RS14100_RS9116 Wireless SAPI Manual

Version 1.5

October 2018

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

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.

Table of Contents

About this Document

This document is an Advance version of Redpine Wireless SAPI Guide provided to select customers under a Non-Disclosure Agreement (NDA).

1 Overview of Wireless SAPI

RS9116 WiSeConnect and R14100 WiSeMCU products include Wi-Fi, TCP/IP, BT 5 and ZigBee PRO 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)

2.3 ZigBee

- Supports switch of Zigbee home automation.
- Supports test harness to test HA, HA1.2,ZLL,SE profiles
- Supports Coordinator, Router, End device modes

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/*

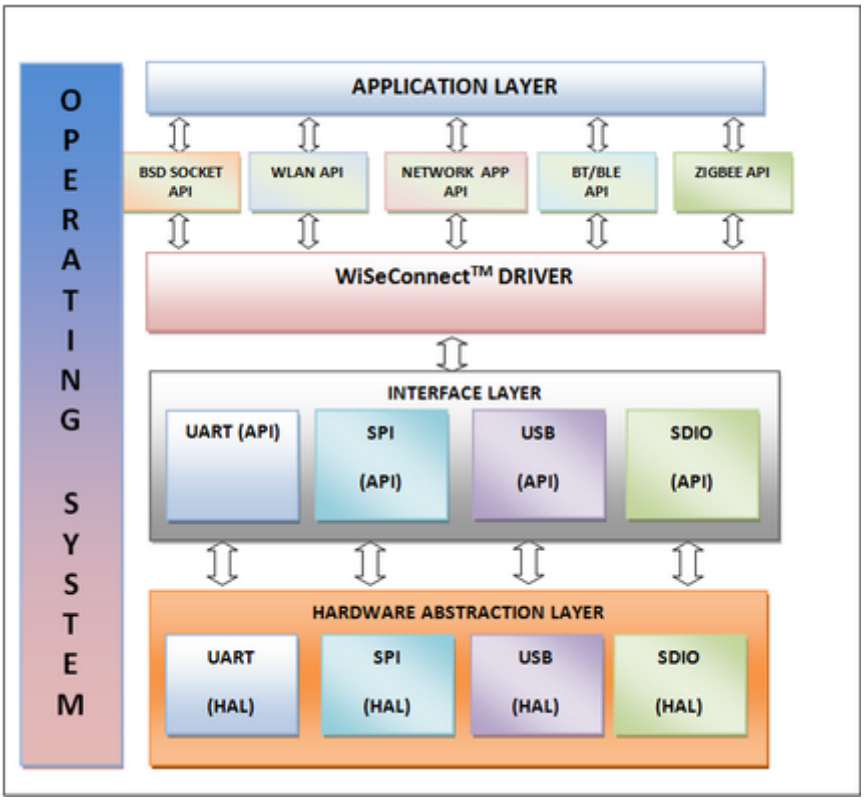
3.2 RS9116 Connectivity Resources

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.



WiSeConnect API Architecture Diagram

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.

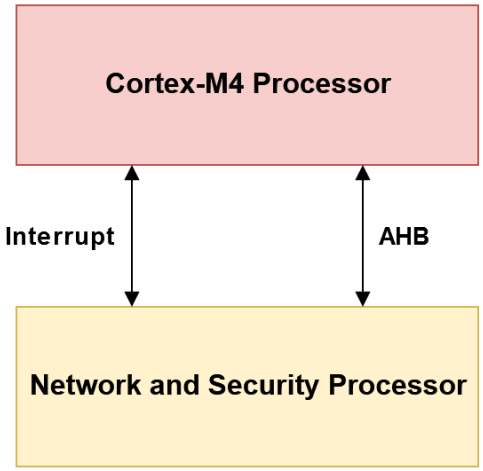
Interface Specific API Layer
The module supports 4 different host interfaces (SPI, USB, UART, USB-CDC). These APIs are collection of functions specific to a particular interface. The interface functions between the Driver API Layer and the Interface Specific API Layer are independent of the Host interface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



WiseMCU Architecture Diagram

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
0	Successful execution of the command
Non Zero Value	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_wlan_init(buffer, BUFFER_LENGTH);
```

5.2 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;
 }
```

5.3 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;
 }
```

5.4 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;
 }
```

5.5 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);
```

5.6 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

Parameter	Description
coex_mode	Coexistence mode

Coex Mode	Description
0	WLAN only mode
1	WLAN
2	ZigBee
3	WLAN + ZigBee
4	Bluetooth
5	WLAN + Bluetooth
6	Bluetooth + Zigbee co-existence*
7	WLAN + Bluetooth + ZigBee co-existence*
8	Dual Mode (Bluetooth and BLE)
9	WLAN + Dual Mode
10	Dual Mode + ZigBee co-existence*
11	WLAN + Dual Mode + ZigBee co-existence*
12	BLE mode
13	WLAN + BLE
14	BLE and ZigBee co-existence*
15	WLAN + BLE + ZigBee co-existence*

* Will be supported in future releases

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
 }
```

5.7 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();
```

5.8 rsi_wireless_driver_task

Prototype

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

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

Precondition
N/A

Parameters
None

Return Values
None

Example

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

5.9 rsi_wireless_antenna

Prototype

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

Description
This API configures the antenna.

Precondition
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

Note : Currently ignoring the gain_2g and gain_5g 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);
```

5.9.1 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
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               0
#define WIRELESS_MODE         0
#define ENABLE_PPP            0
#define AFE_TYPE              1
#define FEATURE_ENABLES       0
```

Parameter

Value	Description
PLL_MODE	0-PLLMODE0 - Used for generating the clocks for 20Mhz Bandwidth operations. 1-PLLMODE1-Used for generating the clocks for 40Mhz Bandwidth operations. 2-PLLMODE2-Reserved
RF_TYPE	0-External_RF_8111, 1-Internal_RF_9116,
WIRELESS_MODE:	12 – To enable LP chain for PER mode 0- LP chain disable
ENABLE_PPP	0-Disable_per_packet_TX_programming, 1-Enable_per_packet_TX_programming_mode_1, 2-Enable_per_packet_TX_programming_mode_2
AFE	0 - AFE BYPASS, 1 - Internal AFE
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.

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
153	153
157	157
161	161

Channel Number	chno
165	165

Channels supported in 5GHz band

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
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

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

2.4GHz Band Channel Mapping

Channels supported in 5GHz band:

Channel Number	chno
All channels	0
36	36

Channel Number	chno
40	40
44	44
48	48
100	100
104	104
108	108
112	112
116	116
132	132
136	136
140	140
149	149
153	153
157	157
161	161
165	165

5GHz Band Channel Mapping

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
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.

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.

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.
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

Security type (sec_type)	secret key structure format (secret_key)
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
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

```
//! 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.
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.

Security type (sec_type)	secret key structure format (secret_key)
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.
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
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

Parameters

Parameter

	Description
type	possible values of this field are 1, 2 and 3,but we only pass 3 for generation of PMK.

	Description
psk	In this field expected parameters are pre shared key(psk) of the access point to which module wants to associate
ssid	This field contains the SSID of the access point, this field will be valid only if TYPE value is 3
pmk	this is an array
length	length of pmk array

Return Values

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

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

Example

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

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_wlan_init() API must be called before this API.

Parameters

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	Failure **If return value is 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));
```

6.1.9 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();
```

6.1.10 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_wlan_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

Parameter	Description
security_type	This is the type of the security modes on which an access point needs to be operated 0: RSI_OPEN 1: RSI_WPA 2: RSI_WPA2 6: RSI_WPA_WPA2_MIXED 11: RSI_WPS_PUSH_BUTTON
encryption_mode	This is the type of the encryption mode 0: RSI_NONE 1: RSI_TKIP 2: RSI_CCMP
password	This is the PSK to be used in security mode
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.

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
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.

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.

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
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.

Parameter	Description
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

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 **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.
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

Value	Description
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.

Note
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.

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

Return Values

Value	Description
0	Successful execution of the command

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

6.1.21 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
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

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

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

6.1.22 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

6.1.23 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.

6.1.24 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.

6.1.25 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

Value	Description
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);
```

6.1.26 rsi_transmit_test_start

Prototype

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

Description
This API starts the transmit test.

> **Note**
> This API is relevant in 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.

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

Note
Before starting Continuous Wave mode, it is required to start Burst mode with power and channel values which is intended to be used in Continuous Wave mode
i.e 1. Start Burst mode with intended power value and channel values
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

Data Rate (Mbps)	Value of rate
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

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

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

Channel Numbers (5GHz)	Center frequencies for 20MHz channel width (MHz)
36	5180
40	5200
44	5220

Channel Numbers (5GHz)	Center frequencies for 20MHz channel width (MHz)
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);
```

6.1.27 rsi_transmit_test_stop

Prototype

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

Description
This API stops the transmit test.

Note
This API is relevant in 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.

6.1.28 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();
```

6.1.29 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);
```

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);
```

6.1.30 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.
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.

Call back id	Function Description
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
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);
```

6.1.31 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
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

Call back id	Function prototype	Parameters	Function Description
RSI_WLAN_NWK_FW_UPGRADE_CB	void (*rsi_wireless_fw_upgrade_handler)(uint8_t type, uint32_t status);	type :type of the handler. status :status of the response.	This callback is used to Register wireless firmware upgrade
RSI_WLAN_NWK_JSON_EVENT_CB	void (*rsi_json_object_event_handler)(uint32_t status, uint8_t *json_object_str, uint32_t length);	status :status of the response. json_object_str :json object string. length :length of the string.	This callback is used to Register json update

Return Values

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

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

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

Note
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
protocol	0: non SSL sockets 1: SSL sockets

Note
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);
```

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.

Note
Bind command is mandatory to call after socket create command.

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

Parameters

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

Return Values

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

Example

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

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
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, struct rsi_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

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

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

Example

```
new_socket = int32_t  rsi_accept_async(int32_t sockID, struct rsi_sockaddr
*ClientAddress, int32_t *addressLength)
```

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 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
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);
```

6.2.13 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

Note
IPv6 is not supported

Return Values

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

6.2.14 rsi_dns_req

Prototype

```
int32_t rsi_dns_req(
uint8_t ip_version,
uint8_t *url_name,
uint8_t *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

Parameter	Description
server_address	This is the IP address of the DNS server. This parameter is optional if module get DNS server address using DHCP.
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.

Note
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.

6.2.15 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.

Note
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

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;
 }
```

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
smtp_client_mail_response_handler	This is the callback when asynchronous response comes from the sent mail The parameters involved are : status and cmd status: status code cmd: sub command type

Note
If the status in callback is nonzero, then the sub command type will be in 6th byte of the descriptor.

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -3: Command given in wrong state -4: Buffer not available to serve the command **If return value is greater than 0** 0x0021,0x002C,0x0015,0x003E, 0xBBA5,0xBBA3,0xBBA0,0xBBA1,0xBBA2,OxBBA4,OxBBA6,0xBBA7,0xBBA8,0xBBA9,0xBBAA, 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;
 }
```

rsi_smtp_client_delete_async

Prototype

```
int32_t rsi_smtp_client_delete_async(
void(*smtp_client_mail_response_handler)
(uint16_t status,
const uint8_t cmd));
```

Description
This API deletes the smtp client.

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

Parameters

Parameter	Description
smtp_client_delete_response_handler	This is the callback when asynchronous response comes for the delete request The parameters involved are: status & cmd status:This is the status code cmd: This is the sub command type

Note
If the status in callback is nonzero, then the sub command type will be in 6th byte of the descriptor.

Return Values

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

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

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
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);
```

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
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);
```

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
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);
```

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);
```

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

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

Parameter	Description
password	This is the password for server Authentication
http_client_get_response_handler	This is the callback when asynchronous response comes for the request The parameters involved are: status , buffer, length & more_data status:This is the status 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);
```

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);
```

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);
```

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();
```

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);
```

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;
```

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();
```

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

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

6.3.6 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;
  }
```

6.3.7 POP3 client API

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

Note
If status in callback is nonzero, then sub command type comes in 6^{th} 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;
 }
```

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

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** 0x0021,0xFF74,0xBB87

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;
}
```

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);
```

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);
```

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);
```

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.

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();
```

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();
```

6.3.8 FTP Client API

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

Parameter	Description
server_ip	This is the FTP server IP address to connect
username	This is the username for server authentication
password	This is the password for server authentication
server_port	This is the port number of FTP server **Note:** FTP server port is configurable on 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;
 }
```

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;
 }
```

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);
```

rsi_ftp_file_write_content

Prototype

```
int32_t rsi_ftp_file_write_content(uint16_t flags, int8_t
*file_content,int16_t content_length,uint8_t end_of_file)
```

Description
This function writes the content into the file which is opened using rsi_ftp_file_write() API.

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

Parameters

Parameter	Description
flags	These are the network flags. Each bit in the flag has its own significance BIT(0) – RSI_IPV6 Set this bit to enable IPv6. By default, it is configured to IPv4 BIT(1) to BIT(15) are reserved for future use
file_content	This is the data stream to be written into the file
content_length	This is the file content length
end_of_file	This flag indicates the end of file 1 – This chunk represents the end of content to be written into the file 0 – This are the extra data which is pending to write into the file **Note:** This API can be called multiple times to append data into the same file and at the last chunk ,this flag should be 1.

Return Values

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

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

Note
File content length should not exceed 1344 bytes in case of IPV4 and 1324 bytes in case of IPV6.If exceeds, this API will break the file content and send it in multiple packets

Example

```
 //! File Write Content
 retval = rsi_ftp_file_write_content(0,
file_data_read,file_data_length,1);
```

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);
```

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.

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.

rsi_ftp_directory_create

Prototype

```
int32_t rsi_ftp_directory_create(int8_t *directory_name)
```

Description
This API creates a directory on the FTP server.

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

Parameters

Parameter	Description
directory_name	This is the directory name (with path if required) to create e.g "example" or "/test/ftp/example"

Return Values

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

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

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.

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.

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.

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);
```

6.3.9 MQTT Client API

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);
```

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);
```

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);
```

rsi_mqtt_publish

Prototype

```
int32_t rsi_mqtt_publish(rsi_mqtt_client_info_t *rsi_mqtt_client,
int8_t *topic, MQTTMessage *publish_msg)
```

Description
This API publishes the message on the topic specified.

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

Parameters

Parameter	Description
rsi_mqtt_client	This is the MQTT client info structure pointer

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid Parameter

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);
```

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

Parameter	Description
call_back_handler_ptr	This is the callback function when asynchronous data comes on the subscribed data. MessageData* md Message data pointer received

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	**If return value is lesser than 0** -2: Invalid Parameter

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);
```

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);
```

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);
```

6.3.10 HTTP Server API

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.
```

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);
```

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));
```

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);
```

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.

6.3.11 DHCP User class (Option-77) API

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
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);
```

6.3.12 Multicast API

rsi_multicast_join

Prototype

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

Description
This API joins to a multicast group.

Note
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);
```

rsi_multicast_leave

Prototype

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

Description
This API is used to leave the multicast group.

Note
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);
```

6.3.13 MDNSD API

rsi_mdnsd_init

Prototype

```
int32_t rsi_mdnsd_init(uint8_t ip_version, uint16_t ttl, uint8_t
*host_name);
```

Description
This API initializes the MDNSD service in WiSeConnect Device. It creates MDNS daemon.

Note
1. Currently registering only one service is supported
2. IPv4 is only supported for MDNS/DNS-SD service

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

Parameters

Parameter	Description
ip_version	To select the IP version. 4 – To select IPv4 6 – To select IPv6
ttl	This is the time to live. THis is the time in seconds for which service should be active
host_name	This is the host name which is used as host name in Type A record.

Return Values

Value	Description
0	Successful execution of the command
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);
```

rsi_mdnsd_register_service

Prototype

```
int32_t rsi_mdnsd_register_service(uint16_t port,
uint16_t ttl,
uint8_t more,
uint8_t *service_ptr_name,
uint8_t *service_name,
uint8_t *service_text);
```

Description
This API is used to add a service / start service discovery.

Note
Currently registering only one service is supported

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

Parameters

Parameter	Description
port	This is the port number on which service should be added.
ttl	This is the time to live. This is the time in seconds for which service should be active
more	This byte should be set to '1' when there are more services to add. 0 – This is last service, starts MDNS service. 1 – Still more services will be added.
service_ptr_name	This is the name to be added in Type-PTR record
service_name	This is the name to be added in Type-SRV record(Service name)
service_text	This is the text field to be added in Type-TXT record

Return Values

Value	Description
0	Successful execution of the command
Non Zero	**If return value is lesser than 0** -4: Buffer not available to serve the command -6: Data size exceeded **If return value is greater than 0** 0x0021, 0x0015, 0x0074

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);
```

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.

6.3.14 Socket configuration API

rsi_socket_config

Prototype

```
int32_t rsi_socket_config()
```

Description
This command is used to set the socket configuration parameters. User is recommended to use this command(optional). Based on the socket configuration, module will use available buffers effectively. This command should be given after IP configuration command and before any socket creation.

Parameters

Parameter	Description
total socket	Desired total number of sockets to open.
total_tcp_sockets	Desired total number of TCP sockets to open
total_udp_sockets	Desired total number of UDP sockets to open.
tcp_tx_only_sockets	Desired total number of TCP sockets to open which are used only for data transmission.
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

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();
```

6.3.15 Web socket API

rsi_web_socket_create

Prototype

```
int32_t rsi_web_socket_create(int8_t flags,
uint8_t *server_ip_addr,
uint16_t server_port,
uint16_t device_port,
uint8_t *webs_resource_name,
uint8_t *webs_host_name,
int32_t *socket_id,
void (*web_socket_data_receive_notify_callback)
(uint32_t sock_no,
uint8_t *buffer,
uint32_t length));
```

Description
This API creates a web socket client .

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

Parameters

Parameter	Description
flags	To select IP version and security BIT(0) – RSI_IPV6 Set this bit to enable IPv6 , by default it is configured to IPv4 BIT(1) – RSI_SSL_ENABLE Set this bit to enable SSL feature
server_ip_addr	This is the web server ip address
server_port	This is the web server socket port.
device_port	This is the local port
webs_resource_name	This is the web resource name **Note:** string of 50 characters maximum

Parameter	Description
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);
```

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));
```

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);
```

6.3.16 OTAF client API

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;
 }
```

6.3.17 PUF API

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();
```

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 );
```

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();
```

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
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 );
```

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

Value	Description
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();
```

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);
```

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 );
```

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)

Parameter	Description
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
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);
```

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
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 );
```

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
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.

Define	Meaning
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

Define	Meaning
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
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
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
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

6.4.14 Configure Transmit test mode parameters

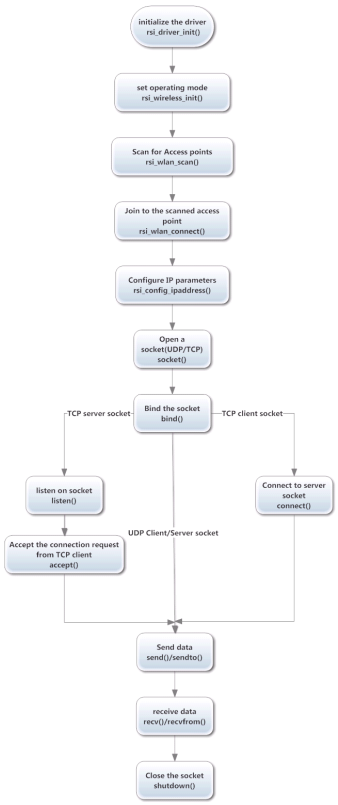
Define	Meaning
RSI_TX_TEST_RATE_FLAGS	Rate flags contain short GI, Greenfield and channel width values To enable short GI – set rate flags value as '1' To enable Greenfield – set rate flags value as '2'
RSI_TX_TEST_PER_CH_BW	Channel width should be set to zero to set 20MHz channel width.
RSI_TX_TEST_AGGR_ENABLE	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.

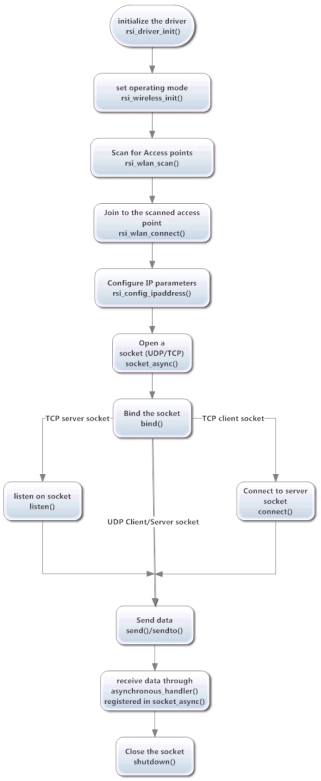
6.4.15 Station mode

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

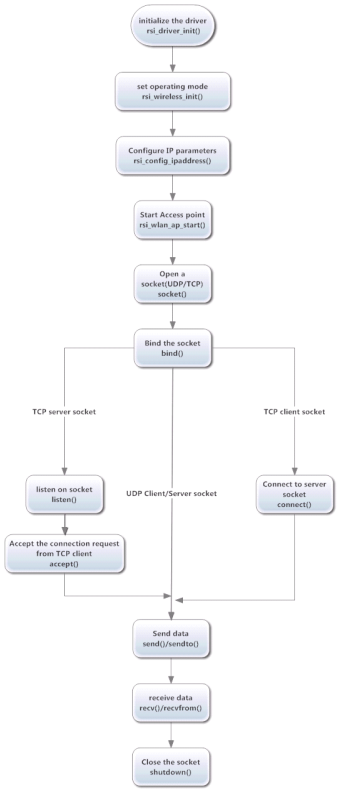
The following example illustrate the data transfer using rsi_socket_async() API



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
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.

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.

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.

Note
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 .

Note
Before starting Continuous Wave mode, it is required to start Burst mode with power and channel values which is intended to be used in Continuous Wave mode
i.e 1. Start Burst mode with intended power value and channel values
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

Data Rate (Mbps)	Value of rate
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

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

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

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	Failure **If return value is 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.

> **Note**
> 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.

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
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);
```

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_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.
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.

Call back id	Function Description
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
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);
```

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

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 ULP GPIO #0. After wakeup, if the module is ready for data transfer, it sends wakeup indication to host by setting ULP GPIO #1. 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 ULP GPIO #0. After recognizing sleep permission from host, module gives confirmation to host by resetting ULP GPIO #1 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
"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 ULP-GPIO #0 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
Limitation of BT APIs - A new BT API has to be called only after getting the response for the previous API.

9.1 GAP API

This section explains the GAP APIs.

9.1.1 rsi_bt_get_local_name

Prototype

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

Description

This API requests the local device name.

Precondition

rsi_wireless_init() API need to be called before this API

Structure

```
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 Variables

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

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.

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.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
static 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);
```

9.1.2 rsi_bt_set_local_name

Prototype

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

Description

This API sets the local device name.

Precondition

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

Parameters

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define RSI_BLE_LOCAL_NAME "WYZBEE_PERIPHERAL"
 status = rsi_bt_set_local_name(RSI_BLE_LOCAL_NAME);
```

9.1.3 rsi_bt_set_local_class_of_device

Prototype

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

Description

This API requests the local COD name.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
class_of_device	This is the class of device

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

class_of_device=284000

status = rsi_bt_set_local_class_of_device(class_of_device)

9.1.4 rsi_bt_get_local_class_of_device

Prototype

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

Description

This API requests the local COD name.

Precondition

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

Parameters

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_get_local_class_of_device(&class_of_device)

9.1.5 rsi_bt_start_discoverable

Prototype

```
int32_t rsi_bt_start_discoverable(void)
```

Description

This API requests the local device to enter discovery mode.

Precondition

rsi_wireless_init API needs to be called before this API.

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
status = rsi_bt_start_discoverable();
```

9.1.6 rsi_bt_start_limited_discoverable

Prototype

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

Description

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

Precondition

rsi_wireless_init API needs to be called before this API.

Parameters

Parameter	Description
time_out_ms	Limited discovery mode time_out in ms.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status=rsi_bt_start_limited_discoverable(100)

9.1.7 rsi_bt_stop_discoverable

Prototype

```
int32_t rsi_bt_stop_discoverable(void)
```

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

Parameters
rsi_wireless_init API needs to be called before this API.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

rsi_bt_get_discoverable_status(&device_state)

9.1.8 rsi_bt_get_discoverable_status

Prototype

int32_t rsi_bt_get_discoverable_status(uint8_t *resp)

Description

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

Precondition

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

Parameters

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

rsi_bt_get_discoverable_status(&device_state)

9.1.9 rsi_bt_set_connectable

Prototype

```
int32_t rsi_bt_set_connectable(void)
```

Description

This API requests the local device to set the connectivity 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

Value	Description
Non Zero	Failure

Example

```
status = rsi_bt_start_discoverable();
 status = rsi_bt_set_connectable();
```

9.1.10 rsi_bt_set_non_connectable

Prototype

```
int32_t rsi_bt_set_non_connectable(void)
```

Description

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

Precondition

rsi_wireless_init API need to be called before this API

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status=rsi_bt_get_non_connectable()

9.1.11 rsi_bt_get_connectable_status

Prototype

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

Description

This API gets the BT Module connectivity status.

Precondition

rsi_bt_set_connectable() API needs to be called before this API

Parameters

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_get_connectable_status(&device_state)

9.1.12 rsi_bt_remote_name_request_async

Prototype

```
int32_t rsi_bt_remote_name_request_async(int8_t *remote_dev_addr,
rsi_bt_event_remote_device_name_t *bt_event_remote_device_name)
```

Description

This API knows the remote device name.

Precondition

rsi_wireless_init() API need to be called before this API

Structure

```
typedef struct rsi_bt_event_remote_device_name_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint8_t name_length;
 INT08 remote_device_name[RSI_BT_DEVICE_NAME_LEN];
 } rsi_bt_event_remote_device_name_t;
```

Structure Variables

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

Variables	Description
dev_addr	This is the BD address of remote device. This is an array of max length - 6.
Name_length	This is the length of remote device name.
Remote_device_name	This is the name of remote device.

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 parameter of rsi_bt_event_remote_device_name_s

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_remote_name_request_async(remote_dev_addr , &remote_dev_name)

9.1.13 rsi_bt_remote_name_request_cancel

Prototype

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

Description

This API is used cancel the remote device name request.

Precondition

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

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

rsi_bt_remote_name_request_cancel(remote_dev_addr)

9.1.14 rsi_bt_inquiry

Prototype

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

Description

This API starts the inquiry.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
inquiry_type	This is the Inquiry type.
inquiry_duration	This is the duration of inquiry
max_devices	This is the maximum number of devices allowed to inquiry

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_inquiry(0,500,0x02)

9.1.15 rsi_bt_cancel_inquiry

Prototype

```
int32_t rsi_bt_cancel_inquiry(void)
```

Description

This API cancels the inquiry.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_cancel_inquiry()

9.1.16 rsi_bt_set_eir_data

Prototype

```
int32_t rsi_bt_set_eir_data(int8_t *data , uint16_t data_len)
```

Description

This API sets the extended Inquiry Response data.

Precondition

rsi_wireless_init() API needs to be called before this API

Parameters

Parameter	Description
fec_required	This is the FEC required
data_length	This is the length of the data
eir_data	EIR data is an array which can stores data upto 200 Bytes.

Note:
Currently EIR data supports upto 200 bytes.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define RSI_BT_LOCAL_NAME "SPP_SLAVE"
 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
 rsi_bt_set_eir_data (eir_data,
 strlen (RSI_BT_LOCAL_NAME) + 5);
```

9.1.17 rsi_bt_connect

Prototype

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

Description

This API initiates the connection request

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define REMOTE_BD_ADDR "00:1B:DC:07:2C:F0"
 //! 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);
```

9.1.18 rsi_bt_cancel_connect

Prototype

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

Description

This API cancels the connection request.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
remote_dev_address	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_cancel_connect(remote_dev_addr)

9.1.19 rsi_bt_disconnect

Prototype

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

Description

This API is used to disconnect the physical connection.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
remote_dev_address	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_disconnect(remote_dev_addr)

9.1.20 rsi_bt_set_ssp_mode

Prototype

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

Description

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

Precondition

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

Parameters

Parameter	Description
pair_mode	This parameter is used to enable or disable SSP mode. This parameters supports the following values. 0 - Disable 1 - Enable
IOcapability	This is the IO capability request for SSP mode. This parameter supports the following values. 0 – DisplayOnly 1 – DisplayYesNo 2 – KeyboardOnly 3 - NoInputNoOutput

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
//! 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);
```

9.1.21 rsi_bt_accept_ssp_confirm

Prototype

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

Description

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

Precondition

rsi_bt_spp_connect() API need to be called before this API

Parameters

Parameter	Description
remote_dev_address	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
rsi_bt_accept_ssp_confirm(str_conn_bd_addr);
```

9.1.22 rsi_bt_reject_ssp_confirm

Prototype

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

Description

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

Precondition

rsi_bt_spp_connect() API need to be called before this API

Parameters

Parameter	Description
remote_dev_address	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_reject_ssp_confirm(remote_dev_addr);

9.1.23 rsi_bt_passkey

Prototype

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

Description

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

Precondition

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

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address
passkey	This is the passkey input
reply_type	This is the positive or negative reply

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_passkey(remote_dev_addr,123456,1)

9.1.24 rsi_bt_pincode_request_reply

Prototype

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

Description

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

Precondition

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

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address
pin_code	This is the pincode input
reply_type	This is the positive or negative reply

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

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

9.1.25 rsi_bt_linkkey_request_reply

Prototype

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

Description

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

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

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

9.1.26 rsi_bt_get_local_device_role

Prototype

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

Description

This API requests the role of a local device.

Precondition

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

Parameters

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_get_local_device_role(rsi_app_resp_get_bond_dev_addr, &device_state);

9.1.27 rsi_bt_sniff_mode

Prototype

```
int32_t rsi_bt_sniff_mode(uint8_t *remote_dev_addr, uint16_t
sniff_max_intv, uint16_t sniff_min_intv, uint16_t sniff_attempt,
uint16_t sniff_tout)
```

Description

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

Precondition

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

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address
sniff_max_intv	This is the sniff maximum interval
sniff_min_intv	This is the sniff minimum interval
sniff_attempt	This is the sniff attempt
sniff_tout	This is the sniff timeout

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
/* here we call the sniff_mode command*/
 status =
rsi_bt_sniff_mode(str_conn_bd_addr,SNIFF_MAX_INTERVAL,SNIFF_MIN_INTER
VAL,SNIFF_ATTEMPT,SNIFF_TIME_OUT);
```

9.1.28 rsi_bt_sniff_exit_mode

Prototype

```
int32_t rsi_bt_sniff_exit_mode(uint8_t remote_dev_addr) *Description
 This API is used to request the local device to exit from sniff/
sniff subrating mode.
```

Precondition

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

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
rsi_bt_sniff_exit_mode(str_conn_bd_addr);
```

9.1.29 rsi_bt_sniff_subrating_mode

Prototype

```
int32_t rsi_bt_sniff_subrating_mode(uint8_t *remote_dev_addr,
uint16_t
max_latency, uint16_t min_remote_tout, uint16_t min_local_tout)
```

Description

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

Parameters

Parameter	Description
remote_dev_addr	Remote device address
max_latency	Maximum latency
min_remote_tout	Minimum remote timeout
min_local_tout	Minimum local timeout

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_sniff_subrating_mode(remote_dev_addr,192,1000,1000)

9.1.30 rsi_bt_get_rssi

Prototype

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

Description

This API requests the RSSI of the remote device.

Precondition

rsi_ble_connect() API need to be called before this API

Parameters

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

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

9.1.31 rsi_bt_get_local_device_address

Prototype

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

Description

This API requests the local device address.

Precondition

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

Parameters

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
//! get the local device address(MAC address).
 status =
 rsi_bt_get_local_device_address(rsi_app_resp_get_dev_addr);
```

9.1.32 rsi_bt_spp_init

Prototype

```
int32_t rsi_bt_spp_init(void)
```

Description

This API sets the SPP profile mode.

Precondition

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

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
//! 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 ();
```

9.1.33 rsi_bt_spp_connect

Prototype

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

Description

This API initiates the SPP profile level connection.

Precondition

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

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_spp_connect (REMOTE_BD_ADDR);

9.1.34 rsi_bt_spp_disconnect

Prototype

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

Description

This API initiates the SPP service level disconnection.

Precondition

rsi_bt_spp_connect() API need to be called before this API

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_spp_disconnect (&remote_dev_addr)

9.1.35 rsi_bt_spp_transfer

Prototype

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

Description

This API transfers the data through SPP profile.

Precondition

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

Parameters

Parameter	Description
remote_dev_addr	This is the remote device address
Data	This is the data for transmission
Length	This is the data length for transfer

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
//! SPP data transfer (loop back)
 rsi_bt_spp_transfer (str_conn_bd_addr, data, data_len);
```

9.1.36 rsi_bt_a2dp_init

Prototype

```
int32_t rsi_bt_a2dp_init(void)
```

Description

This API sets the A2DP 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

```
//! 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();
```

9.1.37 rsi_bt_a2dp_connect

Prototype

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

Description

This API initiates the A2DP profile level connection.

Precondition

`rsi_bt_a2dp_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

status = rsi_bt_a2dp_connect(REMOTE_BD_ADDR);

9.1.38 rsi_bt_a2dp_disconnect

Prototype

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

Description

This API initiates the A2DP service level disconnection.

Precondition

rsi_bt_a2dp_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

Value	Description
Non Zero	Failure

Example

status = rsi_bt_a2dp_disconnect(&remote_dev_addr)

9.1.39 rsi_bt_a2dp_send_pcm_data

Prototype

```
int32_t rsi_bt_a2dp_send_pcm_data(uint8_t *remote_dev_addr, uint8_t
*pcm_data, uint16_t pcm_data_len)
```

Description

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

Precondition

rsi_bt_a2dp_connect() API need to be called before this API

Parameters

Parameter	Description
remote_dev_addr	Remote device address
`pcm_data`	Audio data
`pcm_data_len`	length of the pcm data

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_a2dp_send_pcm_data (str_conn_bd_addr, pcm_data, size);

9.1.40 rsi_bt_a2dp_send_sbc_data

Prototype

```
int32_t rsi_bt_a2dp_send_sbc_data(uint8_t *remote_dev_addr, uint8_t
*sbc_data, uint16_t sbc_data_len)
```

Description

This API will send the sbc data to BT stack from M4, after receiving pcm data.

Precondition

rsi_bt_a2dp_send_pcm_data() API need to be called in host application before calling this API

Parameters

Parameter	Description
remote_dev_addr	Remote device address
sbc_data	encoded sbc data
sbc_data_len	length of the sbc data

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_a2dp_send_sbc_data (str_conn_bd_addr, sbc_data, sbc_pkt_len);

9.1.41 rsi_bt_init

Prototype

```
int32_t rsi_bt_init(void)
```

Description

This API initializes the BT device.

Precondition

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

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_init();

9.1.42 rsi_bt_deinit

Prototype

```
int32_t rsi_bt_deinit(void)
```

Description

This API deinitializes the BT device.

Precondition

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

Parameters

None

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_deinit

9.1.43 rsi_bt_set_antenna

Prototype

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

Description

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

Precondition

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

Parameters

Parameter	Description
antenna_value	This parameter is used to select either internal or external antenna

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
#define RSI_SEL_ANTENNA RSI_SEL_INTERNAL_ANTENNA
 //! select/set the antenna type(internal/external)
 status = rsi_bt_set_antenna(RSI_SEL_ANTENNA);
```

9.1.44 rsi_bt_power_save_profile

Prototype

```
int32_t rsi_bt_power_save_profile(uint8_t psp_mode,
 uint8_t psp_type)
```

Description

This API selects the power save profile mode for BT / BLE.

Precondition

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

Parameters

Parameter	Description
psp_mode	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 on mode. In this sleep mode, SoC will never turn off, therefore no handshake is required before sending data to the module. BT/BLE doesnot 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.

Parameter	Description
psp_type	Follwing 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.

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

```
//! initiateing power save in BT mode
 status = rsi_bt_power_save_profile(RSI_SLEEP_MODE_2, RSI_MAX_PSP);
```

9.1.45 rsi_bt_set_feature_bitmap

Prototype

```
int32_t rsi_bt_set_feature_bitmap(uint32_t feature_bit_map)
```

Description

This API enables /disable the mentioned features.

Precondition

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

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

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_set_feature_bitmap(1)

9.1.46 rsi_bt_set_antenna_tx_power_level

Prototype

```
int32_t rsi_bt_set_antenna_tx_power_level(uint8_t protocol_mode,
int8_t tx_power)
```

Description

This API enables / disables the mentioned features.

Precondition

rsi_wireless_init() API need to be called before this API

Parameters

Parameter	Description
protocol_mode	1 – BT classic 2 – BT Low Energy
tx_power	Antenna transmit power level

Return Values

Value	Description
0	Successful execution of the command
Non Zero	Failure

Example

status = rsi_bt_set_antenna_tx_power_level(1,19);

9.2 Callback functions

9.2.1 GAP event callbacks description

rsi_bt_on_role_change_t

Prototype

```
typedef void (*rsi_bt_on_role_change_t) (uint16_t resp_status,
rsi_bt_event_role_change_t *role_change_status);
 typedef struct rsi_bt_event_role_change_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint8_t role_change_status;
 } rsi_bt_event_role_change_t;
```

Description

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

Precondition

None

Parameters

Variables	Description
resp_status(out)	This is the response status
dev_addr(out)	This is the remote device address
role_change_status(out)	This is the role change status

rsi_bt_on_connect_t

Prototype

```
typedef void (*rsi_bt_on_connect_t) (uint16_t resp_status,
rsi_bt_event_bond_t *bond_response);
 typedef struct rsi_bt_event_bond_response_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 } rsi_bt_event_bond_t;
```

Description

This callback function is called if a new connection completion is received from the module. This event will be given by the module in the following two scenarios:

- In case of slave mode, when the connection is initiated from the remote device
- In case of Master mode, when the connect command is issued to connect to a remote device

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status whether success or error
dev_addr	This is the remote device address

rsi_bt_on_disconnect_t

Prototype

```
typedef void (*rsi_bt_on_disconnect_t) (uint16_t resp_status,
rsi_bt_event_disconnect_t *bt_disconnect);
 typedef struct rsi_bt_event_disconnect_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 } rsi_bt_event_disconnect_t;
```

Description
This callback is called when disconnection event is raised from the module. This event will be given by the module when either slave or master device issues disconnect command to the other.

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

rsi_bt_on_remote_name_resp_t

Prototype

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

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address
name_length	This is the length of remote device name
remote_device_name	This is the name of the remote device

rsi_bt_on_passkey_display_t

Prototype

```
typedef void (*rsi_bt_on_passkey_display_t) (uint16_t resp_status,
rsi_bt_event_user_passkey_display_t *bt_event_user_passkey_display);
 typedef struct rsi_bt_event_user_passkey_display_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint8_t passkey[4];
 } rsi_bt_event_user_passkey_display_t;
```

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address
passkey	This is the passkey to be displayed on our module side

rsi_bt_on_remote_name_request_cancel_t

Prototype

```
typedef void (*rsi_bt_on_remote_name_request_cancel_t) (uint16_t
resp_status, rsi_bt_event_remote_name_request_cancel_t
*remote_name_request_cancel);
 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;
```

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

rsi_bt_on_confirm_request_t

Prototype

```
typedef void (*rsi_bt_on_confirm_request_t) (uint16_t resp_status,
rsi_bt_event_user_confirmation_request_t *user_confirmation_request);
 typedef struct rsi_bt_event_user_confirmation_request_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 uint8_t confirmation_value[4];
 } rsi_bt_event_user_confirmation_request_t;
```

Description

This callback function is called if the user confirmation request is received from the module. The user has to give rsi_bt_accept_ssp_confirm or rsi_bt_reject_ssp_confirm command upon reception of this event.

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address
confirmation_value	This is the passkey to be confirmed

rsi_bt_on_pincode_request_t

prototype

```
typedef void (*rsi_bt_on_pincode_request_t) (uint16_t resp_status,
rsi_bt_event_user_pincode_request_t *user_pincode_request);
 typedef struct rsi_bt_event_user_pincode_request_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 } rsi_bt_event_user_pincode_request_t;
```

Description

This callback function is called if pincode request is received from the module. User has to give rsi_bt_accept_pincode_request command upon reception of this event.

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

rsi_bt_on_passkey_request_t

Prototype

```
typedef void (*rsi_bt_on_passkey_request_t) (uint16_t resp_status,
rsi_bt_event_user_passkey_request_t *user_passkey_request);
 typedef struct rsi_bt_event_user_passkey_request_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 } rsi_bt_event_user_passkey_request_t;
```

Description

This callback function is called if the passkey request is received from the module. User has to give rsi_bt_passkey command upon reception of this event.

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

rsi_bt_on_inquiry_complete_t

prototype

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

Description

This callback function is called if inquiry complete status is received from the module. This event will be given by the module when inquiry command is completely executed.

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status

rsi_bt_on_auth_complete_t

prototype

```
typedef void (*rsi_bt_on_auth_complete_t) (uint16_t resp_status,
rsi_bt_event_auth_complete_t *auth_complete);
 typedef struct rsi_bt_event_auth_complete_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 } rsi_bt_event_auth_complete_t;
```

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

rsi_bt_on_linkkey_request_t

prototype

```
typedef void (*rsi_bt_on_linkkey_request_t) (uint16_t resp_status,
rsi_bt_event_user_linkkey_request_t *user_linkkey_request);
 typedef struct rsi_bt_event_user_linkkey_request_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 } rsi_bt_event_user_linkkey_request_t;
```

Description

This callback is called if linkkey request is received from the module. User has to give linkkey reply command upon reception of this event.

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

rsi_bt_on_ssp_complete_t

prototype

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

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address
status	This is the SSP mode connection with remote device

rsi_bt_on_linkkey_save_t

prototype

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

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address
linkkey	This is the linkkey to be saved

rsi_bt_on_get_services_t

prototype

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

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
num_of_services	This is the number of services fetched
reserved	Reserved
uuid	This is the service uuid

rsi_bt_on_search_service_t

prototype

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

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
remote_dev_addr	This is the remote device address
Status	This is the search service status

rsi_bt_on_mode_change_t

prototype

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

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address
current_mode	This indicates the current state of connection between the local device and the remote device i.e., Active mode / Sniff mode
reserved	Reserved
mode_interval	This specify the time interval to each mode

rsi_bt_on_sniff_subrating_t

prototype

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

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address
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.

9.2.2 SPP profile event callback description

rsi_bt_on_spp_connect_t

prototype

```
typedef void (*rsi_bt_on_spp_connect_t) (uint16_t resp_status,
rsi_bt_event_spp_connect_t *spp_connect);
 typedef struct rsi_bt_event_spp_connect_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 } rsi_bt_event_spp_connect_t;
```

Description

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

Precondition

None

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

rsi_bt_on_spp_disconnect_t

prototype

```
typedef void (*rsi_bt_on_spp_disconnect_t) (uint16_t resp_status,
rsi_bt_event_spp_disconnect_t *spp_disconnect);
 typedef struct rsi_bt_event_spp_disconnect_s
 {
 uint8_t dev_addr[RSI_DEV_ADDR_LEN];
 } rsi_bt_event_spp_disconnect_t;
```

Description

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

Precondition

N/A

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

rsi_bt_on_spp_rx_data_t

prototype

```
typedef void (*rsi_bt_on_spp_rx_data_t) (uint16_t resp_status,
rsi_bt_event_spp_receive_t *bt_event_spp_receive);
 typedef struct rsi_bt_event_spp_receive_s
 {
 uint16_t data_len;
 uint8_t data[200];
 } rsi_bt_event_spp_receive_t;
```

Description

This callback is called when SPP receive event is raised from the module. This event will be given by the local device when it receives data from the remote device.

Precondition

N/A

Parameters

Variables	Description
resp_status	This is the data receive status
spp_receive	This is the SPP profile received data structure

9.2.3 A2DP profile event callback description

`rsi_bt_on_a2dp_connect_t`

prototype

```
typedef void (*rsi_bt_on_a2dp_connect_t) (uint16_t resp_status,
rsi_bt_event_a2dp_connect_t *a2dp_connect);
typedef struct rsi_bt_event_a2dp_connect_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_connect_t;
```

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.

Precondition

rsi_bt_a2dp_init() should be call.

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

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

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.

Precondition

N/A

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

`rsi_bt_on_a2dp_configure_t`

prototype

```
typedef void (*rsi_bt_on_a2dp_configure_t) (uint16_t resp_status,
rsi_bt_event_a2dp_configure_t *a2dp_configure);
typedef struct rsi_bt_event_a2dp_configure_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_configure_t;
```

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.

Precondition

N/A

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

rsi_bt_on_a2dp_start_t

prototype

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

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

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.

Precondition

N/A

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address
sample_freq	Sample frequency

rsi_bt_on_a2dp_open_t

prototype

```
typedef void (*rsi_bt_on_a2dp_open_t) (uint16_t resp_status,
rsi_bt_event_a2dp_open_t *a2dp_open);
typedef struct rsi_bt_event_a2dp_open_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_open_t;
```

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.

Precondition

N/A

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

`rsi_bt_on_a2dp_suspend_t`

prototype

```
typedef void (*rsi_bt_on_a2dp_suspend_t) (uint16_t resp_status,
rsi_bt_event_a2dp_suspend_t *a2dp_suspend);
typedef struct rsi_bt_event_a2dp_suspend_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_suspend_t;
```

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.

Precondition

N/A

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

`rsi_bt_on_a2dp_abort_t`

prototype

```
typedef void (*rsi_bt_on_a2dp_abort_t) (uint16_t resp_status,
rsi_bt_event_a2dp_abort_t *a2dp_abort);
typedef struct rsi_bt_event_a2dp_abort_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_abort_t;
```

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.

Precondition

N/A

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

`rsi_bt_on_a2dp_close_t`

prototype

```
typedef void (*rsi_bt_on_a2dp_close_t) (uint16_t resp_status,
rsi_bt_event_a2dp_close_t *a2dp_close);
typedef struct rsi_bt_event_a2dp_close_s
{
  uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
} rsi_bt_event_a2dp_close_t;
```

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.

Precondition

N/A

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address

`rsi_bt_on_a2dp_encode_data_t`

prototype

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

Description

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

Precondition

N/A

Parameters

Variables	Description
resp_status	This is the response status
dev_addr	This is the remote device address
`encode_data_len`	length of the received data
`encode_data`	received encode data

10 Bluetooth Low Energy APIs

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

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

10.1 Basic Structure defines

10.2 **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.

10.2.1 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.

10.2.2 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.

10.2.3 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.

10.2.4 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

10.2.5 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) .

10.2.6 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.

10.2.7 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.

10.2.8 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.

10.2.9 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.

10.2.10 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.

10.2.11 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
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

10.2.12 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

Variables	Description
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;

10.2.13 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;

10.3 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

10.4 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

10.4.1 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.

10.4.2 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
lcid	local channel identifiervalue
len	length of the data to be sent to remote device.
data	Actual data that has to be sent

10.4.3 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

10.4.4 rsi_ble_rx_test_mode_t;

Structure

```
typedef struct rsi_ble_rx_test_mode_s {
    uint8_t rx_channel;
    uint8_t phy;
    uint8_t modulation;
} rsi_ble_rx_test_mode_t;
```

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

Structure Variables

Variables	Description
rx_channel	Channel in which packet have to be received.
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

10.4.5 rsi_ble_tx_test_mode_t

Structure

```
typedef struct rsi_ble_tx_test_mode_s {
    uint8_t tx_channel;
    uint8_t phy;
    uint8_t tx_len;
    uint8_t tx_data_mode;
} rsi_ble_tx_test_mode_t;
```

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

Structure Variables

Variables	Description
tx_channel	
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.
tx_data_mode	0x00 PRBS9 sequence '11111111100000111101... 0x01 Repeated '11110000' 0x02 Repeated '10101010' 0x03 PRBS15 0x04 Repeated '11111111' 0x05 Repeated '00000000'

10.4.6 rsi_ble_end_test_mode_t

Structure

```
typedef struct rsi_ble_end_test_mode_s {
    uint16_t num_of_pkts;
} rsi_ble_end_test_mode_t;
```

Description

This command is used to stop any test which is in progress.

Structure Variables

Variables	Description
num_of_pkts	No.of RX packets received are displayed

10.5 GAP API

This section describes the GAP APIs

10.5.1 rsi_ble_start_advertising

Prototype

```
int32_t rsi_ble_start_advertising(void)
```

Description
This API starts advertising.

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

Parameters
None

Return Values

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

Example

status = rsi_ble_start_advertising ();

10.5.2 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();

10.5.3 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

10.5.4 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();
```

10.5.5 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);

10.5.6 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);
```

10.5.7 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

Value	Description
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);

10.5.8 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);
```

10.5.9 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

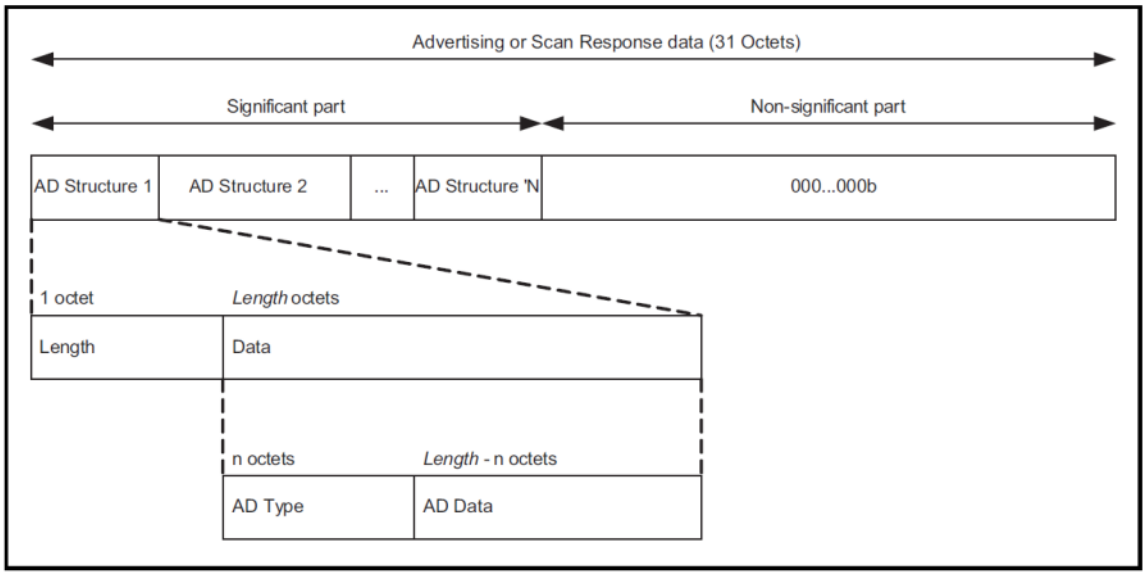
Note
1. The maximum length of advertising data payload is 31 bytes.
2. The basic format of advertise payload record contains length and data as below:

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);
```

10.5.10 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
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);
```

10.5.11 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

Parameter	Description
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);
```

10.5.12 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);
```

10.5.13 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);

10.5.14 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
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);

10.5.15 rsi_ble_set_random_address

Prototype

```
int32_t rsi_ble_set_random_address(void)
```

Description
This API sets the LE random device address.

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

Parameters
none

Return Values

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

Example
status=rsi_ble_set_random_address();

10.5.16 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 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.

10.6.1 GATT Client APIs

rsi_ble_get_profiles_async

Prototype

```
int32_t rsi_ble_get_profiles_async(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 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_async (remote_dev_addr, 1, 0xffff, &ble_profiles);

rsi_ble_get_profile_async

Prototype

```
int32_t rsi_ble_get_profile_async (uint8_t *dev_addr,
 uuid_t profile_uuid,
 profile_descriptors_t *p_profile)
```

Description

- This API gets the specific profile / service of the connected remote device
- rsi_ble_on_profile_resp_t callback function is called once the service characteristics response is received

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

Parameters

Parameter	Description
dev_addr	This is the remote device address
profile_uuid	This is the services/profiles which are searched using profile_uuid
p_profile	This is the profile / service information filled in this structure after retrieving from the remote device.

Return Values

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

Example
status= rsi_ble_get_profile_async (remote_dev_addr, search_serv, &ble_profile);

rsi_ble_get_char_services_async

Prototype

```
int32_t rsi_ble_get_char_services_async(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 rsi_ble_resp_char_services_t *p_char_serv_list);
```

Description

- This API gets the service characteristics of the connected / remote device.
- rsi_ble_on_inc_services_resp_t callback function is called once the included service characteristics response is received

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

Parameters

Parameter	Description
dev_addr	This is the remote device address
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records
p_char_service_list	This is the service characteristics details filled in this structure

Return Values

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

Example

```
//! get characteristics of the immediate alert service
 rsi_6byte_dev_address_to_ascii ((int8_t *)remote_dev_addr, (uint8_t
*)conn_event_to_app.dev_addr);

 rsi_ble_get_char_services_async (remote_dev_addr, *(uint16_t
*)rsi_ble_service.start_handle, *(uint16_t
*)rsi_ble_service.end_handle,
 &char_servs);
```

rsi_ble_get_inc_services_async

Prototype

```
int32_t rsi_ble_get_inc_services_async(uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 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_async (remote_dev_addr, 1, 0xFFFF, &ble_inc_serv);

rsi_ble_get_char_value_by_uuid_async

Prototype

```
int32_t rsi_ble_get_char_value_by_uuid_async (uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 uuid_t char_uuid,
 rsi_ble_resp_att_value_t *p_char_val)
```

Description

- This API gets the characteristic value by UUID (char_uuid).
- rsi_ble_on_read_resp_t callback function is called once the attribute value is received

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

Parameters

Parameter	Description
dev_addr	This is the remote device address
start_handle	This is the start handle (index) of the remote device's service records
end_handle	This is the end handle (index) of the remote device's service records
char_uuid	This is the UUID of the characteristic
p_char_val	This is the characteristic value entered in this structure

Return Values

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

Example
status = rsi_ble_get_char_value_by_uuid_async (remote_dev_addr, 1, 0xFFFF, search_serv, &ble_read_val);

rsi_ble_get_att_descriptors_async

Prototype

```
int32_t rsi_ble_get_att_descriptors_async (uint8_t *dev_addr,
 uint16_t start_handle,
 uint16_t end_handle,
 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_async (remote_dev_addr, 1, 0xFFFF, &ble_desc_services);

rsi_ble_get_att_value_async

Prototype

```
int32_t rsi_ble_get_att_value_async (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_async (remote_dev_addr, 3, &ble_read_val);

rsi_ble_get_multiple_att_values_async

Prototype

```
int32_t rsi_ble_get_multiple_att_values_async (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_async (remote_dev_addr, 2, handles, &ble_read_val);

rsi_ble_get_long_att_value_async

Prototype

```
int32_t rsi_ble_get_long_att_value_async (uint8_t *dev_addr,
 uint16_t handle,
 uint16_t offset,
 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_async (remote_dev_addr, 3, 0, &ble_read_val);

rsi_ble_set_att_value_async

Prototype

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

Description

- This API sets the attribute value.
- rsi_ble_on_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_async (RSI_BLE_DEV_1_ADDR, 0x60, 2, (uint8_t
*)&slave_val);
```

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");

rsi_ble_set_long_att_value_async

Prototype

```
int32_t rsi_ble_set_long_att_value_async(uint8_t *dev_addr,
 uint16_t handle,
 uint16_t offset,
 uint8_t data_len,
 uint8_t *p_data)
```

Description

- This API sets long attribute value.

- rsi_ble_on_write_resp_t callback function is called once the attribute set action is completed

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

Parameter	Description
dev_addr	This is the remote device address
handle	attribute handle
Offset	attribute value offset
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_async  (dev_addr,handle,offset,data_len,&p_data);
```

rsi_ble_prepare_write_async

Prototype

```
int32_t rsi_ble_prepare_write_async(uint8_t *dev_addr,
 uint16_t handle,
 uint16_t offset,
 uint8_t data_len,
 uint8_t *p_data)
```

Description

- This API prepares attribute value.
- rsi_ble_on_write_resp_t callback function is called once the prepare attribute write action is completed

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

Parameters

Parameter	Description
dev_addr	This is the remote device address
handle	This is the attribute handle
offset	This is the attribute value offset
data_len	This is the attribute value length
p_data	This is the attribute value

Return Values

Value

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

Example

status = rsi_ble_prepare_write_async (dev_addr, handle, offset, data_len, &p_data);

rsi_ble_execute_write_async

Prototype

```
int32_t rsi_ble_execute_write_async(uint8_t *dev_addr,
 uint8_t exe_flag)
```

Description

- This API executes the prepared attribute values.
- rsi_ble_on_write_resp_t callback function is called once the execute attribute write action is completed

Precondition

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

Parameters

Parameter	Description
dev_addr	This is the remote device address
exe_flag	This is the execute flag to write

Return Values

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

Example

```
status = rsi_ble_execute_write_async  (dev_addr,  exe_flag);
```

10.6.2 GATT Server APIs

rsi_ble_add_service

Prototype

```
int32_t rsi_ble_add_service (uuid_t serv_uuid,
 uint8_t num_of_attributes,
 uint8_t total_data_size,
 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
num_of_attributes	This is the number of attributes required in the service
total_data_size	This is the total data size required in the service
p_resp_serv	This is the new service handler 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_add_service  (service_uuid, num_of_attributes, total_data_size, &p_resp_serv);
```

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);
```

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);
```

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);

rsi_ble_gatt_read_response

Prototype

```
int32_t rsi_ble_gatt_read_response(uint8_t *dev_addr,
 uint8_t read_type,
 uint16_t handle,
 uint16_t offset,
 uint16_t length,
 uint8_t *p_data)
```

Description
This API sends the local attribute value to the remote device.

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

Parameters

Parameter	Description
dev_addr	This is the remote device BD Address
read_type	0-Read Response 1- Read blob response
handle	This is the local attribute start handle
offset	This is the local attribute value start offset
length	This is the local attribute value length
data	This is the Local attribute value

Return Values

Value	Description
0	Successful execution of the command

Value	Description
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);
```

Feature 4.2

rsi_ble_setphy

Prototype

```
int32_t rsi_ble_setphy(int8_t *remote_dev_address, uint8_t tx_phy,
uint8_t rx_phy, uint16_t coded_phy)
```

Description

This function is used to set tx and rx phy.

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

Parameters

Parameter	Description
`remote_dev_address`	Address of remote device
tx_phy	0 The Host prefers to use the LE 1M transmitter PHY (possibly among others) 1 The Host prefers to use the LE 2M transmitter PHY (possibly among others) 2 The Host prefers to use the LE Coded transmitter PHY (possibly among others) 3 – 7 Reserved for future use
`rx_phy`	0 The Host prefers to use the LE 1M receiver PHY (possibly among others) 1 The Host prefers to use the LE 2M receiver PHY (possibly among others) 2 The Host prefers to use the LE Coded receiver PHY (possibly among others) 3 – 7 Reserved for future use

Parameter	Description
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 );
```

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);

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

Parameter	Description
maxrxtime	Maximum time, in microseconds, that the local Controller supports for reception of a single Link Layer packet on a data connection.

Example

status = rsi_ble_read_max_data_len(&blereadmaxdatalen);

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);

10.6.3 **Privacy**

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

Parameter	Description
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 );
```

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);
```

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
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);
```

10.7 Callback functions

10.7.1 GAP register callbacks

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

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.

10.7.2 GAP event callback descriptions

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

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

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

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.

10.7.3 GATT register callbacks

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

rsi_ble_gatt_register_callbacks

Prototype

```
void 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);
```

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

Return Values

None

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

10.7.4 GATT Response callbacks description

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.

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.

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.

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.

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.

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.

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.

10.7.5 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

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.

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.

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

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

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

10.7.6 SMP register callbacks

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

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.

10.7.7 SMP event Callbacks Declarations

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.

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.

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.

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.

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

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

10.7.8 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

10.7.9 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

10.7.10 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

10.7.11 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

10.7.12 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

10.8 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**".

10.8.1 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

10.8.2 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

10.8.3 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

Define	Meaning
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 ZigBee APIs

This section explains the ZigBee APIs in order to initialize and configure the module in ZigBee mode. This section provides an overview of all the APIs and features present in the stack.

11.1 Management Interface

11.1.1 rsi_zigb_init_stack

Prototype

```
int16_t rsi_zigb_init_stack(void);
```

Description
This API initializes the ZigBee stack.

Precondition
rsi_wlan_init() API must be called with coexistence mode 2 or 3 before this API.

Parameters
None

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/examples/zigbee/switch/rsi_zb_app.c
RSI_DPRINT(RSI_PL1,"\n Sending init stack\n");
  //! zigb stack init
  status = rsi_zigb_init_stack();
```

11.1.2 rsi_zigb_reset_stack

Prototype

```
int16_t rsi_zigb_reset_stack(void);
```

Description
This API resets the ZigBee stack.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters
None

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, please refer to ZigBee Error Code table for description.

Example

```
See <SW Package Path>/host/sAPIs/examples/zigbee/switch/rsi_zb_app.c
RSI_DPRINT(RSI_PL1,"\n Sending reset stack\n");
  //! zigb stack init
  status = rsi_zigb_reset_stack();
```

11.1.3 rsi_zigb_set_profile

Prototype

```
int16_t rsi_zigb_set_profile(uint8_t profile_id);
```

Description
This API sets the profile which we are going to use but it is valid only for ZLL profile.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
Profile_id	uint8_t	This is the profile ID for which the stack is going to use 1 – Enable ZLL profile 0 – Disable ZLL profile

Return Value:

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_set_profile(1);
```

11.1.4 rsi_zigb_update_sas

Prototype

```
struct {
 uint8_t a_extended_pan_id[8];
 uint32_t channel_mask;
 uint8_tstartup_control;
 uint8_t use_insecure_join;
 uint8_t scan_attempts;
 uint8_t parent_retry_threshold;
 uint8_t a_trust_center_address[8];
 uint8_t a_network_key[16];
 uint16_t time_between_scans;
 uint16_t rejoin_interval;
 uint16_t max_rejoin_interval;
 uint16_t indirect_poll_rate;
 uint16_t a_pan_id;
 uint16_t network_manager_address;
 uint8_t a_trustcenter_master_key[16];
 uint8_t a_preconfigured_link_key[16];
 uint8_t end_device_bind_timeout;
 }Startup_Attribute_Set_t
 int16_t rsi_zigb_update_sas(Startup_Attribute_Set_t *Startup);
```

Description
This API sets the default startup attribute parameters.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
Startup	Startup_Attribute_Set_t	This is the pointer to startup the attributes structure

Structure Variables:

Parameters	Data type	Valid Range	Description
a_extended_pan_id	uint8_t [8]	0x0000000000000001 – 0xfffffffffffffffe	This field holds the extended PAN ID of the network in which the device needs to be a member . If the device doesnot know the specific network, then update this 8-byte field with zeros otherwise, specify the specific 8 bytes extended pan id.
channel_mask	uint32_t	32-bit field	The bits (b11, b12,... b26) indicate which channels are to be scanned (1=scan, 0=do not scan) for each of the 16 valid channels.
startup_control	uint8_t	0x00 – 0x03	This field indicates how the device needs to respond or start depending on the startup control value. **0x00** - Indicates that the device considers itself as a part of the network. indicated by the extended PAN ID attribute. In this case device does not perform any explicit join or rejoin operation. **0x01** - Indicates that the device forms a network with extended PAN ID given by the extended PAN ID attribute. The AIB's attribute APS Designated Coordinator is set to TRUE in this case. **0x02** - Indicates that the device rejoins network with extended PAN ID given by the extended PAN ID attribute. **0x03** - Indicates that the device starts "from scratch" and join the network using association. The default value for an un-commissioned device is 0x03.
use_insecure_ join	uint8_t	00 = TRUE 01 = FLASE	This is a flag controlling the use of insecure join at startup.
scan_attempts	uint8_t	1 - 255	This is an integer value representing the number of scan attempts to make before the NWK layer decides which ZigBee coordinator or router to associate with. This attribute has a default value of 5.
parent_retry_ threshold	uint8_t	3-10	This is the number of failed attempts to contact a parent that will cause a "find new parent" procedure to be initiated
a_trust_center_ address	uint16_t	0x0000-0xFFFF	This is the address of the network manager
a_network_key	uint8_t[16]	Variable	This is the network key
time_between_ scans	uint16_t	1 – 0xFFFF	This is the time between scans in milliseconds

Parameters	Data type	Valid Range	Description
rejoin_interval	uint16_t	Max value:60	This is the rejoin interval in seconds
max_rejoin_ interval	uint16_t	Max value:3600	This is the max rejoin interval in seconds
indirect_poll_rate	uint16_t	In msec	This is the rate in milliseconds to poll the parent
a_pan_id	uint16_t	0x0001 – 0xFFFE	This field indicates the PAN ID of the device.
network_ manager_address	uint16_t	0x0000-0xFFFF	This is the address of the network manager.
a_trustcenter_ master_key	uint8_t[16]	Variable	This is the trust center master key
a_preconfigured_link_key	uint8_t[16]	Variable	This is the link key
end_device_bind_timeout	uint8_t	1-60	This is the coordinator which waits (in seconds) for a second for end device bind request to arrive. The default value is 10.

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/utilities/
app_profile_utilities/src/AppCommissioningUtilities.c
rsi_zigb_update_sas((uint8_t*)&localAttributeSet);
```

11.1.5 rsi_zigb_update_zdo_configuration

Prototype

```
struct {
 uint8_t config_permit_join_duration;
 uint8_t config_NWK_secure_all_frames;
 uint8_t config_formation_attempts;
 uint8_t config_scan_duration;
 uint8_t config_join_attempts;
 uint8_t config_preconfigured_key;
 uint16_t a_config_trust_center_short_address;
 uint8_t automatic_poll_allowed;
 uint8_t config_authentication_poll_rate;
 uint16_tconfig_switch_key_time;
 uint8_t config_security_level;
 uint8_t config_aps_ack_poll_time_out;
 uint8_t a_manufacturer_code[2];
 }ZDO_Configuration_Table_t;
 int16_t rsi_zigb_update_zdo_configuration(ZDO_Configuration_Table_t
*pzdo_cnf);
```

Description
This API sets the ZDO configuration.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
*pzdo_cnf	ZDO_Configuration_Table_t	This is the pointer to startup ZDO configuration structure

Structure Variables:

Name	Type	Valid Range	Description
config_permit_ join_duration	uint8_t	00-0xFF 0x00 Indicates that no devices can join 0xFF Indicates that devices are always allowed to join 0x01 - 0xFE Indicates the time in seconds for which the device allows other devices to join	It defines the time for which a coordinator or router device allows other devices to join to itself
config_NWK_ secure_all_frames	uint8_t	0-Enable 1-Disabled	It defines whether security is applied for incoming and outgoing network data frames or not

Name	Type	Valid Range	Description
config_formation_attempts	uint8_t	1	This is the number of times the devices attempts for formation failure
config_scan_duration	uint8_t	00-0xFE	This is the field which indicates the duration of active scan while performing startup, join or rejoin the network.
config_join_attempts	uint8_t	Default = 02	This is the field which indicates the number of times the join command is retried once it gets fail
config_preconfigured_key	uint8_t	Set to 0x01 if supporting only preconfigured nwk key, or else to be set with 0x02 if requires high security.	This is the field which indicates whether a pre configured key is already available in the device or not
a_config_trust_center_short_address	l uint16_t	Default 0x0000	This is the field which holds the short address of the TC
automatic_poll_allowed	uint8_t	Enable- 0x01 Disable- 0x00(default)	This field indicates whether an end device does an auto poll or not.
config_authentication_poll_rate	uint8_t	Default 0x64(100 msec)	This is the poll rate of an end device waiting for authentication.
config_switch_key_time	uint16_t	Default 0x06	This is the time after which active key sequence number is changed once the device receives switch Key request
config_security_level	uint8_t	0x05	This is the security level for outgoing and incoming network frames.
config_aps_ack_poll_time_out	uint8_t	0xFA(250 msec)	This is the maximum number of seconds to wait for an acknowledgement to a transmitted frame.
a_manufacturer_code	uint8_t[2]	0x0000 – 0xFFFF	This is the manufacturer code

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
status = rsi_zigb_update_zdo_configuration(&g_Table_Default)
```

11.1.6 rsi_zigb_form_network

Prototype:

```
int16_t rsi_zigb_form_network ( uint8_t RadioChannel,
 uint8_t power ,uint8_t * pExtendedPanId );
```

Description
This API allows the application to establish the network in the provided channel with the specified extended PAN ID.

Precondition
rsi_zigb_init_stack() must be called before this API.
If the default zdo and sas parameters are to be changed, call rsi_zigb_update_sas() and rsi_zigb_update_zdo_configuration() before using this API.
Supported for Coordinater.

Parameters

Parameters	Data type	Description
RadioChannel	uint8_t	This is the channel on which the network needs to be formed. The valid channels are between 11 – 26
power	uint8_t	This is the TX power to be used by the device. The range of TX power is about 0 – 12dBm.
pExtendedPanId	uint8_t	This is the pointer to the extended PAN ID array of 8 bytes.

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_form_network(channel,power,mac_addr);
```

11.1.7 rsi_zigb_join_network

Prototype

```
int16_t rsi_zigb_join_network(uint8_t DeviceType,
 uint8_t RadioChannel,
 uint8_t power ,
 uint8_t *pExtendedPanId);
```

Description
This API allows the application to join the network in the provided channel with the (coordinator of) specified extended PAN ID.

Precondition
rsi_zigb_init_stack() must be called before this API.
Can only be called after a network is found while scanning.
Supported for End-Device and Router.

Parameters

Parameters	Data type	Description
DeviceType	uint8_t	0x01 – Router 0x02 – End-Device
RadioChannel	uint8_t	This is the channel on which the network needs to be formed. The valid channels are between 11 – 26
power	uint8_t	This is the TX power to be used by the device. The range of TX power is about 0 – 12dBm.
pExtendedPanId	uint8_t	This is the pointer to the extended PAN ID array of 8 bytes.

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/examples/zigbee/switch/rsi_zb_app.c
status =rsi_zigb_join_network(device_type,
      rsi_zigb_app_info.nwkinfo.channel, 0x0f,
rsi_zigb_app_info.nwkinfo.extendedPanId);
```

11.1.8 rsi_zigb_permit_join

Prototype

```
int16_t rsi_zigb_permit_join(uint8_t PermitDuration);
```

Description
This API allows the application to enable join permit on the device for the specified duration in seconds.

Precondition
rsi_zigb_form_network() must be called before this API.
Supported for coordinator.

Parameters:

Parameters	Data type	Description
PermitDuration	uint8_t	This is the length of time in seconds during which the ZigBee coordinator or router will allow associations. The valid values are as below: 0x00 = Disabled. 0xFF = Always allowed associations. 0x01 – 0xFE = Associations allowed for this timeout

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_permit_join(0xFF);
```

11.1.9 rsi_zigb_leave_network

Prototype

```
int16_t rsi_zigb_leave_network(void);
```

Description
This API allows to perform self leave from the network.

Precondition
rsi_zigb_join_network() must be called before this API.
Supported for End-Device and Router.

Parameters
None

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/utilities/
app_profile_utilities/src/AppCBKEUtilities.c
rsi_zigb_leave_network();
```

11.1.10 rsi_zigb_initiate_scan

Prototype

```
int16_t rsi_zigb_initiate_scan(uint8_t scanType,
 uint32_t channelMask,
 uint8_t duration);
```

Description
This API allows the application to initiate scan of the specified type in the specified channel mask for the specified duration.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
scanType	uint8_t	0x00 – Energy Detection scan 0x01 – Active scan
ChannelMask	uint32_t	This is the five most significant bits (b27,..., b31) and 11 least significant bits (b0,b1,…b10) are reserved. The middle 16 bits (b11, b12,... b26) indicate which channels are to be scanned (1=scan, 0=do not scan).
Duration	uint8_t	This is the value used to calculate the length of time to spend scanning each channel. The time spent scanning each channel is (aBaseSuperframeDuration * (2^n + 1)) symbols, where n is the value of the ScanDuration parameter .

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package
Path>/host/sAPIs/examples/zigbee/switch/rsi_zb_app.cstatus =
rsi_zigb_initiate_scan(g_MAC_ACTIVE_SCAN_TYPE_c,
      g_CHANNEL_MASK_c, g_SCAN_DURATION_c);
```

11.1.11 rsi_zigb_stop_scan

Prototype

```
int16_t rsi_zigb_stop_scan(void);
```

Description
This API allows the application to stop the scan that was initiated.

Precondition
rsi_zigb_initiate_scan() must be called before this API.

Parameters
None

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_stop_scan();
```

11.1.12 rsi_zigb_network_state

Prototype:

```
int16_t rsi_zigb_network_state(void);
```

Description
This API allows the application to know if the device is in the process of joining, already Joined or leaving the network.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator)
rsi_zigb_join_network() must be called before this API (End-Device, Router)

Parameters
None

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/utilities/
app_profile_utilities/src/AppZllCommissioningClusterUtilities.c
if ((rsi_zigb_network_state() == g_ZigBeeNotPartOfNWK_c)
        || (rsi_zigb_network_state() ==
g_ZigBeePerformingLeaveFromNWK_c)){


        /* device is factory new device*/
        return;
    }
```

11.1.13 rsi_zigb_stack_is_up

Prototype

```
int16_t rsi_zigb_stack_is_up(void);
```

Description:
This API knows whether the stack is running or not. It returns success after joining to coordinator for End-Device and Router. For coordinator it returns success after forming the network.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters
None

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/utilities/
app_profile_utilities/src/ApplicationThinLayer.c
                    stack_is_up = rsi_zigb_stack_is_up();
```

11.1.14 rsi_zigb_get_self_ieee_address

Prototype

```
int16_t rsi_zigb_get_self_ieee_address(uint8_t* ieee_addr);
```

Description
This API allows the application to read the device self IEEE extended address.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
ieee_addr	uint8_t*	This is the pointer to IEEE address (array of 8 bytes) in which the device self IEEE address to be copied.

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is other than above, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/examples/zigbee/switch/rsi_zb_app.c
      uint8_t ieee_addr[8] = {0x0};
      status = rsi_zigb_get_self_ieee_address(ieee_addr);
```

11.1.15 rsi_zigb_is_it_self_ieee_address

Prototype

```
int16_t rsi_zigb_is_it_self_ieee_address(uint8_t *pIEEEAddress);
```

Description
This API allows the application to know whether the given extended address is self IEEE address or not.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
pIEEEAddress	uint8_t*	This is the pointer to IEEE address (array of 8 bytes) which has to be verified.

Return Value

Value	Description
0	g_TRUE_c, if IEEE address is the local node's ID
Non Zero	Failure : g_FALSE_c Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_is_it_self_ieee_address(resp-
>uCmdRspPayLoad.DevIEEEAddr.Self_ieee )
```

11.1.16 rsi_zigb_get_self_short_address

Prototype

```
int16_t rsi_zigb_get_self_short_address(void);
```

Description
This API allows the application to know self short address.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator)
rsi_zigb_join_network() must be called before this API (End-Device, Router)

Parameters
None

Return Value

Values	Description
0	16-bit short self address
Non Zero	Failure : g_FALSE_c Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is other than above, please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/utilities/
app_profile_utilities/src/AppDiscoveryUtilities.c
if ( App_getTCShortAddress() == rsi_zigb_get_self_short_address() ) {
            ChangeEsiRegState( IDEAL_STATE );
        }
```

11.1.17 rsi_zigb_set_manufacturer_code_for_node_desc

Prototype

```
int16_t rsi_zigb_set_manufacturer_code_for_node_desc(uint16_t code);
```

Description
This API allows the user to set manufacturer code in the node descriptor.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator)
rsi_zigb_join_network() must be called before this API (End-Device, Router)

Parameters

Parameters	Data type	Description
code	uint16_t	The 16-bit manufacturer code for the local node. Range:0x0000 -0xFFFF.

Return Value

Values	Description
0	Successful execution of the command

Values	Description
Non Zero	Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_set_manufacturer_code_for_node_desc(api_test_var-
>ManufacturerCode);
```

11.1.18 rsi_zigb_set_power_descriptor

Prototype

```
struct {
 uint8_t current_powermode_avail_power_sources;
 uint8_t current_powersource_currentpowersourcelevel;
 }Node_Power_Descriptor_t;
 int16_t rsi_zigb_set_power_descriptor(Node_Power_Descriptor_t *
nodePowerDesc);
```

Description
This API allows the application to set power descriptor for the device.

Precondition
rsi_zigb_init_stack() must be called before this API.

Structure Variables

Name	Type	Valid Range	Description
current_powermode_avail_power_sources	uint8_t	00-0xFF	This is the first 4 bits of LSB which gives the current sleep/ power saving mode of the node and the MSB 4 bits gives the power sources available in this node.
current_powersource_currentpowersourcelevel	uint8_t	00-0xFF	This is the first 4 bit of LSB which gives the current power source and 4 bits of MSB which gives the current power source level.

Return Value:

Values	Description
0	Successful execution of the command

Values	Description
Non Zero	Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_set_power_descriptor(&api_test_var->nodePowerDesc_t);
```

11.1.19 rsi_zigb_set_maxm_incoming_txfr_size

Prototype

```
int16_t rsi_zigb_set_maxm_incoming_txfr_size(uint16_t size);
```

Description
This API allows the application to specify the maximum incoming transfer size the device is capable of.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
size	uint16_t	This is the maximum incoming transfer size for the local node. Range:0-128

Return Value

Values	Description
0	Successful execution of the command
Non Zero	Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_set_maxm_incoming_txfr_size(0x1234);
```

rsi_zigb_set_operating_channel

Prototype

```
int16_t rsi_zigb_set_operating_channel(uint8_t channel);
```

Description
This API allows the application to set the operating channel.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
Channel	uint8_t	This is the desired radio channel. Range:11 to 26.

Return Value

Values	Description
0	Successful execution of the command
Non Zero	Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/zcl/ZLL_Utilities/
src/ZLL_Process.c
rsi_zigb_set_operating_channel(ZLL_NetworkGetchannel());
```

rsi_zigb_get_device_type

Prototype

```
int16_t rsi_zigb_get_device_type(uint8_t *dev_type);
```

Description
This API allows the application to get the device type.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
*dev_type	uint8_t	This is the type of the device. 0 – Coordinator. 1 – Router. 2 – EndDevice.

Return Value

Values	Description
0	Successful execution of the command and also dev_type parameter is updated
Non Zero	Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/zcl/ZLL_Utilities/
src/ZLL_Process.c
      uint8_t device_type = 0;
      status = rsi_zigb_get_device_type(&device_type);
```

rsi_zigb_get_operating_channel

Prototype

```
int16_t rsi_zigb_get_operating_channel(void);
```

Description
This API allows the application to get the operating channel.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters
None.

Return Value

Values	Description
0	Successful execution of the command and also dev_type parameter is updated

Values	Description
Non Zero	Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to the ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/zcl/ZLL_Utilities/
src/ZLL_Interpan.c
pDescriptor->logical_channel=rsi_zigb_get_operating_channel();
```

rsi_zigb_get_short_panid

Prototype

```
int16_t rsi_zigb_get_short_panid(void);
```

Description:
This API allows the application to get the short panid.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator)
rsi_zigb_join_network() must be called before this API (End-Device, Router)

Parameters
None.

Return Value

Values	Description
0	16 bit Pan Id
Non Zero	Failure : 0XFFFF Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/zcl/ZLL_Utilities/
src/ZLL_Interpan.c      rsi_zigb_get_short_panid();
```

rsi_zigb_get_extended_panid

Prototype

```
int16_t rsi_zigb_get_extended_panid(uint8_t *p_extended_panid );
```

Description
This API allows the application to get the extended pan id.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator)
rsi_zigb_join_network() must be called before this API (End-Device, Router)

Parameters

Parameters	Data type	Description
p_extended_panid	uint8_t*	This is the pointer to the array in which the extended pan id is to be updated.

Return Value

Values	Description
0	Successful execution of the command. The extended pan-id is updated in p_extended_panid
Non Zero	Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/zcl/ZLL_Utilities/
src/ZLL_Interpan.c
rsi_zigb_get_extended_panid(ext_panid);
```

rsi_zigb_get_endpoint_id

Prototype

```
int16_t rsi_zigb_get_endpoint_id(uint8_t Index);
```

Description
This API allows the application to get the endpoint id.

Precondition
rsi_zigb_set_simple_descriptor() must be called before this API.

Parameters

Parameters	Data type	Description
Index	uint8_t	This indicate the index of the array. This value should be less than the number of endpoints.

Return Value

Values	Description
0	Successful execution of the command. The valid Endpoint ID located in the specified index.
Non Zero	Returns a non-zero value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_get_endpoint_id(0x1);
```

rsi_zigb_get_simple_descriptor

Prototype

```
typedef uint16_t profile_id_t,cluster_id_t;
 typedef struct Simple_Descriptor_Tag {
 profile_id_t app_profile_id;
 uint16_t app_device_id;
 uint8_t app_device_version;
 uint8_t incluster_count;
 cluster_id_t const *p_incluster_list;
 uint8_t outcluster_count;
 cluster_id_t const *p_outcluster_list;
 }Simple_Descriptor_t;
 int16_t rsi_zigb_get_simple_descriptor(
 uint8_t endpointId,
 Simple_Descriptor_t *simple_desc);
```

Description
This API allows the application to get the simple descriptor.

Precondition
rsi_zigb_set_simple_descriptor() must be called before this API.

Parameters

Parameters	Data type	Description
endpoint_id	uint8_t	This is the endpoint on which these clusters are defined
simple_desc	Simple_Descriptor_t *	This is the pointer to the simple descriptor of a specified endpoint.

11.1.20 **Structure: Simple_Descriptor_t**

Name		Type		Range		Description
app_profile_id	profile_id_t		0x0000 – 0xffff		This is the endpoint on which these clusters are defined	
app_device_id	uint16_t		0x0000 - 0xfff7		This is the address of the designated network channel manager function	
app_device_version	uint8_t		1-254		This is the version of the ZigBee protocol in use in the discovered network.	
incluster_count	uint8_t		0x00 – 0x0f		This is the number of the Input Clusters	
p_incluster_list	cluster_id_t		-		This is the pointer to the buffer holding input clusters.	
outcluster_count	uint8_t		0x00 – 0x0f		This is the number of Output Clusters	
p_outcluster_list	cluster_id_t		-		This is the pointer to the buffer holding output clusters.	

Error rendering macro 'captioneditem' : Plugin license error: EXPIRED

Return Value

Value	Description
0	Success - g_TRUE_c.
Non Zero	Failure - g_FALSE_c Returns a negative value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, please refer to ZigBee Error Codes table for description.

Example

```
#define ONOFF_SWITCH_END_POINT     0x01
      rsi_zigb_get_simple_descriptor(ONOFF_SWITCH_END_POINT);
```

rsi_zigb_set_simple_descriptor

Prototype

```
int16_t rsi_zigb_set_simple_descriptor(
 uint8_t endpointId,
 Simple_Descriptor_t *simple_desc);
```

Description
This API allows the application to set the simple descriptor.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator)
rsi_zigb_join_network() must be called before this API (End-Device, Router)

Parameters

Parameters	Data type	Description
endpoint_id	uint8_t	This is the endpoint on which the clusters are defined.
p_simple_desc	Simple_Descriptor_t *	This is the pointer to simple descriptor of specified endpoint.

Return Value

Value	Description
0	Success - g_TRUE_c.
Non Zero	Failure - g_FALSE_c Returns a negative value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/zcl/ZLL_Utilities/
src/ZLL_Interpan.c
#define ONOFF_SWITCH_END_POINT     0x01
Simple_Descriptor_t        *DeviceSimpleDesc;
status = rsi_zigb_set_simple_descriptor(g_ON_OFF_SWITCH_ENDPOINT_c,
      &On_Off_Switch_Simple_Desc);
```

rsi_zigb_get_endpoint_cluster

Prototype

```
int16_t rsi_zigb_get_endpoint_cluster(uint8_t EndPointId, uint8_t
ClusterType,uint8_t ClusterIndex)
```

Description
This API allows the application to read the endpoint's cluster in the specified list at the specified end-point index.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
EndPointId	uint8_t	This is the 8-bit endpoint id whose cluster id needs to be retrieved
ClusterType	uint8_t	This indicates if the incluster list should be read or outcluster list to be read. 0 indicates incluster list and 1 indicates outcluster list.
ClusterIndex	uint8_t	This indicates the index of the list of which cluster id is to be read. This index should be less than the number of clusters supported in the list as read in the simple descriptor.

Return Value

Value	Description
0	Success - Cluster id of the endpoint's simple descriptor located at the specified index.
Non Zero	Failure - g_INVALID_CLUSTER_ID_c Returns a negative value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_get_endpoint_cluster(0x00,0x00,0x00);
```

rsi_zigb_get_short_addr_for_specified_ieee_addr

Prototype

```
int16_t rsi_zigb_get_short_addr_for_specified_ieee_addr(uint8_t *
pIEEEAddress);
```

Description
This API allows the application to get the 16-bit short address of the device for the given 64-bit IEEE address.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
pIEEEAddress	uint8_t*	This is the pointer to IEEE address whose 16-bit short address is to be determined

Return Value:

Value	Description
0	Success - 16-bit short address of the corresponding 64-bit IEEE address if the address is known.
Non Zero	Failure - INVALID_SHORT_ADDRESS Returns a negative value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Code table for description.

Example

```
See <SW Package Path>/host/sAPIs/zigbee/profiles/utilities/
app_profile_utilities/src/AppTunnelingClusterUtilities.c
  uint8_t IEEE_address[8];
rsi_zigb_get_short_addr_for_specified_ieee_addr(pBuffer-
>src_address.IEEE_address);
```

rsi_zigb_get_ieee_addr_for_specified_short_addr

Prototype

```
int16_t rsi_zigb_get_ieee_addr_for_specified_short_addr(
 uint16_t shortAddr,
 uint8_t* ieee_addr);
```

Description
This API allows the application to get the 64-bit IEEE address of the device for the given 16-bit Short address.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters:

Parameters	Data type	Description
shortAddr	uint16_t	This is the short address which gives a 16-bit short address of which corresponding 64-bit IEEE address needs to be determined
ieee_addr	uint8_t*	This is the pointer to the location where the IEEE address needs to be copied.

Return Value

Value	Description
0	Success - **g_TRUE_c**, if successfully retrieved the IEEE address from the neighboring table or Address map table
Non Zero	Failure - **g_FALSE_c** Returns a negative value if command issued in wrong state and packet allocation failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
 uint16_t                pan_id ;
 uint8_t tcEUI[8];
status =rsi_zigb_get_ieee_addr_for_specified_short_addr( pan_id,
tcEUI );
```

rsi_zigb_read_neighbor_table_entry

Prototype

```
typedef struct ZigBeeNeighborTableEntry_Tag {
 uint16_t shortId;
 uint8_t averageLqi;
 uint8_t incomingCost;
 uint8_t outgoingCost;
 uint8_t age;
 uint8_t aIEEEAddress[8];
 }ZigBeeNeighborTableEntry_t;
 int16_t rsi_zigb_read_neighbor_table_entry(
 uint8_t Index,
 ZigBeeNeighborTableEntry_t *neigbor_table);
```

Description
This API allows the application to read the neighboring table entry in the specified index.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
Index	uint8_t	This indicates the index from where the neighboring table entry is to be retrieved.
neigbor_table	ZigBeeNeighborTableEntry_t*	This is the pointer to the location where the NeighbortableEntry needs to be copied.

Structure: ZigBeeNeighborTableEntry_t

Name	type	Range	Description
shortId	uint16_t	0x0000 – 0xffff	This is the neighbor's two byte short address
averageLqi	uint8_t	0x00-0xf0	An exponentially weighted moving average of the link quality values of incoming packets from this neighbor as reported by the PHY.
incomingCost	uint8_t	1-7	This is the incoming cost for this neighbor, computed from the average LQI. Values range from 1 for a good link to 7 for a bad link .
outgoingCost	uint8_t	1-7	This is the outgoing cost for this neighbor, obtained from the most recently received neighbor exchange message from the neighbor

Name	type	Range	Description
age	uint8_t	3-16	This is the number of aging periods elapsed since a neighbor exchange message was last received from this neighbor. An entry with an age greater than 3 is considered stale and may be reclaimed. The aging period is 16 seconds
aIEEEAddress	uint8_t[8]	-	This is the 8 byte IEEE address of the neighbor

Return Value

Value	Description
0	Success
Non Zero	Failure - **ZigBee_Invalid_Argument** Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_read_neighbor_table_entry( 0x0);
```

rsi_zigb_get_route_table_entry

Prototype

```
typedef struct ZigBeeRoutingTableEntry_Tag {
 uint16_t destAddr;
 uint16_t nextHop;
 uint8_t status;
 uint8_t age;
 uint8_t concentratorType;
 uint8_t routeRecordState;
 }ZigBeeRoutingTableEntry_t;
 int16_t rsi_zigb_get_route_table_entry(
 uint8_t Index,
 ZigBeeRoutingTableEntry_t *routing_table);
```

Description
This API allows the application to read the routing table entry in the specified index.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).

rsi_zigb_join_network() must be called before this API (Router).
Supported for Router and Coordinator.

Parameters

Parameters	Data type	Description
Index	uint8_t	This indicates the index from where the neighbor table entry is to be retrieved.
routing_table	ZigBeeRoutingTableEntry_t *	This is the pointer to a location where the Route table entry needs to be copied.

Structure: ZigBeeNeighborTableEntry_t

Name	type	Range	Description
destAddr	uint16_t	0x0000 – 0xffff	This is the short id of the destination
nextHop	uint16_t	0x0000 – 0xffff.	This is the short address of the next hop to this destination
status	uint8_t	1-7	This indicates whether this entry is active (0), being discovered (1), or unused (0x3).
age	uint8_t	1-7	This is the number of seconds since this route entry was last used to send a packet
concentratorType	uint8_t	0-2	This indicates whether this destination is a High RAM Concentrator (2), a Low RAM Concentrator (1), or not a concentrator (0).
routeRecordState	uint8_t	0-2	For a High RAM Concentrator. This indicates whether a route record is needed (2), has been sent (1), or is no long needed (0) because a source routed message from the concentrator has been received

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure **ZigBee_Index_Out_Of_Range : Accessing entry is out of range in the table.** **ZigBee_Invalid_Argument : Argument passed for API is invalid.** Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint8_t                  child_index;
      rsi_zigb_get_route_table_entry(child_index)
```

rsi_zigb_get_neighbor_table_entry_count

Prototype

```
int16_t rsi_zigb_get_neighbor_table_entry_count(void);
```

Description
This API allows the application to know the count of active neighbor table entries.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters
None.

Return Value

Value	Description
	Success - Total count of active neighbor table entries in the neighbor table
Non Zero	Failure Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_get_neighbor_table_entry_count();
```

rsi_zigb_get_child_short_address_for_the_index

Prototype

```
int16_t rsi_zigb_get_child_short_address_for_the_index(uint8_t
ChildIndex);
```

Description
This API allows the application to read the 16-bit short address of the child in the specified index.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).

rsi_zigb_join_network() must be called before this API (Router).
Supported for Router and Coordinator.

Parameters

Parameters	Data type	Description
ChildIndex	uint16_t	This indicates the index from where the 16-bit short address needs to be retrieved

Return Value

Value	Description
	Success - The child address.
Non Zero	Failure - g_INVALID_ADDRESS_c Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint8_t                 child_index;
rsi_zigb_get_child_short_address_for_the_index(child_index);
```

rsi_zigb_get_child_index_for_specified_short_addr

Prototype

```
int16_t rsi_zigb_get_child_index_for_specified_short_addr(uint16_t
childShortAddr)
```

Description
This API allows the application to get the index for the specified 16-bit child address.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).
Suported for Router and Coordinator.

Parameters

Parameters	Data type	Description
childShortAddr	uint16_t	This is the 16-bit short address whose index needs to be determined

Return Value:

Value	Description
	Success - index of the child address received in the input parameter.
Non Zero	Failure - m_NO_ENTRY_c Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint16_t                child_short_addr_t;

rsi_zigb_get_child_index_for_specified_short_addr(child_short_addr_t)
;
```

rsi_zigb_get_child_details

Prototype

```
int16_t rsi_zigb_get_child_details(
 uint8_t Index,
 uint8_t *ieee_addr,
 uint8_t DeviceType);
```

Description
This API allows the application to get the child details at the specified child index.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).
Suported for Router and Coordinator.

Parameters

Parameters	Data type	Description
Index	uint8_t	This is the index of the child of interest.
ieee_addr	uint8_t*	This is the child's EUI64 is copied into here.
DeviceType	uint8_t	This is the child's node type is copied into here. 0 – Coordinator. 1 – Router. 2 – EndDevice.

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure - ZigBeeUnknownDevice Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint8_t                 child_index
;
rsi_zigb_get_child_details(child_index);
```

rsi_zigb_end_device_poll_for_data

Prototype

```
int16_t rsi_zigb_end_device_poll_for_data( void );
```

Description
This API allows the application to poll the parent for data.

Precondition
rsi_zigb_join_network() must be called before this API.
Suported for End-Device.

Parameters
None.

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure - ZigBee_Invalid_Call Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint8_t                 child_index;
rsi_zigb_end_device_poll_for_data();
```

rsi_zigb_read_count_of_child_devices

Prototype

```
int16_t rsi_zigb_read_count_of_child_devices(void);
```

Description
This API allows the application to read the number of child devices on the node.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).
Suported for Router and Coordinator.

Parameters
None.

Return Value

Value	Description
	Success - Number of children joined
Non Zero	Failure Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_read_count_of_child_devices();
```

rsi_zigb_read_count_of_router_child_devices

Prototype

```
int16_t rsi_zigb_read_count_of_router_child_devices(void);
```

Description
This API allows the application to read the number of child devices on the node.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).

rsi_zigb_join_network() must be called before this API (End-Device, Router).
Suported for Router and Coordinator.

Parameters
None.

Return Value

Value	Description
	Success - Number of router children joined
Non Zero	Failure Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_read_count_of_router_child_devices();
```

rsi_zigb_get_parent_short_address

Prototype

```
int16_t rsi_zigb_get_parent_short_address(void);
```

Description
This API allows the application to get the parent's 16 bit short address.

Precondition
rsi_zigb_join_network() must be called before this API.
Suported for Router and End Device.

Parameters
None.

Return Value

Value	Description
	Success - Parent short address
Non Zero	Failure - g_INVALID_ADDRESS_c Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example:

```
rsi_zigb_get_parent_short_address();
```

rsi_zigb_get_parent_ieee_address

Prototype

```
int16_t rsi_zigb_get_parent_ieee_address(uint8_t *ieee_addr);
```

Description
This API allows the application to read it parent's 64-bit IEEE address.

Precondition
rsi_zigb_join_network() must be called before this API.
Suported for Router and End Device.

Parameters

Parameters	Data type	Description
ieee_addr	uint8_t*	This is the pointer to a location where parent's 64-bit IEEE address should be copied

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_get_parent_ieee_address();
```

rsi_zigb_initiate_energy_scan_request

Prototype

```
int16_t rsi_zigb_initiate_energy_scan_request(uint16_t
DestAddr,uint32_t
 ScanChannels, uint8_t ScanDuration, uint16_t ScanRetry);
```

Description
This API allows the application to request energy scan to be performed. This request may only be sent by the current network manager and must be unicasted and not broadcasted.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
DestAddr	uint16_t	This indicates the network address of the device to perform the scan. Range:0x0000-0xFFFF
ScanChannels	uint32_t	This is the five most significant bits (b27,..., b31) and 11 least significant bits (b0,b1,…b10) are reserved. The middle 16 bits (b11, b12,... b26) indicate which channels are to be scanned (1=scan, 0=do not scan).
ScanDuration	uint8_t	This indicates that how long to scan on each channel. The allowed values are 0 – 5.
ScanRetr	uint16_t	This indicates the number of scans to be performed on each channel (1-8)

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint16_t DestAddr;
uint32_t ScanChannels;
uint8_t ScanDuration;
uint16_t ScanRetry
rsi_zigb_initiate_energy_scan_request(DestAddr,ScanChannels,
ScanDuration,ScanRetry);
```

rsi_zigb_broadcast_nwk_manager_request

Prototype

```
int16_t rsi_zigb_broadcast_nwk_manager_request(uint16_t
NWKManagerShortAddr, uint32_t ActiveChannels);
```

Description
This API allows the application to broadcasts a request to change the channel. This request may only be sent by the current network manager.

Precondition
rsi_zigb_form_network() must be called before this API.
Supported for Coordinator.

Parameters

Parameters	Data type	Description
NWKManagerShortAddr	uint16_t	This indicates the 16-bit network address of the network manager.
ActiveChannels	uint32_t	This indicates the new active channel mask. The five most significant bits (b27,..., b31) and 11 least significant bits (b0,b1, …b10) are reserved. The middle 16 bits(b11, b12,... b26) indicate which channels are to be scanned (1=scan, 0=do not scan).

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint32_t channel_mask;
rsi_zigb_broadcast_nwk_manager_request(0x0000,channel_mask);
```

rsi_zigb_zdp_send_nwk_addr_request

Prototype

```
int16_t rsi_zigb_zdp_send_nwk_addr_request(uint8_t *
pIEEEAddrOfInterest, BOOL RequestType, uint8_t StartIndex);
```

Description
This API allows the application to send ZDP network address request to determine the 16-bit short address of the device whose IEEE address is known.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
pIEEEAddrOfInterest	uint8_t*	This is the pointer to a location of IEEE address whose 16-bit Network address is to be determined
RequestType	BOOL	This is the boolean value. if TRUE comes, it indicates single device response and if FALSE comes, it indicates extended device response.
StartIndex	uint8_t	This is the start index of the child devices list.

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure- g_FAILURE_c Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint8_t                  parent_ieee[8];
rsi_zigb_zdp_send_nwk_addr_request(parent_ieee,(BOOL)0,0);
```

rsi_zigb_zdp_send_ieee_addr_request

Prototype
int16_t rsi_zigb_zdp_send_ieee_addr_request(uint16_t shortAddress, BOOL RequestType,uint8_t StartIndex, BOOL APSAckRequired)

Description
This API allows the application to send ZDP IEEE address request to determine the 16-bit short address of the device whose IEEE address is known.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator)
rsi_zigb_join_network() must be called before this API (End-Device, Router)

Parameters

Parameters	Data type	Description
shortAddress	uint16_t	This is the pointer to a location of short address whose IEEE address is to be determined.
RequestType	BOOL	This is the boolean value. if TRUE comes, it indicates single device response and if FALSE comes, it indicates extended device response.
StartIndex	uint8_t	This is the index of the first child to list in the response. Please ignore this parameter, if the RequestType is a single device.
APSAckRequired	BOOL	This is the boolean value. if TRUE comes, it indicates APS ack is required

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure- g_FAILURE_c Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint16_t parent_short_addr;
  uint8    request_type;
  uint8_t  start_index;
  uint8_t  aps_ack_required;
rsi_zigb_zdp_send_ieee_addr_request(parent_short_addr,request_type,st
art_index,aps_ack_required);
```

rsi_zigb_zdp_send_device_announcement

Prototype

```
int16_t rsi_zigb_zdp_send_device_announcement(void);
```

Description
This API allows the application to send a broadcast for a ZDO Device announcement. Normally, it is NOT required to call this as the stack as it automatically sends a device announcement during joining or rejoining, as per the spec. However, if the device wishes to broadcast device announcement it can do through this call.

Precondition
rsi_zigb_join_network() must be called before this API.
Suported for Router and End Device.

Parameters
None.

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure- ZigBee_Device_Down Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_zdp_send_device_announcement();
```

rsi_zigb_send_match_descriptors_request

Prototype

```
int16_t rsi_zigb_send_match_descriptors_request(
 uint16_t shortAddress,
 uint16_t ProfileId,
 uint8_t *InClusterList,
 uint8_t InClusterCnt,
 uint8_t *OutClusterList,
 uint8_t OutClusterCnt,
 BOOL APSAckRequired,
 uint16_t dstAddress);
```

Description
This API allows the application to send a match descriptor request to a destination device.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
shortAddress	uint16_t	This is the device whose matching endpoints are desired. The request can be sent unicast or broadcast ONLY to the "RX-on-when-idle- address" (0xFFFD). If sent as a broadcast, any node that has matching endpoints will send a response.

Parameters	Data type	Description
ProfileId	uint16_t	This is the application profileId to match
InClusterList	uint8_t *	This is the pointer to list of input clusters.
InClusterCnt	uint8_t	This is the number of input clusters.
OutClusterList	uint8_t *	This is the pointer to the list of output clusters.
OutClusterCnt	uint8_t	This is the number of output clusters.
APSAckRequired	BOOL	This is the boolean value. if TRUE comes, APS ack is required
dstAddress	uint16_t	This is the destination short address.

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure- ZigBee_Failure Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, please refer to ZigBee Error Codes table for description.

Example

```
See <SW Package Path>/host/sAPIs/examples/zigbee/switch/
rsi_zb_app.cstatus = rsi_zigb_send_match_descriptors_request(0x0000,
  app_info->DeviceSimpleDesc->app_profile_id,
  (uint8_t*)app_info->DeviceSimpleDesc->p_incluster_list,
  app_info->DeviceSimpleDesc->incluster_count,
  (uint8_t*) app_info->DeviceSimpleDesc->p_outcluster_list,
        app_info->DeviceSimpleDesc->outcluster_count, 1, 0xFFFF);
```

rsi_zigb_active_endpoints_request

Prototype

```
int16_t rsi_zigb_active_endpoints_request(uint16_t shortAddress,
uint8_t APSAckRequired)
```

Description
This API allows the application to send ZDP Active Endpoint request.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
shortAddress	uint16_t	This is the device short address whose active endpoints needs to be obtained.
APSAckRequired	BOOL	This is the boolean value. if TRUE comes, APS ack is required

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure- g_FAILURE_c Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint16_t                parent_short_addr;
uint8_t                 aps_ack_required;
rsi_zigb_active_endpoints_request( parent_short_addr,
aps_ack_required);
```

rsi_zigb_zdp_send_power_descriptor_request

Prototype

```
int16_t rsi_zigb_zdp_send_power_descriptor_request(uint16_t
shortAddress, uint8_t APSAckRequired)
```

Description
This API allows the application to power the descriptor request.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
shortAddress	uint16_t	This is the device short address whose power descriptor needs to be obtained.
APSAckRequired	uint8_t	This is the boolean value. if TRUE comes, APS ack is required

Return Value

Value	Description
0	Successful execution of the command
Non Zero	Failure- g_FAILURE_c Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
```

rsi_zigb_zdp_send_node_descriptor_request

Prototype

```
int16_t rsi_zigb_zdp_send_node_descriptor_request(uint16_t
shortAddress, uint8_t APSAckRequired);
```

Description
This API allows the application to the node descriptor request.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
shortAddress	uint16_t	This is the device short address whose node descriptor needs to be obtained.
APSAckRequired	uint8_t	This is the boolean value. if TRUE comes, APS ack is required

Return Value

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	Failure- g_FAILURE_c Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint16_t                parent_short_addr;uint8_t
aps_ack_required;rsi_zigb_zdp_send_node_descriptor_request(parent_sho
rt_addr,aps_ack_required);
```

rsi_zigb_simple_descriptor_request

Prototype

```
int16_t rsi_zigb_simple_descriptor_request(uint16_t shortAddress,
uint8_t EndPointId);
```

Description
This API allows the application to request for the simple descriptor for a target device.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
shortAddress	uint16_t	This is the device's short address whose simple descriptor needs to be obtained. Range:0x0000 – 0xFFFF.
APSAckRequired	uint8_t	This is the boolean value. if TRUE comes, APS ack is required

Return Value

Value	Description
0	Successful execution of the command

Value	Description
Non Zero	Failure- g_FAILURE_c Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint16_t                parent_short_addr;
uint8_t                 end_pont_id;
rsi_zigb_simple_descriptor_request( parent_short_addr, end_pont_id);
```

rsi_zigb_get_address_map_table_entry

Prototype

```
typedef struct APSME_Address_Map_Table_Tag {
 uint8_t a_IEEE_addr[8];
 uint16_t nwk_addr;
 }APSME_Address_Map_Table_t;
 APSME_Address_Map_Table_t*
rsi_zigb_get_address_map_table_entry(uint8_t Index);
```

Description
This API allows the application to get the address map table entry for the specified index.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
Index	uint8_t	This specifies which entry in the Address Map table.

Structure: APSME_Address_Map_Table_t

Name	type	Range	Description
a_IEEE_addr	uint8_t[8]	-	This indicates extended 64-bit IEEE address.
nwk_addr	uint16_t	0x0000 – 0xffff.	This is the 16 bit network address.

Return Value

Value	Description
	Success - Address map table is updated
Non Zero	Failure- The 64 bit field is updated with all 0xFFs Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
APSME_Address_Map_Table_t *apsme;
uint8_t Index;
apsme = rsi_zigb_get_address_map_table_entry(Index);
```

11.2 Data Interface

11.2.1 rsi_zigb_send_unicast_data

Prototype

```
typedef enum
 {
 ZigBee_Outgoing_Direct,
 ZigBee_Via_Address_Map,
 ZigBee_Via_Binding_Table,
 ZigBee_Via_Multicast,
 ZigBee_Broadcast
 }ZigBee_Outgoing_Msg_Type;
 typedef struct {
 uint8_t DestEndpoint;
 uint8_t SrcEndpoint;
 ProfileID ProfileId;
 ClusterID ClusterId;
 uint8_t AsduLength;
 uint8_t TxOptions;
 uint8_t Radius;
 uint8_t aReserved[0x31];
 uint8_t aPayload[0x33];
 }ZigBeeAPSDEDataRequest_t;
 typedef union Address_Tag {
 uint16_t short_address;
 uint8_t IEEE_address[8];
 } Address;
 int16_t rsi_zigb_send_unicast_data(
 ZigBee_Outgoing_Msg_Type msgType,
 Address DestAddress,
 ZigBeeAPSDEDataRequest_t *pAPSDERequest);
```

Description
This API allows the application to initiate APSDE data request to the specified destination address.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
msgType	uint8_t	This is the type of transmission taking place.
DestAddress	Address	This is the address of the destination device
pAPSDERequest	ZigBeeAPSDEDataRequest_t *	This is the pointer to the memory where data request frame is stored.

Name	type	Range	Description
short_address	uint16_t	0x00 –0xff	This is the 16-bit short address.
IEEE_address	uint8_t[8]	-	This is the 64-bit IEEE extended address.

Structure: Address
Structure: ZigBeeAPSDEDataRequest_t

Name	type	Range	Description
DestEndpoint	uint8_t	0x00 –0xff	This parameter shall be present if, and onlyif, the DstAddrMode parameter has a value of 0x02 or 0x03 and, if present, shall be either the number of the individual endpoint of the entity to which the ASDU is being transferred or the broadcast endpoint (0xff).
SrcEndpoint	uint8_t	0x00 – 0xfe	The individual endpoint of the entity from which the ASDU is being transferred.
ProfileId	uint16_t	0x0000 – 0xffff	The identifier of the profile for which this frame is intended.
ClusterId	uint16_t	0x0000 – 0xffff	The identifier of the object for which this frame is intended
AsduLength	uint8_t	0x00 - 256*(NsduLength - apscMinHeader Overhead)	The number of octets comprising the ASDU to be transferred. The maximum length of an individual APS frame payload is given as NsduLength -*apscMinHeaderOverhead*. Assuming fragmentation is used, there can be 256 such blocks comprising a single maximum sized ASDU.
TxOptions	uint8_t	0000 0000 – 00011111	The transmission options for the ASDU to be transferred. These are a bitwise OR of one or more of the following: 0x01 = Security enabled transmission 0x02 = Use NWK key 0x04 = Acknowledged transmission 0x08 = Fragmentation permitted 0x10 = Include extended nonce in APSsecurity frame
Radius	uint8_t	0x00-0xff	The distance, in hops, that a transmitted frame will be allowed to travel through the network.
aReserved	uint8_t[0x31]	-	Reserved bytes for payload.
aPayload	uint8_t[0x33]	-	Payload.

Return Value

Value	Description
	Success - ZigBee_Success
Non Zero	Failure- ZigBee_Invalid_Argument/ ZigBee_No_Buffer Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error codes table for description.

Example

```
status = rsi_zigb_send_unicast_data(ZigBee_Outgoing_Direct,
                  DestAddress, &APSDERequest);
```

11.2.2 rsi_zigb_send_group_data

Prototype

```
typedef uint16_t GroupID;
 int16_t rsi_zigb_send_group_data( GroupID GroupAddress,
ZigBeeAPSDEDataRequest_t * pAPSDERequest);
```

Description

This API allows the application to initiate APSDE data request to the specified Group address.

Precondition

rsi_zigb_form_network() must be called before this API (Coordinator)
rsi_zigb_join_network() must be called before this API (End-Device, Router)

Parameters

Parameters	Data type	Description
GroupAddress	GroupID	This indicates the group id to which the data is transmitted.
pAPSDERequest	ZigBeeAPSDEDataRequest_t *(refer rsi_zigb_send_unicast_data for the structure definition)	This is the pointer to the memory where data request frame is stored.

Return Value:

Value	Description
	Success - ZigBee_Success

Value	Description
Non Zero	Failure- ZigBee_Invalid_Argument/ ZigBee_No_Buffer Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
ZigBeeAPSDEDataRequest_t  APSDEDataReq;
rsi_zigb_send_group_data(0x0000,APSDEDataReq);
```

11.2.3 rsi_zigb_send_broadcast_data

Prototype

```
int16_t rsi_zigb_send_broadcast_data(
 ZigBeeAPSDEDataRequest_t * pAPSDERequest);
```

Description
This API allows the application to broadcast APSDE data request.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
pAPSDERequest	ZigBeeAPSDEDataRequest_t * (refer rsi_zigb_send_unicast_data for the structure definition)	This is the pointer to the memory where data request frame is stored.

Return Value

Value

Value	Description
	Success - ZigBee_Success
Non Zero	Failure- ZigBee_Invalid_Argument/ ZigBee_No_Buffer Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
ZigBeeAPSDEDataRequest_t APSDEDataReq;
rsi_zigb_send_broadcast_data(APSDEDataReq);
```

11.2.4 rsi_zigb_get_max_aps_payload_length

Prototype

```
int16_t rsi_zigb_get_max_aps_payload_length(void);
```

Description
This API allows the application to get the maximum size of the payload that the Application Support sub-layer can accept. The size depends on the security level in use. The value is the same as that found in the node descriptor.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters
None.

Return Value

Value	Description
	Success - maximum APS payload length
Non Zero	Failure- Negative Value Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_get_max_aps_payload_length();
```

11.3 Security Interface

KeyType	Value
g_Trust_Center_Master_Key_c (Reserved)	**0x0**
g_Network_Key_c	**0x1**
g_Application_Master_Key_c (Reserved)	**0x2**
g_Link_Key_c (Reserved)	**0x3**
g_Trust_Center_Link_Key_c	**0x4**

Key Types

11.3.1 rsi_zigb_get_key

Prototype:
typedef enum Security_Key_Types_Tag {
g_Trust_Center_Master_Key_c,
g_Network_Key_c,
g_Application_Master_Key_c,
g_Link_Key_c,
g_Trust_Center_Link_Key_c,
g_Next_Network_Key_c
} Security_Key_Types;
int16_t rsi_zigb_get_key(Security_Key_Types keytype);

Description
This API allows the application to gets the specified key and its associated data. This can retrieve the Link Key, Current Network Key, or Next Network Key.

Precondition
rsi_zigb_init_stack() must be called before this API.

Parameters

Parameters	Data type	Description
keytype	Security_Key_Types	This is the key type.

Return Value

Value	Description
	Success - ZigBee_Success
Non Zero	Failure- ZigBee_Invalid_Argument/ g_NO_KEY_c/ ZigBee_Failure Returns a negative value if command issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_get_key_table_entry(api_test_var->key_index, &api_test_var-
>KeyStruct);
```

11.3.2 rsi_zigb_have_link_key

Prototype

```
int16_t rsi_zigb_have_link_key(uint8_t *pRemoteDeviceIEEEAddr);
```

Description
This API allows the application to check whether a link key is available or not for securing messages sent to the remote device.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator)
rsi_zigb_join_network() must be called before this API (End-Device, Router)

Parameters

Parameters	Data type	Description
pRemoteDeviceIEEEAddr	uint8_t*	This is the long address of some other device in the network.

Return Value

Value	Description
	Success - g_TRUE_c
Non Zero	Failure- g_FALSE_c Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_have_link_key(api_test_var->parent_ieee);
```

11.3.3 rsi_zigb_request_link_key

Prototype

```
int16_t rsi_zigb_request_link_key(uint8_t* TrustCenterIEEEAddr,
uint8_t* PartnerIEEEAddr);
```

Description
This API allows the application to get the link key for the specified IEEE address.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters:

Parameters	Data type	Description
TrustCenterIEEEAddr	uint8_t*	This is the IEEE address of the Trust Centre device.

Parameters	Data type	Description
PartnerIEEEAddr	uint8_t*	This is the IEEE address of the partner device.

Return Value

Value	Description
	ZigBee_Success
Non Zero	Failure- ZigBee_Failure/ ZigBee_Invalid_Argument Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_request_link_key(api_test_var->parent_ieee , api_test_var->parent_ieee);
```

11.3.4 rsi_zigb_get_key_table_entry

Prototype

```
typedef enum Security_Key_Types_Tag {
 g_Trust_Center_Master_Key_c,
 g_Network_Key_c,
 g_Application_Master_Key_c,
 g_Link_Key_c,
 g_Trust_Center_Link_Key_c,
 g_Next_Network_Key_c
 } Security_Key_Types;
 typedef enum ZigBeeKeyStructBitmask_Tag {
 g_Key_Has_Sequence_Number_c = 0x01,
 g_Key_Has_Outgoing_Frame_Counter_c = 0x02,
 g_Key_Has_Incoming_Frame_Counter_c = 0x04,
 g_Key_Has_Partner_IEEE_Addr_c = 0x08,
 g_Key_Is_Authorized_c = 0x10
 } ZigBeeKeyStructBitmask_t;
 typedef struct ZigBeeKeyStructure_Tag {
 ZigBeeKeyStructBitmask_t bitmask;
 Security_Key_Types type;
 uint8_t key[16];
 uint32_t outgoingFrameCounter;
 uint32_t incomingFrameCounter;
 uint8_t sequenceNumber;
 uint8_t apartnerIEEEAddress[g_EXTENDED_ADDRESS_LENGTH_c];
 }ZigBeeKeyStructure_t;
 int16_t rsi_zigb_get_key_table_entry (uint8_t Index,
ZigBeeKeyStructure_t *keyStruct);
```

Description
This API allows the application to get the link key for the specified IEEE address.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
Index	uint8_t	This is the index in the key table of the entry to get.
keyStruct	ZigBeeKeyStructure_t *	This is a pointer to the location of an **ZigBeeKeyStructure_t** that will contain the results retrieved by the stack.

Parameters: ZigBeeKeyStructBitmask_t

Parameters	Description
g_Key_Has_Sequence_Number_c	This indicates that the key has a sequence number associated with Network Key
g_Key_Has_Outgoing_Frame_Counter_c	This indicates that the key has an outgoing frame counter
g_Key_Has_Incoming_Frame_Counter_c	This indicates that the key has an incoming frame counter
g_Key_Has_Partner_IEEE_Addr_c	This indicates that the key has an associated Partner IEEE address and the corresponding value within the ZigBeeKeyStructure_t has been populated with the data
g_Key_Is_Authorized_c	This indicates the key is authorized for use in APS data messages. If the key is not authorized for use in APS data messages it has not yet gone through a key agreement protocol, such as CBKE (i.e. ECC)

Structure: ZigBeeKeyStructure_t

Name	type	Range	Description
bitmask	ZigBeeKeyStructBitmask_t	—	This bitmask indicates the presence of information about that particular field present in bitmask.
type	Security_Key_Types	—	This is the type of key sent from host. It is one of key from the defined structure Security_Key_Types
key	uint8_t[16]	—	This is the actual value of the key to be used for Encryption and Decryption.
outgoingFrameCounter	uint32_t	0x00000000-0xffffffff	This is the outgoing frame counter associated with the key. It will contain valid data based on the ZigBeeKeyStructBitmask_t.
incomingFrameCounter	uint32_t	0x00000000-0xffffffff	This is the incoming frame counter associated with the key. It will contain valid data based on the ZigBeeKeyStructBitmask_t
sequenceNumber	uint8_t	0x00-0xff	This is the sequence number associated with the key.
apartnerIEEEAddress	uint8_t[8]	0x00000000-0xffffffff	This is the Partner IEEE Address associated with the key (Link Key)

Return Value

Value	Description
	ZigBee_Success

Value	Description
Non Zero	Failure- ZigBee_Invalid_Argument/ g_NO_KEY_c/ ZigBee_Failure Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_get_key_table_entry(api_test_var->key_index, &api_test_var-
>KeyStruct);
```

11.3.5 rsi_zigb_set_key_table_entry

Prototype

```
int16_t rsi_zigb_set_key_table_entry( uint8_t index,
 uint8_t * pIEEEAddress,
 BOOL linkKey,
 uint8_t * pKeyData );
```

Description
This API allows the application to set an entry in the key table.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
index	uint8_t	This is the index in the key table of the entry to set.
pIEEEAddress	uint8_t*	This is the address of the partner device associated with the key.
linkKey	BOOL	This is a boolean indicating whether this is a Link or Master Key.
pKeyData	uint8_t*	This is a pointer to the key data associated with the key entry.

Return Value

Value	Description
	ZigBee_Success

Value	Description
Non Zero	Failure- m_NO_ENTRY_c Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_set_key_table_entry(0, api_test_info-
>SelfAddress.IEEE_address, g_Network_Key_c, api_test_var-
>link_key_ptr);
```

11.3.6 rsi_zigb_add_or_update_key_table_entry

Prototype

```
int16_t rsi_zigb_add_or_update_key_table_entry(
 uint8_t *pIEEEAddress,
 BOOL linkKey,
 uint8_t *pKeyData,
 uint8_t *indx);
```

Description
This API allows the application to add a new entry in the key table or updates an existing entry with a new key.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router)..

Parameters

Parameters	Data type	Description
pIEEEAddress	uint8_t*	This is the IEEE Address of the partner device that shares the key.
linkKey	BOOL	This is a boolean indicating whether this is a Link or Master Key.
pKeyData	uint8_t*	This is a pointer to the actual key data.
indx	uint8_t	This is the index updated on getting response.

Return Value

Value	Description
	ZigBee_Success

Value	Description
Non Zero	Failure- ZigBee_Invalid_Argument/ ZigBee_Failure Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_add_or_update_key_table_entry(api_test_info-
>SelfAddress.IEEE_address, 1, api_test_var->link_key_ptr, 0);
```

11.3.7 rsi_zigb_find_key_table_entry

Prototype

```
int16_t rsi_zigb_find_key_table_entry(uint8_t * pIEEEAddress,
 BOOL linkKey);
```

Description
This API allows the application to search the key table and find an entry matching the specified IEEE address and key type.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters:

Parameters	Data type	Description
pIEEEAddress	uint8_t*	This is the IEEE Address of the partner device that shares the key. To find the first empty entry pass in an address of all zeros.
linkKey	BOOL	This is a boolean indicating whether to search for an entry containing a Link or Master Key.

Return Value

Value	Description
	ZigBee_Success
Non Zero	Failure- m_NO_ENTRY_c Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint8_t parent_ieee[8];
g_Network_Key_c =1
rsi_zigb_find_key_table_entry(parent_ieee, g_Network_Key_c);
```

11.3.8 rsi_zigb_erase_key_table_entry

Prototype

```
int16_t rsi_zigb_erase_key_table_entry(uint8_t index);
```

Description
This API allows the application to clear a single entry in the key table.

Precondition:
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters:

Parameters	Data type	Description
index	uint8_t	This is the index of the trust center link key.

Return Value

Value	Description
	ZigBee_Success
Non Zero	Failure- m_NO_ENTRY_c Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee error code table for description.

Example

```
rsi_zigb_erase_key_table_entry(0x1);
```

11.4 Binding Interface

11.4.1 rsi_zigb_set_binding_entry

Prototype:

```
typedef struct ZDP_Bind_Request_Tag {
 uint8_t a_src_addr[8];
 uint8_t src_endpoint;
 uint8_t a_cluster_id[2];
 uint8_t dest_addr_mode;
 uint8_t a_dest_addr[8];
 uint8_t dest_endpoint;
 }ZDP_Bind_Request_t;
 int16_t rsi_zigb_set_binding_entry(
 ZDP_Bind_Request_t * pSetBindingEntry);
```

Description
This API allows the application to set an entry in the binding table by copying the structure pointed to by pSetBindingEntry into the binding table.

Precondition
// TODO: Create Example

Parameters

Parameters	Data type	Description
pSetBindingEntry	ZDP_Bind_Request_t *	This indicates the pointer to the binding entry which need to be set in the given index.

Structure: ZDP_Bind_Request_t

Name	type	Range	Description
a_src_addr	uint8_t	A valid 64-bit IEEE address	The IEEE address for the source.
src_endpoint	uint8_t	0x01-0xfe	The source endpoint for the binding entry.
a_cluster_id	uint8_t	0x0000-0xffff	The identifier of the cluster on the source device that is bound to the destination.
dest_addr_mode	uint8_t	0x00-0xff	The addressing mode for the destination address used in this command. This field can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddress and DstEndp not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddress and DstEndp present 0x04 – 0xff = reserved

Name	type	Range	Description
a_dest_addr	uint8_t	As specified by the DstAddrMode field	The destination address for the binding entry.
dest_endpoint	uint8_t	0x01-0xfe	This field shall be present only if the DstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.

Return Value

Value	Description
	g_SUCCESS_c
Non Zero	Failure- g_FAILURE_c Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
ZDP_Bind_Request_t      *SetBindingEntry;
rsi_zigb_set_binding_entry(SetBindingEntry);
```

11.4.2 rsi_zigb_get_binding_indices

Prototype

```
int16_t rsi_zigb_get_binding_indices(uint8_t * binding_indices);
```

Description
This API allows the application to read the active binding indices.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
binding_indices	uint8_t*	This is the pointer to the list of binding indices of each uint8_t size.

Return Value

Value	Description
	ZigBee_Success

Value	Description
Non Zero	Failure- ZigBee_Invalid_Argument Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint8_t *binding_indices;
rsi_zigb_get_binding_indices(binding_indices);
```

11.4.3 rsi_zigb_delete_binding

Prototype

```
int16_t rsi_zigb_delete_binding(uint8_t bindIndex);
```

Description
This API allows the application to delete an entry in the binding table for the specified index.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator)
rsi_zigb_join_network() must be called before this API (End-Device, Router)

Parameters

Parameters	Data type	Description
bindIndex	uint8_t	This indicates the index which needs to be deleted.

Return Value

Value	Description
	ZigBee_Success
Non Zero	Failure- g_ZDP_Not_Permitted_c Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_delete_binding(0);
```

11.4.4 rsi_zigb_is_binding_entry_active

Prototype

```
int16_t rsi_zigb_is_binding_entry_active(uint8_t bindIndex);
```

Description
This API allows the application to check whether the binding entry is active or not.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Description
bindIndex	uint8_t	This is the index of a binding table entry.

Return Value

Value	Description
	Success : g_TRUE_c
Non Zero	Failure- g_FALSE_c Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint8_t Dest_EP;
rsi_zigb_is_binding_entry_active(Dest_EP);
```

11.4.5 rsi_zigb_clear_binding_table

Prototype

```
int16_t rsi_zigb_clear_binding_table(void);
```

Description
This API allows the application to clear all the binding table entries.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters
None.

Return Value

Value	Description
	ZigBee_Success
Non Zero	Failure- g_ZDP_Not_Permitted_c Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
rsi_zigb_clear_binding_table();
```

11.4.6 rsi_zigb_bind_request

Prototype

```
int16_t rsi_zigb_bind_request(
 uint16_t shortAddress,
 uint8_t *pIEEEAddrOfSource,
 uint8_t sourceEndpoint,
 uint16_t ClusterId,
 uint8_t destAddrMode,
 Address destAddress,
 uint8_t destinationEndpoint,
 BOOL APSAckRequired);
```

Description
This API allows the application to set an entry in the binding table.

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Parameters

Parameters	Data type	Range	Description
shortAddress	uint16_t	0x0000 - 0xffff	The device short address .

Parameters	Data type	Range	Description
pIEEEAddrOfSource	uint8_t[8]	A valid 64-bit IEEE address	The IEEE address for the source.
sourceEndpoint	uint8_t	0x01-0xfe	The source endpoint for the binding entry.
ClusterId	uint16_t	0x0000-0xffff	The identifier of the cluster on the source device that is bound to the destination.
destAddrMode	uint8_t	0x00-0xff	The addressing mode for the destination address used in this command. This field can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddress and DstEndp not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddress and DstEndp present 0x04 – 0xff = reserved
destAddress	Address	As specified by the DstAddrMode field	The destination address for the binding entry.
destinationEndpoint	uint8_t	0x01-0xfe	This field shall be present only if the DstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.
APSAckRequired	BOOL	0x00 – 0x01	TRUE (0x00) indicates APS ack is required.

Return Value

Value	Description
	ZigBee_Success
Non Zero	Failure- g_ZDP_Not_Permitted_c Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint16_t                parent_short_addr;
  uint8_t IEEE_address[8];
  uint8_t Source_EP;
  uint16_t                ClusterId;
  uint8_t                 DestAddrMode;
  Address ParentAddress;
  uint8_t Dest_EP;
  uint8_t                 APSAckRequired;
rsi_zigb_bind_request(parent_short_addr,IEEE_address,
Source_EP,ClusterId,DestAddrMode,ParentAddress,Dest_EP,APSAckRequired
);
```

11.4.7 rsi_zigb_unbind_request

Prototype:

```
int16_t rsi_zigb_unbind_request(
 uint16_t shortAddress,
 uint8_t *pIEEEAddrOfSource,
 uint8_t sourceEndpoint,
 uint16_t ClusterId,
 uint8_t destAddrMode,
 Address destAddress,
 uint8_t destinationEndpoint,
 BOOL APSAckRequired);
```

Precondition
rsi_zigb_form_network() must be called before this API (Coordinator).
rsi_zigb_join_network() must be called before this API (End-Device, Router).

Description
This API allows the application to remove bind entry between pair of device.

Parameters

Parameters	Data type	Range	Description
shortAddress	uint16_t	0x0000 - 0xffff	The device short address .
pIEEEAddrOfSource	uint8_t[8]	A valid 64-bit IEEE address	The IEEE address for the source.
sourceEndpoint	uint8_t	0x01-0xfe	The source endpoint for the binding entry.
ClusterId	uint16_t	0x0000-0xffff	The identifier of the cluster on the source device that is bound to the destination.

Parameters	Data type	Range	Description
destAddrMode	uint8_t	0x00-0xff	The addressing mode for the destination address used in this command. This field can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddress and DstEndp not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddress and DstEndp present 0x04 – 0xff = reserved
destAddress	Address	As specified by the DstAddrMode field	The destination address for the binding entry.
destinationEndpoint	uint8_t	0x01-0xfe	This field shall be present only if the DstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.
APSAckRequired	BOOL	0x00 – 0x01	TRUE (0x00) indicates APS ack is required.

Return Value

Value	Description
	ZigBee_Success
Non Zero	Failure- ZigBee_Invalid_Argument/ ZigBee_Failure Returns a negative value if command is issued in wrong state and packet allocation gets failure -3 : if command issued in wrong state -4 : if packet allocation fails

If the return value is greater than 0, Please refer to ZigBee Error Codes table for description.

Example

```
uint16_t           parent_short_addr;
  uint8_t            IEEE_address[8];
  uint8_t            Source_EP;
  uint16_t           ClusterId;
  uint8_t            DestAddrMode;
  Address            ParentAddress;
  uint8_t            Dest_EP;
  uint8_t            APSAckRequired;
rsi_zigb_unbind_request(parent_short_addr,IEEE_address,
Source_EP,ClusterId,DestAddrMode,ParentAddress,Dest_EP,APSAckRequired
);
```

11.5 Callbacks

11.5.1 rsi_zigb_register_callbacks

Prototype

```
void
rsi_zigb_register_callbacks( rsi_zigb_app_scan_complete_handler_t
zigb_app_scan_complete_handler,
 rsi_zigb_app_energy_scan_result_handler_t
zigb_app_energy_scan_result_handler,
 rsi_zigb_app_network_found_handler_t zigb_app_network_found_handler,
 rsi_zigb_app_stack_status_handler_t zigb_app_stack_status_handler,
 rsi_zigb_app_incoming_many_to_one_route_req_handler_t
zigb_app_incoming_many_to_one_route_req_handler,
 rsi_zigb_app_handle_data_indication_t
zigb_app_handle_data_indication,
 rsi_zigb_app_handle_data_confirmation_t
zigb_app_handle_data_confirmation,
 rsi_zigb_app_child_join_handler_t zigb_app_child_join_handler
 );
```

Description
This API registers the GAP callbacks.

Parameters

Parameter	Prototype name	Description
zigb_app_scan_complete_handler	rsi_zigb_app_scan_complete_handler_t zigb_app_scan_complete_handler	Scan complete callback
zigb_app_energy_scan_result_handler	rsi_zigb_app_energy_scan_result_handler_t zigb_app_energy_scan_result_handler	Energy Scan callback
zigb_app_network_found_handler	rsi_zigb_app_network_found_handler_t zigb_app_network_found_handler	Network Found Callback
zigb_app_stack_status_handler	rsi_zigb_app_stack_status_handler_t zigb_app_stack_status_handler	Stack status Callback
zigb_app_incoming_many_to_one_route_req_handler	rsi_zigb_app_incoming_many_to_one_route_req_handler_t zigb_app_incoming_many_to_one_route_req_handler	Route request callback
zigb_app_handle_data_indication	rsi_zigb_app_handle_data_indication_t zigb_app_handle_data_indication	Data indication Callback
zigb_app_handle_data_confirmation	rsi_zigb_app_handle_data_confirmation_t zigb_app_handle_data_confirmation	Data Conirmation Callback

Parameter	Prototype name	Description
zigb_app_child_join_handler	rsi_zigb_app_child_join_handler_t zigb_app_child_join_handler	Child join Callback

Return Values:
None

11.5.2 rsi_zigb_app_scan_complete_Handler

Prototype

```
void rsi_zigb_app_scan_complete_handler ( uint32_t channel, uint8_t
status );
```

Description
This API is called from the stack to inform status about the status of the current scan to the application.

Parameters

Parameters	Data type	Description
channel	uint32_t	This is the channel on which the scan is enabled.
status	uint8_t	This is the mac status obtained

MAC Scan Status	Value
g_MAC_Success_c	0x0
g_PAN_At_Capacity_c	0x1
g_PAN_Access_denied_c	0x2
g_MAC_Scan_In_Progress_c	0xAA
g_MAC_Beacon_Loss_c	0xE0
g_MAC_Channel_Access_Failure_c	0xE1
g_MAC_Denied_c	0xE2
g_MAC_Disable_TRX_Failure_c	0xE3
g_MAC_Failed_Security_Check_c	0xE4
g_MAC_Frame_Too_Long_c	0xE5
g_MAC_Invalid_GTS_c	0xE6
g_MAC_Invalid_Handle_c	0xE7
g_MAC_Invalid_Parameter_c	0xE8
g_MAC_No_ACK_c	0xE9

MAC Scan Status	Value
g_MAC_No_Beacon_c	0xEA
g_MAC_No_Data_c	0xEB
g_MAC_No_Short_Address_c	0xEC
g_MAC_Out_Of_CAP_c	0xED
g_MAC_PAN_ID_Conflict_c	0xEE
g_MAC_Realignment_c	0xEF
g_MAC_Transaction_Expired_c	0xF0
g_MAC_Transaction_Overflow_c	0xF1
g_MAC_TX_Active_c	0xF2
g_MAC_Unavailable_Key_c	0xF3
g_MAC_Unsupported_Attribute_c	0xF4
g_MAC_Missing_Address_c	0xF5
g_MAC_Past_Time_c	0xF6

ZigBee MAC Status

11.5.3 rsi_zigb_app_energy_scan_result_handler

Prototype

```
void rsi_zigb_app_energy_scan_result_handler( uint32_t
channel,uint8_t *pEnergyValue);
```

Description
This API is called from the stack to report RSSI value measured on the required channel to the application.

Parameters

Parameters	Data type	Description
channel	uint32_t	This is the channel on which the scan is enabled.
pEnergyValue	uint8_t*	This is a pointer to an array of energy values in 16 channels.

11.5.4 rsi_zigb_app_network_found_handler

Prototype

```
Struct {
 uint16_t shortPanId,
 uint8_t channel,
 uint8_t extendedPanId[8],
 uint8_t stackProfile,
 uint8_t nwkUpdateId
 bool allowingJoining,
 } ZigBeeNetworkDetails;
 void rsi_zigb_app_network_found_handler(ZigBeeNetworkDetails
networkInformation);Description
```

This function is called from the application to get information about network found in the current channel.

Parameters

Parameters	Data type	Description
networkInformation	ZigBeeNetworkDetails	This is the channel on which the scan is enabled.

Structure: ZigBeeNetworkDetails

Parameters	Data type	Range	Description
shortPanId	uint16_t	0x0000 - 0xffff	The network's PAN identifier
channel	uint8_t	0x0000-0xffff	The 802.15.4 channel associated with the network.
extendedPanId	uint8_t[8]	A valid 64-bit IEEE address	The network's extended PAN identifier.
stackProfile	uint8_t	0x01-0x2	The Stack Profile associated with the network
nwkUpdateId	uint8_t	0x01-0x3	The instance of the Network
allowingJoining	BOOL	0 -1	Whether the network is allowing MAC associations.

11.5.5 rsi_zigb_app_stack_status_handler

Prototype

```
enum {
 ZigBeeNWKIsUp,
 ZigBeeNWKIsDown,
 ZigBeeJoinFailed,
 ZigBeeCannotJoinAsRouter,
 ZigBeeChangedNodeID,
 ZigBeeChangedPANID,
 ZigBeeChangedChannel,
 ZigBeeNoBeacons,
 ZigBeeReceivedKeyInClear,
 ZigBeeNoNWKKeyReceived,
 ZigBeeNoLinkKeyReceived,
 ZigBeePreconfiguredKeyRequired,
 ZigBeeChangedManagerAddress
 } ZigBeeNWKStatusInfo;
 void rsi_zigb_app_stack_status_handler(ZigBeeNWKStatusInfo
*statusInfo);
```

Description
This callback is invoked by the ZigBee Stack to indicate any kind of Network status to the application. For example: upon establishing the network, this function shall be called by the stack to indicate status ZigBeeNetworkIsUp. If the device leaves the network, a status of ZigBeeNWkisDown status is indicated via this function call.
.
Parameters

Parameters	Data type	Description
statusInfo	ZigBeeNWKStatusInfo _t	Stack status is one of the status mentioned in below table.

enum: ZigBeeNWKStatusInfo

Parameters	Description
ZigBeeNWKIsUp	This indicates that Network is formed or joined successfully.
ZigBeeNWKIsDown	This indicates that NWK formation failed or the device left the network.
ZigBeeJoinFailed	This indicates that network join failed
ZigBeeCannotJoinAsRouter	This indicates that network was unable to start as Router.
ZigBeeChangedNodeID	This indicates that PANID is changed after resolving PAN ID conflict.
ZigBeeChangedChannel	This indicates that the channel is changed due to frequency agility mechanism
ZigBeeReceivedKeyInClear	This indicates the Network Key is received is inclear.
ZigBeeNoNWKKeyReceived	This indicates no Network key is received.
ZigBeeNoLinkKeyReceived	This indicates no Link key is received.

Parameters	Description
ZigBeePreconfiguredKeyRequired	This indicates Preconfigured link key is required.
ZigBeeChangedManagerAddress	This indicates network manager changed.

11.5.6 rsi_zigb_app_child_join_handler

Prototype:

```
void rsi_zigb_app_child_join_handler(uint16_t short_address,
 uint8_t joining);
```

Description
This callback is invoked is called from stack to intimate application about child device joining or leaving the network.

Parameters

Parameters	Data type	Description
short_address	ZigBeeNWKStatusInfo _t	Child Device's short .
joining	uint8_t	TRUE indicates child device joined. FALSE indicates child device left network.

11.5.7 rsi_zigb_app_handle_data_confirmation

Prototype

```
Struct APSE_Data_Confirmation_Tag {
 Address dest_address;
 uint8_t dest_addr_mode;
 uint8_t dest_endpoint;
 uint8_t src_endpoint;
 uint8_t status;
 }APSDE_Data_Confirmation_t;
 void rsi_zigb_app_handle_data_confirmation
(APSDE_Data_Confirmation_t *pDataConfirmation);
```

Description
This callback is invoked from stack to intimate application about child device joining or leaving the network.

Parameters:

Parameters	Data type	Description
dest_address	Address	This field indicates the individual device address or group address of the transmitted message

Parameters	Data type	Description
dest_addr_mode	uint8_t	This field indicates the destination address mode
dest_endpoint	uint8_t	This field indicates the destination endpoint to which the data frame was sent.
src_endpoint	uint8_t	This field indicates the source endpoint from which the data frame was originated.
status	uint8_t	This field indicates the status of data confirmation as shown in below Table .

Status	Description	Value
ZigBee_Success	No error occured while parsing the required API parameters	0x00
ZigBee_Failure	Error occured while parsing the required API parameters	0x01
ZigBee_Address_Table_ Entry_Is_Active	Requested address table entry is active	0x02
ZigBee_Table_Full	requested Stack table is full	0x03
ZigBee_No_Buffer	Out of buffers	0x04
ZigBee_Error_Fatal	Error occured in stack	0x05
ZigBee_Invalid_Argument	Argument passed for API is invalid	0x06
ZigBee_Fragment_Tx_Aborted	Transmission stopped inbetween in Fragmentation process	0x07
ZigBee_Fragment_Tx_Complet e	Transmission Complete in Fragmentation process	0x08
ZigBee_Fragment_Rx_Aborted	Receiving stopped inbetween in Fragmentation process	**0x09**
ZigBee_Fragment_Reception_ Completed	Receiving Complete in Fragmentation process	0x0a
ZigBee_Fragment_Message_To o_Long	Message Too Long	0x0b
ZigBee_Invalid_Call	Request might be not valid for the flashed device type or it is not in a state to receive call	0x0c
ZigBee_Device_Down	Device is not in network	0x0d
ZigBee_Unsupported	Feature not supported	0x0e
ZigBee_Unknown_Device_Type	Device type is unknown	0x0f
ZigBee_No_Key	No Requested Key	0x10
ZigBee_No_Entry	Entry in the table is empty	0x11
ZigBee_Index_Out_Of_Range	Accessing entry is out of range in the table	0x12
ZigBee_MAC_No_Data	No data pending	0x13
ZigBee_MAC_No_ACK	No ACK received	0x14

Status	Description	Value
ZigBee_Channel_Access_Failure	MAC Channel Access Failure	0x15
ZigBee_MAC_Unavailable_Key	MAC key unavailable	0x06
ZigBee_Failed_Security_Check	MAC Failed Security Check	0x07
ZigBee_MAC_Invalid_Parameter	MAC Invalid Parameter	0x08

ZigBee Data Confirmation Status

11.5.8 rsi_zigb_app_incoming_many_to_one_route_request_handler

Prototype

```
void rsi_zigb_app_incoming_many_to_one_route_req_handler( uint16_t
SourceAddr, uint8_t * pSrcIEEEAddr,uint8_t PathCost );
```

Description
This callback allows the Application to handle many to One Route Request

Parameters

Parameters	Data type	Description
SourceAddr	uint16_t	The short address of the concentrator that initiated the many-to-one route request.
pSrcIEEEAddr	uint8_t*	The IEEE address of the concentrator.
PathCost	uint8_t	The path cost to the concentrator.

11.5.9 rsi_zigb_app_handle_data_indication

Prototype

```
struct APSDE_Data_Indication_Tag {
 Address dest_address;
 uint8_t dest_addr_mode;
 uint8_t dest_endpoint;
 uint8_t src_addr_mode;
 Address src_address;
 uint8_t src_endpoint;
 profile_id_t profile_id;
 cluster_id_t cluster_id;
 uint8_t asdulength;
 uint8_t was_broadcast;
 uint8_t security_status;
 uint8_t link_quality;
 uint8_t a_asdu[1];
 } APSDE_Data_Indication_t;
 void rsi_zigb_api_test_data_indication_handler(
 APSDE_Data_Indication_t *pDataIndication);
```

Description
This callback allows the Application to handle data indication for the data request.

Parameters

Parameters	Data type	Description
pDataIndication	APSDE_Data_Indication_t*	Contains the data indication results.

Structure: **APSDE_Data_Indication_t**

Parameters	Description
dest_address	This field the destination address in the received message.
dest_addr_mode	This field indicates the destination address mode in the receivedmessage. This field takes one of the following values: 0x00 - Indirect data transmission (destination address and destination endpoint are not present) 0x01 - 16-bit group address 0x02 - 16-bit address of destination device 0x03 - 64-bit extended address of destination device 0x04 - 0xff - Reserved
dest_endpoint	This field indicates the destination endpoint in the received message
src_addr_mode	This field indicates the source address mode in the received message
src_address	This field indicates the source address from which the message is originated
profile_id	This field indicates the 16-bit profile ID

Parameters	Description
cluster_id	This field indicates the cluster ID
Asdulength	This field indicates the length of the data received.
was_broadcast	This field indicates whether the data frame is received through broadcast
security_status	This field indicates whether the received message was secured or not and type of the security applied.
link_quality	This field indicates the LQI of the received message.
a_asdu	This field points to the actual message received

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
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

Example	Description	Application path
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_a pis_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
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 ZigBee Example Applications

Example	Description	Application path
switch	This example demonstrates how to send on/off/toggle command to a Light coordinator.	sapis/examples/zigbee/ switch/rsi_zb_app.c

ZigBee Example Applications

13.1.4 Coexistence Example Applications

Example	Description	Application path
Controlling switch using TCP IP socket on remote side	This example demonstrates how to send on/ off/toggle command to a Light coordinator using TCP client socket. On/off commands are triggered by the TCP server.	sapis/examples/wlan_zigbee/ wlan_zigbee_switch/
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/

Example	Description	Application path
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
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

Error Codes (in hexadecimal format)	Description
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
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
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

Error Codes (in hexadecimal format)	Description
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
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

Error Codes (in hexadecimal format)	Description
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
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

Error Codes (in hexadecimal format)	Description
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
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

Error Codes (in hexadecimal format)	Description
0xFFC4	Unable to send tcp data
0xFFBC	Socket buffer too small
0xFFBB	Invalid content in the DNS response to the DNS Resolution query
0xFFBA	DNS Class error in the response to the DNS Resolution query
0xFFB8	DNS count error in the response to the DNS Resolution query
0xFFB7	DNS Return Code error in the response to the DNS Resolution query
0xFFB6	DNS Opcode error in the response to the DNS Resolution query
0xFFB5	DNS ID mismatch between DNS Resolution request and response
0xFFAB	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
0xFF36	Wait On Host feature not enabled
0xFF35	Store configuration checksum validation failed
0xFF33	TCP keep alive timed out

Error Codes (in hexadecimal format)	Description
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
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

Error Code	Description
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
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

Error Code	Description
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
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

Error Code	Description
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
0x4D08	Profile record full
0x4D09	Attribute record full
0x4D0A	BLE profile not found (profile handler invalid)

BLE Mode Error Codes

Error rendering macro 'captioneditem' : Plugin license error: EXPIRED

13.5 Zigbee Pro Stack Error codes

Status	Description	Status Code
ZigBee_Success/ g_SUCCESS_c	No Error occurred while parsing the required API parameters	0
ZigBee_Failure	Error occurred while parsing the required API parameters	1
g_FAILURE_c	Error occurred while parsing the required API parameters	0xFF
ZigBee_No_Buffer	Out of buffers	4
ZigBee_Invalid_ Argument	Argument passed in API is invalid	6
ZigBee_Invalid_Call	Request might not be valid for flashed device type	12
ZigBee_Device_Down	Device not a part of network.	13
g_FALSE_c	Boolean Operator	0
g_TRUE_c	Boolean Operator	1

Status	Description	Status Code
INVALID_SHORT_ADDRESS		0xFF
INVALID_NODE_ID		0xFFFF
ZigBeeUnknownDevice		16
g_INVALID_ADDRESS_c		0xFFFF
g_INVALID_CLUSTER_ID_c		0xFFFF
g_NO_KEY_c	if there is no valid entry is in table.	17
m_NO_ENTRY_c		0xFF
g_INVALID_ENDPOINT_ID_c		0xF1
ZigBee_Index_Out_Of_Range		0x12
g_ZDP_Not_Permitted_c		0x91

Zigbee Pro Stack Error codes

Error rendering macro 'captioneditem' : Plugin license error: EXPIRED

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 in Master Common API 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