RS14100_RS9116 Wireless SAPI Manual

Version 1.7

April 2019

Redpine Signals, Inc.

2107 North First Street, Suite #540, San Jose, California 95131,

United States of America.

Phone: +1-408-748-3385, Fax: +1-408-705-2019

Email: sales@redpinesignals.com

Website: www.redpinesignals.com

Table of Contents

About this Document

This document is Redpine Wireless SAPI Guide for the customers

1 Overview of Wireless SAPI

RS9116 WiSeConnect and R14100 WiSeMCU products include Wi-Fi, TCP/IP and BT 5 stacks embedded in its internal flash memory. The RS9116 WiSeConnect requires a separate application processor while the RS14100 includes the application processor in the same package. This document describes the Redpine Simple API ("SAPI") for the application processor to access the embedded stack features.

Note:
These APIs are applicable to RS9116 WiSeConnect, RS14100 WiSeMCU and other WiSeMCU products.

2 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, USB-CDC), which enables easy migration to different host interfaces.
- Supports bare-metal and FreeRTOS out-of-box. Other RTOSes can be supported through OS Abstraction changes.
- Supports Keil uvision and IAR IDEs – can be ported to other toolchains.

2.1 Wi-Fi

- Supports Wi-Fi Station, Access Point, WFD, WPS, and WPA/Enterprise.
- Supports BSD socket interface
- Supports interfaces for embedded Application protocols, such as HTTP, SMTP, SNMP, FTP and MQTT.

2.2 BT/BLE

- Supports GAP (BT/BLE)
- Supports GATT Client and Server interfaces (BLE)

3 Wireless SAPI Related Resources

The following Table contains guides, examples, and references for developing applications with WiSeConnect and WiSeMCU Documentation, Software Packages, and more are available on Redpine's document portal. Contact Redpine Sales office to obtain the NDA and instructions to login.

3.1 RS14100 WiSeMCU Resources

Name	Location on Redpine Document Portal
RS14100 WiSeMCU Family Common Documents	
RS14100 WiSeMCU Module Family Datasheet	*/RS14100 WiSeMCU/Datasheets, Manuals, and Guides/*
RS14100 EVK User Guide	*/RS14100 WiSeMCU/EVK/*
Module Integration Guide	*/RS14100 WiSeMCU/Datasheets, Manuals, and Guides/*
Regulatory and Compliance Certificates	
FCC compliance certificates	*/RS14100 WiSeMCU/Certifications and Declarations/*
CE compliance certificates	*/RS14100 WiSeMCU/Certifications and Declarations/*
IC compliance certificates	*/RS14100 WiSeMCU/Certifications and Declarations/*
RS14100 WiSeMCU Documents	
WiSeMCU Getting Started Guide	*/Application Notes/*
WiSeMCU Software Package including examples	*/RS14100 WiSeMCU/Firmware/*
WiSeMCU SAPI Guide	*/RS14100 WiSeMCU/Firmware/*
WiSeMCU SAPI Porting Guide	*/RS14100 WiSeMCU/Firmware/*
Miscellaneous Resources	
3D Models	*/RS14100 WiSeMCU/CAD Files/*
IBIS Models	*/RS14100 WiSeMCU/CAD Files/*
Application Notes	*/Application Notes/*

3.2 RS9116 Connectivity Resources

Name	Location on Redpine Document Portal
RS9116 Connectivity Family Common Documents	
RS9116 Connectivity Module Family Datasheet	*/RS9116 Connectivity/Datasheets, Manuals, and Guides/*
RS9116 EVK User Guide	*/RS9116 Connectivity/EVK/*
Module Integration Guide	*/RS9116 Connectivity/Datasheets, Manuals, and Guides/*
Regulatory and Compliance Certificates	
FCC compliance certificates	*/RS9116 Connectivity/Certifications and Declarations/*
CE compliance certificates	*/RS9116 Connectivity/Certifications and Declarations/*
IC compliance certificates	*/RS9116 Connectivity/Certifications and Declarations/*
RS9116 WiSeConnect Documents	
WiSeConnect Getting Started Guide	*/Application Notes/*
WiSeConnect Software Package including examples	*/RS9116 Connectivity/Software/Firmware/*
WiSeConnect Programmer's Reference Manual	*/RS9116 Connectivity/Software/Firmware/*
WiSeConnect SAPI Guide	*/RS9116 Connectivity/Software/Firmware/*
WiSeConnect SAPI Porting Guide	*/RS9116 Connectivity/Software/Firmware/*
RS9116 n-Link Documents	
n-Link Software Package	*/RS9116 Connectivity/Software/*
n-Link Technical Reference Manual	*/RS9116 Connectivity/Software/*
Miscellaneous Resources	
3D Models	*/RS9116 Connectivity/CAD Files/*
IBIS Models	*/RS9116 Connectivity/CAD Files/*
Application Notes	*/Application Notes/*

4 SAPI Architecture

SAPI APIs are designed in layers, where each Layer is independent and uses the service of underlying layers. Figure 1 describes the WiSeConnect API architecture and Figure 2 describes the WiSeMCU architecture. The SAPIs used in WiSeMCU and WiSeConnect are the same unless specified in a section. In case of WiSeMCU, they are running on the internal Cortex-M4F processor and in case of WiSeConnect, they are running on the external host processor.

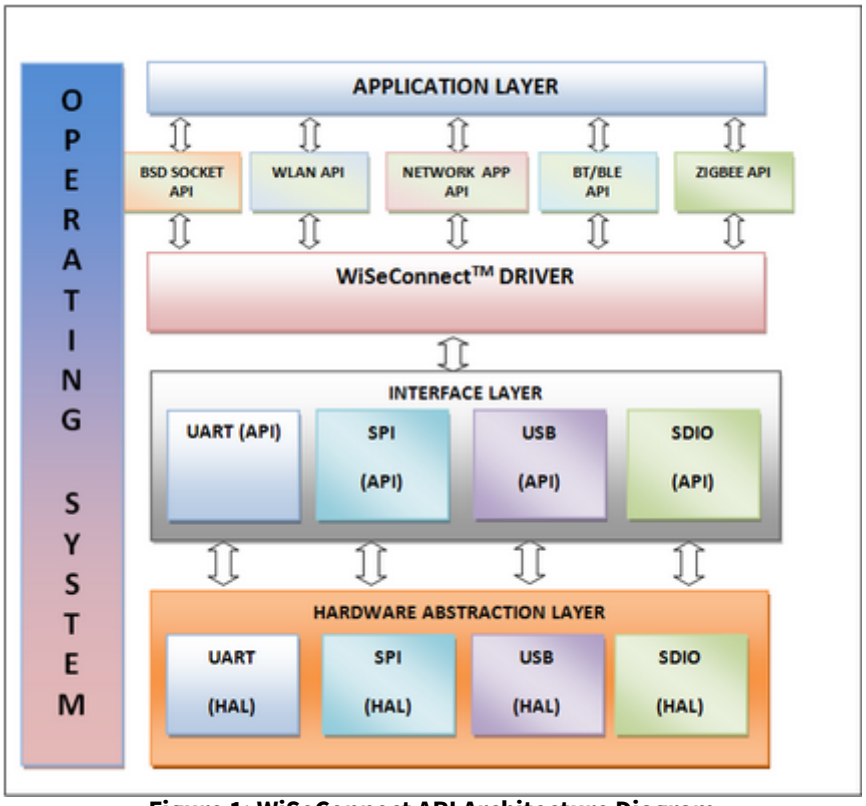


Figure 1: WiSeConnect API Architecture Diagram

Application Layer
Application Layer contains application specific functionality. Application Layer needs to call WiSeConnectDriver APIs to configure and operate the module.

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.

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 the application to initialize and configure the embedded TCP/IP stack and perform data transfers.

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

Interface Specific API Layer
The module supports 4 different host interfaces (SPI, USB, UART, 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 API Layer
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.

For WiSeMCU, the software architecture is similar to WiSeConnect expect that there is no need for the HAL and Interface layers. The SAPI layer is the same between the two unless specified in a section.

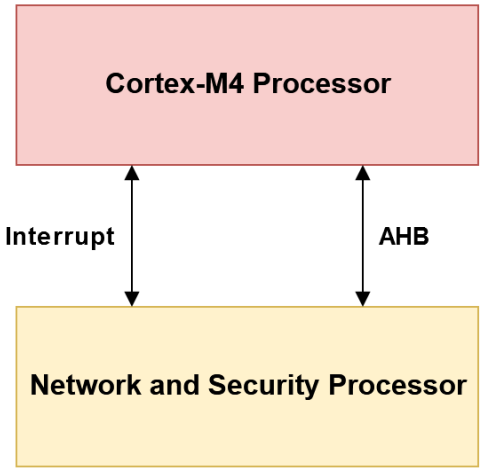


Figure 2: WiseMCU Architecture Diagram

5 Common API

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

5.1 rsi_driver_init

Prototype

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

Description
This API initializes the WiSeConnect/WiSeMCU 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
< 0	Failure. -1: if UART initialization fails in SPI / UART mode -2: if maximum sockets is greater than 10

Example

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

5.2 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;
 }
```

5.3 rsi_device_init

Prototype

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

Description
This API power cycles the module and set the boot up option for WiSeConnect/WiSeMCU 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;
 }
```

5.4 rsi_bl_module_power_off

Prototype

```
int32_t rsi_bl_module_power_off(void);
```

Description
This API turns off the WiSeConnect/WiSeMCU device.

Precondition
N/A

Parameters
None

Return Values

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

Example

```
// power cycle the device
 status = rsi_bl_module_power_off();
 status = rsi_bl_module_power_on();
 // reinitialize device (drivers already initialized)
 status = rsi_device_init(RSI_LOAD_IMAGE_I_FW);
 if(status != RSI_SUCCESS)
 {
        return status;
 }
```

5.5 rsi_bl_module_power_on

Prototype

```
int32_t rsi_bl_module_power_on(void);
```

Description
This API turns on the WiSeConnect/WiSeMCU device.

Precondition
N/A

Parameters
None

Return Values

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

Example

```
// power cycle the device
 status = rsi_bl_module_power_off();
 status = rsi_bl_module_power_on();
 // reinitialize device (drivers already initialized)
 status = rsi_device_init(RSI_LOAD_IMAGE_I_FW);
 if(status != RSI_SUCCESS)
 {
        return status;
 }
```

5.6 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/WiSeMCU 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);
```

5.7 rsi_wireless_init

Prototype

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

Description
This API initializes the WiSeConnect/WiSeMCU 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
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

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 **If return value is greater than 0** 0x0021,0x0025,0xFF73,0x002C,0xFF6E,0xFF6F, 0xFF70,0xFFC5

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

Example

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

5.8 rsi_wireless_deinit

Prototype

```
int32_t rsi_wireless_deinit()
```

Description
This API de-initializes WiSeConnect/WiSeMCU software feature. This API should be called before rsi_wireless_init command if user wants to change the previous configuration.

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

```
// Deinitialize after calling rsi_wireless_init().
// Call rsi_wireless_init() again to reinitialize.
rsi_wireless_deinit();
```

5.9 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();
 }
```

5.10 rsi_wireless_antenna

Prototype

```
int32_t rsi_wireless_antenna(uint8_t type, uint8_t gain_2g,
                                                    uint8_t
gain_5g,uint8_t antenna_path, uint8_t antenna_type)
```

Description
This API configures the antenna.

Precondition
N/A

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
antenna_path	RSI_ANTENNA_PATH_INTERNAL, RSI_ANTENNA_PATH_EXTERNAL
antenna_type	RSI_ANTENNA_TYPE_REDPINE, RSI_ANTENNA_TYPE_FRACTUS, RSI_ANTENNA_TYPE_MOLEX

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

Return Values

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

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

Example

```
// select antenna and specify gain
 rsi_wireless_antenna(RF_OUT_2, 5,
0,RSI_ANTENNA_PATH_INTERNAL,RSI_ANTENNA_TYPE_REDPINE);
```

5.10.1 rsi_send_feature_frame

Prototype

```
int32_t rsi_send_feature_frame();
```

Description
This API is used to select internal RF type or External RF type and clock frequency.

Precondition
rsi_wireless_init() 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	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, 0xFF74

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

Example
See <SW Package Path>host/sapis/examples/power_measurement/wlan_standby/rsi_wlan_standby.c

```
rsi_send_feature_frame();
```

See <SW Package Path>host/sapis/include/rsi_wlan_config.h file and update/modify following feature frame macros

```
#define PLL_MODE              0
#define RF_TYPE               1
#define WIRELESS_MODE         0
#define AFE_TYPE              1
#define FEATURE_ENABLES       0
```

Parameter

Value	Description
PLL_MODE	0-PLLMODE0 - Used for generating the clocks for 20Mhz Channel Bandwidth operations. (default) 1-PLLMODE1 - Used for generating the clocks for 40Mhz Channel Bandwidth operations. 2-PLLMODE2-Reserved
RF_TYPE	0-External_RF_8111 1-Internal_RF_9116 (default)
WIRELESS_MODE:	12 – To enable LP chain for PER mode 0 - LP chain disable (default)
RESERVED	0

Value	Description
AFE	0 - AFE BYPASS, 1 - Internal AFE (default)
FEATURE_ENABLES	BIT[4] - To enable LP chain for stand-by associate mode. BIT[5] - To enable hardware beacon drop during power save. **Note:** Remaining bits are not user configurable.

Note: The default values provided are valid for all module variants and no change is necessary.

6 WLAN API

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

6.1 WLAN Core API

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

6.1.1 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.
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 network information. This is an output parameter.
length	This is the length of the resulted buffer measured in bytes to hold scan results.

Channels supported in 2.4 GHz Band

Channel Number	chno
All channels	0
1	1
2	2
3	3

Channel Number	chno
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 2.4 GHz Band

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

Channel Number	chno
153	153
157	157
161	161
165	165

Channels supported in 5GHz band

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_scan_info_s
 {
        uint8_t rf_channel;
        uint8_t security_mode;
        uint8_t rssi_val;
        uint8_t network_type;
        uint8_t ssid[34];
        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

Structure Fields	Description
rssi_val	This is the RSSI value of the Access Point
network_type	This is the type of the network 1 : Infrastructure mode
ssid	This is the SSID of an access point
bssid	This is the MAC address of an access point

```
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_info	This is the information about scanned Access Point in rsi_scan_info_t structure

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 **If return value is greater than 0** 0x0002, 0x0003, 0x0005, 0x000A, 0x0014, 0x0015, 0x001A, 0x0021,0x0024,0x0025,0x0026,0x002C,0x003c

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

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);
```

6.1.2 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)))
```

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.
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 response buffer length

Channels supported in 2.4GHz band:

Channel Number	ch no
All channels	0

Channel Number	ch no
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

2.4GHz Band Channel Mapping

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

Channel Number	chno
136	136
140	140
149	149
153	153
157	157
161	161
165	165

5GHz Band Channel Mapping

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_scan_info_s
{
 uint8_t rf_channel;
 uint8_t security_mode;
 uint8_t rssi_val;
 uint8_t network_type;
 uint8_t ssid[34];
 uint8_t bssid[6];
 uint8_t reserved[2];
 }rsi_scan_info_t;
```

Structure Fields	Description
rf_channel	This is the access point channel number

Structure Fields	Description
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

```
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_info	This is the information about scanned Access points in rsi_scan_info_t structure

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** 0x0002,0x0003,0x0005,0x000A,0x0014,0x0015,0x001A, 0x0021, 0x0024, 0x0025, 0x0026, 0x002C, 0x003c

Please refer to WLAN Error codes 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;
        }
 }
```

6.1.3 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, 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_WPA2	PSK string terminated with NULL. Length of PSK should be at least 8 and less than 64 bytes.

Security type (sec_type)	secret key structure format (secret_key)
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 indicate to use PIN generated using rsi_wps_generate_pin API

Security type (sec_type)	secret key structure format (secret_key)
RSI_WPS_PUSH_BUTTON	NULL string indicate 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 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** 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

Please refer to WLAN Error codes 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;
}
```

6.1.4 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 **Length:** This is the response buffer length

Security type (sec_type)	secret key structure format (secret_key)
RSI_OPEN	No secret key in open security mode.
RSI_WPA2	PSK string terminated with NULL. Length of PSK should be at least 8 and less than 64 bytes.

Security type (sec_type)	secret key structure format (secret_key)
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; **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 indicate to use PIN generated using rsi_wps_generate_pin API.

Security type (sec_type)	secret key structure format (secret_key)
RSI_WPS_PUSH_BUTTON	NULL string indicate to generate push button event.

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 **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

Please refer to WLAN Error codes 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);
```

6.1.5 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
None

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 **If return value is greater than 0** 0x0006,0x0021,0x002C,0x004A,0x0025,0x0026

Please refer to WLAN Error codes for a description of the above error codes.

6.1.6 rsi_wlan_disconnect

Prototype

```
int32_t rsi_wlan_disconnect();
```

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

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** 0x0006,0x0021,0x002C,0x0015

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

Example

```
rsi_wlan_disconnect();
```

6.1.7 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.

Parameters

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’.

Value	Description
Non Zero Value	Failure **If return value is greater than 0** 0x0021, 0x0025,0x0026,0x0028,0x002C,0x0039,0x003a, 0x003b

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

Example
rsi_wlan_pmk_generate(PMK_TYPE, (int8_t *)PSK, (int8_t *)SSID, pmk, PMK_SIZE);

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/WiSeMCU 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

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** 0x0015,0x0021,0x0025,0x0026,0x002C

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

Example

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

6.1.8 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();
```

6.1.9 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.

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

Parameter	Description
Band	1 → 2GHz 2 → 5GHz
Bandwidth	0 → 20 MHz 1 → 40 MHz
payload	Pass channel gain values for different regions in an array
payload_len	Max payload length is 128 bytes

Note:
1. Length of the payload should match with payload_len parameter value.

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

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

Example

```
 // query the status
 int32_t status = int32_t rsi_wlan_update_gain_table(BAND, BANDWIDTH,
payload,PAYLOAD_LEN);
```

6.1.10 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

Parameter	Description
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
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 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,0x0026,0x004C,0x0028,0x001A,0x000A,0x001D

Please refer to WLAN Error codes 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;
 }
```

6.1.11 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 lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021

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

Example

```
rsi_wlan_wps_push_button_event((int8_t *)SSID);
```

6.1.12 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

Value	Description
Non Zero Value	Failure **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, 0x0037, 0x0038

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

Example

```
rsi_wlan_wps_generate_pin(wps_pin,wps_pin_length);
```

6.1.13 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 lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0013, 0x0021, 0x002C, 0x0015

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

Example

```
rsi_wlan_disconnect_stations(station_mac_address);
```

6.1.14 rsi_wlan_wfd_start_discovery

Prototype

```
int32_t rsi_wlan_wfd_start_discovery(
 uint16_t go_intent,
 int8_t *device_name,
 uint16_t channel,
 int8_t *ssid_post_fix,
 uint8_t *psk,
 void (*wlan_wfd_discovery_notify_handler)
 (uint16_t status,
 const uint8_t *buffer,
 const uint16_t length),
 void (*wlan_wfd_connection_request_notify_handler)
 (uint16_t status,
 const uint8_t *buffer,
 const uint16_t length))
```

Description
This API starts the discovery in Wi-Fi-Direct mode.

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

Parameters

Parameter	Description
go_intent	This determines whether the device is intended to form a GO (group owner) or work as a Wi-Fi Direct Peer node. This value is used in the GO negotiation process, when the module negotiates with another Wi-Fi Direct Node on who would become the Group Owner. The valid range of values for this parameter is: 0 to 16. Higher the number, higher is the willingness of the module to become a GO. If the number is between 0 and 15, a GO negotiation takes place. If the value is 16, the module forms an Autonomous GO without negotiating with any other device.
device_name	This is the device name for the module. The maximum length of this field is 32 characters and the remaining bytes are filled with 0x00. Another Wi-Fi Direct device would see this name when it scans for Wi-Fi Direct nodes.
channel	This is the operating channel number. The specified channel is used if the device becomes a GO or Autonomous GO

Parameter	Description
ssid_post_fix	This parameter is used to add a postfix to the SSID in Wi-Fi Direct GO mode and Autonomous GO mode. **Note:** ssid_post_fix should be maximum of 23 bytes.
psk	This is a passphrase of a maximum length of 63 characters (a null character should be supplied to make it 64 bytes in the structure). This PSK is used if the module becomes a GO owner.
wlan_wfd_discovery_notify_handler	This is the asynchronous message sent from module to the host when module finds any Wi-Fi Direct node. The parameters involved are status, buffer & length **Status:** This is the response status If status is zero, it means that the wfd device response has some device information **Buffer:** This is the response buffer **Length:** This is the response buffer length
wlan_wfd_connection_request_notify_handler	This is the asynchronous message sent from module to the host when module receives a connection request from any remote Wi-Fi Direct node. The parameters involved are status, buffer & length **Status:** This is the response status If status is zero, it means that the connection request has come from some device. **Buffer:** This is the response buffer **Length:** This is the response buffer length

Response Strucure:

```
typedef struct rsi_wfd_device_info_s
 {
 uint8_t device_state;
 uint8_t device_name[32];
 uint8_t mac_address[6];
 uint8_t device_type[2];
 }rsi_wfd_device_info_t;
 typedef struct rsi_rsp_wfd_device_info_s
 {
 uint8_t device_count;
 rsi_wfd_device_info_t wfd_dev_info[RSI_MAX_WFD_DEVICE_COUNT];
 } rsi_rsp_wfd_device_info_t;
 typedef struct rsi_rsp_p2p_connection_request_s
 {
 uint8_t device_name[32];
 }rsi_rsp_p2p_connection_request_t;
```

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	Failure **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** 0x001D, 0x0021, 0x002C, 0x0015

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

Example

```
 void rsi_wlan_wfd_discovery_notify_handler(uint16_t status,
 const uint8_t *buffer, const uint16_t length)
 { … }
 void rsi_wlan_wfd_connection_request_notify_handler(uint16_t status,
const uint8_t *buffer, const uint16_t length)
 { … }
 // start Wi-Fi Direct discovery

 status = rsi_wlan_wfd_start_discovery(GO_INTENT,
RSI_DEVICE_NAME,CHANNEL, POST_FIX_SSID , PSK,
rsi_wlan_wfd_discovery_notify_handler,
rsi_wlan_wfd_connection_request_notify_handler);
```

6.1.15 rsi_wlan_ wfd_connect

Prototype

```
int32_t rsi_wlan_wfd_connect(int8_t *device_name
 void (*join_response_handler)
 (uint16_t status,
 const uint8_t *buffer,
 const uint16_t length))
```

Description
This API connects to the specified Wi-Fi-Direct device.

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

Parameters

Parameter	Description
device_name	This is the device name of the Wi-Fi Direct node to connect.
join_response_handler	This callback is called when the response for join has come from the module The parameters involved are status, buffer & length **Status:** This is the response status If status is zero, join response is stated as success **Buffer:** This is the response buffer **Length:** This is the response buffer length

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	Failure **If return value is lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0014, 0x0009, 0x0003, 0x0021, 0x002C, 0x0015

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

Example

```
 status = rsi_wlan_wfd_connect(device_name,
rsi_join_response_handler);
```

6.1.16 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
We have to call rsi_wireless_init api first then call 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
response	This is the the response of the requested command. This is an output parameter.

Parameter	Description
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]
RSI_MAC_ADDRESS	uint8_t response[6]
RSI_RSSI	uint8_t response[2]

cmd_type	Response structure
RSI_WLAN_INFO	typedef struct rsi_rsp_wireless_info_s { uint16_t wlan_state; uint16_t channel_number; uint8_t ssid[34]; 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
RSI_CONNECTION_STATUS	

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.

cmd_type	Response structure

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
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

Return Value

Value	Description
0	Successful execution of the command

Value	Description
Non Zero Value	Failure **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

Please refer to WLAN Error codes and SAPI Error Codes for the description of the above error codes.

Example

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

6.1.17 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.

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]; }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 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

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

6.1.18 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 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 lesser than 0** -2 : Invalid parameters -4 : Buffer not available to serve the command **If return value is greater than 0** 0x0015,0xBB21,0xBB4B,0xBB55

Please refer to WLAN Error codes 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);
```

6.1.19 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. 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.

1. psp_type is only valid in psp_mode 1 and 2.
2. psp_type UAPSD is applicable only if WMM_PS is enable in rsi_wlan_config.h file.

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 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, 0xFFF8,0x0015,0x0026,0x0052

Please refer to WLAN Error codes 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;
}
```

6.1.20 rsi_wlan_set_sleep_timer

Prototype
int16_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.

Parameters

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

Pre condition
This command can be issued any time in case of power save mode 9.

Return Values

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

6.1.21 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.

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 dissconnect of the STA 1 - filter_bcast_in_tim is valid till next update by giving the same command

Pre Condition
rrsi_wlan_filter_broadcast() API needs to be called after opermode command only.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**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

Please refer to **WLAN Error codes** for description of the above error codes.

Example

NA

6.1.22 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
option_value	value of the parameter
option_len	length of the parameter

Pre Condition
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 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

Please refer to **WLAN Error codes** for description of the above error codes.

Example
NA

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

6.1.23 rsi_configure_tx_rx_ratio

Prototype
int rsi_configure_tx_rx_ratio()

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

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

Pre Condition
rsi_configure_tx_rx_ratio() API needs to be called after opermode command only.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**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

Please refer to **WLAN Error codes** for description of the above error codes.

Example
NA

6.1.24 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 lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002c, 0x000A

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

6.1.25 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 **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

Please refer to WLAN Error codes for a description of the above error codes.

6.1.26 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 lesser than 0** -2: Invalid Parameters -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002c

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

Example

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

6.1.27 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.

> This API is relevant in opermode 8.

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/WiSeMCU 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 .

> 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
> Pass any valid values for rate and length
> 2. Stop Burst mode
> 3. 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
24	137
36	141
48	136
54	140
MCS0	256
MCS1	257
MCS2	258
MCS3	259
MCS4	260
MCS5	261
MCS6	262
MCS7	263

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

Channel Numbers (2.4GHz)	Center frequencies for 20MHz channel width(MHz)
1	2412
2	2417
3	2422
4	2427
5	2432

Channel Numbers (2.4GHz)	Center frequencies for 20MHz channel width(MHz)
6	2437
7	2442
8	2447
9	2452
10	2457
11	2462
12	2467
13	2472
14	2484

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 lesser than 0** -4 : Buffer not available to serve the command **If return value is greater than 0** 0x000A, 0x0021, 0x0025, 0x002C

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

Example

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

6.1.28 rsi_transmit_test_stop

Prototype

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

Description
This API stops the transmit test.

> This API is relevant in opermode 8.

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 lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002C

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

Example

```
rsi_transmit_test_stop(void);
```

6.1.29 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 lesser than 0** -2: Invalid Parameters -4: Buffer not available to serve the command

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

Example

```
rsi_req_wireless_fwup();
```

6.1.30 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

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

Example

```
uint8_t recv_buffer[1000];
rsi_fwup_start(recv_buffer);
```

6.1.30.1 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

Please refer to WLAN Error codes 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);
```

6.1.31 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 uint16_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: 0 - RSI_JOIN_FAIL_CB 1 - RSI_IP_FAIL_CB 2 - RSI_REMOTE_SOCKET_TERMINATE_CB 3 - RSI_IP_CHANGE_NOTIFY_CB 4 - RSI_STATIONS_CONNECT_NOTIFY_CB 5 - RSI_STATIONS_DISCONNECT_NOTIFY_CB 6 – RSI_WLAN_DATA_RECEIVE_NOTIFY_CB 13 - RSI_WLAN_SOCKET_CONNECT_NOTIFY_CB 14 - RSI_WLAN_SERVER_CERT_RECEIVE_NOTIFY_CB
void(*callback_handler_ptr)(uint16_t *status, const uint8_t *buffer, const uint16_t length)	This is the Call back handler **status:** status of the asynchronous response **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_CONNECT_NOTIFY_CB	This callback is called when asynchronous station connect notification is received from the module in AP mode.
RSI_STATIONS_DISCONNECT_NOTIFY_CB	This callback is called when asynchronous station disconnect 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.

Call back id	Function Description
RSI_WLAN_WFD_DISCOVERY_NOTIFY_CB	This callback is called whenever a Wi-Fidirect device is discovered and its details is given to host.
RSI_WLAN_WFD_CONNECTION_REQUEST_NOTIFY_CB	This callback is called when a connection request comes from the discovered Wi-Fi direct device.
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

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

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

Response structure for PER stats

```
typedef struct rsi_per_stats_rsp_s{
uint8_t tx_pkts[2];
uint8_t reserved_1[2];
uint8_t tx_retries[2];
uint8_t crc_pass[2];
uint8_t crc_fail[2];
uint8_t cca_stk[2];
uint8_t cca_not_stk[2];
uint8_t pkt_abort[2];
uint8_t fls_rx_start[2];
uint8_t cca_idle[2];
uint8_t reserved_2[26];
uint8_t rx_retries[2];
uint8_t reserved_3[2];
uint8_t cal_rssi[2];
uint8_t reserved_4[4];
uint8_t xretries[2];
uint8_t max_cons_pkts_dropped[2];
uint8_t reserved_5[2];
uint8_t bss_broadcast_pkts[2];
uint8_t bss_multicast_pkts[2];
uint8_t bss_filter_matched_multicast_pkts[2];
}rsi_per_stats_rsp_t;
```

Structure members	Description
tx_pkts	Number of TX packets transmitted
reserved_1	Reserved
tx_retries	Number of TX retries happened
crc_pass	Number of RX packets that passed CRC
crc_fail	Number of RX packets that failed CRC
cca_stk	Number of times cca got stuck
cca_not_stk	Number of times cca didn't get stuck
pkt_abort	Number of times RX packet aborts happened
fls_rx_start	Number of false rx starts.If Valid wlan packet is recived and is dropped due to some reasons.
cca_idle	CCA idle time
reserved_2	Reserved

Structure members	Description
rx_retries	Number of RX retries happened
reserved_3	Reserved
cal_rssi	The calculated RSSI value of recently received RX packet
reserved_4	Reserved
xretries	Number of TX Packets dropped after maximum retries
max_cons_pkts_dropped	Number of consecutive packets dropped after maximum retries
reserved_5	Reserved
bss_broadcast_pkts	BSSID matched broadcast packets count.
bss_multicast_pkts	BSSID matched multicast packets count.
bss_filter_matched_multicast_pkts	BSSID and multicast filter matched packets count.

Response structure for WFD discovery and connection request:

```
typedef struct rsi_wfd_device_info_s
 {
 uint8_t device_state;
 uint8_t device_name[32];
 uint8_t mac_address[6];
 uint8_t device_type[2];
 }rsi_wfd_device_info_t;
 typedef struct rsi_rsp_wfd_device_info_s
 {
 uint8_t device_count;
 rsi_wfd_device_info_t wfd_dev_info[RSI_MAX_WFD_DEVICE_COUNT];
 } rsi_rsp_wfd_device_info_t;
 typedef struct rsi_rsp_p2p_connection_request_s
 {
 uint8_t device_name[32];
 }rsi_rsp_p2p_connection_request_t;
```

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[3];
}rsi_cert_recv_t;
```

Examples

```
     //! Register certificate receive notify handler

rsi_wlan_register_callbacks(RSI_WLAN_SERVER_CERT_RECEIVE_NOTIFY_CB,
certificate_response_handler);
```

6.1.32 WLAN network callback handlers.

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

Call back id	Function prototype	Parameters	Function Description
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

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

6.2 BSD Socket API

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

6.2.1 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
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

IPv6 is not supported

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

Please refer to WLAN Error codes 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);
```

6.2.2 socket

Prototype

```
int32_t rsi_socket(int32_t protocolFamily,
int32_t type,
int32_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

Parameter	Description
protocol	0: non SSL sockets 1: SSL sockets

IPv6 is not supported in the current release

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);
```

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

6.2.3 bind

Prototype

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

Description
This API assigns an address to a socket.

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

Parameter	Description
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)
```

6.2.4 connect

Prototype

```
int32_t rsi_connect(int32_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

Parameter	Description
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));
```

6.2.5 listen

Prototype

```
int32_t rsi_listen(int32_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);
```

6.2.6 accept

Prototype

```
int32_t rsi_accept(int32_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);
```

6.2.7 rsi_accept_async

Prototype

```
int32_t  rsi_accept_async(int32_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)
```

6.2.8 recvfrom

Prototype

```
int32_t rsi_recvfrom(int32_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
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);
```

6.2.9 recv

Prototype

```
int32_t rsi_recv(int32_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);
```

6.2.10 sendto

Prototype

```
int32_t rsi_sendto(int32_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));
```

6.2.11 send

Prototype

```
int32_t rsi_send(int32_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);
```

6.2.12 rsi_send_large_data_async

Prototype

```
int32_t rsi_send_large_data_async(int32_t sockID,
 const int8_t *msg,
 int32_t msgLength,
 int32_t flags,
 void (*callback)(uint8_t sockID,
 int16_t status,
 uint16_t total_data_sent));
```

Description
This API sends the large amount of 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	socket descriptor ID
msg	pointer to the buffer containing data to send to the remote peer
msgLength	length of the buffer

Parameter	Description
flags	Reserved
callback	This is the callback when total data packet is received on the created socket. The parameters involved are : sockID, status and total_data_sent **sockID:** Application socket ID **status:** Status of the data transfer. **total_data_sent:** Total length of data sent.

Return Values

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

Example

```
status = rsi_send_large_data_async(client_socket, (int8_t *)"Hello
from TCP client!!!", (sizeof("Hello from TCP client!!!") - 1), 0,
&rsi_sock_data_tx_done_cb);
```

Note:
This API is supported in only WiSeConnect mode.

6.2.13 shutdown

Prototype

```
int32_t rsi_shutdown(int32_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

Parameter	Description
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.

Return Values

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

Example

```
rsi_shutdown(server_socket, 0);
```

6.2.14 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

IPv6 is not supported

Return Values

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

6.2.15 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.

IPv6 is not supported in current release

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

Please refer to WLAN Error codes 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));

6.2.16 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 hostname 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.

IPv6 is not supported in current release

Return Values

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

Please refer to WLAN Error codes 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);
```

6.3 Network Application Protocol

6.3.1 SMTP client API

6.3.1.1 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 an 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

Parameter	Description
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

Please refer to WLAN Error codes 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;
 }
```

6.3.1.2 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 **Note:** The total maximum length of mail_recipient_address, mail_subject & mail_body is 1024 bytes

Parameter	Description
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

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,OxBBA6,0xBBA7,0xBBA8,0xBBA9,0xBBAA,0xBBAB,0xBBAC,0xBBAD,0xBBAE,0xBBAF,0xBBB0,0xBBB1,0xBBB2

Please refer to WLAN Error codes 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;
 }
```

6.3.1.3 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

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

Please refer to WLAN Error codes 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)
;
```

6.3.2 SNTP Client API

6.3.2.1 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.

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

Value	Description
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

Please refer WLAN Error codes for the description of 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);
```

6.3.2.2 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.

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

Value	Description
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

Please refer WLAN Error codes for the description of above error codes.

Example

```
status = rsi_sntp_client_gettime((uint16_t)length, (uint8_t
*)sntp_time);
```

6.3.2.3 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.

Parameter	Description
Length	This is the length of the buffer
sntp_time_date_rsp	This is the cuurent time and date response

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 , 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

Please refer WLAN Error codes for the description of above error codes.

Example

```
status = rsi_sntp_client_gettime_date((uint16_t)length, (uint8_t
*)sntp_date_time);
```

6.3.2.4 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.

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

Please refer WLAN Error codes for the description of above error codes.

Example

```
  status = rsi_sntp_client_server_info((uint16_t)length, (uint8_t
*)&sntp_server_info_rsp);
```

6.3.2.5 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

Please refer WLAN Error codes for the description of above error codes.

Example

```
  status = rsi_sntp_client_delete_async();
```

6.3.3 HTTP Client API

6.3.3.1 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);
```

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
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

Parameter	Description
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 code buffer: This is the buffer pointer length: This is the length of data more_data: 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 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

Please refer to WLAN Error codes 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);
```

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);
```

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
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
post_data	The is the HTTP data to be posted to server
post_data_length	This is the post data length

Parameter	Description
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

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

Please refer to WLAN Error codes 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);
```

6.3.3.2 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));
```

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
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

Parameter	Description
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 code buffer: This is the buffer pointer length: This is the length of data more_data: 1 – No more data 0 – more data present
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

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

Please refer to WLAN Error codes 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);
```

6.3.3.3 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);
```

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 code 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

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

Please refer to WLAN Error codes 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);
```

6.3.3.4 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.

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

Example

```
//! Create HTTP client for PUT method

status = rsi_http_client_put_create();
```

6.3.3.5 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
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

Parameter	Description
`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.

Please refer to WLAN Error codes 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);
```

6.3.3.6 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.

Please refer to WLAN Error codes 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;
```

6.3.3.7 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.

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

Example

```
//! Delete HTTP Client

status = rsi_http_client_put_delete();
```

6.3.3.8 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

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

Example

```
http_client_abort();
```

6.3.4 rsi_http_credentials

Prototype

```
int32_t rsi_http_credentials(int8_t *username , int8_t *password)
```

Description
This API creates a http server credentials request.

Parameter

Parameter	Description
username	This is the user given usename
password	This is the user given password

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

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

6.3.5 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;
  }
```

6.3.6 POP3 client API

6.3.6.1 rsi_pop3_session_create_async

Prototype

```
int32_t rsi_pop3_session_create_async(uint8_t flags,
uint8_t *server_ip_address,
uint16_t server_port_number,
uint8_t auth_type,
uint8_t *username,
uint8_t *password,
void(*rsi_pop3_response_handler)(uint16_t status,
uint8_t type,
uint8_t *buffer));
```

Description
This API creates a POP3 client session.

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_address	POP3 server IP address IPv4 address – 4 Bytes hexa-decimal, IPv6 address – 16 Bytes hexa-decimal
server_port_number	POP3 server TCP port **Note:** SMTP server port is configurable on non standard port also
auth_type	This is the client authentication type (Resserved)
client_domain	This is the domain name of the client Should be NULL terminated string
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
rsi_pop3_response_handler	This is the callback when asynchronous response comes for the session create. The parameters involved are : status,type and buffer status : This is the status code type : This is the sub command type buffer : This is the buffer pointer

If status in callback is nonzero, then sub command type comes 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** 0x0021,0x0015,0xBB87,0xff74

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

Example

```
 //! Create pop3 client session

 status = rsi_pop3_session_create_async((uint8_t )FLAGS, (uint8_t
*)&server_ip, (uint16_t)POP3_PORT, (uint8_t)POP3_AUTH_TYPE,(uint8_t
*)POP3_USERNAME, (uint8_t *)POP3_PASSWORD,
rsi_pop3_client_mail_response_handler);
 if(status != RSI_SUCCESS)
 {
 return status;
 }
```

6.3.6.2 rsi_pop3_get_mail_stats

Prototype

```
int32_t rsi_pop3_get_mail_stats (void)
```

Description
This API gets the mail stats.

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

Parameters
No Parameters

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** 0x0021,0xFF74,0xBB87

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

```
typedef struct rsi_pop3_client_resp_s
{
  uint8_t  command_type;
  uint8_t  mail_count[2];
  uint8_t  size[4];
} rsi_pop3_client_resp_t;
```

Example

```
//! Get mail stats from POP3 server
status = rsi_pop3_get_mail_stats();
if(status != RSI_SUCCESS)
{
return status;
}
```

6.3.6.3 rsi_pop3_get_mail_list

Prototype

```
int32_t rsi_pop3_get_mail_list(uint16_t mail_index)
```

Description
This API gets the size of the mail for the passed mail index.

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

Parameters

Parameter	Description
mail_index	mail index to get the size of the mail

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** 0x0021,0xBB87,0xFF74,0xBBFF

Example

```
//! Get mail list
status = rsi_pop3_get_mail_list(*(uint16_t *) pop3_resp.mail_count);
```

6.3.6.4 rsi_ pop3_retrive_mail

Prototype

```
int32_t rsi_pop3_retrive_mail(uint16_t mail_index)
```

Description
This API retrieves the mail content for the passed mail index

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

Parameters

Parameter	Description
mail_index	This is the mail index to get the mail content for the passed index

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** 0x0021,0xBB87,0xFF74,0xBBFF,0xBBC5

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

```
typedef struct rsi_pop3_mail_data_resp_s
{
  uint8_t command_type;
  uint8_t more;
  uint8_t length[2];
} rsi_pop3_mail_data_resp_t;
```

Example

```
 status = rsi_pop3_retrive_mail(*(uint16_t *) pop3_resp.mail_count);
```

6.3.6.5 rsi_pop3_mark_mail

Prototype

```
int32_t rsi_pop3_mark_mail(uint16_t mail_index)
```

Description
This API marks a mail as **deleted** for the passed mail index.

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

Parameters

Parameter	Description
mail_index	This is the mail index to mark the mail as deleted

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** 0x0021,0xFF74,0xBB87,0xBBFF

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

```
status = rsi_pop3_mark_mail(*(uint16_t *) pop3_resp.mail_count);
```

6.3.6.6 rsi_pop3_unmark_mail

Prototype

```
int32_t rsi_pop3_unmark_mail(void)
```

Description
This API unmarks all the marked (deleted) mails in the current session.

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

Parameters
No parameters

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** 0x0021,0xFF74,0xBB87

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

Example

rsi_pop3_unmark_mail();

6.3.6.7 rsi_ pop3_get_server_status

Prototype

```
int32_t rsi_pop3_get_server_status(void)
```

Description
This API gets the pop3 server status.

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

Parameters
No parameters

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** 0x0021,0xBB87,0xFF74

Example

```
//! Get POP3 server status
status = rsi_pop3_get_server_status();
```

6.3.6.8 rsi_pop3_session_delete

Prototype

```
int32_t rsi_pop3_session_delete(void)
```

Description
This API deletes pop3 client session.

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

Parameters
No parameters

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** 0x0021,0xFF74,0xBB87

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

```
//! Delete POP3 client session
status = rsi_pop3_session_delete();
```

6.3.7 FTP Client API

6.3.7.1 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

Parameter	Description
server_port	This is the port number of FTP server **Note:** FTP 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

Please refer to WLAN Error codes for a 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;
 }
```

6.3.7.2 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
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

Please refer to WLAN Error codes for a description of the above error codes.

Example

```
//! Disconnect from FTP server
retval = rsi_ftp_disconnect();
if(retval != RSI_SUCCESS)
{
return retval;
 }
```

6.3.7.3 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

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

Example

```
//! File Write
retval = rsi_ftp_file_write(FTP_FILE_TO_WRITE);
```

6.3.7.4 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

Parameter	Description
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

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

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);
```

6.3.7.5 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

Please refer to WLAN Error codes 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);
```

6.3.7.6 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

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

Example

rsi_ftp_file_delete(file_name);

6.3.7.7 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

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

Example

rsi_ftp_file_rename(old_file_name , new_file_name);

6.3.7.8 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

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

Please refer to WLAN Error codes for a description of the above error codes.

Example

rsi_ftp_directory_create(directory_name);

6.3.7.9 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

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

Example

rsi_ftp_directory_delete(directory_name);

6.3.7.10 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

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

Example

rsi_ftp_directory_set(directory_name);

6.3.7.11 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

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

Example
rsi_ftp_directory_list_async(/test/ftp/example.txt,call_back_handler_ptr);

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

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

Example

```
// set to Active Mode
rsi_ftp_mode_set(0);
```

6.3.8 MQTT Client API

6.3.8.1 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)
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 (Not supported), 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);
```

6.3.8.2 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

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

Example

```
status = rsi_mqtt_connect(rsi_mqtt_client,0,clientID,NULL,NULL);
```

6.3.8.3 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

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

Example

```
//! Disconnect to the MQTT broker
rsi_mqtt_disconnect(rsi_mqtt_client);
```

6.3.8.4 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
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

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

Example

```
rsi_mqtt_publish(rsi_mqtt_client,RSI_MQTT_TOPIC,&publish_msg);
```

6.3.8.5 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

Value	Description
Non Zero Value	**If return value is lesser than 0** -2: Invalid Parameter

Please refer to WLAN Error codes 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_rec
eived);
```

6.3.8.6 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

Please refer to WLAN Error codes for a description of the above error codes.

Example

```
//! UnSubscribe to the topic given
rsi_mqtt_unsubscribe(rsi_mqtt_client,RSI_MQTT_TOPIC);
```

6.3.8.7 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_config_ipaddress() API needs to be called before this API

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

Please refer to WLAN Error codes 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);
```

6.3.9 HTTP Server API

6.3.9.1 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/WiSeMCU 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

Please refer to WLAN Error codes 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.
```

6.3.9.2 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

Please refer to WLAN Error codes 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);
```

6.3.9.3 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.

Please refer to WLAN Error codes 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));
```

6.3.9.4 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

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

Example

```
rsi_webpage_erase(FILE_NAME);
```

6.3.9.5 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

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

6.3.10 DHCP User class (Option-77) API

6.3.10.1 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 ued to enable DHCP user class.

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

Parameter	Description
Status	This is the status of the DHCP user class 0 = success <0 = failure

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

Please refer WLAN Error codes for the description of 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);
```

6.3.11 Multicast API

6.3.11.1 rsi_multicast_join

Prototype

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

Description
This API joins to a multicast group.

> Device 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

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

Example

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

6.3.11.2 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.

Device 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

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

Example

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

6.3.12 MDNSD API

6.3.12.1 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.

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
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

Please refer to WLAN Error codes 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);
```

6.3.12.2 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.

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

Please refer to WLAN Error codes 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);
```

6.3.12.3 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

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

6.3.13 Socket configuration API

6.3.13.1 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

Parameter	Description
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.
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.

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

Example

```
status = rsi_socket_config();
```

6.3.14 Web socket API

6.3.14.1 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

Parameter	Description
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

Please refer to the WLAN Error Codes 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);
```

6.3.14.2 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));
```

6.3.14.3 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);
```

6.3.15 OTAF client API

6.3.15.1 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 retransmisstions 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

Please refer to WLAN Error codes for a 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;
 }
```

6.3.16 PUF API

6.3.16.1 rsi_puf_start_req

Prototype

```
int32_t rsi_puf_start_req(void)
```

Description
This API starts the PUF if valid activation code is available in the flash. If activation code on flash is valid, this operation returns success or else fails to start PUF. Start operation is must for any further operation with PUF.

Parameters
None

Pre Condition
rsi_wireless_init() API needs to be called before this API

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** 0xCC31, 0xCC32, 0xCC33, 0xCC35

Please refer to **PUF Error codes** for description of the above error codes.

Example

```
status = rsi_puf_start_req();
```

6.3.16.2 rsi_puf_set_key_req

Prototype

```
int32_t rsi_puf_set_key_req(uint8_t key_index,uint8_t
key_size,uint8_t *key_ptr,uint8_t *set_key_resp,uint16_t length)
```

Description
This API is used to request for a set of key operation for the given key, a Key code is generated by PUF which is returned if operation is success. This API will return failure if there is a failure in system or if this feature is blocked prior.

Parameters

Parameter	Description
Key_index	Key Index of Key to be generated (0-15) To increase the randomness
key_size	Key Size in bytes, 0: 128bit key 1: 256bit key
key_ptr	This is the pointer to key provided by application
set_key_resp	This is the keycode to the key provided. This is an output parameter.
Length	This is the length of the result buffer in bytes to hold keycode.

Pre Condition
rsi_puf_start_req() API needs to be called before this API

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**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** 0xCC2F, 0xCC32, 0xCC33, 0xCC34

Please refer to **PUF Error codes** for description of the above error codes.

Example

```
status = rsi_puf_set_key_req(1, PUF_KEY_SIZE_128 , RSI_AES_KEY,
keycode, KEY_CODE_SIZE_BYTES );
```

6.3.16.3 rsi_puf_set_key_disable_req

Prototype

```
int32_t rsi_puf_set_key_disable_req(void)
```

Description
This API blocks the set key for further operations on PUF.

Parameters
None

Pre Condition
rsi_wireless_init() API needs to be called before this API

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** 0xCC32, 0xCC33, 0xCC34

Please refer to **PUF Error codes** for description of the above error codes.

Example

```
status = rsi_puf_set_key_disable_req();
```

6.3.16.4 rsi_puf_get_key_req

Prototype

```
int32_t rsi_puf_get_key_req(uint8_t *keycode_ptr,uint8_t
*get_key_resp,uint16_t length)
```

Description
This API regenerates the key for the given key code using PUF. If operation is success, key is returned or else error is returned. This API will return failure if there is a failure in system or if this feature is blocked prior.

Parameters

Parameter	Description
keycode_ptr	This is the pointer to KeyCode
get_key_resp	This is the pointer to key, this is an output parameter
length	This is the length of the result buffer in bytes to hold key.

Pre Condition
rsi_puf_start_req() API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	**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** 0xCC2F, 0xCC32, 0xCC33, 0xCC34

Please refer to **PUF Error codes** for description of the above error codes.

Example

```
status = rsi_puf_get_key_req(keycode, get_key, KEY_CODE_SIZE_BYTES );
```

6.3.16.5 rsi_puf_get_key_disable_req

Prototype

```
int32_t rsi_puf_get_key_disable_req(void)
```

Description
This API blocks the further get key operations on PUF.

Parameters
None

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

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** 0xCC32, 0xCC33, 0xCC34

Please refer to **PUF Error codes** for description of the above error codes.

Example

```
status = rsi_puf_get_key_disable_req();
```

6.3.16.6 rsi_puf_load_key_req

Prototype

```
int32_t rsi_puf_load_key_req(uint8_t *keycode_ptr)
```

Description
This API regenerates the key for the given key code using PUF, and loads it into AES engine. If operation is success, key is loaded into AES or else error is returned. This API will return failure if there is a failure in system or if this feature is blocked prior.

Parameters

Parameter	Description
key_ptr	This is the pointer to keycode provided by an application

Pre Condition
rsi_puf_start_req() API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**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** 0xCC2F, 0xCC32, 0xCC33, 0xCC34

Please refer to **PUF Error codes** for description of above error codes.

Example

```
status = rsi_puf_load_key_req(keycode);
```

6.3.16.7 rsi_puf_intr_key_req

Prototype

```
int32_t rsi_puf_set_key_req(uint8_t key_index,uint8_t
key_size,uint8_t *intr_key_resp,uint16_t length)
```

Description
This API requests for intrinsic key operation for the given keysize , a Key code is generated by PUF which is returned if operation is success. This API will return failure if there is a failure in system or if this feature is blocked prior.

Parameters

Parameter	Description
Key_index	This is the key index of the Key to be generated (0-15) To increase the randomness
key_size	This is the key size in bytes 0: 128bit key 1: 256bit key
set_key_resp	This is the keycode to the key provided. This is an output parameter.
Length	This is the length of the result buffer in bytes to hold keycode.

Pre Condition
rsi_puf_start_req() API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**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** 0xCC2F, 0xCC32, 0xCC33, 0xCC34

Please refer to **PUF Error codes** for description of the above error codes.

Example

```
status = rsi_puf_intr_key_req(2, PUF_KEY_SIZE_128 , keycodeI,
KEY_CODE_SIZE_BYTES );
```

6.3.16.8 rsi_puf_aes_encrypt_req

Prototype

```
int32_t rsi_puf_aes_encrypt_req(uint8_t mode,uint8_t
key_source,uint16_t key_size,uint8_t *key_ptr,uint16_t
data_size,uint8_t *data_ptr,uint16_t iv_size,uint8_t *iv_ptr,uint8_t
*aes_encry_resp,uint16_t length)
```

Description
This API encrypts data inputted with Key provided or with key which is already loaded into AES by PUF. This API provides provision for encryption with AES engine into modes (ECB, CBC). 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
mode	This is AES encryption mode 0: ECB 1: CBC
Key_source	This is the encyption key source, 1: AES engine, provided by application as key_ptr 0: PUF
key_size	This is the Key Size in bytes, 0: 128bit key 1: 256bit key
key_ptr	This is the pointer to the key provided by application
data_size	This is the size of data in bytes
data_ptr	This is the pointer to data to be encrypted
iv_size	This is the intialization vector size(if CBC mode) 0: 128bit key
iv_ptr	This is the pointer to IV (if CBC mode)
Aes_encry_resp	This is the pointer to the encrypted data. This is an output parameter.
length	This is the length of the resultant buffer in bytes to hold encrypted data.

Pre Condition
rsi_puf_start_req() API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	**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** 0xCC32

Please refer to **PUF Error codes** for the description of the above error codes.

Example

```
status = rsi_puf_aes_encrypt_req(AES_ECB_MODE, AES_AS_KEY_SOURCE,
PUF_KEY_SIZE_128, RSI_AES_KEY , 16, RSI_AES_PLAIN_TXT , 0, NULL,
aes_encry_data , 16);
```

6.3.16.9 rsi_puf_aes_decrypt_req

Prototype

```
int32_t rsi_puf_aes_decrypt_req (uint8_t mode,uint8_t
key_source,uint16_t key_size,uint8_t *key_ptr,uint16_t
data_size,uint8_t *data_ptr,uint16_t iv_size,uint8_t *iv_ptr,uint8_t
*aes_decry_resp,uint16_t length)
```

Description
This API decrypts data inputted with Key provided or with key which is already loaded into AES by PUF. This API provides provision for decryption with AES engine in two modes (ECB, CBC). Parameters should be provided to API depending on mode of usage. API will return failure if there is any error in input.

Parameters

Parameter	Description
mode	This is the AES encryption mode 0: ECB 1: CBC
Key_source	This is the decryption key source, 1: AES engine, provided by application as key_ptr 0: PUF

Parameter	Description
key_size	This is the Key Size in bytes, 0: 128bit key 1: 256bit key
key_ptr	This is the pointer to key provided by application
data_size	This is the size of data in bytes
data_ptr	This is the pointer to data to be decrypted
iv_size	This is the initialization vector size (if CBC mode) 0 : 128bit key
iv_ptr	This is the pointer to IV (if CBC mode)
aes_decry_resp	This is the pointer to the decrypted data. This is an output parameter.
length	This is the length of the resultant buffer in bytes to hold decrypted data.

Pre Condition
rsi_puf_start_req() API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**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** 0xCC32

Please refer to **PUF Error codes** for description of the above error codes.

Example

```
status = rsi_puf_aes_decrypt_req(AES_ECB_MODE, AES_AS_KEY_SOURCE,
PUF_KEY_SIZE_128, RSI_AES_KEY , 16, aes_encry_data , 0, NULL,
aes_decry_data , 16 );
```

6.3.16.10 rsi_puf_aes_mac_req

Prototype

```
int32_t rsi_puf_aes_mac_req (uint8_t key_source, uint16_t
key_size,uint8_t *key_ptr,uint16_t data_size,uint8_t
*data_ptr,uint16_t iv_size,uint8_t *iv_ptr,uint8_t
*aes_mac_resp,uint16_t length)
```

Description
This API generates Message authentication check (MAC) for the data inputted with provided key as well as Initialization Vector (IV). Parameters should be provided to API depending on mode of usage. API will return failure if there is any error in input.

Parameters

Parameter	Description
Key_source	This is the key source to generate MAC, 1: provided by application as key_ptr 0: PUF
key_size	This is the Key Size in bytes, 0: 128bit key 1: 256bit key
key_ptr	This is the pointer to the key provided by application
data_size	This is the size of data in bytes
data_ptr	This is the pointer to Data
iv_size	This is the intialization of vector size (if CBC mode) 0: 128bit key
iv_ptr	This is the pointer to IV(if CBC mode)
aes_mac_resp	This is the pointer to MAC data, this is an output parameter.
length	This is the length of the result buffer in bytes to hold MAC data.

Pre Condition
rsi_puf_start_req() API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	**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** 0xcc32

Please refer to **PUF Error codes** for description of the above error codes.

Example

```
status = rsi_puf_aes_mac_req(AES_AS_KEY_SOURCE, PUF_KEY_SIZE_128,
RSI_AES_KEY , (16), RSI_AES_PLAIN_TXT, PUF_IV_SIZE_128,
RSI_AES_CBC_IV , aes_mac , 16);
```

6.4 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**" .

6.4.1 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 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

Define	Meaning
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 – Selelct to use FTP client feature TCP_IP_FEAT_IPV6 – Selelct 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

Define	Meaning
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

6.4.2 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.

6.4.3 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 with in time out
RSI_AP_KEEP_ALIVE_PERIOD	To configure keep alive period
RSI_MAX_STATIONS_SUPPORT	To configure maximum stations supported

6.4.4 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

6.4.5 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>

6.4.6 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

Define	Meaning
RSI_REJOIN_FIRST_TIME_RETRY	ENABLE or DISABLE retry for the first time join failure

6.4.7 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.
RSI_INSTANT_BG	Is it 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)

6.4.8 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

Define	Meaning
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

6.4.9 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'.

6.4.10 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.

6.4.11 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

6.4.12 Configure SSL parameters

Define	Meaning
RSI_SSL_VERSION	To select ssl version. Bydefault 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

Define	Meaning
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 **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)

6.4.13 Configure Power Save parameters

Define	Meaning
RSI_HAND_SHAKE_TYPE	To set handshake type of power mode MSG_BASED :
RSI_SELECT_LP_OR_ULP_MODE	RSI_LP_MODE :
RSI_DTIM_ALIGNED_TYPE	set DTIM aligment 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 1. 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.

6.4.14 Configure Transmit test mode parameters

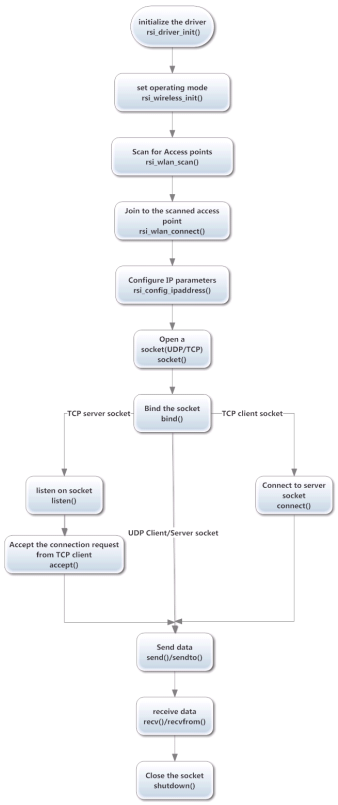
Define	Meaning
RSI_TX_TEST_RATE_FLAGS	Rate flags contain short GI, Greenfield and channel width values To enable short GI – set rate flags value as '1' To enable Greenfield – set rate flags value as '2'
RSI_TX_TEST_PER_CH_BW	Channel width should be set to zero to set 20MHz channel width.
RSI_TX_TEST_AGGR_ENABLE	This flag is for enabling or disabling aggregation support
RSI_TX_TEST_DELAY	Used to set the delay between the packets in burst mode. Delay should be given in micro seconds i.e. if the value is given as 'n' then a delay of 'n' micro seconds will be added for every transmitted packet in the burst mode. If this field is set to zero (0) then packets will be sent continuously without any delay

WLAN API call sequence examples

This section explains the sequence of API calls to configure the module in different modes.

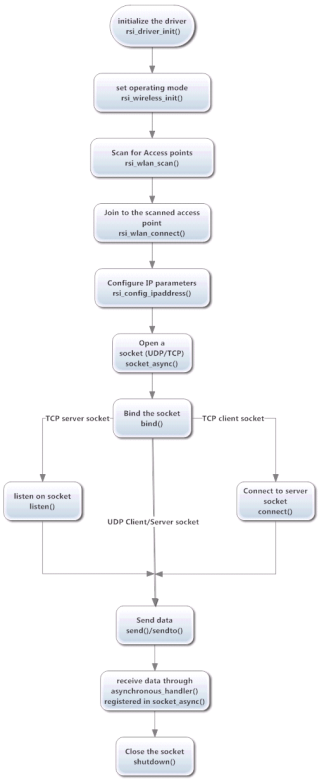
6.4.15 Station mode

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.



API flow to configure module in station mode

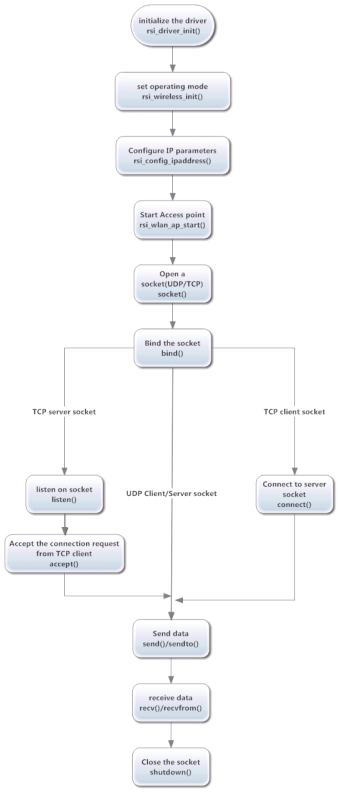
The following example illustrate the data transfer using rsi_socket_async() API



API flow to configure module in station mode

6.4.16 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.



API flow to configure module in Access point mode

rsi_wlan_buffer_config

Prototype
int wlan_buffer_config()

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

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

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

Return Values

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	**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

Please refer to **WLAN Error codes** for description of the above error codes.

Example

NA

6.4.17 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 lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002c, 0x000A

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

Example
rsi_wlan_receive_stats_start(channel);

6.4.18 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 **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

Please refer to WLAN Error codes for a description of the above error codes.

Example
rsi_transmit_test_stop();

6.4.19 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 lesser than 0** -2: Invalid Parameters -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002c

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

Example

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

6.4.20 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.

This API is relevant in opermode 8.

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/WiSeMCU 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 .

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
Pass any valid values for rate and length
2. Stop Burst mode
3. 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
24	137
36	141

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

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

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

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 lesser than 0** -4 : Buffer not available to serve the command **If return value is greater than 0** 0x000A, 0x0021, 0x0025, 0x002C

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

Example

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

6.4.21 rsi_transmit_test_stop

Prototype

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

Description
This API stops the transmit test.

> This API is relevant in opermode 8

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 lesser than 0** -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021, 0x0025, 0x002C

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

Example
rsi_transmit_test_stop();

6.4.22 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

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

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

Example

```
rsi_req_wireless_fwup();
```

6.4.23 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

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

Example

```
uint8_t recv_buffer[1000];
rsi_fwup_start(recv_buffer);
```

6.4.23.1 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

Please refer to WLAN Error codes 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);
```

6.4.24 rsi_cmd_m4_ta_secure_handshake

Prototype

```
int32_t rsi_cmd_m4_ta_secure_handshake(
uint8_t sub_cmd_type,
uint8_t input_len,
uint8_t *input_data,
uint8_t output_len,
uint8_t *output_data);
```

Description
This API handshake between M4 and TA.

Precondition
None**.**

Parameters

Parameter	Description
`sub_cmd_type`	This is the Subcommand of main command RSI_COMMON_REQ_TA_M4_COMMANDS Following Subcommand are supported: -> RSI_ENABLE_XTAL :This varaible is used for accurate alarm based sleep_wakeup of M4 and If the application uses this function power consumption will slightly increase.
Input_length	Input data length.
Input_data	**First byte of this input data is used to enable or disable the sub_cmd_type** (first argument)and remaining byte are used for input data.
Output_len	Ouput data length.
Output_data	Ouput data.

Return Values

Value	Description
0	Success.
Non Zero value	Failure.

Example

```
// see rsi_tx_on_periodic_wakeup.cfor a detailed example
 xtal_enable = 1;
 status = rsi_cmd_m4_ta_secure_handshake(RSI_ENABLE_XTAL,
1,&xtal_enable,0,NULL);
```

6.4.25 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 uint16_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: 0 - RSI_JOIN_FAIL_CB 1 - RSI_IP_FAIL_CB 2 - RSI_REMOTE_SOCKET_TERMINATE_CB 3 - RSI_IP_CHANGE_NOTIFY_CB 4 - RSI_STATIONS_CONNECT_NOTIFY_CB 5 - RSI_STATIONS_DISCONNECT_NOTIFY_CB 6 – RSI_WLAN_DATA_RECEIVE_NOTIFY_CB
void(*callback_handler_ptr)(uint16_t *status, const uint8_t *buffer, const uint16_t length)	This is the Call back handler **status:** status of the asynchronous response **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.

Call back id	Function Description
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_CONNECT_NOTIFY_CB	This callback is called when asynchronous station connect notification is received from the module in AP mode.
RSI_STATIONS_DISCONNECT_NOTIFY_CB	This callback is called when asynchronous station disconnect 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_WFD_DISCOVERY_NOTIFY_CB	This callback is called whenever a Wi-Fidirect device is discovered and its details is given to host.
RSI_WLAN_WFD_CONNECTION_REQUEST_NOTIFY_CB	This callback is called when a connection request comes from the discovered Wi-Fi direct device.

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

Structure members	Description
netmask	Holds Assigned subnet address
gateway	Holds Assigned gateway address

Response structure for PER stats

```
typedef struct rsi_per_stats_rsp_s{
uint8_t tx_pkts[2];
uint8_t reserved_1[2];
uint8_t tx_retries[2];
uint8_t crc_pass[2];
uint8_t crc_fail[2];
uint8_t cca_stk[2];
uint8_t cca_not_stk[2];
uint8_t pkt_abort[2];
uint8_t fls_rx_start[2];
uint8_t cca_idle[2];
uint8_t reserved_2[26];
uint8_t rx_retries[2];
uint8_t reserved_3[2];
uint8_t cal_rssi[2];
uint8_t reserved_4[4];
uint8_t xretries[2];
uint8_t max_cons_pkts_dropped[2];
uint8_t reserved_5[2];
uint8_t bss_broadcast_pkts[2];
uint8_t bss_multicast_pkts[2];
uint8_t bss_filter_matched_multicast_pkts[2];
}rsi_per_stats_rsp_t;
```

Structure members	Description
tx_pkts	Number of TX packets transmitted
reserved_1	Reserved
tx_retries	Number of TX retries happened
crc_pass	Number of RX packets that passed CRC
crc_fail	Number of RX packets that failed CRC
cca_stk	Number of times cca got stuck

Structure members	Description
cca_not_stk	Number of times cca didn't get stuck
pkt_abort	Number of times RX packet aborts happened
fls_rx_start	Number of false rx starts.If Valid wlan packet is recived and is dropped due to some reasons.
cca_idle	CCA idle time
reserved_2	Reserved
rx_retries	Number of RX retries happened
reserved_3	Reserved
cal_rssi	The calculated RSSI value of recently received RX packet
reserved_4	Reserved
xretries	Number of TX Packets dropped after maximum retries
max_cons_pkts_dropped	Number of consecutive packets dropped after maximum retries
reserved_5	Reserved
bss_broadcast_pkts	BSSID matched broadcast packets count.
bss_multicast_pkts	BSSID matched multicast packets count.
bss_filter_matched_multicast_pkts	BSSID and multicast filter matched packets count.

Response structure for WFD discovery and connection request:

```
typedef struct rsi_wfd_device_info_s
 {
 uint8_t device_state;
 uint8_t device_name[32];
 uint8_t mac_address[6];
 uint8_t device_type[2];
 }rsi_wfd_device_info_t;
 typedef struct rsi_rsp_wfd_device_info_s
 {
 uint8_t device_count;
 rsi_wfd_device_info_t wfd_dev_info[RSI_MAX_WFD_DEVICE_COUNT];
 } rsi_rsp_wfd_device_info_t;
 typedef struct rsi_rsp_p2p_connection_request_s
 {
 uint8_t device_name[32];
 }rsi_rsp_p2p_connection_request_t;
```

Examples

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

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
Pass any valid values for rate and length
2. Stop Burst mode

3. Start Continuous wave mode

6.4.26 Certificate valid indication

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

Prototype

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

Parameters

socket_id	Socket identifier

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

7 Crypto API

7.1 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

Value	Description
Non Zero	**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);
```

7.2 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	**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 );
```

7.3 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	**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);
```

7.4 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	**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);
```

7.5 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	**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);
```

7.6 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	**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);
```

7.7 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

Value	Description
Non Zero	**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);
```

7.8 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	**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);
```

7.9 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
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	**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);
```

8 BT-Classic and BT-LE Common Features

8.1 Power Save

8.1.1 Description

This feature configures the Power Save mode of the module and can be issued at any time after Opermode command. Power Save is disabled by default.
There are three different modes of Power Save.

1. Power Save mode 0
2. Power Save mode 2
3. Power Save mode 8

Note

1. Power Save modes 2 and 8 are not supported in USB / USB-CDC interface. Instead, they are supported in UART / SPI interfaces.
2. In SPI interface, when Power Save mode is enabled, after wakeup from sleep, host has to re-initialize SPI interface of the module.

8.2 Power Save Operations

The behavior of the module differs according to the Power Save mode it is configured with.

8.2.1 Power Save Mode 0

In this mode module is active and power save is disabled. It can be configured at any time while power save is enable with Power Save mode 2 or Power Save mode 8.

8.2.2 Power Save Mode 2

Once the module is configured to power save mode 2, it can be woken up either by the Host or periodically during its sleep-wakeup cycle. Power Save mode 2 can be either GPIO based or message based.

GPIO based mode:
In case of GPIO based mode, whenever host wants to send data to module, it gives wakeup indication by setting UULP GPIO #2. After wakeup, if the module is ready for data transfer, it sends wakeup indication to host by setting UULP GPIO #3. Host is required to wait until module gives wakeup indication before sending any data to the module.
After the completion of data transfer, host can give sleep permission to module by resetting UULP GPIO #2. After recognizing sleep permission from host, module gives confirmation to host by resetting UULP GPIO #3 and again gets back to its sleep-wakeup cycle.
Module can send received packets or responses to host at any instant of time. No handshake is required on Rx path.

Message based mode:
In case of message based power save, both radio and SOC of the module are in power save mode. Module wakes up periodically upon every deep sleep duration and gives wakeup message ("WKP") to host. Module can not be woken up asynchronously. Every time module intends to go to sleep it sends a sleep request message ("SLP") to the host and expects host to send the ack ("ACK") message. Host either sends ack ("ACK") or any other pending message. But once ack ("ACK") is sent, Host should not send any other message unless next wakeup message from module is received.
Module shall not go into complete power-save state if ack is not received from host for given sleep message.
Module can send received packets or responses to host at any instant of time. No handshake is required on Rx path.

Command Description	Binary Mode
"WKP"	0xDD

Command Description	Binary Mode
"SLP"	0xDE

Messages from module in Power Save mode 2
Table 1: Messages from module in Power Save mode 2

Command Description	Binary Mode
"ACK"	0xDE

Message from host in Power Save mode 2
Table 2: Message from host in Power Save mode 2

Usage in BT-Classic Mode:
In Classic, Power Save mode 2 can be used during Discoverable / Connectable / Connected sniff states.

- **Discoverable Mode State:** In this state, module is awake during Inquiry Scan window duration and sleeps till Inquiry Scan interval.

Default Inquiry scan window value is 11.25 msec, and Inquiry scan interval is 320 msec.

- **Connectable Mode State:** In this state, module is awake during Page Scan window duration and sleeps till Page Scan interval.

Default Page scan window value is 11.25 msec, and Page scan interval is 320 msec.

- **Connected Sniff State:** While the module is in connected state as a master or slave, once the module has configured with Power Save mode 2 with GPIO based or message based then the module will goes into power save mode in connected state. This will be work when the module and peer device supports sniff feature. And module should configure with sniff command after a successful connection, before configure with power save command.

Module will goes into power save after serving a sniff anchor point, and wakes up before starting a sniff anchor point.
Sniff connection anchor point may varies based on the remote device t_sniff value.

Usage in BT-LE Mode:
In LE, Power Save mode 2 can be used during Advertise / Scan / Connected states.

- **Advertise State:** In this state, module is awake during advertising event duration and sleeps till Advertising interval.
- **Scan State:** In this state, module is awake during Scanning window and sleeps till Scanning Interval. Default scan window is 50 msec, default scan interval is 160 msec.
- **Connected state:** In this state, module wakes up for every connection interval. Default connection interval is 200 msec which was configurable.

8.2.3 Power Save mode 8

In Power save mode 8, both RF and SOC of the module are in complete power save mode. This mode is significant only when module is in standby mode. Power mode 8 is GPIO based/message based. Power Save mode 8 can be either GPIO based or message based.

GPIO based mode:
In case of GPIO based, host can wakeup the module from power save by making UULP-GPIO #2 high.
Once the module wakes up, it continues to be in wakeup state until it gets power mode 8 commands from host.

Message based mode:
In case of message based, module goes to sleep immediately after issuing power save command and wakes up after 3sec. Upon wakeup, module sends a wakeup message "WKP" to the host and expects host to give ack "ACK"

before it goes into next sleep cycle. Host can either send ack or any other messages. But once ACK is sent, no other packet should be sent before receiving next wakeup message.

Command Description	Binary Mode
"WKP"	0xDD

Messages from module in Power Save mode 8
Table 3: Messages from module in Power Save mode 8

Command Description	Binary Mode
"ACK"	0xDE

Message from host in Power Save mode 8
Table 4: Message from host in Power Save mode 8

Note

- In BT Classic/LE, Power Save mode 8 can be used in Standby (idle) state.

Example: Suppose if Power Save is enabled in advertising state, to move to Scanning state, first Power Save disable command need to be issue before giving Scan command.

- For Page scan, Inquiry scan, sniff parameters related information, please verify Bluetooth protocol specification document.
- When the module is configured in a co-ex mode and WLAN is in INIT_DONE state, powersave mode 2 & 3 are valid after association in the WLAN. Where as in BT & BLE alone modes, it will enter into power save mode (2&3) in all states (except in standby state).
- Power save disable command has to be given before changing the state from standby to remaining states and vise-versa.

9 Bluetooth Classic APIs

This section explains the BT APIs required to initialize and configure the module in BT mode.

Note
A new BT API has to be called only after getting the response for the previous API.

9.1 Test Mode API

This section describes the APIs that are being used for testing purposes

9.1.1 rsi_bt_enable_device_under_testmode

9.1.1.1 Prototype

```
int32_t rsi_bt_enable_device_under_testmode(void);
```

9.1.1.2 Description

This API is used to keep the device in the LMP_TEST_MODE

9.1.1.3 Precondition

rsi_wireless_init() API need to be called before this API

9.1.1.4 Parameters

None

9.1.1.5 Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.1.1.6 Example

```
int32_t status = 0;
status = rsi_bt_enable_device_under_testmode();
```

9.1.2 rsi_bt_per_rx

9.1.2.1 Prototype

```
int32_t rsi_bt_per_rx(uint32_t *bt_rx_per);
```

9.1.2.2 Description

This API is used to configure the per receive parameters in the controller and start/stop the PER

9.1.2.3 Precondition

rsi_wireless_init() API need to be called before this API

9.1.2.4 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.

9.1.2.5 **rsi_bt_rx_per_params_t**

Structure 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;
```

Structure Desciption

This structure describes the format for accepting bt per rx parameters to be sent to local device.

Structure 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
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

Variables	Description
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
tx_chnl_in	This field corresponds to tx channel number to be used to recieve from
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	This field takes the scrambler seed value configured to 0
hopping_type	This field defines the frequency hopping type to be used 0 → NO_HOPPING 1 → FIXED_HOPPING 2 → RANDOM_HOPPING
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 1 → PLL_MODE1
rf_type	This field defines the selection of RF type (internal/external) 0 → BT_EXTERNAL_RF 1 → BT_INTERNAL_RF
rf_chain	This field corresponds to the selection of RF chain (HP/LP) to be used 2 → BT_HP_CHAIN 3 → BT_LP_CHAIN
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);
```

9.1.3 **rsi_bt_per_tx**

9.1.3.1 **Prototype**

```
int32_t rsi_bt_per_tx(uint32_t *bt_tx_per);
```

9.1.3.2 **Description**

This API is used to configure the per transmit parameters in the controller and start/stop the PER

9.1.3.3 **Precondition**

rsi_wireless_init() API need to be called before this API

9.1.3.4 **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.

9.1.3.5 **rsi_bt_tx_per_params_t**

Structure 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;
```

Structure Desciption

This structure describes the format for accepting bt per rx parameters to be sent to local device.

Structure 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 to be transmitted

Variables	Description
pkt_len	length of the packet to be transmitted
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
tx_chnl_in	This field corresponds to tx channel number to be used
link_type	This field corresponds to link type to be setup for trasnmitting per packets 0 → SCO_LINK 1 → ACL_LINK 2 → ESCO_LINK
scrambler_seed	This field takes the scrambler seed value configured to 0
hopping_type	This field defines the frequency hopping type to be used 0 → NO_HOPPING 1 → FIXED_HOPPING 2 → RANDOM_HOPPING
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 define the pll_mode type to be used 0 → PLL_MODE0 1 → PLL_MODE1
rf_type	This field defines the selection of RF type (internal/external) 0 → BT_EXTERNAL_RF 1 → BT_INTERNAL_RF
rf_chain	This field corresponds to the selection of RF chain (HP/LP) to be used 2 → BT_HP_CHAIN 3 → BT_LP_CHAIN

Variables	Description
payload_type	This field corresponds to the payload sequence of data to be transmitted 0 → SEQUENCE_0 1 → SEQUENCE_1 2 → SEQUENCE_2 3 → SEQUENCE_F0 4 → SEQUENCE_PRBS
tx_power	This field corresponds to the transmit power
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
no_of_packets	This field defines the number of packets to be transmitted, default to zero for coninuous 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);
```

9.1.4 **rsi_bt_per_stats**

9.1.4.1 **Prototype**

```
int32_t rsi_bt_per_stats(uint8_t cmd_type, rsi_bt_per_stats_t
*per_stats);
```

9.1.4.2 **Description**

This API requests the local device for BT PER operation.

9.1.4.3 **Precondition**

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

9.1.4.4 **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

9.1.4.5 **rsi_bt_per_stats_t**

Structure 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
```

Structure Description

This structure describes the format of PER statistics

Structure Variables

Variables	Description
crc_fail_cnt	Packet count of CRC fails
crc_pass_cnt	Packet count of CRC pass
tx_abort_cnt	Packet count of aborted Tx
rx_drop_cnt	Packet count of dropped Rx
rx_cca_idle_cnt	Packet count of CCA Idle
rx_start_idle_cnt	Packet count of Rx start
rx_abrt_cnt	Packet count of aborted Rx
tx_dones	Packet count of Tx Dones
rssi	RSSI of received 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);
```

9.2 GAP API

This section explains the BT GAP APIs.

9.2.1 rsi_bt_set_local_class_of_device

9.2.1.1 Prototype

```
int32_t rsi_bt_set_local_class_of_device(uint32_t class_of_device);
```

9.2.1.2 Description

This API requests the local COD name.

9.2.1.3 Precondition

rsi_wireless_init() API need to be called before this API

9.2.1.4 Parameters

Parameter	Description
class_of_device	This is the class of device

9.2.1.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.1.6 **Example**

```
uint32_t class_of_device = 284000;
int32_t status = 0;
status = rsi_bt_set_local_class_of_device(class_of_device);
```

9.2.2 **rsi_bt_get_local_class_of_device**

9.2.2.1 **Prototype**

```
int32_t rsi_bt_get_local_class_of_device(uint8_t *resp);
```

9.2.2.2 **Description**

This API requests the local COD name.

9.2.2.3 **Precondition**

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

9.2.2.4 **Parameters**

Parameter	Description
resp	This parameter is to hold the response of this API

9.2.2.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.2.6 **Example**

```
int32_t status = 0;
status = rsi_bt_get_local_class_of_device(&class_of_device);
```

9.2.3 **rsi_bt_start_discoverable**

9.2.3.1 **Prototype**

```
int32_t rsi_bt_start_discoverable(void);
```

9.2.3.2 **Description**

This API requests the local device to enter discovery mode.

9.2.3.3 **Precondition**

rsi_wireless_init API needs to be called before this API.

9.2.3.4 **Parameters**

None

9.2.3.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.3.6 **Example**

```
int32_t status = 0;
status = rsi_bt_start_discoverable();
```

9.2.4 rsi_bt_start_limited_discoverable

9.2.4.1 Prototype

```
int32_t rsi_bt_start_limited_discoverable(int32_t time_out_ms);
```

9.2.4.2 Description

This API requests the local device to enter limited discovery mode.

9.2.4.3 Precondition

rsi_wireless_init API needs to be called before this API.

9.2.4.4 Parameters

Parameter	Description
time_out_ms	Limited discovery mode time_out in ms.

9.2.4.5 Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.4.6 Example

```
int32_t status = 0;
status = rsi_bt_start_limited_discoverable(100);
```

9.2.5 rsi_bt_stop_discoverable

9.2.5.1 Prototype

```
int32_t rsi_bt_stop_discoverable(void);
```

9.2.5.2 **Description**

This API request the local device to exit the discovery mode.

9.2.5.3 **Parameters**

rsi_wireless_init API needs to be called before this API.

9.2.5.4 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.5.5 **Example**

```
int32_t status = 0;
status = rsi_bt_stop_discoverable();
```

9.2.6 **rsi_bt_get_discoverable_status**

9.2.6.1 **Prototype**

```
int32_t rsi_bt_get_discoverable_status(uint8_t *resp)
```

9.2.6.2 **Description**

This API is used to request the local device discovery mode status.

9.2.6.3 **Precondition**

rsi_bt_start_discoverable() API need to be called before this API.

9.2.6.4 **Parameters**

Parameter	Description
resp	This parameter is to hold the response of this API

9.2.6.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.6.6 **Example**

```
uint8_t device_state = 0;
int32_t status = 0;
status = rsi_bt_get_discoverable_status(&device_state);
```

9.2.7 **rsi_bt_set_connectable**

9.2.7.1 **Prototype**

```
int32_t rsi_bt_set_connectable(void);
```

9.2.7.2 **Description**

This API requests the local device to set the connectivity mode.

9.2.7.3 **Precondition**

rsi_wireless_init API needs to be called before this API.

9.2.7.4 **Parameters**

None

9.2.7.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.7.6 **Example**

```
int32_t status = 0;
status = rsi_bt_set_connectable();
```

9.2.8 **rsi_bt_set_non_connectable**

9.2.8.1 **Prototype**

```
int32_t rsi_bt_set_non_connectable(void);
```

9.2.8.2 **Description**

This API sets the BT Module in non-connectable mode.

9.2.8.3 **Precondition**

rsi_wireless_init API need to be called before this API

9.2.8.4 **Parameters**

None

9.2.8.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_set_non_connectable();
```

9.2.9 rsi_bt_get_connectable_status

9.2.9.1 Prototype

```
int32_t rsi_bt_get_connectable_status(uint8_t *resp);
```

9.2.9.2 Description

This API gets the BT Module connectivity status.

9.2.9.3 Precondition

rsi_bt_set_connectable() API needs to be called before this API

9.2.9.4 Parameters

Parameter	Description
resp	This parameter is to hold the response of this API

9.2.9.5 Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.9.6 Example

```
uint8_t device_state = 0;
int32_t status = 0;
status = rsi_bt_get_connectable_status(&device_state);
```

9.2.10 rsi_bt_set_afh_host_channel_classification

9.2.10.1 Prototype

```
int32_t rsi_bt_set_afh_host_channel_classification (uint8_t enable,
uint8_t *channel_map)
```

9.2.10.2 Description

This API is used to set the number of channels to be used

9.2.10.3 Precondition

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

9.2.10.4 Parameters

Parameter	Description
enable	This parameter is used to enable or disable afh host channel classification. This parameters supports the following values. 0 - Disable 1 - Enable
channel map	Array of channel mapping values

9.2.10.5 Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.10.6 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);
```

9.2.11 rsi_bt_get_afh_host_channel_classification

9.2.11.1 Prototype

```
int32_t rsi_bt_get_afh_host_channel_classification (uint8_t *mode)
```

9.2.11.2 Description

This API is used to get the afh channel assessment mode

9.2.11.3 Precondition

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

9.2.11.4 Parameters

Parameter	Description
mode	This parameter is to hold the response of this API

9.2.11.5 Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.11.6 Example

```
int32_t status = 0;
uint8_t mode = 0;
status = rsi_bt_get_afh_host_channel_classification(&mode);
```

9.2.12 **rsi_bt_remote_name_request_async**

9.2.12.1 **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);
```

9.2.12.2 **Description**

This API is used to know the remote device name.

9.2.12.3 **Precondition**

rsi_wireless_init() API need to be called before this API

9.2.12.4 **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

9.2.12.5 rsi_bt_event_remote_device_name_t

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_DEV_NAME_LEN];
} rsi_bt_event_remote_device_name_t;
```

Structure Description

This structure describes the format of the remote device name event structure.

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)
```

9.2.13 **rsi_bt_remote_name_request_cancel**

9.2.13.1 **Prototype**

```
int32_t rsi_bt_remote_name_request_cancel(int8_t *remote_dev_addr);
```

9.2.13.2 **Description**

This API is used cancel the remote device name request.

9.2.13.3 **Precondition**

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

9.2.13.4 **Parameters**

Parameter	Description
remote_dev_addr	This is the remote device address

9.2.13.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.13.6 **Example**

```
int32_t status = 0;
status = rsi_bt_remote_name_request_cancel(remote_dev_addr);
```

9.2.14 **rsi_bt_inquiry**

9.2.14.1 **Prototype**

```
int32_t rsi_bt_inquiry(uint8_t inquiry_type, uint32_t
inquiry_duration, uint8_t max_devices);
```

9.2.14.2 **Description**

This API starts the inquiry.

9.2.14.3 **Precondition**

rsi_wireless_init() API need to be called before this API

9.2.14.4 **Parameters**

Parameter	Description
inquiry_type	This is the Inquiry type.

Parameter	Description
inquiry_duration	This is the duration of inquiry
max_devices	This is the maximum number of devices allowed to inquiry

9.2.14.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.14.6 **Example**

```
int32_t status = 0;
status = rsi_bt_inquiry(0, 500, 0x02);
```

9.2.15 **rsi_bt_cancel_inquiry**

9.2.15.1 **Prototype**

```
int32_t rsi_bt_cancel_inquiry(void);
```

9.2.15.2 **Description**

This API cancels the inquiry.

9.2.15.3 **Precondition**

rsi_wireless_init() API need to be called before this API

9.2.15.4 **Parameters**

None

9.2.15.5 **Return Values**

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_cancel_inquiry();
```

9.2.16 rsi_bt_set_eir_data

9.2.16.1 Prototype

```
int32_t rsi_bt_set_eir_data(int8_t *eir_data, uint16_t data_len);
```

9.2.16.2 Description

This API sets the extended Inquiry Response data.

9.2.16.3 Precondition

rsi_wireless_init() API needs to be called before this API

9.2.16.4 Parameters

Parameter	Description
eir_data	Pointer to the EIR data buffer which is an array that can stores data upto 200 Bytes.
data_len	This is the length of the eir data

Note:
Currently EIR data supports upto 200 bytes.

9.2.16.5 Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.16.6 **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);
```

9.2.17 **rsi_bt_connect**

9.2.17.1 **Prototype**

```
int32_t rsi_bt_connect(int8_t *remote_dev_addr);
```

9.2.17.2 **Description**

This API initiates the connection request

9.2.17.3 **Precondition**

rsi_wireless_init() API need to be called before this API

9.2.17.4 **Parameters**

Parameter	Description
remote_dev_addr	Remote device address

9.2.17.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.17.6 **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);
```

9.2.18 **rsi_bt_cancel_connect**

9.2.18.1 **Prototype**

```
int32_t rsi_bt_cancel_connect(int8_t *remote_dev_address);
```

9.2.18.2 **Description**

This API cancels the connection request.

9.2.18.3 **Precondition**

rsi_wireless_init() API need to be called before this API

9.2.18.4 **Parameters**

Parameter	Description
remote_dev_address	This is the remote device address

9.2.18.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.18.6 **Example**

```
int32_t status = 0;
status = rsi_bt_cancel_connect(remote_dev_addr);
```

9.2.19 **rsi_bt_disconnect**

9.2.19.1 **Prototype**

```
int32_t rsi_bt_disconnect(int8_t *remote_dev_address);
```

9.2.19.2 **Description**

This API is used to disconnect the physical connection.

9.2.19.3 **Precondition**

rsi_wireless_init() API need to be called before this API

9.2.19.4 **Parameters**

Parameter	Description
remote_dev_address	This is the remote device address

9.2.19.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.19.6 **Example**

```
int32_t status = 0;
status = rsi_bt_disconnect(remote_dev_addr);
```

9.2.20 **rsi_bt_set_ssp_mode**

9.2.20.1 **Prototype**

```
int32_t rsi_bt_set_ssp_mode (uint8_t pair_mode, uint8_t
IOcapability);
```

9.2.20.2 **Description**

This API enables / disables Simple Secure Profile (SSP) mode.

9.2.20.3 **Precondition**

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

9.2.20.4 **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

9.2.20.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.20.6 **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);
```

9.2.21 **rsi_bt_accept_ssp_confirm**

9.2.21.1 **Prototype**

```
int32_t rsi_bt_accept_ssp_confirm(int8_t *remote_dev_address);
```

9.2.21.2 **Description**

This API gives the confirmation for the passkey sent by local BT device at the time of pairing.

9.2.21.3 **Precondition**

rsi_bt_set_ssp_mode() API need to be called before this API

9.2.21.4 **Parameters**

Parameter	Description
remote_dev_address	This is the remote device address

9.2.21.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.21.6 **Example**

```
int32_t status = 0;
status = rsi_bt_accept_ssp_confirm(str_conn_bd_addr);
```

9.2.22 **rsi_bt_reject_ssp_confirm**

9.2.22.1 **Prototype**

```
int32_t rsi_bt_reject_ssp_confirm(int8_t *remote_dev_address)
```

9.2.22.2 **Description**

This API rejects the confirmation for the passkey sent by the local BT device at the time of pairing.

9.2.22.3 **Precondition**

rsi_bt_set_ssp_mode() API need to be called before this API

9.2.22.4 **Parameters**

Parameter	Description
remote_dev_address	This is the remote device address

9.2.22.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.22.6 **Example**

```
int32_t status = 0;
status = rsi_bt_reject_ssp_confirm(remote_dev_addr);
```

9.2.23 **rsi_bt_passkey**

9.2.23.1 **Prototype**

```
int32_t rsi_bt_passkey(int8_t *remote_dev_addr, uint32_t passkey,
uint8_t reply_type);
```

9.2.23.2 **Description**

This API sends the passkey or rejects the incoming pass key request.

9.2.23.3 **Precondition**

rsi_bt_spp_connect() and rsi_bt_on_passkey_request() APIs need to be called before this API.

9.2.23.4 **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

9.2.23.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.23.6 **Example**

```
int32_t status = 0;
status = rsi_bt_passkey(remote_dev_addr, 123456, 1);
```

9.2.24 **rsi_bt_pincode_request_reply**

9.2.24.1 **Prototype**

```
int32_t rsi_bt_pincode_request_reply(int8_t *remote_dev_addr, uint8_t
*pin_code, uint8_t reply_type);
```

9.2.24.2 **Description**

This API sends the pincode or rejects the incoming pincode request.

9.2.24.3 **Precondition**

rsi_bt_ssp_mode() and rsi_bt_app_on_pincode_req() APIs need to be called before this API.

9.2.24.4 **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

9.2.24.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.24.6 **Example**

```
#define PIN_CODE "4321"
int32_t status = 0;
status = rsi_bt_pincode_request_reply(str_conn_bd_addr, PIN_CODE, 1);
```

9.2.25 rsi_bt_linkkey_request_reply

9.2.25.1 Prototype

```
int32_t rsi_bt_linkkey_request_reply (int8_t *remote_dev_addr,
uint8_t *linkkey, uint8_t reply_type);
```

9.2.25.2 Description

This API sends either positive (along with the link key) or negative reply to the incoming link key request.

9.2.25.3 Precondition

rsi_wireless_init() API need to be called before this API

9.2.25.4 Parameters

Parameter	Description
remote_dev_addr	Remote device address
linkkey	Linkkey input
reply_type	Positive or negative reply

9.2.25.5 Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.25.6 Example

```
int32_t status = 0;
//! sending the linkkey request negative reply
status = rsi_bt_linkkey_request_reply (str_conn_bd_addr, NULL, 0);
```

9.2.26 **rsi_bt_get_local_device_role**

9.2.26.1 **Prototype**

```
int32_t rsi_bt_get_local_device_role(int8_t *remote_dev_addr, uint8_t
*resp);
```

9.2.26.2 **Description**

This API requests the role of a local device.

9.2.26.3 **Precondition**

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

9.2.26.4 **Parameters**

Parameter	Description
remote_dev_addr	This is the remote device address
resp	This parameter is to hold the response of this API

9.2.26.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.26.6 **Example**

```
int32_t status = 0;
status = rsi_bt_get_local_device_role(rsi_app_resp_get_bond_dev_addr,
&device_state);
```

9.2.27 rsi_bt_set_local_device_role

9.2.27.1 Prototype

```
int32_t rsi_bt_set_local_device_role(int8_t *remote_dev_addr, uint8_t
set_role, uint8_t *resp);
```

9.2.27.2 Description

This API sets the role of a local device.

9.2.27.3 Precondition

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

9.2.27.4 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

9.2.27.5 Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.27.6 Example

```
int32_t status = 0;
status = rsi_bt_set_local_device_role(rsi_app_resp_get_bond_dev_addr,
0, &device_state);
```

9.2.28 **rsi_bt_sniff_mode**

9.2.28.1 **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);
```

9.2.28.2 **Description**

This API requests the local device to enter into sniff mode.

9.2.28.3 **Precondition**

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

9.2.28.4 **Parameters**

Parameter	Description
remote_dev_addr	This is the remote device address
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

9.2.28.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.28.6 **Example**

```
//! Sniff Parameters
#define SNIFF_MAX_INTERVAL    0xA0
#define SNIFF_MIN_INTERVAL    0XA0
#define SNIFF_ATTEMPT         0X04
#define SNIFF_TIME_OUT        0X02
int32_t status = 0;
/* here we call the sniff_mode command*/
status = rsi_bt_sniff_mode(str_conn_bd_addr, SNIFF_MAX_INTERVAL,
SNIFF_MIN_INTERVAL, SNIFF_ATTEMPT, SNIFF_TIME_OUT);
```

9.2.29 **rsi_bt_sniff_exit_mode**

9.2.29.1 **Prototype**

```
int32_t rsi_bt_sniff_exit_mode(uint8_t remote_dev_addr);
```

9.2.29.2 **Description**

This API is used to request the local device to exit from sniff/sniff subrating mode.

9.2.29.3 **Precondition**

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

9.2.29.4 **Parameters**

Parameter	Description
remote_dev_addr	This is the remote device address

9.2.29.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.29.6 **Example**

```
int32_t status = 0
status = rsi_bt_sniff_exit_mode(str_conn_bd_addr);
```

9.2.30 **rsi_bt_sniff_subrating_mode**

9.2.30.1 **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);
```

9.2.30.2 **Description**

This API requests the device entered into the sniff sub rating mode.

9.2.30.3 **Precondition**

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

9.2.30.4 **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

9.2.30.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0
status = rsi_bt_sniff_subrating_mode(remote_dev_addr, 192, 1000,
1000)
```

9.2.31 **rsi_bt_add_device_id**

9.2.31.1 **Prototype**

```
int32_t 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);
```

9.2.31.2 **Description**

This API adds device Identification in SDP protocol.

9.2.31.3 **Precondition**

rsi_wireless_init() API need to be called before this API

9.2.31.4 **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.

9.2.31.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.2.31.6 **Example**

```
#define RSI_DID_SPEC_ID                                  0x0200
#define RSI_DID_VENDOR_ID                              0x0201
#define RSI_DID_PRODUCT_ID                              0x0202
#define RSI_DID_VERSION                                  0x0203
#define RSI_DID_PRIMARY_RECORD                  0x0001
#define RSI_DID_VENDOR_ID_SOURCE          0x0002
int32_t status = 0
status = rsi_bt_add_device_id(RSI_DID_SPEC_ID,RSI_DID_VENDOR_ID,
RSI_DID_PRODUCT_ID,

RSI_DID_VERSION, RSI_DID_PRIMARY_RECORD, RSI_DID_VENDOR_ID_SOURCE);
```

9.3 SPP API

9.3.1 **rsi_bt_spp_init**

9.3.1.1 **Prototype**

```
int32_t rsi_bt_spp_init(void);
```

9.3.1.2 **Description**

This API sets the SPP profile mode.

9.3.1.3 **Precondition**

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

Parameters

None

9.3.1.4 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.3.1.5 **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();
```

9.3.2 **rsi_bt_spp_connect**

9.3.2.1 **Prototype**

```
int32_t rsi_bt_spp_connect(uint8_t *remote_dev_addr);
```

9.3.2.2 **Description**

This API initiates the SPP profile level connection.

9.3.2.3 **Precondition**

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

9.3.2.4 **Parameters**

Parameter	Description
remote_dev_addr	Remote device address

9.3.2.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.3.2.6 **Example**

```
#define REMOTE_BD_ADDR        "00:1A:57:45:32:BA"
int32_t status = 0;
status = rsi_bt_spp_connect(REMOTE_BD_ADDR);
```

9.3.3 **rsi_bt_spp_disconnect**

9.3.3.1 **Prototype**

```
int32_t rsi_bt_spp_disconnect(uint8_t *remote_dev_addr);
```

9.3.3.2 **Description**

This API initiates the SPP service level disconnection.

9.3.3.3 **Precondition**

rsi_bt_spp_connect() API need to be called before this API

9.3.3.4 **Parameters**

Parameter	Description
remote_dev_addr	This is the remote device address

9.3.3.5 **Return Values**

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	Failure

9.3.3.6 **Example**

```
int32_t status = 0;
status = rsi_bt_spp_disconnect(&remote_dev_addr);
```

9.3.4 **rsi_bt_spp_transfer**

9.3.4.1 **Prototype**

```
int32_t rsi_bt_spp_transfer(uint8_t *remote_dev_addr, uint8_t *data,
uint16_t length);
```

9.3.4.2 **Description**

This API transfers the data through SPP profile.

9.3.4.3 **Precondition**

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

9.3.4.4 **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

9.3.4.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.3.4.6 **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);
```

9.4 A2DP API

9.4.1 **rsi_bt_a2dp_init**

9.4.1.1 **Prototype**

```
int32_t rsi_bt_a2dp_init(void);
```

9.4.1.2 **Description**

This API sets the A2DP profile mode.

9.4.1.3 **Precondition**

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

9.4.1.4 **Parameters**

None

9.4.1.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.4.1.6 **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 A2DP profile
status = rsi_bt_a2dp_init();
```

9.4.2 **rsi_bt_a2dp_connect**

9.4.2.1 **Prototype**

```
int32_t rsi_bt_a2dp_connect(uint8_t *remote_dev_addr);
```

9.4.2.2 **Description**

This API initiates the A2DP profile level connection.

9.4.2.3 **Precondition**

rsi_bt_a2dp_init() API needs to be called before this API.

9.4.2.4 **Parameters**

Parameter	Description
remote_dev_addr	Remote device address

9.4.2.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.4.2.6 **Example**

```
#define REMOTE_BD_ADDR        "00:1A:57:45:32:BA"
int32_t status = 0;
status = rsi_bt_a2dp_connect(REMOTE_BD_ADDR);
```

9.4.3 **rsi_bt_a2dp_disconnect**

9.4.3.1 **Prototype**

```
int32_t rsi_bt_a2dp_disconnect(uint8_t *remote_dev_addr);
```

9.4.3.2 **Description**

This API initiates the A2DP service level disconnection.

9.4.3.3 **Precondition**

rsi_bt_a2dp_connect() API need to be called before this API

9.4.3.4 **Parameters**

Parameter	Description
remote_dev_addr	This is the remote device address

9.4.3.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.4.3.6 **Example**

```
int32_t status = 0;
status = rsi_bt_a2dp_disconnect(&remote_dev_addr);
```

9.4.4 **rsi_bt_a2dp_send_pcm_mp3_data**

9.4.4.1 **Prototype**

```
int32_t rsi_bt_a2dp_send_pcm_mp3_data(uint8_t *remote_dev_addr,
uint8_t *pcm_mp3_data, uint16_t pcm_mp3_data_len, uint8_t
audio_type);
```

9.4.4.2 **Description**

This API will send the pcm/mp3 data to BT stack from host, after A2DP configuration.

9.4.4.3 **Precondition**

rsi_bt_a2dp_connect() API need to be called before this API

9.4.4.4 **Parameters**

Parameter	Description
remote_dev_addr	Remote device address
pcm_mp3_data	Buffer pointer for audio data
pcm_mp3_data_len	length of the pcm/mp3 data
audio_type	Audio data type whether it is PCM or MP3 data 1 → PCM Audio 3 → MP3 Audio

9.4.4.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.4.4.6 **Example**

```
#define PCM_AUDIO                          1
int32_t status = 0;
uint8_t pcm_data[512];
pcm_data_len = sizeof(pcm_data);
status = rsi_bt_a2dp_send_pcm_mp3_data (str_conn_bd_addr, pcm_data,
pcm_data_len, PCM_AUDIO);
```

9.4.5 **rsi_bt_a2dp_send_sbc_aac_data**

9.4.5.1 **Prototype**

```
int32_t rsi_bt_a2dp_send_sbc_aac_data(uint8_t *remote_dev_addr,
uint8_t *sbc_aac_data, uint16_t sbc_aac_data_len, uint8_t
audio_type);
```

9.4.5.2 **Description**

This API will send the sbc/aac data to BT stack from host

9.4.5.3 **Precondition**

rsi_bt_a2dp_connect() API need to be called before this API

9.4.5.4 **Parameters**

Parameter	Description
remote_dev_addr	Remote device address
sbc_aac_data	Buffer pointer for audio data
sbc_aac_data_len	length of the sbc/aac data
audio_type	reserved

9.4.5.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.4.5.6 **Example**

```
int32_t status = 0;
uint8_t sbc_data[512];
sbc_data_len = sizeof(pcm_data);
status = rsi_bt_a2dp_send_sbc_aac_data (str_conn_bd_addr, sbc_data,
sbc_data_len, 0);
```

9.4.6 **rsi_bt_a2dp_start**

9.4.6.1 **Prototype**

```
int32_t rsi_bt_a2dp_start (uint8_t *remote_dev_addr);
```

9.4.6.2 **Description**

This API requests BT stack to start A2DP Audio stream.

9.4.6.3 **Precondition**

rsi_bt_a2dp_connect() API need to be called before this API

9.4.6.4 **Parameters**

Parameter	Description
remote_dev_addr	Remote device address

9.4.6.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.4.6.6 **Example**

```
int32_t status = 0;
status = rsi_bt_a2dp_start(str_conn_bd_addr);
```

9.4.7 **rsi_bt_a2dp_suspend**

9.4.7.1 **Prototype**

```
int32_t rsi_bt_a2dp_suspend (uint8_t *remote_dev_addr);
```

9.4.7.2 **Description**

This API requests BT stack to suspend A2DP Audio stream.

9.4.7.3 **Precondition**

rsi_bt_a2dp_start() API need to be called before this API

9.4.7.4 **Parameters**

Parameter	Description
remote_dev_addr	Remote device address

9.4.7.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

9.4.7.6 **Example**

```
int32_t status = 0;
status = rsi_bt_a2dp_suspend(str_conn_bd_addr);
```

9.5 AVRCP API

9.5.1 rsi_bt_avrcp_init

Prototype

```
int32_t rsi_bt_avrcp_init(void)
```

Description
This API sets the AVRCP 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 AVRCP profile
 status = rsi_bt_avrcp_init();
```

9.5.2 rsi_bt_avrcp_conn

Prototype

```
int32_t rsi_bt_avrcp_conn(uint8_t *remote_dev_addr)
```

Description
This API initiates the AVRCP profile level connection.

Precondition
rsi_bt_avrcp_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

```
int32_t status = 0;
status = rsi_bt_avrcp_connect(REMOTE_BD_ADDR);
```

9.5.3 rsi_bt_avrcp_disconn

Prototype

```
int32_t rsi_bt_avrcp_disconn(uint8_t *remote_dev_addr)
```

Description
This API initiates the AVRCP service level disconnection.

Precondition
rsi_bt_avrcp_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_avrcp_disconn(REMOTE_BD_ADDR);
```

9.5.4 rsi_bt_avrcp_play

Prototype

```
int32_t rsi_bt_avrcp_play(uint8_t *remote_dev_addr)
```

Description
This API requests avrcp to play the audio song.

Precondition
This api is used after AVRCP profile 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;
status = rsi_bt_avrcp_play(REMOTE_BD_ADDR);
```

9.5.5 rsi_bt_avrcp_pause

Prototype

```
int32_t rsi_bt_avrcp_pause(uint8_t *remote_dev_addr)
```

Description
This API requests avrcp to pause the audio song.

Precondition
This api is used after AVRCP profile 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;
status = rsi_bt_avrcp_pause(REMOTE_BD_ADDR);
```

9.5.6 rsi_bt_avrcp_stop

Prototype

```
int32_t rsi_bt_avrcp_stop(uint8_t *remote_dev_addr)
```

Description
This API requests avrcp to stop the audio song.

Precondition
This api is used after AVRCP profile 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;
status = rsi_bt_avrcp_stop(REMOTE_BD_ADDR);
```

9.5.7 rsi_bt_avrcp_next

Prototype

```
int32_t rsi_bt_avrcp_next(uint8_t *remote_dev_addr)
```

Description
This API requests avrcp to play the next audio song.

Precondition
This api is used after AVRCP profile 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;
status = rsi_bt_avrcp_next(REMOTE_BD_ADDR);
```

9.5.8 rsi_bt_avrcp_previous

Prototype

```
int32_t rsi_bt_avrcp_previous(uint8_t *remote_dev_addr)
```

Description
This API requests avrcp to play the previous audio song.

Precondition
This api is used after AVRCP profile 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;
status = rsi_bt_avrcp_previous(REMOTE_BD_ADDR);
```

9.5.9 rsi_bt_avrcp_vol_up

Prototype

```
int32_t rsi_bt_avrcp_vol_up(uint8_t *remote_dev_addr)
```

Description
This API requests avrcp to volume up.

Precondition
This api is used after AVRCP profile 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;
status = rsi_bt_avrcp_vol_up(REMOTE_BD_ADDR);
```

9.5.10 rsi_bt_avrcp_vol_down

Prototype

```
int32_t rsi_bt_avrcp_vol_down(uint8_t *remote_dev_addr)
```

Description
This API requests avrcp to volume down.

Precondition
This api is used after AVRCP profile 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;
status = rsi_bt_avrcp_vol_down(REMOTE_BD_ADDR);
```

9.5.11 rsi_bt_avrcp_get_remote_version

Prototype

```
int32_t rsi_bt_avrcp_get_remote_version (uint8_t *remote_dev_addr,
rsi_bt_rsp_avrcp_remote_version_t *version)
```

Description
This API requests avrcp to get the AVRCP profile version from remote device.

Precondition
This api is used after AVRCP profile connection.

Parameters

Parameter	Description
remote_dev_addr	Remote device address
version	AVRCP profile version

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
rsi_bt_rsp_avrcp_remote_version_t version;
status = rsi_bt_avrcp_get_remote_version (REMOTE_BD_ADDR, &version);
```

9.5.12 rsi_bt_avrcp_get_capabilities

Prototype

```
int32_t rsi_bt_avrcp_get_capabilities(uint8_t *remote_dev_addr,
uint8_t capability_type, rsi_bt_rsp_avrcp_get_capabilities_t
*cap_list)
#define  RSI_MAX_ATT                        5
typedef struct rsi_bt_rsp_avrcp_get_capabilities_s{
  uint32_t  nbr_ids;
  uint32_t  ids[RSI_MAX_ATT];
} rsi_bt_rsp_avrcp_get_capabilities_t;
```

Description
This API requests avrcp to get the media player support events and campany IDs.

Precondition
This api is used after AVRCP profile connection.

Parameters

Parameter	Description
remote_dev_addr	Remote device address
`capability_type`	AVRCP_SUPPORT_EVENTS_CAPABILITIES_ID 3
`cap_list`	Number of IDs and ID list

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
rsi_bt_rsp_avrcp_get_capabilities_t  cap_list = {0};
status = rsi_bt_avrcp_get_capabilities (REMOTE_BD_ADDR,
AVRCP_SUPPORT_EVENTS_CAPABILITIES_ID, &cap_list)
```

9.5.13 rsi_bt_avrcp_reg_notification

Prototype

```
int32_t rsi_bt_avrcp_reg_notification (uint8_t *remote_dev_addr,
uint8_t  event_id)
```

Description
This API requests avrcp to register the media player notification events at remote device.

Precondition
This api is used after AVRCP profile connection.

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Parameter	Description
event_id	AVRCP_EVENT_PLAYBACK_STATUS_CHANGED 0x01 AVRCP_EVENT_TRACK_CHANGED 0x02 AVRCP_EVENT_TRACK_REACHED_END 0x03 AVRCP_EVENT_TRACK_REACHED_START 0x04 AVRCP_EVENT_PLAYBACK_POS_CHANGED 0x05 AVRCP_EVENT_BATT_STATUS_CHANGED 0x06 AVRCP_EVENT_SYSTEM_STATUS_CHANGED 0x07 AVRCP_EVENT_PLAYER_APPLICATION_SETTING_CHANGED 0x08 AVRCP_EVENT_NOW_PLAYING_CONTENT_CHANGED 0x09 AVRCP_EVENT_AVAILABLE_PLAYERS_CHANGED 0x0a AVRCP_EVENT_ADDRESSED_PLAYER_CHANGED 0x0b AVRCP_EVENT_UIDS_CHANGED 0x0c AVRCP_EVENT_VOLUME_CHANGED 0x0d

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
status = rsi_bt_avrcp_reg_notification (REMOTE_BD_ADDR,
AVRCP_EVENT_VOLUME_CHANGED)
```

9.5.14 rsi_bt_avrcp_reg_notify_resp

Prototype

```
int32_t rsi_bt_avrcp_reg_notify_resp (uint8_t *remote_dev_addr,
uint8_t event_id, uint8_t event_data_len, uint8_t *event_data)
```

Description
This API gives response for the register notification requist.

Precondition
This api is used after AVRCP profile connection.

Parameters

Parameter	Description
remote_dev_addr	Remote device address
event_id	AVRCP event Id
event_data	interim data for event id

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
uint8_t event_data[8] = {0};
status = rsi_bt_avrcp_reg_notify_resp (REMOTE_BD_ADDR,
AVRCP_EVENT_TRACK_CHANGED, sizeof(event_data), event_data);
```

9.5.15 rsi_bt_avrcp_notify

Prototype

```
int32_t rsi_bt_avrcp_notify (uint8_t *remote_dev_addr, uint8_t
event_id, notify_val_t *p_notify_val)
```

Description
This API is used to send player notification to remote device.

Precondition
This api is used after AVRCP profile connection.

Parameters

Parameter	Description
remote_dev_addr	Remote device address
event_id	AVRCP event Id
p_notify_val	pointer to notify value info structure

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
notify_val_t  notify = {0};
status = rsi_bt_avrcp_notify (REMOTE_BD_ADDR,
AVRCP_EVENT_TRACK_CHANGED, &notify);
```

9.5.16 rsi_bt_avrcp_get_play_status

Prototype

```
int32_t rsi_bt_avrcp_get_play_status (uint8_t *remote_dev_addr,
rsi_bt_rsp_avrcp_get_player_status_t *play_status)
typedef struct rsi_bt_rsp_avrcp_get_player_status_s{
    uint32_t song_len;
    uint32_t song_pos;
    uint8_t  play_status;
} rsi_bt_rsp_avrcp_get_player_status_t;
```

Description
This API requests avrcp to get the media player status from remote device.

Precondition
This api is used after AVRCP profile connection.

Parameters

Parameter	Description
remote_dev_addr	Remote device address
`play_status`	player status info

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
int32_t status = 0;
rsi_bt_rsp_avrcp_get_player_status_t play_status;
status = rsi_bt_avrcp_get_play_status (REMOTE_BD_ADDR, &play_status);
```

9.6 GAP Event Callback APIs

9.6.1 rsi_bt_on_role_change_t

9.6.1.1 Prototype

```
typedef void (*rsi_bt_on_role_change_t) (uint16_t resp_status,
rsi_bt_event_role_change_t *role_change_status);
```

9.6.1.2 Description

This callback function will be called if the role change status event is received from the module.

9.6.1.3 Precondition

None

9.6.1.4 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

9.6.1.5 rsi_bt_event_role_change_t

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

9.6.2 **rsi_bt_on_connect_t**

9.6.2.1 **Prototype**

```
typedef void (*rsi_bt_on_connect_t) (uint16_t resp_status,
rsi_bt_event_bond_t *bond_response);
```

9.6.2.2 **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

9.6.2.3 **Precondition**

None

9.6.2.4 **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

9.6.2.5 rsi_bt_event_bond_t

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

9.6.3 **rsi_bt_on_unbond_t**

9.6.3.1 **Prototype**

```
typedef void (*rsi_bt_on_unbond_t) (uint16_t resp_status,
rsi_bt_event_unbond_t *unbond_status);
```

9.6.3.2 **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.

9.6.3.3 **Precondition**

None

9.6.3.4 **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

9.6.3.5 rsi_bt_event_unbond_t

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

9.6.4 **rsi_bt_on_disconnect_t**

9.6.4.1 **Prototype**

```
typedef void (*rsi_bt_on_disconnect_t) (uint16_t resp_status,
rsi_bt_event_disconnect_t *bt_disconnect);
```

9.6.4.2 **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.

9.6.4.3 **Precondition**

None

9.6.4.4 **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

9.6.4.5 rsi_bt_event_disconnect_t

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

9.6.5 **rsi_bt_on_scan_resp_t**

9.6.5.1 **Prototype**

```
typedef void (*rsi_bt_on_scan_resp_t) (uint16_t resp_status,
rsi_bt_event_inquiry_response_t *single_scan_resp);
```

9.6.5.2 **Description**

This callback function will be called if the single scan response is received from the module in response to inuiry command.

9.6.5.3 **Precondition**

None

9.6.5.4 **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

9.6.5.5 rsi_bt_event_inquiry_response_t

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

9.6.6 **rsi_bt_on_remote_name_resp_t**

9.6.6.1 **Prototype**

```
typedef void (*rsi_bt_on_remote_name_resp_t) (uint16_t resp_status,
rsi_bt_event_remote_device_name_t *name_resp);
```

9.6.6.2 **Description**

This callback is called if the remote name request command response is received from the module.

9.6.6.3 **Precondition**

None

9.6.6.4 **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

9.6.6.5 rsi_bt_event_remote_device_name_t

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

9.6.7 **rsi_bt_on_passkey_display_t**

9.6.7.1 **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);
```

9.6.7.2 **Description**

This callback function is called if the passkey display request is received from the module.

9.6.7.3 **Precondition**

None

9.6.7.4 **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

9.6.7.5 rsi_bt_event_user_passkey_display_t

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

9.6.8 **rsi_bt_on_remote_name_request_cancel_t**

9.6.8.1 **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);
```

9.6.8.2 **Description**

This callback is called if the remote name request cancels the command response received from the module.

9.6.8.3 **Precondition**

None

9.6.8.4 **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 This is the structure pointer for rsi_bt_event_remote_name_request_cancel_t structure

9.6.8.5 rsi_bt_event_remote_name_request_cancel_t

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

9.6.9 rsi_bt_on_confirm_request_t

9.6.9.1 Prototype

```
typedef void (*rsi_bt_on_confirm_request_t) (uint16_t resp_status,
rsi_bt_event_user_confirmation_request_t *user_confirmation_request);
```

9.6.9.2 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.

9.6.9.3 Precondition

None

9.6.9.4 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

9.6.9.5 rsi_bt_event_user_confirmation_request_t

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

9.6.10 **rsi_bt_on_pincode_request_t**

9.6.10.1 **Prototype**

```
typedef void (*rsi_bt_on_pincode_request_t) (uint16_t resp_status,
rsi_bt_event_user_pincode_request_t *user_pincode_request);
```

9.6.10.2 **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.

9.6.10.3 **Precondition**

None

9.6.10.4 **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

9.6.10.5 rsi_bt_event_user_pincode_request_t

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

9.6.11 **rsi_bt_on_passkey_request_t**

9.6.11.1 **Prototype**

```
typedef void (*rsi_bt_on_passkey_request_t) (uint16_t resp_status,
rsi_bt_event_user_passkey_request_t *user_passkey_request);
```

9.6.11.2 **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.

9.6.11.3 **Precondition**

None

9.6.11.4 **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

9.6.11.5 rsi_bt_event_user_passkey_request_t

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

9.6.12 **rsi_bt_on_inquiry_complete_t**

9.6.12.1 **Prototype**

```
typedef void (*rsi_bt_on_inquiry_complete_t) (uint16_t resp_status);
```

9.6.12.2 **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.

9.6.12.3 **Precondition**

None

9.6.12.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure

9.6.13 **rsi_bt_on_auth_complete_t**

9.6.13.1 **Prototype**

```
typedef void (*rsi_bt_on_auth_complete_t) (uint16_t resp_status,
rsi_bt_event_auth_complete_t *auth_complete);
```

9.6.13.2 **Description**

This callback function is called if authentication complete indication is received from the module.

9.6.13.3 **Precondition**

None

9.6.13.4 **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

9.6.13.5 rsi_bt_event_auth_complete_t

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

9.6.14 **rsi_bt_on_linkkey_request_t**

9.6.14.1 **Prototype**

```
typedef void (*rsi_bt_on_linkkey_request_t) (uint16_t resp_status,
rsi_bt_event_user_linkkey_request_t *user_linkkey_request);
```

9.6.14.2 **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.

9.6.14.3 **Precondition**

None

9.6.14.4 **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

9.6.14.5 rsi_bt_event_user_linkkey_request_t

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

9.6.15 **rsi_bt_on_ssp_complete_t**

9.6.15.1 **Prototype**

```
typedef void (*rsi_bt_on_ssp_complete_t) (uint16_t resp_status,
rsi_bt_event_ssp_complete_t *ssp_complete);
```

9.6.15.2 **Description**

This callback function is called if SSP complete status is received from the module.

9.6.15.3 **Precondition**

None

9.6.15.4 **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

9.6.15.5 rsi_bt_event_ssp_complete_t

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

9.6.16 **rsi_bt_on_linkkey_save_t**

9.6.16.1 **Prototype**

```
typedef void (*rsi_bt_on_linkkey_save_t) (uint16_t resp_status,
rsi_bt_event_user_linkkey_save_t *user_linkkey_save);
```

9.6.16.2 **Description**

This callback function is called if linkkey save is received from the module.

9.6.16.3 **Precondition**

None

9.6.16.4 **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

9.6.16.5 rsi_bt_event_user_linkkey_save_t

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

9.6.17 **rsi_bt_on_get_services_t**

9.6.17.1 **Prototype**

```
typedef void (*rsi_bt_on_get_services_t) (uint16_t resp_status,
rsi_bt_resp_query_services_t  *service_list);
```

9.6.17.2 **Description**

This callback function is called if the get services command response is received from the module.

9.6.17.3 **Precondition**

None

9.6.17.4 **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

9.6.17.5 rsi_bt_resp_query_services_t

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

9.6.18 **rsi_bt_on_search_service_t**

9.6.18.1 **Prototype**

```
typedef void (*rsi_bt_on_search_service_t) (uint16_t resp_status,
uint8_t *remote_dev_addr, uint8_t status);
```

9.6.18.2 **Description**

This callback function is called if the search service command response is received from the module.

9.6.18.3 **Precondition**

None

9.6.18.4 **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

9.6.19 **rsi_bt_on_mode_change_t**

9.6.19.1 **Prototype**

```
typedef void (*rsi_bt_on_mode_chnage_t) (uint16_t resp_status,
rsi_bt_event_mode_change_t  *mode_change);
```

9.6.19.2 **Description**

This callback function is called when the local device enters / exits the Sniff mode.

9.6.19.3 **Precondition**

None

9.6.19.4 **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

9.6.19.5 rsi_bt_event_mode_change_t

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

Variables	Description
mode_interval	This specifies the time interval to each mode

9.6.20 **rsi_bt_on_sniff_subrating_t**

9.6.20.1 **Prototype**

```
typedef void (*rsi_bt_on_sniff_subrating_t) (uint16_t resp_status,
rsi_bt_event_sniff_subrating_t  *sniff_subrating);
```

9.6.20.2 **Description**

This callback function is called when Sniff subrating is enabled or the parameters are negotiated with the remote deivce.

9.6.20.3 **Precondition**

None

9.6.20.4 **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_eventt_sniff_subrating_t structure

9.6.20.5 rsi_bt_event_sniff_subrating_t

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 shall use
min_local_timeout	This is the base sniff subrate timeout in baseband slots that the local device shall use

9.7 SPP Event Callback APIs

9.7.1 rsi_bt_on_spp_connect_t

9.7.1.1 Prototype

```
typedef void (*rsi_bt_on_spp_connect_t) (uint16_t resp_status,
rsi_bt_event_spp_connect_t *spp_connect);
```

9.7.1.2 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.

9.7.1.3 **Precondition**

None

9.7.1.4 **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

9.7.1.5 rsi_bt_event_spp_connect_t

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

9.7.2 **rsi_bt_on_spp_disconnect_t**

9.7.2.1 **Prototype**

```
typedef void (*rsi_bt_on_spp_disconnect_t) (uint16_t resp_status,
rsi_bt_event_spp_disconnect_t *spp_disconnect);
```

9.7.2.2 **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.

9.7.2.3 **Precondition**

None

9.7.2.4 **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

9.7.2.5 rsi_bt_event_spp_disconnect_t

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

9.7.3 **rsi_bt_on_spp_rx_data_t**

9.7.3.1 **Prototype**

```
typedef void (*rsi_bt_on_spp_rx_data_t) (uint16_t resp_status,
rsi_bt_event_spp_receive_t *bt_event_spp_receive);
```

9.7.3.2 **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.

9.7.3.3 **Precondition**

None

9.7.3.4 **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

9.7.3.5 rsi_bt_event_spp_receive_t

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

9.8 A2DP Event Callback APIs

9.8.1 rsi_bt_on_a2dp_connect_t

9.8.1.1 Prototype

```
typedef void (*rsi_bt_on_a2dp_connect_t) (uint16_t resp_status,
rsi_bt_event_a2dp_connect_t *a2dp_connect);
```

9.8.1.2 Description

This callback is called when A2DP connected event is raised from the module. This event will be given by the module when A2DP profile level connection happens from either side.

9.8.1.3 Precondition

rsi_bt_a2dp_init() should be call.

9.8.1.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_connect	Structure pointer holding the response payload for the a2dp connection event This is the structure pointer for rsi_bt_event_a2dp_connect_t structure

9.8.1.5 rsi_bt_event_a2dp_connect_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_connect_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_connect_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

9.8.2 **rsi_bt_on_a2dp_disconnect_t**

Prototype

```
typedef void (*rsi_bt_on_a2dp_disconnect_t) (uint16_t resp_status,
rsi_bt_event_a2dp_disconnect_t *a2dp_disconnect);
```

9.8.2.1 **Description**

This callback is called when A2DP disconnected event is raised from the module. This event will be given by the module when a2dp profile level disconnection happens from either side.

9.8.2.2 **Precondition**

None

9.8.2.3 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_disconnect	Structure pointer holding the response payload for the a2dp disconnection event This is the structure pointer for rsi_bt_event_a2dp_disconnect_t structure

9.8.2.4 rsi_bt_event_a2dp_disconnect_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_disconnect_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_disconnect_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

9.8.3 **rsi_bt_on_a2dp_configure_t**

9.8.3.1 **Prototype**

```
typedef void (*rsi_bt_on_a2dp_configure_t) (uint16_t resp_status,
rsi_bt_event_a2dp_configure_t *a2dp_configure);
```

9.8.3.2 **Description**

This callback is called when A2DP configured event is raised from the module. This event will be given by the module when a2dp profile onfiguration happens from either side.

9.8.3.3 **Precondition**

None

9.8.3.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_configure	Structure pointer holding the response payload for the a2dp configuratoin event This is the structure pointer for rsi_bt_event_a2dp_configure_t structure

9.8.3.5 rsi_bt_event_a2dp_configure_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_configure_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_configure_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

9.8.4 **rsi_bt_on_a2dp_open_t**

9.8.4.1 **Prototype**

```
typedef void (*rsi_bt_on_a2dp_open_t) (uint16_t resp_status,
rsi_bt_event_a2dp_open_t *a2dp_open);
```

9.8.4.2 **Description**

This callback is called when A2DP open event is raised from the module. This event will be given by the module when a2dp opens in either side.

9.8.4.3 **Precondition**

None

9.8.4.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_open	Structure pointer holding the response payload for the a2dp open event This is the structure pointer for rsi_bt_event_a2dp_open_t structure

9.8.4.5 rsi_bt_event_a2dp_open_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_open_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_open_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

9.8.5 **rsi_bt_on_a2dp_start_t**

9.8.5.1 **Prototype**

```
typedef void (*rsi_bt_on_a2dp_start_t) (uint16_t resp_status,
rsi_bt_event_a2dp_start_t *a2dp_start);
```

9.8.5.2 **Description**

This callback is called when A2DP start event is raised from the module. This event will be given by the module when a2dp profile starts from either side.

9.8.5.3 **Precondition**

None

9.8.5.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_start	Structure pointer holding the response payload for the a2dp start event This is the structure pointer for rsi_bt_event_a2dp_start_t structure

9.8.5.5 rsi_bt_event_a2dp_start_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_start_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
        uint16_t  sample_freq;
        uint16_t  rem_mtu_size;
} rsi_bt_event_a2dp_start_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
sample_freq	Sampling frequency of the A2DP Stream codec
rem_mtu_size	MTU size supported by the remote device

9.8.6 **rsi_bt_on_a2dp_suspend_t**

9.8.6.1 **Prototype**

```
typedef void (*rsi_bt_on_a2dp_suspend_t) (uint16_t resp_status,
rsi_bt_event_a2dp_suspend_t *a2dp_suspend);
```

9.8.6.2 **Description**

This callback is called when A2DP suspend event is raised from the module. This event will be given by the module when a2dp profile level suspend happens from either side.

9.8.6.3 **Precondition**

None

9.8.6.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_suspend	Structure pointer holding the response payload for the a2dp suspend event This is the structure pointer for rsi_bt_event_a2dp_suspend_t structure

9.8.6.5 rsi_bt_event_a2dp_suspend_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_suspend_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_suspend_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

9.8.7 **rsi_bt_on_a2dp_abort_t**

9.8.7.1 **Prototype**

```
typedef void (*rsi_bt_on_a2dp_abort_t) (uint16_t resp_status,
rsi_bt_event_a2dp_abort_t *a2dp_abort);
```

9.8.7.2 **Description**

This callback is called when A2DP abort event is raised from the module. This event will be given by the module when a2dp profile level abort happens from either side.

9.8.7.3 **Precondition**

None

9.8.7.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_abort	Structure pointer holding the response payload for the a2dp abort event This is the structure pointer for rsi_bt_event_a2dp_abort_t structure

9.8.7.5 rsi_bt_event_a2dp_abort_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_abort_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_abort_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

9.8.8 **rsi_bt_on_a2dp_close_t**

9.8.8.1 **Prototype**

```
typedef void (*rsi_bt_on_a2dp_close_t) (uint16_t resp_status,
rsi_bt_event_a2dp_close_t *a2dp_close);
```

9.8.8.2 **Description**

This callback is called when A2DP close event is raised from the module. This event will be given by the module when a2dp closed from either side.

9.8.8.3 **Precondition**

None

9.8.8.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_close	Structure pointer holding the response payload for the a2dp close event This is the structure pointer for rsi_bt_event_a2dp_close_t structure

9.8.8.5 rsi_bt_event_a2dp_close_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_close_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_close_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

9.8.9 **rsi_bt_on_a2dp_encode_data_t**

9.8.9.1 **Prototype**

```
typedef void (*rsi_bt_on_a2dp_encode_data_t) (uint16_t resp_status,
rsi_bt_event_a2dp_encode_data_t *a2dp_encode_data);
```

9.8.9.2 **Description**

This callback is called when A2DP encode event is raised from the module. This event will be given by the module when a2dp encoded data received.

9.8.9.3 **Precondition**

None

9.8.9.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_encode_data	Structure pointer holding the response payload for the a2dp encoded data event This is the structure pointer for rsi_bt_event_a2dp_encode_data_t structure

9.8.9.5 rsi_bt_event_a2dp_encode_data_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_encode_data_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
        uint16_t encode_data_len;
        uint8_t  encode_data[RSI_BT_MAX_PAYLOAD_SIZE];
} rsi_bt_event_a2dp_encode_data_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
encode_data_len	Length of the received encoded data
encode_data	Buffer holding the received a2dp encoded data. It holds the max payload length of 1k bytes

9.8.10 **rsi_bt_on_a2dp_pcm_data_t**

9.8.10.1 **Prototype**

```
typedef void (*rsi_bt_on_a2dp_pcm_data_t) (uint16_t resp_status,
rsi_bt_event_a2dp_pcm_data_t *a2dp_pcm_data);
```

9.8.10.2 **Description**

This callback is called when A2DP pcm event is raised from the module. This event will be given by the module when a2dp pcm data received.

9.8.10.3 **Precondition**

None

9.8.10.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_pcm_data	Structure pointer holding the response payload for the a2dp pcm data event This is the structure pointer for rsi_bt_event_a2dp_pcm_data_t structure

9.8.10.5 rsi_bt_event_a2dp_pcm_data_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_pcm_data_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
        uint16_t pcm_data_len;
        uint8_t  pcm_data[RSI_BT_MAX_PAYLOAD_SIZE];
} rsi_bt_event_a2dp_pcm_data_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes
pcm_data_len	Length of the received pcm data
pcm_data	Buffer holding the received a2dp pcm data. It holds the max payload length of 1k bytes

9.8.11 **rsi_bt_on_a2dp_data_req_t**

9.8.11.1 **Prototype**

```
typedef void (*rsi_bt_on_a2dp_data_req_t) (uint16_t resp_status,
rsi_bt_event_a2dp_more_data_req_t *a2dp_more_data_req);
```

9.8.11.2 **Description**

This callback is called when more data request is raised from the module. This event will be given by the module requires data to be transmitted.

9.8.11.3 **Precondition**

None

9.8.11.4 **Parameters**

Variables	Description
resp_status	Response status for the event Zero on success Nonzero on failure
a2dp_more_data_req	Structure pointer holding the response payload for the a2dp more data request event This is the structure pointer for rsi_bt_event_a2dp_more_data_req_t structure

9.8.11.5 rsi_bt_event_a2dp_more_data_req_t

Structure Prototype

```
typedef struct rsi_bt_event_a2dp_more_data_req_s
{
        uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_more_data_req_t;
```

Structure Variables

Variables	Description
dev_addr	This array holds the device address of length 6 bytes

10 Bluetooth Low Energy APIs

This section contains description about BLE APIs to initialize and configure the module in BLE mode.

Note
A new BLE API has to be called only after getting the response for the previous API.

Test Mode API

This section describes the APIs that are being used for testing purposes

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 command is used to start a test where the DUT receives test reference packets at a fixed interval.

Parmeters

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 */
```

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 a test where the DUT generates test reference packets at a fixed interval.

Parameters

Variables	Description
tx_channel	RF Channel (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.
tx_len	Length in bytes of payload data in each packet (1 - 251 bytes).

Variables	Description
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 */
```

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 le 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);
```

rsi_ble_per_transmit

Prototype

```
int32_t rsi_ble_per_transmit (struct rsi_ble_per_transmit_s
*rsi_ble_per_tx);
```

Description

This API initiate 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 contain 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

Variables	Description
	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
tx_chnl_num	Tx channel number
scrambler_seed	This field takes the scrambler seed value configured to 0
le_chnl_type	This is the LE channel type (data or advertise channel) 0x00 → Advertise Channel 0x01 → Data Channel
freq_hop_en	This field defines the frequency hopping type to be used 0 → No Hopping 1 → Fixed Hopping 2 → Random Hopping
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 1 → PLL_MODE1
rf_type	This field defines the selection of RF type (internal/external) 0 → BT_EXTERNAL_RF 1 → BT_INTERNAL_RF
rf_chain	This field corresponds to the selection of RF chain (HP/LP) to be used 2 → BT_HP_CHAIN

Variables	Description
	3 → BT_LP_CHAIN
pkt_len	length of the packet to be transmitted
payload_type	Type of payload data sequence 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_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
num_pkts	This field defines the number of packets to be transmitted, default to zero for coninuous 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);
```

rsi_ble_per_receive

Prototype

```
int32_t rsi_ble_per_receive (struct rsi_ble_per_receive_s
*rsi_ble_per_rx);
```

Description

This API initiate 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 contain 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

Variables	Description
tx_chnl_num	Tx channel number
scrambler_seed	This field takes the scrambler seed value configured to 0
le_chnl_type	This is the LE channel type (data or advertise channel) 0x00 → Advertise Channel 0x01 → Data Channel
freq_hop_en	This field defines the frequency hopping type to be used 0 → No Hopping 1 → Fixed Hopping 2 → Random Hopping
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 1 → PLL_MODE1
rf_type	This field defines the selection of RF type (internal/external) 0 → BT_EXTERNAL_RF 1 → BT_INTERNAL_RF
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

Variables	Description
	0 → Duty Cycling Disabled 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);
```

10.1 GAP API

This section describes the GAP APIs

10.2 **rsi_ble_set_random_address**

10.2.1 **Prototype**

```
int32_t rsi_ble_set_random_address(void);
```

10.2.2 **Description**

This API sets the LE random device address.

10.2.3 **Precondition**

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

10.2.4 Parameters

None

10.2.5 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

10.2.6 Example

```
int32_t status = 0;
status = rsi_ble_set_random_address();
```

10.3 rsi_ble_set_random_address_with_value

10.3.1 Prototype

```
int32_t rsi_ble_set_random_address_with_value(uint8_t *random_addr);
```

10.3.2 Description

This API request the local device to set a random address.

10.3.3 Precondition

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

10.3.4 Parameters

Parameter	Description
random_addr	random address of the device to be set

10.3.5 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

10.3.6 **Example**

```
int32_t status = 0;
status = rsi_ble_set_random_address_with_value(random_addr);
```

10.4 rsi_ble_start_advertising

10.4.1 **Prototype**

```
int32_t rsi_ble_start_advertising(void);
```

10.4.2 **Description**

This API requests the local device to start advertising.

10.4.3 **Precondition**

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

10.4.4 **Parameters**

None

10.4.5 **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

10.4.6 **Example**

```
status = rsi_ble_start_advertising();
```

10.5 rsi_ble_start_advertising_with_values

10.5.1 Prototype

```
int32_t rsi_ble_start_advertising_with_values(void *rsi_ble_adv);
```

10.5.2 Description

This API request the local device to start advertising with specified values.

10.5.3 Precondition

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

10.5.4 Parameters

Parameter	Description
rsi_ble_adv	This structure pointer holds the information of advertising values This variable is pointer of rsi_ble_req_adv_t structure

10.5.5 rsi_ble_req_adv_t

10.5.5.1 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;
```

10.5.5.2 Structure Description

This structure defines the format for the advertising values

10.5.5.3 **Structure Variables**

Variables	Description
status	advertising status - disable/enable
adv_type	advertising type used during advertising
filter_type	advertising filter type
direct_addr_type	address type of the device to which directed advertising has to be done
direct_addr	address of the device to which directed advertising has to be done
adv_int_min	advertising interval min
adv_int_max	advertising interval max
own_addr_type	address of the local device
adv_channel_map	advertising channel map

10.5.5.4 **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

10.5.5.5 **Example**

```
int32_t status = 0;
rsi_ble_req_adv_t rsi_ble_adv;
status = rsi_ble_start_advertising_with_values(&rsi_ble_adv);
```

10.6 **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);

10.6.1 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();

10.6.2 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

10.6.3 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();
```

10.6.4 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);

10.6.5 rsi_ble_connect_cancel

Prototype

```
int32_t rsi_ble_connect_cancel(int8_t *remote_dev_address)
```

Description
This API cancesl 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);
```

10.6.6 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);

10.6.7 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);
```

10.6.8 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

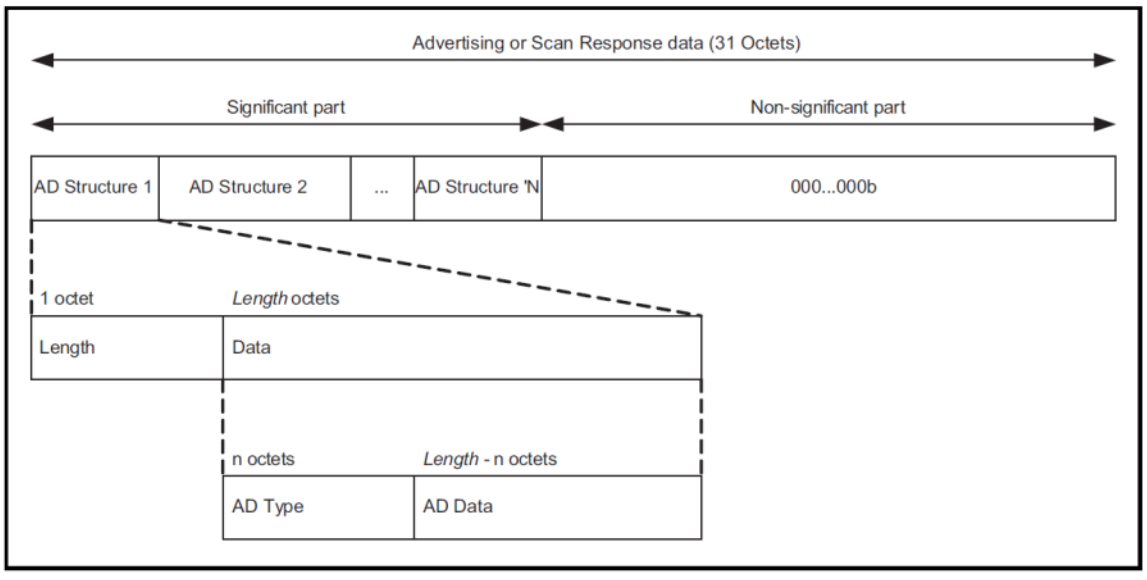
Note
1. The maximum length of advertising data payload is 31 bytes.
2. The basic format of advertise payload record contains length and data as below:

1 octet 1 octet (length-1) octets

Length	AD type	AD data

3. Multiple advertise records can be included in the advertise data but the total length should be less than or equal to 31.

The details regarding AD type field is specified in Volume 3, Part C, Chapter 18, Appendix C in Bluetooth 4.0 specification.



Error rendering macro 'captioneditem' : Plugin license error: EXPIRED
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);
```

10.6.9 rsi_ble_smp_pair_request

Prototype

```
int32_t rsi_ble_smp_pair_request (uint8_t *remote_dev_address,
uint8_t io_capability)
```

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

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_smp_pair_request (str_remote_address,
RSI_BLE_SMP_IO_CAPABILITY);
```

10.6.10 rsi_ble_smp_pair_response

Prototype

```
int32_t rsi_ble_smp_pair_response(uint8_t *remote_dev_address,
uint8_t io_capability)
```

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

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_response (str_remote_address,
RSI_BLE_SMP_IO_CAPABILITY);
```

10.6.11 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);
```

10.6.12 rsi_ble_get_le_ping_timeout

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);

10.6.13 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 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 input parameter holds timeout value for authentication payload command.

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_set_le_ping_timeout (remote_dev_address, time_out);

10.7 rsi_ble_set_scan_response_data

10.7.1 Prototype

```
int32_t rsi_ble_set_scan_response_data(uint8_t *data, uint16_t
data_len);
```

10.7.2 Description

This API request the local devuce to set the scanresponse data.

10.7.3 Precondition

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

10.7.4 Parameters

Parameter	Description
data	data to be sent is filled in this
data_len	length of data to be sent

10.7.5 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

10.7.6 Example

```
status = rsi_ble_set_scan_response_data (adv_arr, strlen (adv));
```

10.8 rsi_ble_clear_whitelist

10.8.1 Prototype

```
int32_t rsi_ble_clear_whitelist(void);
```

10.8.2 Description

This API clears all the BD address present in white list

10.8.3 Precondition

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

10.8.4 Parameters

None

10.8.5 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

10.8.6 Example

```
status = rsi_ble_clear_whitelist();
```

10.9 rsi_ble_addto_whitelist

10.9.1 Prototype

```
int32_t rsi_ble_addto_whitelist( int8_t *dev_address,uint8_t
dev_addr_type);
```

10.9.2 Description

This API Adds perticular BD address to white list.

10.9.3 Precondition

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

10.9.4 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

10.9.5 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

10.9.6 Example

```
status =
rsi_ble_addto_whitelist(RSI_BLE_WHITELIST_DEV_ADDR1,RSI_BLE_WHITELIST
_DEV_ADDR1_TYPE);
```

10.10 rsi_ble_deletefrom_whitelist

10.10.1 Prototype

```
int32_t rsi_ble_deletefrom_whitelist(int8_t *dev_address,uint8_t
dev_addr_type)
```

10.10.2 Description

This API Deletes perticular BD address from white list.

10.10.3 Precondition

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

10.10.4 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

10.10.5 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

10.10.6 Example

```
status =
rsi_ble_deletefrom_whitelist(RSI_BLE_WHITELIST_DEV_ADDR2,RSI_BLE_WHIT
ELIST_DEV_ADDR2_TYPE);
```

10.11 rsi_ble_vendor_rf_type

10.11.1 Prototype

```
int32_t rsi_ble_vendor_rf_type (uint8_t perf_mode,uint8_t
duty_cycling);
```

10.11.2 Description

This API gives vendor specific command to the controller to set rf type.

10.11.3 Precondition

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

10.11.4 Parameters

Parameter	Description
perf_mode	performance mode
duty_cycling	duty cycling

10.11.5 **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

10.11.6 **Example**

```
status = rsi_ble_vendor_rf_type (perf_mode, duty_cycling);
```

10.12 rsi_ble_start_encryption

10.12.1 **Prototype**

```
int32_t rsi_ble_start_encryption (uint8_t *remote_dev_address,
uint16_t ediv, uint8_t *rand, uint8_t *ltk);
```

10.12.2 **Description**

This API start the encription process with remote device.

10.12.3 **Precondition**

Encryption enabled event should come before calling this api for seconf time smp connection.

10.12.4 **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

10.12.5 **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

10.12.6 **Example**

```
status =
rsi_ble_start_encryption(remote_dev_address,ediv,rand_arr,ltk_arr);
```

10.13 Basic Structure defines

10.14 **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.

10.14.1 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.

10.14.2 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.

10.14.3 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.

10.14.4 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

10.14.5 att_desc_t

Structure

```
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) .

10.14.6 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.

10.14.7 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.

10.14.8 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
services	This include service list. The maximum value is 5.

10.14.9 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.

10.14.10 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.

10.14.11 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 maintainthe 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.

10.14.12 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.

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;

10.14.13 rsi_ble_resp_local_att_value_t

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.

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;

10.15 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

10.15.1 rsi_ble_cbfc_conn_req_t

Structure

```
typedef struct rsi_ble_cbfc_conn_req_s {
    uint8_t     dev_addr[RSI_DEV_ADDR_LEN];
    uint8_t     psm;
} rsi_ble_cbfc_conn_req_t;
```

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.

10.15.2 rsi_ble_cbfc_data_tx_t

Structure

```
typedef struct rsi_ble_cbfc_data_tx_s {
    uint8_t     dev_addr[RSI_DEV_ADDR_LEN];
    uint16_t    lcid;
    uint16_t    len;
    uint8_t     data[RSI_DEV_ATT_LEN];
} rsi_ble_cbfc_data_tx_t;
```

Description
This command is used to initiate new PSM connection with remote device.

Structure Variables

Variables	Description
dev_addr	Remote Device address

Variables	Description
lcid	local channel identifiervalue
len	length of the data to be sent to remote device.
data	Actual data that has to be sent

10.15.3 rsi_ble_cbfc_disconn_t

Structure

```
typedef struct rsi_ble_cbfc_disconn_s {
    uint8_t     dev_addr[RSI_DEV_ADDR_LEN];
    uint16_t    lcid;
} rsi_ble_cbfc_disconn_t;
```

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

10.16 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.

10.16.1 GATT Client APIs

10.16.1.1 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
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);

10.16.1.2 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

Parameter	Description
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);

10.16.1.3 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

Parameter	Description
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);
```

10.16.1.4 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 **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);

10.16.1.5 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

Parameter	Description
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);

10.16.1.6 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);

10.16.1.7 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);

10.16.1.8 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);

10.16.1.9 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
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);

10.16.1.10 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);
```

10.16.1.11 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, "redpine");

10.16.1.12 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

Parameter	Description
handle	attribute handle
Offset	attribute value offset
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);

10.16.1.13 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

Parameter	Description
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);

10.16.1.14 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);

10.17 rsi_ble_notify_value

10.17.1 Prototype

```
int32_t rsi_ble_notify_value (uint8_t  *dev_addr,
                              uint16_t  handle,
                              uint16_t  data_len,
                              uint8_t  *p_data);
```

10.17.2 Description

This API Notify the local value to remote device.

10.17.3 Precondition

Connection has to be established before calling this api

10.17.4 Parameters

Parameter	Description
dev_addr	remote bd address in string format
handle	local attributre handle
data_len	attribute value len
p_data	attribute value

10.17.5 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

10.17.6 **Example**

```
status int32_t rsi_ble_notify_value
(dev_addr,handle,data_len,p_data);
```

10.18 **rsi_ble_indicate_value**

10.18.1 **Prototype**

```
int32_t rsi_ble_indicate_value (uint8_t  *dev_addr,
                                uint16_t  handle,
                                uint16_t  data_len,
                                uint8_t  *p_data);
```

10.18.2 **Description**

This API Indicate the local value to remote device.

10.18.3 **Precondition**

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

10.18.4 **Parameters**

Parameter	Description
dev_addr	remote bd address in string format
handle	local attributre handle
data_len	attribute value len
p_data	attribute value

10.18.5 **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

10.18.6 Example

```
status = rsi_ble_indicate_value (dev_addr,handle,data_len,p_data);
```

10.19 rsi_ble_remove_gatt_service

10.19.1 Prototype

```
int32_t rsi_ble_remove_gatt_service (uint32_t  service_handler);
```

10.19.2 Description

This API Remove the GATT service record.

10.19.3 Precondition

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

10.19.4 Parameters

Parameter	Description
service_handler	GATT service record handler

10.19.5 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

10.19.6 Example

```
status = rsi_ble_remove_gatt_service (service_handler);
```

10.20 rsi_ble_remove_gatt_attibute

10.20.1 Prototype

```
int32_t rsi_ble_remove_gatt_attibute (uint32_t  service_handler,
uint16_t att_hndl);
```

10.20.2 Description

This API Remove the GATT service record.

10.20.3 Precondition

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

10.20.4 Parameters

Parameter	Description
service_handler	GATT service record handler
att_hndl	attribute handle

10.20.5 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

10.20.6 Example

```
status = rsi_ble_remove_gatt_attibute (service_handler,att_handle);
```

10.20.7 GATT Server APIs

Note
Please refer definitions for the following structures in "**Basic Structure defines**" section.

10.20.7.1 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 redpine 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 the BT ERROR Codes for more details.

Example

status = rsi_ble_add_service (service_uuid, &p_resp_serv);

10.20.7.2 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);
```

10.20.7.3 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** -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);
```

10.20.7.4 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);

10.20.7.5 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

Parameter	Description
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);
```

10.20.7.6 Feature 4.2

10.20.7.7 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

Parameter	Description
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

None.

Example

status = rsi_ble_setphy(str_addr, 1, 2, 0);

10.20.7.8 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 Connenction Interval
max_int	Maximum Connenction Interval
latency	Connection Latency
timeout	Supervision Timeout

Return Values

Value	Description
0	Connection Params Update command succeded
Non Zero	Failure

Example

status = rsi_ble_conn_params_update(str_addr, 160, 160, 0, 2000);

10.20.7.9 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 txoctets 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);

10.20.7.10 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 txoctets and tx time.

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

Parameters:

None

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

Parameter	Description
maxrxoctets	Maximum number of payload octets that the local Controller supports for reception of a single Link Layer packet on a data connection. .
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);

10.20.7.11 **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.

Parameters:

None

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);

10.20.8 **Privacy**

10.20.8.1 **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 resolvlist 2 for remove a device from resolvlist 3 for clear the entire resolvlist
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 );
```

10.20.8.2 **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
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);
```

10.20.8.3 **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 of time 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);
```

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 particular 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
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

Value	Description
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);
```

10.21 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 Remote device address reply_type 0 – negative reply, 1 – positive reply ltk 16Bytes long term key

Return Values

On Success : 0
On Failure : if return value is less than 0
-4 : Buffer not available to serve the command

10.22 Callback functions

10.22.1 GAP register callbacks

This function is used to register the call-back functions for synchronous GAP events.

10.22.1.1 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);
```

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

Return Values
None

For more information about each callback, please refer to GAP Callback Descriptions section.

10.22.2 GAP event callback descriptions

10.22.2.1 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

10.22.2.2 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

- 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
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

10.22.2.3 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:

- In case, disconnection is issued locally
- 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

10.22.2.4 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.

10.22.3 GAP Extended register callbacks

This function is used to register the callback functions for GAP Extended responses and events.

10.22.3.1 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

For more information about each callback, please refer to GAP Extended callbacks description section.

10.22.4 **GAP Extended callbacks description**

10.22.4.1 **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

10.22.4.2 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).

10.22.5 GATT register callbacks

This function is used to register the callback functions for GATT responses and events.

10.22.5.1 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
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

For more information about each callback, please refer to GATT response callbacks description section.

10.22.6 GATT Response callbacks description

10.22.6.1 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 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
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.

10.22.6.2 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 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
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.

10.22.6.3 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.

10.22.6.4 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.

- This callback function is called if the included service 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 included services found
reserved	uint8_t	Reserved
services	inc_service_t	This is the list of included services. The maximum value is 5.

10.22.6.5 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.

10.22.6.6 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.

10.22.6.7 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 has to 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.

10.22.7 GATT Event callbacks

- This callback function will be called if the GATT write / notify / indicate events are received.
- This callback has to be registered using rsi_ble_gatt_register_callbacks API

10.22.7.1 GATT Write event

Prototype

```
typedef struct rsi_ble_event_write_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint8_t handle[2];
 uint8_t length;
 uint8_t att_value[100];
 } 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
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.

10.22.7.2 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.

10.22.7.3 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

10.22.7.4 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

10.22.7.5 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 has to 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 has to 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 has to 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.

- This callback function is called if the included service 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 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
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.

10.22.8 SMP register callbacks

This function is used to register the call-back functions for an asynchronous SMP events.

10.22.8.1 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);
```

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

Return Values

None

For more information about each callback, please refer to SMP callbacks description section.

10.22.9 SMP event Callbacks Declarations

10.22.9.1 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.

10.22.9.2 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.

10.22.9.3 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.

10.22.9.4 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.

10.22.9.5 rsi_ble_on_encrypt_started_t

Prototype

```
typedef void (*rsi_ble_on_encrypt_started_t) (uint16_ resp_status);
```

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

10.22.9.6 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	
TxPhy	
TxPhy	

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	This is the device address of the disconnected device.
MaxTxOctates	Preferred maximum number of payload octets that the local Controller should include in a single Link Layer Data Channel PDU
MaxTxTime	The maximum time that the local Controller will take to send a Link Layer Data Channel PDU on this connection
MaxRxOctates	The maximum number of payload octets in a Link Layer packet that the local Controller expects to receive on this connection
MaxRxTime	The maximum time that the local Controller expects to take to receive a Link Layer packet on this connection

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

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

10.22.10 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

10.22.11 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 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 transmited.
Lcid	local device channel identifier

10.22.12 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 transferd 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

10.22.13 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 functions 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

10.22.14 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

10.23 Configuration parameters

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

10.23.1 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. 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

10.23.2 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 advetisers

10.23.3 Configure connection parameters

Define	Meaning
LE_SCAN_INTERVAL	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 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_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

11 BT/BLE Common API

11.1 Common API

This section explains the BT Common APIs.

11.1.1 rsi_bt_set_local_name

11.1.1.1 Prototype

```
int32_t rsi_bt_set_local_name(int8_t *local_name);
```

11.1.1.2 Description

This API sets the given name to local device.

11.1.1.3 Precondition

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

11.1.1.4 Parameters

Parameter	Description
local_name	This is the name to be set to the local device

11.1.1.5 Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

11.1.1.6 Example

```
#define RSI_BT_LOCAL_NAME "BT_SAMPLE_APP"
int32_t status = 0;
status = rsi_bt_set_local_name(RSI_BT_LOCAL_NAME);
```

11.1.2 rsi_bt_get_local_name

11.1.2.1 Prototype

```
int32_t rsi_bt_get_local_name (rsi_bt_resp_get_local_name_t
*bt_resp_get_local_name);
```

11.1.2.2 Description

This API requests the local device name.

11.1.2.3 Precondition

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

11.1.2.4 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.

11.1.2.5 rsi_bt_resp_get_local_name_t

Structure 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;
```

Structure Description

This structure describes the format of the get local name response structure

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);
```

11.1.3 **rsi_bt_get_rssi**

11.1.3.1 **Prototype**

```
int32_t rsi_bt_get_rssi(int8_t *dev_addr, uint8_t *resp);
```

11.1.3.2 **Description**

This API requests the RSSI of the remote device.

11.1.3.3 **Precondition**

rsi_bt_connect() API need to be called before this API

11.1.3.4 **Parameters**

Parameter	Description
dev_addr	This is the remote device address
resp	This parameter is to hold the response of this API

11.1.3.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

11.1.3.6 **Example**

```
uint8_t rsi_app_resp_rssi = 0;
status = rsi_bt_get_rssi(remote_dev_addr, &rsi_app_resp_rssi);
```

11.1.4 **rsi_bt_get_local_device_address**

11.1.4.1 **Prototype**

```
int32_t rsi_bt_get_local_device_address(uint8_t *resp);
```

11.1.4.2 **Description**

This API requests the local device address.

11.1.4.3 **Precondition**

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

11.1.4.4 **Parameters**

Parameter	Description
resp	This parameter is to hold the response of this API

11.1.4.5 **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);
```

11.1.5 **rsi_bt_init**

11.1.5.1 **Prototype**

```
int32_t rsi_bt_init(void);
```

11.1.5.2 **Description**

This API initializes the BT device.

11.1.5.3 **Precondition**

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

11.1.5.4 **Parameters**

None

11.1.5.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

11.1.5.6 **Example**

```
int32_t status = 0;
status = rsi_bt_init();
```

11.1.6 **rsi_bt_deinit**

11.1.6.1 **Prototype**

```
int32_t rsi_bt_deinit(void);
```

11.1.6.2 **Description**

This API deinitializes the BT device.

11.1.6.3 **Precondition**

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

11.1.6.4 **Parameters**

None

11.1.6.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

11.1.6.6 **Example**

```
int32_t status = 0;
status = rsi_bt_deinit();
```

11.1.7 rsi_bt_set_antenna

11.1.7.1 Prototype

```
int32_t rsi_bt_set_antenna(uint8_t antenna_value);
```

11.1.7.2 Description

This API selects either internal / external antenna on the chip.

11.1.7.3 Precondition

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

11.1.7.4 Parameters

Parameter	Description
antenna_value	This parameter is used to select either internal or external antenna

11.1.7.5 Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

11.1.7.6 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);
```

11.1.8 rsi_bt_set_feature_bitmap

11.1.8.1 Prototype

```
int32_t rsi_bt_set_feature_bitmap(uint32_t feature_bit_map);
```

11.1.8.2 Description

This API enables /disable the mentioned features.

11.1.8.3 Precondition

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

11.1.8.4 Parameters

Parameter	Description
Bits	
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

11.1.8.5 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);
```

11.1.9 **rsi_bt_set_antenna_tx_power_level**

11.1.9.1 **Prototype**

```
int32_t rsi_bt_set_antenna_tx_power_level(uint8_t protocol_mode,
int8_t tx_power);
```

11.1.9.2 **Description**

This API enables / disables the mentioned features.

11.1.9.3 **Precondition**

rsi_wireless_init() API need to be called before this API

11.1.9.4 **Parameters**

Parameter	Description
protocol_mode	1 – BT classic 2 – BT Low Energy
tx_power	Antenna transmit power level

11.1.9.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

11.1.9.6 **Example**

```
int32_t status = 0;
status = rsi_bt_set_antenna_tx_power_level(1,19);
```

11.1.10 **rsi_bt_power_save_profile**

11.1.10.1 **Prototype**

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

11.1.10.2 **Description**

This API selects the power save profile mode for BT / BLE.

11.1.10.3 **Precondition**

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

11.1.10.4 **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 support.

Note:
1) psp_type is only valid in psp_mode 2.
2) BT/BLE doesnot support in RSI_SLEEP_MODE_1.

11.1.10.5 **Return Values**

Value	Description
0	Successful execution of the command
Non Zero	Failure

11.1.10.6 **Example**

```
//! initiating power save in BT mode
status = rsi_bt_power_save_profile(RSI_SLEEP_MODE_2, RSI_MAX_PSP);
```

12 SAPI Error Codes

S. No.	Error codes (in hexadecimal format)	Description
1	0x00000000	None
2	0xFFFFFFFF	Time out error
3	0xFFFFFFFE	Invalid parameters
4	0xFFFFFFFD	Command given in incorrect state
5	0xFFFFFFFC	Packet allocation failure
6	0xFFFFFFFB	Command not supported
7	0XFFFFFFFA	Insufficient input buffer given
8	0xFFFFFFF9	Error in OS operation

13 Appendix

13.1 Example Applications

13.1.1 WLAN Example Applications

Example	Description	Application path
UDP server	This example demonstrates how to configure module in station mode and receive the data from the remote side using UDP socket	sapis/examples/wlan/ udp_server/rsi_udp_server.c
UDP client	This example demonstrates how to configure module in station mode and send data to the remote side using UDP client socket	sapis/examples/wlan/ udp_client/rsi_udp_client.c
TCP server	This example demonstrates how to configure module in station mode and receive the data from the remote side using TCP socket	sapis/examples/wlan/ tcp_server/rsi_tcp_server.c
TCP client	This example demonstrates how to configure module in station mode and send data to the remote side using TCP client socket	sapis/examples/wlan/tcp_client/ rsi_tcp_client.c
Enterprise mode connectivity	This example demonstrates how to connect the module to enterprise security enabled access point	sapis/examples/wlan/eap/ rsi_eap_connectivity.c
ssl client	This example application demonstrates how to send data using TCP client socket over SSL in station mode.	sapis/examples/wlan/ssl_client/ rsi_ssl_client.c
Access point creation	This example demonstrates how to Configure module in access point mode and receive the data from the remote side using TCP server socket.	sapis/examples/wlan/ access_point/rsi_ap_start.c
ap udp echo	This example demonstrates how to Configure module in access point mode and echo the udp data sent by the remote side connected client device	sapis/examples/wlan/ ap_udp_echo/ rsi_ap_udp_echo.c

Example	Description	Application path
http client	This example application demonstrates how HTTP client is able to request a page and post the data to simple HTTP server	sapis/examples/wlan/ http_client/ rsi_http_client_app.c
smtp client	This example application demonstrates how SMTP client is able to send a mail to simple SMTP mail server	sapis/examples/wlan/ smtp_client/ rsi_smtp_client_app.c
ftp client	This example application demonstrates how file content is read from the ftp server and copy this to a new file created	sapis/examples/wlan/ smtp_client/rsi_ftp_client.c
Concurrent mode	This example application demonstrates how concurrent mode is used in allowing Device to act as Access point and Wi-Fi station simultaneously	sapis/examples/wlan/ concurrent_mode/ rsi_concurrent_mode.c
Connected sleep	This example application demonstrates how	sapis/examples/wlan/ connected_sleep/ rsi_wlan_connected_sleep_app.c
Connection using asynchronous apis app	This example application demonstrates how asynchronous apis are to to connect the device to access point	sapis/examples/wlan/ connection_using_asynchronous_apis_app/ rsi_connection_using_asynchronous_apis_app.c
Firmware upgradation	This example application demonstrates how firmware is upgraded to the device remote TCP server socket	sapis/examples/wlan/fwup/ rsi_fwup_app.c
multicast	This example application demonstrates how to data is receive from multicast group	sapis/examples/wlan/multicast/ rsi_multicast_app.c
Power save	This example application demonstrates how power save feature is implemented in the device	sapis/examples/wlan/ power_save/ rsi_wlan_powersave_profile.c

Example	Description	Application path
provisioning	This example application demonstrates how provisioning is used to configure the device to join to an AP.	sapis/examples/wlan/ provisioning/ rsi_provisioning_app.c
Station ping	This example application demonstrates how ping to ping the remote peer	sapis/examples/wlan/ station_ping/rsi_station_ping.c
Transmit test	This example application demonstrates how FCC certification test is run	sapis/examples/wlan/ transmit_test/ rsi_transmit_test_app.c
Websocket client	This example application demonstrates how to create a web socket client and send data	sapis/examples/wlan/ websocket_client/ rsi_websocket_client_app.c
Wi-Fi direct	This example application demonstrates how to create a Wi-Fi direct client and send data	sapis/examples/wlan/Wi-Fi_direct/rsi_wfd_client.c
Wps station	This example application demonstrates how to connect to a WPS supported AP using push bu tton method	sapis/examples/wlan/ wps_station/rsi_wps_station.c

WLAN Example Applications

13.1.2 BLE Example Applications

Example	Description	Application path
simple_peripheral	This example demonstrates simple BLE peripheral mode	sapis/examples/ble/ simple_peripheral/ rsi_ble_peripheral.c
simple_central	This example demonstrates simple BLE central mode	sapis/examples/ble/simple_central/ rsi_ble_central.c
simple_chat	This example demonstrates simple data exchange (loopback) b/w module and peer device.	sapis/examples/ble/simple_chat/ rsi_ble_simple_chat.c
immediate_alert_client	This example demonstrates GATT client role for immediate alert service.	sapis/examples/ble/ immediate_alert_client/ rsi_ble_immediate_alert_client.c

BLE Example Applications

13.1.3 Coexistence Example Applications

Example	Description	Application path
Simple chat using BLE and WLAN station applications	This example demonstrates the data exchanges between BLE and WLAN applications.	sapis/examples/wlan_ble/wlan_station_ble_bridge/
Simple chat using BLE and WLAN access point applications	This example demonstrates the data exchanges between BLE and WLAN applications.	sapis/examples/wlan_ble/wlan_ap_ble_bridge/
Simple chat using BLE and WLAN access point applications in tcpip bypass mode	This example demonstrates the data exchanges between BLE and WLAN applications	sapis/examples/wlan_ble/wlan_ap_ble_bridge_tcpipbypass/
Simple chat using BT and WLAN station applications	This example demonstrates the data exchanges between BT and WLAN applications	sapis/examples/wlan_bt/wlan_bt_bridge/
Simple chat using BT and WLAN station applications in tcpip bypass mode	This example demonstrates the data exchanges between BT and WLAN applications	sapis/examples/wlan_bt/wlan_bt_bridge_tcpipbypass/
Simple chat using BT and WLAN access point applications in tcpip bypass mode	This example demonstrates the data exchanges between BT and WLAN applications	sapis/examples/wlan_bt/wlan_ap_bt_bridge_tcpipbypass/
Coex power save and Simple chat using BT and WLAN station applications in tcpip bypass mode	This example demonstrates the data exchanges between BT and WLAN applications with power save	sapis/examples/wlan_bt/power_save /

Coexistence Example Applications

13.2 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

Error Codes (in hexadecimal format)	Description
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	Wi-Fi Direct or EAP configuration is not done
0x0015	Memory allocation failed or Store configuration check sum failed
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)
0x002A	Group Owner negotiation failed in Wi-Fi Direct mode

Error Codes (in hexadecimal format)	Description
0x002B	Association between nodes failed in Wi-Fi Direct mode/ 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
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 34 bytes)
0x003C	Band not supported
0x003D	User store configuration invalid length
0x003E	Error in length of the command(Excceds 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 misscount reaches max beacon miss count(Deauth due to beacon miss)
0x0045	Deauth received from supplicant
0x0046	Deauth received from AP after channel switching
0x0047	Synchronization missed
0x0048	Authentication timeout occurred
0x0049	Association timeout

Error Codes (in hexadecimal format)	Description
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 has not 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
0x0061	Address family not supported by protocol.
0x0062	Invalid beacon interval provided.
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 webpage data than the total length initially specified
0x00C7	Error in set region command
0x00C8	Webpage current chunk length is incorrect
0x00CA	Error in Ap set region command
0X00CB	Error in AP set region command parameters

Error Codes (in hexadecimal format)	Description
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
0x0104	SNMP invalid IP protocol error
0xBB01	No data received or receive time out
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-existenting TCP server socket
0xBB3E	Error in length of the command(Excceds number of characters is mentioned in the PRM)
0xBB42	Socket is still bound
0xBB45	No free port
0xBB46	Invalid port

Error Codes (in hexadecimal format)	Description
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
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 0xBBA1	Empty DNS server list, SMTP message reply errorNo DNS server was specified
0xBBAC 0xBBA3	SMTP data reply errorDNS query failed
0xBBAD 0xBBA4	SMTP authentication reply errorBad DNS address
0xBBAE 0xBBA8	SMTP server error replyParameter error
0xBBAF 0xBBAB	DNS duplicate entry.Empty DNS server list
0xBBB1 0xBBAF	SMTP oversize server replyDNS duplicate entry
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

Error Codes (in hexadecimal format)	Description
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
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
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
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
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

Error Codes (in hexadecimal format)	Description
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	Invalid input to the DNS Resolution query
0xFF42	DNS response was timed out
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

Error Codes (in hexadecimal format)	Description
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
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

WLAN Error codes

Error rendering macro 'captioneditem' : Plugin license error: EXPIRED

13.3 Bluetooth Generic Error Codes

Error Code	Description
0x103	Command timeout
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

Error Code	Description
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
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

Error Code	Description
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
0X4E3C	Directed Advertising Timeout
0X4E60	Invalid Handle Range

Bluetooth Generic Error Codes

Error rendering macro 'captioneditem' : Plugin license error: EXPIRED

13.4 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

Error Code	Description
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
0x4FF8	ERR_INVALID_COMMAND
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

Error Code	Description
0x4D08	Profile record full
0x4D09	Attribute record full
0x4D0A	BLE profile not found (profile handler invalid)

BLE Mode Error Codes

14 Revision History

Revision No.	Version No.	Date	Changes
1	v1.0	November 2017	Advance version
2	v1.2	June 2018	1. Added rsi_wlan_filter_broadcast API description 2. Added command type for multicast 3. Clarified the differences between WiSeConnect/WiSeMCU
3	v1.3	June 2018	Alignment issues are resolved
4	v1.4	July 2018	Structural issues are resolved
6	v1.5	October 2018	1. Added coex mode table 2. Added WLAN network callback handlers. 3. Modified http_client_put_response_handler() callback info in rsi_http_client_put_start() api 4. Modified the post_data_length in rsi_http_client_post_async() API from 16bit to 32bit 5. Added rsi_http_client_async() Information 6. Added HTTP_client_put_pkt server response structure information
7	v1.6	January 2018	1. Added rsi_ble_conn_params_update API description in BLE API's 2. Added rsi_set_intr_type API in common API section 3. Added default values and a note for rsi_feature_frame API
8	v1.7	April 2019	1. Added description of rsi_set_intr_type API 2. Zigbee deleted 3. Modified rsi_wireless_antenna api 4. Corrected rsi_wlan_init API to rsi_wireless_init 5. Added gain table user configurable API information 6. Added Note for rsi_socket API

Disclaimer

The information in this document pertains to information related to Redpine Signals, Inc. products. This information is provided as a service to our customers, and may be used for information purposes only.

Redpine assumes no liabilities or responsibilities for errors or omissions in this document. This document may be changed at any time at Redpine's sole discretion without any prior notice to anyone. Redpine is not committed to updating this document in the future.