RS9113 WiSeConnect^{TM} Simple API Guide

Version 0.8

May 2017

Redpine Signals, Inc.
2107 N. First Street, #680
San Jose, CA 95131.
Tel: (408) 748-3385
Fax: (408) 705-2019
Email: info@redpinesignals.com
Website: www.redpinesignals.com

Disclaimer:

Table Of Contents

Table of Figures

Table of Tables

1 Overview

RS9113-WiSeConnect module support WiFi/BT/BLE/Zigbee only and Coexistence modes (WiFi+BT/ WiFi+BLE/ WiFi+Zigbee) modes. This document contains description about RS9113-WiSeConnect Plus WLAN and Networking stack (TCP/IP) API's. The source code of API's, driver framework and reference application are provided in the software package. The developer can customize the application as per their application requirement. The API library is platform independent and is written in C language.

Note: These APIS are applicable to all the WiSeConnect variants like **WiSeConnect Plus, WiSeMCU** and **WYZBEE**. The term WiSeConnect refers to its appropriate variant.

2 Architecture

RS9113-WiSeConnect API's are designed in layers, where each Layer is independent and uses the service of underlying layers.

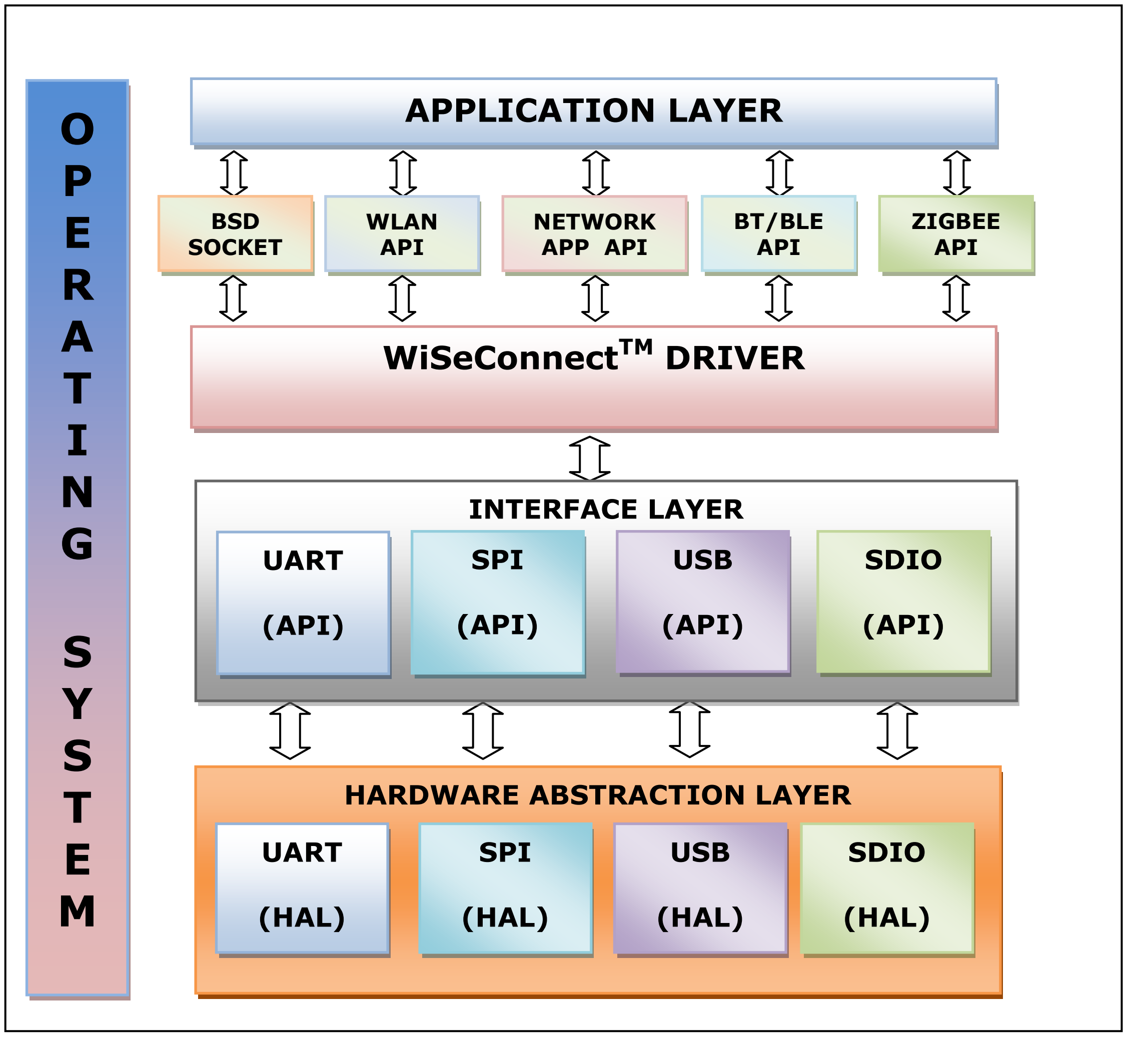


Figure 1 : API Architecture Diagram

Application Layer:

Application Layer contains application specific functionality. Application Layer need to call WiSeConnect^{TM} Driver API's to configure and operate the RS9113-WiSeConnect module.

WLAN API:

This Layer contains set of API's called from application to initialize and configure Wi-Fi Module. User is recommended to use given API's without any modification for transparent migration to API's enhancement in next releases.

BSD Socket API:

This Layer contains BSD Socket API Compliancy Wrapper supports some of the basic BSD Socket API calls. This API's can call from application to initialize and Configure embedded TCP/IP stack and perform data transfers.

WiSeConnect^{TM} Driver:

WiSeConnect^{TM} Driver software framework contains core functions to maintain state machines, command preparation, command response parsing.

Interface Specific API Layer:

RS9113-WiSeConnect Plus module support 4 different host interfaces (UART, SPI, USB).These API's are collection of functions specific to particular interface. Interface functions between Driver API Layer and Interface Specific API Layer are independent of Host interface used. So Application Layer can migrate to different interfaces without any change.

Note: SDIO, USB host interface is not supported in current release.

HAL API Layer:

Hardware Abstraction Layer API's are platform specific API's. User need to implement or modify these API's to their platforms.

Reference Applications:

Packages contain reference applications to operate the module in different modes. User can use these applications as reference or customize these applications as per their requirement.

3 Common API

This section contains common API to initialize driver and handle common features independent of module configuration mode.

3.1 rsi_driver_init

Prototype

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

Description

This API is used to initialize WiSeConnect^{TM} driver.

Parameters

Parameter	Description
buffer	Pointer to buffer from application. Driver used this buffer to hold driver control for it's operation.
length	Length of the buffer

Return Values

Actual buffer length required by the driver .

Success : If Actual buffer length is less than Provided Buffer length

On Failure :

Returns a non-zero value if buffer provided by application is less than the driver requirement. Return value represents the buffer required by the driver

Returns -1 , if UART initialization fails in SPI /UART mode

Returns -2 , if maximum sockets is greater than 10

3.2 rsi_device_init

Prototype

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

Description

This API to used to power cycle the module and set the boot up option for WiSeConnect^{TM} features. This API also initialize the module SPI.

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 `RSI_UPGRADE_IMAGE_I_FW` : To upgrade firmware file

Return Values

On Success : 0

On Failure : -1

3.3 rsi_bl_module_power_off

Prototype

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

Description

This API to used to poweroff the WiSeConnect™ device

Parameters

None

Return Values

On Success : 0

On Failure : -1

3.4 rsi_bl_module_power_on

Prototype

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

Description

This API to used to power on the WiSeConnect™ device

Parameters

None

Return Values

On Success : 0

On Failure : -1

3.5 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 to used to upgrade the firmware in the WiSeConnectTM device from the host. Firmware file is given to this API in chunks .

Each chunk must be multiple of 4096 bytes unless it is 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.

Parameters

Parameter	Description
firmware_image	pointer to firmware image buffer
fw_image_size	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

On Success : 0

On Failure : -1

3.6 rsi_wireless_init

Prototype

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

Description

This API to enable and initialize WiSeConnect^{TM} features.

Parameters

Parameter	Description
`opermode`	Operating mode 0 - Client mode 2 - Enterprise security client mode 6 - Access point mode 8 – Transmit test mode
`coex_mode`	Coexistence mode 0: WLAN only mode 3: WLAN & Zigbee coexistence mode 5: WLAN & BT coexistence mode 13: WLAN & BTLE coexistence mode.

Return Values

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x0025,0xFF73,0x002C,0xFF6E,0xFF6F,`

`0xFF70,0xFFC5`

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

3.7 rsi_wireless_deinit

Prototype

```
int32_t rsi_wireless_deinit()
```

Description

This API is used to de-initialize WiSeConnect^{TM} software feature. This API should be called before `rsi_wireless_init` if user wants to change previous configuration.

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

3.8 rsi_wireless_driver_task

Prototype

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

Description

This API is used to handle driver's events.

This API should be called in application main loop for non-OS platforms

Parameters

None

Return Values

None

3.9 rsi_wireless_antenna

Prototype

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

Description

This API is used to configure the antenna.

Parameters

Parameter	Description
`type`	`0- RF_OUT_2/Internal Antenna` is selected `1-RF_OUT_1/uFL connector` is selected.
`gain_2g`	Antenna gain in db for 2.4 GHz band and valid values are 0 to 10 .
`gain_5g`	Antenna gain in db for 5 GHz band and valid values are 0 to 10..

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0025, 0x002C`

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

4 WLAN API

This Section contains description about Wi-Fi API to initialize and configure module in Wi-Fi mode.

4.1 WLAN Core API

This section contains core API to configure the module in

4.1.1 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 is used to scan surrounding Wi-Fi networks.

Parameters

Parameter	Description
ssid	SSID of the Access points to be scanned. This parameter is to scan Wi-Fi network with given ssid. SSID size should be less than or equal to 32 bytes. SSID should be NULL to scan all Access points
chno	Channel number to perform scan, if 0 then module will scan in all channels.
result	Scanned Wi-Fi network information, this is an output parameter.
length	Length of the result buffer in bytes to hold scan results.

Channels supported

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

Table 1: 2.4GHz Band Channel Mapping

Channels supported in 5GHz band:

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

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

Table 2: 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`	Access point channel number
`security_mode`	Security mode `0` : Open `1` : WPA `2` : WPA2 `3` : WEP `4` : WPA Enterprise `5` : WPA2 Enterprise
`rssi_val`	RSSI value of Access Point
`network_type`	Type of network `1` : Infrastructure mode
`ssid`	SSID of access point

Structure Fields	Description
`bssid`	MAC address of 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`	Number of Access points scanned
`scan_info`	Information about scanned Access point in `rsi_scan_info_t` structure

Return Values

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

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

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

4.1.2 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 is used to scan surrounding Wi-Fi networks.A scan response handler is registered to get the response for scan

Parameters

Parameter	Description
`ssid`	SSID of the Access points to be scanned. This parameter is to scan Wi-Fi network with given `ssid`. SSID size should be less than or equal to 32 bytes. SSID should be NULL to scan all 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 come from the module Parameters `status, buffer , length` `Status:` response status If status is zero,scan response success `Buffer`: response buffer `Length`: response buffer length

Channels supported

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

Channel Number	chno
12	12
13	13
14	14

Table 3: 2.4GHz Band Channel Mapping

Channels supported in 5GHz band:

Channel Number	chno
All channels	0
36	36
40	40
44	44
48	48
100	100
104	104
108	108
112	112
116	116
132	132
136	136
140	140
149	149
153	153
157	157
161	161
165	165

Table 4: 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`	Access point channel number
`security_mode`	Security mode 0 : Open 1 : WPA 2 : WPA2 3 : WEP 4 : WPA Enterprise 5 : WPA2 Enterprise
`rssi_val`	RSSI value of Access Point
`network_type`	Type of network 1 : Infrastructure mode
`ssid`	SSID of access point
`bssid`	MAC address of 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

Structure Fields	Description
scan_count	Number of Access points scanned
scan_info	Information about scanned Access point in rsi_scan_info_t structure

Return Values

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if status in the handler is greater than 0, Error has occurd during scan

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

Please refer WLAN Error codes for description

4.1.3 rsi_wlan_connect

Prototype

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

Description

This API is used to connect to the specified WiFi network.

Parameters

Parameter	Description
ssid	SSID of Access point to connect, SSID should be less than or equal to 32 bytes.
sec_type	Security type of the Access point to connect 0 : RSI_OPEN, 1 : RSI_WPA, 2 : RSI_WPA2, 3 : RSI_WEP, 4 : RSI_WPA_EAP,

Parameter	Description
	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`	Point to a buffer 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 greater than equal to 8 and less than 64 bytes.

RSI_WEP	WEP keys should be in 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 in following format
	`typedef struct`
	`rsi_eap_credentials_s`
	`{`
	`  uint8_t username[64];`
	`  uint8_t password[128];`
	`}rsi_eap_credentials_t;`
	
	`username` : user name to be used in enterprise.
	`password` : password for given username.

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

RSI_WPA2_EAP	Enterprise credentials in following format
	`typedef struct rsi_eap_credentials_s`
	`{`
	`  uint8_t username[64];`
	`  uint8_t password[128];`
	`}rsi_eap_credentials_t;`
	
	`username` : user name to be used in enterprise.
	`password` : password for given username.

RSI_WPA_WPA2_MIXED	PSK string terminated with NULL. Length of PSK should be greater than equal to 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

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

```
0x0002,0x0003,0x0005,0x0008,0x0009,0x000A,0x000E,0x0014,
0x0015,0x0016,0x0019,0x001A,0x001E,0x0020,0x0021,0x0024,
0x0025,0x0026,0x0028,0x0039,0x003C,0x0044,0x0045,0x0046,
0x0047,0x0048,0x0049,0xFFF8
```

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

4.1.4 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 is used to connect to the specified WiFi network. A join response handler is registered to get the response for join

Parameters

Parameter	Description
`ssid`	SSID of Access point to connect, SSID should be less than or equal to 32 bytes.
`sec_type`	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`	Point to a buffer contains security information based on `sec_type`.
`join_response_handler`	This callback is called when the response for join has come from the module Parameters `status, buffer , length` `Status:` response status If status is zero join response success `Buffer`: response buffer

Parameter	Description
	`Length`: 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 greater than equal to 8 and less than 64 bytes.

RSI_WEP	WEP keys should be in 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 in following format
	`typedef struct rsi_eap_credentials_s`
	`{`
	`  uint8_t username[64];`
	`  uint8_t password[128];`
	`}rsi_eap_credentials_t;`
	
	`username` : user name to be used in enterprise.
	`password` : password for given username.

RSI_WPA2_EAP	Enterprise credentials in following format
	`typedef struct rsi_eap_credentials_s`
	`{`
	`  uint8_t username[64];`
	`  uint8_t password[128];`
	`}rsi_eap_credentials_t;`
	
	`username` : user name to be used in enterprise.
	`password` : password for given username.

RSI_WPA_WPA2_MIXED	PSK string terminated with NULL. Length of PSK should be greater than equal to 8 and less than 64 bytes.
RSI_WPA_PMK	PMK string, should be 32 bytes in length.

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

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if status in the handler is greater than 0, Error has occured during join

```
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 WLAN Error codes for description of above error codes.

4.1.5 rsi_wlan_execute_post_connect_cmds

Prototype

```
int32_t rsi_wlan_execute_post_connect_cmds(void)
```

Description

This API is used to enable Bgscan and roaming after connecting to Access Point

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0006,0x0021,0x002C,0x004A,0x0025,0x0026`

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

4.1.6 rsi_wlan_disconnect

Prototype

```
int32_t rsi_wlan_disconnect();
```

Description

This API is used to disconnect module from the connected Access point

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0006,0x0021,0x002C,0x0015`

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

4.1.7 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 is used load SSL/EAP certificate on WiSeConnect^{TM} module.

Parameters

Parameter	Description
`Certificate_type`	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`	Pointer to buffer which contain certificate
`certificate_length`	Certificate length

Return Values

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0015,0x0021,0x0025,0x0026,0x002C`

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

4.1.8 rsi_wlan_get_status

Prototype

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

Description

This API is used to check the status (specific error code) of error 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 error).

Parameters

None

Return Values

Returns the error code that previous occurred. If no error occurred, then returns 0

4.1.9 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 is used to start the module in Access point mode with the given configuration

Parameters

Parameter	Description
`ssid`	SSID of the Access Point. Length of the SSID should less than or equal to 32 bytes.
`channel`	channel number, refer following channels for valid channel numbers supported Table 1: 2.4GHz Band Channel Mapping Table 2: 5GHz Band Channel Mapping Note: channel number 0 is not valid in

Parameter	Description
	Access Point mode
`security_type`	Type of the Security mode the Access point to operate 0 : RSI_OPEN 1 : RSI_WPA 2 : RSI_WPA2 6 : RSI_WPA_WPA2_MIXED 11 : RSI_WPS_PUSH_BUTTON
`encryption_mode`	Type of the encryption mode 0 : RSI_NONE 1 : RSI_TKIP 2 : RSI_CCMP
`password`	PSK to use in security mode
`beacon_interval`	Beacon interval
`dtim_period`	DTIM period

Return Values

On Success: 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021,0x0025,0x002C,0x0026,0x004C,0x0028,0x001A,0x000A,0x001D

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

4.1.10 rsi_wlan_wps_push_button_event

Prototype

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

Description

This API is used to start WPS Push button in AP mode.

This API should be called after `rsi_wlan_ap_start` API has returned success.

Parameters

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

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021

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

4.1.11 rsi_wlan_wps_generate_pin

Prototype

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

Description

This API is used to generate WPS pin

Parameters

Parameter	Description
`wps_pin`	WPS pin is the 8 byte pin generated by the device .This is the output parameter

Parameter	Description
`length`	Length of the result buffer in bytes to hold WPS pin.

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0025, 0x002C, 0x0037, 0x0038

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

4.1.12 rsi_wlan_disconnect_stations

Prototype

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

Description

This API is used to disconnect the connected stations in AP mode.

Parameters

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

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0013, 0x0021, 0x002C, 0x0015

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

4.1.13 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 is to start discovery in WiFi-Direct mode.

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 remaining bytes filled with 0x00. Another Wi-Fi Direct device would see this name when it scans for Wi-Fi Direct nodes.
`channel`	operating channel number. The specified channel is

Parameter	Description
	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 WiFi Direct GO mode and Autonomous GO mode. `Note: ssid_post_fix` should be maximum of 23 bytes.
`psk`	Passphrase of a maximum length of 63 characters (a null character should be supplied to make it 64 bytes in the structure). This PSK is used if the module becomes a GO owner.
`wlan_wfd_discovery_notify_handler`	Asynchronous Message from module to Host, sent when module founds any Wi-Fi Direct node Parameters `status, buffer , length` `Status:` response status If status is zero wfd device response has some device information `Buffer`: response buffer `Length:` response buffer length
`wlan_wfd_connection_request_notify_handler`	Asynchronous message from Module to Host, sent when module receives a connection request from any remote Wi-Fi Direct node. Parameters `status, buffer , length` `Status:` response status If status is zero connection request has comefrom some device `Buffer`: response buffer `Length:` 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

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

0x001D, 0x0021, 0x002C, 0x0015

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

4.1.14 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 is used to connect to the specified WiFi-Direct device

Parameters

Parameter	Description
device_name	Device name of the WiFi Direct node to connect.
join_response_handler	This callback is called when the response for join has come from the module Parameters status, buffer , length Status: response status If status is zero join response success Buffer: response buffer Length: response buffer length

Return Values

On Success : 0

On Failure :

if return value is less 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 WLAN Error codes for description of above error codes.

4.1.15 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 is used to get required information based on the type of command.

Parameters

Parameter	Description

`cmd_type`	`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`	Response of requested command, this is an output parameter.
`length`	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]

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

cmd_type	Response structure
	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	

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;`

cmd_type	Response structure
	`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.

`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

cmd_type	Response structure
	`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

On Success: 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0025, 0x002c

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

4.1.16 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 is used to set the requested configuration based on the command type.

Parameters

Parameter	Description

`cmd_type`	Set command type:
	`1: RSI_SET_MAC_ADDRESS`
	`2: RSI_MULTICAST_FILTER`

`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 :`
	`mac_address :` MAC address to which filter has to be applied

Return Value

On Success: 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0025, 0x002c

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

4.1.17 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 is used to send the ping request to the target IP address.

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 the response for ping sent has come from the module Parameters `status, buffer , length` `Status:` response status If status is zero ping response success `Buffer`: response buffer

Parameter	Description
	`Length`: 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

On Success: 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0025, 0x002C,0x002F, 0xBB29, 0xFF74,

0x0015,0xBB21,0xBB4B,0xBB55

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

4.1.18 rsi_wlan_ power_save_profile

Prototype

int32_t rsi_wlan_power_save_profile(uint8_t psp_mode, uint8_t psp_type);

Description

This API is used to set power save profile in wlan mode.

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 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 connected sleep mode. In this sleep mode, SoC will go to sleep based on GPIO or Message, therefore handshake is required before sending data to the module. `RSI_SLEEP_MODE_8 (8)`: This is disconnected sleep mode. In this sleep mode, module will turn off the SoC. Since SoC is turn off, therefore handshake is required before sending data to the module.
`psp_type`	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.

Return Values

This API returns command response status.

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0025, 0x002C, 0xFFF8,0x0015,0x0026,0x0052

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

4.1.19 rsi_wlan_receive_stats_start

Prototype

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

Description

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

Parameters

Parameter	Description
`channel`	Valid channel number 2.4GHz or 5GHz **Table 1: 2.4GHz Band Channel Mapping** **Table 2: 5GHz Band Channel Mapping**

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0025, 0x002c, 0x000A

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

4.1.20 rsi_wlan_receive_stats_stop

Prototype

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

Description

This API is used to stop the Transmit(TX) & Receive(RX) packets statistics.

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0025, 0x002c

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

4.1.21 rsi_wlan_send_data

Prototype

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

Description

This API is used to send raw data in TCP/IP bypass mode .

Parameters

Parameter	Description
`buffer`	Pointer to the buffer to send
`length`	Length of the buffer to send

Return Values

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-4 : Buffer not available to serve the command

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

4.1.22 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 is used to start the transmit test.

Note: This API is relevant in opermode 8

Parameters

Parameter	Description
`power`	To set TX power in dbm. The valid values are from 2dbm to 18dbm for WiSeConnect^{TM}

Parameter	Description
	module.
`rate`	To set transmit data rate.
`length`	To configure length of the TX packet. 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
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`

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

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

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

0x000A, 0x0021, 0x0025, 0x002C

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

4.1.23 rsi_transmit_test_stop

Prototype

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

Description

This API is used to stop the transmit test.

Note: This API is relevant in opermode 8

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0025, 0x002C

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

4.1.24 rsi_fwup_app

Prototype

```
int32_t  rsi_fwup_app();
```

Description

This API is uses TCP client to get the firmware file from remote server and uses firmware upgradation APIs to upgrade.

Parameters

None

Return Values

On Success : 0

On Firmware upgradation completed successfully : 3

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0025, 0x002C

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

4.1.25 rsi_fwup

Prototype

```
static inline rsi_fwup(
                   uint8_t type,
                   uint8_t *content,
                    uint16_t length);
```

Description

This API is a helper function and called in actual firmware upgradation APIs, it take care of filling upgradation request and sends it to device.

Parameters

Parameter	Description
`type`	firmware upgrade chunk type
`content`	firmware content
`length`	Length of the content

Return Values

On Success : 0

On Firmware upgradation completed successfully : 3

On Failure : <0

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

4.1.26 rsi_fwup_start

Prototype

```
int32_t  rsi_fwup_start(
                   uint8_t *rps_header);
```

Description

This API is used to send the RPS header content of firmware file .

Parameters

Parameter	Description
`rps_header`	pointer to the rps header content

Return Values

On Success : 0

On Failure : <0

if return value is less than 0

-2 : Invalid parameters

-4 : Buffer not available to serve the command

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

4.1.27 rsi_fwup_load

Prototype

```
int32_t  rsi_fwup_load(
                    uint8_t *content,
                    uint16_t length);
```

Description

This API is used to send the firmware file content.

Parameters

Parameter	Description
`content`	Pointer to the firmware file content
`length`	Length of the content

Return Values

On Success : 0

On Firmware upgradation completed successfully : 3

On Failure : <0

if return value is less than 0

-2 : Invalid parameters

-4 : Buffer not available to serve the command

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

4.1.28 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 is used to register WLAN call back functions.

Parameters

Parameter	Description

`callback_id`	Id of 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)`
	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 when ever a wifidirect 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 wifi direct device

Return Values

On Success : 0

On Failure : 1

If `call_back_id` is greater than Maximum call backs to register, returns 1

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

4.2 BSD Socket API

This section contains description about network stack API's, used to configure embedded TCP/IP stack and data transfer over network.

4.2.1 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 is used to configure IP address to module.

Parameters

Parameter	Description
version	IP version RSI_IP_VERSION_4 (4) – to select IPv4 RSI_IP_VERSION_6 (6) – to select IPv6
mode	IP configuration mode RSI_STATIC (0) – to give static address RSI_DHCP (1) – to use DHCP
ip_addr	Pointer to IP address
mask	Pointer to network mask

Parameter	Description
gw	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	Length of `ipconfig_rsp` buffer
vap_id	Vap id is bit to differentiate AP and station in concurrent mode 0 – for station 1 – for Access point

Note : IPv6 is not supported

Return Values

On Success : 0

On Failure :

if return value is less than 0

-2 : Invalid parameters

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x0025,0x002C,0xFFFC,0xFF74,0xFF9C,0xFF9D`

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

4.2.2 socket

Prototype

```
int32_t  socket(int32_t protocolFamily,
                 int32_t type,
                 int32_t protocol);
```

Description

This API is used to create socket.

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

Note : `recv()` ,`recvfrom()`API's are not supported in **WiseConnect plus**. So, recommended to use socket_async() instead of `socket()` API.

Return Values

On success: socket descriptor

On Failure: -1

4.2.3 bind

Prototype

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

Description

This API is used to assign an address to socket.

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

Parameters

Parameter	Description
`sockID`	Socket descriptor
`localAddress`	Address assigned to socket, format is compatible with BSD socket
`addressLength`	Length of the address in bytes

Return Values

On Success : 0

On Failure : -1

4.2.4 connect

Prototype

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

Description

This API is used to connect the socket to specified `remoteAddress`.

Parameters

Parameter	Description
`sockID`	Socket descriptor
`remoteAddress`	Remote peer address, format is compatible with BSD socket
`addressLength`	Length of the address in bytes

Return Values

On Success : 0

On Failure : -1

4.2.5 listen

Prototype

```
int32_t  listen(int32_t sockID,
                int32_t backlog);
```

Description

This API is used to make socket to listen for remote connection request in passive mode.

Parameters

Parameter	Description
`sockID`	Socket descriptor
`backlog`	Maximum length to which the queue of pending connections can hold

Return Values

On Success : 0

On Failure : -1

4.2.6 accept

Prototype

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

Description

This API is used to accept the connection request from remote peer. This API extract the connection request from the queue of pending connections on listening socket and accept it.

Parameters

Parameter	Description
`sockID`	Socket Descriptor
`ClientAddress`	Remote peer address. This parameter is out parameter filled by driver on successful connection acceptance, format is compatible with BSD socket
`addressLength`	Length of the address in bytes

Return Values

On Success : socket descriptor of accepted socket

On Failure : -1

Note : `accept` API is not supported in **WiSeConnect plus**

4.2.7 recvfrom

Prototype

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

Description

This API is used to retrieve data receive data from remote peer on given socket descriptor.

Parameters

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

Return Values

On Success : number of bytes successfully received

On Failure : -1

Note : recvfrom API is not supported in **WiSeConnect plus**

4.2.8 recv

Prototype

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

Description

This API is used to retrieve data receive data from remote peer on specified socket.

Parameters

Parameter	Description
sockID	Socket Descriptor
rcvBuffer	Pointer to buffer to hold data received from remote peer
bufferLength	Length of the buffer
flags	Reserved

Return Values

On Success : number of bytes successfully received

On Failure   : -1

Note : recv API is not supported in **WiSeConnect plus**

4.2.9 sendto

Prototype

```
int32_t  sendto(int32_t sockID,
                int8_t *msg,
                int32_t msgLength,
                int32_t flags,
                struct sockaddr *destAddr,
                int32_t destAddrLen);
```

Description

This API is used to send data to specified remote peer on given socket.

Parameters

Parameter	Description
`sockID`	Socket Descriptor
`msg`	Pointer to data buffer contain data to send to remote peer
`msgLength`	Length of the buffer
`flags`	Reserved
`destAddr`	Address of the remote peer to send data
`destAddrLen`	Length of the address in bytes

Return Values

On Success : number of bytes successfully sent

On Failure : -1

4.2.10 send

Prototype

```
int32_t  send(int32_t sockID,
              const int8_t *msg,
              int32_t msgLength,
              int32_t flags);
```

Description

This API is used to send data to remote peer on given socket.

Parameters

Parameter	Description
`sockID`	Socket Descriptor

Parameter	Description
`msg`	Pointer to buffer contain data to send to remote peer
`msgLength`	Length of the buffer
`flags`	Reserved

Return Values

On Success : number of bytes successfully sent

On Failure : -1

4.2.11 shutdown

Prototype

```
int32_t  shutdown(int32_t sockID,int32_t how);
```

Description

This API is used to close the socket specified in socket descriptor.

Parameters

Parameter	Description
`sockID`	Socket descriptor
`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

On Success : 0

On Failure : -1

4.2.12 socket_async

Prototype

```
int32_t  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 is used to create socket and register a callback which will be used by driver to forward received packets asynchronously to application (on packet reception) without waiting for `recv` API call.

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
`Callback`	Call back function called by driver on reception of receive packet on socket.

Note : IPv6 is not supported

Return Values

On Success : 0

On Failure : -1

4.2.13 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 is used to query IP address for given domain name.

Parameters

Parameter	Description
`ip_version`	IP version 4 : IPv4 6 : IPv6
`url_name`	Pointer to domain name, to resolve IP address
`server_address`	IP address of DNS server, this parameter is optional if module get DNS server address using DHCP.
`dns_query_resp`	Pointer to hold DNS query results, this is an out parameter.
`length`	Length of the results 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`	IP version 4 : IPv4 6 : IPv6
`ip_count`	Number of IP addresses resolved for given domain name
`ip_address`	IP address of given domain name.This field is union of IPv4 and IPv6 address (16 bytes in size).Number of bytes filled depends on IP version.

Return Values

On Success : 0

On Failure : -1

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

4.2.14 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 is used to update the hostname for given host and zone name.

Parameters

Parameter	Description
`ip_version`	IP version 4 : IPv4 6 : IPv6
`zone_name`	Pointer to zone name, to update host name
`host_name`	Pointer to host name, to update host name
`server_address`	IP address of DNS server, this parameter is optional if module get DNS server address using DHCP.
`ttl`	Time to live value of the host name.
dns_update_rsp_handler	Call back function called by driver on reception of dns update response.

Note : IPv6 is not supported in current release

Return Values

On Success : 0

On Failure : -1

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

4.3 Network Application Protocol

Note : **`SMTP client`**, **`FTP client`**, **`MQTT client`**, **`HTTP`** **server,** **`MDNSD`**, **`multicast`**, **`Web socket`** `API` s are not supported in **WiSeConnect plus** in current release

4.3.1 SMTP client API

4.3.1.1 rsi_smtp_client_create

Prototype

```
int32_t rsi_smtp_client_create(uint8_t flags,
                               uint8_t *username,
                               uint8_t *password,
                               uint8_t *from_address,
                               uint8_t *client_domain,
                               uint8_t auth_type,
                               uint8_t *server_ip,
                               uint32_t port);
```

Description

This API is used to create an smtp client.This will initialize the client with given configuration

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

Parameter	Description
username	Username for authentication Should be NULL terminated string
password	Password for authentication Should be NULL terminated string
from_address	sender's address Should be NULL terminated string
client_domain	domain name of the client Should be NULL terminated string
auth_type	client authentication type 1 - SMTP_CLIENT_AUTH_LOGIN 3 - SMTP_CLIENT_AUTH_PLAIN BIT(2) :SMTP_SSL_ENABLED BIT(3) : SMTP_SSL_TLSV_1 BIT(4): SMTP_SSL_TLSV_1_2 BIT(5) : SMTP_SSL_TLSV_1_1
server_ip	SMTP server IP address IPv4 address – 4 Bytes hexa-decimal, IPv6 address – 16 Bytes hexa-decimal
port	SMTP server TCP port Note : SMTP server port is configurable on non standard port also

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015,0xBBA5,0xBB21,0x003E,0xBBB2`

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

4.3.1.2 rsi_smtp_client_mail_send_async

Prototype

```
int32_t rsi_smtp_client_mail_send_async(
                        uint8_t *mail_recipient_address,
                        uint8_t priority,
                        uint8_t *mail_subject,
                        uint8_t *main_body,
                        uint16_t mail_body_length,
      void(*smtp_client_mail_response_handler)
                                  (uint16_t status,
                              const uint8_t cmd));
```

Description

This API is used to send mail to the recipient from the smtp client.

Parameters

Parameter	Description
`mail_recipient_address`	mail recipient address
`priority`	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`	Subject line text Null terminated string.
`mail_body`	mail message
`mail_body_length`	length of mail body

Parameter	Description
	Note : Maximum length of `mail_recipient_address`, `mail_subject`, `mail_body` together is `1024 bytes`
`smtp_client_mail_response_handler`	callback when asynchronous response comes for the sent mail parameters: `status , cmd` `status:`status code `cmd:` sub command type

Note : If status in callback is nonzero, sub command type is in 6th byte of descriptor

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

```
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 WLAN Error codes for description of above error codes.

4.3.1.3 rsi_smtp_client_delete_async

Prototype

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

Description

This API is used to delete the smtp client.

Parameters

Parameter	Description
`smtp_client_delete_response_handler`	callback when asynchronous response comes for the delete request parameters: `status , cmd` `status:`status code `cmd:` sub command type

Note : If status in callback is nonzero, sub command type is in 6th byte of descriptor

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

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

4.3.2 SNTP Client API

4.3.2.1 rsi_ sntp_client_create_async

Prototype

```
int32_t rsi_sntp_client_create_async(uint8_t flags,
uint8_t *server_ip, uint8_t sntp_method, uint16_t
sntp_timeout,void(*rsi_sntp_client_create_response_handl
er)(uint16_t status,const uint8_t cmd_typr, const
uint8_t *buffer));
```

Description

This API is used to create the sntp client

Parameter	Description

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	server IP address
sntp_method	SNTP method to use 1-For Broadcast Method 2-For Unicast Method
sntp_timeout	SNTP timeout value
rsi_sntp_client_create_response_handler	allback when asynchronous response comes for the request parameters: status, cmd_type, buffer status:status code cmd_type:command type buffer: buffer pointer

Return Values

On Success : 0

On Failure :

if return value is less 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

These responses may be observed as asynchronous message in the response handler function

0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0

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

4.3.2.2 rsi_sntp_client_gettime

Prototype

```
 int32_t rsi_sntp_client_gettime(uint16_t
length,uint8_t*sntp_time_rsp);
```

Description

This API is used to get the current time Parameters

Parameter	Description
Length	Length of the buffer
sntp_time_rsp	Get the current time response

Return Values

On Success : 0

On Failure :

if return value is less 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

These responses may be observed as asynchronous message in the response handler function

0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0

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

4.3.2.3 rsi_sntp_ client_gettime_date

Prototype

```
int32_t rsi_sntp_client_gettime_date(uint16_t length, uint8_t
*sntp_time_date_rsp)
```

Description

This API is used to get the current time in time date format Parameters

Parameter	Description
Length	Length of the buffer
sntp_time_date_rsp	Get the cuurent time and date response

Return Values

On Success : 0

On Failure :

if return value is less 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

These responses may be observed as asynchronous message in the response handler function

0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0

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

4.3.2.4 rsi_ sntp_client_server_info

Prototype

```
int33_t rsi_sntp_client_info(uint16_t length, uint8_t
*sntp_server_response);
```

Description

This API is used to get the SNTP server details Parameters

Parameter	Description
Length	Length of the buffer
sntp_server_response	Get the server details

Return Values

On Success : 0

On Failure :

if return value is less 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

These responses may be observed as asynchronous message in the response handler function

0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0

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

4.3.2.5 rsi_ sntp_client_delete_async

Prototype

```
int32_t rsi_sntp_client_delete_async(void);
```

Description

This API is used to delete the SNTP client

Return Values

On Success : 0

On Failure :

if return value is less 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

These responses may be observed as asynchronous message in the response handler function

0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0

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

4.3.3 HTTP Client API

4.3.3.1 rsi_http_client_get_async

Prototype

```
int32_t rsi_http_client_get_async(uint8_t flags,
                                  uint8_t *ip_address,
                                  uint16_t port,
                                  uint8_t *resource,
                                  uint8_t *host_name,
                                  uint8_t *extended_header,
                                  uint8_t *user_name,
                                  uint8_t *password,
              void(*http_client_get_response_handler)
                                       (uint16_t status,
                                   const uint8_t *buffer,
                                   const uint16_t length)
                                   const uint32_t more_data);
```

Description

This API is used to send http get request to remote HTTP server.

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`	server IP address
`port_no`	port number of HTTP server Note : HTTP server port is configurable on non standard port also
`resource`	URL string for requested resource
`hostname`	host name
`extended_header`	user defined extended header
`username`	username for server Authentication
`password`	password for server Authentication
`http_client_get_response_handler`	callback when asynchronous response comes for the request parameters: `status , buffer, length, more_data`

Parameter	Description
	`status`:status code `buffer`: buffer pointer `length`: length of data `more_data`: if 1 – No more data 0 – more data present

Return Values

On Success : 0

On Failure :

if return value is less 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

These responses may be observed as asynchronous message in the response handler function

`0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0`

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

4.3.3.2 rsi_http_client_post_async

Prototype

```
int32_t rsi_http_client_post_async(uint8_t flags,
                             uint8_t *ip_address,
                             uint16_t port,
                             uint8_t *resource,
                             uint8_t *host_name,
                             uint8_t *extended_header,
                             uint8_t *user_name,
                             uint8_t *password,
                             uint8_t *post_data,
                             uint16_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 is used to send http post request to remote HTTP server.

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`	server IP address
`port_no`	port number of HTTP server
`resource`	URL string for requested resource
`hostname`	host name
`extended_header`	user defined extended header
`username`	username for server Authentication
`password`	password for server Authentication
`post_data`	HTTP data to be posted to server
`post_data_length`	post data length
`http_client_post_response_handler`	callback when asynchronous response comes for the request parameters: `status , buffer, length, more_data` `status:`status code `buffer:` buffer pointer `length:` length of data `more_data:` if 1 – No more data 0 – more data present

Parameter	Description
	2 – HTTP post success response

Return Values

On Success : 0

On Failure :

if return value is less 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

These responses may be observed as asynchronous message in the response handler function

`0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0`

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

4.3.3.3 rsi_http_client_post_data

Prototype

```
int32_t rsi_http_client_post_data(uint8_t *file_content,
                                  uint16_t current_chunk_length,
         void(*http_client_post_data_response_handler)
                                           (uint16_t status,
                                       const uint8_t *buffer,
                                       const uint16_t length)
                                       const uint32_t more_data);
```

Description

This API is used to send http post data packet to remote HTTP server.

Parameters

Parameter	Description
`file_content`	User given http file content

Parameter	Description
`current_chunk_length`	Length of the current http data
`http_client_post_data_response_handler`	callback when asynchronous response comes for the request parameters: `status , buffer, length, more_data` `status:`status code `buffer:` buffer pointer `length:` length of data `more_data:` if 1 – No more data 0 – more data 4 – HTTP post data response 8 – HTTP post data receive response

Return Values

On Success : 0

On Failure :

if return value is less 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

These responses may be observed as asynchronous message in the response handler function

`0x0021,0x002C,0x0015,0x0025,0xFF74,0xBBF0, 0xBB38, 0xBB3E, 0xBBEF`

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

4.3.3.4 rsi_http_client_abort

Prototype

```
int32_t rsi_http_client_abort(void)
```

Description

This API is used to abort any ongoing HTTP request from the client.

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

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

4.3.4 POP3 client API

4.3.4.1 rsi_pop3_session_create_async

Prototype

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

Description

This API is used to create 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`	client authentication type `(Resserved)`
`client_domain`	domain name of the client Should be NULL terminated string
`username`	Username for authentication Should be NULL terminated string
`password`	Password for authentication Should be NULL terminated string
`rsi_pop3_response_handler`	Callback when asynchronous response comes for the session create. Parameters : status,type,buffer status : status code type : sub command type buffer : buffer pointer

Note : If status in callback is nonzero, sub command type is in 6th byte of descriptor

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x0015,0xBB87,0xff74`

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

4.3.4.2 rsi_pop3_get_mail_stats

Prototype

```
int32_t rsi_pop3_get_mail_stats (void)
```

Description

This API is used to get the mail stats.

Parameters

No Parameters

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021,0xFF74,0xBB87

4.3.4.3 rsi_pop3_get_mail_list

Prototype

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

Description

This API is used to get the size of the mail for the passed mail index.

Parameters

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

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0xBB87,0xFF74,0xBBFF`

4.3.4.4 rsi_ pop3_retrive_mail

Prototype

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

Description

This API is used to retrive the mail content for the passed mail index

Parameters

Parameter	Description
`mail_index`	mail index to get the mail content for the passed index

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0xBB87,0xFF74,0xBBFF,0xBBC5`

4.3.4.5 rsi_pop3_mark_mail

Prototype

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

Description

This API is used to mark a mail as deleted for the passed mail index

Parameters

Parameter	Description
mail_index	mail index to mark the mail as deleted

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021,0xFF74,0xBB87,0xBBFF

4.3.4.6 rsi_pop3_unmark_mail

Prototype

```
int32_t rsi_pop3_unmark_mail(void)
```

Description

This API is used to unmark all the marked (deleted) mails in the current session

Parameters

No parameters

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0xFF74,0xBB87`

4.3.4.7 rsi_ pop3_get_server_status

Prototype

`int32_t rsi_pop3_get_server_status(void)`

Description

This API is used to get the pop3 server status.

Parameters

No parameters

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0xBB87,0xFF74`

4.3.4.8 rsi_pop3_session_delete

Prototype

`int32_t rsi_pop3_session_delete(void)`

Description

This API is used to delete pop3 client session.

Parameters

No parameters

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

```
0x0021,0xFF74,0xBB87
```

4.3.5 FTP Client API

4.3.5.1 rsi_ftp_connect

Prototype

```
int32_t rsi_ftp_connect(uint16_t flags, int8_t
*server_ip, int8_t *username, int8_t *password, uint32_t
server_port)
```

Description

This API is used to create FTP objects and connect to the FTP server on the given server port.This should be the first command for accessing FTP server.

Parameters

Parameter	Description
`flags`	Network flags.Each bit in the flag is is own significance `BIT(0) - RSI_IPV6` Set this bit to enable IPv6 , by default it is configured to IPv4 `BIT(1) to BIT(15)` are reserved for future use
`server_ip`	FTP server IP address to connect
`username`	username for server Authentication
`password`	password for server Authentication
`server_port`	port number of FTP server Note : FTP server port is configurable on non standard port also

Return Values

On Success: 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

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

4.3.5.2 rsi_ftp_disconnect

Prototype

```
int32_t rsi_ftp_disconnect(void)
```

Description

This function is used to disconnect from the FTP server and destroy the FTP objects.Once FTP objects are destroyed,FTP server cannot be accessed.For the further accessing,FTP objects should be created again

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

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

4.3.5.3 rsi_ftp_file_write

Prototype

```
int32_t rsi_ftp_file_write(int8_t *file_name)
```

Description

This function is used to open a file in the specified path on the FTP server.

Parameters

Parameter	Description

Parameter	Description
`file_name`	File name or filename including path can be given. e.g "`example.txt`" or "`/test/ftp/example.txt`"

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

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

4.3.5.4 rsi_ftp_file_write_content

Prototype

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

Description

This function is used to write the content into the file opened using `rsi_ftp_file_write()` API

Parameters

Parameter	Description
`flags`	Network flags.Each bit in the flag is is 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`	Data stream to be written into the

Parameter	Description
	file
content_length	file content length
end_of_file	This flag indicates the end of file 1 – This chunk is end of content to write into the file 0 – more data 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

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

Please refer WLAN Error codes for description of 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 breaks the file content and send it in multiple packets.

4.3.5.5 rsi_ftp_file_read_async

Prototype

```
int32_t rsi_ftp_file_read_aysnc(int8_t *file_name, void
(*call_back_handler_ptr)(uint16_t status, int8_t
*file_content, uint16_t content_length, uint8_t
end_of_file))
```

Description

This function is used to read the content from the file content from the specified file on the FTP server

Parameters

Parameter	Description
`file_name`	File name or filename including path can be given. e.g "`example.txt`" or "`/test/ftp/example.txt`"
`call_back_handler_ptr`	callback when asynchronous response comes for the file read request parameters:`status` , `file_content`, `content_length`, `end_of_file` `status`: status code.Other parameters are valid only if status is 0 `file_content`: file content `content_length`:length of file content `end_of_file`: indicates end of file if 1 – No more data 0 – more data present

Return Values

On Success : 0

On Failure :

if return value is less 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 WLAN Error codes for description of above error codes.

4.3.5.6 rsi_ftp_file_delete

Prototype

```
int32_t rsi_ftp_file_delete(int8_t *file_name)
```

Description

This API is used to delete the file which is present in the specified path on the FTP server.

Parameters

Parameter	Description
`file_name`	File name or filename including path can be given to delete e.g "`example.txt`" or "`/test/ftp/example.txt`"

Return Values

On Success: 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

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

.

4.3.5.7 rsi_ftp_file_rename

Prototype

```
int32_t rsi_ftp_file_rename(int8_t *old_file_name,
int8_t *new_file_name)
```

Description

This API is used to rename the file with the new name on the FTP server

Parameters

Parameter	Description
`old_file_name`	filename/file name which has to be renamed
`new_file_name`	new file name

Return Values

On Success: 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

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

4.3.5.8 rsi_ftp_directory_create

Prototype

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

Description

This API is used to create a directory on the FTP server

Parameters

Parameter	Description
`directory_name`	directory name(with path if required) to create e.g “`example`” or “`/test/ftp/example`”

Return Values

On Success: 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

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

4.3.5.9 rsi_ftp_directory_delete

Prototype

```
int32_t rsi_ftp_directory_delete(int8_t *directory_name)
```

Description

This API is used to delete directory on the FTP server

Parameters

Parameter	Description
`directory_name`	directory name(with path if required) to delete e.g "example" or "/test/ftp/example"

Return Values

On Success: 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

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

4.3.5.10 rsi_ftp_directory_set

Prototype

```
int32_t rsi_ftp_directory_set(int8_t *directory_name)
```

Description

This function is used to change the current working directory to the specified directory path on the FTP server

Parameters

Parameter	Description
`directory_name`	directory name(with path if required) to create

Return Values

On Success: 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021,0x002C,0x0015`

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

4.3.5.11 rsi_ftp_directory_list_async

Prototype

```
int32_t rsi_ftp_directory_list_async(int8_t
*directory_path,void (*call_back_handler_ptr)(uint16_t
status, int8_t *directory_list, uint16_t length ,
uint8_t end_of_list))
```

Description

This function is used to get the list of directories present in the specified directory on the FTP server

Parameters

Parameter	Description
`file_name`	File name or filename including path can be given.

Parameter	Description
	e.g "`example.txt`" or "`/test/ftp/example.txt`"
`call_back_handler_ptr`	callback when asynchronous response comes for the directory list request parameters:`status , directory_list, length, end_of_list` `status:` status code.Other parameters are valid only if status is 0 `directory_list:` Stream of data with directory list as content `length:` length of content `end_of_list`: indicates end of list if 1 – No more data 0 – more data present

Return Values

On Success : 0

On Failure :

if return value is less 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 WLAN Error codes for description of above error codes.

4.3.5.12 rsi_ ftp_mode_set

Prototype

```
int32_t rsi_ftp_mode_set(uint8_t mode)
```

Description

This function is Used to set the FTP client mode , either in Passive mode or Active Mode

Parameters

Parameter	Description
`mode`	Used to select the mode of FTP client if FTP enable is 0-Active Mode 1-Passive Mode.

Return Values

On Success : 0

On Failure :

if return value is less 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 WLAN Error codes for description of above error codes.

4.3.6 MQTT Client API

4.3.6.1 rsi_mqtt_client_init

Prototype

```
rsi_mqtt_client_info_t * rsi_mqtt_client_init( int8_t
*buffer, uint32_t length, int8_t *server_ip, uint32_t
server_port, uint32_t client_port, uint16_t flags,
uint16_t keep_alive_interval)
```

Description

This API initilalises the MQTT client structure memory with the linear buffer pointed.This memory is used by MQTT client for the further MQTT operations

Parameters

Parameter	Description
`buffer`	Linear buffer required to initialize

Parameter	Description
	MQTT client structure
`length`	length of the linear buffer pointed
`server_ip`	MQTT broker IP address to connect
`server_port`	port number of MQTT broker
`client_port`	port number of MQTT client(local port)
`flags`	Network flags.Each bit in the flag is is 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`	MQTT client keep alive interval If there are no transactions between MQTT client and broker with in this time period, MQTT Broker disconnects the MQTT client

Return Values

On Success: Returns MQTT client info structure pointer

On Failure : NULL

4.3.6.2 rsi_mqtt_connect

Prototype

```
int32_t rsi_mqtt_connect (rsi_mqtt_client_info_t
*rsi_mqtt_client, uint16_t flags, int8_t
*client_id,int8_t *username,int8_t *password)
```

Description

This API establishes TCP connection with the given MQTT client port and establishes MQTT protocol level connection

Parameters

Parameter	Description
`rsi_mqtt_client`	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`	clientID string of the MQTT Client and should be unique for each device.
`username`	username for server Authentication
`password`	password for server Authentication

Return Values

On Success: 0

On Failure   :

-2 : Invalid parameter,expects call back handler

if return value is greater than 0

`0x0021, 0x002C,0x0015`

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

4.3.6.3 rsi_mqtt_disconnect

Prototype

```
int32_t rsi_mqtt_disconnect(rsi_mqtt_client_info_t
*rsi_mqtt_client)
```

Description

This API is used to disconnect the client from MQTT broker

Parameters

Parameter	Description
`rsi_mqtt_client`	MQTT client info structure pointer

Return Values

On Success: 0

On Failure   :

    -2 : Invalid parameter

if return value is greater than 0

`0x0021, 0x002C,0x0015`

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

4.3.6.4 rsi_mqtt_publish

Prototype

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

Description

This API is used publish the message on the topic specified

Parameters

Parameter	Description
`rsi_mqtt_client`	MQTT client info structure pointer
`topic`	Topic string on which MQTT client wants to publish data
`publish_msg`	Publish message structure

Return Values

On Success: 0

On Failure   :

    -2 : Invalid parameter

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

4.3.6.5 rsi_mqtt_subscribe

Prototype

```
int32_t rsi_mqtt_subscribe(rsi_mqtt_client_info_t
*rsi_mqtt_client,uint8_t qos, int8_t *topic,void
(*call_back_handler_ptr)(MessageData* md))
```

Description

This API is used subscribe to the topic specified.Thus MQTT client will receive any data which is published on this topic further and callback registered will be called.

Parameters

Parameter	Description
`rsi_mqtt_client`	MQTT client info structure pointer
`qos`	Quality of Service of message at MQTT protocol level valid values are 0,1,2
`topic`	Topic string on which MQTT client wants to subscribe
`call_back_handler _ptr`	callback when asynchronous data comes on the subscribed data `MessageData* md` Message data pointer received

Return Values

On Success: 0

On Failure :

-2 : Invalid parameter

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

4.3.6.6 rsi_mqtt_unsubscribe

Prototype

```
int32_t rsi_mqtt_unsubscribe( rsi_mqtt_client_info_t
*rsi_mqtt_client, int8_t *topic
```

Description

This API is used unsubscribe to the topic specified.Thus MQTT client will not receive any data published on this topic further

Parameters

Parameter	Description
`rsi_mqtt_client`	MQTT client info structure pointer
`topic`	Topic string on which MQTT client wants to unsubscribe to

Return Values

On Success: 0

On Failure:

-2 : Invalid parameter

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

4.3.6.7 rsi_mqtt_recv

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.

Parameters

Parameter	Description
`rsi_mqtt_client`	MQTT client info structure pointer
`time_out`	Time out in milli seconds for which MQTT client has to wait for the messages to receive on the subscribed topic

Return Values

On Success: 0

On Failure:

-2 : Invalid parameter

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

4.3.7 HTTP Server API

4.3.7.1 rsi_webpage_load

Prototype

int32_t rsi_webpage_load(uint8_t flags, uint8_t *file_name, uint8_t *webpage, uint32_t length);

Description

This API is used to load webpage to the HTTP Server's file system which is present in the WiSeConnect^{TM} module.

Parameters

Parameter	Description
`flags`	BIT(2) is used to set webpage is associated with json object
`file_name`	File name of the html webpage
`webpage`	Pointer to the html webpage which contains the html webpage content
`length`	Webpage length

Return Values

On Success : 0

On Failure :

if return value is less 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 WLAN Error codes for description of above error codes.

4.3.7.2 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 is used to create the json object to the webpage which is already present in the WiSeConnect^{TM} module's HTTP server file system.

Parameters

Parameter	Description
`flags`	Rserved
`file_name`	File name of the json object data
`json_object`	Pointer to the json object data
`length`	Length of the json object data

Return Values

On Success : 0

On Failure :

if return value is less 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 WLAN Error codes for description of above error codes.

4.3.7.3 rsi_webpage_erase

Prototype

int32_t rsi_webpage_erase(uint8_t *file_name);

Description

This API is used to erase the webpage from HTTP server's file system which is present in the WiSeConnect^{TM} module.

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

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021, 0x0025,0x002C, 0x00C4`

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

4.3.7.4 rsi_json_object_delete

Prototype

int32_t rsi_json_object_delete(uint8_t *file_name);

Description

This API is used to delete the json object of the HTTP server's file system which is already present in the WiSeConnect^{TM} module.

Parameters

Parameter	Description
`file_name`	To delete the particular json object which is already created in the HTTP server's file system

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

if return value is greater than 0

`0x0021, 0x0025, 0x002C,0x00B4`

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

4.3.8 DHCP User class API

4.3.8.1 rsi_dhcp_user_class

Prototype

```
int32_t  rsi_dhcp_user_class(uint8_t count, uint8_t
*(usr_cls_arr)[count], void(*dhcp_usr_cls_rsp_handler)(uint16_t
status));
```

Description

This API is used to enable DHCP user class

Parameter	Description
`count`	`DHCP User Class count`
`usr_cls_arr`	Containing User class data
`Status`	Status of the DHCP user class 0 = success <0 = failure

Return Values

On Success: 0

On Failure:

-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 WLAN Error codes for description of above error codes.

4.3.9 Multicast API

4.3.9.1 rsi_multicast_join

Prototype

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

Description

This API is used to join 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

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

On Success: 0

On Failure:

-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 WLAN Error codes for description of above error codes.

4.3.9.2 rsi_multicast_leave

Prototype

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

Description

This API is used to leave the multicast group.

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

Parameters

Parameter	Description
`flags`	To select the IP version.

Parameter	Description
	`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

On Success: 0

On Failure:

-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 WLAN Error codes for description of above error codes.

4.3.10 MDNSD API

4.3.10.1 rsi_mdnsd_init

Prototype

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

Description

This API is used to initialize the MDNSD service in WiSe connect/WiSe connect Plus Device. It creates MDNS daemon.

Note :

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

Parameters

Parameter	Description
`ip_version`	To select the IP version. `4 –` To select IPv4

Parameter	Description
	6 – To select IPv6
ttl	time to live , Time in seconds for which service should be active
host_name	Host name which is used as host name in Type A record.

Return Values

On Success: 0

On Failure:

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0015, 0x0074

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

4.3.10.2 rsi_mdnsd_register_service

Prototype

```
int32_t rsi_mdnsd_register_service(uint8_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

Parameters

Parameter	Description
port	Port number on which service which should be added.
ttl	time to live , Time in seconds for

Parameter	Description
	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`	Name to be added in Type-PTR record
`service_name`	Name to be added in Type-SRV record(Service name)
`service_text`	Text field to be added in Type-TXT record

Return Values

On Success: 0

On Failure:

-4 : Buffer not available to serve the command

-6 : Data size exceeded

if return value is greater than 0

0x0021, 0x0015, 0x0074

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

4.3.10.3 rsi_mdnsd_deinit

Prototype

```
int32_t rsi_mdnsd_deinit(void);
```

Description

This API is used to delete the mdnsd service.

Parameters

None

Return Values

On Success: 0

On Failure:

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0015, 0x0074,0xFF2B

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

4.3.11 Web socket API

4.3.11.1 rsi_web_socket_create

Prototype

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

Description

This API is used to create a web socket client .

Parameters

Parameter	Description

`flags`	To select IP version and security
	`BIT(0) - RSI_IPV6`
	Set this bit to enable IPv6 , by default it is configured to IPv4
	`BIT(1) - RSI_SSL_ENABLE`
	Set this bit to enable SSL feature

`server_ip_addr`	Web server ip address
`server_port`	Web server socket port.
`device_port`	Local port
`webs_resource_name`	web resource name `Note`:string of 50 characters maximum
`webs_host_name`	web host name `Note`:string of 50 characters maximum
`web_socket_data_receive_notify_callback)`	callback when data packet is

Parameter	Description
	received on the created socket. parameters: `sock_no , buffer, length, more_data` `sock_no`:Application socket ID `buffer:` buffer pointer `length:` length of data

Return Values

On Success: 0

On Failure:

-2 : Invalid parameter

-4 : Buffer not available to serve the command

if return value is greater than 0

0x0021, 0x0015, 0x0074

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

4.3.11.2 rsi_web_socket_send_async

Prototype

```
int32_t rsi_web_socket_send_async(uint32_t sockID,
                                  uint8_t opcode,
                                  int8_t *msg,
                                  int32_t msg_length);
```

Description

This API is used to send data from the web socket client .

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

Parameter	Description
	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

On Success: 0

On Failure: -1

4.3.11.3 rsi_web_socket_close

Prototype

```
int32_t rsi_web_socket_close(int32_t sockID);
```

Description

This API is used to close the web socket client .

Parameters

None

Return Values

On Success: 0

On Failure: -1

4.3.12 OTAF client API

4.3.12.1 rsi_ota_firmware_upgradation

Prototype

```
int32_t rsi_ota_firmware_upgradation(uint8_t flags,
                                    uint8_t *server_ip,
                                    uint32_t server_port,
                                    uint16_t chunk_number,
                                    uint16_t timeout,
                                    uint16_t tcp_retry_count,
                 void(*ota_fw_up_response_handler)(uint16_t
                        status, uint16_t chunk_number));
```

Description

This API is used to create an otaf client.This will initialize the client with given configuration

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`	OTAF server IP address
`server_port`	OTAF server port number
`chunk_number`	firmware content request chunk number
`timeout`	TCP receive packet timeout
`tcp_retry_count`	TCP retransmisstions count
`ota_fw_up_response_handler`	Callback when asynchronous response comes for the firmware upgrade request

Parameter	Description
	parameters: `status , chunk_number` `status`: status code `chunk_number`: chunk number of the firmware content

Return Values

On Success : 0

On Failure :

if return value is less than 0

-3 : Command given in wrong state

-4 : Buffer not available to serve the command

if return value is greater than 0

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

4.3.13 PUF API

4.3.13.1 rsi_puf_enroll_req

Prototype

```
int32_t rsi_puf_enroll_req(void)
```

Description

This API is used to Enroll PUF. API upon success will save activation code on flash. The stored activation code shall be used for every further start operation on PUF.

Parameters

None

Pre Condition

This should be given after OPERMODE.

Return Values

On Success: 0

On Failure :

If return value is less than 0

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

`0xCC2F, 0xCC33, 0xCC34, 0xCC35`

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

4.3.13.2 rsi_puf_enroll_disable_req

Prototype

```
int32_t rsi_puf_enroll_disable_req(void)
```

Description

This API will block further Enrollment of PUF.

Parameters

None

Pre Condition

This should be given after OPERMODE.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

0xCC32, 0xCC33, 0xCC34

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

4.3.13.3 rsi_puf_start_req

Prototype

```
int32_t rsi_puf_start_req(void)
```

Description

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

Parameters

None

Pre Condition

This should be given after OPERMODE.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

0xCC31, 0xCC32, 0xCC33, 0xCC35

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

4.3.13.4 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 set 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)
key_size	Key Size in bytes, 0 : 128bit key 1 : 256bit key
key_ptr	Pointer to key provided by application
set_key_resp	Keycode to the key provided, this is an output parameter.
Length	Length of the result buffer in bytes to hold keycode.

Pre Condition

This should be given after PUF START.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-2: Invalid parameters

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

`0xCC2F, 0xCC32, 0xCC33, 0xCC34`

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

4.3.13.5 rsi_puf_set_key_disable_req

Prototype

```
int32_t rsi_puf_set_key_disable_req(void)
```

Description

This API is used to block set key for further operations on PUF.

Parameters

None

Pre Condition

This should be given after OPERMODE.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

`0xCC32, 0xCC33, 0xCC34`

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

4.3.13.6 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`	Pointer to KeyCode
`get_key_resp`	Pointer to key, this is an output parameter
`length`	Length of the result buffer in bytes to hold key.

Pre Condition

This should be given after PUF START.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-2: Invalid parameters

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

`0xCC2F, 0xCC32, 0xCC33, 0xCC34`

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

4.3.13.7 rsi_puf_get_key_disable_req

Prototype

```
int32_t rsi_puf_get_key_disable_req(void)
```

Description

This API is used to block further get key operations on PUF.

Parameters

None

Pre Condition

This should be given after OPERMODE.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

0xCC32, 0xCC33, 0xCC34

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

4.3.13.8 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`	Pointer to keycode provided by application

Pre Condition

This should be given after PUF START.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-2: Invalid parameters

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

`0xCC2F, 0xCC32, 0xCC33, 0xCC34`

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

4.3.13.9 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 is used to request 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	Key Index of Key to be generated (0-15)
key_size	Key Size in bytes, 0 : 128bit key 1 : 256bit key
set_key_resp	Keycode to the key provided, this is an output parameter.
Length	Length of the result buffer in bytes to hold keycode.

Pre Condition

This should be given after PUF START.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-2: Invalid parameters

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

`0xCC2F, 0xCC32, 0xCC33, 0xCC34`

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

4.3.13.10 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	AES encryption mode 0 : ECB 1 : CBC
Key_source	Encyption key source, 0 : AES engine, provided by application as key_ptr 1 : PUF
key_size	Key Size in bytes, 0 : 128bit key 1 : 256bit key
key_ptr	Pointer to key provided by application
data_size	Size of data in bytes
data_ptr	Pointer to Data to be encrypted
iv_size	Intialization vector size(if CBC mode) 0 : 128bit key
iv_ptr	Pointer to IV(if CBC mode)
Aes_encry_resp	Pointer to encrypted data, this is an output parameter.
length	Length of the result buffer in bytes to hold encrypted data.

Pre Condition

This should be given after PUF START.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-2: Invalid parameters

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

`0xCC32`

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

4.3.13.11 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	AES encryption mode 0 : ECB 1 : CBC

Parameter	Description
Key_source	Decryption key source, 0 : AES engine, provided by application as key_ptr 1 : PUF
key_size	Key Size in bytes, 0 : 128bit key 1 : 256bit key
key_ptr	Pointer to key provided by application
data_size	Size of data in bytes
data_ptr	Pointer to Data to be decrypted
iv_size	Intialization vector size(if CBC mode) 0 : 128bit key
iv_ptr	Pointer to IV(if CBC mode)
aes_decry_resp	Pointer to decrypted data, this is an output parameter.
length	Length of the result buffer in bytes to hold decrypted data.

Pre Condition

This should be given after PUF START.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-2: Invalid parameters

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

0xCC32

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

4.3.13.12 rsi_puf_aes_mac_req

Prototype

```
int32_t rsi_puf_aes_mac_req (
                        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_size`	Key Size in bytes, 0 : 128bit key 1 : 256bit key
`key_ptr`	Pointer to key provided by application
`data_size`	Size of data in bytes
`data_ptr`	Pointer to Data
`iv_size`	Intialization vector size (if CBC mode) 0 : 128bit key
`iv_ptr`	Pointer to IV(if CBC mode)
`aes_mac_resp`	Pointer to MAC data, this is an output parameter.
`length`	Length of the result buffer in bytes to hold MAC data.

Pre Condition

This should be given after PUF START.

Return Values

On Success: 0

On Failure:

If return value is less than 0

-2: Invalid parameters

-3: Command given in wrong state

-4: Buffer not available to serve the command

If return value is greater than 0

`0xcc32`

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

4.4 Configuration parameters

This section contain description of configuration macro's may need to change based on application requirement. These macro's with default values are placed in "**`rsi_wlan_config.h`**" .

4.4.1 Configure opermode parameters

Define	Meaning
`RSI_FEATURE_BIT_MAP`	To select WiSeConnect^{TM} 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^{TM} module
`RSI_TCP_IP_FEATURE_BIT_MAP`	`TCP_IP_FEAT_BYPASS` – Select to use TCP/IP bypass `TCP_IP_FEAT_HTTP_SERVER` – Select to use HTTP SERVER `TCP_IP_FEAT_DHCPV4_CLIENT` – Select to use DHCPv4 client `TCP_IP_FEAT_DHCPV6_CLIENT` – Select to use DHCPv6 client `TCP_IP_FEAT_DHCPV4_SERVER` – Select to use DHCPv4 Server `TCP_IP_FEAT_DHCPV6_SERVER` – Select to use DHCPv6 server `TCP_IP_FEAT_JSON_OBJECTS` – Select to use JSON objects `TCP_IP_FEAT_HTTP_CLIENT` -Select to use HTTP CLIENT

Define	Meaning
	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 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

Define	Meaning
	CUSTOM_FEAT_BT_IAP – To enable IAP support in Bluetooth classic

4.4.2 Configure scan parameters

Define	Meaning
RSI_SCAN_CHANNEL_BIT_MAP_2_4	To select channels in 2.4GHz band to do selective channel scan. This macro is valid only if channel 0 is selected in rsi_wlan_scan API.
RSI_SCAN_CHANNEL_BIT_MAP_5	To select channels in 5GHz band to do selective channel scan. This macro is valid only if channel 0 is selected in rsi_wlan_scan API.
RSI_SCAN_FEAT_BITMAP	RSI_ENABLE_QUICK_SCAN - If enabled, module scans for the AP given in scan API and posts the scan results immediately to the host after finding the Access point. This bit is valid only if specific channel and ssid to scan is given.

4.4.3 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

4.4.4 Configure Set Region parameters

Define	Meaning
RSI_SET_REGION_SUPPORT	Enable to send set region command during wifi 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

Define	Meaning
	2 - EUROPE 3 - JAPAN

4.4.5 Configure Set Region AP parameters

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

4.4.6 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 first time join failure

4.4.7 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 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

Define	Meaning
	is more than `RSI_RSSI_TOLERANCE_THRESHOLD` then BG scan will be triggered irrespective of periodicity.
`RSI_BG_SCAN_PERIODICITY`	This is time period in seconds to trigger BG scan
`RSI_ACTIVE_SCAN_DURATION`	This is active scan duration per channel in milli seconds
`RSI_PASSIVE_SCAN_DURATION`	This is passive scan duration per DFS channel in 5GHz in milli seconds
`RSI_MULTIPROBE`	If 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)

4.4.8 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

4.4.9 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

Define	Meaning
	1 -Green field support enabled ht_caps_bit_map[0:3]:Set to '0'.

4.4.10 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 supported inner methods are MSCHAP/MSCHAPV2. In case of TLS/FAST should be fixed to MSCHAPV2.

4.4.11 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.

Define	Meaning
	RSI_DATA_RATE_AUTO : Auto rate

4.4.12 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)

4.4.13 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 0- TX BASED 1 - PERIODIC
RSI_WMM_PS_WAKE_INTERVAL	To set wmm wake up interval
RSI_WMM_PS_UAPSD_BITMAP	To set wmm UAPSD bitmap

4.4.14 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 to 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

4.5 WLAN API call sequence examples

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

4.5.1 Station mode

The following flowchart briefs the sequence of API calls to configure module in station mode and connect to an access point. Edit `rsi_wlan_config.h` for the required configuration.

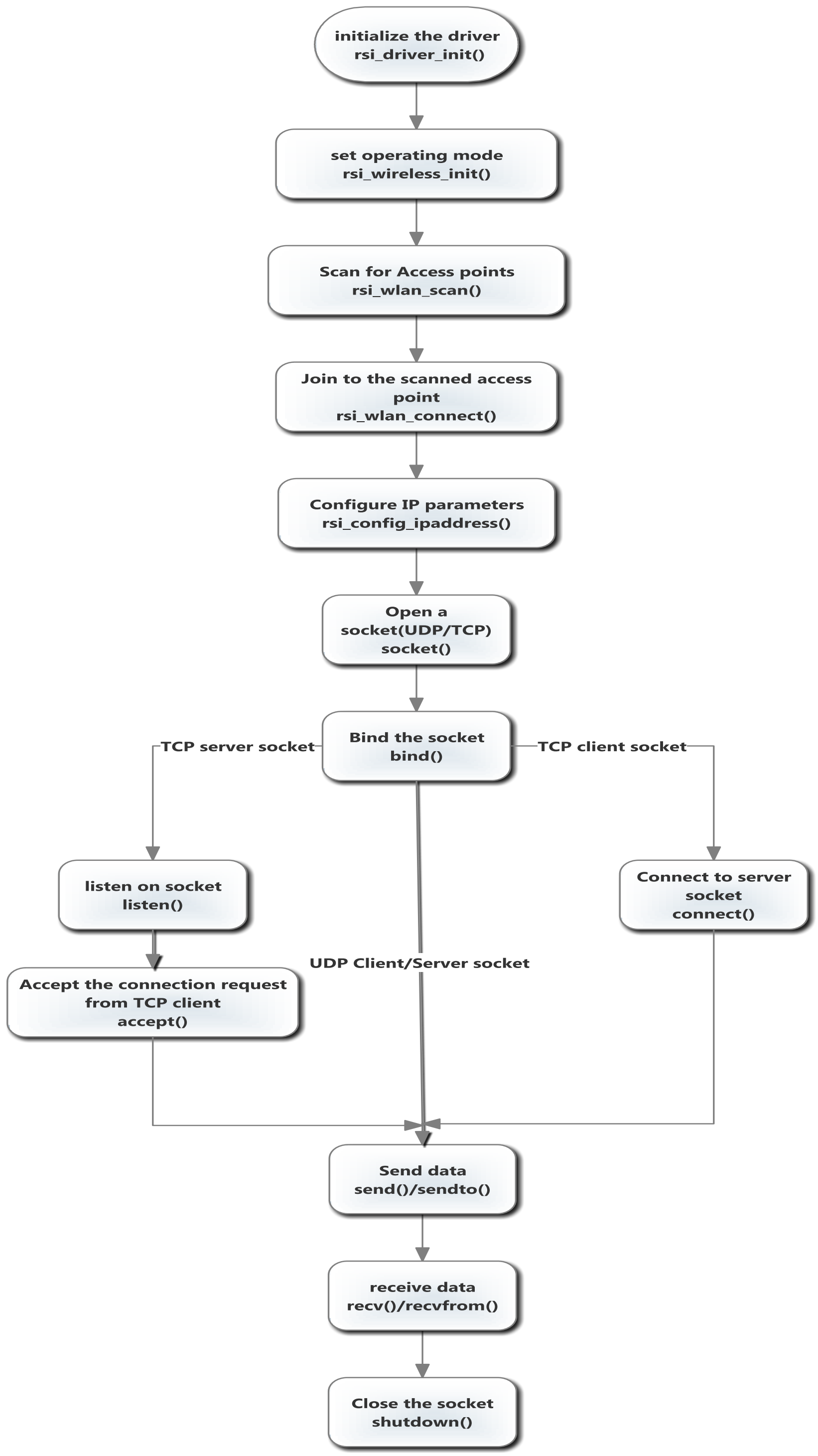


Figure 2 : API flow to configure module in station mode

The following example illustrate the data transfer using `socket_async()` API

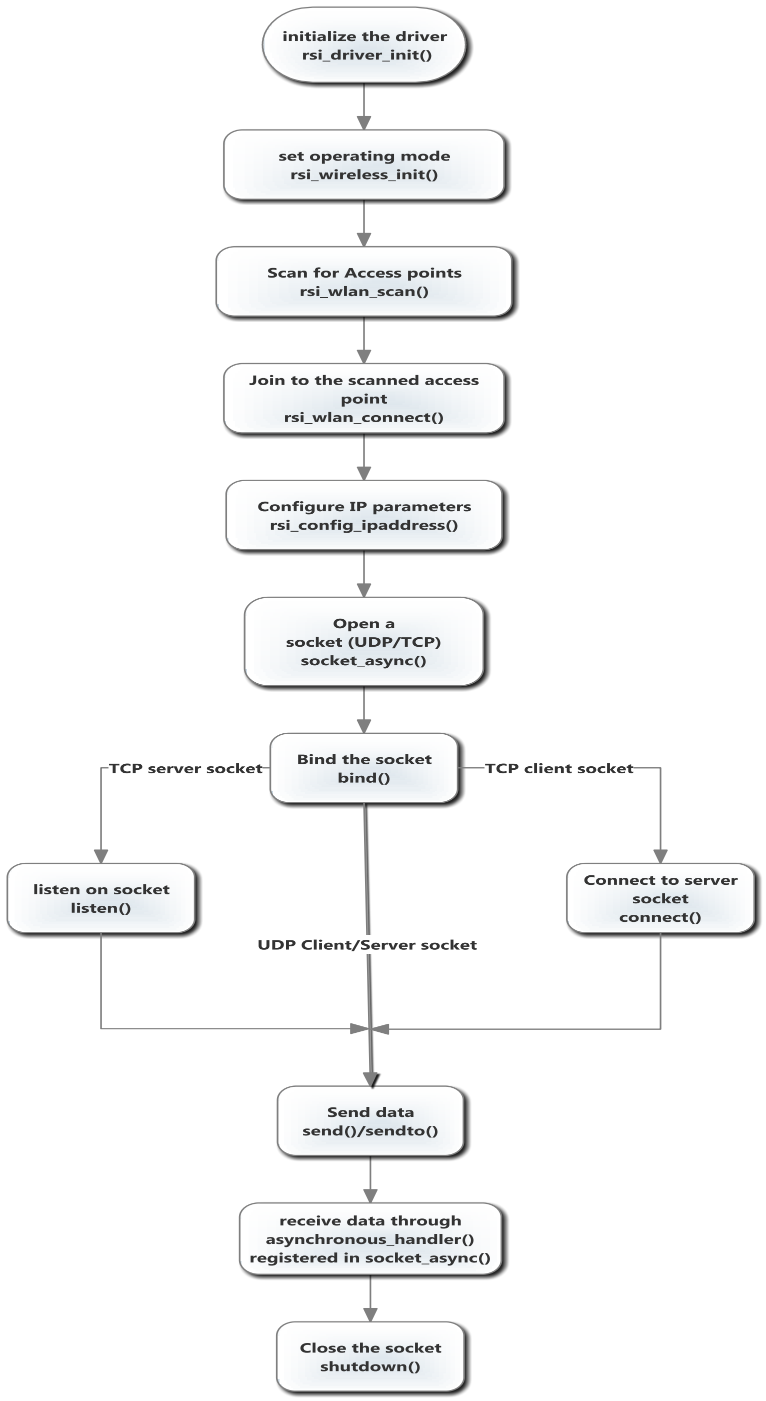


Figure 3 : API flow to configure module in station mode

4.5.2 Access point mode

The following flowchart briefs the sequence of API calls to configure module in access point mode. Edit `rsi_wlan_config.h` for the required features.

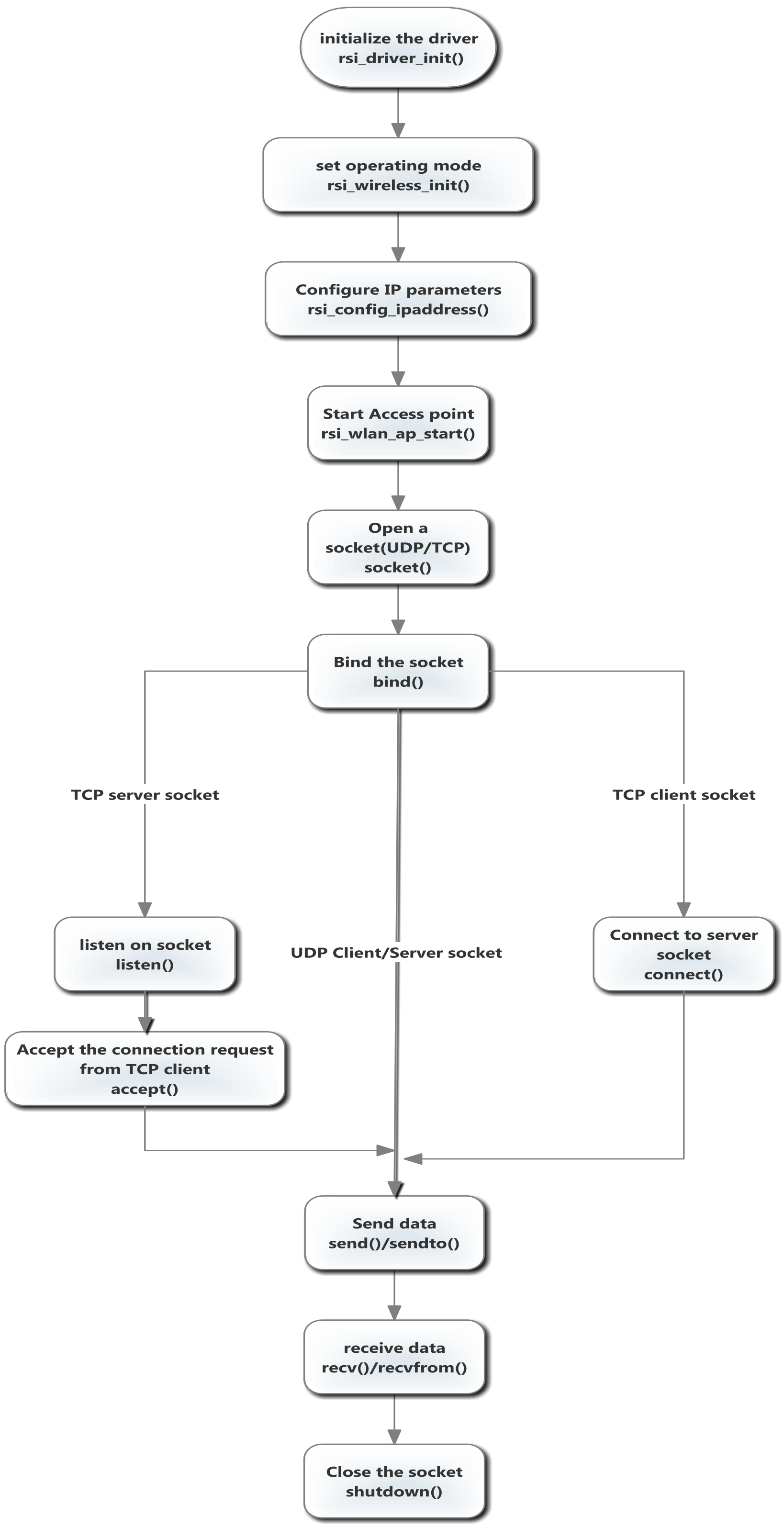


Figure 4 : API flow to configure module in Access point mode

5 BT-Classic and BT-LE Common Features

5.1 Power Save

5.1.1 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.

5.2 Power Save Operations

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

5.2.1.1 Power Save Mode 0

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

5.2.1.2 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 RS9113-WiSeConnect 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

Table 1: Messages from module in Power Save mode 2

Command Description	Binary Mode
"ACK"	0xDE

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.

5.2.1.3 Power Save mode 8

In Power save mode 8, both RF and SOC of RS9113-WiSeConnect 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

Table 3: Messages from module in Power Save mode 8

Command Description	Binary Mode
“ACK”	0xDE

Table 4: Message from host in Power Save mode 8

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

Note:

1) Power save disable command has to be given before changing the state from standby to remaining states and vise-versa.

Ex: 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.

2) For Page scan, Inquiry scan, sniff parameters related information please verify Bluetooth protocol specification document.
3) 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).

6 BT API

This section contains description about BT APIs 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

6.1 GAP API

This section describes the GAP APIs.

6.1.1 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 is used to request the local device name.

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	Name length
name	This is an array which consists name of the local device. Max 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

On Success : 0

On Failure : non zero

6.1.2 rsi_bt_set_local_name

Prototype

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

Description

This API is used to sets the local device name.

Parameters

Parameter	**Description**
`local_name`	Name to be set to the local device.

Return Values

On Success: 0

On Failure : non zero

6.1.3 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 is used to request the local COD name.

Parameters

Parameter	**Description**
`class_of_device`	Class of device

Return Values

On Success: 0

On Failure : non zero

6.1.4 rsi_bt_get_local_class_of_device

Prototype

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

Description

This API is used to request the local COD name.

Parameters

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

Return Values

On Success: 0

On Failure : non zero

6.1.5 rsi_bt_start_discoverable

Prototype

```
int32_t rsi_bt_start_discoverable(void)
```

Description

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

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.6 rsi_bt_start_limited_discoverable

Prototype

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

Description

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

Parameters

Parameter	Description
`time_out_ms`	Limited discovery mode time_out in ms.

Return Values

On Success: 0

On Failure : non zero

6.1.7 rsi_bt_stop_discoverable

Prototype

```
int32_t rsi_bt_stop_discoverable(void)
```

Description

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

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.8 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.

Parameters

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

Return Values

On Success: 0

On Failure : non zero

6.1.9 rsi_bt_set_connectable

Prototype

```
int32_t rsi_bt_set_connectable(void)
```

Description

This API is used to request the local device to set connectablity mode.

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.10 rsi_bt_set_non_connectable

Prototype

```
int32_t rsi_bt_set_non_connectable(void)
```

Description

This API is used to set the BT Module in non-connectable mode.

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.11 rsi_bt_get_connectable_status

Prototype

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

Description

This API is used to get BT Module connectablity status.

Parameters

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

Return Values

On Success: 0

On Failure : non zero

6.1.12 rsi_bt_enable_authentication

Prototype

```
int32_t rsi_bt_enable_authentication(void)
```

Description

This API is used to enable authentication.

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.13 rsi_bt_disable_authentication

Prototype

```
int32_t rsi_bt_disable_authentication(void)
```

Description

This API is used to disable authentication.

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.14 rsi_bt_get_authentication

Prototype

```
int32_t rsi_bt_get_authentication(void)
```

Description

This API is used to initiate authentication.

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.15 rsi_bt_remote_name_request_async

Prototype

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

Description

This API is used know the remote device name.

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`	BD address of remote device. Which an array max length is 6.
`Name_length`	Length of remote device name.
`Remote_device_name`	Name of remote device.

Parameters

Parameter	Description
`remote_dev_addr`	Remote device address
`bt_event_remote_device_name`	This parameter is a response buffer to hold the name of remote device. This is a structure parameter of `rsi_bt_event_remote_device_name_s`

Return Values

On Success: 0

On Failure : non zero

6.1.16 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.

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Return Values

On Success: 0

On Failure : non zero

6.1.17 rsi_bt_inquiry

Prototype

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

Description

This API is used to start inquiry.

Parameters

Parameter	Description
inquiry_type	Inquiry type.
inquiry_duration	Dduration of inquiry
max_devices	Maximum number of devices allowed to inquiry

Return Values

On Success: 0

On Failure : non zero

6.1.18 rsi_bt_cancel_inquiry

Prototype

```
int32_t rsi_bt_cancel_inquiry(void)
```

Description

This API is used to cancel inquiry.

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.19 rsi_bt_set_eir_data

Prototype

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

Description

This API is used to set Extended Inquiry Response data.

Parameters

Parameter	Description
fec_required	FEC Required
data_length	length of data
eir_data	EIR data which is an array which stores data upto 200 Bytes.

NOTE :
Currently EIR data supports upto 200 bytes.

Return Values

On Success: 0

On Failure : non zero

6.1.20 rsi_bt_connect

Prototype

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

Description

This API is used to initiate the connection request

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Return Values

On Success: 0

On Failure : non zero

6.1.21 rsi_bt_cancel_connect

Prototype

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

Description

This API is used to cancel the connection request.

Parameters

Parameter	Description
remote_dev_address	Remote device address

Return Values

On Success: 0

On Failure : non zero

6.1.22 rsi_bt_disconnect

Prototype

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

Description

This API is used to disconnect the physical connection.

Parameters

Parameter	Description
remote_dev_address	Remote device address

Return Values

On Success: 0

On Failure : non zero

6.1.23 rsi_bt_set_ssp_mode

Prototype

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

Description

This API is used to enable/disable Simple Secure Profile (SSP) mode.

Parameters

Parameter	Description
pair_mode	This parameter is used to enable or disable SSP mode. This parameters supports following values. 0 - Disable

	1 - Enable
IOcapability	IO capability request for SSP mode. This parameter supports following values. 0 – DisplayOnly 1 – DisplayYesNo 2 – KeyboardOnly 3 - NoInputNoOutput

Return Values

On Success: 0

On Failure : non zero

6.1.24 rsi_bt_accept_ssp_confirm

Prototype

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

Description

This API is used to give the confirmation for the passkey sent by local BT device at the time of pairing.

Parameters

Parameter	Description
remote_dev_address	Remote device address

Return Values

On Success: 0

On Failure : non zero

6.1.25 rsi_bt_reject_ssp_confirm

Prototype

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

Description

This API is used to reject the confirmation for the passkey sent by local BT device at the time of pairing.

Parameters

Parameter	Description
remote_dev_address	Remote device address

Return Values

On Success: 0

On Failure : non zero

6.1.26 rsi_bt_passkey

Prototype

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

Description

This API is used to send passkey or reject the incoming pass key request.

Parameters

Parameter	Description
`remote_dev_addr`	Remote device address
`passkey`	Passkey input
`reply_type`	Positive or negative reply

Return Values

On Success: 0

On Failure : non zero

6.1.27 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 is used to send pincode or reject the incoming pincode request.

Parameters

Parameter	Description
`remote_dev_addr`	Remote device address
`pin_code`	Pincode input
`reply_type`	Positive or negative reply

Return Values

On Success: 0

On Failure : non zero

6.1.28 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 is used to send either positive(along with the link key) or negative reply to the incoming linkkey request.

Parameters

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

Return Values

On Success: 0

On Failure : non zero

6.1.29 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 is used to request the role of local device.

Parameters

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

Return Values

On Success: 0

On Failure : non zero

6.1.30 rsi_bt_get_services_async

Prototype

```
int32_t rsi_bt_get_services_async(int8_t *remote_dev_addr,
            rsi_bt_resp_query_services_t *bt_resp_query_services)
```

Description

This API is used to query the remote device service list.

structure

```
typedef struct rsi_bt_resp_query_services_s

{

    uint8_t  num_of_services;
    uint8_t  reserved[3];;
    uint32_t uuid[32];
} rsi_bt_resp_query_services_t;
```

Structure Variables

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

Variables	Description
`num_of_services`	Number of services to fetch
`reserved`	Reserved
`uuid`	Service uuid to fetch

Parameters

Parameter	Description
`remote_dev_addr`	Remote device address
`bt_resp_query_services`	This parameter describes the response structure to hold the response of this API. This is a structure variable of a `rsi_bt_resp_query_services_s` structure.

Return Values

On Success: 0

On Failure : non zero

6.1.31 rsi_bt_search_service_async

Prototype

```
int32_t rsi_bt_search_service_async(int8_t *remote_dev_addr,
                                    uint32_t service_uuid)
```

Description

This API is used to search service of the given uuid.

Parameters

Parameter	Description
`remote_dev_addr`	Remote device address
`service_uuid`	UUID of the service to search

Return Values

On Success: 0

On Failure : non zero

6.1.32 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 is used to request the local device to enter into sniff mode.

Parameters

Parameter	Description
remote_dev_addr	Remote device address
sniff_max_intv	Sniff maximum interval
sniff_min_intv	Sniff minimum interval
sniff_attempt	Sniff attempt
sniff_tout	Sniff timeout

Return Values

On Success: 0

On Failure : non zero

6.1.33 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.

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Return Values

On Success: 0

On Failure : non zero

6.1.34 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 is used to request the device enter into sniff subrating 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

On Success: 0

On Failure : non zero

6.1.35 rsi_bt_get_rssi

Prototype

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

Description

This API is used to request the RSSI of the remote device.

Parameters

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

Return Values

On Success: 0

On Failure : non zero

6.1.36 rsi_bt_get_local_device_address

Prototype

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

Description

This API is used to request the local device address.

Parameters

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

Return Values

On Success: 0

On Failure : non zero

6.1.37 rsi_bt_spp_init

Prototype

```
int32_t rsi_bt_spp_init(void)
```

Description

This API is used to set the SPP profile mode.

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.38 rsi_bt_spp_connect

Prototype

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

Description

This API is used to initiate SPP profile level connection.

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Return Values

On Success: 0

On Failure : non zero

6.1.39 rsi_bt_spp_disconnect

Prototype

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

Description

This API is used to initiate SPP service level disconnection.

Parameters

Parameter	Description
remote_dev_addr	Remote device address

Return Values

On Success: 0

On Failure : non zero

6.1.40 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 is used to transfer data through SPP profile.

Parameters

Parameter	Description
`remote_dev_addr`	Remote device address
`Data`	Data for transmission
`Length`	Data length for transfer

Return Values

On Success: 0

On Failure : non zero

6.1.41 rsi_bt_init

Prototype

```
int32_t rsi_bt_init(void)
```

Description

This API is used to initialize the BT device.

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.42 rsi_bt_deinit

Prototype

```
int32_t rsi_bt_deinit(void)
```

Description

This API is used to deinitialize the BT device.

Parameters

None

Return Values

On Success: 0

On Failure : non zero

6.1.43 rsi_bt_set_antenna

Prototype

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

Description

This API is used to select either internal/external antenna on the chip.

Parameters

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

Return Values

On Success: 0

On Failure : non zero

6.1.44 rsi_bt_power_save_profile

Prototype

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

Description

This API is used to select power save profile mode for BT/BLE.

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 connected sleep 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

Parameter	Description
	connected sleep mode. In this sleep mode, SoC will go to sleep based on GPIO or Message, therefore handshake is required before sending data to the module. `RSI_SLEEP_MODE_8 (8)`: This is disconnected sleep mode. In this sleep mode, module will turn off the SoC. Since SoC is turn off, therefore handshake is required before sending data to the module.
`psp_type`	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

On Success: 0

On Failure : non zero

6.1.45 rsi_bt_set_feature_bitmap

Prototype

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

Description

This API is used to enable/disable the mentioned features.

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

On Success: 0

On Failure : non zero

6.1.46 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 is used to enable/disable the mentioned features.

Parameters

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

Return Values

On Success: 0

On Failure : non zero

6.2 Callback functions

6.2.1 GAP event callbacks description

6.2.1.1 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 will be called if the role change status event is received from the module.

Parameters

Variables	Description
`resp_status(out)`	Response status whether success or failure
`dev_addr(out)`	Remote device address
`role_change_status(out)`	Role change status whether success or failure

6.2.1.2 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 will be called if new connection complete is received from the module. This event will be given by 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

Parameters

Variables	Description
`resp_status`	Response status whether success or Error
`dev_addr`	Remote device address

6.2.1.3 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 module. This event will be given by module when either slave or master device may issue disconnect to the other.

Parameters

Variables	Description
resp_status	Response status whether success or Error
dev_addr	Remote device address

6.2.1.4 rsi_bt_on_scan_resp_t

Prototype

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

typedef struct rsi_bt_event_inquiry_response_s

{

    uint8_t  inquiry_type;

    uint8_t  dev_addr[RSI_DEV_ADDR_LEN];

    uint8_t  name_length;

    uint8_t   remote_device_name[RSI_BT_DEVICE_NAME_LEN];

    uint8_t  cod[3];

    uint8_t  rssi;

} rsi_bt_event_inquiry_response_t;
```

Description

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

Parameters

Variables	Description
resp_status	Response status whether success or Error
inquiry_type	Type of inquiry whether standard, extended, extended with RSSI
dev_addr	Remote device address
name_length	Length of remote device name
remote_device_name	Remote device name
Cod	Class of device
Rssi	RSSI of module from remote device

6.2.1.5 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 function will be called if the remote name request command response is received from the module.

Parameters

Variables	Description
resp_status	Response status whether success or Error
dev_addr	Remote device address
name_length	Length of remote device name
remote_device_name	Name of remote device

6.2.1.6 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 will be called if passkey display request is received from the module.

Parameters

Variables	Description

`resp_status`	Response status whether success or Error
`dev_addr`	Remote device address
`passkey`	Passkey to be displayed on our module side

6.2.1.7 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 function will be called if the remote name request canacel command response is received from the module.

Parameters

Variables	Description
`resp_status`	Response status whether success or Error
`dev_addr`	Remote device address

6.2.1.8 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 will be called if user confirmation request is received from the module. User has to give rsi_bt_accept_ssp_confirm or rsi_bt_reject_ssp_confirm command upon reception of this event.

Parameters

Variables	Description

resp_status	Response status whether success or Error
dev_addr	Remote device address
confirmation_value	Passkey to be confirmed

6.2.1.9 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 will be called if pincode request is received from the module. User has to give rsi_bt_accept_pincode_request command upon reception of this event.

Parameters

Variables	Description
resp_status	Response status whether success or Error
dev_addr	Remote device address

6.2.1.10 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 will be called if passkey request is received from the module. User has to give rsi_bt_passkey command upon reception of this event.

Parameters

Variables	Description
resp_status	Response status whether success or Error

dev_addr	Remote device address

6.2.1.11 rsi_bt_on_inquiry_complete_t

prototype

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

Description

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

Parameters

Variables	Description
resp_status	Response status whether success or Error

6.2.1.12 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 will be called if authentication complete indication is received from the module.

Parameters

Variables	Description
resp_status	Response status whether success or Error
dev_addr	Remote device address

6.2.1.13 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 function will be called if linkkey request is received from the module. User has to give linkkey reply command upon reception of this event.

Parameters

Variables	Description
`resp_status`	Response status whether success or Error
`dev_addr`	Remote device address

6.2.1.14 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 will be called if SSP complete status is received from the module.

Parameters

Variables	Description
`resp_status`	Response status whether success or Error
`dev_addr`	Remote device address
`status`	SSP mode connection with remote device is success or failure.

6.2.1.15 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 will be called if linkkey save is received from the module.

Parameters

Variables	Description
`resp_status`	response status whether success or Error
`dev_addr`	Remote device address
`linkkey`	Linkkey to be saved

6.2.1.16 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 will be called if the get services command response is received from the module.

Parameters

Variables	Description
`resp_status`	Response status whether success or failure
`num_of_services`	Number of services fetched
`reserved`	Reserved
`uuid`	Service uuid

6.2.1.17 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 will be called if the search service command response is received from the module.

Parameters

Variables	Description
`resp_status`	Response status whether success or failure
`remote_dev_addr`	Remote device address
`Status`	Search service status whether success or failure

6.2.1.18 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 will be called when the local device enters/exits the Sniff mode.

Parameters

Variables	Description
`resp_status`	Response status whether success or failure
`dev_addr`	Remote device address
`current_mode`	Indicates the current state of connection between the local device and the remote device i.e., Active mode/ Sniff mode
`reserved`	Reserved
`mode_interval`	Specify a time amount specific to each mode

6.2.1.19 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 will be called when Sniff subrating is enabled or the parameters are negotiated with the remote deivce.

Parameters

Variables	Description
`resp_status`	Response status whether success or failure
`dev_addr`	Remote device address
`max_tx_latency`	Maximum latency for data being transmitted from the local device to the remote device.
`min_remote_timeout`	The base sniff subrate timeout in baseband slots that the remote device shall use.
`min_local_timeout`	The base sniff subrate timeout in baseband slots that the local device shall use.

6.2.2 SPP profile event callback description

6.2.2.1 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 module. This event will be given by module when spp profile level connection happens from either side.

Parameters

Variables	Description
`resp_status`	Response status whether success or Error
`dev_addr`	Remote device address

6.2.2.2 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 module. This event will be given by module when spp profile level disconnection happens from either side.

Parameters

Variables	Description
resp_status	Response status whether success or Error
dev_addr	Remote device address

6.2.2.3 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 module. This event will be given by local device when it receives data from remote device.

Parameters

Variables	Description
resp_status	Data receive status
spp_receive	SPP profile received data structure

7 BLE API

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

7.1 Basic Structure defines

7.1.1 uuid_t

structure

```
typedef struct bt_uuid128 {
      UINT08      data1[4];
      UINT08      data2[2];
      UINT08      data3[2];
      UINT08      data4[8];
} UUID128;

typedef  UINT16     UUID16;

typedef  UINT32    UUID32;
```

/* Main UUID structure */

```
typedef struct bt_uuid {
      UINT08      size; // Size of the UUID
      UINT08      reserved[3];
      union bt_uuid_t {
            UUID128  val128;
            UUID32   val32;
            UUID16   val16;
      } val; // UUID value
} uuid_t;
```

Description

The size of a UUID (Universal Unique IDentifier) can be 2byte (16bit), 4byte (32bit) or 16byte (128bit). This structure defines the size of uuid and uuid value.

Structure Variables

Variables	Description
size	Size of uuid
reserved	Reserved
val	It is value of one of 3 types

	((128 bit, 32 bit or 16 bit) of UUIDs.

7.1.2 inc_service_data_t

structure

```
typedef struct inc_serv_data_s
{
      uint16_t  start_handle;
      uint16_t  end_handle;
      uuid_t    uuid;
} inc_serv_data_t;
```

Description

This structure describes the format of the include service attribute data

Structure Variables

Variables	Description
start_handle	Include service start handle
end_handle	Include service end handle
uuid	UUID value of the include service.

7.1.3 inc_service_t

structure

```
typedef struct inc_serv_s
{
      Uint16_t handle;
      uint8_t reserved[2];
      inc_serv_data_t inc_serv;
} inc_serv_t;
```

Description

This structure defines the include service attribute record.

Structure Variables

Variables	Description
handle	Include service defined handle
reserved	Reserved
inc_service	Include service attribute data structure.

7.1.4 char_serv_data_t

structure

```
typedef struct char_serv_data_s
{
      uint8_t   char_property;
      uint8_t   reserved;
      uint16_t  char_handle;
      uuid_t    char_uuid;
} char_serv_data_t;
```

Description

This structure defines the service characteristic data format.

Structure Variables

Variables	Description
char_property	Characteristic value property.
reserved	Reserved
char_handle	characteristic value handle
char_uuid	Characteristic value attributes uuid.

7.1.5 char_serv_t

structure

```
typedef struct char_serv_s
{
      uint16_t  handle;
      uint8_t reserved[2];
      char_serv_data_t char_data;
} char_serv_t;
```

Description

This structure defines the service characteristic attribute record format.

Structure Variables

Variables	Description
handle	Characteristic service attribute handle.
reserved	Reserved
char_data	Characteristic data structure variable

7.1.6 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	Attribute handle
reserved	Reserved
att_type_uuid	Attribute uuid(attribute type) .

7.1.7 rsi_ble_resp_profiles_list_t

Structure

```
typedef struct rsi_ble_resp_profiles_list_s
{
uint8_t                number_of_profiles;
uint8_t                reserved[3];
profile_descriptors_t  profile_desc[RSI_BLE_MAX_RESP_LIST];
} rsi_ble_resp_profiles_list_t;
```

Description

This structure defines GATT profiles list response structure.

Structure Variables

Variables	Description
number_of_profiles	Number of profiles found
reserved	Reserved
profile_desc	List of found profiles Max value is 5.

7.1.8 rsi_ble_resp_char_services_t

Structure

```
typedef struct rsi_ble_resp_char_serv_s
{
      uint8_t      num_of_services;
      uint8_t      reserved[3];
      char_serv_t  char_services[RSI_BLE_MAX_RESP_LIST];
} rsi_ble_resp_char_services_t;
```

Description

This is a GATT characteristic query service response structure.

Structure Variables

Variables	Description
num_of_services	Number of profiles found
reserved	Reserved
char_services	Characteristic service array. Max value is 5.

7.1.9 rsi_ble_resp_inc_services_t

Structure

```
typedef struct rsi_ble_resp_inc_serv
{
      uint8_t     num_of_services;
      uint8_t     reserved[3];
      inc_serv_t  services[RSI_BLE_MAX_RESP_LIST];
} rsi_ble_resp_inc_services_t;
```

Description

This is a GATT include service response structure.

Structure Variables

Variables	Description
num_of_services	Number of profiles found
reserved	Reserved
services	Include service list. Max value is 5.

7.1.10 rsi_ble_resp_att_value_t

Structure

```
typedef struct rsi_ble_resp_att_value_s
{
      uint8_t  len;
      uint8_t  att_value[100];
} rsi_ble_resp_att_value_t;
```

Description

This is a GATT attribute value response structure.

Structure Variables

Variables	Description
`len`	Length of the attribute value. Max length is 30.
`att_value`	Attribute value.

7.1.11 rsi_ble_resp_att_descs_t

Structure

```
typedef struct rsi_ble_resp_att_descs_s
{
      uint8_t     num_of_att;
      uint8_t     reserved[3];
      att_desc_t  att_desc[RSI_BLE_MAX_RESP_LIST];
} rsi_ble_resp_att_descs_t;
```

Description

This is a GATT attribute descriptors response structure.

Structure Variables

Variables	Description
`num_of_att`	Number of descriptors found
`reserved`	Reserved
`att_desc`	Attribute descriptors list. Max value is 5.

7.1.12 rsi_ble_req_add_att_t

Structure

```
typedef struct  rsi_ble_req_add_att_s
{
      void       *serv_handler;
      uint16_t    handle;
      uint16_t    reserved;
      uuid_t      att_uuid;
      uint8_t     property;
      uint8_t     data[31];
      uint16_t    data_len;
} rsi_ble_req_add_att_t;
```

Description

This structure is used to add new attributes to a service record

Structure Variables

Variables	Description
serv_handler	service handler
handle	Handle
att_uuid	Attribute type UUID
property	Attribute property
data	Attribute data. Max value is 31.
data_len	Attribute data len

7.1.13 rsi_ble_resp_local_att_value_t

Structure

```
typedef struct  rsi_ble_resp_local_att_value_s
{
      uint16_t    handle;
      uint16_t    data_len;
      uint8_t     data[31];
} rsi_ble_resp_local_att_value_t;
```

Description

Read local attribute value response structure.

Structure Variables

Variables	Description
`handle`	Attribute handle
`data_len`	Attribute value length
`data`	Attribute value (data). Max value is 31.

7.2 GAP API

This section describes the GAP APIs

7.2.1 rsi_ble_start_advertising

Prototype

```
int32_t rsi_ble_start_advertising(void)
```

Description

This API is used to start advertising.

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.2 rsi_ble_stop_advertising

Prototype

```
int32_t rsi_ble_stop_advertising(void)
```

Description

This API is used to stop advertising.

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.3 rsi_ble_start_scanning

Prototype

```
int32_t rsi_ble_start_scanning(void)
```

Description

This API is used to start scanning.

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.4 rsi_ble_stop_scanning

Prototype

```
int32_t rsi_ble_stop_scanning(void)
```

Description

This API is used to stop scanning.

Parameters

None

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.5 rsi_ble_connect

Prototype

```
int32_t rsi_ble_connect(uint8_t remote_dev_addr_type,
                        int8_t  *remote_dev_addr)
```

Description

This API is used to connect to the remote BLE device.

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

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.6 rsi_ble_connect_cancel

Prototype

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

Description

This API is used to cancel the connection to the remote BLE device.

Parameters

Parameter	Description
`remote_dev_address`	This parameter describes the device address of remote device

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.7 rsi_ble_disconnect

Prototype

```
int32_t rsi_ble_disconnect(int8_t *remote_dev_address)
```

Description

This API is used to disconnect with the remote BLE device.

Parameters

Parameter	Description
`remote_dev_address`	This parameter describes the device address of remote device

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.8 rsi_ble_get_device_state

Prototype

```
int32_t rsi_ble_get_device_state(uint8_t *resp)
```

Description

This API is used to get the local device state. State value is filled in "resp".

Parameters

Parameter	Description
`resp`	State is a 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

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.9 rsi_ble_set_advertise_data

Prototype

```
int32_t rsi_ble_set_advertise_data(uint8_t *data, uint16_t
                                   data_len)
```

Description

This API is used to set the advertising data.

Parameters

Parameter	Description
data	The advertise data.
data_len	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:

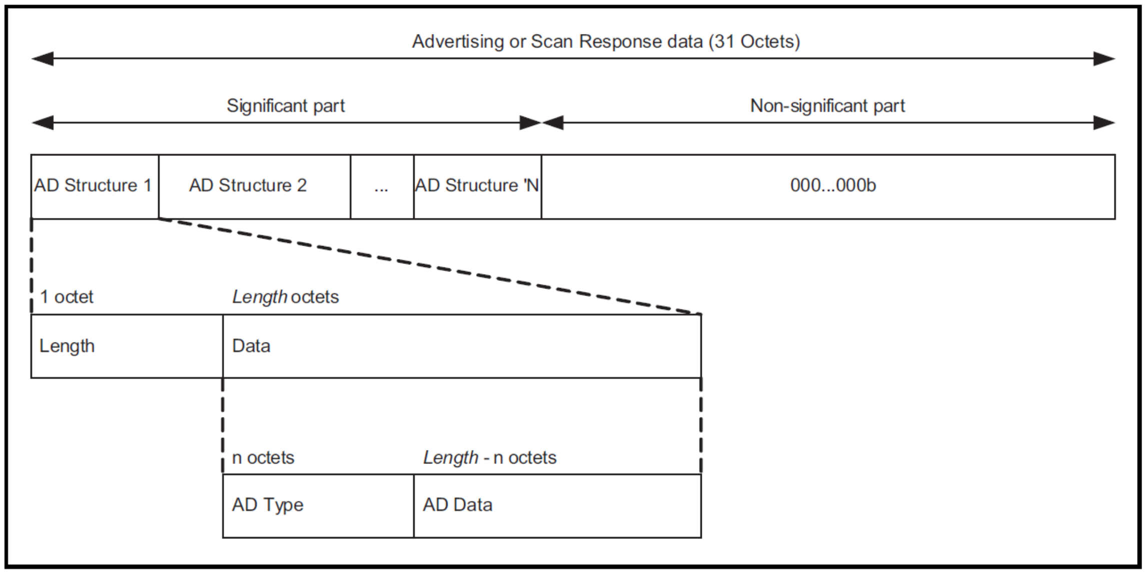
   1 octet 1 octet (length-1) octets

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.

Advertise and Scan response data format



For more information on this advertising data with types, information is available at following link:

https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.10 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 is used to request the SMP pairing process with the remote device.

Parameters

Parameter	Description
remote_dev_address	Remote device address
io_capability	Device input output capability

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.11 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 is used to send SMP pairing response during the process of pairing with the remote device.

Parameters

Parameter	Description
`remote_dev_address`	Remote device address
`io_capability`	Device input output capability

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.12 rsi_ble_smp_passkey

Prototype

```
int32_t rsi_ble_smp_passkey (uint8_t *remote_dev_address,
                             uint32_t passkey)
```

Description

This API is used to send SMP passkey during SMP pairing process with the remote device

Parameters

Parameter	Description
`remote_dev_address`	Remote device address
`passkey`	Key required in pairing process

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.13 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 is used to get the timeout value of the LE ping.

Parameters

Parameter	Description

`remote_dev_address`	Remote device address
`time_out`	This a response parameter which holds timeout value for Authentication payload command.

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.2.14 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 is used to get the timeout value of the LE ping.

Parameters

Parameter	Description
`remote_dev_address`	Remote device address
`time_out`	This input parameter which holds timeout value for Authentication payload command.

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command.

7.2.15 rsi_ble_set_random_address

Prototype

```
int32_t rsi_ble_set_random_address(void)
```

Description

This API is used to set the LE random device address.

Parameters

none

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command.

7.2.16 rsi_ble_encrypt

Prototype

```
int32_t rsi_ble_encrypt(uint8_t *key,uint8_t *data,uint8_t *resp)
```

Description

This API is used to Encrypt the data.

Parameters

Parameter	Description
Key	16Bytes key for Encryption of data.
Data	16Bytes of Data request to encrypt.
resp	Encrypted data

Return Values

On Success : 0

On Failure : if return value is less than 0

-4 : Buffer not available to serve the command

7.3 GATT API

This section describes the GATT APIs. Response payload for all the aysnc APIs will be indicated to application using the corresponding callback functions as described in the below sections. Response payload structure format is described along with the callback functions

7.3.1 GATT Client APIs

7.3.1.1 rsi_ble_get_profiles_async

Prototype

```
int32_t rsi_ble_get_profiles_async(uint8_t *dev_addr,
                   uint16_t start_handle,
                   uint16_t end_handle,
                   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

Parameters

Parameter	Description
dev_addr	Remote device address in ASCII string format
start_handle	Start handle (index) of the remote device's service records
end_handle	End handle (index) of the remote device's service records
p_prof_list	Profiles/services information will be filled in this structure after retrieving from the remote device.

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.2 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 is used to get the specific profile/service of the connected / remote device
- rsi_ble_on_profile_resp_t callback function will be called after the service characteristics response is received

Parameters

Parameter	Description
dev_addr	Remote device address
profile_uuid	Services/profiles are searched using profile_uuid
p_profile	Profile/service information will be filled in this structure after retrieving from the remote device.

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.3 rsi_ble_get_char_services_async

Prototype

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

Description

- This API is used to get service characteristics of the connected / remote device.
- rsi_ble_on_inc_services_resp_t callback function will be called after the include service characteristics response is received

Parameters

Parameter	Description
`dev_addr`	Remote device address
`start_handle`	Start handle (index) of the remote device's service records
`end_handle`	End handle (index) of the remote device's service records
`p_char_service_list`	service characteristics details are filled in this structure

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.4 rsi_ble_get_inc_services_async

Prototype

```
int32_t rsi_ble_get_inc_services_async(uint8_t *dev_addr,
            uint16_t start_handle,
            uint16_t end_handle,
            rsi_ble_resp_inc_services_t *p_inc_service_list);
```

Description

- This API is used to get the supported include services of the connected / remote device.
- rsi_ble_on_att_desc_resp_t callback function will be called after the service characteristics response received

Parameters

Parameter	Description
`dev_addr`	Remote device address
`start_handle`	Start handle (index) of the remote device's service records
`end_handle`	End handle (index) of the remote device's service records
`p_inc_service_list`	include service characteristics details are filled in this structure

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.5 rsi_ble_get_char_value_by_uuid_async

Prototype

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

Description

- This API is used to get the characteristic value by UUID (char_uuid).
- rsi_ble_on_read_resp_t callback function will be called upon receiving the attribute value

Parameters

Parameter	Description
`dev_addr`	Remote device address
`start_handle`	Start handle (index) of the remote device's service records
`end_handle`	End handle (index) of the remote device's service records

`char_uuid`	UUID of the characteristic
`p_char_val`	characteristic value is filled in this structure

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.6 rsi_ble_get_att_descriptors_async

Prototype

```
int32_t rsi_ble_get_att_descriptors_async (uint8_t *dev_addr,
                                 uint16_t start_handle,
                                 uint16_t end_handle,
                                 rsi_ble_resp_att_descs_t  *p_att_desc)
```

Description

- This API is used to get the characteristic descriptors list from remote device**.**
- rsi_ble_on_att_desc_resp_t callback function will be called after the attribute descriptors response is received

Parameters

Parameter	Description
`dev_addr`	Remote device address
`start_handle`	Start handle (index) of the remote device's service records
`end_handle`	End handle (index) of the remote device's service records
`p_att_desc`	characteristic descriptors are filled in this structure

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.7 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 is used to get the attribute by handle.
- rsi_ble_on_read_resp_t callback function will be called upon receiving the attribute value

Parameters

Parameter	Description
dev_addr	Remote device address
handle	handle of attribute
p_att_val	Attribute value is filled in this structure.

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.8 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 is used to get multiple attribute values by using multiple handles
- rsi_ble_on_read_resp_t callback function will be called upon receiving the attribute value

Parameters

Parameter	Description
dev_addr	Remote device address
num_of_handlers	Number of handles in the list
handles	List of attribute handles
p_att_vals	Attribute values are filled in this structure

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.9 rsi_ble_get_long_att_value_async

Prototype

```
int32_t rsi_ble_get_long_att_value_async (uint8_t *dev_addr,
                             uint16_t handle,
                             uint16_t offset,
                             rsi_ble_resp_att_value_t *p_att_vals)
```

Description

- This API is used to get the long attribute value by using handle and offset
- rsi_ble_on_read_resp_t callback function will be called upon receiving the attribute value

Parameters

Parameter	Description
`dev_addr`	Remote device address
`handle`	Attribute handle
`offset`	Offset with in the attribute value
`p_att_vals`	Attribute value is filled in this structure

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.10 rsi_ble_set_att_value_async

Prototype

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

Description

- This API is used to set attribute value.

- rsi_ble_on_write_resp_t callback function will be called if the attribute set action is completed

Parameters

Parameter	Description
dev_addr	Remote device address
handle	Attribute handle
data_len	Attribute value length
p_data	Attribute value

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.11 rsi_ble_set_att_cmd

Prototype

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

Description

- This API is used to set attribute value with out waiting for any ack from remote device
- rsi_ble_on_write_resp_t callback function will be called if the attribute set action is completed

Parameters

Parameter	Description
dev_addr	Remote device address
handle	Attribute handle
data_len	Attribute value length
p_data	Attribute value

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.12 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 is used to set long attribute value.
- rsi_ble_on_write_resp_t callback function will be called if the attribute set action is completed

Parameters

Parameter	Description
dev_addr	Remote device address
handle	attribute handle
Offset	attribute value offset
data_len	Attribute value length
p_data	Attribute value

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.13 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 is used to prepare attribute value.
- rsi_ble_on_write_resp_t callback function will be called if the prepare attribute write action is completed

Parameters

Parameter	Description
`dev_addr`	Remote device address
`handle`	attribute handle
`offset`	attribute value offset
`data_len`	Attribute value length
`p_data`	Attribute value

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.1.14 rsi_ble_execute_write_async

Prototype

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

Description

- This API is used to execute the prepared attribute values.
- rsi_ble_on_write_resp_t callback function will be called if the execute attribute write action is completed

Parameters

Parameter	Description
`dev_addr`	Remote device address
`exe_flag`	Execute flag to write the values or not

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.2 GATT Server APIs

7.3.2.1 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 is used to add a new service to local GATT Server.

Parameters

Parameter	Description
serv_uuid	New service uuid value
num_of_attributes	Number of attributes required in the service
total_data_size	Total data size required in the service
p_resp_serv	New service handler is filled in this structure

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.2.2 rsi_ble_add_attribute

Prototype

```
int32_t rsi_ble_add_attribute (rsi_ble_req_add_att_t
                              *p_attribute)
```

Description

This API is used to add a new attribute to a specific service.

Parameters

Parameter	Description
p_attribute	This is used to add a new attribute to the service.

Return Values

On Success: 0

On Failure:

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.2.3 rsi_ble_set_local_att_value

Prototype

```
int32_t rsi_ble_set_local_att_value (uint16_t  handle,
                                     uint16_t  data_len,
                                     uint8_t *p_data)
```

Description

This API is used to change the local attribute value.

Parameters

Parameter	Description
`handle`	local attribute handle
`data_len`	Attribute value length
`p_data`	Attribute value

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.2.4 rsi_ble_get_local_att_value

Prototype

```
int32_t rsi_ble_get_local_att_value (uint16_t  handle,
rsi_ble_resp_local_att_value_t *p_resp_local_att_val)
```

Description

This API is used to get the local attribute value

Parameters

Parameter	Description
`handle`	local attribute handle
`p_resp_local_att_val`	local attribute value

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.3.2.5 rsi_ble_gatt_read_response

Prototype

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

Description

This API is used to send local attribute value to the remote device.

Parameters

Parameter	Description
`dev_addr`	Remote device BD Address
`read_type`	0-Read Response 1- Read blob response
`handle`	Local attribute start handle
`offset`	Local attribute value start offset
`length`	Local attribute value length
`data`	Local attribute value

Return Values

On Success : 0

On Failure :

if return value is less than 0

-4 : Buffer not available to serve the command

7.4 Callback functions

7.4.1 GAP register callbacks

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

7.4.1.1 rsi_ble_gap_register_callbacks

Prototype

```
void rsi_ble_gap_register_callbacks (
     rsi_ble_on_adv_report_event_t   ble_on_adv_report_event,
     rsi_ble_on_connect_t            ble_on_conn_status_event,
     rsi_ble_on_disconnect_t         ble_on_disconnect_event,
     rsi_ble_on_le_ping_payload_timeout_t timeout_expired_event);
```

Description

This API used to register GAP callbacks.

Parameters

Parameter	Description
`ble_on_adv_report_event`	Advertise report callback
`ble_on_conn_status_event`	Connection status callback
`ble_on_disconnect_event`	Disconnection status callback
`timeout_expired_event`	Ping payload timeout callback

Return Values

None

For more information about each call back please refer GAP Callback Descriptions section.

7.4.2 GAP event callback descriptions

7.4.2.1 rsi_ble_event_adv_report_t

Structure

```
typedef struct rsi_ble_event_adv_report_s
{
      uint8_t dev_addr_type;
      uint8_t dev_addr[RSI_DEV_ADDR_LEN];
      uint8_t adv_data_len;
      uint8_t adv_data[RSI_MAX_ADV_REPORT_SIZE];
      uint8_t rssi;
}rsi_ble_event_adv_report_t;
```

Prototype

```
typedef void (*rsi_ble_on_adv_report_event_t)
(rsi_ble_event_adv_report_t *rsi_ble_event_adv)
```

Description

This event occurs when rsi_ble_start_scanning command is issued. This callback is called when an advertising event is raised from the module and advertising report is filled in rsi_ble_event_adv structure.

Structure Variables

Variables	Description
`dev_addr_type`	Address type of the advertising device
`dev_addr`	Device address of the advertising device.
`Rssi`	Signal strength
`Adv_data_len`	Raw advertisement data length
`Adv_data`	advertisement data

7.4.2.2 rsi_ble_event_conn_status_t

Structure

```
typedef struct rsi_ble_event_conn_status_s
{
      uint8_t dev_addr_type;
      uint8_t dev_addr[RSI_DEV_ADDR_LEN];
      uint8_t status;
}rsi_ble_event_conn_status_t;
```

Prototype

```
typedef void (*rsi_ble_on_connect_t)
(rsi_ble_event_conn_status_t *rsi_ble_event_conn)
```

Description

This call back is called when the module gives the connection status event. And the status will be filled in rsi_ble_event_conn structure. This event will be given by 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

Structure Variables

Variables	Description
`dev_addr_type`	Address type of the connected device
`dev_addr`	Device address of the connected device.

`status`	status of the connection – It consists success/failure

7.4.2.3 rsi_ble_event_disconnect_t

Structure

```
typedef struct rsi_ble_event_disconnect_s
{
      uint8_t dev_addr[RSI_DEV_ADDR_LEN];
}rsi_ble_event_disconnect_t;
```

Prototype

```
typedef void (*rsi_ble_on_disconnect_event_t)
(rsi_ble_event_disconnect_t *rsi_ble_event_disconnect,
                             uint16_t reason)
```

Description

This callback is called when the disconnect event is raised from the module. This callback will be called in the following two scenarios:

- In case, disconnection is issued locally
- In case, disconnection is initiated from a remote device

Structure Variables

Variables	Description
`dev_addr`	Device address of the disconnected device.
`reason`	Reason for disconnection

7.4.2.4 rsi_ble_on_le_ping_payload_timeout_t

Structure

```
typedef struct rsi_ble_event_le_ping_time_expired_s
{
      //!uint8_t, address of the disconnected device
      uint8_t dev_addr[RSI_DEV_ADDR_LEN];
}rsi_ble_event_le_ping_time_expired_t;
```

Prototype

```
typedef void (*rsi_ble_on_le_ping_payload_timeout_t)
(rsi_ble_event_le_ping_time_expired_t
*rsi_ble_event_timeout_expired);
```

Description

This callback is called when the LE ping payload timeout event is raised from the module i.e when the timer exceeds threshold value our module get disconnected with remote device and raise this event to host.

Structure Variables

Variables	Description
dev_addr	Device address of the disconnected device.

7.4.3 GATT register callbacks

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

7.4.3.1 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 is used to register GATT response callbacks.

Parameters

Parameter	Description
ble_on_profiles_list_resp	Callback for rsi_ble_req_profiles command
ble_on_profile_resp	Callback for rsi_ble_req_profile command
ble_on_char_services_resp	Callback for rsi_ble_req_char_services command
ble_on_inc_services_resp	Callback for rsi_ble_req_inc_services command

ble_on_att_desc_resp	Callback for rsi_ble_req_att_descriptors command
ble_on_read_resp	Callback for all read requests command
ble_on_write_resp	Callback for all write commands
ble_on_gatt_event	Callback for gatt write event
ble_on_gatt_prepare_write_event	Callback for gatt prepare write event
ble_on_execute_write_event	Callback for gatt execute write event
ble_on_read_req_event	Callback for gatt read request event
ble_on_mtu_event	Callback for MTU size

Return Values

None

For more information about each call back please refer GATT response callbacks description section.

7.4.4 GATT Response callbacks description

7.4.4.1 rsi_ble_on_profiles_list_resp_t

Prototype

```
typedef void (*rsi_ble_on_profiles_list_resp_t) (
      uint16_t  resp_status,
      rsi_ble_resp_profiles_list_t  *rsi_ble_resp_profiles);
```

Structure

```
typedef struct rsi_ble_resp_profiles_list_s
{
      uint8_t number_of_profiles;
      uint8_t reserved[3];
     profile_descriptors_t profile_desc[RSI_BLE_MAX_RESP_LIST];
} rsi_ble_resp_profiles_list_t;
```

Description

This callback function will be 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	Number of profiles found
`reserved`	uint8_t	Reserved
`profile_desc`	profile_descriptors_t	Contains the profiles list. Maximum of 5 profiles are filled.

7.4.4.2 rsi_ble_on_profile_resp_t

Prototype

```
typedef void (*rsi_ble_on_profile_resp_t) (uint16_t
resp_status, profile_descriptors_t *rsi_ble_resp_profile);
```

Structure

```
typedef struct profile_descriptor_s

{

      uint8_t  start_handle[2];

      uint8_t  end_handle[2];

      uuid_t   profile_uuid;

} profile_descriptors_t ;
```

Description

This callback function will be 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	start handle.
`End_handle`	uint8_t	end handle.

profile_uuid	uuid_t	profile uuid.

7.4.4.3 rsi_ble_on_char_services_resp_t

Prototype

```
typedef void (*rsi_ble_on_char_services_resp_t) (
      uint16 resp_status,
      rsi_ble_resp_char_services_t *rsi_ble_resp_char_serv);
```

Structure

```
typedef struct rsi_ble_resp_char_service_s
{
      uint8_t      num_of_services;
      uint8_t      reserved[3];
      char_service_t  char_services[RSI_BLE_MAX_RESP_LIST];
} rsi_ble_resp_char_services_t;
```

Description

- This callback function will be called if the service characteristics response is received from the module.
- This callback has to be registered using rsi_ble_gatt_register_callbacks API.
- resp_status, contains the response status (Success(0) or Error code).

Structure Variables

Variables	Data type	Description
num_of_services	uint8_t	Number of characteristic services found
Reserved	uint8_t	Reserved
char_services	char_service_t	It contains the characteristic service list. Max value is 5.

7.4.4.4 rsi_ble_on_inc_services_resp_t

Prototype

```
typedef void (*rsi_ble_on_inc_services_resp_t) (
        uint16_t resp_status,
        rsi_ble_resp_inc_services_t *rsi_ble_resp_inc_serv);
```

Structure

```
typedef struct rsi_ble_resp_inc_service
{
        uint8_t      num_of_services;
        uint8_t      reserved[3];
        inc_service_t  services[RSI_BLE_MAX_RESP_LIST];
} rsi_ble_resp_inc_services_t;
```

Description

This callback uses GATT include service response structure.

- This callback function will be called if the include 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	number of include services found
`reserved`	uint8_t	Reserved
`services`	inc_service_t	List of include services. Max value is 5.

7.4.4.5 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 will be 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	number of descriptors found
reserved	uint8_t	Reserved
att_desc	att_desc_t	Attribute descriptors list. Max value is 5.

7.4.4.6 rsi_ble_on_read_resp_t

Prototype

```
typedef void (*rsi_ble_on_read_resp_t) (uint16_t resp_status,
       uint16_t resp_id,
       rsi_ble_resp_att_value_t *rsi_ble_resp_att_val);
```

Structure

```
typedef struct rsi_ble_resp_att_value_s
{
      uint8_t  len;
      uint8_t  att_value[100];
} rsi_ble_resp_att_value_t;
```

Description

- This callback function will be 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	Length of attribute value
`att_value`	uint8_t	Attribute values list, each attribute value is maximum of size 30.

7.4.4.7 ble_on_write_resp

Prototype

```
typedef void (*rsi_ble_on_write_resp_t) (uint16_t resp_status,
                                          uint16_t resp_id);
```

Description

- This callback function will be 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).

Parameters

Parameters	Data type	Description
`resp_id`	uint16_t	Response ID for corresponding write command
`status`	uint16_t	Status of corresponding writes command.

7.4.5 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

7.4.5.1 GATT Write event

Prototype

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

typedef void (*rsi_ble_on_gatt_write_event_t) (
uint16_t event_id, rsi_ble_event_write_t *rsi_ble_write);
```

Description

This event callback uses GATT Write event structure.

Structure Variables

Variables	Data type	Description
dev_addr	uint8	Remote device address
handle	uint8	Attribute handle.
length	uint8	Length of attribute value
att_value	uint8	Contains the attribute value. Max value is 100.

7.4.5.2 GATT Prepare Write event

Prototype

```
typedef struct rsi_ble_event_prepare_write_s
{
      uint8_t  dev_addr[RSI_DEV_ADDR_LEN];
      uint8_t  handle[2];
      uint8_t  offset[2];
      uint16_t  length;
      uint8_t  att_value[100];
} rsi_ble_event_prepare_write_t;

typedef void (*rsi_ble_on_gatt_prepare_write_event_t) (uint16_t
event_id, rsi_ble_event_prepare_write_t *rsi_ble_write);
```

Description

This event callback uses GATT Prepare Write event structure.

Structure Variables

Variables	Data type	Description
`dev_addr`	uint8	Remote device address
`handle`	uint8	Attribute handle.
`offset`	uint8	Value offset
`length`	uint8	Length of attribute value
`att_value`	uint8	Contains the attribute value. Max value is 100.

7.4.5.3 GATT Execute Write event

Prototype

```
typedef struct rsi_ble_execute_write_s {
    uint8_t dev_addr[RSI_DEV_ADDR_LEN];
    uint8_t exeflag;
} rsi_ble_execute_write_t;
typedef void (*rsi_ble_on_execute_write_event_t) (uint16_t
event_id, rsi_ble_execute_write_t *rsi_ble_execute_write);
```

Description

This event callback uses GATT Execute Write event structure.

Structure Variables

Variables	Data type	Description
`dev_addr`	uint8	Remote device address
`exeflag`	uint8	Execute flag set after prepare writes are completely sent

7.4.5.4 GATT Read request event

Prototype

```
typedef struct rsi_ble_read_req_s {
    uint8_t   dev_addr[RSI_DEV_ADDR_LEN];
    uint16_t  handle;
    uint8_t   type;
    uint8_t   reserved;
    uint16_t  offset;
} rsi_ble_read_req_t;
typedef void (*rsi_ble_on_read_req_event_t) (uint16_t
event_id, rsi_ble_read_req_t *rsi_ble_read_req);
```

Description

This event callback uses GATT Read request event structure.

Structure Variables

Variables	Data type	Description
dev_addr	uint8	Remote device address
handle	uint16	Attribute handle.
type	uint8	0 – Read request 1 – Read Blob request
offset	uint16	Offset of attribute value to be read

7.4.5.5 MTU event

Prototype

```
typedef struct rsi_ble_event_mtu_s {
    uint8_t   dev_addr[RSI_DEV_ADDR_LEN];
    uint16_t  mtu_size;
} rsi_ble_event_mtu_t


typedef void (*rsi_ble_on_mtu_event_t) (rsi_ble_event_mtu_t
*rsi_ble_event_mtu)
```

Description

This event callback uses MTU event structure.

Structure Variables

Variables	Data type	Description
dev_addr	uint8	Remote device address
mtu_size	uint16	MTU size

7.4.6 SMP register callbacks

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

7.4.6.1 rsi_ble_smp_register_callbacks

Prototype

```
void rsi_ble_smp_register_callbacks (
rsi_ble_on_smp_request_t   ble_on_smp_request_event,
rsi_ble_on_smp_response_t   ble_on_smp_response_event,
rsi_ble_on_smp_passkey_t   ble_on_smp_passkey_event,
rsi_ble_on_smp_failed_t   ble_on_smp_fail_event,
rsi_ble_on_encrypt_started_t
rsi_ble_on_encrypt_started_event);
```

Description

This API used to register SMP callbacks.

Parameters

Variables	Description
`ble_on_smp_request_event`	SMP request callback
`ble_on_smp_response_event`	SMP response callback
`ble_on_smp_passkey_event`	SMP passkey callback
`ble_on_smp_fail_event`	SMP failed callback
`rsi_ble_on_encrypt_started_event`	SMP encryption callback

Return Values

None

For more information about each call back please refer SMP callbacks description section.

7.4.7 SMP event Callbacks Declarations

7.4.7.1 rsi_ble_on_smp_request_t

Prototype

```
typedef struct rsi_bt_event_smp_req_s {
      //!uint8_t, address of remote device
    uint8_t  dev_addr[6];
} rsi_bt_event_smp_req_t;
typedef void (*rsi_ble_on_smp_request_t)
(rsi_bt_event_smp_req_t  *remote_dev_address);
```

Description

This callback is called when SMP request is received from the remote device.

Structure Variables

Variables	Description
`dev_addr`	Device address of the advertising device.

7.4.7.2 rsi_ble_on_smp_response_t

Prototype

```
typedef struct rsi_bt_event_smp_resp_s {
      //!uint8_t, address of remote device
    uint8_t  dev_addr[6];
} rsi_bt_event_smp_resp_t;
typedef void (*rsi_ble_on_smp_response_t)
(rsi_bt_event_smp_resp_t  *remote_dev_address);
```

Description

This callback is called when SMP response is received from the remote device.

Structure Variables

Variables	Description
`dev_addr`	Device address of the connected device.

7.4.7.3 rsi_ble_on_smp_passkey_t

Prototype

```
typedef struct rsi_bt_event_smp_passkey_s {
      //!uint8_t, address of remote device
    uint8_t  dev_addr[6];
} rsi_bt_event_smp_passkey_t;
typedef void (*rsi_ble_on_smp_passkey_t)
(rsi_bt_event_smp_passkey_t  *remote_dev_address);
```

Description

This callback is called when SMP passkey is received from the remote device.

Structure Variables

Variables	Description
`dev_addr`	Device address of the disconnected device.

7.4.7.4 rsi_ble_on_smp_failed_t

Prototype

```
typedef struct rsi_bt_event_smp_failed_s {
      //!uint8_t, address of remote device
```

```
    uint8_t  dev_addr[6];
} rsi_bt_event_smp_failed_t;
typedef void (*rsi_ble_on_smp_failed_t) (uint16_t resp_status,
rsi_bt_event_smp_failed_t  *remote_dev_address);
```

Description

This callback function will be called if the SMP process is failed with remote device.

Structure Variables

Variables	Description
`resp_status`	Response status whether success or ERROR
`dev_addr`	Device address of the disconnected device.

7.4.7.5 rsi_ble_on_encrypt_started_t

Prototype

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

Description

This callback function will be called if the encryption process is started with the remote device.

Structure Variables

Variables	Description
`resp_status`	Response status whether success or ERROR

7.5 Configuration parameters

This section contain description of configuration macro’s may need to change based on application requirement. These macro’s with default values are placed in “**`rsi_ble_config.h`**”.

7.5.1 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

Define	Meaning
	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

7.5.2 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

7.5.3 Configure connection parameters

Define	Meaning
LE_SCAN_INTERVAL	Interval between the start of two consecutive scan windows. It shall be a multiple of 0.625 ms Recommended value: 0x0100 (0x0100*0.625 = 160ms) Range: 0x0050 to 0x1000
LE_SCAN_WINDOW	During scanning, the Link Layer listens on an advertising channel index for the duration of the scan window. It shall be a multiple of 0.625 ms. Recommended value: 0x0050 (0x0050*0.625 = 50ms) Range: 0x0050 to 0x1000
CONNECTION_INTERVAL_MIN	The start of connection events are spaced regularly with an interval of connInterval. It shall be a multiple of 1.25 ms.

	Recommended value: 0x0050 (0x0050*0.625 = 50ms) Range: 0x0050 to 0x320
`CONNECTION_INTERVAL_MAX`	The start of connection events are spaced regularly with an interval of connInterval. It shall be a multiple of 1.25 ms. Recommended value: 0x0050 (0x0050*0.625 = 50ms) Range: 0x0050 to 0x320
`CONNECTION_LATENCY`	The conn_latency parameter defines the maximum allowed connection Latency. Recommended value: 0x0000
`SUPERVISION_TIMEOUT`	The supervision_tout parameter defines the link supervision timeout for the connection. Recommended value: 0x07D0 (*10) ms 0x07D0 (0x07D0 *10=20s) Range: 0x64 to 0xC80

7.6 BLE API call sequence examples

This section explains the sequence of API calls to configure the module that can act for different purposes. The following are the examples to configure the module.

7.6.1 BLE peripheral

The following flowchart briefs the sequence of API calls to configure module so that it can act as a peripheral and can connect with central device. Edit `rsi_ble_config.h` for the required features.

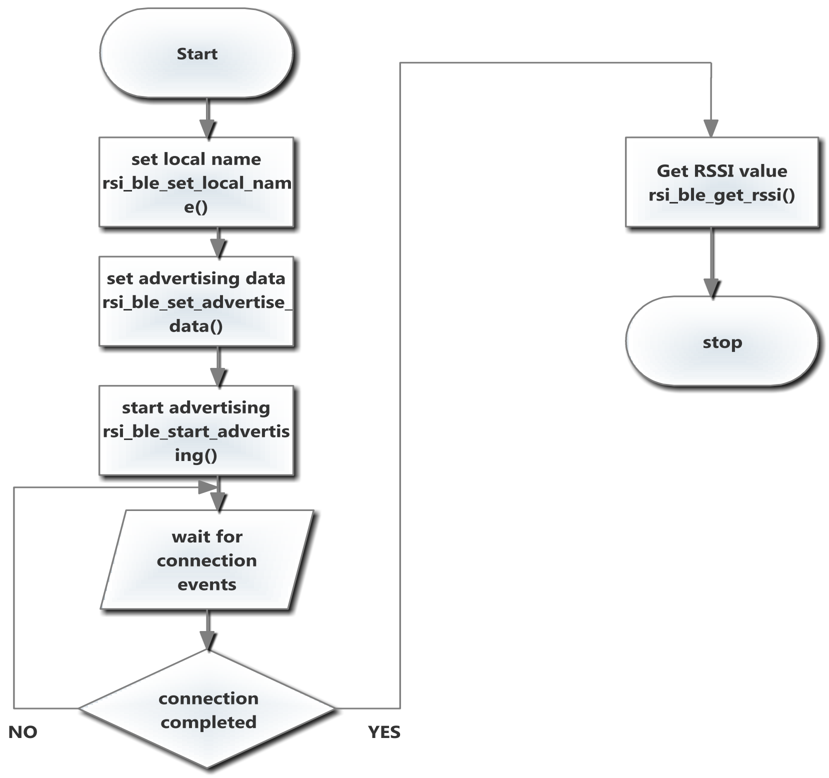

Start
set local name rsi_ble_set_local_name()
set advertising data rsi_ble_set_advertise_data()
start advertising rsi_ble_start_advertising()
wait for connection events
connection completed
NO
YES
Get RSSI value rsi_ble_get_rssi()
stop

Figure 5 : BLE peripheral application API flow

7.6.2 BLE central

The following flowchart briefs the sequence of API calls to configure module so that it can act as a central device. Edit `rsi_ble_config.h` for the required features.

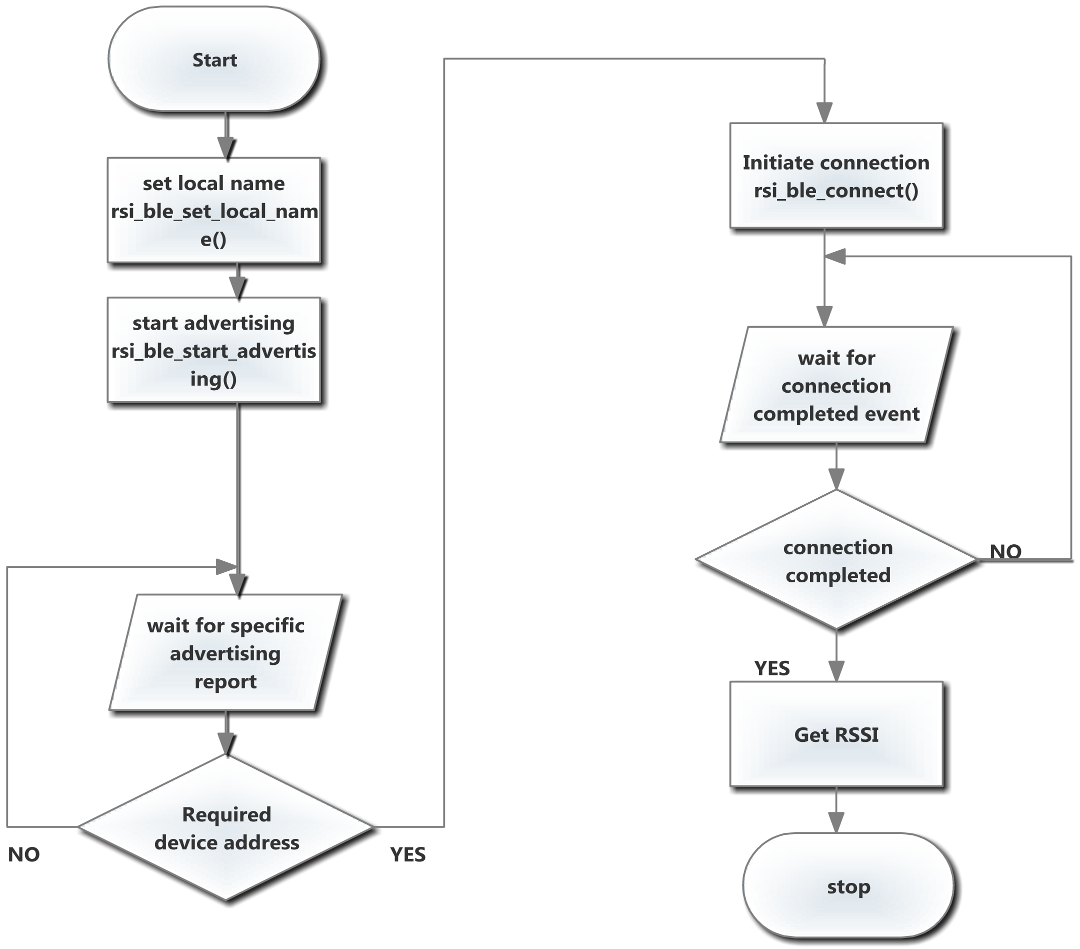

Start
set local name rsi_ble_set_local_name()
start advertising rsi_ble_start_advertising()
wait for specific advertising report
Required device address
NO
YES
Initiate connection rsi_ble_connect()
wait for connection completed event
connection completed
NO
YES
Get RSSI
stop

Figure 6 : BLE central application API flow

7.6.3 GATT client

The following flowchart briefs the sequence of API calls to configure module as a GATT client device which accesses a GATT server device. Edit `rsi_ble_config.h` for the required features.

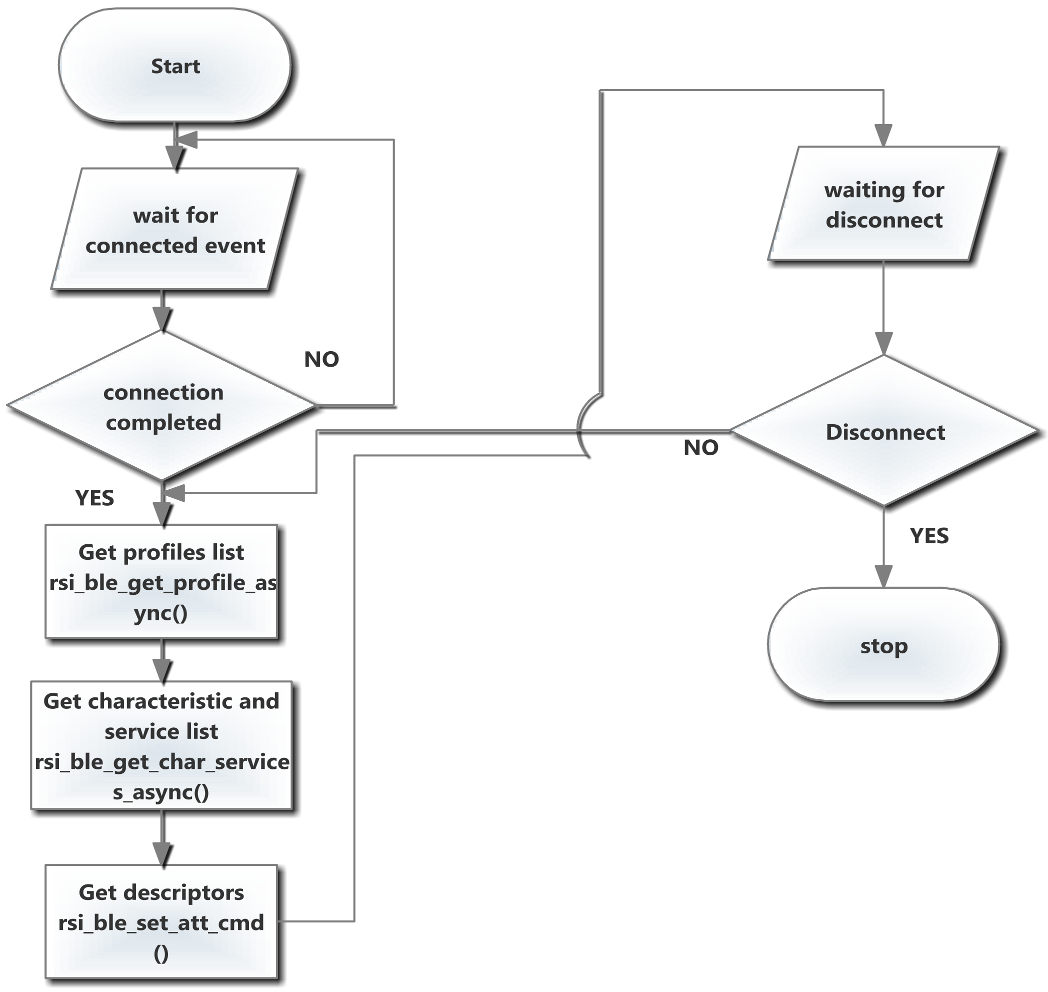

Start
wait for connected event
connection completed
NO
YES
Get profiles list rsi_ble_get_profile_async()
Get characteristic and service list rsi_ble_get_char_services_async()
Get descriptors rsi_ble_set_att_cmd ()
waiting for disconnect
Disconnect
NO
YES
stop

Figure 7 : BLE GATT client application API flow

7.6.4 GATT server

The following flowchart briefs the sequence of API calls to configure module as GATT server that can create different services by adding service attributes. Edit `rsi_ble_config.h` for the required features.

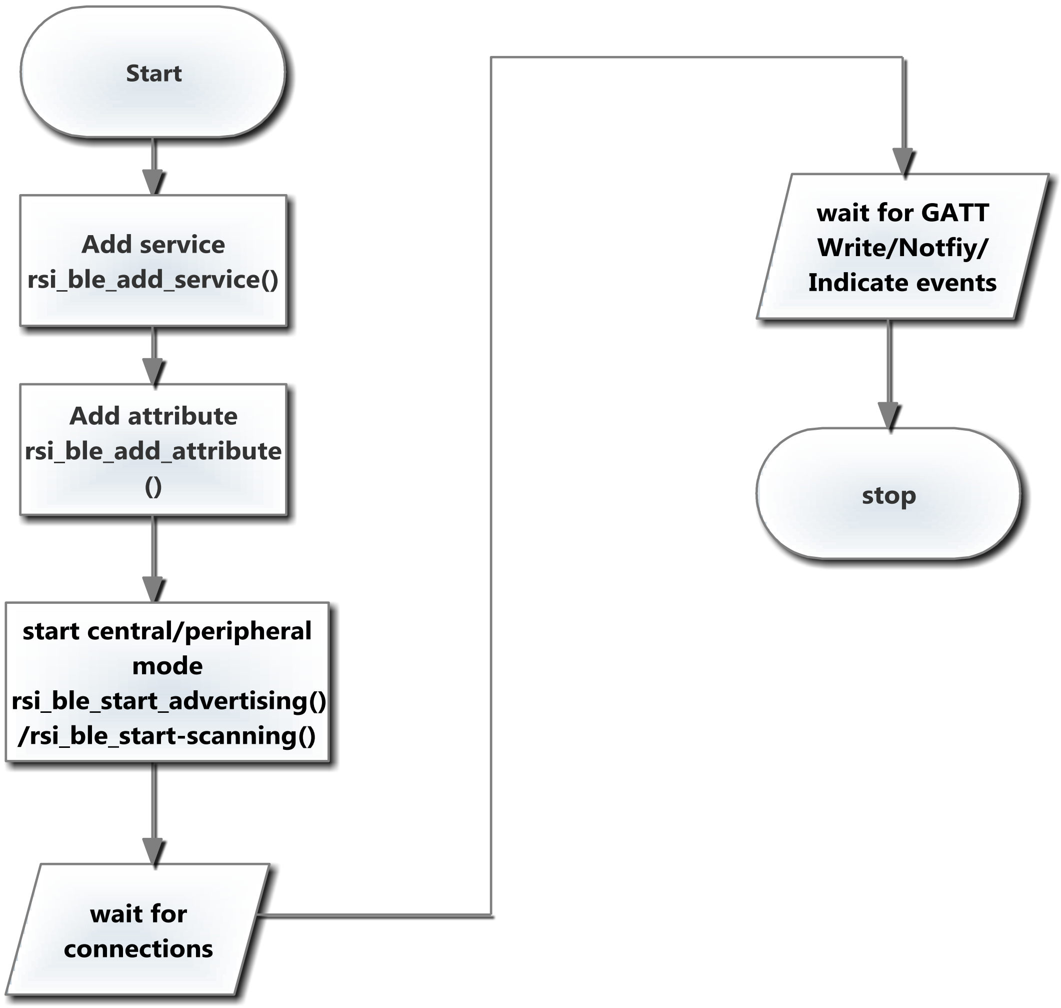

Start
Add service
rsi_ble_add_service()
Add attribute
rsi_ble_add_attribute()
start central/peripheral mode
rsi_ble_start_advertising()
/rsi_ble_start-scanning()
wait for connections
wait for GATT Write/Notfiy/ Indicate events
stop

Figure 8 : BLE GATT Server application API flow

8 ZIGBEE APIS

This section contains description about ZigBee API to initialize and configure module in ZigBee mode. this section provides an overview of all the APIs and features present in the stack.

8.1 Management Interface

8.1.1.1 rsi_zigb_init_stack

Prototype:

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

Description:

This API is used to initialize the ZigBee stack.

Parameters:

None

Return Value:

On Success : 0

On Failure   : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state.

Returns -4, if packet allocation fails.

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

8.1.1.2 rsi_zigb_reset_stack

Prototype:

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

Description:

This API is used to reset the ZigBee stack.

Parameters:

None

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state.

Returns -4, if packet allocation fails.

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

8.1.1.3 rsi_zigb_set_profile

Prototype:

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

Description:

This API is used to set the profile which we are going to use but valid for ZLL profile

Parameters:

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

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.4 rsi_zigb_update_sas

Prototype:

```
  struct {
            uint8_t     a_extended_pan_id[8];
            uint32_t    channel_mask;
            uint8_t     startup_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 is used to set the default startup attribute parameters

Parameters:

Parameters	Data type	Description
`Startup`	Startup_Attribute_Set_t	Pointer to startup 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 doesn't 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	A flag controlling the use of insecure join at startup.
scan_attempts	uint8_t	1 - 255	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 default value of 5
parent_retry_ threshold	uint8_t	3-10	The number of failed attempts to contact a parent that will cause a "find new parent" procedure to be initiated
a_trust_cente r_ address	uint16_t	0x0000-0xFFFF	Address of the network manager.
a_network_key	uint8_t[16]	Variable	The network key.
time_between_ scans	uint16_t	1 – 0xFFFF	Time between scans in milliseconds
rejoin_interv al	uint16_t	Max value:60	Rejoin interval in seconds
max_rejoin_ interval	uint16_t	Max value:3600	Max Rejoin interval in seconds
indirect_poll _rate	uint16_t	In msec	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	Address of the network manager.
a_trustcenter_master_key	uint8_t[16]	Variable	The Trust Center master key
a_preconfigured_link_key	uint8_t[16]	Variable	The Link key
end_device_bind_timeout	uint8_t	1-60	The time the coordinator will wait (in seconds) for a second end device bind request to arrive. The default value is 10.

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.5 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_t    config_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 is used to set the ZDO configuration.

Parameters:

Parameters	Data type	Description
`*pzdo_cnf`	ZDO_Configuration_Table_t	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	defines the time for which a coordinator or router device allows other devices to join to itself.

		join	
config_NWK_ secure_all_frames	uint8_t	0-Enable 1-Disabled	defines whether security is applied for incoming and outgoing network data frames or not
config_formation_attempts	uint8_t	1	the number of times the devices attempts for formation failure.
config_scan _duration	uint8_t	00-0xFE	The field indicates the duration of active scan while performing startup, join or rejoin the network.
config_join_ attempts	uint8_t	Default = 02	This field indicates the number of times join is retried once the join fails
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 field indicates whether a preconfigured key is already available in the device or not
a_config_trust_center_short_address	I uint16_t	Default 0x0000	This field 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)	The poll rate of end device while waiting for authentication.
config_switch_key_time	uint16_t	Default 0x06	The time after which active key sequence number is changed, once the device receives Switch Key request
config_security_level	uint8_t	0x05	The security level for outgoing and incoming network frames.
config_aps_ack_poll_time_out	uint8_t	0xFA(250 msec)	The maximum number of seconds to wait for an acknowledgement to a transmitted frame.
a_manufacturer_code	uint8_t[2]	0x0000 – 0xFFFF	Manufacturer code

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.6 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.

Parameters:

Parameters	Data type	Description
RadioChannel	uint8_t	Channel on which the network needs to be formed. Valid Channels are between 11 – 26 included
power	uint8_t	TX power to be used by the device. Range of TX power is about 0 – 12dBm.
pExtendedPanId	uint8_t	Pointer to extended PANID array of 8 bytes.

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.7 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.

Parameters:

Parameters	Data type	Description
`DeviceType`	uint8_t	0x01 – Router 0x02 – End-Device
`RadioChannel`	uint8_t	Channel on which the network needs to be formed. Valid Channels are between 11 – 26 included
`power`	uint8_t	TX power to be used by the device. Range of TX power is about 0 – 12dBm.
`pExtendedPanId`	uint8_t	Pointer to extended PANID array of 8 bytes.

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.8 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.

Parameters:

Parameters	Data type	Description
`PermitDuration`	uint8_t	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:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.9 rsi_zigb_leave_network

Prototype:

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

Description:

This API allows to perform self leave from the network.

Parameters:

None

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.10 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 specified type in the specified channel mask for the specified duration.

Parameters:

Parameters	Data type	Description
scanType	uint8_t	0x00 – Energy Detection scan 0x01 – Active scan
ChannelMask	uint32_t	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	A 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:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.11 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.

Parameters:

None

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.12 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.

Parameters:

None

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.13 rsi_zigb_stack_is_up

Prototype:

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

Description:

This API is used to know 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.

Parameters:
None

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.14 rsi_zigb_get_self_ieee_address

Prototype:

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

Description:

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

Parameters:

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

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

If the return value is other than above, please refer ZigBee error code table for description.

8.1.1.15 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 the given Extended address is self IEEE address.

Parameters:

Parameters	**Data type**	**Description**
`pIEEEAddress`	uint8_t*	Pointer to IEEE address (array of 8 bytes) which has to verified.

Return Value:

On Success : g_TRUE_c, if IEEE address is the local node's ID.

On Failure : g_FALSE_c

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.16 rsi_zigb_get_self_short_address

Prototype:

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

Description:

This api allows the application know self short address.

Parameters:

None

Return Value:

On Success: 16-bit short self address.

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

If the return value is other than above, please refer ZigBee error code table for description.

8.1.1.17 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 ser manufacturer code in the node descriptor.

Parameters:

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

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.18 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.

Structure Variables:

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

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.19 rsi_zigb_set_maxm_incoming_txfr_size

Prototype:

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

Description:

The api allows the application to specify the maximum incoming transfer size the device is capable of.

Parameters:

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

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.20 rsi_zigb_set_maxm_outgoing_txfr_size

Prototype:

```
int16_t rsi_zigb_set_maxm_outgoing_txfr_size(uint16_t);
```

Description:

The api allows the application to specify the maximum outgoing transfer size the device is capable of.

Parameters:

Parameters	Data type	Description
`Size`	uint16_t	The maximum outgoing transfer size for the local node. Range:0-128

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.21 rsi_zigb_set_operating_channel

Prototype:

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

Description:

The api allows the application to set the operating channel.

Parameters:

Parameters	Data type	Description
Channel	uint8_t	The desired radio channel. Range:11 to 26.

Return Value:

On Success : 0

On Failure : non-zero

Returns a non-zero value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.22 rsi_zigb_get_device_type

Prototype:

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

Description:

The api allows the application to get the device type.

Parameters:

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

Return Value:

On Success : 0,the dev_type parameter is updated.

On Failure : non-zero

Returns a non-zero value for unknown device or if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.23 rsi_zigb_get_operating_channel

Prototype:

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

Description:

The api allows the application to get the operating channel.

Parameters:

None.

Return Value:

On Success : The channel number.

On Failure : other than 11-26.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.24 rsi_zigb_get_short_panid

Prototype:

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

Description:

The api allows the application to get the short panid.

Parameters:
None.

Return Value:

On Success : 16 bit Pan Id.

On Failure : 0XFFFF.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.25 rsi_zigb_get_extended_panid

Prototype:

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

Description:

The api allows the application to get the extended pan id.

Parameters:

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

Return Value:

On Success : 0, the extended pan-id is updated in p_extended_panid.

On Failure : non-zero

Returns a negative value if command is issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.26 rsi_zigb_get_endpoint_id

Prototype:

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

Description:

The api allows the application to get the endpoint id.

Parameters:

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

Return Value:

On Success : The valid Endpoint ID located in the specified index.

On Failure : g_INVALID_ENDPOINT_ID_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.27 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:

The api allows the application to get the simple descriptor.

Parameters:

Parameters	Data type	Description
endpoint_id	uint8_t	The Endpoint on which these clusters are defined
simple_desc	Simple_Descriptor_t *	Pointer to simple descriptor of specified endpoint.

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

p_outclu ster_lis t	cluster_id _t	-	pointer to buffer holding output clusters.

Structure: Simple_Descriptor_t

Table 5: Simple Descriptor structure

Return Value:

On Success : g_TRUE_c.

On Failure : g_FALSE_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.28 rsi_zigb_set_simple_descriptor

Prototype:

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

Description:

The api allows the application to set the simple descriptor.

Parameters:

Parameters	Data type	Description
endpoint_id	uint8_t	The Endpoint on which the clusters are defined.
p_simple_desc	Simple_Descriptor_t *	Pointer to simple descriptor of specified endpoint.

Return Value:

On Success : g_TRUE_c.

On Failure : g_FALSE_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.29 rsi_zigb_get_endpoint_cluster

Prototype:

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

Description:

The api allows the application to read the endpoint's cluster in the specified list at the specified end-point index.

Parameters:

Parameters	Data type	Description
EndPointId	uint8_t	The 8-bit endpoint id whose cluster id needs to be retrieved
ClusterType	uint8_t	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	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:

On Success : Cluster id of the endpoint's simple descriptor located at the specified index.

On Failure : g_INVALID_CLUSTER_ID_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.30 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:

The api allows the application to get the 16-bit short address of the device for the given 64-bit IEEE address.

Parameters:

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

Return Value:

On Success : 16-bit short address of the corresponding 64-bit IEEE address if the address is known.

On Failure : INVALID_SHORT_ADDRESS

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.31 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:

The api allows the application to get the 64-bit IEEE address of the device for the given 16-bit Short address.

Parameters:

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

Return Value:

On Success : **g_TRUE_c,** if successfully retrieved IEEE address from neighbor table or Address map table.

On Failure : **g_FALSE_c**

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.32 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:

The api allows the application to read the Neighbor table entry in the specified index.

Parameters:

Parameters	Data type	Description
Index	uint8_t	Indicates index from where the neighbor table entry is to be retrieved.
neigbor_table	ZigBeeNeighbor TableEntry_t*	Pointer to location where the NeighbortableEntry needs to be copied.

Structure: ZigBeeNeighborTableEntry_t

Name	type	Range	Description
shortId	uint16_t	0x0000 – 0xffff	The neighbor's two byte short address
average Lqi	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.
incomin gCost	uint8_t	1-7	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 .
outgoin gCost	uint8_t	1-7	The outgoing cost for this neighbor, obtained from the most recently received neighbor exchange message from the neighbor
age	uint8_t	3-16	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
aIEEEAd dress	uint8_t[8]	-	The 8 byte IEEE address of the neighbor

Return Value:

On Success : **0**.

On Failure : **ZigBee_Invalid_Argument**

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.33 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:

The api allows the application to read the Routing table entry in the specified index.

Parameters:

Parameters	Data type	Description
Index	uint8_t	Indicates index from where the neighbor table entry is to be retrieved.
routing_table	ZigBeeRouting TableEntry_t *	Pointer to location where the Route table Entry needs to be copied.

Structure: ZigBeeNeighborTableEntry_t

Name	type	Range	Description
destAddr	uint16_t	0x0000 – 0xffff	short id of the destination
nextHop	uint16_t	0x0000 – 0xffff.	short address of the next hop to this destination
status	uint8_t	1-7	Indicates whether this entry is active (0), being discovered (1), or unused (0x3).
age	uint8_t	1-7	The number of seconds since this route entry was last used to send a packet
concentratorType	uint8_t	0-2	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, 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:

On Success : **0**.

On 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 failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.34 rsi_zigb_get_neighbor_table_entry_count

Prototype:

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

Description:

The api allows the application to know the count of active neighbor table entries.

Parameters:

None.

Return Value:

On Success : Total count of active neighbor table entries in the neighbor table.

On Failure : non-zero

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.35 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:

The api allows the application to read the 16-bit short address of the child in the specified index.

Parameters:

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

Return Value:

On Success : The child address.

On Failure : g_INVALID_ADDRESS_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.36 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:

The api allows the application to get the index for the specified 16-bit child address.

Parameters:

Parameters	Data type	Description
childShortAddr	uint16_t	The 16-bit short address whose index need to be determined

Return Value:

On Success : index of the child address received in the input parameter.

On Failure : m_NO_ENTRY_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.37 rsi_zigb_get_child_details

Prototype:

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

Description:

The api allows the application to get the child details at the specified child index.

Parameters:

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

Return Value:

On Success : **0,**

On Failure : ZigBeeUnknownDevice

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.38 rsi_zigb_end_device_poll_for_data

Prototype:

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

Description:

The api allows the application to poll the parent for data.

Parameters:

None.

Return Value:

On Success : **0,**

On Failure : ZigBee_Invalid_Call

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.39 rsi_zigb_read_count_of_child_devices

Prototype:

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

Description:

The api allows the application to read the number of child devices on the node.

Parameters:

None.

Return Value:

On Success **:** Number of children joined**.**

On Failure : negative value

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.40 rsi_zigb_read_count_of_router_child_devices

Prototype:

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

Description:

The api allows the application to read the number of child devices on the node.

Parameters:

None.

Return Value:

On Success : Number of router children joined.

On Failure : negative value

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.41 rsi_zigb_get_parent_short_address

Prototype:

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

Description:

The api allows the application to get the parent's 16 bit short address.

Parameters:

None.

Return Value:

On Success : parent short address.

On Failure : g_INVALID_ADDRESS_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.42 rsi_zigb_get_parent_ieee_address

Prototype:

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

Description:

The api allows the application to read it parent's 64-bit IEEE address.

Parameters:

Parameters	Data type	Description
ieee_addr	uint8_t*	Pointer to location where parent's 64-bit IEEE address should be copied

Return Value:

On Success : **0,**

On Failure : negative value

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.43 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:

The api allows the application to request energy scan be performed and its results returned. This request may only be sent by the current network manager and must be unicast, not broadcast.

Parameters:

Parameters	Data type	Description
DestAddr	uint16_t	Indicates the network address of the device to perform the scan. Range:0x0000-0xFFFF
ScanChannels	uint32_t	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	Indicates How long to scan on each channel. Allowed values are 0 – 5.
ScanRetr	uint16_t	Indicates The number of scans to be performed on each channel (1-8)

Return Value:

On Success : **0,**

On Failure : non-zero

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.44 rsi_zigb_broadcast_nwk_manager_request

Prototype:

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

Description:

The api allows the application to broadcasts a request to change the channel. This request may only be sent by the current Network manager.

Parameters:

Parameters	Data type	Description
NWKManagerShortAddr	uint16_t	Indicates the 16-bit network address of the Network Manager.
ActiveChannels	uint32_t	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:

On Success : **0,**

On Failure : non-zero

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.45 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:

The 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.

Parameters:

Parameters	Data type	Description
pIEEEAddrOfInterest	uint8_t*	Pointer to location of IEEE address whose 16-bit Network address is to be determined
RequestType	BOOL	boolean if TRUE indicates single device response if FALSE indicates extended device response.

StartIndex	uint8_t	Start index of the child devices list.

Return Value:
On Success : **0,**
On Failure : g_FAILURE_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.46 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:

The 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.

Parameters:

Parameters	Data type	Description
shortAddress	uint16_t	Pointer to location of short address whose IEEE address is to be determined.
RequestType	BOOL	TRUE indicates single device response if FALSE indicates extended device response.
StartIndex	uint8_t	The index of the first child to list in the response. Ignored if the RequestType is single device.
APSAckRequired	BOOL	TRUE indicates APS ack is required

Return Value:

On Success **: 0,**

On Failure : g_FAILURE_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.47 rsi_zigb_zdp_send_device_announcement

Prototype:

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

Description:

The api allows the application to send a broadcast for a ZDO Device announcement. Normally, it is NOT required to call this as the stack 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.

Parameters:

None.

Return Value:

On Success **: 0,**

On Failure : ZigBee_Device_Down.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.48 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:

The api allows the application to send a match descriptor request to a destination device.

Parameters:

Parameters	Data type	Description
shortAddress	uint16_t	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.
ProfileId	uint16_t	The application profileId to match
InClusterList	uint8_t *	The pointer to list of input clusters.
InClusterCnt	uint8_t	Number of input clusters.
OutClusterList	uint8_t *	The pointer to list of output clusters.
OutClusterCnt	uint8_t	Number of output clusters.
APSAckRequired	BOOL	TRUE indicates APS ack is required
dstAddress	uint16_t	Destination short address.

Return Value:

On Success : **0,**

On Failure : ZigBee_Failure

.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.49 rsi_zigb_active_endpoints_request

Prototype:

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

Description:

The api allows the application to send ZDP Active Endpoint request.

Parameters:

Parameters	Data type	Description
shortAddress	uint16_t	Device short address whose active endpoints needs to be obtained.
APSAckRequired	BOOL	TRUE indicates APS ack is required

Return Value:

On Success : **0,**

On Failure : g_FAILURE_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.50 rsi_zigb_zdp_send_power_descriptor_request

Prototype:

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

Description:

The api allows the application to power descriptor request.

Parameters:

Parameters	Data type	Description
shortAddress	uint16_t	Device short address whose power descriptor needs to be obtained.
APSAckRequired	uint8_t	TRUE indicates APS ack is required

Return Value:

On Success : **0,**

On Failure : g_FAILURE_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.51 rsi_zigb_zdp_send_node_descriptor_request

Prototype:

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

Description:

The api allows the application to node descriptor request.

Parameters:

Parameters	Data type	Description
shortAddress	uint16_t	Device short address whose node descriptor needs to be obtained.
APSAckRequired	uint8_t	TRUE indicates APS ack is required

Return Value:
On Success **: 0,**
On Failure : g_FAILURE_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails If the return value is greater than 0, please refer ZigBee error code table for description.

8.1.1.52 rsi_zigb_simple_descriptor_request

Prototype:

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

Description:

The api allows the application to request for the simple descriptor for a target device.

Parameters:

Parameters	**Data type**	**Description**
shortAddress	uint16_t	Device short address whose simple descriptor needs to be obtained. Range:0x0000 – 0xFFFF.
APSAckRequired	uint8_t	TRUE indicates APS ack is required

Return Value:
On Success **: 0,**
On Failure : g_FAILURE_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.1.1.53 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:

The api allows the application to get the address map table entry for the specified index.

Parameters:

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

Structure: APSME_Address_Map_Table_t

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

Return Value:

On Success : Address map table is updated.

On Failure : The 64 bit field is updated with all 0xFFs.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.2 Data Interface

8.2.1.1 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:

The api allows the application to to initiate APSDE data request to the specified destination address.

Parameters:

Parameters	**Data type**	**Description**
`msgType`	`uint8_t`	Type of transmission taking place.
`DestAddress`	`Address`	Address of the destination device

pAPSDERequest	ZigBeeAPSDEDa taRequest_t *	Pointer to memory where data request frame is stored.

Structure: Address

Name	type	Ran ge	Description
short_addr ess	uint16_t	0x00 – 0xff	16-bit short address.
IEEE_addre ss	uint8_t[8]	-	64-bit IEEE extended address.

Structure: ZigBeeAPSDEDataRequest_t

Name	type	Ran ge	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*(NsduL ength - apscMinHea der Overhe ad)	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:

On Success : ZigBee_Success.

On Failure :

ZigBee_Invalid_Argument/ ZigBee_No_Buffer.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.2.1.2 rsi_zigb_send_group_data

Prototype:

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

Description:

The api allows the application to initiate APSDE data request to the specified Group address.

Parameters:

Parameters	Data type	Description
`GroupAddress`	`GroupID`	Indicates the group id to which the data is transmitted.
`pAPSDERequest`	`ZigBeeAPSDEDataRequest_t`	Pointer to memory where

	*(refer rsi_zigb_send_unicast_data for the structure definition)	data request frame is stored.

Return Value:

On Success : ZigBee_Success.

On Failure :

ZigBee_Invalid_Argument/ ZigBee_No_Buffer.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.2.1.3 rsi_zigb_send_broadcast_data

Prototype:

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

Description:

The api allows the application to broadcast APSDE data request.

Parameters:

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

Return Value:

On Success : ZigBee_Success.

On Failure :

ZigBee_Invalid_Argument/ ZigBee_No_Buffer.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.2.1.4 rsi_zigb_get_max_aps_payload_length

Prototype:

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

Description:

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

Parameters:

None.

Return Value:

On Success : maximum APS payload length.

On Failure : negative value.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.3 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**

Table 6: Key Types

8.3.1.1 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:

The 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.

.

Parameters:

Parameters	Data type	Description
keytype	Security_Key_ Types	key type.(refer Table 4)

Return Value:

On Success : ZigBee_Success.

On Failure :

ZigBee_Invalid_Argument/ g_NO_KEY_c/ ZigBee_Failure

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.3.1.2 rsi_zigb_have_link_key

Prototype:

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

Description:

The api allows the application to check a link key is available for securing messages sent to the remote device.

Parameters:

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

Return Value:

On Success : g_TRUE_c.

On Failure : g_FALSE_c

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.3.1.3 rsi_zigb_request_link_key

Prototype:

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

Description:

The api allows the application to get the link key for the specified IEEE address.

Parameters:

Parameters	**Data type**	**Description**
TrustCenterIEEE Addr	uint8_t*	The IEEE address of the Trust Centre device.
PartnerIEEEAddr	uint8_t*	The IEEE address of the partner device.

Return Value:

On Success : ZigBee_Success.

On Failure : ZigBee_Failure/ ZigBee_Invalid_Argument.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.3.1.4 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:

The api allows the application to get the link key for the specified IEEE address.

Parameters:

Parameters	Data type	Description
`Index`	`uint8_t`	The index in the key table of the entry to get.
`keyStruct`	`ZigBeeKeyStructure_t *`	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	---	Type of key sent from host. It is one of key from the defined structure Security_Key_Types
key	uint8_t[16]	---	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:
On Success **:** ZigBee_Success.
On Failure : ZigBee_Invalid_Argument/ g_NO_KEY_c/ ZigBee_Failure.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.3.1.5 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:

The api allows the application to set an entry in the key table.

Parameters:

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

Return Value:

On Success : ZigBee_Success.

On Failure : m_NO_ENTRY_c.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.3.1.6 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:

The api allows the application to add a new entry in the key table or updates an existing entry with a new key.

Parameters:

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

Return Value:

On Success **:** ZigBee_Success.

On Failure : ZigBee_Invalid_Argument/ ZigBee_Failure.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.3.1.7 rsi_zigb_find_key_table_entry

Prototype:

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

Description:

The api allows the application to search the key table and find an entry matching the specified IEEE address and key type.

Parameters:

Parameters	Data type	Description
pIEEEAddress	uint8_t*	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	A boolean indicating whether to search for an entry containing a Link or Master Key.

Return Value:

On Success : ZigBee_Success.

On Failure : m_NO_ENTRY_c.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.3.1.8 rsi_zigb_erase_key_table_entry

Prototype:

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

Description:

The api allows the application to clear a single entry in the key table.

Parameters:

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

Return Value:

On Success : ZigBee_Success.

On Failure : m_NO_ENTRY_c.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.4 Binding Interface

8.4.1.1 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:

The api allows the application to set an entry in the binding table by copying the structure pointed to by pSetBindingEntry into the binding table.

Parameters:

Parameters	Data type	Description
pSetBindingEntry	ZDP_Bind_Request_t*	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
`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:
On Success **:** g_SUCCESS_c.
On Failure : g_FAILURE_c.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.4.1.2 rsi_zigb_get_binding_indices

Prototype:

```
int16_t rsi_zigb_get_binding_indices(uint8_t * binding_indices);
```

Description:

The api allows the application to read the active binding indices.

Parameters:

Parameters	Data type	Description
`binding_indices`	`uint8_t*`	Pointer to the list of binding indices of each uint8_t size.

Return Value:

On Success **:** ZigBee_Success.

On Failure : ZigBee_Invalid_Argument.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.4.1.3 rsi_zigb_delete_binding

Prototype:

```
int16_t rsi_zigb_delete_binding(uint8_t bindIndex);
```

Description:

The api allows the application to delete an entry in the binding table for the specified index.

Parameters:

Parameters	Data type	Description
`bindIndex`	`uint8_t`	Indicates the index which needs to be deleted.

Return Value:

On Success **:** ZigBee_Success.

On Failure : g_ZDP_Not_Permitted_c.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.4.1.4 rsi_zigb_is_binding_entry_active

Prototype:

```
int16_t rsi_zigb_is_binding_entry_active(uint8_t bindIndex);
```

Description:

The api allows the application to check whether the binding entry is active or not.

Parameters:

Parameters	Data type	Description
bindIndex	uint8_t	The index of a binding table entry.

Return Value:

On Success : g_TRUE_c.

On Failure : g_FALSE_c.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.4.1.5 rsi_zigb_clear_binding_table

Prototype:

```
int16_t rsi_zigb_clear_binding_table(void);
```

Description:

The api allows the application to clear all the binding table entries.

Parameters:

None.

Return Value:

On Success : ZigBee_Success.

On Failure : g_ZDP_Not_Permitted_c.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.4.1.6 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:

The api allows the application to set an entry in the binding table.

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.
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:

On Success : ZigBee_Success.

On Failure : ZigBee_Invalid_Argument/ ZigBee_Failure.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.4.1.7 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);
```

Description:

The 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.
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:

On Success : ZigBee_Success.

On Failure : ZigBee_Invalid_Argument/ ZigBee_Failure.

Returns a negative value if command issued in wrong state and packet allocation failure.

Returns -3, if command issued in wrong state

Returns -4, if packet allocation fails

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

8.5 Callbacks

8.5.1.1 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 used to register 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
zigb_app_child_join_handler	rsi_zigb_app_child_join_handler_t zigb_app_child_join_handler	Child join Callback

Return Values:

None

8.5.1.2 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	The channel on which the scan is enabled.
status	uint8_t	Mac status obtained would be one of the specified status in below table 5

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_Failur	0xE1
g_MAC_Denied_c	0xE2
g_MAC_Disable_TRX_Failure_c	0xE3
g_MAC_Failed_Security_Check_	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
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_	0xF1
g_MAC_TX_Active_c	0xF2
g_MAC_Unavailable_Key_c	0xF3
g_MAC_Unsupported_Attribute	0xF4
g_MAC_Missing_Address_c	0xF5
g_MAC_Past_Time_c	0xF6

Table 7: ZigBee MAC Status

8.5.1.3 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	The channel on which the scan is enabled.
pEnergyValue	uint8_t*	*maxRSSIValue it's a pointer to an array of energy values in 16 channels.

8.5.1.4 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	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.

8.5.1.5 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	indicates that Network is formed or joined successfully.
ZigBeeNWKIsDown	indicates that NWK formation failed or the device left the

	network.
ZigBeeJoinFailed	indicates that network join failed
ZigBeeCannotJoinAsRouter	indicates that network was unable to start as Router.
ZigBeeChangedNodeID	indicates that PANID is changed after resolving PAN ID conflict.
ZigBeeChangedChannel	indicates that the channel is changed due to frequency agility mechanism
ZigBeeReceivedKeyInClear	indicates the Network Key is received is inclear.
ZigBeeNoNWKKeyReceived	indicates no Network key is received.
ZigBeeNoLinkKeyReceived	indicates no Link key is received.
ZigBeePreconfiguredKeyRequired	indicates Preconfigured link key is required.
ZigBeeChangedManagerAddress	indicates network manager changed.

8.5.1.6 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.

8.5.1.7 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
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 6.

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_Complete	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_Too_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
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

Table 8: ZigBee Data Confirmation Status

8.5.1.8 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.

8.5.1.9 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
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

9 APPENDIX

9.1 Example Applications

9.1.1 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	`sapis/examples/wlan/access_point/rsi_ap_start.c`

Example	Description	Application path
	server socket.	
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
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 Wifi station simultaneously	sapis/examples/wlan/concurrent_mode/ rsi_concurrent_mode.c
Connected sleep	This example application demonstrates how	sapis/examples/wlan/connected_sleep/rsi_wlan_connected_sleep_app.c
Connection using asynchronous apis app	This example application demonstrates how asynchronous apis are to to connect the device to access point	sapis/examples/wlan/connection_using_asynchronous_apis_app/ rsi_connection_using_asynchronous_apis_app.c
Firmware upgradation	This example application demonstrates how firmware is upgraded to the device remote TCP server socket	sapis/examples/wlan/fwup/rsi_fwup_app.c
multicast	This example application demonstrates how to data is receive from multicast group	sapis/examples/wlan/multicast/rsi_multicast_app.c
Power save	This example application demonstrates how power	sapis/examples/wlan/power_save/rsi_wlan_powersave_pr

Example	Description	Application path
	save feature is implemented in the device	ofile.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
Wifi direct	This example application demonstrates how to create a wifi direct client and send data	sapis/examples/wlan/wifi_direct/rsi_wfd_client.c
Wps station	This example application demonstrates how to connect to a WPS supported AP using push button method	sapis/examples/wlan/wps_station/rsi_wps_station.c

9.1.2 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

9.1.3 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

9.1.4 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/
Simple chat using BT and WLAN access point	This example demonstrates the data exchanges between BT and WLAN applications	sapis/examples/wlan_bt/wlan_ap_bt_bridge_tcpipbypass/

Example	Description	Application path
applications in tcpip bypass mode		
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 /

9.2 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	De authentication 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

Error Codes (in hexadecimal format)	Description
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
0x001E	Unable to start Group Owner negotiation
0x0020	Unable to join
0x0021	Command given in incorrect state
0x0022	command issued in incorrect operating mode
0x0023	Unable to form Access Point
0x0024	Wrong Scan input parameters supplied to “Scan” command
0x0025	Command issued during re-join in progress
0x0026	Wrong parameters the command request
0x0028	PSK length less than 8 bytes or more than 63 bytes
0x0029	Failed to clear or to set the Enterprise Certificate (Set Certificate)
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
0x0030	Send data packet exceeded the limit or length that is mentioned
0x0031	ARP Cache entry not found
0x0032	UART command timeout happened
0x0037	Wrong WPS PIN
0x0038	Wrong WPS PIN length
0x0039	Wrong PMK length
0x003C	Band not supported
0x003E	Error in length of the http command(Excceds number

Error Codes (in hexadecimal format)	Description
	of characters in http command, that 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
0x004F	ZB/BT/BLE packet received and protocol is not enabled.
0x0050	Parameters error
0x0051	4 way handshake failure.
0x00B1	Memory Error: No memory available.
0x00CA	Error in Ap set region command
0X00CB	Error in AP set region command parameters
0x00CC	Region code not supported
0x00D1	SSL Context Create Failed.

Error Codes (in hexadecimal format)	Description
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.
0xBB0A	Invalid SNTP server address
0xBB0B	SNTP client not started
0xBB10	SNTP server not available
0xBB15	SNTP server authentication failed
0xBB21	IP address error
0xBB22	Socket already bound.
0xBB23	Port not available.
0xBB27	Socket is not created
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	Invalid length in command parameters
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

Error Codes (in hexadecimal format)	Description
0xBB87	POP3 session creation failed/ POP3 session got terminated
0xBB9C	DHCPv6 Handshake failure
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	Empty DNS server list, SMTP message reply error
0xBBAC	SMTP data reply error
0xBBAD	SMTP authentication reply error
0xBBAE	SMTP server error reply
0xBBAF	DNS duplicate entry, SMTP transmit error
0xBBB1	SMTP oversize server reply
0xBBB2	SMTP client not initialized
0xBBB3	DNS IPv6 not suported
0xBBC5	Invalid mail index for POP3 mail retrieve command
0xBBD1	SSL Context Create Failed.
0xBBD2	SSL Handshake Failed. Socket will be closed.
0xBBD3	SSL Max sockets reached. Or FTP client is not connected

Error Codes (in hexadecimal format)	Description
0xBBD4	FTP client is not disconnected
0xBBD5	FTP file is not opened
0xBBD6	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
0xBBEB	Authentication Error
0xBBED	Invalid packet length, content length and received data length is mismatching
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
0xFFFE	Sockets not available. The error comes if the Host tries to open more than 10 sockets
0xFFFC	IP configuration failed
0xFFF8	1) Invalid command (e.g. parameters insufficient or invalid in the command). 2) 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

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
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.
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
0xFF70	SSL not supported
0xFF71	Error in AP set region command parameters
0xFF6E	Invalid operating mode

Error Codes (in hexadecimal format)	Description
0xFF6D	Invalid socket configuration parameters
0xFF6B	Parameter maximum allowed value is exceeded
0xFF69	Invalid command in sequence
0xFF42	DNS response timed out
0xFF40	TCP socket close command is issued before getting the response of the previous close command
0xFF41	HTTP socket creation failed
0xFF36	Wait On Host feature not enabled
0xFF33	TCP keep alive timed out
0xFF2D	TCP ACK failed for TCP SYN-ACK
0xFF2C	Memory limit exceeded in a given operating mode
0xFF62	Failure because of remote terminate
0xFF2B	MDNS deinit failed

Table 9: WiSeConnect^{TM} Error Codes

9.3 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
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
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

Table 10: Bluetooth Generic Error codes

9.4 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
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)

Table 11: Bluetooth Generic Error Codes

9.5 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
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

Table 12: Zigbee Pro Stack Error codes

9.6 PUF Error codes

Error Codes (in hexadecimal format)	Description
0xCC2F	PUF Operation is blocked
0xCC31	PUF Activation code invalid
0xCC32	PUF input parameters invalid

Error Codes (in hexadecimal format)	**Description**
0xCC33	PUF in error state
0xCC34	PUF Operation not allowed
0xCC35	PUF operation Failed