RS9113 WiSeConnect™

Bluetooth Low Energy Software Programming Reference Manual

Version 1.7.9

May 2020

About this Document

This document describes the commands to operate the RS9113-WiSeConnect Module Family for Bluetooth. Bluetooth stack is used for Host layers and the commands describe various profiles supported by Bluetooth stack. This document should be used by the developer to write software on Host MCU to control and operate the module.

Table of Contents

1 Overview 10
2 Bluetooth Software Programming 11
2.1 Bluetooth Software Architecture 11
2.1.1 Application 11
2.1.2 Profiles 11
2.1.3 Bluetooth Core 11
2.1.4 OS Abstraction Layer 12
2.2 Bluetooth Command Format 12
2.3 Interfaces 17
2.3.1 UART 17
2.3.1.1 Features 17
2.3.1.2 Default Parameters 17
2.3.2 USB 18
2.3.2.1 Features 18
2.3.3 SPI 18
2.3.3.1 Features 18
2.3.3.2 Communication through SPI 18
2.3.3.3 SPI Settings 18
2.3.3.4 Interrupt 19
2.4 BLE commands 19
2.4.1 Generic commands 19
2.4.1.1 Set Operating Mode 19
2.4.1.2 Set Local name 30
2.4.1.3 Query Local name 31
2.4.1.4 Query RSSI 31
2.4.1.5 Query Local BD Address 32
2.4.2 BLE Core commands 33
2.4.2.1 Advertise Local Device 33
2.4.2.2 Scan 36
2.4.2.3 Connect 38
2.4.2.4 Disconnect 41
2.4.2.5 Query Device State 41
2.4.2.6 Start Encryption 42
2.4.2.7 SMP Pair Request 43
2.4.2.8 SMP Response 43
2.4.2.9 SMP Passkey 44
2.4.2.10 Initialize BLE module 45
2.4.2.11 Deinitialize BLE module 45
2.4.2.12 BT Antenna Select 45
2.4.2.13 BLE Set Advertise Data 46
2.4.2.14 BLE Set Scan Response Data 47
2.4.2.15 BLE Set LE ping timeout 48
2.4.2.16 BLE Get LE ping timeout 49
2.4.2.17 BLE Set Random Address 49
2.4.2.18 BLE Data Encrypt 50
2.4.2.19 Set Antenna Tx power level 51
2.4.2.20 BLE Whitelist 51
2.4.2.21 Read /Blob Read Response 52

2.4.2.22 LE LTK Request Reply 53
2.4.2.23 SMP Reject Response 53
2.4.3 BLE GATT Profile commands 55
2.4.3.1 Query profiles list 55
2.4.3.2 Query Profile 56
2.4.3.3 Query Characteristic Services 57
2.4.3.4 Query Include Services 59
2.4.3.5 Read Characteristic Value by UUID 61
2.4.3.6 Query Attribute 62
2.4.3.7 Query Attribute Value 63
2.4.3.8 Query Multiple Attribute Values 64
2.4.3.9 Query Long Attribute Value 65
2.4.3.10 Set Attribute Value 66
2.4.3.11 Set Attribute Value no Ack 67
2.4.3.12 Set Long Attribute Value 67
2.4.3.13 Set Prepare Long Attribute Value 68
2.4.3.14 Execute Long Attribute Value 69
2.4.4 BLE Create New Service Commands 69
2.4.4.1 Add GATT Service Record 69
2.4.4.2 Add Attribute Record 71
2.4.4.3 Set Local Attribute Value 72
2.4.4.4 Get Local Attribute Value 72
2.4.5 BLE Core Events 73
2.4.5.1 Advertise Report Event 73
2.4.5.2 Connection Complete Event 74
2.4.5.3 Disconnected 75
2.4.5.4 SMP Request Event 75
2.4.5.5 SMP Response Event 75
2.4.5.6 SMP Passkey Event 76
2.4.5.7 SMP Passkey Display Event 76
2.4.5.8 SMP Failed Event 77
2.4.5.9 SMP Encrypt Enabled Event 77
2.4.5.10 LE ping payload timeout 78
2.4.5.11 LE MTU Size 78
2.4.5.12 LE LTK Request Event 79
2.4.6 BLE GATT Events 79
2.4.6.1 GATT Notification 79
2.4.6.2 GATT Indication 80
2.4.6.3 GATT Write 80
2.4.6.4 GATT READ /Blob Read Request 81
2.5 Bluetooth Generic Error Codes 81
2.6 BLE Mode Error Codes 83
2.7 Power Save 85
2.7.1 Power Save Operations 85
2.7.1.1 Power Save Mode 0 85
2.7.1.2 Power Save Mode 2 (GPIO based mode) 85
2.7.1.3 Power Save Mode 3 (Message based mode) 85
2.7.1.4 Power Save Mode 8 (GPIO based mode) 86
2.7.1.5 Power Save Mode 9 (Message based mode) 86
3 Bluetooth API Library 87
3.1 API File Organization 87
3.2 API Prototypes 87

3.2.1 Generic Prototypes 87
3.2.1.1 Set Local name 87
3.2.1.2 Query Local name 87
3.2.1.3 Query RSSI 87
3.2.1.4 Query Local BD Address 87
3.2.2 BLE Core Prototypes 87
3.2.2.1 Advertise Local Device 87
3.2.2.2 Scan 87
3.2.2.3 Connect 87
3.2.2.4 Disconnect 87
3.2.2.5 Query Device State 88
3.2.2.6 Start Encryption 88
3.2.2.7 SMP Pair Request 88
3.2.2.8 SMP Response 88
3.2.2.9 SMP Passkey 88
3.2.2.10 BLE Init 88
3.2.2.11 BLE Deinit 88
3.2.2.12 BT Antenna Select 88
3.2.2.13 Set Advertise Data Payload 88
3.2.2.14 Set LE Ping Timeout 88
3.2.2.15 Get LE Ping Timeout 88
3.2.2.16 Set Random Address 88
3.2.2.17 LE Data Encrypt 88
3.2.2.18 SMP Pairing Rejection 88
3.2.3 BLE GATT Prototypes 89
3.2.3.1 Query profiles list 89
3.2.3.2 Query Profile 89
3.2.3.3 Query Characteristic Services 89
3.2.3.4 Query Include Services 89
3.2.3.5 Read Characteristic Value by UUID 89
3.2.3.6 Query Attribute 89
3.2.3.7 Query Attribute Value 89
3.2.3.8 Query Multiple Attribute Values 89
3.2.3.9 Query Long Attribute Value 89
3.2.3.10 Set Attribute Value 89
3.2.3.11 Set Attribute Value no Ack 89
3.2.3.12 Set Long Attribute Value 89
3.2.3.13 Set Prepare Long Attribute Value 90
3.2.3.14 Execute Long Attribute Value 90
3.2.3.15 Add GATT Service Record 90
3.2.3.16 Add Attribute Record 90
3.2.3.17 Set Local Attribute Record value 90
3.2.3.18 Get Local Attribute Record value 90
3.2.3.19 BLE Whitelist 90
3.3 Application 90
4 Appendix A: Sample flow 93
4.1 Configure BLE device in Central Mode 93
4.2 Configure BLE device in Peripheral Mode 94
4.3 Configure BLE device in Central Mode to connect to multiple slaves 94
4.4 Configure BLE device to act as both Central and Peripheral simultaneously(Dual Role) 95

4.5 AT command flow to configure BLE device in Peripheral Mode 96
4.6 AT command flow to configure BLE device in Central Mode 97
4.7 Security Management Protocol(SMP) in Slave mode.....98
4.8 Security Management Protocol(SMP) in Master mode.....98
5 Appendix B: Sample flow of APIs for WiFi+BT LE Co-ex mode.....99

Table of Figures

Figure 1 Bluetooth Software Architecture 11
Figure 2 Command frame format 12
Figure 3: Host Interface Block Diagram 17
Figure 4 Sample command sequence of BLE Central Mode 93
Figure 5 Sample command sequence of BLE Peripheral Mode 94
Figure 7 Sample command sequence of BLE dual role 95
Figure 8 Sample AT command sequence of BLE Peripheral Mode 96
Figure 9 Sample AT command sequence of BLE Central Mode 97
Figure 10 Sample flow for WiFi + BT LE response 100

Table of Tables

Table 1 Frame Descriptor **13**
Table 2 Command IDs in BLE mode **15**
Table 3 Response IDs in BLE mode **16**
Table 4 Event IDs in BLE mode **16**
Table 5: Coex Modes Supported **21**
Table 6 Bluetooth Generic Error Codes **83**
Table 7 BLE Error Codes **85**

1 Overview

This document describes the commands to operate RS9113-WiSeConnect Module Family in Bluetooth. The parameters in the commands and their valid values with the expected responses from the modules are also described. The document should be used by the developer to write software on the Host MCU to control and operate the module.

Commands are different for both Classic (BR/EDR) and LE modes.

This section describes commands to operate the module in LE mode. Module can be operated either in Classic mode or in LE mode at a time.

2 Bluetooth Software Programming

The following sections describe Bluetooth software architecture and commands to operate and configure the RS9113 modules in Bluetooth.

2.1 Bluetooth Software Architecture

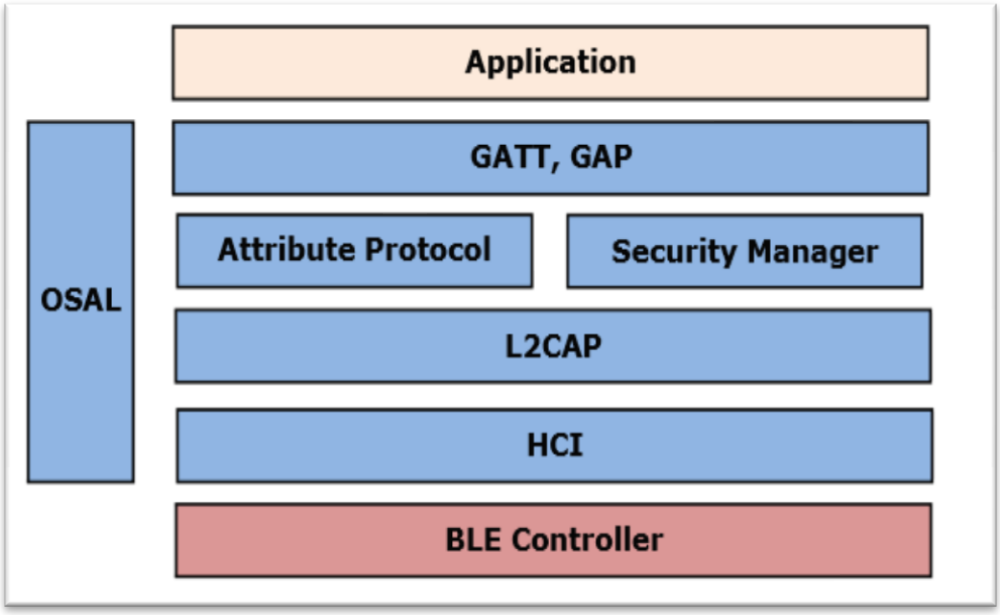


Figure 1 Bluetooth Software Architecture

2.1.1 Application

The application layer launches the Bluetooth stack and uses the commands to access various profiles on the remote Bluetooth devices over the network.

2.1.2 Profiles

There are number of Bluetooth profiles defined in the Bluetooth specification. We currently supports profiles including Generic Attribute Profile (GATT) and Generic Access Profile (GAP). We provide framework to develop new profiles very easily. We will continue to add new profiles.

2.1.3 Bluetooth Core

The Bluetooth core contains the following higher layers of the stack.

SMP

ATT

L2CAP

BLE Controller

SMP is a security management protocol which provides

Services like pairing and key distribution.

ATT (Attribute Protocol) provides a server to expose attribute values.

L2CAP (Logical Link Control and Adaption Protocol) provides

Connection-oriented and connectionless data services to upper layer

Protocols with data packet size upto 64KB in length. L2CAP performs

The segmentation and reassemble of I/O packets from the baseband

Controller.

BLE Controller which includes link controller layers.

2.1.4 OS Abstraction Layer

This layer abstracts RTOS services (semaphores, mutexes and critical sections) that are used by the whole stack and the applications. The stack, which is designed in an RTOS-independent manner, can be used with any RTOS by porting this layer. It is also possible to use the Bluetooth stack standalone without RTOS.

2.2 Bluetooth Command Format

This section explains the general command format. The commands should be sent to the Module in the specified format. The format is same for both Classic and LE modes.

The commands are sent to the module and the responses are read from the module using frame write/frame read (as mentioned in the preceeding sections). These commands are called as command frames.

The format of the command frames are divided into two parts:

1. Frame descriptor
2. Frame Body(Frame body is often called as Payload)

Frame Descriptor (16 bytes)	Frame Body (multiples of 4 bytes)

Command frame format is shown below. This description is for a Little Endian System.

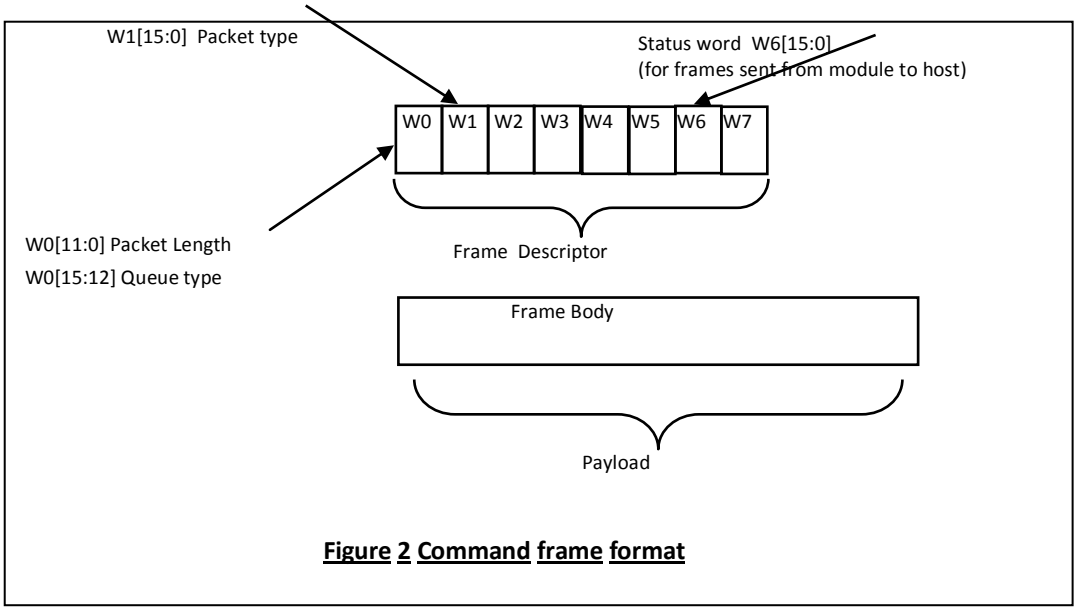


Figure 2 Command frame format

The following table provides the general description of the frame descriptor.

Word	Frame Descriptor
Word0 W0[15:0]	Bits [11:0] – Length of the frame Bits [15:12] – 2(indicates Bluetooth packet).
Word1 W1[15:0]	Bits [15:0] – Packet type
Word2 W2[15:0]	Reserved
Word3 W3[15:0]	Reserved
Word4 W4[15:0]	Reserved
Word5 W5 [15:0]	Reserved
Word6 W6 [15:0]	1. (0x0000) when sent from host to module. 2. When sent from module to host (as response frame), it contains the status.
Word7 W7 [15:0]	Reserved

Table 1 Frame Descriptor

Three types of frames will get exchanged between the module and the host.

1. Request/Command frames - These are sent from Host to Module. Each Request/ Command has an asscociated response with it.
2. Response frames – These are sent from Module to Host. These are given in response to the previous Request/Command from the Host. Each command has a single reponse.
3. Event frames – These are sent from Module to Host. These are given when

- There are multiple reponses for a particular Request/ Command frame
- There is Asynchonous message to be sent to host.

The following are the types of frame requests and responses and the corresponding codes. The commands are different for both Classic and LE modes. The below table lists the Command, Response and Event frames in LE mode.

In both the modes, the corresponding code is to be filled in W1 [15:0] mentioned in the table above.

Command	Command ID
Set Local Name	0x0001
Query Local Name	0x0002
Get RSSI	0x0005
Get Local BD Address	0x0007
Advertise	0x0075
Scan	0x0076
Connect	0x0077
Disconnect	0x0078
Query Device State	0x0079
Start Encryption	0x007B
SMP Pair Request	0x007C
SMP Response	0x007D
SMP Passkey	0x007E
Query Profiles list	0x007F
Query Profile	0x0080
Query Characteristic Services	0x0081
Query Include Services	0x0082
Read Characteristic Value by UUID	0x0083
Query Attribute Descriptor	0x0084
Query Attribute Value	0x0085
Query Multiple Attribute Values	0x0086
Query Long Attribute Values	0x0087
Set Attribute Value	0x0088
Set Attribute Value No Ack	0x0089
Set Long Attribute Value	0x008A
Set Prepare Long Attribute Value	0x008B
Execute Long Attribute Value Write	0x008C
Initialize BLE module	0x008D
Deinitialize BLE module	0x008E
Antenna Select	0x008F
Add New Service	0x0092
Add New Attribute	0x0093
Set local attribute value	0x0094
Get local attribute value	0x0095
Set advertise data	0x009C
Get LE ping timeout	0x00A1

Set LE ping timeout	0x00A2
Set Random Address	0x00A3
Data Encrypt	0x00A4
Set antenna Tx power level	0x00A7
Set scan response data	0x00A8
Le LTK request reply	0x00BA
LE SMP Reject Response	0x00F0

Table 2 Command IDs in BLE mode

Response	Response ID
Card Ready	0x0505
Set Local Name	0x0001
Query Local Name	0x0002
Get RSSI	0x0005
Get Local BD Address	0x0007
Advertise	0x0075
Scan	0x0076
Connect	0x0077
Disconnect	0x0078
Query Device State	0x0079
Start Encryption	0x007B
SMP Pair Request	0x007C
SMP Response	0x007D
SMP Passkey	0x007E
Query Profiles list	0x007F
Query Profile	0x0080
Query Characteristic Services	0x0081
Query Include Services	0x0082
Read Characteristic Value by UUID	0x0083
Query Attribute Descriptor	0x0084
Query Attribute Value	0x0085
Query Multiple Attribute Values	0x0086
Query Long Attribute Values	0x0087
Set Attribute Value	0x0088
Set Attribute Value No Ack	0x0089
Set Long Attribute Value	0x008A
Set Prepare Long Attribute Value	0x008B

Execute Long Attribute Value Write	0x008C
Initialize BLE module	0x008D
Deinitialize BLE module	0x008E
Antenna Select	0x008F
Add New Service	0x0092
Add New Attribute	0x0093
Set local attibute value	0x0094
Get local attribute value	0x0095
Set advertise data	0x009C
Get le ping timeout	0x00A1
Set le ping timeout	0x00A2
Set Random Address	0x00A3
Data Encrypt	0x00A4
Set antenna Tx power level	0x00A7
Set scan response data	0x00A8
Le LTK request reply	0x00BA
LE SMP Reject Response	0x00F0

Table 3 Response IDs in BLE mode

Event	Event ID
Disconnected	0x1006
Scan Response	0x150E
Connection Status	0x150F
SMP Request	0x1510
SMP Response	0x1511
SMP Passkey	0x1512
SMP Failed	0x1513
GATT Notification	0x1514
GATT Indication	0x1515
Encrypt Status	0x1516
GATT Write	0x1517
LE ping timeout expired	0x1518
GATT Read	0x151B
MTU size	0x151C
SMP PassKey Display	0x151D
LE LTK request	0x152A

Table 4 Event IDs in BLE mode

2.3 Interfaces

Host can interface with RS9113-WiSeConnect Module using following list of host interfaces to configure and send/receive data.

- UART
- SPI
- USB

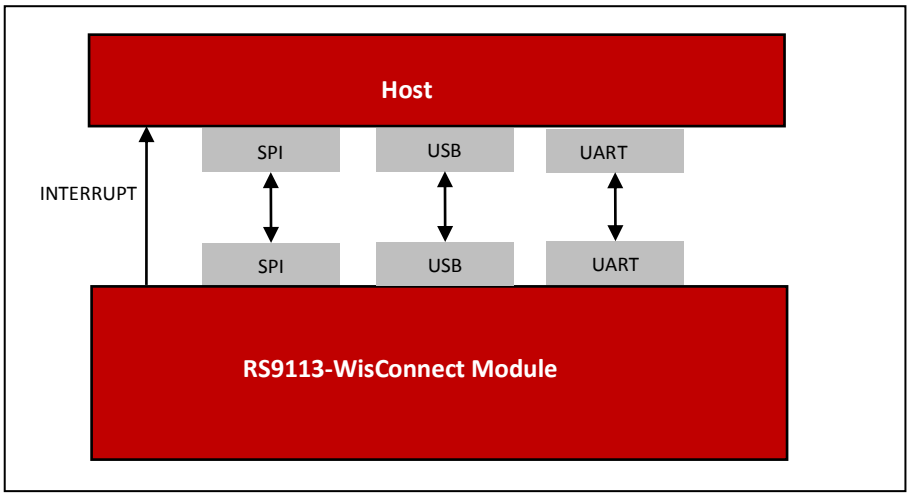


Figure 3: Host Interface Block Diagram

2.3.1 UART

The UART on the RS9113-WiSeConnect module is used as a host interface to configure the module, send and receive data.

2.3.1.1 Features

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

NOTE : For BT/ BLE there is no support for 921600 bps

2.3.1.2 Default Parameters

- Data bits - 8

- Stop bits - 1
- Parity – None
- Flow control - None

2.3.2 USB

RS9113-WiSeConnect module supports USB interface, allow host to configure and send/receive data through module using USB interface.

2.3.2.1 Features

- USB 2.0 (USB-HS core)
  - USB 2.0 offers the user a longer bandwidth with increasing data throughput.
  - USB 2.0 supports additional data rate of 480 Mbits/Sec in addition to 1.5Mbits/Sec and 12 Mbits/Sec.
- Supports USB-CDC

2.3.3 SPI

This section describes RS9113-WiSeConnect module SPI interface and the commands & processes to operate the module using the SPI interface.

2.3.3.1 Features

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

2.3.3.2 Communication through SPI

The RS9113-WiSeConnect module can be configured and operated from the Host by sending commands through the SPI interface.

2.3.3.3 SPI Settings

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

SPI_MOSI (Input) – Serial data input for the module.

SPI_MISO (Output) – Serial data output for the module.

SPI_CS (Input) – Active low slave select signal. This should be low when SPI transactions are to be carried out.

SPI_CLK (Input) – SPI clock. Maximum value allowed is 80 MHz

INTR (Output) – Active high (Default), Active low, level interrupt output from the module.

The module acts as a SPI slave only while the Host is the SPI master.

Following parameters should be in the host SPI interface.

CPOL (clock polarity) = 0,

CPHA (clock phase) = 0.

2.3.3.4 Interrupt

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

1) When the module needs to indicate to the Host that it has received data from the remote terminal and the data needs to be read by the Host.
2) When the module needs to indicate to the Host that a response to a command sent by the Host is ready to be read from the module.
3) To indicate to the Host that it should read a CARD READY message from module. This operation is described in the subsequent sections.

2.4 BLE commands

The following sections will explain various RS9113-WiSeConnect Bluetooth LE commands, their structures, the parameters they take and their responses. For API prototypes of these commands, please refer to the API Library Section

Note:

1. All BT/BLE AT commands are case sensitive and lower case.

2. A command should NOT be issued by the Host before receiving the response of a previously issued command from the module.

NOTE: In the following commands, wherever the BD Address is applicable, it should be given in hex format and upper case characters Please refer to the example commands in each section for more information

2.4.1 Generic commands

2.4.1.1 Set Operating Mode

Description:

This is the first command that needs to be sent from the Host after receiving card ready frame from module. This command configures the module in different functional modes.

Command Format:

AT Mode:

```
at+rsi_opermode=
<oper_mode>,<feature_bit_map>,<tcp_ip_feature_bit_map>,<custom
_feature_bit_map>,<ext_custom_feature_bit_map>,<
bt_custom_feature_bit_map>r\n
```

Binary Mode:

The structure of the payload is give below

```
typedef struct
{
      uint32   oper_mode;
```

```
        uint32    feature_bit_map;
        uint32    tcp_ip_feature_bit_map;
        uint32    custom_feature_bit_map;
        uint32    ext_custom_feature_bit_map;
        uint32    bt_custom_feature_bit_map;
        uint32   ext_tcp_ip_feature_bitmap;
} operModeFrameSnd;
```

Command Parameters:

`Oper_mode`:

Sets the mode of operation. oper_mode contains two parts <wifi_oper_mode, coex_mode>. Lower two bytes represent wifi_oper_mode and higher two bytes represent coex_modes.

`oper_mode = ((wifi_oper_mode) | (coex_mode << 16))`

`Wifi_oper_mode` values:

0 - Wi-Fi Client Mode. The module works as a normal client that can connect to an Access Point with different security modes other than enterprise security.

1 – Wi-Fi Direct™ or Autonomous GO**.** In this mode, the module either acts as a Wi-Fi Direct node or as an Autonomous GO (with intent value 16), depending on the inputs supplied for the command "**Configure Wi-Fi Direct Peer-to-Peer Mode**" in RS9113-WiseConnect-Software-PRM-v1.7.6.pdf at docs folder at release package

. In Autonomous GO and in Wi-Fi Direct GO mode, a maximum of 4 client devices are supported.

2 – Enterprise Security Client Mode. The module works as a client that can connect to an Access Point with WPA/WPA2-Enterprise security.

6 – Access Point mode**.** In this mode, the module acts as an Access Point, depending on the inputs supplied for the command "**Configure AP Mode**" in RS9113-WiseConnect-Software-PRM-v1.7.6.pdf at docs folder at release package

. In Access Point mode, a maximum of 8 client devices are supported.

8 - PER Mode. This mode is used for calculating packet error rate and mostly used during RF certification tests.

9 – Cocurrent mode. This mode is used to run module in concurrent mode. In concurrent mode, host can connect to a AP and can create AP simultaneously.

NOTE: In concurrent mode

1. AP MAC address last byte will differ and it will be one plus the station mode MAC last byte.
2. In TCP/IP non bypass mode, Broadcast/Multicast packet will go to first created interface (e.g. if Station mode connects first the broadcast/multicast packet will go to network belonging to station mode).

3. IPV6 support is not present in the current release.

`coex_mode` bit values: enables respective protocol

BIT 0 : Enable/Disable WLAN mode.

0 – Disable WLAN mode

1 – Enable WLAN mode

BIT 1 : Enable/Disable ZigBee mode.

0 – Disable ZigBee mode

1 – Enable ZigBee mode

BIT 2 : Enable/Disable BT mode.

0 – Disable BT mode

1 – Enable BT mode

BIT 3 : Enable/Disable BTLE mode.

0 – Disable BTLE mode

1 – Enable BTLE mode

NOTE: In BTLE mode, need to enable BT mode also.

Following table represents possible coex modes supported:

Coex_mode	Description
0	WLAN only mode
3	WLAN and ZigBee coexistence mode.
5	WLAN and BT coexistence mode.
13	WLAN and BTLE coexistence mode.
14	BTLE and ZigBee coexistence mode.

Table 5: Coex Modes Supported

NOTE: Following CoeX mode is supported currently 1. WLAN STA+BT (Only TCP/IP Bypass mode) 2. WLAN STA + BLE 3. WLAN STA + ZB 4. BLE + ZB 5. WLAN AP + BT (Only support is present in TCP/IP Bypass mode) 6. WLAN AP + BLE 7. WLAN AP + ZB

To select proper CoeX mode please refer

WiSeConnect_TCPIP_Feature_Selection_v1.7.6.xlsx at docs folder given in the release package

NOTE: If coex mode enabled in opermode command, then BT/BLE or ZigBee protocol will start and give corresponding card ready in parallel with opermode command response (which will be handled by corresponding application).

BT card ready frame is described in

RS9113-WiseConnect-BT-Classic-Software-PRM-API-Guide-v1.7.6.pdf,

BLE card ready frame is described in

RS9113-WiseConnect-BLE-Software-PRM-API-Guide-v1.7.6.pdf

and ZigBee card ready frame is described in
RS9113-WiseConnect-ZigBee-Software-PRM-API-Guide-v1.7.6.pdf

at docs folder in release package.

`feature_bit_map`: this bitmap is used to enable following WLAN features:

`feature_bit_map[0]` – To enable open mode

0 - Open Mode Disabled

1- Open Mode enabled (No Security)

`feature_bit_map[1]` – To enable PSK security

0 - PSK security disabled

1 - PSK security enabled

`feature_bit_map[2]` –To enable Aggregation in station mode

0-Aggregation disabled

1-Aggregation enabled

`feature_bit_map[3]` –To enable LP GPIO hand shake

0 – LP GPIO hand shake disabled

1 – LP GPIO hand shake enabled

`feature_bit_map[4]` –To enable ULP GPIO hand shake

0 – ULP GPIO hand shake disabled

1 – ULP GPIO hand shake enabled

`feature_bit_map[5]` –To select module to host wakeup pin

0 – GPIO_21 is used as module to host wakeup pin

1 – ULP_GPIO_1 is used as module to host wakeup pin

`feature_bit_map[6]` –To select RF supply voltage

0 – RF voltage is set to 1.9V

1 – RF voltage is set to 3.3V

`feature_bit_map[7]`-To disable WPS support

0 – WPS enable

1 - WPS disable in AP mode and station Mode

`feature_bit_map[8:31]` – Reserved. Should set to be '0'

NOTE: `feature_bit_map[0], feature_bit_map[1]` are valid only in Wi-Fi client mode.

`tcp_ip_feature_bit_map`: To enable TCP/IP related features.

`tcp_ip_feature_bit_map[0]` – To enable TCP/IP bypass

0 - TCP/IP bypass mode disabled

1 - TCP/IP bypass mode enabled

`tcp_ip_feature_bit_map[1]` – To enable http server

0 - HTTP server disabled

1 - HTTP server enabled

`tcp_ip_feature_bit_map[2]` – To enable DHCPv4 client

0 - DHCPv4 client disabled

1 - DHCPv4 client enabled

`tcp_ip_feature_bit_map[3]` – To enable DHCPv6 client

0 - DHCPv6 client disabled

1 - DHCPv6 client enabled

`tcp_ip_feature_bit_map[4]` – To enable DHCPv4 server

0 - DHCPv4 server disabled

1 - DHCPv4 server enabled

`tcp_ip_feature_bit_map[5]` – To enable DHCPv6 server

0 - DHCPv6 server disabled

1 - DHCPv6 server enabled

`tcp_ip_feature_bit_map[6]` – To enable Dynamic update of web pages (JSON objects)

0 - JSON objects disabled

1 - JSON objects enabled

`tcp_ip_feature_bit_map[7]` – To enable HTTP client

0 - To disable HTTP client

1 - To enable HTTP client

`tcp_ip_feature_bit_map[8]` – To enable DNS client

0 - To disable DNS client

1 - To enable DNS client

`tcp_ip_feature_bit_map[9]`– To enable SNMP agent

0 - To disable SNMP agent

1 - To enable SNMP agent

`tcp_ip_feature_bit_map[10]`– To enable SSL

0 - To disable SSL

1 - To enable SSL

`tcp_ip_feature_bit_map[11]`– To enable PING from module(ICMP)

0 - To disable ICMP

1 - To enable ICMP

`tcp_ip_feature_bit_map[12]`– To enable HTTPS Server

0 - To disable HTTPS Server

1 - To enable HTTPS Server

`tcp_ip_feature_bit_map[14]`– To send configuration details to host on submitting configurations on wireless configuration page

0 - Do not send configuration details to host

1 - Send configuration details to host

`tcp_ip_feature_bit_map[15]`– To enable FTP client

0 - To disable FTP client

1 - To enable FTP client

`tcp_ip_feature_bit_map[16]`– To enable SNTP client

0 - To disable SNTP client

1 - To enable SNTP client

`tcp_ip_feature_bit_map[17]`– To enable IPv6 mode

0 - To disable IPv6 mode

1 - To enable IPv6 mode

IPv6 will also get enabled if DHCP v6 client/DHCP v6 server is enabled irrespective of `tcp_ip_feature_bit_map[17]`.

`tcp_ip_feature_bit_map[19]`– To MDNS and DNS-SD

0 - To disable MDNS and DNS-SD

1 - To Enable MDNS and DNS-SD

`tcp_ip_feature_bit_map[20]`– To enable SMTP client

0 - To disable SMTP client

1 - To Enable SMTP client

`tcp_ip_feature_bit_map[21 – 24]`– To select no of sockets

possible values are 1 to 10 . If User tried to select more than 10 sockets it will be reset to 10 sockets only . Default no of sockets is 10, if this selection is not done by the user.

`tcp_ip_feature_bit_map[25]`- To select Single SSL socket

0 – selecting single socket is Disabled

1- Selecting single socket is enabled

NOTE: By default two SSL sockets are supported

`tcp_ip_feature_bit_map[26]`- To allow loading Private & Public certificates

0 – Disable loading private & public certificates

1- Allow loading private & public certificates

NOTE : If Secure handshake is with CA – certificate alone , then disable loading Private and public keys and erase these certificates from the flash using load_cert API .

Or if Secure handshake needed verification of Private and Public keys , then enable loading of private and public keys.

`tcp_ip_feature_bit_map[27]`- To load SSL certificate on to the RAM

`tcp_ip_feature_bit_map[28]`- To enable TCP-IP data packet Dump on UART2

`tcp_ip_feature_bit_map[29]`- To enable POP3 client

0 - To disable POP3 client

1 - To Enable POP3 client

`tcp_ip_feature_bit_map[13]`, `tcp_ip_feature_bit_map[18]`, `tcp_ip_feature_bit_map[30:31]`-All set to '0'.

NOTE: SSL(`tcp_ip_feature_bit_map[10]`, `tcp_ip_feature_bit_map[12]`) is supported only in opermode 0

NOTE: **Feature selection utility** is provided in the package. WiSeConnect device supports the selected features combination only if it is feasible according to the WiSeConnect TCPIP Feature Selection v1.7.6.xlsx

at docs folder given in the release package.

`custom_feature_bit_map`:

This bitmap used to enable following custom features:

`BIT[2]`: If this bit is set to '1', the DHCP server behavior, when the module is in AP mode, changes. The DHCP server, when it assigns IP addresses to the client nodes, does not send

out a Gateway address, and sends only the assigned IP and Subnet values to the client. It is highly recommended to keep this value at '0' as the changed behavior is required in only very specialised use cases and not in normal AP functionality. The default value of this bit is '0'.

BIT[5]: If this bit is set to '1', Hidden SSID is enabled in case of AP mode. The default value of this bit is '0'.

BIT[6]: To enable/disable DNS server IP address in DHCP offer response in AP mode.

1- In AP mode, DHCP server sends DNS server IP address in DHCP offer

0- Not to include DNS server address in DHCP offer response

BIT[8]: - Enable/Disable DFS channel passive scan support

1- Enable

0-Disable

BIT[9]: – To Enable/disable LED(GPIO_16) after module initialization(INIT).

1- Enable LED support

0– Disable LED support

BIT[10]: Used to enable/disable Asynchronous messages to host to indicate the module state.

1- Enable asynchronous message to host

0-Disable asynchronous message to host

BIT[11] : To enable/disable packet pending (Wakeon wireless) indication in UART mode

1 – Enable packet pending indication

0- Disable packet pending indication

BIT[12]: Used to enable or disable AP blacklist feature in client mode during roaming or rejoin. By default module maintains AP blacklist internally to avoid some access points.

1 – Disable AP black list feature

0 – Enable AP black list feature

BIT[13-16]: Used to set the maximum number of stations or client to support in AP or Wi-Fi Direct mode. Possible values are 1 to 8 in AP mode and 1 to 4 in Wi-Fi Direct mode.

Note1: If these bits are not set, default maximum clients supported is set to 4.

BIT[17] : to select between de-authentication or Null data (with power management bit set) based roaming, Depending on selected method station will send deauth or Null data to connected AP when roam from connected AP to newly selected AP.

0 – To enable de-authentication based roaming

1 – To enable Null data based roaming

BIT[18]: Reserved

BIT[19]: Reserved

BIT[20]: Used to start/stop auto connection process on bootup, until host triggers it using Trigger Auto Configuration command

1 – Enable

0 – Disable

BIT[22]: Used to enable per station power save packet buffer limit in AP mode. When enabled, only two packets per station will be buffered when station is in power save

1 – Enable

0 – Disable

BIT[23] : To enable/disable HTTP/HTTPs authentication

1 - Enable

0 – Disable

BIT[24]: To enable/disable higher clock frequency in module to improve throughputs

1 - Enable

0 – Disable

BIT[25]: To give HTTP server credentials to host in get configuration command

1 – To include HTTP server credentials in get configuration command response

0 – To exclude HTTP server credentials in get configuration command response

BIT[26]: To accept or reject new connection request when maximum clients are connected in case of LTCP.

1 - Reject

0 – Accept

By default this bit value is zero.

When BIT[26] is zero: For a LTCP socket when maximum clients are connected if a new connection request is received, then this connection request will not be rejected. Instead module will maintain this connection request in LTCP pending list.

This request will be served when any of the connected client is disconnected.

When BIT[26] is set: For a LTCP socket when maximum clients are connected if a new connection request is received, then this connection request will be rejected immediately. Module will not maintain this connection request in LTCP pending list.

BIT[27]: To enable dual band roaming and rejoin feature this bit is used.

1 - Enable dual band roaming and rejoin

0 – Disable dual band roaming and rejoin.

BIT[28]: To enable real time clock from host

1 - Enable real time clock feature given by host

0 – Disable real time clock feature

BIT[29]: To Enable IAP support in BT mode

1 - Enable

0 – Disable

BIT[31]: This bit is used to validate extended custom feature bitmap.

1 – Extended feature bitmap valid

0 – Extended feature bitmap is invalid

`BIT[0:1],BIT[3:4],BIT[7],BIT[21], BIT[30]`: Reserved, should be set to all '0'.

NOTE: For UART/USB-CDC in AT mode:

When user does not give any `tcp_ip_feature_bit_map` value then default settings for client mode, Enterprise client mode, WiFi-Direct mode are:

HTTP server, DHCPv4 client, DHCPv6 client and JSON objects are enabled.

When user does not give any `tcp_ip_feature_bit_map` value then default settings for Access point mode are:

HTTP server, DHCPv4 server, DHCPv6 server and JSON objects are enabled.

Parameters- `feature_bit_map, tcp_ip_feature_bit_map` and `custom_feature_bit_map` are optional in opermode command in UART mode for AT mode. If user does not give these parameters then default configuration gets selected, as explained above, based upon the operating mode configured.

If opermode is 8 (PER mode is selected) - `feature_bit_map, tcp_ip_feature_bit_map` and `custom_feature_bit_map` can be ignored or not valid. Set to zero.

`ext_custom_feature_bit_map:`

This feature bitmap is extention of custom feature bitmap and is valid only if BIT[31] of custom feature bitmap is set. This enables the following feature.

`BIT[0]`: To enable antenna diversity feaute.

1 – Enable antenna diversity feature

0 – Disble antenna diversity feature

`BIT[1]`:This `bit is used to enable 4096 bit RSA key support`

`1 – Enable 4096 bit RSA key support`

`0 – Disable 4096 bit RSA key support`

Note: This bit is required to set for 4096 bit RSA key support. If key size is 4096 bit, module will use software rountine for exponentiation, so connection time will increase.

`BIT[2]`:This bit is used to set the module type.This is applicable only if manufacturing software version of the module is below 3.1 (i.e. manufacturing version 3 and subversion 1).

```
0 - Module will ignore the module type given
through the set region command.
```

```
1 - Module will accept the module type given
through the set region command.
```

`bt_custom_feature_bit_map:`

To configure the number of GATT Records.

This bitmap is valid only if BIT[31] of extended custom feature bit map is set.

This bitmap is used to set number of multiple slaves supported, number of services and number of attributes.

bt_custom_feature_bit_map [0:7] – number of attributes

bt_custom_feature_bit_map [8:11] – number of services

bt_custom_feature_bit_map [12:15] – reserved for future use

bt_custom_feature_bit_map [16:23] – number of slaves supported by LE

bt_custom_feature_bit_map [24:31] – reserved

NOTE:

If bit bt_custom_feature_bit_map is set:

1. User can enter maximum of 8 BLE slaves.
2. Maximum of 10 services in total can exist out of which two services namely GAP and GATT are added by default. So if this bitmap has value 10 user can add upto 8 services.
3. Maximum of 80 attributes in total can exist out of which ten attributes of GAP and GATT are added by default. So if this bitmap has value 80 user can add upto 70 attributes.

If bit bt_custom_feature_bit_map is not set:

4. Default number of BLE slaves supported is 1.
5. Maximum of 5 services in total can exist out of which two services namely GAP and GATT are added by default. So user can add upto 3 services.
6. Maximum of 20 attributes in total can exist out of which ten attributes of GAP and GATT are added by default. So user can add upto 10 attributes.

If module have to support connection with multiple devices then those many devices count value must be assigned to BLE slaves.

For one service – 48 (In Bytes)

For one attribute – 65 (In Bytes)

For one LE slave info – 300(In Bytes)

If SSL feature is to be used along with BLE, enable `Tcp_ip_feature_bit_map[25]` bit only field otherwise 0xff2c error will be returned.

Response:

AT Mode:

Result Code	Description
OK	Successful execution of the command
ERROR<Error code>	Failure

Binary Mode:

There is no response payload for this command.

Example 1: To enable wlan and ble operating mode.

AT Mode:

```
at+rsi_opermode=851968,0,1,0\r\n
```

Response:

```
OK
bt_loaded\r\n
```

Example 2: To enable 7 ble slaves , 5 services and 25 attributes.

Command in AT Mode:

```
at+rsi_opermode=851968,0,1,2147483648,2147483648,460057\r\n
```

Response:

```
OK\r\n
```

Possible error codes:

```
0x0059,0xff2c
```

2.4.1.2 Set Local name

Description: This is used to set name to the local device.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_local_name {
        UINT08  NameLength;
        INT08   Name[50];
} RSI_BT_CMD_SET_LOCAL_NAME;
```

AT command format:
at+rsibt_setlocalname=<NameLength>,<Name>\r\n

Parameters:

NameLength – Length of the name of the local device.

Name – Name of the local device.

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_setlocalname=8,redpines\r\n

Response: OK\r\n

Note:
1. Name set by this command is used for GAP service's device name characteristic value only.

2. For ios, this local name will be shown after the connection using the GAP services. If we change the local name it will affect after the connection only.
3. Name in BLE mode is limited to 16 bytes only. If even more than 16 characters are set remaining characters after 16th character are truncated. Namelength parameter should be in decimal format.

2.4.1.3 Query Local name

Description: This is used to query the name of the local device.

Binary Payload Structure:
No Payload required.

AT command format:
at+rsibt_getlocalname?\r\n

Response Payload:

```
typedef struct rsi_bt_resp_query_local_name {
   UINT08   NameLength;
   INT08   Name[50];
} RSI_BT_RESP_QUERY_LOCAL_NAME;
```

Result Code	Description
OK <name_length>,<local_device_name>	Command Success.
ERROR <Error_code>	Command Fail.

Response Parameters:

NameLength – Length of the name of the local device.

Name – Name of the local device.

AT command Ex: at+rsibt_getlocalname?\r\n

Response: OK 8,redpines\r\n

Note:

Name returned is limited to 16 bytes only.

This command returns GAP service's device name characteristic value which is previously set by set local name command.

2.4.1.4 Query RSSI

Description: This is used to query RSSI of the connected remote BD device

Payload Structure:

```
typedef union {

struct {
  UINT08 BDAddress[RSI_BT_BD_ADDR_LEN];
}QueryRssiFrameSnd;

UINT08 uQueryRSSIBuf[RSI_BT_BD_ADDR_LEN];

} RSI_BT_CMD_QUERY_RSSI;
```

AT command format:

at+rsibt_getrssi=<BDAddress>?\r\n

Parameters:

BDAddress – BT Address of the connected remote device.

Response Payload:

```
typedef struct rsi_bt_resp_query_rssi {
   UINT08 RSSI;
} RSI_BT_RESP_QUERY_RSSI;
```

Result Code	Description
OK <rssi value>	Command Success.
ERROR <Error_code>	Command Fail.

Response parameters:

RSSI – RSSI value of the connected remote device.

AT command Ex: at+rsibt_getrssi=AA-BB-CC-DD-EE-FF?\r\n **Response**: OK 130\r\n

2.4.1.5 Query Local BD Address

Description: This is used to query the BD address of the local device

Payload Structure:

No Payload required.

AT command format:

at+rsibt_getlocalbdaddr?\r\n

Response Payload:

```
typedef struct rsi_bt_resp_query_local_bd_address {
   UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
} RSI_BT_RESP_QUERY_LOCAL_BD_ADDRESS;
```

Result Code	Description
OK <bd_addr>	Command Success with valid response.
ERROR <Error_code>	Command Fail.

Response Parameters:

BDAddress - BT Address of the local device

AT command Ex: at+rsibt_getlocalbdaddr?\r\n

Response: OK AA-BB-CC-DD-EE-FF\r\n

2.4.2 BLE Core commands

2.4.2.1 Advertise Local Device

Description: This is used to expose or advertise about local device to the remote BT devices.

Payload Structure:

```
typedef union {
   struct {
   UINT08  Status;
   UINT08  AdvertiseType;
   UINT08  FilterType;
   UINT08  DirectAddrType;
   UINT08  DirectAddr[RSI_BT_BD_ADDR_LEN];
   UINT16  adv_int_min;
   UINT16  adv_int_max;
   UINT08  own_add_type;
   UINT08  adv_channel_map;
  }AdvFrameSnd  ;
  UINT08 uAdvBuf[RSI_BT_BD_ADDR_LEN + 10];
} RSI_BLE_CMD_ADVERTISE  ;
```

AT Command format:

at+rsibt_advertise=< Status >,< AdvertiseType >,< FilterType >,<DirectAddrType>,<DirectAddr>,< adv_int_min >,< adv_int_max >,< own_add_type >,< adv_channel_map >\r\n

NOTE: All parameters should be in decimal except DirectAddr, it should be in hexadecimal.

Parameters:

Status – To enable/disable Advertising.

1 – Enable Advertising

0 – Disable Advertising

AdvertiseType –

State	Description
0x80	Connectable undirected
0x81	Connectable directed with high duty cycle
0x82	Scannable undirected
0x83	Non connectable undirected
0x84	Connectable directed with low duty cycle

FilterType –

Filter type	Description
0	Allow Scan Request from Any, Allow Connect Request from Any.
1	Allow Scan Request from White List Only, Allow Connect Request from Any.
2	Allow Scan Request from Any, Allow Connect Request from White List Only.
3	Allow Scan Request from White List Only, Allow Connect Request from White List Only.

DirectAddrType – 0 – Public address

1 – Random address

DirectAddr – Remote device BD Address

adv_int_min-

Value	Parameter Description

Value	Parameter Description
N = 0xXXXX	Minimum advertising interval for non-directed advertising. Range: 0x0020 to 0x4000 Default: N = 0x0800 (1.28 second) Time = N * 0.625 msec Time Range: 20 ms to 10.24 sec.

adv_int_max-

Value	Parameter Description
N = 0xXXXX	Minimum advertising interval for non-directed advertising. Range: 0x0020 to 0x4000 Default: N = 0x0800 (1.28 second) Time = N * 0.625 msec Time Range: 20 ms to 10.24 sec.

own_add_type-

Value	Parameter Description
0x00	Public Device Address (default)
0x01	Random Device Address
0x02 – 0xFF	Reserved for future use

adv_channel_map-

Value	Parameter Description
00000000b	Reserved for future use
xxxxxxx1b	Enable channel 37 use
xxxxxx1xb	Enable channel 38 use
xxxxx1xxb	Enable channel 39 use
00000111b	Default (all channels enabled)

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_advertise=1,128,0,0,0,20,30,0,7\r\n(enable)

Response: OK\r\n

NOTE: For scannable undirected and non-connectable undirected advertising modes, minimum advertising interval should be 41ms and maximum advertising interval should be 1.28s

2.4.2.2 Scan

Description: This is used to scan for remote LE advertise devices.
Payload Structure:

```
typedef union {
   struct {
   UINT08  Status;
   UINT08  Scantype;
   UINT08  FilterType;
    UINT08  own_add_type;
   UINT16  scan_int;
   UINT16  scan_win;
  }ScanFrameSnd  ;
  UINT08 uScanBuf[8];
} RSI_BLE_CMD_SCAN  ;
```

AT Command format:

at+rsibt_scan=< Status >, < Scantype >, < FilterType >,< own_add_type >,< scan_int >,< scan_win >\r\n

Note: All Parameters should be in Decimal.

Parameters:

Status – To enable/disable Scanning

1 – Enable Scanning

0 – Disable Scanning

Scantype –

Scan type	Description
0	Passive Scanning
1	Active Scanning

FilterType –

Value	Parameter Description
0	Accept all advertisement packets.
1	Accept only white listed device advertisement packets.

Scan_int-

value	Description
N = 0xXXXX	This is defined as the time interval from when the Controller started its last LE scan until it begins the subsequent LE scan. Range: 0x0004 to 0x4000 Default: 0x0010 (10 ms) Time = N * 0.625 msec Time Range: 2.5 msec to 10 . 24 seconds

scan_win-

value	Description
N = 0xXXXX	The duration of the LE scan. LE_Scan_Window shall be less than or equal to LE_Scan_Interval Range: 0x0004 to 0x4000 Default: 0x0010 (10 ms) Time = N * 0.625 msec Time Range: 2.5 msec to 10240 msec

own_add_type-

Value	Parameter Description
0x00	Public Device Address (default)
0x01	Random Device Address
0x02 – 0xFF	Reserved for future use

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_scan=1,0,0,0,100,10\r\n (enable scan)

Response: OK\r\n

AT command Ex: at+rsibt_scan=0,0,0,0,100,10\r\n (disable scan)

Response: OK\r\n

Note: In parameters Scan Interval must be greater than scan window.

2.4.2.3 Connect

Description: This is used to create connection with remote LE device.

Payload Structure:

```
typedef union {
   struct {
   UINT08  AddressType;
   UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
   UINT08  Reserved;
   UINT16  LeScanInterval;
   UINT16  LeScanWindow;
   UINT16  ConnIntervalMin;
   UINT16  ConnIntervalMax;
   UINT16  ConnLatency;
   UINT16  SupervisionTimeout;
  }ConnectFrameSnd  ;
  UINT08 uConnectBuf[RSI_BT_BD_ADDR_LEN + 14];
} RSI_BLE_CMD_CONNECT  ;
```

AT Command format:

at+rsibt_connect=< AddressType >, < BDAddress >< LeScanInterval >,< LeScanWindow >,< ConnIntervalMin >,< ConnIntervalMax >,< ConnLatency >,< SupervisionTimeout >\r\n

Parameters:

AddressType – Specifies the type of the address mentioned in BDAdress

0 – Public Address

1 – Random Address

BDAdress – BD Address of the remote device

le_scan_interval and le_scan_window parameters are recommendations from the Host on how long (LE_Scan_Window) and how frequently (LE_Scan_Interval) the Controller should scan.

LE_Scan_Interval:

value	Description
N = 0xXXXX	This is defined as the time interval from when the Controller started its last LE scan until it begins the subsequent LE scan. Range: 0x0004 to 0x4000 Time = N * 0.625 msec Time Range: 2.5 msec to 10 . 24 seconds

LE_Scan_Window:

value	Description
N = 0xXXXX	Amount of time for the duration of the LE scan. LE_Scan_Window shall be less than or equal to LE_Scan_Interval Range: 0x0004 to 0x4000 Time = N * 0.625 msec Time Range: 2.5 msec to 10 . 24 seconds

The conn_interval_min and conn_interval_max parameters define the minimum and maximum allowed connection interval.

Conn_Interval_Min:

value	Description
N = 0xXXXX	Minimum value for the connection event interval. This shall be less than or equal to Conn_Interval_Max. Range: 0x0006 to 0x0C80 Time = N * 1.25 msec Time Range: 7.5 msec to 4 seconds.
0x0000 – 0x0005 and 0x0C81 – 0xFFFF	Reserved for future use

Conn_Interval_Max:

value	Description
N = 0xXXXX	Minimum value for the connection event interval. This shall be greater than or equal to Conn_Interval_Min. Range: 0x0006 to 0x0C80 Time = N * 1.25 msec Time Range: 7.5 msec to 4 seconds.

value	Description
0x0000 – 0x0005 and 0x0C81 – 0xFFFF	Reserved for future use

Conn_Latency:

The conn_latency parameter defines the maximum allowed connection Latency.

Value	Description
N = 0xXXXX	Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F4

Supervision_Timeout:

The supervision_tout parameter defines the link supervision timeout for the connection.

Value	Description
N = 0xXXXX	Supervision timeout for the LE Link. Range: 0x000A to 0x0C80 Time = N * 10 msec Time Range: 100 msec to 32 seconds
0x0000 – 0x0009 and 0x0C81 – 0xFFFF	Reserved for future use

Response Payload:

There is no response payload for this command.

AT command Ex:

at+rsibt_connect=0,B4-99-4C-64-BE-F5,96,32,160,160,0,100\r\n

Response: OK\r\n

Note:

Reception of the response doesn't mean that Connection with the remote device is completed.

Connection is said to complete after it receives **Connection Complete Event**. The user is recommended not to give further commands before receiving the above Event. But the user is allowed to give **Disconnect** even before **Connection Complete Event** is received.

2.4.2.4 Disconnect

Description: This is used to cancel create Connection or disconnect HCI Connection, if already connected.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
  }DisconnectFrameSnd;

  UINT08 uDisconnectReqbuf[RSI_BT_BD_ADDR_LEN];
} RSI_BLE_CMD_DISCONNECT;
```

AT Command format*:*
at+rsibt_disconnect=<BDAddress>\r\n

Parameters:

BDAdress – BD Address of the remote device

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_disconnect=53-41-CC-FF-91-2A\r\n

Response: OK\r\n

NOTE: Make sure that BD address is given same as create connection BD address or already connected BD address.

2.4.2.5 Query Device State

Description: This is used to query state of the local device.

Payload Structure: No Payload required.

AT Command format:
at+rsibt_getdevstate?\r\n

Response Payload:

```
typedef struct rsi_ble_resp_query_device_state {
   UINT08  DeviceState;
} RSI_BLE_RESP_QUERY_DEVICE_STATE;
```

Result Code	Description
OK,< state>	Command Success.

Result Code	Description
ERROR <Error_code>	Command Fail.

Response Parameters:

DeviceState –

Bit	Description
0	Advertise(0-disable/1-enable)
1	scan state(0-disable/ 1-enable)
2	connection initiated(0-not initiated/1-initiated)
3	connected state(0-not connected/1-connected)

AT command Ex: at+rsibt_getdevstate?\r\n

Response: OK 8\r\n

2.4.2.6 Start Encryption

Description: This is used to initiate the Encryption procedure.

Payload Structure:

```
typedef union {

  struct {
    UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
  }SmpEncriptionFrameSnd  ;

  UINT08 uSmpEncryptReqbuf[RSI_BT_BD_ADDR_LEN];

} RSI_BLE_CMD_ENCRYPTTION;
```

AT command Format: at+rsibt_startencrypt=<BDAddress>\r\n
Parameters:

BDAddress – Remote BD Address.

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_startencrypt=B4-99-4C-64-BE-F5\r\n

Response: OK\r\n

2.4.2.7 SMP Pair Request

Description: This is used to send SMP Pair Request command to the connected remote device.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
   UINT08  IOCapability;
  }SmpPairReqFrameSnd  ;
  UINT08 uSmpPairReqbuf[RSI_BT_BD_ADDR_LEN + 1];
} RSI_BLE_CMD_SMP_PAIR_REQUEST  ;
```

AT Command format:at+rsibt_smpreq=<BDAddress>,<IO Capability>\r\n

Parameters:

IOCapability –

IO Capability type	Description
0	Display Only
1	Display Yes No
2	Keyboard only
3	No input no output

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_smpreq=B4-99-4C-64-BE-F5,1\r\n

Response: OK\r\n

2.4.2.8 SMP Response

Description: This is used to send SMP response to the SMP request made by the remote device.

Payload Structure:

```
typedef union {
  struct {
    UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
    UINT08  IOCapability;
  }SmpRespFrameSnd  ;
  UINT08 uSmpRespBuf[RSI_BT_BD_ADDR_LEN + 1];
} RSI_BLE_CMD_SMP_RESPONSE  ;
```

AT Command format:at+rsibt_smpresp=<BDAddress>,<IOCapability>\r\n

Parameters:

IOCapability –

IO Capability type	Description
0	Display Only
1	Display Yes No
2	Keyboard only
3	No input no output

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_smpresp=74-04-2B-7A-93-EF,1\r\n

Response: OK\r\n

2.4.2.9 SMP Passkey

Description: This is used to send SMP Passkey required to connect with the remote device.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
   UINT08  Reserved[2];
   UINT08  Passkey[4];
  }SmpPasskeyFrameSnd  ;
  UINT08 uSmpPasskeyBuf[RSI_BT_BD_ADDR_LEN + 6];
} RSI_BLE_CMD_SMP_PASSKEY  ;
```

AT Command format:
at+rsibt_smppasskey=<BDAddress>,<Passkey>\r\n

Parameters:

Passkey – Passkey to authenticate with the Remote device.

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_smppasskey=B4-99-4C-64-BE-F5,12345\r\n

Response: OK\r\n

2.4.2.10 Initialize BLE module

Description: This is used to initialize the BLE module

AT Command format:

at+rsibt_btinit\r\n

Payload Structure:
No Payload required.

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_btinit\r\n

Response: OK\r\n

2.4.2.11 Deinitialize BLE module

Description: This is used to deinitialize the BLE module.To again initialize the module command is used.

AT Command format:

at+rsibt_btdeinit\r\n

Payload Structure:
No Payload required.

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_btdeinit\r\n

Response: OK\r\n

2.4.2.12 BT Antenna Select

Description: This is used to select the internal or external antenna of the BT module.

Payload Structure:

typedef struct rsi_bt_cmd_antenna_select{

UINT08 AntennaVal;

} RSI_BT_CMD_ANTENNA_SELECT;

AT Command format:

at+rsibt_btantennaselect=<AntennaVal>\r\n

Parameters:

AntennaVal – To select the internal or external antenna

0 – Internal Antenna.

1 – External Antenna.

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_btantennaselect=1\r\n

Response: OK\r\n

2.4.2.13 BLE Set Advertise Data

Description: This command is used to set the advertise data to expose remote devices.

Payload Structure:

```
typedef union {
  struct {
UINT08  DataLen;
  UINT08  Data[31];
}SetAdvertiseDataFrameSnd  ;
 UINT08 uSetAdvertiseDataBuf[32];
} RSI_BLE_CMD_SET_ADVERTISE_DATA  ;
```

AT Command format:

at+rsibt_setadvertisedata=<DataLen>,<Data>\r\n

Parameters:

Length – data length. Max advertise data length is 31.

Data – Actual data

Response Payload:

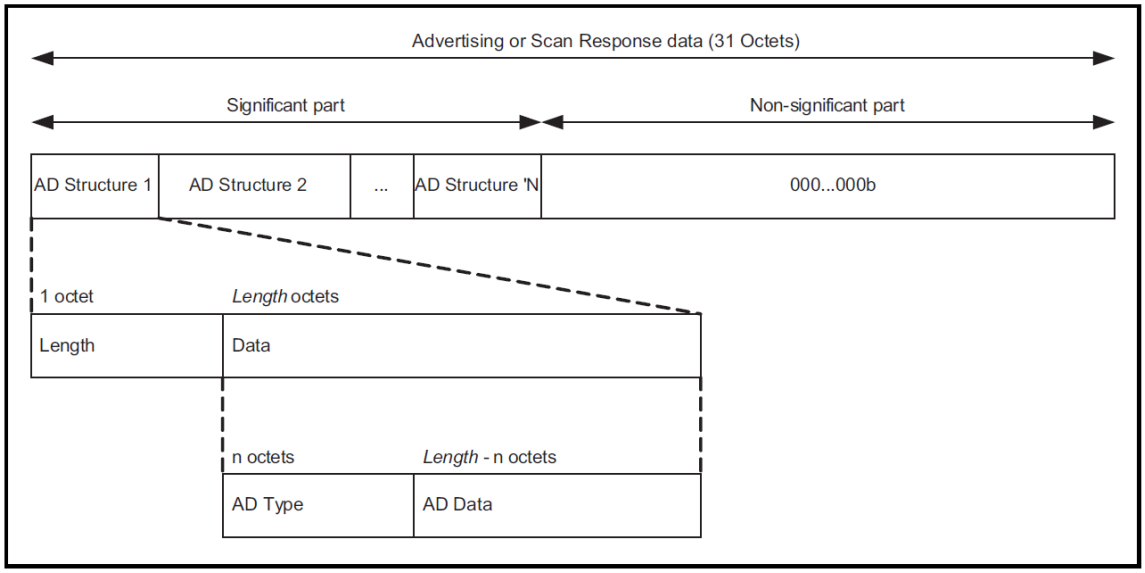
There is no response payload for this command.

AT command Ex: at+rsibt_setadvertisedata=8,2,1,6,4,9,72,72,72\r\n

Response: OK\r\n

> **NOTE:**
>
> Name set in this command will be shown on remote device when remote device scan for advertising device(for ios, refer NOTE in section: 2.4.1.2)

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

2.4.2.14 BLE Set Scan Response Data

Description: This command is used to set the scan response data.

Payload Structure:

```
typedef union {
  struct {
UINT08  DataLen;
  UINT08  Data[31];
}SetScanResponseDataFrameSnd  ;
 UINT08 uSetScanResponseDataBuf[32];
} RSI_BLE_CMD_SET_SCANRESP_DATA  ;
```

AT Command format:

at+rsibt_setscanrspdata=<DataLen>,<Data>\r\n

Parameters:

Length – data length. Max Scan Response data length is 31.

Data – Actual data

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_setscanrspdata=8,2,1,6,4,9,72,72,72\r\n

Response: OK\r\n

NOTE: Name set in this command will be shown on remote device when remote device scan for advertising device(for ios, refer NOTE in section: 2.4.1.2)

NOTE:

Controller has default scan response data which enable the flags 2,1,6.

2.4.2.15 BLE Set LE ping timeout[1]

Description: This command is used to set the LE ping timeout.

Payload Structure:

```
typedef union {
  struct {
  UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
  UINT16  Timeout;
 }SetLePingTimeoutFrameSnd  ;
 UINT08 uSetLePingTimeoutBuf[RSI_BT_BD_ADDR_LEN + 2];
} RSI_BLE_CMD_SET_PING_TIMEOUT  ;
```

AT command Format:

at+rsibt_setlepingtimeout=<BDAddress>, <Timeout>\r\n

Parameters:

BDAddress: Remote BD Address.

Timeout

Value	Parameter Description
N = 0xXXXX	Maximum amount of time specified between packets authenticated by a MIC. Default = 0x0BB8 (30 seconds) Range: 0x0001 to 0xFFFF Time = N * 10 msec Time Range: 10 msec to 655,350 msec

Response Payload:

There is no response payload for this command.

[1] Currently Set LE ping is not supported

AT command Ex: at+rsibt_setlepingtimeout=B4-99-4C-64-BE-F5,30000\r\n

Response: OK\r\n

2.4.2.16 BLE Get LE ping timeout[1]

Description: This command is used to get the LE ping timeout.

Payload Structure:

typedef struct rsi_ble_resp_get_ping_timeout{

UINT16 Timeout;

} RSI_BLE_RESP_GET_PING_TIMEOUT;

AT command Format:

at+rsibt_getlepingtimeout?\r\n

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_setlepingtimeout?\r\n

Response: OK 30000\r\n

2.4.2.17 BLE Set Random Address

Description: This command is used by the Host to set the LE Random Device Address in the Controller.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
  }SetRandAddFrameSnd;
  UINT08 uSetRandAddbuf[RSI_BT_BD_ADDR_LEN];
} RSI_BLE_CMD_SET_RANDOM_ADDRESS;
```

AT command Format:

at+rsibt_setrandadd=<BDAddress>\r\n

Parameters:

BDAddress:

Value	Parameter Description
0xXXXXXXXXXXXX	Random Device Address as defined by Device Address

[1] Currently Get LE ping is not supported

Return Parameters:

Status:

Value	Parameter Description
0x00	LE_Set_Random_Address command succeeded.
0x01 – 0xFF	LE_Set_Random_Address command failed.

Response Payload:

OK 0 \r\n

AT command Ex: at+rsibt_setrandadd= B4-99-4C-64-BE-F5\r\n

Response: OK 0\r\n

2.4.2.18 BLE Data Encrypt

Description: This command is used to encrypt the data.

Payload Structure:

```
typedef struct {
        UINT08  key[16];
        UINT08 data[16];
}RSI_BLE_CMD_ENCRYPT;
```

AT command Format:

at+rsibt_leencrypt=<key>,<data>\r\n

Parameters:

key – key length is 16 Bytes.

Data – Actual data , length is 16 Bytes.

Response Payload:

```
typedef struct rsi_ble_resp_data_encrypt {
        UINT08  EncData[16];
}RSI_BLE_RESP_DATA_ENCRYPT;
```

Result Code	Description
OK <EncData>	Command Success.
ERROR <Error_code>	Command Fail.

AT command Ex: at+rsibt_leencrypt=1,2,3,4,5,6,7,8,9,0,B,C,D,E,F,10,
1,2,3,4,5,6,7,8,9,0,B,C,D,E,F,10\r\n

Response: OK 10 93 c1 5a e4 a5 db fd 5e c2 4c 61 8a a3 11 28 \r\n

2.4.2.19 Set Antenna Tx power level

Description: This is used to set the Bluetooth antenna transmit power level.

Binary Payload Structure:

```
typedef struct rsi_bt_cmd_set_antenna_tx_power_level {
  UINT08 protocol_mode;
  INT08  tx_power;
} RSI_BT_CMD_SET_ANTENNA_TX_POWER_LEVEL;
```

AT command format:

at+rsibt_setantennatxpowerlevel=<protocol_mode>,<power_level>\r\n

Parameters:

protocol_mode –

Figure 1 – BT Low Energy

Power_level -
Minimum value – 1
Maximum value – 8

Response Payload:

There is no response payload for this command

AtcommandEx:

at+rsibt_setantennatxpowerlevel =2,6\r\n

Response: OK\r\n

2.4.2.20 BLE Whitelist

Description: This is used to add a particular BD-Address to the white list.

Binary Payload Structure:

```
typedef struct rsi_ble_cmd_le_addwhitelist {
  UINT08 AddorDeleteToWhitelist;
  UINT08  BDAddress[6];
  UINT08   BDAddressType;
} RSI_BLE_CMD_LE_WHITELIST;
```

AT command format:

at+rsibt_lewhitelist=<Add/deletebit>,<BDAddress>,<BDAddressType>\r\n

Parameters:

Add/Delete bit – This bit specifies the operation to be done
0 – Clear all entries
1 – Add entry to white list
2 – Delete entry from white list

BDAddress – BDAddress of the remote device that needed to be added to white list

BDAddressType – Type of the BDAddress
0- Public Device Address
1 – Random Device Address

Response Payload:

There is no response payload for this command

ATcommandEx:

at+rsibt_lewhitelist=1,00-23-A7-80-6F-CD,1\r\n

Response: OK\r\n

2.4.2.21 Read /Blob Read Response

Description : This is used to send local attribute value to the remote device

Binary Payload Structure:

```
typedef struct rsi_ble_gatt_read_response_s
{
  UINT08  BDAddress[6];
  UINT08  Type;
  UINT08  Reserved;
  UINT16  DataLength;
  UINT08  Data[MAX_GATT_EVENT_DATA_LEN];
}RSI_BLE_CMD_GATT_READ_RESONSE;
```

AT Command format: at+rsibt_readresp=<BDAddress>,<type>,<len>,<data>\r\n
Parameters:

BDAddress : BD address of remote device

Type : Attribute type (read response or read blob respose)

Reserved : Reserved for future use

DataLength: length of data our module is going to send (0 to MTU size). Data : Data to be sent to remote device, data size must be less than MTU size
(MAX_GATT_EVENT_DATA_LEN < MTU_SIZE)

Response Payload: There is no response payload for this command.

AT command Ex: at+rsibt_readresp=C0-FF-EE-C0-FF-EE,0,4,1,2,4,a\r\n

AT command Ex: at+rsibt_readresp=C0-FF-EE-C0-FF-EE,1,4,1,2,4,a\r\n

Response: OK\r\n

2.4.2.22 LE LTK Request Reply

Description : This is used to intimate controller about the long term key in host.

Binary Payload Structure:

```
typedef struct rsi_ble_cmd_le_ltkreqreply{
        UINT08  BDAddress[6];
        UINT08  ReplyType;
        UINT08  LocalLTK[16];
}RSI_BLE_CMD_LE_LTKREQREPLY;
```

AT command format:

at+rsibt_leltkreqreply=<BD Address>,<reply type>, <Local Long term key> \r\n

Parameters:

BDAddress – BDAddress of the remote device that needed to be added to white list

Reply Type –
0- Negative reply
1 – Positive reply

Local Long term key – Either NULL or 16 bytes value.

Response Payload:

There is no response payload for this command

ATcommandEx:

at+rsibt_leltkreqreply=6E-35-7C-35-50-2F,0,0 \r\n

at+rsibt_leltkreqreply=6E-35-7C-35-50-2F,1,<LocalLTK of 16 bytes> \r\n

Response: OK\r\n

2.4.2.23 SMP Reject Response

Description : This is used to reject the SMP pairing .

Binary Payload Structure:

```
struct {
   UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
   UINT08  Reserved[2];
   UINT08  ReasonCode;
 }BleSmpRejectResp;
```

AT command format:

at+rsibt_smpfailresp=<BD Address>,<Reson code> \r\n

Parameters:

BDAddress – BDAddress of the remote device that needed to reject the SMP process

Reson code –

Value	Name	Description
0x00	Reserved	Reserved for future use.
0x01	Passkey Entry Failed	The user input of passkey failed, for example, the user cancelled the operation
0x02	OOB Not Available	The OOB data is not available
0x03	Authentication Requirements	The pairing procedure cannot be performed as authentication requirements cannot be met due to IO capabilities of one or both devices
0x04	Confirm Value Failed	The confirm value does not match the calculated compare value
0x05	Pairing Not Supported	Pairing is not supported by the device
0x06	Encryption Key Size	The resultant encryption key size is insufficient for the security requirements of this device
0x07	Command Not Supported	The SMP command received is not supported on this device
0x08	Unspecified Reason	Pairing failed due to an unspecified reason
0x09	Repeated Attempts	Pairing or authentication procedure is disallowed because too little time has elapsed since last pairing request or security request
0x0A- 0xFF	Invalid Parameters	The Invalid Parameters error code indicates the command length is invalid a parameter is outside of the specified range.

Note:

If need to disable security use reson code as 0x05 i.e pairing is not supported.

Response Payload:

There is no response payload for this command

ATcommandEx:

at+rsibt_smpfailresp= 6E-35-7C-35-50-2F,5\r\n

Response: OK\r\n

.

2.4.3 BLE GATT Profile commands

2.4.3.1 Query profiles list

Description: This is used to query all the supported profiles list from the connected remote device.

Payload Structure:

```
typedef struct {
   UINT08    BDAddress[6];
   UINT16    StartHandle;
   UINT16    EndHandle;
} RSI_BLE_CMD_QUERY_PROFILES_LIST;
```

AT Command format:

at+rsibt_getallprofiles=<BDAddress>,<StartHandle>,<EndHandle>\r\n

Parameters:

BDAddress – Remote BD Address.

StartHandle – Start of the handle from which Include services are to be known.

EndHandle – End of the handle till which Include services are to be known.

Response Payload:

```
typedef struct bt_uuid {
   UINT08     size;
   UINT08     reserved[3];
   union bt_uuid_t {
      UUID128  val128;
      UUID32   val32;
      UUID16   val16;
   } Val;
} UUID_T;
typedef struct profile_descriptor{
   UINT16  StartHandle[2];
   UINT16  EndHandle[2];
   UUID_T    ProfileUUID;
}PROFILE_DESCRIPTOR;
typedef struct rsi_ble_resp_query_profiles_list {
  UINT08  NumberOfProfiles;
  UINT08  reserved[3];
PROFILE_DESCRIPTOR ProfileDescriptor[BLE_MAX_RESP_LIST];
} RSI_BLE_RESP_QUERY_PROFILES_LIST;
```

Result Code	Description
OK ,<nbr_profiles>,<profile descriptors list>	Command Success.
ERROR <Error_code>	Command Fail.

Response Parameters:

UUID Structure members	Size	Description
Size	8 bit	Size of the profile UUID
val128	128 bit	128 bit UUID
Val32	32 bit	32 bit UUID
Val16	16 bit	16 bit UUID

PROFILE_DESCRIPTOR Structure members	Size	Description
StartHandle[2]	8 bit	Start of the handle
EndHandle[2]	8 bit	End of the handle
ProfileUUID	UUID	UUID value

RSI_BLE_RESP_QUERY_PROFILES_LIST Structure members	Size/type	Description
NumberOfProfiles	8 bit	No of profiles supported by the remote device
ProfileDescriptor[10]	PROFILE_DESCRIPTOR	Profiles descriptors of the profiles

AT command Ex: at+rsibt_getallprofiles=B4-99-4C-64-BE-F5,1,10\r\n

Response: OK 3\n

1,B,2,1800\n

C,F,2,1801\n

10,FFFF,2,180A\r\n

2.4.3.2 Query Profile

Description: This is used to query particular profile details from the connected remote device.

Payload Structure:

```
typedef union {
  struct {
  UINT08  BDAddress[6];
          UINT08  Reserved[2];
  UUID_T    ProfileUUID;
```

```
 }QueryProfileDescFrameSnd  ;
 UINT08 uQueryProfileDescBuf[28];
} RSI_BLE_CMD_QUERY_PROFILE  ;
```

AT command format:

at+rsibt_getprofile=<BDAddress>,<size_uuid>,<ProfileUUID>\r\n

Parameters:

BDAddress – Remote BD Address.

ProfileUUID – UUID of the profile.

Response Payload:

```
typedef struct rsi_ble_resp_query_profile_descriptor {
   PROFILE_DESCRIPTOR  ProfileDescriptor;
} RSI_BLE_RESP_QUERY_PROFILE_DESCRIPTOR;
```

Result Code	Description
OK ,<profile descriptor>\r\n	Command Success.
ERROR <Error_code>	Command Fail.

Response Parameters:

ProfileDescriptor – PROFILE_DESCRIPTOR is explained above.

AT command Ex: at+rsibt_getprofile=77-A8-E3-CC-41-CB,2,1800\r\n

Response: OK 1,5,2,1800\r\n

2.4.3.3 Query Characteristic Services

Description: This is used to query characteristic services, with in the particular range, from the connected remote device.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[6];
   UINT08  StartHandle[2];
   UINT08  EndHandle[2];
  }QueryCharSerFrameSnd  ;
   UINT08 uQueryCharSerBuf[10];
```

} RSI_BLE_CMD_QUERY_CHARACTERISTIC_SERVICES ;

AT command format: at+rsibt_getcharservices=<BDAddress>,<StartHandle>,<EndHandle>\r\n

Parameters:

BDAddress – Remote BD Address

StartHandle – Start of the handle from which Characteristics services are to be known.

EndHandle – End of the handle till which Characteristics services are to be known.

Response Payload:

```
typedef struct bt_le_char_serv {
        UINT08  CharacterProperty;
        UINT08  Reserved;
        UINT16  CharacterHandle;
        UUID_T    CharacterUUID;
} BT_LE_CHAR_SERV;

typedef struct characteristic_service {
        UINT16   Handle;
        UINT08   Reserved[2];
        BT_LE_CHAR_SERV  CharServ;
}CHARACTERISTIC_SERVICE  ;

typedef struct rsi_ble_resp_query_char_services {
        UINT08          NumberOfCharServices;
        UINT08          reserved[3];
        CHARACTERISTIC_SERVICE  CharacteristicService[5];
} RSI_BLE_RESP_QUERY_CHARACTERISTIC_SERVICES;
```

Result Code	Description
OK ,<NumberOfCharServices>,< CharacteristicService >	Command Success.
ERROR <Error_code>	Command Fail.

Response parameters:

CHARACTERISTIC_SERVICE Structure members	Size/type	Description

CHARACTERISTIC_SERVICE Structure members	Size/type	Description
Handle	16 bit	Handle of the Characteristic service.
CharacterProperty	8 bit	Characteristic property
CharacterHandle	16 bit	Attribute handle of the Characteristic value
CharacterUUID	UUID	Characteristic UUID

RSI_BLE_RESP_QUERY_CHARACTERISTIC_SERVICES Structure members	Size/type	Description
NumberOfCharServices	8 bit	No of Characteristic services.
CharacteristicService[5]	CHARACTERISTIC_SERVICE	Each Characteristic Service details.

AT COMMAND Ex: at+rsibt_getcharservices=B4-99-4C-64-BE-F5,1,10\r\n

Response: OK 5, 2,2,3,2,2A00\n

4,2,5,2,2A01\n

6,2,7,2,2A02\n

8,8,9,2,2A03\n

A,2,B,2,2A04\r\n

2.4.3.4 Query Include Services

Description: This is used to query include services, with in the particular range, from the connected remote device.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[6];
   UINT08  StartHandle[2];
   UINT08  EndHandle[2];
  }QueryIncludeServFrameSnd  ;

  UINT08 uQueryIncludeServBuf[10];

} RSI_BLE_CMD_QUERY_INCLUDE_SERVICES  ;
```

AT Command format*:*

at+rsibt_getincservices=<BDAddress>,<StartHandle>,<EndHandle>\r\n

Parameters:

BDAddress – Remote BD Address.

StartHandle – Start of the handle from which Include services are to be known.

EndHandle – End of the handle till which Include services are to be known.

Response Payload:

```
typedef struct bt_le_inc_serv {
        UINT16  IncludeStartHandle;
        UINT16  IncludeEndHandle;
        UUID    IncludeUUID;
} BT_LE_INC_SERV;
typedef struct include_service {
   UINT16       Handle;
   UINT08       Reserved[2];
   BT_LE_INC_SERV  IncServ;
}INCLUDE_SERVICE  ;

typedef struct rsi_ble_resp_query_include_service {
   UINT08        NumberOfIncludeServices;
UINT08        reserved[3];
   INCLUDE_SERVICE  IncludeServices[BLE_MAX_RESP_LIST];
} RSI_BLE_RESP_QUERY_INCLUDE_SERVICE;
```

Response Parameters:

INCLUDE_SERVICE Structure members	Size/type	Description
Handle	16 bit	Handle of the Include service.
IncludeStartHandle	16 bit	Included Service Start Handle
IncludeEndHandle	16 bit	Included Service End Handle
IncludeUUID	UUID	Service UUID

RSI_BLE_RESP_QUERY_INCLUDE_ SERVICE Structure members	Size/type	Description
NumberOfIncludeServices	8 bit	No of Include services.
IncludeServices[5]	INCLUDE_SE RVICE	Each Include Service details.

AT command Ex: at+rsibt_getincservices=68-2F-10-0B-62-63,1,10\r\n

Response: OK 0,\r\n

2.4.3.5 Read Characteristic Value by UUID

Description: This is used to get the characteristic attribute value of specified UUID.

Payload Structure:

```
typedef union {
 struct {
  UINT08  BDAddress[6];
  UINT08  StartHandle[2];
  UINT08  EndHandle[2];
          UINT08  Reserved[2];
  UUID_T   CharacterUUID;
 }ReadCharValByUuidFrameSnd  ;

 UINT08 uReadCharValByUuidBuf[12 + sizeof(UUID_T)];

} RSI_BLE_CMD_READ_CHAR_VALUE_BY_UUID  ;
```

AT Command format:

at+rsibt_readbytype=<BDAddress>,<StartHandle>,<Endhandle>,<size>, <UUID>\r\n

Parameters:

BDAddress – Remote BD Address.

StartHandle – Start of the handle from which Attribute values are to be known.

EndHandle - End of the handle till which Attribute values are to be known.

Reserved - Padding

CharacterUUID – UUID whose Attribute values are to be known.

Response Payload:

```
typedef struct rsi_ble_resp_read_char_value_by_uuid {
  UINT08  NumberOfValues;
  UINT08  CharacterValue[30];
} RSI_BLE_RESP_READ_CHAR_VALUE_BY_UUID;
```

Result Code	Description
OK ,<NumberOfValues>,< CharacterValue >\r\n	Command Success.
ERROR <Error_code>	Command Fail.

Response Parameters:

NumberOfValues – Number of valid bytes in the CharacterValue.

CharacterValue – Attribute value of the given CharacterUUID

AT command Ex: at+rsibt_readbytype=65-65-11-B4-8C-08,1,10,2,2A00\r\n

Response: OK 6,3,0,69,50,61,64\r\n

2.4.3.6 Query Attribute

Description: This is used to query the Attribute Descriptors from the connected remote device. The Descriptor includes both the Handle and UUID.

Payload Structure:

```
typedef union {
  struct {
    UINT08  BDAddress[6];
    UINT08  StartHandle[2];
    UINT08  EndHandle[2];
  }QueryAttFrameSnd  ;

  UINT08 uQueryAttBuf[10];
} RSI_BLE_CMD_QUERY_ATT_DESC  ;
```

AT Command Format:

at+rsibt_getdescriptors=<BDAddress>,<StartHandle>,<EndHandle>\r\n

Parameters:

BDAddress – Remote BD Address.

StartHandle – The handle from which Attribute Descriptors are to be known.

EndHandle – The handle till which Attribute Descriptors are to be known.

Response Payload:

```
typedef struct attribute_descriptor {
    UINT16  Handle[2];
    UINT08  reserved[2];
    UUID_T    AttributeTypeUUID;
} ATTRIBUTE_DESCRIPTOR;


typedef struct rsi_ble_resp_query_att_desc {
    UINT08            NumberOfAttributes;
    UINT08            reserved[3];
    ATTRIBUTE_DESCRIPTOR  AttributeDescriptor[BLE_MAX_RESP_LIST];
} RSI_BLE_RESP_QUERY_ATT_DESC;
```

Result Code	Description
OK <NumberOfAttributes >,< AttributeDescriptor >\r\n	Command Success.
ERROR <Error_code>	Command Fail.

Response Parameters:

Handle – Handle of each Attribute

AttributeTypeUUID – UUID of each Attribute

NumberOfAttributes – No of attributes

AttributeDescriptor – Descriptor of each Attribute

AT command Ex: at+rsibt_getdescriptors=B4-99-4C-64-BE-F5,1,ffff\r\n

Response: OK 5

1,2,2800

2,2,2803

3,2,2A00

4,2,2803

5,2,2A01\r\n

2.4.3.7 Query Attribute Value

Description: This is used to query Attribute value from the connected remote device.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[6];
   UINT08  Handle[2];
  }QueryAttValFrameSnd  ;

  UINT08 uQueryAttValBuf[8];

} RSI_BLE_CMD_QUERY_ATT_VALUE  ;
```

AT Command format:

at+rsibt_readvalue=<BDAddress>,<Handle>\r\n

Parameters:

BDAddress – Remote BD Address.

Handle – Handle of the Attribute whose value is to be known.

Response Payload:

```
typedef struct rsi_ble_resp_query_att_value {
   UINT08  NumberOfValues;
   UINT08  AttributeValues[30];
} RSI_BLE_RESP_QUERY_ATT_VALUE;
```

Result Code	Description

Result Code	Description
OK ,<NumberOfValues>,< AttributeValues >\r\n	Command Success.
ERROR <Error_code>	Command Fail.

Response Parameters:

NumberOfValues – No of valid bytes in the AttributeValues

AttributeValues – Attribute value of the specified Handle

AT command Ex: at+rsibt_readvalue=65-65-11-B4-8C-08,1\r\n

Response: OK 2,0,18\r\n

2.4.3.8 Query Multiple Attribute Values

Description: This is used to query Multiple Attribute values from the connected remote device.

Payload Structure:

```
typedef union {
  struct {
    UINT08  BDAddress[6];
    UINT08  NumberOfHandles;
UINT08  Reserved;
    UINT16  Handles[5];
  }QueryMulAttValFrameSnd  ;

  UINT08 uQueryMulAttValBUF[18];

} RSI_BLE_CMD_QUERY_MULTIPLE_ATT_VALUES  ;
```

AT Command format:

at+rsibt_readmultiple=<BDAddress>,<NumberofHandles>,<Handles>\r\n

Parameters:

BDAddress – Remote BD Address.

NumberOfHandles – No of handles whose Attribute values are to be known.

Handles – The handle whose Attribute value is to be known.

Response Payload:

```
typedef struct rsi_ble_resp_query_multiple_att_values {
   UINT16  NumberOfValues;
   UINT08  AttributeValues[30];
} RSI_BLE_RESP_QUERY_MULTIPLE_ATT_VALUES;
```

Result Code	Description
OK ,<NumberOfValues>,< AttributeValues >	Command Success.
ERROR <Error_code>	Command Fail.

Response Parameters:

NumberOfValues – No of valid bytes in the AttributeValues

AttributeValues – Attribute value of the specified Handles

AT command Ex: at+rsibt_readmultiple=65-65-11-B4-8C-08,3,1,2,5\r\n

Response: OK 9,0,18,2,3,0,0,2A,80,2\r\n

2.4.3.9 Query Long Attribute Value

Description: This is used to query long Attribute value from the connected remote device. This is useful when the Attribute value is more than 30 bytes.

Payload Structure:

```
typedef union {
  struct {
    UINT08  BDAddress[6];
    UINT16  Handle;
    UINT16  Offset;
  }QueryLongAttValFrameSnd  ;

  UINT08 QueryLongAttValBuf[10];

} RSI_BLE_CMD_QUERY_LONG_ATT_VALUE  ;
```

AT Command format:

at+rsibt_longread=<BDAddress>,<Handle>,<Offset>\r\n

Parameters:

BDAddress – Remote BD Address.

Handle – Handle of the Attribute whose value is to be known.

Offset – Offset from which Attribute values are to be known.

Response Payload:

```
typedef struct rsi_ble_resp_query_long_att_value {
   UINT08  NumberOfValues;
   UINT08  LongAttValue[50];
} RSI_BLE_RESP_QUERY_LONG_ATT_VALUE;
```

Result Code	Description
OK ,<NumberOfValues>,< LongAttrValue >	Command Success.
ERROR <Error_code>	Command Fail.

Response Parameters:

NumberOfValues – No of valid bytes in the LongAttValue.

LongAttValue – Attribute value of the specified Handle.

AT command Ex: at+rsibt_longread=65-65-11-B4-8C-08,10,1\r\n

Response: OK 12,0,D9,D9,AA,FD,BD,9B,21,98,A8,49,E1,45,F3,D8,D1,69\r\n

2.4.3.10 Set Attribute Value

Description: This is used to Set attribute value of the connected remote device.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[6];
   UINT08  Handle[2];
   UINT08  Length;
   UINT08  Value[25];
  }SetAttValFrameSnd  ;

  UINT08 uSetAttValBuf[34];

} RSI_BLE_CMD_SET_ATT_VALUE  ;
```

AT Command format:

at+rsibt_writevalue=<BDAddress>,<Handle>,<Length>,<value>\r\n

Parameters:

BDAddress – Remote BD Address.

Handle – Handle of the Attribute whose value has to be set.

Length – No of bytes to be set in the specified Handle

Value – Value to be set in the specified Handle

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_writevalue=65-65-11-B4-8C-08,34,2,1,0\r\n

Response: OK \r\n

2.4.3.11 Set Attribute Value no Ack

Description: This is used to Set attribute value of the connected remote device. If Attribute value is set using this command, Ack will not be received from the remote device.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[6];
   UINT08  Handle[2];
   UINT08  Length;
   UINT08  Value[25];
 }SetAttValNoAckFrameSnd  ;
  UINT08 uSetAttValNoAckbuf[34];
} RSI_BLE_CMD_SET_ATT_VALUE_NO_ACK  ;
```

AT Command format:

at+rsibt_writecmd=<BDAddress>,<Handle>,<Length>,<value>\r\n

Parameters:

BDAddress – Remote BD Address.

Handle – Handle of the Attribute whose value has to be set.

Length – No of bytes to be set in the specified Handle

Value – Value to be set in the specified Handle

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_writecmd=65-65-11-B4-8C-08,1,1,2\r\n

Response: OK\r\n

2.4.3.12 Set Long Attribute Value

Description: This is used to Set long attribute value of the connected remote device. This is useful when the no of bytes to be set are more than 25 and when from a particular offset bytes are to be written.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[6];
   UINT08  Handle[2];
   UINT08  Offset[2];
   UINT08  Length;
   UINT08  Value[40];
```

```
 }SetLongAttValFrameSnd;

 UINT08 uSetLongAttValBuf[51];

}RSI_BLE_CMD_SET_LONG_ATT_VALUE;
```

AT Command format:

at+rsibt_longwrite=<BDAddress>,<Handle>,<Offset>,<Length>,<value>\r\n

Parameters:

BDAddress – Remote BD Address.

Handle – Handle of the Attribute whose value has to be set.

Offset – Offset from which Value has to set in the specified Handle

Length – No of bytes to be set in the specified Handle

Value – Value to be set in the specified Handle

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_longwrite =C0-FF-EE-C0-FF-EE,1,1,1,2\r\n

Response: OK\r\n

2.4.3.13 Set Prepare Long Attribute Value

Description: This is used to Set Long Attribute value in the connected remote device. When Value is set using this API, the remote device will wait for “Execute Long Attribute Value ” to come before updating its handle with the value received.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[6];
   UINT08  Handle[2];
   UINT08  Offset[2];
   UINT08  Length;
   UINT08  Value[40];
  }SetPreLongAttValFrameSnd;
  UINT08 uSetPreLongAttValBuf[51];
}RSI_BLE_CMD_SET_PREPARE_LONG_ATT_VALUE;
```

AT Command format:

at+rsibt_preparewrite=<BDAddress>,<Handle>,<Offset>,<Length>,<Value>\r\n

Parameters:

BDAddress – Remote BD Address.

Handle – Handle of the Attribute whose value has to be set.

Offset – Offset from which Value has to set in the specified Handle

Length – No of bytes to be set in the specified Handle

Value – Value to be set in the specified Handle

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_preparewrite=B4-99-4C-64-BC-AF,1,1,1,2\r\n

Response: OK \r\n (or)

ERROR, err_no\r\n

2.4.3.14 Execute Long Attribute Value

Description: This is used to send Execute Long Attribute Value to the connected remote device. Depending on the "flag" of this command, the remote device will either set/not set the previously sent Prepare Long Attribute values.

Payload Structure:

```
typedef union {
   struct {
   UINT08  BDAddress[6];
   UINT08  Flag;
  }ExeLongAttValWrFrameSnd  ;

  UINT08 uExeLongAttValWrbuf;

} RSI_BLE_CMD_EXECUTE_LONG_ATT_VALUE_WRITE  ;
```

AT Command format:

at+rsibt_executewrite=<BDAddress>,<Flag>\r\n

Parameters:

BDAddress – Remote BD Address.

Flag – This parameter decides whether to set/not set the previously sent Prepare Long Attribute values.

0 – Don't set the values

1 – Set the values

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_executewrite=C0-FF-EE-C0-FF-EE,1\r\n

Response: OK\r\n

2.4.4 BLE Create New Service Commands

2.4.4.1 Add GATT Service Record

Description: This is used to add the new service Record in BLE GATT record list. We can get service record handle if service is created successfully. Else get error value.

Payload Structure:

```
typedef union {
   struct  {
     UUID_T       ServiceUUID;
     UINT16      NbrAttributes;
     UINT16      MaxAttDataSize;
  } AddServiceRecord;

  UINT08 uAddServiceRecord[24];

} RSI_BLE_CMD_ADD_GATT_SERVICE;
```

AT Command format:

at+rsibt_addservice=<uuid_size>,<ServiceUUID>,<NbrAttributes>,<MaxAttDataSize>\r\n

Parameters:

ServiceUUID – BLE supporting service UUID value.

NbrAttributes – number of attributes need to add for this service.

MaxAttDataSize – maximum number of data length that can be used in attribute list.

Response Payload:

```
typedef struct  rsi_ble_resp_add_gatt_service {
   void       *ServiceHndlerPtr;
   UINT16      StartHndl;
} RSI_BLE_RESP_ADD_GATT_SERVICE;
```

Result Code	Description
OK ,< ServiceHndlerPtr >,< StartHndl >	Command Success.
ERROR <Error_code>	Command Fail.

Response Parameters:

ServiceHndlerPtr – created GATT service record handle.

StartHndl - service record starting attribute handle value.

AT command Ex: at+rsibt_addservice=2,18ff,3,30\r\n

Ex 2:Adding 128 bit service

at+rsibt_addservice=10,11223344,-5566,-7788,-9910,12,11,16,15,14,13,3,30\r\n

Explanation : First 10 bytes(11223344-5566-7788-9910) has to be added as given in command as it is. Then next 6 bytes (111213141516) has to be splitted in to 2+4 bytes,This 2 and 4 bytes has to be given as bytes format in little enidian format.

Response: OK 17F24,A\r\n

Response: OK 157A8,A\r\n

NOTE:

NbrAttributes is used for number of attributes of a specific service.

MaxAttDataSize is sum up of all attributes data size of a specific service.

NbrAttributes, MaxAttDataSize – Currently reserved.

2.4.4.2 Add Attribute Record

Description: This is used to add the attribute record to the specific service using service record handle.

Payload Structure:

```
typedef union {
    struct  {
    void     *ServiceHndlerPtr;
    UINT16    Hndl;
    UINT16    Reserved;
    UUID_T    AttUUID;
    UINT08    Prop;
    UINT08    Data[31];
    UINT16    DataLen;
  } AddAttRecord;

 UINT08 uAddAttRecord[62];

} RSI_BLE_CMD_ADD_GATT_ATTRIBUTE;
```

Note:

If reserved bit has set to 1, then application needs to send the response for the read request event. It is the same when data length is more than 20bytes.

AT Command format:

at+rsibt_addattribute=<ServiceHndlerPtr>,<Hndl>,<AttUUIDsize>,<AttUUID>,<prop>,<DataLen>,<Data>\r\n

Parameters:

ServiceHndlerPtr – service record handle.

Hndl – handle of the attribute record.
AttUUID – attribute record UUID.
Data – attribute record data value.
DataLen – attribute record data length.
Prop - property of the attribute.

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_addattribute=157A8,A,2,2A02,1,1,FF\r\n

Response: OK\r\n

EX 2: Adding 128 bit attribute

at+rsibt_addattribute=17F24,B,2,2803,2,14,2,0,C,0,40,30,20,10,60,50,80,70,11,90,BB,AA,FF,EE,DD,CC\r\n

Explanation : In this command 16 byte char(128 bits) uuid must be given as data bytes in little endian order for 2 and 4 bytes(First 10 bytes which are 10203040-5060-7080-9011 in this example) and for last 6 bytes in uuid (AABBCCDDEEFF) split into 2 + 4 bytes and give in little endian order.

Response: OK\r\n

2.4.4.3 Set Local Attribute Value

Description: This is used to set/change the local attribute record value to the specific service using service record handle.

Payload Structure:

```
typedef struct  {
   UINT16        Hndl;
   UINT16        DataLen;
   UINT08        Data[31];
} RSI_BLE_CMD_SET_LOCAL_ATT_VALUE;
```

AT Command format:

at+rsibt_setlocalattvalue=<Hndl>,<DataLen>,<Data>\r\n

Parameters:

Hndl – handle of the attribute record.
DataLen – attribute record data length.
Data – attribute record data value.

Response Payload:

There is no response payload for this command.

AT command Ex: at+rsibt_setlocalattvalue=A,1,EE\r\n

Response: OK\r\n

2.4.4.4 Get Local Attribute Value

Description: This is used to read the local attribute record value to the specific service using service record handle.

Payload Structure:

```
typedef struct  {
   UINT16        Hndl;
} RSI_BLE_CMD_GET_LOCAL_ATT_VALUE;
```

Parameters:

Hndl – handle of the attribute record.

AT Command format:

at+rsibt_getlocalattvalue=<Hndl>\r\n

Response Payload:

```
typedef struct  {
   UINT16        Hndl;
   UINT16        DataLen;
   UINT08        Data[31];
} RSI_BLE_CMD_SET_LOCAL_ATT_VALUE;
```

Result Code	Description
OK ,<Handle >,<data_len>,<data>	Command Success.
ERROR <Error_code>	Command Fail.

Parameters:

Hndl – handle of the attribute record.
DataLen – attribute record data length.
Data – attribute record data value.

AT command Ex: at+rsibt_getlocalattvalue=A\r\n

Response: OK A,1,EE\r\n

2.4.5 BLE Core Events

2.4.5.1 Advertise Report Event

Description: This event indicates the remote device's Advertisement. It comes when **Scan** is enabled in the local device.

Payload Structure:

```
typedef struct rsi_bt_event_le_advertise_report {
  UINT08   BDAddresstype;
  UINT08   BDAddress[6];
  UINT08   AdvDataLen;
  UINT08   AdvData[31];
  UINT08   RSSI;
} RSI_BT_EVENT_LE_ADVERTISE_REPORT;
```

AT Event format:

AT+RSIBT_ADVRTISE, <<addr_type >, <bd_addr >,<RSSI>,<adv_data_len>,< adv_data >>\r\n

Parameters:

addr_type: Type of address of advertiser

0- Public address

1- Random address

addr: Advertiser address

RSSI: Signal strength indication between devices

adv_data_len: total raw advertisement data length

adv_data: advertisement data

AT event Ex:

AT+RSIBT_ADVRTISE,addr_type:0,addr:B4-99-4C-64-BC-AF,RSSI:-31,adv_data_len:3,adv_data:2,1,6

In the above example

addr_type = 0

addr = B4-99-4C-64-BC-AF

RSSI = -31

adv_data_len = 3

adv_data = 2,1,6

NOTE:

1. RSSI in decimal format
2. addr_type, addr, adv_data_len are in ascii format

2.4.5.2 Connection Complete Event

Description: This event indicates either the Connection is Success or not.

Payload Structure:

```
typedef struct rsi_bt_event_le_connection_status {
  UINT08  BDAddresstype;
  UINT08  BDAddress[RSI_BT_BD_ADDR_LEN];
  UINT08  Status;
} RSI_BT_EVENT_LE_CONNECTION_STATUS;
```

AT Event format:

AT+RSIBT_ LE_DEVICE_CONNECTED= <addr_type>,<bd_addr>,<status>\r\n

Parameters:

BDAddresstype – Address type of the connected device

0 – Public address

1 – Random address

BDAddress – BD Address of the connected device

Status – 0 - Success

2 - Create connection cancel success when disconnect command is issued

Non-zero – Failure

AT event Ex: AT+RSIBT_LE_DEVICE_CONNECTED=0,B4-99-4C-64-BE-F5,0\r\n

2.4.5.3 Disconnected

Description:

This event is raised when disconnection happens between the local BT device and the remote device.

Payload Structure:

```
typedef  struct rsi_bt_event_disconnected {
   UINT08  BDAddress[6];
   UINT08  Type;
} RSI_BT_EVENT_DISCONNECTED;
```

AT Event Format:

AT+RSIBT_LE_DISCONNECTED <bd_addr>,<reason>\r\n

Parameters:

BDAddress – BD address of the remote BT device.

AT event Ex: AT+RSIBT_LE_DISCONNECTED AA-BB-CC-DD-EE-FF,0\r\n

> **NOTE:**
>
> After disconnection happens with a remote device, information related to connection, encryption of both local and remote devices is cleared. Services and attributes related information which is present in local device is not cleared.

2.4.5.4 SMP Request Event

Description:

This event is raised when the SMP Request comes from the connected remote device.

Action:

Upon reception of this event **SMP Response** command has to be given.

Payload Structure:

```
typedef struct  rsi_bt_event_smp_req{
   UINT08  BDAddress[6];
} RSI_BT_EVENT_SMP_REQ;
```

Event format:

AT+RSIBT_SMPREQ <bd_addr>,<i/o capability>\r\n

Parameters:

BDAddress – Remote BD Address.

AT event Ex: AT+RSIBT_SMPREQ AA-BB-CC-DD-EE-FF,0\r\n

2.4.5.5 SMP Response Event

Description:

This event is raised when the SMP Response comes from the connected remote device.

Action:

Upon reception of this event **SMP Passkey** command has to be given.

Payload Structure:

```
typedef  struct  rsi_bt_event_smp_resp  {
   UINT08  BDAddress[6];
} RSI_BT_EVENT_SMP_RESP;
```

AT event format: AT+RSIBT_SMP_RESPONSE <bd_addr>\r\n

Parameters:

BDAddress – Remote BD Address.

AT Event Ex: AT+RSIBT_SMP_RESPONSE AA-BB-CC-DD-EE-FF \r\n

2.4.5.6 SMP Passkey Event

Description:
This event is raised when the SMP Passkey comes from the connected remote device.

Action:

Upon reception of this event **SMP Passkey** command has to be given.

Payload Structure:

```
typedef struct  rsi_bt_event_smp_passkey {
   UINT08  BDAddress[6];
} RSI_BT_EVENT_SMP_PASSKEY;
```

Event format*:*
AT+RSIBT_SMPPASSKEY <bd_addr>\r\n
Parameters:

BDAddress – Remote BD Address.

AT event Ex: AT+RSIBT_SMP_PASSKEY AA-BB-CC-DD-EE-FF\r\n

2.4.5.7 SMP Passkey Display Event

Description:
This event is raised when the SMP Passkey comes from the connected remote device.

> **Note:**
>
> If IO_Capability request in SMP Passkey Requset is DISPLAY_ONLY then only this event occurs.

Action:

Upon reception of this event **SMP Passkey** command is not required.

Payload Structure:

```
typedef struct  rsi_bt_event_smp_passkey_display {
   UINT08  BDAddress[6];
   UINT08  Reserved;
   UINT32  Passkey;
} RSI_BT_EVENT_SMP_PASSKEY_DISPLAY;
```

Event format*:*
AT+RSIBT_SMPPASSKEY_DISPLAY <bd_addr>,<Passkey>\r\n

Parameters:

BDAddress – Remote BD Address.

Passkey - Passkey

AT event Ex: AT+RSIBT_SMP_PASSKEY_DISPLAY AA-BB-CC-DD-EE-FF,123456\r\n

> **Note:**
>
> Iif IO_Capability request as DISPLAY_ONLY in both sides then SMP will won't work.

2.4.5.8 SMP Failed Event

Description:

This event is raised when the SMP exchange fails. The reason for failure would be present in the Descriptor frame.

Payload Structure:

```
typedef struct rsi_bt_event_smp_failed{
   UINT08  BDAddress[6];
} RSI_BT_EVENT_SMP_FAILED;
```

Event format*:*

AT+RSIBT_SMPFAILED ,<bd_addr>,<error>\r\n

Parameters:

BDAddress – Remote BD Address.

Error – reason for SMP failing.

Event Ex: AT+RSIBT_SMPFAILED ,AA-BB-CC-DD-EE-FF,4B01\r\n

2.4.5.9 SMP Encrypt Enabled Event

Description:

This event is raised when the encryption gets started.

Payload Structure:

```
typedef struct rsi_ble_event_encryption_enabled {
   UINT08  BDAddress[6];
   UINT08  Enabled;
   UINT08  Reserved;
   UINT16  LocalEDIV;
   UINT08  LocalRand[8];
   UINT08  LocalLTK[16];
} RSI_BT_EVENT_ENCRYPTION_ENABLED;
```

Event format:

AT+RSIBT_ENCRYPTION_ENABLED <BDAddr>,< LocalEDIV >,< LocalRand >,< LocalLTK >\r\n

AT+RSIBT_ENCRYPTION_DISABLED\r\n

Parameters:

LocalEDIV is a 16-bit stored value used to identify the LTK distributed during LE legacy pairing. A new EDIV is generated each time a unique LTK is distributed.

LocalRand is a 64-bit stored valued used to identify the LTK distributed during LE legacy pairing. A new Rand is generated each time a unique LTK is distributed.

LocalLTK is a 128-bit key used to generate the contributory session key for an encrypted connection.

Event Ex: AT+RSIBT_ENCRYPTION_DISABLED 4E06

2.4.5.10 LE ping payload timeout[1]

Description:

This event is raised when the LE ping timeout exceeds.

Payload Structure:

```
typedef struct {
  UINT08  BDAddress[6];
} RSI_BT_EVENT_LE_PING_TIMEOUT_EXPIRED;
```

Event format*:*

AT+RSIBT_LE_PING_TIMEOUT_EXPIRED <bd_addr>, <error>\r\n

Parameters:

BDAddress – Remote BD Address.

Event Ex: AT+RSIBT_LE_PING_TIMEOUT_EXPIRED 00-23-A7-12-23-24, 0103\r\n

2.4.5.11 LE MTU Size

Description:

This event is raised after LE connection.

Payload Structure:

```
typedef struct {
  UINT08  BDAddress[6];
  UINT08  MTU_size[2];
} RSI_BT_EVENT_MTU_SIZE;
```

Parameters:

BDAddress – Remote BD Address.

MTU_size - Size of MTU

NOTE: This event is not supported in AT mode

[1] Currently LE ping is not supported

2.4.5.12 LE LTK Request Event

Description:

This event is raised when the LE long term key of local device is requested by it's LE controller.

Payload Structure:

```
typedef struct rsi_bt_event_le_ltk_request {
  UINT08  BDAddress[6];
  UINT16  LocalEDIV;
  UINT08  LocalRand[8];
} RSI_BT_EVENT_LE_LTK_REQUEST;
```

Event format*:*
AT+RSIBT_LTK_REQUEST <bd_addr>, < LocalEDIV >, < LocalRand >\r\n

Parameters:

BDAddress – Remote BD Address.

LocalEDIV is a 16-bit stored value used to identify the LTK distributed during LE legacy pairing. A new EDIV is generated each time a unique LTK is distributed.

LocalRand is a 64-bit stored valued used to identify the LTK distributed during LE legacy pairing. A new Rand is generated each time a unique LTK is distributed.

2.4.6 BLE GATT Events

2.4.6.1 GATT Notification

Description:

This event is raised when GATT Notification packet is received from the connected remote device.

Payload Structure:

```
typedef struct rsi_ble_event_gatt_char_value_notifications {
  UINT08  BDAddress[6];
  UINT08  Handle[2];
  UINT08  Length;
  UINT08  Value[50];
} RSI_BLE_EVENT_GATT_CHAR_VALUE_NOTIFICATIONS;
```

Event format:
AT+RSIBT_NOTIFY,<bd_addr>,<handle>,<length>,<value>\r\n

Parameters:

BDAddress – Remote BD Address.

Handle – Handle related to the Value

Length – Number of valid bytes in the Value

Value – The contents of the received Notification packet

Event Ex: AT+RSIBT_NOTIFY,C0-FF-EE-C0-FF-EE,33,1,2\r\n

2.4.6.2 GATT Indication

Description:

This event is raised when GATT Indication packet is received from the connected remote device.

Payload Structure:

```
typedef struct rsi_ble_event_gatt_char_value_indication {
  UINT08  BDAddress[6];
  UINT08  Handle[2];
  UINT08  Length;
  UINT08  Value[50];
} RSI_BLE_EVENT_GATT_CHAR_VALUE_INDICATION;
```

Event format:

AT+RSIBT_INDICATION,<bd_addr>,<handle>,<length>,<value>\r\n

Parameters:

BDAddress – Remote BD Address.

Handle – Handle related to the Value

Length – Number of valid bytes in the Value

Value – The contents of the received Indication packet

Event Ex: AT+RSIBT_INDICATION,C0-FF-EE-C0-FF-EE,33,1,2\r\n

2.4.6.3 GATT Write

Description:

This event is raised when GATT write packet is received from the connected remote device.

Payload Structure:

```
typedef struct rsi_ble_event_gatt_write{
  UINT08  BDAddress[6];
  UINT08  Handle[2];
  UINT08  Length;
  UINT08  Value[50];
} RSI_BLE_EVENT_GATT_WRITE;
```

Event format:

AT+RSIBT_WRITE,<bd_addr>,<handle>,<length>,<value>\r\n

Parameters:

BDAddress – Remote BD Address.

Handle – Handle related to the Value

Length – Number of valid bytes in the Value

Value – The contents of the received Notification packet

Event Ex: AT+RSIBT_INDICATION,C0-FF-EE-C0-FF-EE,33,1,2\r\n

2.4.6.4 GATT READ /Blob Read Request

Description:

This event is raised when read request is received from the connected remote device.

Payload Structure:

```
typedef struct rsi_bt_event_le_gatt_read_req{
   UINT08  BDAddress[6];
   UINT16  Handle;
   UINT08  Type;
   UINT08  Reserved;
   UINT16  Offset;
} RSI_BT_EVENT_LE_READ_REQ;
```

Parameters:

BDAddress – Remote BD Address.

Handle – Handle related to the Value

Type - To indicate read type

0 – read request

1 – read blob request

Reserved – Reserved for future use

Offset – The offset of the first octet to be read.

Event format:

AT+RSIBT_READ_REQ <bd_addr>,<handle>\r\n

Event Ex: AT+RSIBT_READ_REQ C0-FF-EE-C0-FF-EE,C\r\n

2.5 Bluetooth Generic Error Codes

Error Code	Description
0x0103	Timeout
0x0059	BT feature bitmap invalid
0xff2c	Memory limit exceeded for given opermode
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
0X4E3D	Connection Terminated due to MIC Failure
0X4E60	Invalid Handle Range
0x4FF8	Invalid command string

Table 6 Bluetooth Generic Error Codes

2.6 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
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)
0x4E3E	Connection Failed to be Established

0x4057	Hardware Buffer Overflow
0x4046	Invalid Args

Table 7 BLE Error Codes

2.7 Power Save

Description:

This feature explains the configuration of **Power Save** modes of the module.These can be issued at any time after Opermode command. By default, Power Save is in disable state.

There are five different modes of Power Save. They are outlined below.

1. Power Save mode 0
2. Power Save mode 2
3. Power Save mode 3
4. Power Save mode 8
5. Power Save mode 9

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, the host has to re-initialize SPI interface of the module.
3. When co-ex mode is enable, power save mode is not applicable for WLAN also.

2.7.1 Power Save Operations

The behavior of the module differs as per the Power Save mode it is configured with.

2.7.1.1 Power Save Mode 0

In this mode the module is in active state and power save is been disabled. It can be configured at any time while power save is enable with Power Save mode 2 and 3 or Power Save mode 8 and 9.

2.7.1.2 Power Save Mode 2 (GPIO based mode)

For detail description for power save mode 2, please refer to the section **8.16.1.2** in **RS9113-WiseConnect-Software-PRM-v1.X.X.pdf**.

2.7.1.3 Power Save Mode 3 (Message based mode)

For detail description for power save mode 3, please refer to the section **8.16.1.3** in **RS9113-WiseConnect-Software-PRM-v1.X.X.pdf**.

In LE, Power Save mode 2 and 3 can be used during Advertise /Scan /Connected states. Operational behavior is as below depending on the state.

- **Advertise State:** In this state, the module is awake during advertising event duration and sleeps till Advertising interval. **Scan State:** In this state, the module is awake during scanning window duration and sleeps till scanning interval.
- **Connected state:** In this state, the module wakes up for every connection interval. The default connection interval is 200 msec which is configurable.

2.7.1.4 Power Save Mode 8 (GPIO based mode)

For detail description for power save mode 8, please refer to the section **8.16.1.4** in **RS9113-WiseConnect-Software-PRM-v1.X.X.pdf**.

2.7.1.5 Power Save Mode 9 (Message based mode)

For detail description for power save mode 9, please refer to the section **8.16.1.4** in **RS9113-WiseConnect-Software-PRM-v1.X.X.pdf**.

Note:

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

Ex:

Suppose if Power Save is enabled in standby state, so, in order to move to Scanning state, first Power Save disable command need to be issued before giving Scan command.

2) 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).

3 Bluetooth API Library

3.1 API File Organization

Bluetooth APIs are organized into following directory structure

	Path(Within *RS9113.WC.GENR.x.x.x* folder)
BLE APIs	host/apis/ble/core/
Interface Specific APIs	host/apis/intf/
HAL APIs	host/apis/hal/
BLE Reference Applications	host/apis/ble/ref_apps/
BLE Linux Application	RS9113.WC.GENR.xxx/host/reference_projects/LINUX/Application/ble/src
Linux USB Driver	host/reference_projects/LINUX/Driver/usb/src

3.2 API Prototypes

3.2.1 Generic Prototypes

3.2.1.1 Set Local name

INT16 rsi_bt_set_local_name(RSI_BT_CMD_SET_LOCAL_NAME *SetLocalName);

3.2.1.2 Query Local name

INT16 rsi_bt_query_local_name(void);

3.2.1.3 Query RSSI

INT16 rsi_bt_query_rssi(RSI_BT_CMD_QUERY_RSSI *GetRSSI);

3.2.1.4 Query Local BD Address

INT16 rsi_bt_query_local_bd_address(void);

3.2.2 BLE Core Prototypes

3.2.2.1 Advertise Local Device

INT16 rsi_ble_advertise(RSI_BLE_CMD_ADVERTISE *LEAdv);

3.2.2.2 Scan

INT16 rsi_ble_scan(RSI_BLE_CMD_SCAN *LEScan);

3.2.2.3 Connect

INT16 rsi_ble_connect(RSI_BLE_CMD_CONNECT *LEConn);

3.2.2.4 Disconnect

INT16 rsi_ble_disconnect(RSI_BLE_CMD_DISCONNECT *uLEDisconnect);

3.2.2.5 Query Device State

INT16 rsi_ble_query_device_state(void);

3.2.2.6 Start Encryption

INT16 rsi_ble_start_encryption(RSI_BLE_CMD_ENCRYPTTION *uLEEncrypt);

3.2.2.7 SMP Pair Request

INT16 rsi_ble_smp_pair_request(RSI_BLE_CMD_SMP_PAIR_REQUEST *SMPPairReq);

3.2.2.8 SMP Response

INT16 rsi_ble_smp_response(RSI_BLE_CMD_SMP_RESPONSE *SMPResp);

3.2.2.9 SMP Passkey

INT16 rsi_ble_smp_passkey(RSI_BLE_CMD_SMP_PASSKEY *SMPPasskey);

3.2.2.10 BLE Init

INT16 rsi_bt_device_init();

3.2.2.11 BLE Deinit

INT16 rsi_bt_device_deinit();

3.2.2.12 BT Antenna Select

INT16 rsi_bt_antenna_select(RSI_BT_CMD_ANTENNA_SELECT *uAntennaSelect);

3.2.2.13 Set Advertise Data Payload

INT16 rsi_ble_device_SetAddvertiseData(RSI_BLE_CMD_SET_ADVERTISE_DATA *uLEAdvertiseData);

3.2.2.14 Set LE Ping Timeout[1]

INT16 rsi_ble_device_SetLePingTimeout(RSI_BLE_CMD_SET_PING_TIMEOUT * uSetLePingTimeout);

3.2.2.15 Get LE Ping Timeout[2]

INT16 rsi_ble_device_GetLePingTimeout(void);

3.2.2.16 Set Random Address

INT16 rsi_ble_set_random_address(RSI_BLE_CMD_SET_RANDOM_ADDRESS *uSetRandAdd);

3.2.2.17 LE Data Encrypt

INT16 rsi_ble_data_encrypt(RSI_BLE_CMD_DATA_ENCRYPT *uDataEncrypt);

3.2.2.18 SMP Pairing Rejection

INT16 rsi_ble_smp_pairing_rejection(RSI_BLE_CMD_SMP_REJECT_REPLY *uBleRejectResp);

[1] Currently LE ping is not supported
[2] Currently LE ping is not supported

3.2.3 BLE GATT Prototypes

3.2.3.1 Query profiles list

INT16 rsi_ble_query_profiles_list(RSI_BLE_CMD_PROFILE_LIST *uGetProfileList);

3.2.3.2 Query Profile

INT16 rsi_ble_query_profile(RSI_BLE_CMD_QUERY_PROFILE *GetProf);

3.2.3.3 Query Characteristic Services

INT16
rsi_ble_query_characteristic_services(RSI_BLE_CMD_QUERY_CHARACTERISTIC_SERVICES *GetCharServ);

3.2.3.4 Query Include Services

INT16 rsi_ble_query_include_service(RSI_BLE_CMD_QUERY_INCLUDE_SERVICES *GetIncludeServ);

3.2.3.5 Read Characteristic Value by UUID

INT16
rsi_ble_read_char_value_by_UUID(RSI_BLE_CMD_READ_CHAR_VALUE_BY_UUID *ReadCharValByUUID);

3.2.3.6 Query Attribute

INT16 rsi_ble_query_att(RSI_BLE_CMD_QUERY_ATT_DESC *GetAtt);

3.2.3.7 Query Attribute Value

INT16 rsi_ble_query_att_value(RSI_BLE_CMD_QUERY_ATT_VALUE *GetAttVal);

3.2.3.8 Query Multiple Attribute Values

INT16
rsi_ble_query_multi_att_values(RSI_BLE_CMD_QUERY_MULTIPLE_ATT_VALUES *GetMulAttVal);

3.2.3.9 Query Long Attribute Value

INT16 rsi_ble_query_long_att_value(RSI_BLE_CMD_QUERY_LONG_ATT_VALUE *GetLongAttVal);

3.2.3.10 Set Attribute Value

INT16 rsi_ble_set_att_value(RSI_BLE_CMD_SET_ATT_VALUE *SetAttVal);

3.2.3.11 Set Attribute Value no Ack

INT16 rsi_ble_set_att_value_no_ack(RSI_BLE_CMD_SET_ATT_VALUE_NO_ACK *SetAttValNoAck);

3.2.3.12 Set Long Attribute Value

INT16 rsi_ble_set_long_att_value(RSI_BLE_CMD_SET_LONG_ATT_VALUE *SetLongAttVal);

3.2.3.13 Set Prepare Long Attribute Value

INT16 rsi_ble_set_prep_long_att_value(RSI_BLE_CMD_SET_PREPARE_LONG_ATT_VALUE * uLePrepareAttVal);

3.2.3.14 Execute Long Attribute Value

INT16 rsi_ble_execute_long_att_value(RSI_BLE_CMD_EXECUTE_LONG_ATT_VALUE_WRITE *ExeLongAttVal);

3.2.3.15 Add GATT Service Record

INT16 rsi_ble_device_AddService(RSI_BLE_CMD_ADD_GATT_SERVICE *uLEAddService);

3.2.3.16 Add Attribute Record

INT16 rsi_ble_device_AddServiceAttribute(RSI_BLE_CMD_ADD_GATT_ATTRIBUTE *uLEAddAttribute);

3.2.3.17 Set Local Attribute Record value

INT16 rsi_ble_device_SetLocalAttValue(RSI_BLE_CMD_SET_LOCAL_ATT_VALUE *uLESetLocalAttVal);

3.2.3.18 Get Local Attribute Record value

INT16 rsi_ble_device_GetLocalAttValue(RSI_BLE_CMD_GET_LOCAL_ATT_VALUE *uLEGetLocalAttVal);

3.2.3.19 BLE Whitelist

INT16 rsi_ble_white_list(RSI_BLE_CMD_WHITE_LIST *uBleWhiteListl);

3.3 Application

The files in the Applications folders contain files for the application layer of the Host MCU. These have to be modified to setup the application for the system which the user wants to realize. The user has to call the APIs provided in the API library to configure the BLE device.

1. ble_main.c – This file contains the entry point for the application. It also has the initialization of parameters of the global structure and the operations to control & configure the module, like advertising, connecting etc. Here we just provided sample code for the user to understand the flow of commands. This is not must to use the same. User can write his own application code instead of that.
2. rsi_app_util.h and rsi_app_util.c – These files contain list of utility functions which are used by rsi_ble_config_init API and debug prints.
3. rsi_ble_config.h and rsi_ble_config_init.c – These files contain all the parameters to configure the module. Some example parameters are BD Address of the device with which the module should connect, etc.

To facilitate Application development we have defined a data structure named RSI_BLE_API as described below.This structure is initialized by the application using rsi_ble_init_struct API of the rsi_ble_config_init.c file (application layer file). The user may change the values assigned to the macros without worrying about understanding the contents of the structure.
The contents of this structure are explained in brief below, using the declaration of the structure in rsi_ble_global.h file (which is also an application layer file and is placed in core APIs include folder).

```
Typedef union{
 RSI_BT_CMD_SET_LOCAL_COD              uSetLocalCOD;
 RSI_BT_CMD_QUERY_RSSI                 uQryRssi;
 RSI_BT_CMD_QUERY_LINK_QUALITY         uQryLinkQuality;
 RSI_BT_CMD_ANTENNA_SELECT             uAntennaSelect;
 RSI_BLE_CMD_ADVERTISE                 uLeAdvertise;
 RSI_BLE_CMD_SCAN                      uLeScan;
 RSI_BLE_CMD_CONNECT                   uLeConnect;
 RSI_BLE_CMD_DISCONNECT                uLeDisConnect;
 RSI_BLE_CMD_ENCRYPTION                uLeSmpEncrypt;
 RSI_BLE_CMD_SMP_PAIR_REQUEST          uLeSmpReq;
 RSI_BLE_CMD_SMP_RESPONSE              uLeSmpResp;
 RSI_BLE_CMD_SMP_PASSKEY               uLeSmppasskey;
 RSI_BLE_CMD_SET_PING_TIMEOUT          uLeSetPingTimeout;
 RSI_BLE_CMD_SET_ADVERTISE_DATA        uLeSetAdvertiseData;
 RSI_BLE_CMD_QUERY_PROFILE             uLeSev;
 RSI_BLE_CMD_PROFILE_LIST              uLeAllServ;
 RSI_BLE_CMD_QUERY_CHARACTERISTIC_SERVICES  uLeCharServ;
 RSI_BLE_CMD_QUERY_INCLUDE_SERVICES    uLeIncServ;
 RSI_BLE_CMD_READ_CHAR_VALUE_BY_UUID   uLeCharVal;
 RSI_BLE_CMD_QUERY_ATT_DESC            uLeAttDesc;
 RSI_BLE_CMD_QUERY_ATT_VALUE           uLeAttVal;
 RSI_BLE_CMD_QUERY_MULTIPLE_ATT_VALUES uLeMulAttVals;
 RSI_BLE_CMD_QUERY_LONG_ATT_VALUE      uLeLongAttVal;
 RSI_BLE_CMD_SET_ATT_VALUE             uLeSetAttVal;
 RSI_BLE_CMD_SET_ATT_VALUE_NO_ACK      uLeSetCmdAttVal;
 RSI_BLE_CMD_SET_LONG_ATT_VALUE        uLeSetLongAttVal;
 RSI_BLE_CMD_SET_PREPARE_LONG_ATT_VALUE uLePrepareAttVal;
 RSI_BLE_CMD_EXECUTE_LONG_ATT_VALUE_WRITE uLeExecuteWrite;
```

```
  RSI_BLE_CMD_ADD_GATT_SERVICE          uLeAddService;
  RSI_BLE_CMD_ADD_GATT_ATTRIBUTE        uLeAddAttribute;
  RSI_BLE_CMD_SET_LOCAL_ATT_VALUE       uLeSetLocalAttVal;
  RSI_BLE_CMD_GET_LOCAL_ATT_VALUE       uLeGetLocalAttVal;
  RSI_BLE_CMD_SET_RANDOM_ADDRESS        uLeRandAdd;
}RSI_BLE_API;
```

4 Appendix A: Sample flow

4.1 Configure BLE device in Central Mode

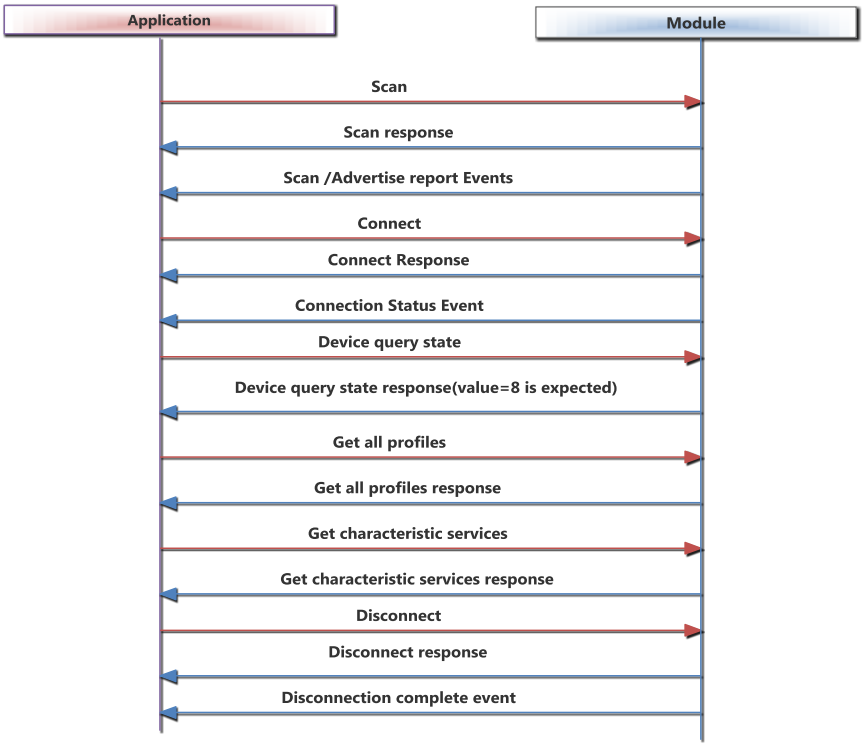


Figure 4 Sample command sequence of BLE Central Mode

4.2 Configure BLE device in Peripheral Mode

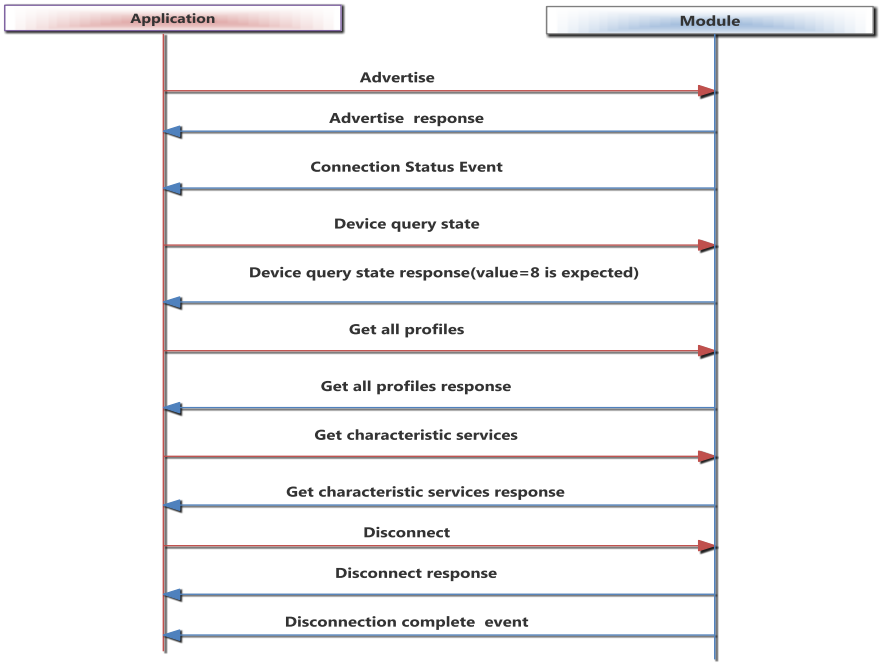


Figure 5 Sample command sequence of BLE Peripheral Mode

4.3 Configure BLE device in Central Mode to connect to multiple slaves

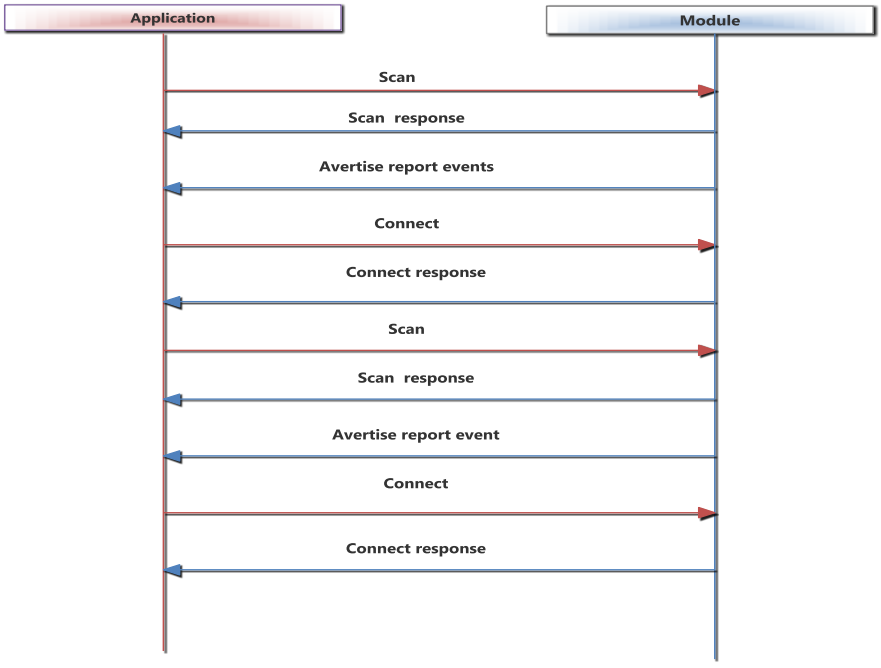


Figure 6 Sample command sequence of BLE multiple slaves

4.4 Configure BLE device to act as both Central and Peripheral simultaneously(Dual Role)

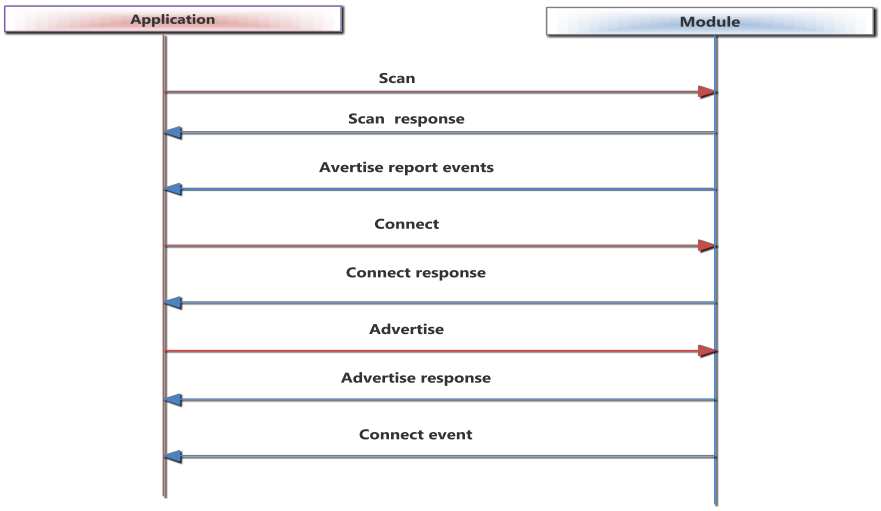


Figure 7 Sample command sequence of BLE dual role

4.5 AT command flow to configure BLE device in Peripheral Mode

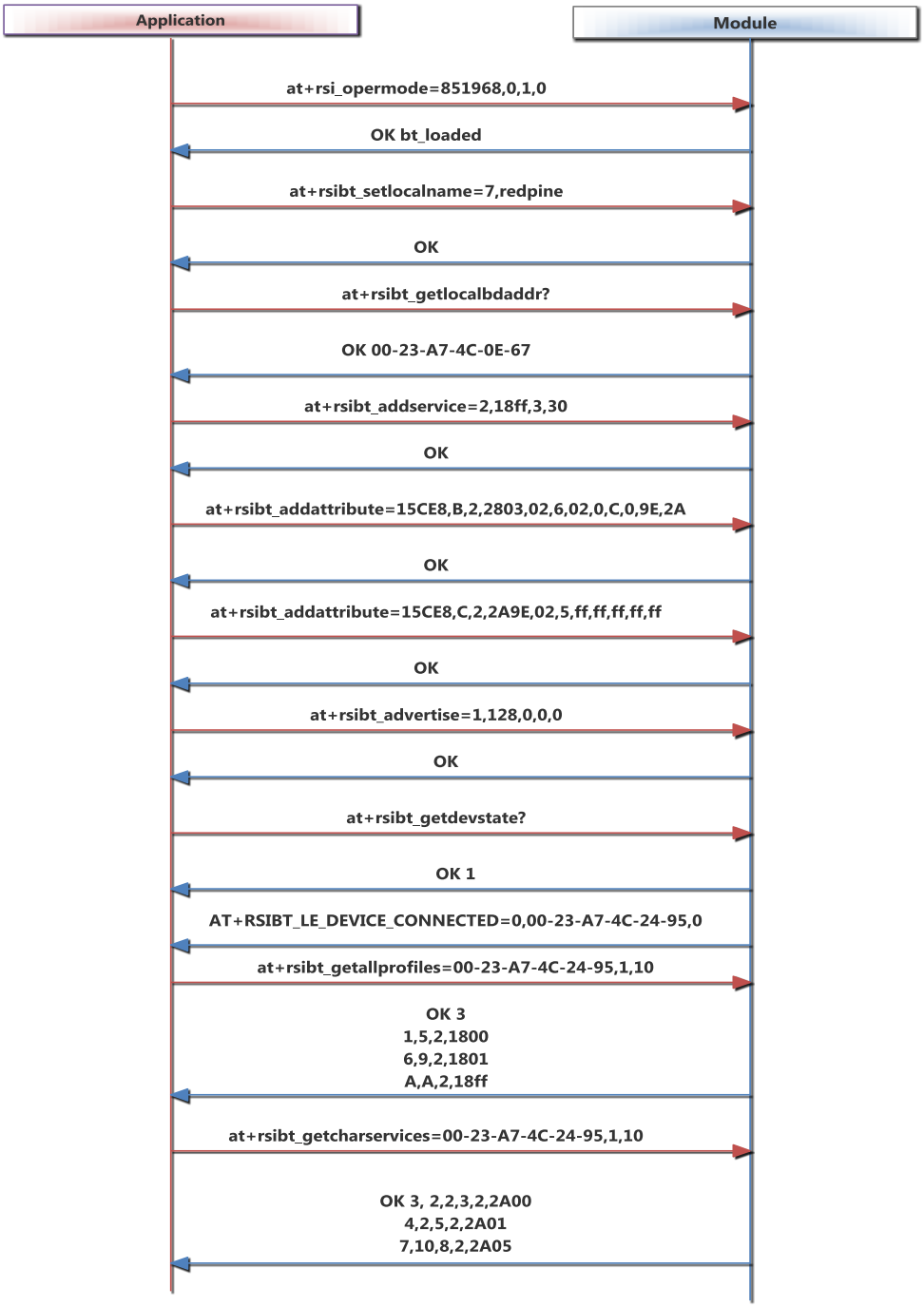


Figure 8 Sample AT command sequence of BLE Peripheral Mode

4.6 AT command flow to configure BLE device in Central Mode

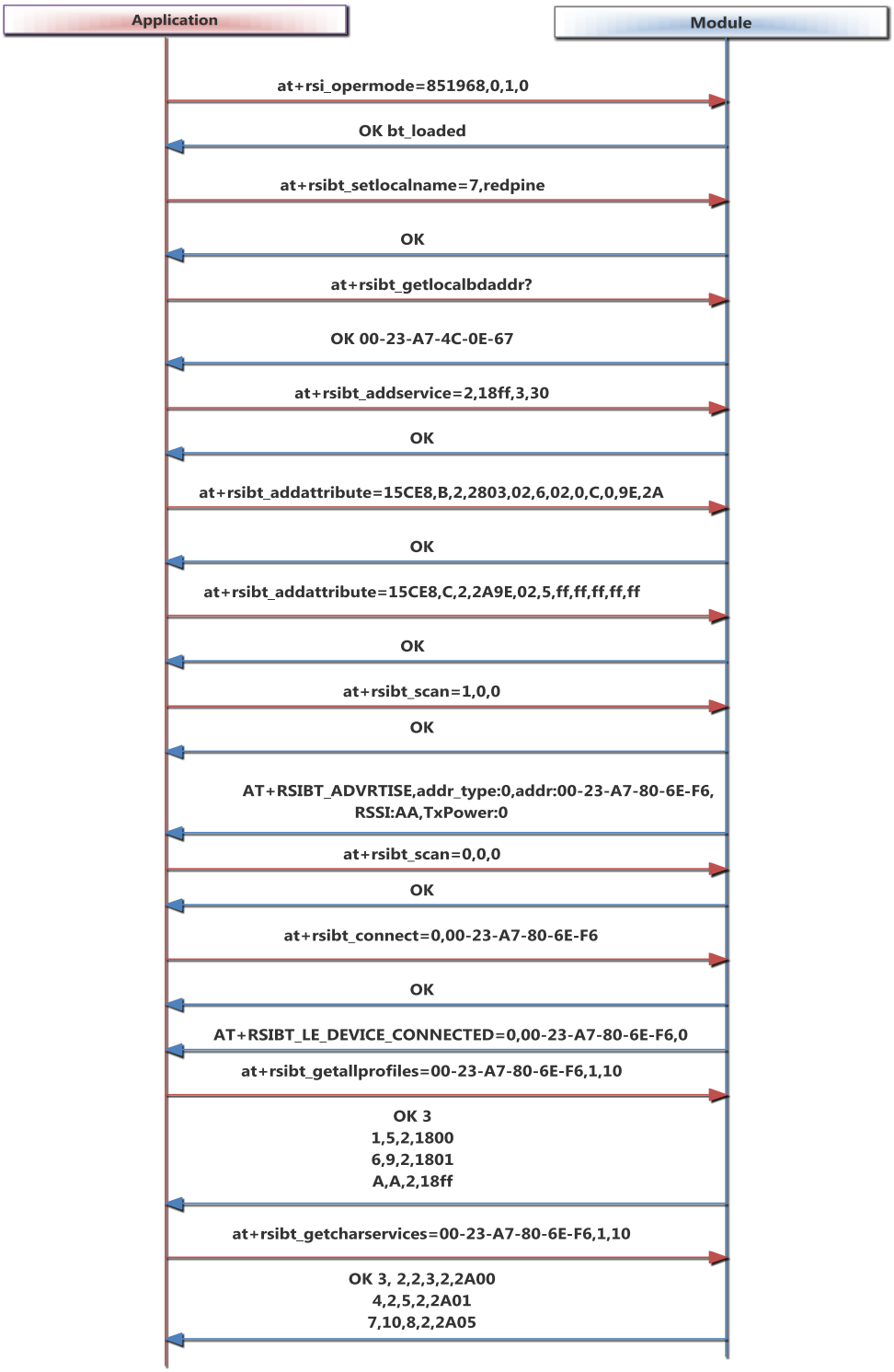


Figure 9 Sample AT command sequence of BLE Central Mode

4.7 Security Management Protocol(SMP) in Slave mode.

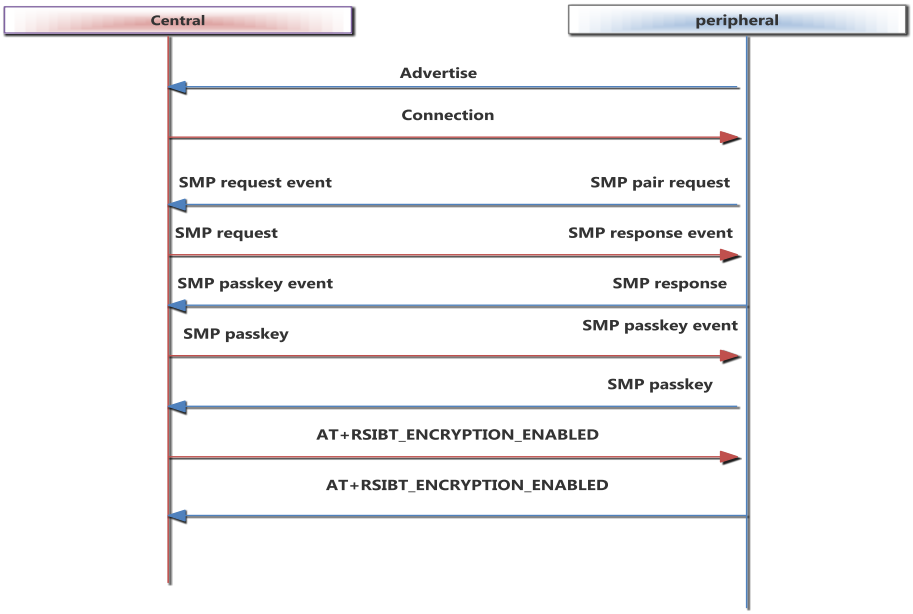


Figure 9 SMP command sequence of BLE Slave (peripheral) Mode

4.8 Security Management Protocol(SMP) in Master mode.

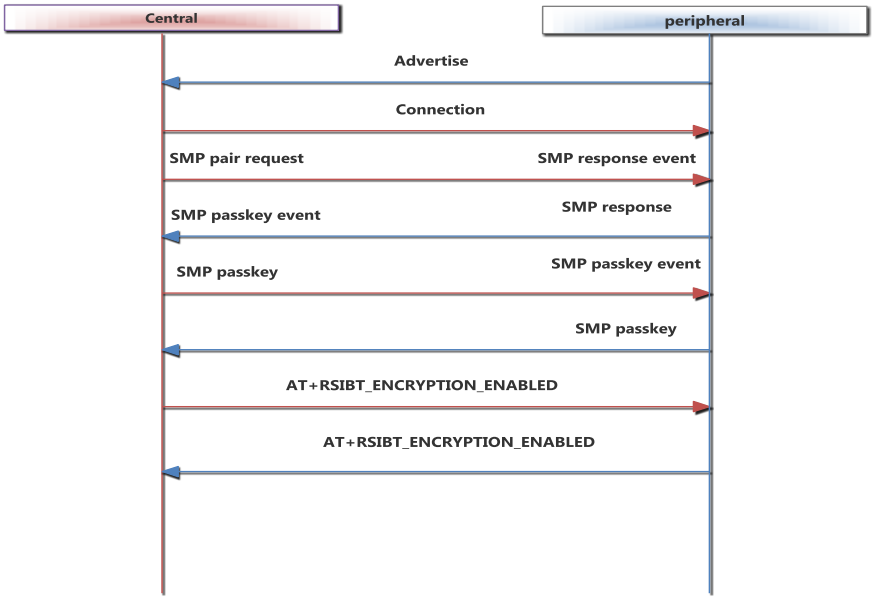


Figure 10 SMP command sequence of BLE Master (Central) Mode

5 Appendix B: Sample flow of APIs for WiFi+BT LE Co-ex mode

Inorder to run the Wi-Fi client and BT-LE coexistence mode, user has to issue the operating mode as first command with coex parameters.

After operating mode command module will operate in both WiFi STA mode and BT LE mode. So user can issue WiFi commands as well as BT LE commands in parallel on host interface.

Common Command:

Set the operating mode command with below parameters to run in Wi-Fi + BT-LE Coex Mode.

Oper_mode = ((wifi_oper_mode) | (coex_mode << 16))

Wifi_oper_mode = 0 (to operate wifi in STA mode)

Coex_mode=13(to operate in WiFi+BT LE coex mode)

Feature_bit_map = 1(to operate WiFi in open security mode)

Tcp_ip_feature_bit_map = 1 (TCP/IP Bypass mode)

Custom_feature_bit_map=(1<<31)

ext_custom_feature_bit_map=(1<<31)

bt_custom_feature_bit_map

To configure the number of GATT Records the following parameter is required;

bt_custom_feature_bit_map i.e, 6th parameter.

Bt_custom_feature_bit_map is valid when the 31st bit of ext_custom_feature_bit_map is set.

Wi-Fi Command Sequence to Associate with Access Point:

- Band :- This command sets the operating mode of the module
- Init :- This command initializes the module
- Scan :- This command scans for Aps and reports the Aps found
- Join :- This command associates the module to the AP

Please refer RS9113-Wiseconnect-Software-PRM-vx.x.x.pdf for Wi-Fi commands description.

BT LE Command Sequence:

- Scan :- This command scans for BT LE devices and reports the devices found
- Connect :- This command associates the module to the remote device

After successful WiFi and BT LE connection user can send WiFi raw data packets into air and also can issue GATT commands

WiFi+BT LE coex Rx flow:

Upon reception of response from module to host, host has to check whether it is Wi-Fi or BT-Le response based on word0[15:12] in frame descriptor.

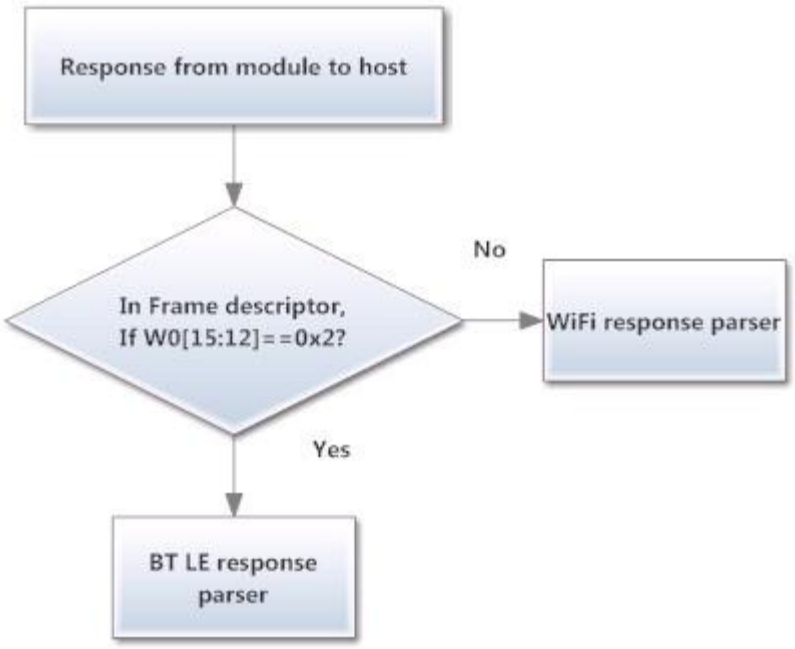


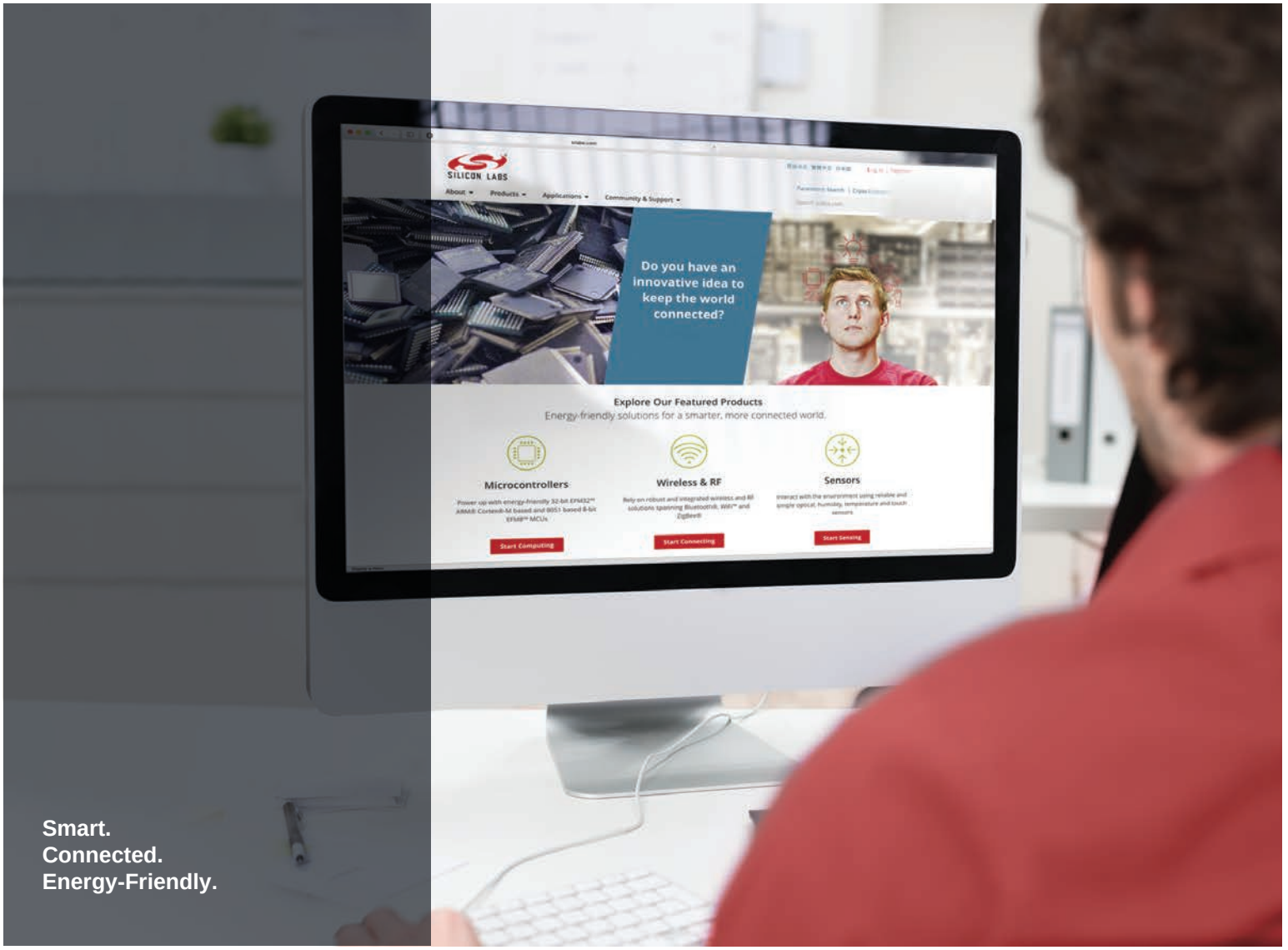
Figure 10 Sample flow for WiFi + BT LE response

Revision History

Revision No.	Version No.	Date	Changes
1	1.0	Sep 2014	Added Bluetooth LE Section
2	1.1	Oct 2014	Added init & deinit commands
3	1.1.0	Jan 2015	Added GATT Commands, added error base 0x4E00 for HCI error list, chaged ATT error base.
4	1.3.0	June 2015	added GATT commands (set and get local attribute values)
5	1.3.0	July 2015	added AT commands format
6	1.3.1	Aug 2015	Added at command format of Init, deinit and antenna select functions. Change the expected device state in 4.1 and 4.2 sections
7	1.4.0	Nov 2015	Added LE ping sections. Updated connection params in binary.
8	1.4.1	Dec 2015	Added AT commands sequence in Appendix. Modified advertise type section. Added example sequences for dual role and mutilple slaves. Corrected LE Ping timeout event. Removed the instances of Query link quality, Set local COD and Get local COD, as they are not applicable for LE.
9	1.5.0	March 2016	Added some parameters in BLE Adevertise and scan. Added BLE Set Random Address Command. Added LE Encrypt Command.
10	1.5.0	May 2016	Explained the parameters of Connect Command. Corrected the scan Parameters. Added SMP flow chart in master & Slave mode. Disconnect event At command format is changed as LE_DISCONNECTED.
11	1.6.0	May 2016	Added Opermode Parameters explanation in Appendix-B
12	1.6.0	June 2016	Added Set Antenna Tx power level command

Revision No.	Version No.	Date	Changes
13	1.6.0	August 2016	Added interfaces, Opermode sections and modified advertisedata command descriptions
14	1.6.0	September 2016	Modified Set antenna Tx power level
15	1.6.2	October 2016	Added GATT read feature Added Scan response data feature Added power save related information
16	1.6.5	Febraury 2017	Added LE whitelist command Added SMP passkey display event
17	1.6.5	March 2017	Added Read Response command
18	1.7.2	May 2018	Added read-request event
19	1.7.3	July 2018	Added LE SMP Reject Response command
20	1.7.6	May 2019	Updated the baud rate 921600 related information
21	1.7.9	May 2020	None

****



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com