi_reg_rd2

Prototype

```
int16_t rsi_reg_rd2(uint8_t regAddr, uint8_t *dBuf)
```

Description

This API reads from a register in the Wi-Fi module from the address specified.

Precondition
NA

Parameters

Parameter	Description
regAddr	address of the register from which data is to be read, address is 6-bits, upper 2 bits must be cleared
dBuf	pointer to the buffer of data to write, assumed to be at least 2 bytes long

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_reg_rd2(regAddr, dBuf);
```

15.9.23 rsi_reg_wr

Prototype

```
int16_t rsi_reg_wr(uint8_t regAddr, uint8_t *dBuf)
```

Description

This API is used to write to a register in the Wi-Fi module with an address specified.

Precondition
NA

Parameters

Parameter	Description
regAddr	address of the spi register to be written
dBuf	pointer to the buffer of data to write, assumed to be at least 2 bytes long

Return Values

Value	Description
0	Success
-1	SPI busy / Timeout
-2	SPI Failure

Example

```
status = rsi_reg_wr(regAddr, dBuf);
```

15.10 BT BLE

15.10.1 rsi_bt_common_register_callbacks

Prototype

```
void rsi_bt_common_register_callbacks (rsi_bt_get_ber_pkt_t rsi_bt_get_ber_pkt_from_app)
```

Description
This function registers the bt-common callbacks.

Precondition
NA

Parameters

Parameter	Description
rsi_bt_get_ber_pkt_from_app	BER Call back

Return Values

Value	Description
0	Success
Non Zero Value	Failure

Example

```
rsi_bt_common_register_callbacks (rsi_bt_get_ber_pkt_t rsi_bt_get_ber_pkt_from_app)
```

15.10.2 rsi_bt_get_proto_type

Prototype

```
uint16_t rsi_bt_get_proto_type(uint16_t rsp_type, rsi_bt_cb_t **bt_cb)
```

Description
This function determines the BT protocol (BT COMMON / BT classic / BLE) using the packet type.

Precondition
NA

Parameters

Parameter	Description
rsp_type	packet type
bt_cb	BT control back

Return Values

Returns the protocol type (BT COMMON / BT classic / BLE)

Example

```
status =rsi_bt_get_proto_type(uint16_t rsp_type, rsi_bt_cb_t **bt_cb)
```

15.10.3 rsi_bt_get_timeout

Prototype

```
uint32_t rsi_bt_get_timeout(uint16_t cmd_type, uint16_t protocol_type)
```

Description
This function calculates semaphore wait time for a protocol (BT COMMON / BT classic / BLE) using the packet type.

Precondition
NA

Parameters

Parameter	Description
cmd_type	Command Type
protocol_type	Protocol type , whether it is BT Common/BT Classic/BLE

Return Values

Returns semaphore wait time

Example

```
status = rsi_bt_get_timeout(uint16_t cmd_type, uint16_t protocol_type)
```

15.10.4 rsi_bt_common_tx_done

Prototype

```
void rsi_bt_common_tx_done(rsi_pkt_t *pkt)
```

Description
This function handles BT data transfer completion.

Precondition
NA

Parameters

Parameter	Description
pkt	Pointer to packet

Return Values

None

Example

```
status = rsi_bt_common_tx_done(rsi_pkt_t *pkt)
```

15.10.5 rsi_get_bt_state

Prototype

```
uint32_t rsi_get_bt_state(rsi_bt_cb_t *bt_cb)
```

Description

This function returns BT status.

Precondition

NA

Parameters

Parameters	Description
bt_cb	BT Control block

Return values

Return the BT state.

Example

```
status =   rsi_get_bt_state(rsi_bt_cb_t *bt_cb)
```

15.10.6 rsi_bt_set_status

Prototype

```
 void rsi_bt_set_status(rsi_bt_cb_t *bt_cb, int32_t status)
```

Description

This function sets BT status.

Precondition

None

Parameters

Parameters	Description
bt_cb	BT Control block
status	status value to be set

Return Values

None

Example

```
rsi_bt_set_status(rsi_bt_cb_t *bt_cb, int32_t status)
```

15.10.7 rsi_bt_get_status

Prototype

```
uint32_t rsi_bt_get_status(rsi_bt_cb_t *bt_cb)
```

Description

This function get BT status.

Precondition

None

Parameters

Parameters	Description
bt_cb	BT Control block

Return Values

None

Example

```
rsi_bt_get_status(rsi_bt_cb_t *bt_cb)
```

15.10.8 rsi_ble_update_le_dev_buf

Prototype

```
void rsi_ble_update_le_dev_buf(rsi_ble_event_le_dev_buf_ind_t *rsi_ble_event_le_dev_buf_ind)
```

Description

This function updates local Device buffer availability per slave in global ble cb structure.

Precondition

None

Parameters

None

Return Values

Parameter	Description
0	Success
NonZero	Failure

Example

```
status = rsi_ble_update_le_dev_buf(rsi_ble_event_le_dev_buf_ind_t *rsi_ble_event_le_dev_buf_ind)
```

15.10.9 rsi_add_remote_ble_dev_info

Prototype

```
void rsi_add_remote_ble_dev_info(rsi_ble_event_enhance_conn_status_t *remote_dev_info)
```

Description

This function updates Remote BLE Device info in global ble cb structure.

Precondition

None

Parameters

None

Return Values

Parameter	Description
0	Success
NonZero	Failure

Example

```
status = rsi_add_remote_ble_dev_info(rsi_ble_event_enhance_conn_status_t *remote_dev_info)
```

15.10.10 rsi_remove_remote_ble_dev_info

Prototype

```
 void rsi_remove_remote_ble_dev_info(rsi_ble_event_disconnect_t  *remote_dev_info)
```

Description

This function removes Remote BLE Device info in global ble cb structure.

Precondition

None

Parameters

None

Return Values

Parameter	Description
0	Success
NonZero	Failure

Example

```
status = rsi_remove_remote_ble_dev_info(rsi_ble_event_disconnect_t  *remote_dev_info)
```

15.10.11 rsi_driver_process_bt_resp

Prototype

```
 int32_t rsi_driver_process_bt_resp (rsi_bt_cb_t *bt_cb, rsi_pkt_t *pkt,
    void (*rsi_bt_async_callback_handler) (rsi_bt_cb_t *cb, uint16_t type, uint8_t *data, uint16_t
length) )
```

Description

This function processes BT RX packets.

Precondition

None

Parameters

Parameters	Description
bt_cb	BT Control block
pkt	Pointer to RX packets

Return Values

Parameter	Description
0	Success
NonZero	Failure

Example

```
status =rsi_driver_process_bt_resp (rsi_bt_cb_t *bt_cb, rsi_pkt_t *pkt,
    void (*rsi_bt_async_callback_handler) (rsi_bt_cb_t *cb, uint16_t type, uint8_t *data, uint16_t
length) )
```

15.10.12 rsi_driver_process_bt_resp_handler

Prototype

```
uint16_t rsi_driver_process_bt_resp_handler(rsi_pkt_t *pkt)
```

Description

This function processes BT RX packets.

Precondition

None

Parameters

Parameters	Description
pkt	Pointer to RX packets

Return Values

Parameter	Description
0	Success
Non Zero	Failure

Example

```
uint16_t rsi_driver_process_bt_resp_handler(rsi_pkt_t *pkt)
```

15.10.13 rsi_bt_cb_init

Prototype

```
int8_t rsi_bt_cb_init(rsi_bt_cb_t *bt_cb)
```

Description

This function initializes bt control block structure.

Precondition

None

Parameters

Parameters	Description
bt_cb	BT Control block

Return Values

None

Example

```
status =  rsi_bt_cb_init(rsi_bt_cb_t *bt_cb)
```

15.10.14 rsi_bt_common_init

Prototype

```
 void rsi_bt_common_init(void)
```

Description

Waits for BT card ready.

Precondition

None

Parameters

None

Return Values

None

Example

```
status =  rsi_bt_common_init(void)
```

15.10.15 rsi_bt_gatt_extended_register_callbacks

Prototype

```
 void rsi_bt_gatt_extended_register_callbacks (
    rsi_bt_on_gatt_connection_t          bt_on_gatt_connection_event,
    rsi_bt_on_gatt_disconnection_t       bt_on_gatt_disconnection_event)
```

Description

This function registers the GAP callbacks.

Precondition

None

Parameters

Parameter	Description
bt_on_gatt_connection_event	GATT Connection status callback
rsi_bt_on_gatt_disconnection_t	GATT Disconnection status callback

Return Values

Parameter	Description
0	Success
Non Zero	Failure

Example

```
rsi_bt_gatt_extended_register_callbacks (
    rsi_bt_on_gatt_connection_t          bt_on_gatt_connection_event,
    rsi_bt_on_gatt_disconnection_t       bt_on_gatt_disconnection_event)
```

15.10.16 rsi_bt_avdtp_events_register_callbacks

Prototype

```
void rsi_bt_avdtp_events_register_callbacks (
    rsi_bt_on_avdtp_stats_t              bt_on_avdtp_stats_event)
```

Description

This function registers the avdtp stats callbacks.

Precondition

None

Parameters

Parameter	Description
rsi_bt_on_avdtp_stats_t	AVDP stats callback

Return Values

Parameter	Description
0	Success
Non Zero	Failure

Example

```
rsi_bt_avdtp_events_register_callbacks (
    rsi_bt_on_avdtp_stats_t              bt_on_avdtp_stats_event)
```

15.10.17 rsi_bt_on_chip_memory_status_callbacks_register

Prototype

```
void rsi_bt_on_chip_memory_status_callbacks_register (
    rsi_bt_on_chip_memory_stats_handler_t              bt_on_chip_memory_stats_event)
```

Description

This function registers the chip memory stats callbacks.

Precondition

None

Parameters

Parameter	Description
bt_on_chip_memory_stats_event	memory utilization Event callback

Return Values

Parameter	Description
0	Success
Non Zero	Failure

Example

```
rsi_bt_on_chip_memory_status_callbacks_register (
    rsi_bt_on_chip_memory_stats_handler_t          bt_on_chip_memory_stats_event)
```

15.10.18 rsi_bt_hfp_register_callbacks

Prototype

```
void rsi_bt_hfp_register_callbacks (
    rsi_bt_on_hfp_connect_t          bt_on_hfp_connect_event,
    rsi_bt_on_hfp_disconnect_t          bt_on_hfp_disconnect_event,
    rsi_bt_on_hfp_ring_t                bt_on_hfp_ring_event,
    rsi_bt_on_hfp_callcallerid_t           bt_on_hfp_callcallerid_event,
    rsi_bt_on_hfp_audioconnected_t         bt_on_hfp_audioconnected_event,
    rsi_bt_on_hfp_audiodisconnected_t      bt_on_hfp_audiodisconnected_event,
    rsi_bt_on_hfp_dialcomplete_t           bt_on_hfp_dialcomplete_event,
    rsi_bt_on_hfp_answercomplete_t         bt_on_hfp_answercomplete_event,
    rsi_bt_on_hfp_hangupcomplete_t         bt_on_hfp_hangupcomplete_event,
    rsi_bt_on_hfp_senddtmfcomplete_t       bt_on_hfp_senddtmfcomplete_event,
    rsi_bt_on_hfp_callwait_t               bt_on_hfp_callwait_event,
    rsi_bt_on_hfp_callvoicerecogdeactivated_t  bt_on_hfp_callvoicerecogdeactivated_event,
    rsi_bt_on_hfp_callvoicerecogactivated_t    bt_on_hfp_callvoicerecogactivated_event,
    rsi_bt_on_hfp_servicenotfound_t            bt_on_hfp_servicenotfound_event,
    rsi_bt_app_on_hfp_callstatus_t             bt_on_hfp_callstatus_event,
    rsi_bt_app_on_hfp_signalstrength_t         bt_on_hfp_signalstrength_event,
    rsi_bt_app_on_hfp_batterylevel_t           bt_on_hfp_batterylevel_event,
    rsi_bt_app_on_hfp_phoneservice_t           bt_on_hfp_phoneservice_event,
    rsi_bt_app_on_hfp_roamingstatus_t          bt_on_hfp_roamingstatus_event,
    rsi_bt_app_on_hfp_callsetup_t              bt_on_hfp_callsetup_event,
    rsi_bt_app_on_hfp_callheld_t               bt_on_hfp_callheld_event)
```

Description

This function registers rsi_bt_hfp_regiter_callbacks.

Precondition

None

Parameters

Parameter	Description
bt_on_hfp_connect_event	HFP connect event
bt_on_hfp_ring_event,	hfp ring event
bt_on_hfp_callcallerid_event	hfp call callerid event
bt_on_hfp_audioconnected_event	hfp audio connected event
bt_on_hfp_audiodisconnected_event	hfp audio disconnected event
bt_on_hfp_dialcomplete_event	hfp dail complete event
bt_on_hfp_answercomplete_event	hfp answer complete event
bt_on_hfp_hangupcomplete_event	hfp hangup complete event
bt_on_hfp_senddtmfcomplete_event	hfp send dtmf complete event
bt_on_hfp_callwait_event	hfp call wait event
bt_on_hfp_callvoicerecogdeactivated_event	hfp call voice record deactivated event
bt_on_hfp_callvoicerecogactivated_event	hfp call voice record activated
bt_on_hfp_servicenotfound_event	hfp service not found
bt_on_hfp_callstatus_event	hfp call status event
bt_on_hfp_signalstrength_event	hfp signal strength event
bt_on_hfp_batterylevel_event	hfp battery level event
bt_on_hfp_phoneservice_event	hfp phone service event

Parameter	Description
bt_on_hfp_roamingstatus_event	hfp roaming status event
bt_on_hfp_callsetup_event	hfp call setup event
bt_on_hfp_callheld_event	hfp call held event

Return Values

Parameter	Description
0	Success
Non Zero	Failure

rsi_bt_callbacks_handler

Prototype

```
void rsi_bt_callbacks_handler(rsi_bt_cb_t *bt_classic_cb, uint16_t rsp_type, uint8_t *payload, uint16_t payload_length)
```

Description

This function initailizes the BT callbacks register.

Parameters

Parameters	Description
bt_classic_cb	BT BLE struct pointer
rsp_type	BT Packet type

Return Values

Parameter	Description
0	Success
Non Zero	Failure

Example

```
status = rsi_bt_callbacks_handler(rsi_bt_cb_t *bt_classic_cb, uint16_t rsp_type, uint8_t *payload, uint16_t payload_length)
```

15.10.19 rsi_ble_callbacks_handler

Prototype

```
void rsi_ble_callbacks_handler(rsi_bt_cb_t *ble_cb, uint16_t rsp_type, uint8_t *payload, uint16_t payload_length)
```

Description

This function initailizes the BT callbacks register.

Parameters

Parameters	Description
ble_cb	BLE control back
rsp_type	BLE Packet type

Return Values

Parameter	Description
0	Success
Non Zero	Failure

Example

```
status = rsi_ble_callbacks_handler(rsi_bt_cb_t *ble_cb, uint16_t rsp_type, uint8_t *payload, uint16_t payload_length)
```

15.10.20 rsi_ble_on_chip_memory_status_callbacks_register

Prototype

```
 void rsi_ble_on_chip_memory_status_callbacks_register (chip_ble_buffers_stats_handler_t ble_on_chip_memory_status_event)
```

Description

This function registers the BLE chip memory stats callbacks.

Precondition

None

Parameters

Parameter	Description
ble_on_chip_memory_status_event	memory status Event callback

Return Values

Parameter	Description
0	Success
Non Zero	Failure

Example

```
rsi_ble_on_chip_memory_status_callbacks_register (chip_ble_buffers_stat_handler_t ble_on_chip_memory_status_event)
```

15.10.21 rsi_bt_prepare_common_pkt

Prototype

```
uint16_t rsi_bt_prepare_common_pkt(uint16_t cmd_type, void *cmd_struct, rsi_pkt_t *pkt)
```

Description

This function forms the payload of the BT command packet.

Precondition

None

Parameters

Parameters	Description
cmd_type	Command Type
cmd_struct	pointer of the command structure
pkt	pointer of the packet to fill the contents of the payload

Return Values

payload_size, size of the payload.

Example

```
payload rsi_bt_prepare_common_pkt(uint16_t cmd_type, void *cmd_struct, rsi_pkt_t *pkt)
```

15.10.22 rsi_bt_prepare_classic_pkt

Prototype

```
uint16_t rsi_bt_prepare_classic_pkt(uint16_t cmd_type, void *cmd_struct, rsi_pkt_t *pkt)
```

Description

This function forms the payload of the BT Classic command packet.

Precondition

None

Parameters

Parameters	Description
cmd_type	Command Type
cmd_struct	pointer of the command structure
pkt	pointer of the packet to fill the contents of the payload

Return Values

payload_size, size of the payload.

Example

```
status = rsi_bt_prepare_classic_pkt(uint16_t cmd_type, void *cmd_struct, rsi_pkt_t *pkt)
```

15.10.23 rsi_bt_prepare_le_pkt

Prototype

```
uint16_t rsi_bt_prepare_le_pkt(uint16_t cmd_type, void *cmd_struct, rsi_pkt_t *pkt)
```

Description

This function forms the payload of the BT Classic command packet.

Precondition

None

Parameters

Parameters	Description
cmd_type	Command Type
cmd_struct	pointer of the command structure
pkt	pointer of the packet to fill the contents of the payload

Return Values

payload_size, size of the payload.

Example

```
status = rsi_bt_prepare_le_pkt(uint16_t cmd_type, void *cmd_struct, rsi_pkt_t *pkt)
```

15.10.24 rsi_bt_driver_send_cmd

Prototype

```
int32_t rsi_bt_driver_send_cmd(uint16_t cmd, void *cmd_struct, void *resp)
```

Description

This function fills commands and places into BT TX queue.

Precondition

None

Parameters

Parameters	Description
cmd	Type of command to send
cmd_struct	pointer of the command structure to send
resp	pointer of the packet to fill the contents of the payload

Return Values

payload_size, size of the payload.

Example

```
status = rsi_bt_driver_send_cmd(uint16_t cmd, void *cmd_struct, void *resp)
```

16 SAPI Error Codes

16.1 SAPI Generic Error Codes

Table 10: SAPI Generic Error Codes

Error Codes	Error codes (in hexadecimal format)	Description
-1	0xffffffff	Host Time out error
-2	0xfffffffe	Invalid parameters
-3	0xfffffffd	Command given in wrong state
-4	0xfffffffc	Packet allocation failure
-5	0xfffffffb	Command not supported
-6	0xfffffffa	Insufficient buffer
-7	0xfffffff9	Error in O.S Operation
-8	0xfffffff8	Error due to Invalid Memory
-9	0xfffffff7	Boot_up Options not saved
-10	0xfffffff6	Boot up Options check sum fail
-11	0xfffffff5	Boot loader Version not Match
-12	0xfffffff4	Waiting for Board Ready
-13	0xfffffff3	Invalid Address
-14	0xfffffff2	Valid Firmware not present
-15	0xfffffff1	Invalid option
-16	0xfffffff0	Maximum Callbacks are Exceeded
-17	0xffffffef	Set_Timer error
-18	0xffffffee	SIGACTION error
-19	0xffffffed	Not in Connected state
-20	0xffffffec	Not in IP Config state
-21	0xffffffeb	SPI busy error
-22	0xffffffea	SPI fail error
-23	0xffffffe9	SPI Timeout error
-24	0xffffffe8	Card ready timeout error
-25	0xffffffe7	Board ready timeout error
-26	0xffffffe6	Invalid Packet error
-27	0xffffffe5	Firmware Upgrade timeout error
-28	0xffffffe4	Firmware Load or Upgrade timeout error
-29	0xffffffe3	GPIO wakeup timeout error
-30	0xffffffe2	Response timeout error

-31	0xffffffe1	BLE buffer full error
-32	0xffffffe0	Network command in progress
-33	0xffffffdf	Socket command in progress
-34	0xffffffde	WLAN command in progress
-35	0xffffffdd	Common command in progress
-36	0xffffffdc	ANT Buffer full error
-37	0xffffffdb	BT or BLE command in progress
-38	0xffffffda	BLE Attribute command in progress
-39	0xffffffd9	Memory not aligned error

16.2 Bluetooth Generic Error Codes

Table 11: Bluetooth Generic Error Codes

Error Code	Description
0x4E01	Unknown HCI command
0x4E02	Unknown Connection Identifier
0x4E03	Hardware failure
0x4E04	Page timeout
0x4E05	Authentication failure
0x4E06	Pin missing
0x4E07	Memory capacity exceeded
0x4E08	Connection timeout
0x4E09	Connection limit exceeded
0x4E0A	SCO limit exceeded
0x4E0B	ACL Connection already exists
0x4E0C	Command disallowed
0x4E0D	Connection rejected due to limited resources
0x4E0E	Connection rejected due to security reasons
0x4E0F	Connection rejected for BD address
0x4E10	Connection accept timeout
0x4E11	Unsupported feature or parameter
0x4E12	Invalid HCI command parameter
0x4E13	Remote user terminated connection
0x4E14	Remote device terminated connection due to low resources
0x4E15	Remote device terminated connection due to power off
0x4E16	Local device terminated connection
0x4E17	Repeated attempts

Error Code	Description
0x4E18	Pairing not allowed
0x4E19	Unknown LMP PDU
0x4E1A	Unsupported remote feature
0x4E1B	SCO offset rejected
0x4E1C	SCO interval rejected
0x4E1D	SCO Air mode rejected
0x4E1E	Invalid LMP parameters
0x4E1F	Unspecified
0x4E20	Unsupported LMP Parameter
0x4E21	Role change not allowed
0x4E22	LMP response timeout
0x4E23	LMP transaction collision
0x4E24	LMP PDU not allowed
0x4E25	Encryption mode not acceptable
0x4E26	Link key cannot change
0x4E27	Requested QOS not supported
0x4E28	Instant passed
0x4E29	Pairing with unit key not supported
0x4E2A	Different transaction collision
0x4E2B	Reserved 1
0x4E2C	QOS parameter not acceptable
0x4E2D	QOS rejected
0x4E2E	Channel classification not supported
0x4E2F	Insufficient security
0x4E30	Parameter out of mandatory range
0x4E31	Reserved 2
0x4E32	Role switch pending
0x4E33	Reserved 3
0x4E34	Reserved slot violation
0x4E35	Role switch failed
0x4E36	Extended Inquiry Response too large
0x4E37	Extended SSP not supported
0X4E38	Host busy pairing
0x4E39	Wrong BD Address
0x4E3E	Connection Failed to be Established

Error Code	Description
0x4FF8	BT Invalid Command
0x0102	Unknown
0x0103	Firmware Timeout
0x0104	Memory alloc fail
0x0106	Io fail
0x0108	Unsupported
0x0109	Short buf
0x010A	Buf overflow
0x010B	Too large buf
0x010C	Io abort
0x010D	File open fail
0x1010	Os task invalid prio
0x1011	Os task prio exists
0x1012	Os task not stopped
0x1020	Os sem max value
0x1021	Os sem not available
0x1022	Os sem reset
0x1030	Os mutex not owner
0x1031	Os mutex not locked
0x1032	Os mutex lock failed
0x1033	Os mutex try lock failed
0x1040	Os msg queue full
0x1041	Os message queue empty
0x1050	Pipe empty
0x1051	Pipe full
0x1052	Invalid len
0x1053	Pipe read in use
0x1054	Pipe write in use
0x1060	Os timer expired
0x1061	Os timer state running
0x1070	Os cannot wait
0x1080	Os mem pool empty
0x1081	Os mem pool size short

16.3 Bluetooth Classic Error Codes

16.3.1 BT Core Error Codes

Table 12: BT Core Error Codes

Error Code	Description
0x4040	IO Fail
0x4041	Unknown
0x4042	HW Busy
0x4043	Max Sock
0x4044	Short Buf
0x4045	Max Name Size
0x4046	Invalid Args
0x4047	Sock open fail
0x4048	Timeout
0x4049	Socket state invalid
0x404A	Bad bd address
0x404B	Acl packet error
0x404C	Pool alloc fail
0x404D	Tx fail
0x404E	Connection refused
0x404F	Confirmation result
0x4050	Remote user disconnected
0x4051	Remote device not responding
0x4052	Invalid command
0x4053	Unsupported feature param value
0x4054	Thread create fail
0x4055	Sem wait fail
0x4056	Pool full
0x4057	Hw buffer overflow
0x4058	Tx buffer empty
0x4059	HCI connection fail
0x405A	Operation incomplete
0x405B	Operation cancel
0x405C	BSP error
0x4060	Sco connection fail
0x4061	No HCI connection
0x4062	Socket disconnected

Error Code	Description
0x4063	Socket timeout
0x4064	HCI connection encrypt fail
0x4065	Max acl packet buffer length
0x4066	Max nbr acl packets
0x4067	Invalid state
0x4069	Remote name fail
0x406A	Invalid response
0x4071	Invalid psm
0x4072	Psm in use
0x4073	Invalid hci connection handle
0x4074	Invalid cid
0x4075	Invalid pkt
0x4080	Scn is in use
0x4081	Max acl connections
0x4082	Sock already exists
0x4100	Invalid pdu
0x4101	Invalid pdu data element
0x4102	Sdp service not found
0x4103	Sdp attribute not found
0x4104	Sdp max service attribute
0x4200	Max RF communication channels
0x4201	RF communication disconnected
0x4202	RF communication channel not found
0x4203	RF communication invalid packet
0x4204	RF communication remote credits zero
0x4205	RF communication invalid state
0x4206	RF communication fcoff
0x4207	No service connection
0x4300	HCI connection already exists
0x4301	Max hci connection
0x4302	SCO invalid state
0x4510	A2DP Not Initialized
0x4511	A2DP Connection Already Exists
0x4512	A2DP Not Streaming
0x4540	AVRCP Not Initialized

Error Code	Description
0x4541	AVRCP Non Valid Capability ID
0x4542	AVRCP Non Valid Event ID
0x454E	AVRCP Invalid Response
0x454F	AVRCP Command Not Implemented
0x4550	L2CAP Not Connected
0x4551	L2CAP Not Initialized
0x4552	L2CAP Data Length Exceeds Remote MTU
0x4A00	AVDTP Invalid Packet
0x4A01	AVDTP Closed
0x4A02	AVDTP Max Caps Length
0x4A03	AVDTP Invalid State
0x4A04	AVDTP Max Local Seps
0x4C00	AVRCP Function Busy
0x4C01	AVRCP CMD Rejected
0x4C02	AVRCP CMD Attribute Not Found
0x4FF9	Inquiry cancel command is given when device is not in Inquiry State
0x4604	SPP Tx FAIL

BT Event Queue Error Codes

Table 13: BT Event Queue Error Codes

Error Code	Description
0x1090	OS Event queue full
0x1091	OS Event not available
0x1092	OS Event not created
0x1093	OS Event prio not created
0x1094	OS Event no event created

16.4 Bluetooth Low Energy Error Codes

16.4.1 BLE Generic Error Codes

Table 14: BLE Generic Error Codes

Error Code	Description
0x4E3C	Directed Advertising Timeout
0x4E3D	Connection terminated due to MIC failure
0x4E60	Invalid Handle Range
0x4E62	Invalid Parameters

Error Code	Description
0x4E63	BLE Buffer Count Exceeded
0x4E64	BLE Buffer already in use
0x4E65	Invalid Attribute Length When Small Buffer Mode is Configured

16.4.2 BLE Mode Error Codes

Table 15: BLE Mode Error Codes

Error Code	Description
0x4A01	Invalid Handle
0x4A02	Read not permitted
0x4A03	Write not permitted
0x4A04	Invalid PDU
0x4A05	Insufficient authentication
0x4A06	Request not supported
0x4A07	Invalid offset
0x4A08	Insufficient authorization
0x4A09	Prepare queue full
0x4A0A	Attribute not found
0x4A0B	Attribute not Long
0x4A0C	Insufficient encryption key size
0x4A0D	Invalid attribute value length
0x4A0E	Unlikely error
0x4A0F	Insufficient encryption
0x4A10	Unsupported group type
0x4A11	Insufficient resources
0x4B01	SMP Passkey entry failed
0x4B02	SMP OOB not available
0x4B03	SMP Authentication Requirements
0x4B04	SMP confirm value failed
0x4B05	SMP Pairing not supported
0x4B06	SMP Encryption key size insufficient
0x4B07	SMP command not supported
0x4B08	SMP pairing failed
0x4B09	SMP repeated attempts
0x4B0C	SMP Failed
0x4C02	PSM Conn Failed

Error Code	Description
0x4D00	BLE Remote device found
0x4D01	BLE Remote device not found
0x4D02	BLE Remote device structure full
0x4D03	Unable to change state
0x4D04	BLE not Connected
0x4D05	BLE socket not available.
0x4D06	Attribute record not found
0x4D07	Attribute entry not found
0x4D08	Profile record full
0x4D09	Attribute record full
0x4D0A	BLE profile not found (profile handler invalid)
0x4D0B	BLE Attribute Buffer Full
0x4D10	BLE Connection Sock not Available
0x4D11	BLE Remote Credits not Available
0x4057	HW Buffer Overflow

16.5 WLAN Error codes

Table 16: WLAN Error Codes

Error Codes (in hexadecimal format)	Description
0x0002	Scan command issued while module is already associated with an Access Point
0x0003	No AP found
0x0004	Wrong PSK is issued while the module client tries to join an Access Point with WEP security enabled
0x0005	Invalid band
0x0006	Association not done or in unassociated state
0x0008	Deauthentication received from AP
0x0009	Failed to associate to Access Point during "Join"
0x000A	Invalid channel
0x000E	1. Authentication failure during "Join" 2. Unable to find AP during join which was found during scan.
0x000F	Missed beacon from AP during join
0x0013	Non-existent MAC address supplied in "Disassociate" command
0x0014	EAP configuration is not done
0x0015	Memory allocation failed or Store configuration check sum failed

Error Codes (in hexadecimal format)	Description
0x0016	Information is wrong or insufficient in Join command
0x0018	Push button command given before the expiry of previous push button command
0x0019	1. Access Point not found 2. Rejoin failure
0x001A	Frequency not supported
0x001C	EAP configuration failed
0x001D	P2P configuration failed
0x001E	Unable to start Group Owner negotiation
0x0020	Unable to join
0x0021	Command given in incorrect state
0x0022	Query GO parameters issued in incorrect operating mode
0x0023	Unable to form Access Point
0x0024	Wrong Scan input parameters supplied to "Scan" command
0x0025	Command issued during re-join in progress
0x0026	Wrong parameters the command request
0x0028	PSK length less than 8 bytes or more than 63 bytes
0x0029	Failed to clear or to set the Enterprise Certificate (Set Certificate)
0x002B	Association between nodes failed in WPS Failed due to timeout
0x002C	If a command is issued by the Host when the module is internally executing auto-join or auto-create
0x002D	WEP key is of wrong length
0x002E	ICMP request timeout error
0x002F	ICMP data size exceeds maximum limit
0x0030	Send data packet exceeded the limit or length that is mentioned (or) Mqtt publish data and publish data length mismatched (or) MQTT Send data packet exceeded the limit.
0x0031	ARP Cache entry not found
0x0032	UART command timeout happened
0x0033	Fixed data rate is not supported by connecting AP
0x0037	Wrong WPS PIN
0x0038	Wrong WPS PIN length
0x0039	Wrong PMK length
0x003a	SSID not present for PMK generation
0x003b	SSID incorrect for PMK generation(more than 32 bytes)

Error Codes (in hexadecimal format)	Description
0x003C	Band not supported
0x003D	User store configuration invalid length
0x003E	Error in length of the command (Exceeds number of characters is mentioned in the PRM)
0x003F	Data packet dropped
0x0040	WEP key not given
0x0041	Wrong PSK length
0x0042	PSK or PMK not given
0x0043	Security mode given in join command is invalid
0x0044	Beacon miscount reaches max beacon miscount (Deauthentication due to beacon miss)
0x0045	Deauthentication received from supplicant
0x0046	Deauthentication received from AP after channel switching
0x0047	Synchronization missed
0x0048	Authentication timeout occurred
0x0049	Association timeout
0x004A	BG scan in given channels is not allowed
0x004B	Scanned SSID and SSID given in Join are not matching
0x004C	Given number of clients exceeded max number of stations supported
0x004D	Given HT capabilities are not supported
0x004E	Uart Flow control not supported
0x004F	ZB/BT/BLE packet received and protocol is not enabled
0x0050	Parameters error
0x0051	Invalid RF current mode
0x0052	Power save support is not present for a given interface
0x0053	Concurrent AP in connected state
0x0054	Connected AP or Station channel mismatch
0x0055	IAP co processor error
0x0056	WPS not supported in current operating mode
0x0057	Concurrent AP doesn't have same channel as connected station channel
0x0058	PBC session overlap error
0x005A	4/4 confirmation of 4 way handshake failed
0X005C	Concurrent mode, both AP and Client should UP, to enable configuration
0x005D	Certificate load not allowed in flash
0x005E	Certificate load not allowed in RAM

Error Codes (in hexadecimal format)	Description
0x005F	Certificate load failed due to wrong inx
0x0060	AP HT caps not enabled for 40 mhz
0x0061	Address family not supported by protocol.
0x0062	Invalid beacon interval provided.
0x0063	Invalid range of the configuration provided
0x0064	RTS THRESHOLD Config type is invalid.
0x0065	Error with MQTT command
0x0066	listen interval in power save is greater than the join listen interval
0x005B	MAC address not present in MAC based join
0x00B1	Memory Error: No memory available
0x00B2	Invalid characters in JSON object
0x00B3	Update Commands: No such key found
0x00B4	No such file found: Re-check filename
0x00B5	No corresponding webpage exists with same filename
0x00B6	Space unavailable for new file
0x00C1	Invalid input data, Re-check filename, lengths etc
0x00C2	Space unavailable for new file
0x00C3	Existing file overwrite: Exceeds size of previous file. Use erase and try again
0x00C4	No such file found. Re-check filename
0x00C5	Memory Error: No memory available
0x00C6	Received more web-page data than the total length initially specified
0x00C7	Error in set region command
0x00C8	Web-page current chunk length is incorrect
0x00CA	Error in Ap set region command
0X00CB	Error in AP set region command parameters
0x00CC	Region code not supported
0x00CD	Error in extracting country region from beacon
0x00CE	Module does not have selected region support
0x00D1	SSL Context Create Failed
0x00D2	SSL Handshake Failed. Socket will be closed
0x00D3	SSL Max sockets reached. Or FTP client is not connected
0x00D4	Cipher set failure
0x00F1	HTTP credentials maximum length exceeded
0x0100	SNMP internal error

Error Codes (in hexadecimal format)	Description
0x0104	SNMP invalid IP protocol error
0xBB01	No data received or receive timeout
0xBB0A	Invalid SNTP server address
0xBB0B	SNTP client not started
0xBB10	SNTP server not available, Client will not get any time update service from current server
0xBB15	SNTP server authentication failed
0xBB0E	Internal error
0xBB16	Entry not found for multicast IP address
0xBB17	No more entries found for multicast
0xBB21	IP address error
0xBB22	Socket already bound
0xBB23	Port not available
0xBB27	Socket is not created
0xBB29	ICMP request failed
0xBB33	Maximum listen sockets reached
0xBB34	DHCP duplicate listen
0xBB35	Port Not in close state
0xBB36	Socket is closed or in process of closing
0xBB37	Process in progress
0xBB38	Trying to connect non-existing TCP server socket
0xBB3E	Error in length of the command(Exceeds number of characters is mentioned in the PRM)
0xBB42	Socket is still bound
0xBB45	No free port
0xBB46	Invalid port
0xBB4B	Feature not supported
0xBB50	Socket is not in connected state. Disconnected from server. In case of FTP, user need to give destroy command after receiving this error
0xBB87	POP3 session creation failed/ POP3 session got terminated
0xBB9C	DHCPv6 Handshake failure
0xBB9D	DHCP invalid IP response
0xBBA0	SMTP Authentication error
0xBBA1	No DNS server was specified, SMTP over size mail data

Error Codes (in hexadecimal format)	Description
0xBBA2	SMTP invalid server reply
0xBBA3	DNS query failed, SMTP internal error
0xBBA4	Bad DNS address, SMTP server error code received
0xBBA5	SMTP invalid parameters
0xBBA6	SMTP packet allocation failed
0xBBA7	SMTP GREET reply failed
0xBBA8	Parameter error, SMTP Hello reply error
0xBBA9	SMTP mail reply error
0xBBAA	SMTP RCPT reply error
0xBBAB	SMTP message reply error
0xBBAC	SMTP data reply error
0xBBAD	SMTP authentication reply error
0xBBAE	SMTP server error reply
0xBBAF	DNS duplicate entry.
0xBBB1	SMTP oversize server reply
0xBBB2	SMTP client not initialized
0xBBB3	DNS IPv6 not suported
0xBBC5	Invalid mail index for POP3 mail retrieve command
0xBBD2	SSL handshake failed
0xBBD3	FTP client is not connected or disconnected with the FTP server
0xBBD4	FTP client is not disconnected
0xBBD5	FTP file is not opened
0xBBD6	SSL handshake timeout or FTP file is not closed
0xBBD9	Expected [1XX response from FTP server but not received
0xBBDA	Expected [2XX response from FTP server but not received
0xBBDB	Expected [22X response from FTP server but not received
0xBBDC	Expected [23X response from FTP server but not received
0xBBDD	Expected [3XX response from FTP server but not received
0xBBDE	Expected [33X response from FTP server but not received
0xBBE1	HTTP Timeout
0xBBE2	HTTP Failed
0xBBE7	HTTP Timeout for HTTP PUT client
0xBBEB	Authentication Error

Error Codes (in hexadecimal format)	Description
0xBBED	Invalid packet length, content length and received data length is mismatching
0xBBEF	Server responds before HTTP client request is complete
0xBBF0	HTTP/HTTPS password is too long
0xBBF1	MQTT ping time out error
0xBBF2	MQTT command sent in incorrect state
0XBBF3	MQTT ACK time out error
0xBBFF	POP3 error for invalid mail index
0XFFFF	Listening TCP socket in module is not connected to the remote peer, or the LTCP socket is not yet opened in the module
0xFFFE	Sockets not available. The error comes if the Host tries to open more than 10 sockets.
0xFFFB	Cannot create IP in same interface in concurrent mode
0xFFFE	Sockets not available. The error comes if the Host tries to open more than 10 sockets
0xFFFC	IP configuration failed
0xFFF5	Store configuration profile type mismatch or Invalid profile type
0xFFF6	MQTT REMOTE TERMINATE ERROR
0xFFF7	Byte stuffing error in AT mode
0xFFF8	1. Invalid command (e.g. parameters insufficient or invalid in the command). Invalid operation (e.g. power save command with the same mode given twice, accessing wrong socket, creating more than allowed sockets)
0XFFFA	TCP socket is not connected
0xFFC5	Station count exceeded max station supported
0xFFC4	Unable to send tcp data
0xFFBC	Socket buffer too small
0xFFBB	Invalid content in the DNS response to the DNS Resolution query
0xFFBA	DNS Class error in the response to the DNS Resolution query
0xFFB8	DNS count error in the response to the DNS Resolution query
0xFFB7	DNS Return Code error in the response to the DNS Resolution query
0xFFB6	DNS Opcode error in the response to the DNS Resolution query
0xFFB5	DNS ID mismatch between DNS Resolution request and response
0xFFAB	An invalid input to the DNS Resolution query
0xFF42	DNS response was timed out

Error Codes (in hexadecimal format)	Description
0xFFA1	ARP request failure
0xFF9D	DHCP lease time expired
0xFF9C	DHCP handshake failure
0xFF88	This error is issued when WebSocket creation failed
0xFF87	This error is issued when module tried to connect to a non-existent TCP server socket on the remote side
0xFF86	This error is issued when tried to close non-existent socket. or invalid socket descriptor
0xFF85	Invalid socket parameters
0xFF82	Feature not supported
0xFF81	Socket already open
0xFF80	Attempt to open more than the maximum allowed number of sockets
0XFF7E	Data length exceeds mss
0xFF74	Feature not enabled
0xFF73	DHCP server not set in AP mode
0xFF71	Error in AP set region command parameters
0xFF70	SSL not supported
0xFF6F	JSON not supported
0xFF6E	Invalid operating mode
0xFF6D	Invalid socket configuration parameters
0xFF6C	Web socket creation timeout
0xFF6B	Parameter maximum allowed value is exceeded
0xFF6A	Socket read timeout
0xFF69	Invalid command in sequence
0xFF42	DNS response timed out
0xFF41	HTTP socket creation failed
0xFF40	TCP socket close command is issued before getting the response of the previous close command
0xFF36	Wait On Host feature not enabled
0xFF35	Store configuration checksum validation failed
0xFF33	TCP keep alive timed out
0xFF2D	TCP ACK failed for TCP SYN-ACK
0xFF2C	Memory limit exceeded in a given operating mode
0xFF2A	Memory limit exceeded in operating mode during auto join/create

Error Codes (in hexadecimal format)	Description
0xCC2F	PUF Operation is blocked
0xCC31	PUF Activation code invalid
0xCC32	PUF input parameters invalid
0xCC33	PUF in error state
0XCC34	PUF Operation not allowed
0XCC35	PUF operation Failed
0x5a5a	Auto join or user store configuration going on.
0xFFE1	Improper RSNIE from AP to station

16.6 Socket Error Codes

Table 17: Socket Error Codes

Error Code	Description
RSI_ERROR_EBADF	Bad file number
RSI_ERROR_EPROTOTYPE	Protocol wrong type for socket
RSI_ERROR_EINVAL	Invalid argument
RSI_ERROR_ENOBUFS	No buffer space available
RSI_ERROR_ENOPROTOOPT	Protocol not available
RSI_ERROR_EFAULT	Bad address
RSI_ERROR_EAFNOSUPPORT	Address family not supported by protocol
RSI_ERROR_EMSGSIZE	Message too long

17 Changes/Enhancements in APIs, Configurations and Mechanisms

This section describes details of new changes and enhancements in RS9116W SAPI library which should be considered to ensure backward compatibility.

17.1 Changes and Enhancements from release 1.X.X to 2.X.X

17.1.1 Wi-Fi

This section describes the changes and enhancements in Wi-Fi and Network applications part of SAPIs.

Change/ Enhancement	Description	Example	Resolution for Backward Compatibility

Added support for Socket ID based return status for Socket APIs.	**Purpose:**
	To get socket ID based return status for socket APIs.
	Type:
	Enhancement
	Existing Mechanism (1.X.X):
	• **Description:** A single API was used to get the return status of any SAPI command in case for a failure. The same API was used to get socket API status as well.
	• **API:** r*si_wlan_get_status()*
	New Mechanism (2.X.X):
	• **Description:** A new API is added to get return status for socket APIs based on socket ID input. Latest SAPI driver stores response for each socket id.
	• **New API:** *rsi_wlan_socket_get_status(int32_t sockID)*
	Impact:
	In multi-threaded applications using *rsi_wlan_get_status()*, may not return exact Socket ID based status, as it may get over-written.
	Applications should use *rsi_wlan_socket_get_status()* instead of *rsi_wlan_get_status()* for getting actual return status for Socket APIs, especially in muti-threaded applications.
		• **rsi_wlan_get_status()**
		`status = rsi_connect(client_socket, (struct rsi_sockaddr *) &server_addr, sizeof(server_addr)); if(status != RSI_SUCCESS) { status = rsi_wlan_get_status(); rsi_shutdown(client_socket, 0); return status; }`
		• **rsi_wlan_socket_get_status(int32_t sockID)**
		`status = rsi_connect(client_socket, (struct rsi_sockaddr *) &server_addr, sizeof(server_addr)); if(status != RSI_SUCCESS) { status = rsi_wlan_socket_get_status(client_socket); rsi_shutdown(client_socket, 0); return status; }`
			No changes required for achieving backward compatibility.
			By default, both APIs r*si_wlan_get_status()* and *rsi_wlan_socket_get_status(int32_t sockID)* returns actual status of Socket APIs.
			Macro Usage:
			• Enable **RSI_WLAN_STATUS** - to use *rsi_wlan_socket_get_status(int32_t sockID)* API for Socket APIs return status. In this case, r*si_wlan_get_status() will not return Socket APIs return status.*
			• Disable **RSI_WLAN_STATUS** - to use both APIs r*si_wlan_get_status()* and *rsi_wlan_socket_get_status(int32_t sockID)* for getting Socket APIs return status.
			By default, this macro is not defined in any application examples.
			Note:
			New users (from 2.X.X) should use *rsi_wlan_socket_get*

Change/ Enhancement	Description	Example	Resolution for Backward Compatibility
			_status(int32_t sockID) as it is effective for multi-threaded applications.

Enable 'socket wait close' configuration if using TCP Sockets in Application.	If using TCP sockets in user application enable ***EXT_TCP_IP_WAIT_FOR_SOCKET_CLOSE*** (BIT(16)) in '***RSI_EXT_TCPIP_FEATURE_BITMAP***' in rsi_wlan_config.h.
	For **enabling** 'RSI_EXT_TCPIP_FEATURE_BITMAP' need to enable '***TCP_IP_FEAT_EXTENSION_VALID***' in '***RSI_TCP_IP_FEATURE_BIT_MAP***' in rsi_wlan_config.h.
		In rsi_wlan_config.h,**TCP wait close**
		`//! TCP/IP feature select bitmap for selecting TCP/IP features`
		`#define RSI_TCP_IP_FEATURE_BIT_MAP (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_EXTENSION_VALID)`
		`#define RSI_EXT_TCPIP_FEATURE_BITMAP EXT_TCP_IP_WAIT_FOR_SOCKET_CLOSE`
			For more details refer to 'rsi_shutdown()' API in RS9116W SAPI Programming Reference Manual at https://docs.silabs.com/rs9116/

17.1.2 BT/BLE

This will impact only for BLE not for BT-Classic.

Change/ Enhancement	Description	Example	Resolution for Backward Compatibility

Mandatory 384K memory configuration for all BT, BLE and COEX application examples	It is mandatory to configure 384K MEMORY configuration for all BT Classic, BLE and COEX application examples. RS9116W firmware, from 2.X.X release supports only this mode.	`//! To set custom feature select bit map`
		`#define RSI_CUSTOM_FEATURE_BIT_MAP FEAT_CUSTOM_FEAT_EXTENTION_VALID`
		`//! To set Extended custom feature select bit map`
		`#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP EXT_FEAT_384K_MODE`
			Configurations:
			In rsi_wlan_config.h,
			set **'RSI_EXT_CUSTOM_FEATURE_BIT_MAP'** to **'EXT_FEAT_384K_MODE'**.
			For enabling **'RSI_EXT_CUSTOM_FEATURE_BIT_MAP'**, it is required to set **'RSI_CUSTOM_FEATURE_BIT_MAP'** with **'FEAT_CUSTOM_FEAT_EXTENTION_VALID'**
			Note:
			In case, if '**EXT_FEAT_256K_MO** used for afore mentio cases, module hangs.

Change/ Enhancement	Description	Example	Resolution for Backward Compatibility

Support for HEX based input BD address in SAPI applications	**Purpose:**
	To avoid input BD address conversion from ASCII to HEX as RS9116W expect BD address in HEX format.
	Type:
	Enhancement.
	Existing Mechanism (1.X.X):
	RS9116W expects BD address in 6 bytes of HEX as input. But existing SAPI applications expect BD address input in ASCII and coverts it into HEX (internally in SAPI core) and then passes it to RS9116W.
	New Mechanism (2.X.X):
	Added support to take the input as a 6 bytes of HEX value which avoids ASCII to HEX conversion.
	• **New API:** None
	Impact:
	Existing designs which are developed with ASCII based BD address input will not work.
		`//! BD ADDR in ASCII`
		`#define RSI_BLE_DEV_1_ADDR "00:1A:7D:DA:71:16"`
		
		`//! BD ADDR in HEX`
		`uint8_t ble_addr[6] = {16,71,DA,7D,1A,0};`
		
		`int32_t rsi_ble_gatt_write_response(uint8_t *dev_addr,`
		
		`uint8_t type)`
		`{`
		`rsi_ble_gatt_write_response_t local_write_resp = {{0}};`
		`#ifdef BD_ADDR_IN_ASCII`
		`rsi_ascii_dev_address_to_6bytes_rev(local_write_resp.dev_addr, dev_addr);`
		`#else`
		`memcpy ((uint8_t *)local_write_resp.dev_addr, (int8_t *)dev_addr, 6);`
		`#endif`
		`local_write_resp.type = type;`
		
		`return rsi_bt_driver_send_cmd(RSI_BLE_CMD_WRITE_RESP, &local_write_resp, NULL);`
		`}`
			Backward compatibility is achieved by enabling '**BD_ADDR_IN_ASCII**' macro in rsi_bt_config.h /rsi_ble_config.h.
			Macro Usage:
			• Enable **BD_ADDR_IN_ASCII -** To take input Bluetooth Address (Device Address) in ASCII format in SAPI based applications.
			• Disable **BD_ADDR_IN_ASCII -** To take input Bluetooth Address (Device Address) in 6 bytes HEX format in SAPI based applications.
			By default, '**BD_ADDR_IN_ASCII**' is not defined in RS9116W SAPI examples and library.
			Note:
			Existing users should enable **BD_ADDR_IN_ASCII** in their applications to make sure SAPI library supports ASCII based BD address input.

BLE Data flow mechanism to support flow control and multiple connections	**Purpose:**
	To maintain multiple BLE connections and serve them independently, RS9116W firmware and SAPIs are enhanced with support for connection based buffer management and flow control
		`status = rsi_ble_notify_value( rsi_connected_dev_addr, notification_tx_handle, max_data_length, (uint8_t *) read_data1);`
		`if(status != RSI_SUCCESS)`
		`{`
		`if (status == RSI_ERROR_BLE_DEV_BUF_FULL)`
			Application flow:
			APIs should check for buffer full response from RS9116W module. Module returns '**RSI_ERROR_BLE_DEV_BUF_FULL**' to host whenever all buffers associated to that

Change/ Enhancement	Description	Example	Resolution for Backward Compatibility

	between host (application) and RS9116W (firmware)
	
	Type:
	
	Enhancement
	
	Existing Mechanism (1.X.X):
	
	In earlier design, connection based buffer management was not supported. This effects data throughput, connectivity and simultaneous connections in different ways.
	
	In this design, when there is a data write request to remote BLE device from application, RS9116W provides response only after receiving the response from the remote device. There was no flow control mechanism in this case and one connection can consume all the buffers in RS9116W impacting other connection's control and data flows.
	
	New Mechanism (2.X.X):
	
	In the new design, connection based buffer management is implemented (using dedicated buffers for each connected device). If there are no buffers available in the RS9116W module, it returns an error to the host with that particular device address, stating that buffers are not available for that connection. Once the buffer are freed (module serves pending events), then the module will send an event (more data request) to the host requesting to send the data.
	
	• **New API/Event:** *ble_on_le_more_data_req_event*
	
	Impact:
	
	All existing applications which use attribute write, long write or prepare write should be changed as per the new design, otherwise will not work.
		{
		#if RSI_DEBUG_EN
		LOG_PRINT_D("\r\n notify %d failed with buffer full error -conn%d \r\n", notfy_cnt, l_conn_id);
		#endif
		
		rsi_ble_clear_event_based_on_conn(l_conn_id,RSI_DATA_TRANSMIT_EVENT);
		
		rsi_current_state[l_conn_id] |= BIT64(RSI_DATA_TRANSMIT_EVENT);
		break;
		}
		}
			connections are occupied. When this error is observed, application should clear '**RSI_DATA_TRANSMIT_EVENT**' for that particular connection and should not call the API again buffers in the module are freed.
			
			if (status == RSI_ERROR_BLE_DEV_BUF_FULL) { #if RSI_DEBUG_EN LOG_PRINT_D("\r\n notify %d failed with buffer full error -conn%d \r\n", notfy_cnt, l_conn_id); #endif rsi_ble_clear_event_based_on_conn(l_conn_id, RSI_DATA_TRANSMIT_EVENT); rsi_current_state[l_conn_id] |= BIT64(RSI_DATA_TRANSMIT_EVENT); break; }
			
			Once buffers are freed, RS9116W module sends an asynchronous request to host application to indicate module is ready to accept more data from host application. '**RSI_BLE_MORE_DATA_REQ_EVT**' is the asynchronous event sent by the module.
			
			Upon receiving this event, host application should set '***RSI_DATA_TRANSMIT_EVENT***' and resume API calls.
			
			case RSI_BLE_MORE_DATA_REQ_EVT: { rsi_ble_clear_event_based_on_conn(l_conn_id, RSI_BLE_MORE_DATA_REQ_EVT); #if RSI_DEBUG_EN LOG_PRINT_D("\r\n more data request - conn%d \r\n" ,l_conn_id); #endif

Change/ Enhancement	Description	Example	Resolution for Backward Compatibility

	Note:
	This impacts only BLE data flows. Not applicable for BT-Classic.
			if(rsi_current_state[l_conn_id] & BIT64(RSI_DATA_TRANSMIT_EVENT))
			{
			
			rsi_current_state[l_conn_id] &= ~BIT64(RSI_DATA_TRANSMIT_EVENT);
			
			rsi_ble_set_event_based_on_conn(l_conn_id, RSI_DATA_TRANSMIT_EVENT);
			}
			...
			...
			
			For following APIs this flow should be implemented,
			• rsi_ble_get_profiles_async()
			• rsi_ble_get_profile_async()
			• rsi_ble_get_char_services_async()
			• rsi_ble_get_att_value_async()
			• rsi_ble_set_att_value_async()
			• rsi_ble_set_att_cmd()
			• rsi_ble_indicate_value()
			• rsi_ble_notify_value()
			• rsi_ble_set_att_value()
			• rsi_ble_set_att_value_async()

18 Revision History

Revision No.	Version No.	Date	Changes
1	1.0	November 2017	Advance version
2	1.2	June 2018	1. Added rsi_wlan_filter_broadcast API description 2. Added command type for multicast 3. Clarified the differences between WiSeConnect/WiSeMCU
3	1.3	June 2018	Alignment issues are resolved
4	1.4	July 2018	Structural issues are resolved
5	1.5	June 2019	Added rsi_get_ram_log API
6	1.51	September 2019	• Modified the BLE CBFC API's and other missed API's , Parameters • Modified the rsi_ble_gap_register_callbacks structure and added the new events • Updated missing API's for BT Classic
9	1.6	October 2019	Added SAPI error codes
10	1.7	November 2019	Added SAPI error codes
11	1.8	January 2020	• Added MITM params info for BLE section. • Modified the GATT Write event structure
13	1.9	February 2020	Added async behaviour note for rsi_connect() api
14	1.10	April 2020	• Added WLAN STATS in rsi_wlan_get API • Added Weekday in set and get rtc timer • Added New rsi_bt_set_bd_addr API and its related information
15	2.0	September 2020	**Document Structure Changes:** New sections for API categories: 1. Driver APIs 2. NWK APIs 3. API Configuration, Mechanism Changes and Enhancements' which describes major changes in SAPI library from previous release. Modified sections: 1. Moved 'BSD Socket API' section from WLAN APIs to 'NWK APIs' section. 2. Moved 'Network Application Protocol' section from WLAN APIs to 'NWK APIs' section. 3. Moved 'WLAN API call sequence examples' from WLAN APIs to 'Appendix A: WLAN API Call Sequence Examples'. 4. Renamed 'RS9116 Connectivity Resources' section to 'RS9116W Resources'. Removed sections: 5. RS9116 Connectivity Resources

Revision No.	Version No.	Date	Changes
			6. BT-Classic and BT-LE Common Features **Other Changes:** 1. Added description for the bits BIT(5), BIT(6) and BIT(7) in 'FLAGS' parameter in all the HTTP_CLIENT related API's. 2. Added supported bits information of MODE argument in 'rsi_config_ipaddress()'. 3. Added note in rsi_wireless_init(), regarding coex mode memory configuration. 4. Update BT Classic and BLE APIs. 5. Removed WMCU references. 6. Corrected maximum length of SSID to 32 bytes from 34 bytes. 7. Added EMB_MQTT_APIs. 8. Updated description, response parameters and parameters for some APIs. 9. Added description for parameter payload length in rsi_wlan_update_gain_table() API. 10. Added missing APIs from SAPI library. 11. Added the information required for the support of 3 SSL_Certificates loaded to FLASH in rsi_wlan_set_certificate_index( ) API 12. Modified the parameters of MODE argument in 'rsi_config_ipaddress( )' API. 13. Added description for the bits BIT(2), BIT(3) and BIT(4) in 'FLAGS' parameter in all the HTTP_CLIENT related API's. 14. Added note points for corresponding parameters of rsi_wlan_get() API in Master WLAN API. 15. Removed PBAP API, HFP, PBAP register callbacks, AVRCP register callbacks from Master Bluetooth classic APIs 16. Added precondition for rsi_mqtt_poll_for_recv_data API in NWK API section. 17. Added a note in rsi_socket API in NWK APIs section. 18. Updated the rsi_wlan_scan() in WLAN APIs section. 19. Added a note WLAN_STATS only support in AP and STATION mode in WLAN API section. 20. Added note about 384K in wireless_init() API. 21. USB and SDIO Interfaces are removed from Host Interfaces section, as they are not currently supported. 22. Modified ‘Socket create command response’ structure according to the source code. 23. Modified NOTE for 40MHz limitation. 24. Added Operating modes in Features section. 25. Added information for GPIO based power save handshake mechanism in ‘rsi_wlan_ power_save_profile’ section. 26. Added recommended api flow for restarting RS9116W module from host in ‘rsi_wireless_deinit’ section. 27. Added supported Curve IDs for Ciphers in section 'Configure SSL parameters'. **Newly added APIs:** **WLAN:** 1. int32_t rsi_wlan_socket_get_status(int32_t sockID) 2. int32_t rsi_wlan_get_nwk_status(void)

Revision No.	Version No.	Date	Changes
			3. int32_t rsi_wlan_scan_with_bitmap_options(int8_t *ssid, uint8_t chno, rsi_rsp_scan_t *result,uint32_t length,uint32_t scan_bitmap) 4. int32_t rsi_wlan_scan_async_with_bitmap_options(int8_t *ssid, uint8_t chno,uint32_t bitmap void (*scan_response_handler)(uint16_t status, const uint8_t *buffer, const uint16_t length)) 5. int32_t rsi_wlan_power_save_with_listen_interval(uint8_t psp_mode, uint8_t psp_type,uint16_t listen_interval) 6. static int rsi_load_bootup_params(strruct rsi_common *common) 7. int32_t rsi_set_rtc_timer(module_rtc_time_t *timer) 8. int32_t rsi_get_rtc_timer(module_rtc_time_t *response) 9. int32_t rsi_efuse_write(rsi_efuse_write_t *efuse_write_p) 10. int32_t rsi_efuse_read(rsi_efuse_read_t *efuse_read_p, uint8_t *buf, uint32_t length) 11. int32_t rsi_flash_mem_wr(rsi_flash_write_t *write_req) 12. int32_t rsi_reset_mac() 13. uint8_t translate_channel(uint8_t channelNum) 14. int32_t rsi_radio_caps(uint8_t operating_channel) 15. int32_t rsi_get_ram_dump(uint32_t addr, uint16_t length, uint8_t *buf) 16. int32_t rsi_certificate_valid(uint16_t valid, uint16_t socket_id) 17. void rsi_select_set_status(int32_t status,int32_t selectid) 18. int32_t rsi_select_get_status(int32_t selectid) 19. int rsi_fd_isset(uint32_t fd, struct rsi_fd_set_s *fds_p) 20. void rsi_set_fd(uint32_t fd, struct rsi_fd_set_s *fds_p) 21. void rsi_fd_clr(uint32_t fd, struct rsi_fd_set_s *fds_p) **BT/BLE:** 1. int32_t rsi_ble_start_scanning_with_values(void *rsi_ble_scan_params) 2. int32_t rsi_ble_set_local_irk_value (uint8_t *l_irk) 3. int32_t rsi_ble_set_wo_resp_notify_buf_info (uint8_t *dev_addr,uint8_t buf_mode,uint8_t buf_cnt) 4. int32_t rsi_ble_gatt_write_response(uint8_t *dev_addr,uint8_t type) 5. int32_t rsi_ble_gatt_prepare_write_response(uint8_t *dev_addr,uint16_t handle,uint16_t offset,uint16_t length,uint8_t *data) 6. int32_t rsi_bt_change_pkt_type(uint8_t *remote_dev_addr, uint16_t pkt_type) 7. int32_t rsi_bt_set_bd_addr(uint8_t *dev_addr) 8. int32_t rsi_memory_stats_enable (uint8_t protocol, uint8_t memory_stats_enable, uint32_t memory_stats_interval_ms) 9. void update_modified_mtu_size (uint16_t rem_mtu_size) 10. int32_t rsi_bt_per_cw_mode(struct rsi_bt_per_cw_mode_s *bt_cw_mode) **Driver:** 1. uint32_t rsi_bt_get_timeout(uint16_t cmd_type, uint16_t protocol_type)

Revision No.	Version No.	Date	Changes
			2. void (*rsi_bt_async_callback_handler) (rsi_bt_cb_t *cb, uint16_t type, uint8_t *data, uint16_t length)) 3. void rsi_ble_update_le_dev_buf(rsi_ble_event_le_dev_buf_ind_t *rsi_ble_event_le_dev_buf_ind) 4. void rsi_add_remote_ble_dev_info(rsi_ble_event_enhance_conn_status_t *remote_dev_info) 5. void rsi_remove_remote_ble_dev_info(rsi_ble_event_disconnect_t *remote_dev_info) 6. void rsi_bt_pkt_change_events_register_callbacks (rsi_bt_pkt_change_stats_t bt_pkt_change_stats_event) 7. void rsi_bt_on_chip_memory_status_callbacks_register (rsi_bt_on_chip_memory_stats_handler_t bt_on_chip_memory_stats_event) 8. int32_t rsi_check_and_update_cmd_state(uint8_t cmd_type,uint8_t cmd_state) 9. int32_t rsi_atoi(const int8_t* str) 10. uint64_t ip_to_reverse_hex(char *ip) 11. int16_t rsi_nlink_pkt_rd(uint8_t *buf, uint16_t total_len) 12. int32_t rsi_get_ram_log(uint32_t addr, uint32_t length) 13. int32_t rsi_get_fw_version(uint8_t *response, uint16_t length) 14. int32_t rsi_common_debug_log(int32_t assertion_type, int32_t assertion_level) 15. int32_t rsi_driver_deinit(void) 16. int32_t rsi_device_deinit(void) 17. int8_t asciihex_2_num(int8_t ascii_hex_in) 18. int8_t rsi_charhex_2_dec (int8_t *cBuf) 19. void rsi_ascii_mac_address_to_6bytes(uint8_t *hexAddr, int8_t *asciiMacAddress) 20. void rsi_ascii_dot_address_to_4bytes(uint8_t *hexAddr, int8_t *asciiDotAddress) 21. int32_t rsi_gpio_pininit(uint8_t pin_num, uint8_t configuration) 22. int32_t rsi_gpio_writepin(uint8_t pin_num, uint8_t value) 23. int32_t rsi_gpio_readpin(uint8_t pin_num,uint8_t *gpio_value)

19 Appendix A: WLAN API Call Sequence Examples

This section explains the sequence of API calls to configure the module in different modes.

Station Mode

The following flowchart briefs the sequence of API calls to configure module in station mode and connect to an access point. Edit rsi_wlan_config.h for the required configuration.

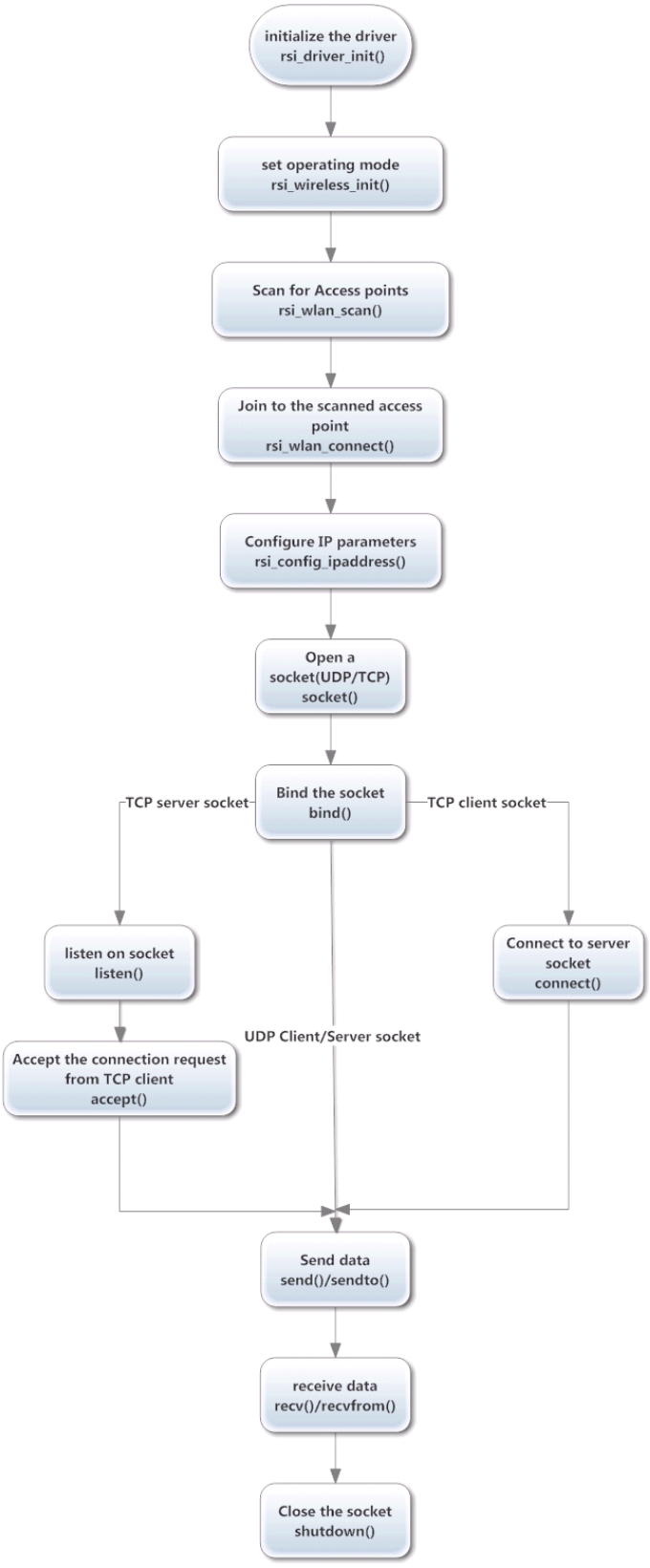


Figure 24: API Flow to Configure Module in Station Mode

The following example illustrate the data transfer using rsi_socket_async() API.

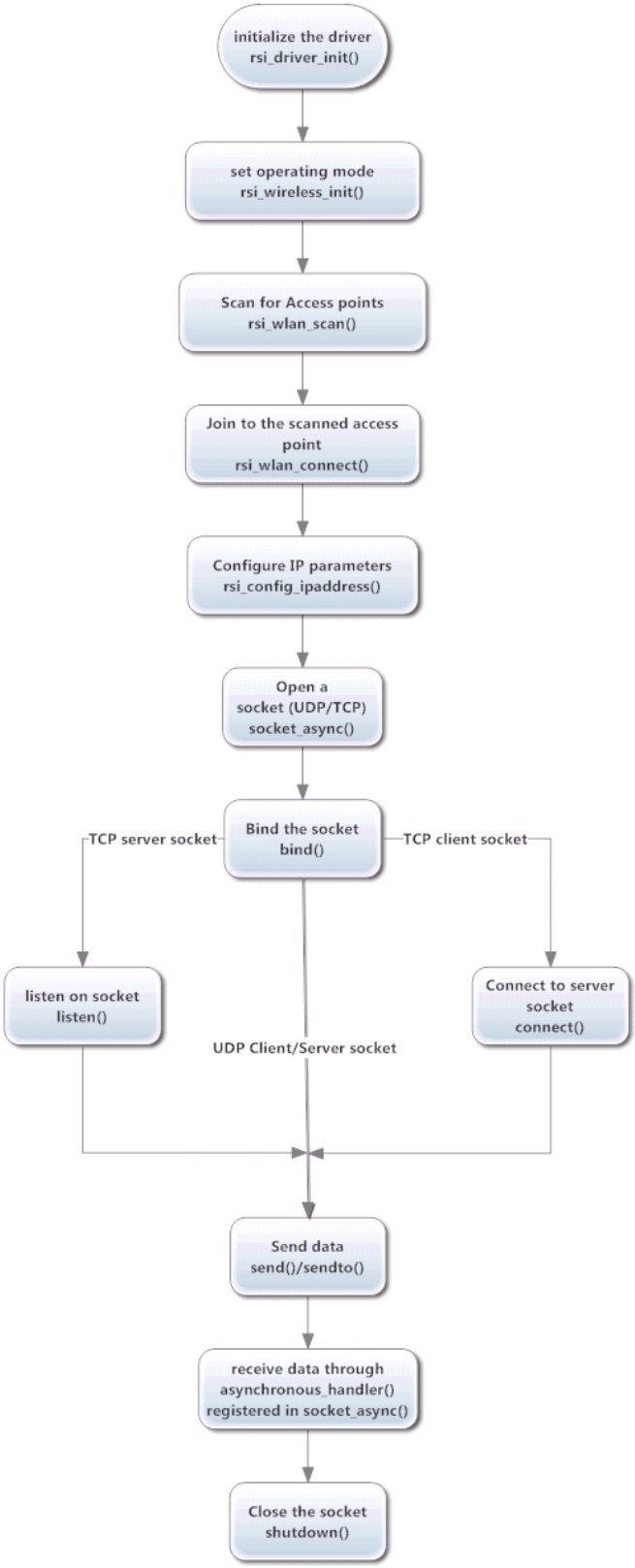


Figure 25: API Flow to Configure Module in Station Mode

Access Point Mode

The following flowchart briefs the sequence of API calls to configure module in access point mode. Edit rsi_wlan_config.h for the required features.

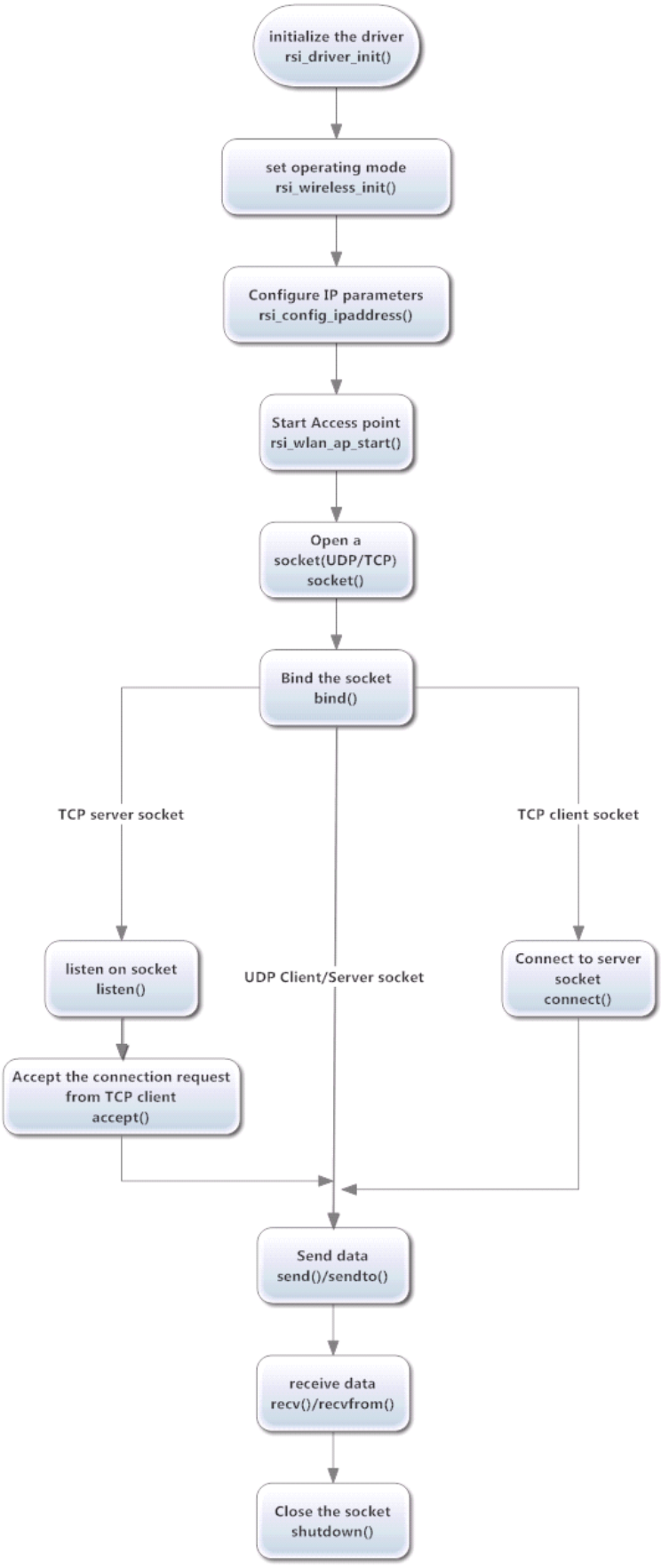
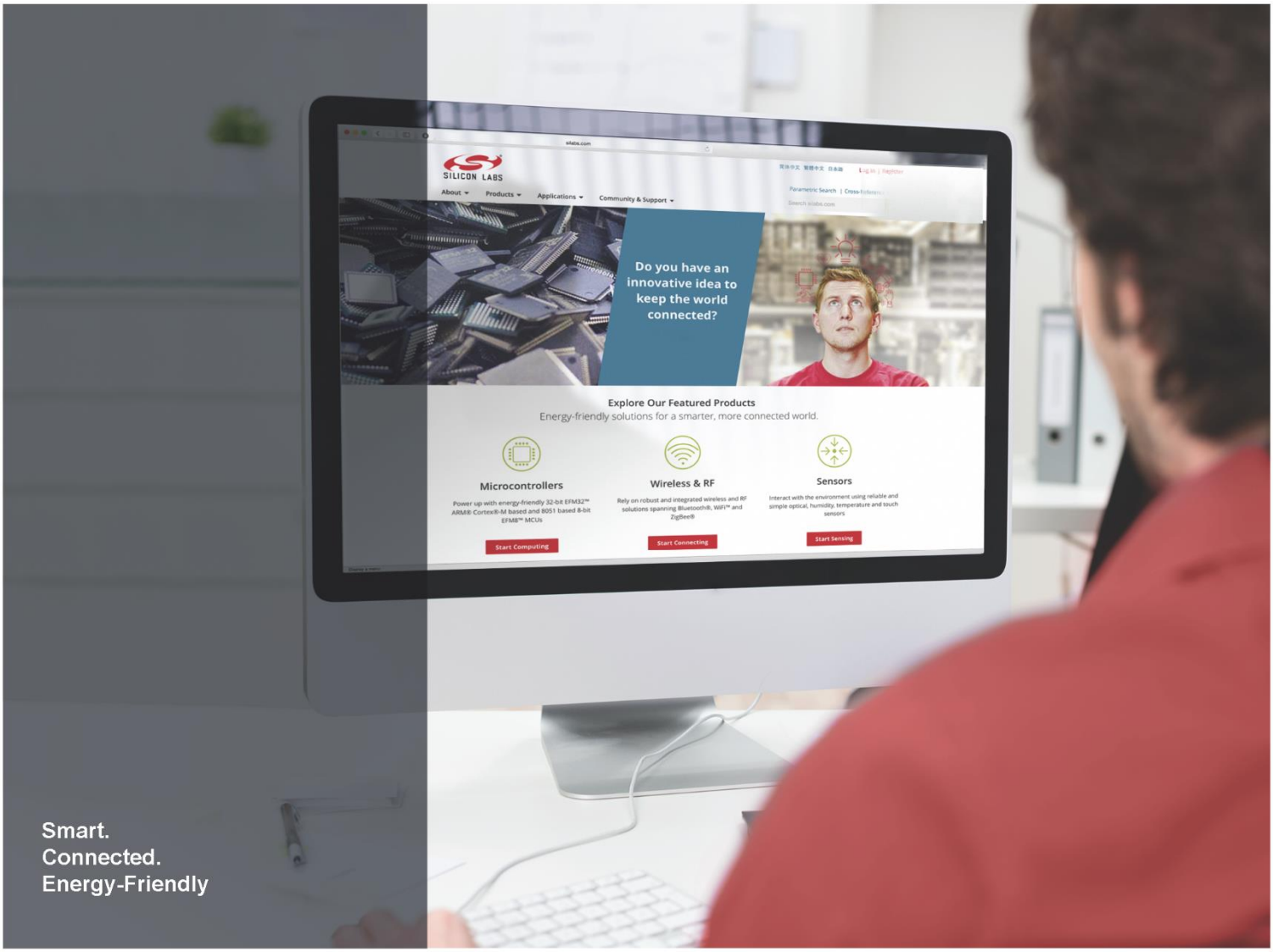


Figure 26: API Flow to Configure Module in Access Point Mode



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

http://www.silabs.com