RS9113 WiSeConnect®

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 Wi-Fi. This document should be used by the developer to write software on Host MCU to control and operate the module.

Table of Contents

List of Figures

Table of Tables

1 Overview

This document describes the commands to operate RS9113-WiSeConnect Module Family in Wi-Fi. 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.

Section AT Command Mode describes commands to operate the module using the UART/USB-CDC interfaces. Section Binary Command Mode describes Binary commands to operate the module using the SPI/USB/UART/USB-CDC interfaces.

Note:

Please ensure that the module connections are made as described in MIG, for your chosen host interface.

2 Interfaces

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

- UART
- SPI
- USB

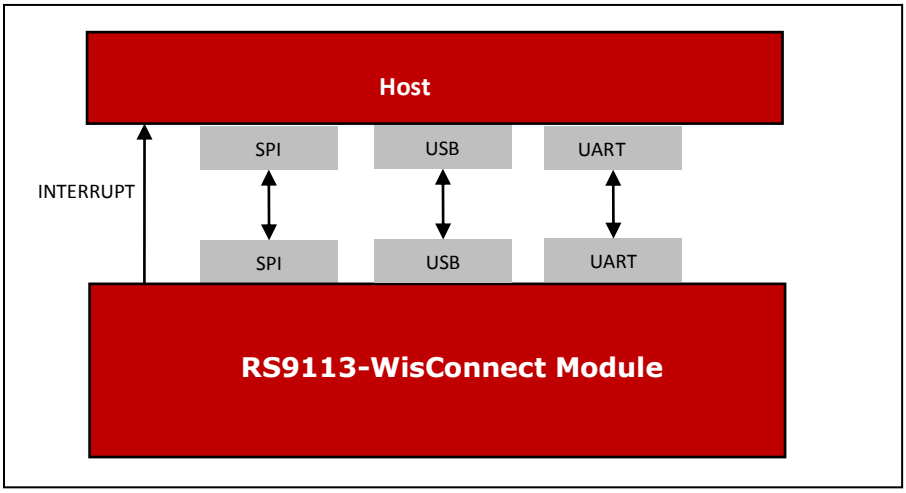


Figure 1: Host Interface Block Diagram

2.1 UART

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

2.1.1 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
  - 921600 bps

Note:

921600 bps is supported only if hardware flow control is enabled.

For BT/BLE, 921600bps rate is not supported.

2.1.2 Default Parameters

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

2.2 USB

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

2.2.1 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 M bits/Sec in addition to 1.5Mbits/Sec and 12 M bits/Sec.
- Supports USB-CDC

2.3 SPI

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

2.3.1 Features

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

2.3.2 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 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.

2.3.4 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.

Note:

If the host does not support active high level triggered and has used rising edge triggered there are chances that the interrupt is missed, or a false interrupt is seen. Host can check the interrupt status register if the module has raised an interrupt.

We have provided a code snippet for that to be added. Please check SAPI examples.

2.3.5 SPI Commands

The SPI interface is programmed to perform a certain transfer using commands C1, C2, C3 and C4 and optional 32-bit address. For all the Commands and Addresses, the Host is configured to transmit data with **8-bit mode**. At the end of all the Commands and Addresses, the Host is reconfigured to transmit data with 8-bit or 32-bit mode depending on the commands issued. The Module responds to all the commands with a certain response pattern.

The four commands C1, C2, C3, and C4 indicate to the SPI interface all the aspects of the transfer.

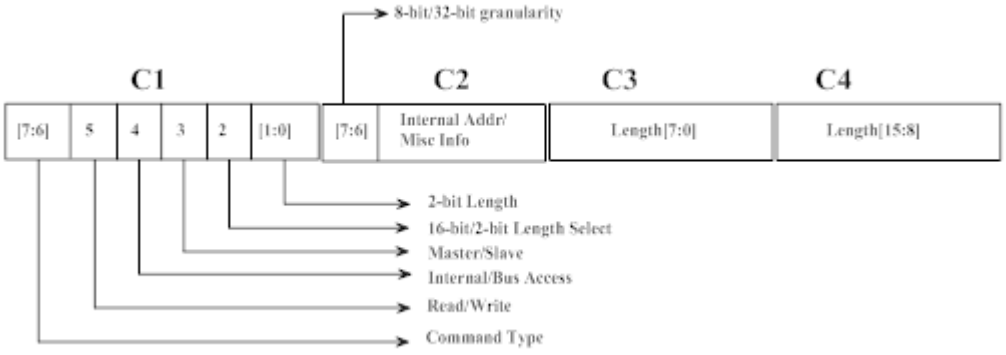


Figure 2: SPI Command Description

The command description is as follows:

Command	Bit Number	Description
C1	[7:6]	Command Type “00”- Initialization Command “01”- Read/Write Command “10”, “11”- Reserved for future use
	5	Read/Write ‘0’- Read Command ‘1’- Write Command
	4	Register/ (Memory & Frame) read/write Access ‘0’- register read/write ‘1’- Memory read/write or Frame read/write
	3	Memory/Frame Access ‘0’- Memory read/write ‘1’- Frame read/write
	2	2-bit or 16-bit length for the transfer ‘0’- 2-bit length for the transfer ‘1’- 16-bit length for the transfer
	1:0	2-bit length (in terms of bytes) for the transfer (valid only if bit 2 is cleared) “00”- 4 Bytes length “01”- 1 Byte length “10”- 2 Bytes length “11”- 3 Bytes length
C2	7:6	8-bit or 32-bit mode. Indicates the granularity of the write/read data. Note: The SPI (C1, C2, C3, C4) commands and addresses (A1, A2, A3, A4) will always be 8-bit irrespective of this value. “00”- 8-bit mode “01”- 32-bit mode “10”, “11”- Reserved for future use
	5:0	This carries Register address if bit 4 for Command C1 is cleared (i.e. register read/write

Command	Bit Number	Description
		selected). Otherwise, reserved for future use.
C3	7:0	Length (7:0) LSB of the transfer's length (which is in terms of bytes) in case bit 2 of C1 is set. This command is skipped if bit 2 of C1 is cleared.
C4	7:0	MSB of the transfer Length (15:8) (Which is in terms of bytes) in case bit 2 of C1 is set. This command is skipped if bit 2 of C1 is cleared i.e. if 2-bit length is selected.

Table 1: SPI Command Description

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

2.3.5.1 Module Response

The RS9113 WiSeConnect module gives responses to the host SPI command requests through SPI interface. These are as follows

- A success/failure response at the end of receiving the command. This response is driven with 8-bit mode during the Command and Address phase and is then switched to 8-bit or 32-bit mode during the Data phase as per the command issued.
  - Success: 0x58 or0x00000058
  - Failure: 0x52 or 0x00000052
- An 8-bit or 32-bit start token is transmitted once the four commands (C1, C2, C3, C4) indicating a read request are received and the Module is ready to transmit data. The start token is immediately followed by the read-data.
  - Start Token: 0x55 or 0x00000055
- An 8-bit or 32-bit busy response in case a new transaction is initiated while the previous transaction is still pending from the slave side.
  - Busy Response: 0x54 or 0x00000054

2.3.5.2 Module Bit Ordering of SPI Transmission/Reception

8-bit Mode:

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

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

B0 is sent first, then B1, then B2 and so on.

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

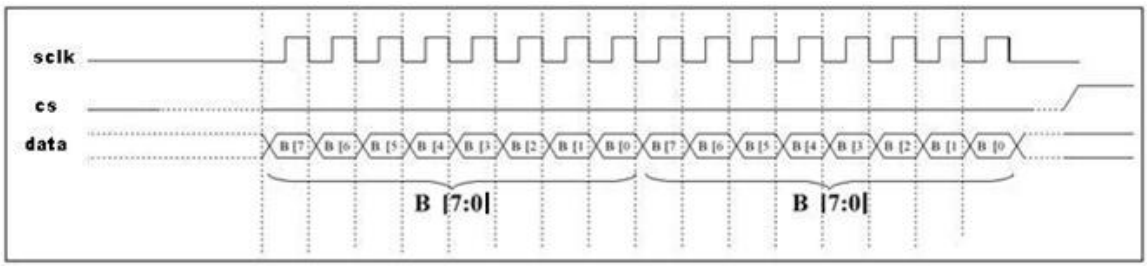


Figure 3: 8-bit Mode

32-bit Mode:

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

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

W0 is sent first, then W1, then W2 and so on.

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

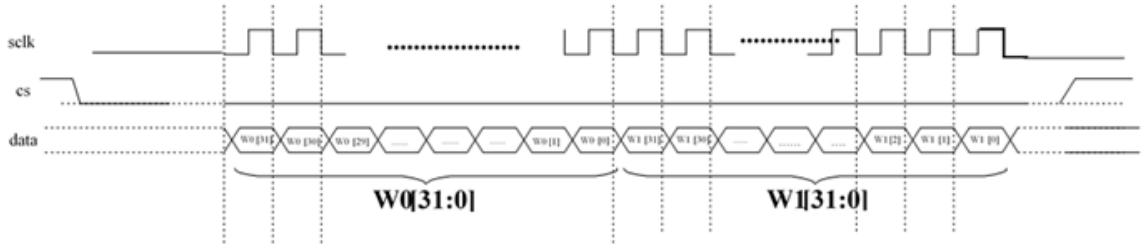


Figure 4: 32-bit Mode

Bit Ordering of Module Response

The bit ordering is same as explained in Module bit Ordering of SPI Transmission/Reception. For example, 0x58 response for 8-bit success is sent as

0 -> 1 -> 0 -> 1-> 1 -> 0 -> 0 -> 0 . That is 0 is sent first, then 1, then 0, then 1, and so on.

2.3.6 Module SPI Interface Initialization

The Initialization Command is given to the Module to initialize the SPI interface. The SPI interface remains non-functional to any command before initialization and responds only after successful initialization. Initialization should be done only once after the power-on. Module treats any subsequent initialization commands before the reset as errors. For the

initialization command, the Host drives C1 command followed by an 8-bit dummy data. Bits [7:6] of C1 are cleared and 0x15 is driven on bits [5:0]. Status response from the SPI Interface is driven during the transmission of the dummy data i.e. after the transfer of 8-bits of command C1.

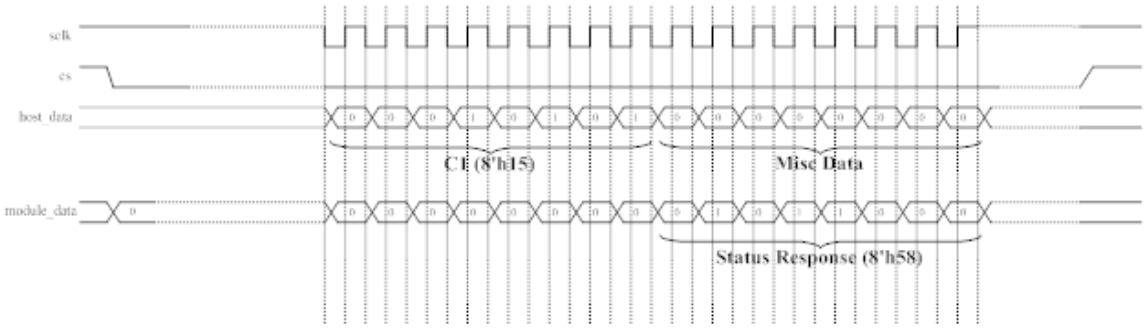


Figure 5: Module SPI Initialization

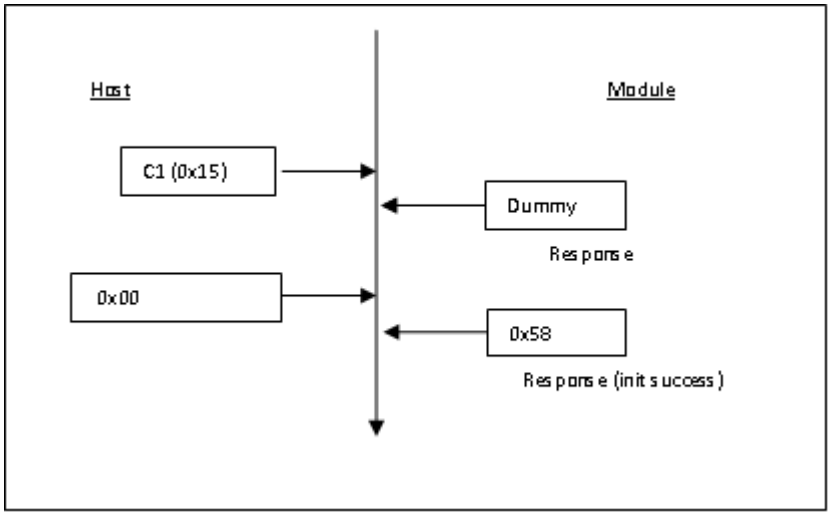


Figure 6: SPI Initialization exchanges between Host and Module

2.3.7 Host Interactions Using SPI Command

This section describes the procedures to be followed by the Host to interact with the RS9113-WiSeConnect Module using SPI commands.

The Host interactions to the module could be categorized as below.

Command	Command Description
Memory write	Memory write commands are used to write the data to specified memory/Register address in the module.
Memory read	Memory read commands are used to read the data from specified Memory/Register address in the module.

Command	Command Description
Frame write	Frame write commands are used to send the data in frame format to the module. All management/data frames are sent to module using Frame write commands.
Frame read	Frame read commands are used to read data in frame format from the module. All management responses/data frames are read from module using Frame read commands.
Register write	Register write commands are used to write content to specified register in the module.
Register read	Register read commands are used to read content from specified register in the module.

Table 2: Command Types

2.3.7.1 Memory Type

Host need to accesses the memory/registers of the RS9113-WiSeConnect Module for configuration and operation.

To write data into a memory/register address, memory write command must be framed as described in the figure below. If a “busy” or “failure” response is sent from the Module, the retries need to be done or the module should be reset.

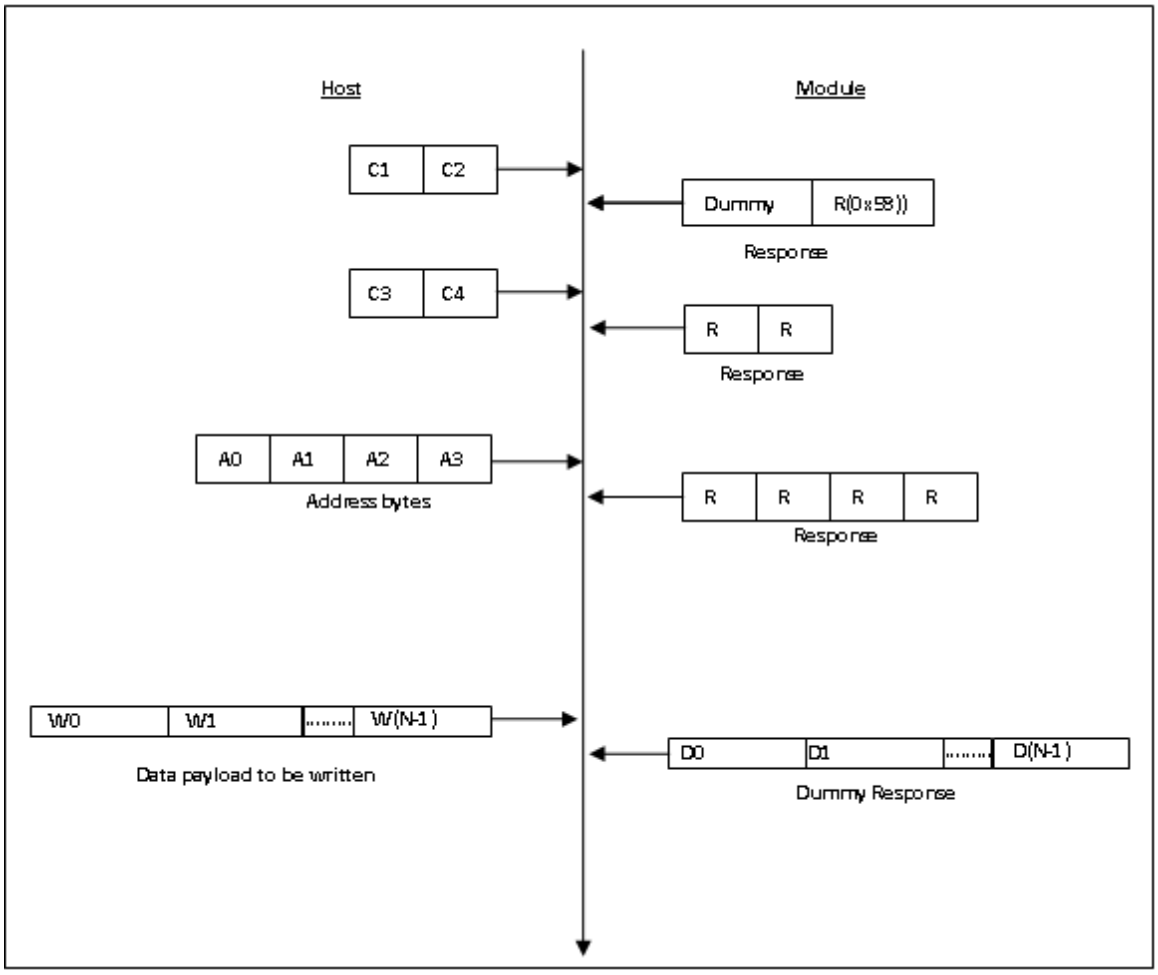


Figure 7: Memory Write

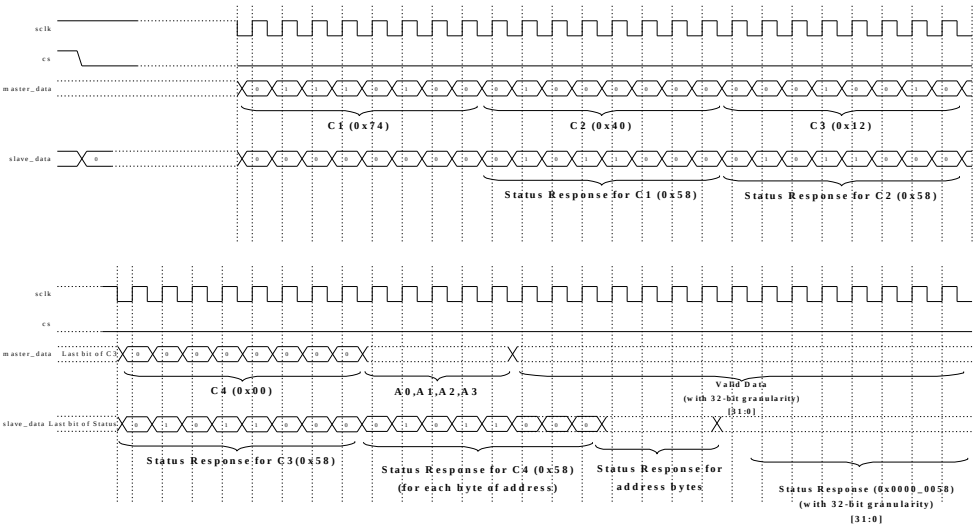


Figure 8: Interactions in the physical interface

The above figure's structure is similar for all following read/write operations)

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

- Send the commands C1, C2.
- Read the response (R) from the Module. Response will as described in RS9113-WiSeConnect module Module Response. Status 0x58 indicates that the Module is ready.
- Host should send the commands C3 and C4, followed by the 4 byte address (corresponding to the memory/register address) and data. Status 0x54 indicates that the device is busy. Host has to retry. Status 0x52 indicates a failure response.

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

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

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

Original data <W0[7:0]><W1[7:0]><W2[7:0]><W3[7:0]><W4[7:0]>

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

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

2.3.7.2 Frame Write

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

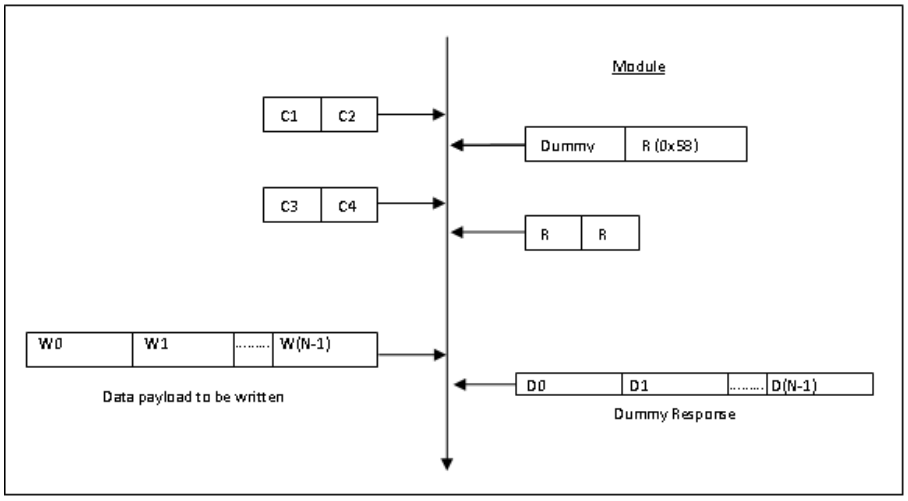


Figure 9: Frame Write

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

1. Prepare and send the commands C1, C2 together as described for Frame write.
2. Read the response (R) from the Module (RS9113-WiSeConnect Module).
3. Status 0x58 indicates that the Module is ready. Host can send the commands C3 and C4, followed by the data. Status 0x54 indicates that the device is busy. Host has to retry. Status equal to 0x52 indicates a failure response.

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

Original data <W4[7:0]><W3[7:0]><W2[7:0]><W1[7:0]><W0[7:0]>

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

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

2.3.7.3 Memory Read

To read data from a memory/register address, Memory Read command has to be formed.

The following figure gives the flow for memory reads between the Host and the Module.

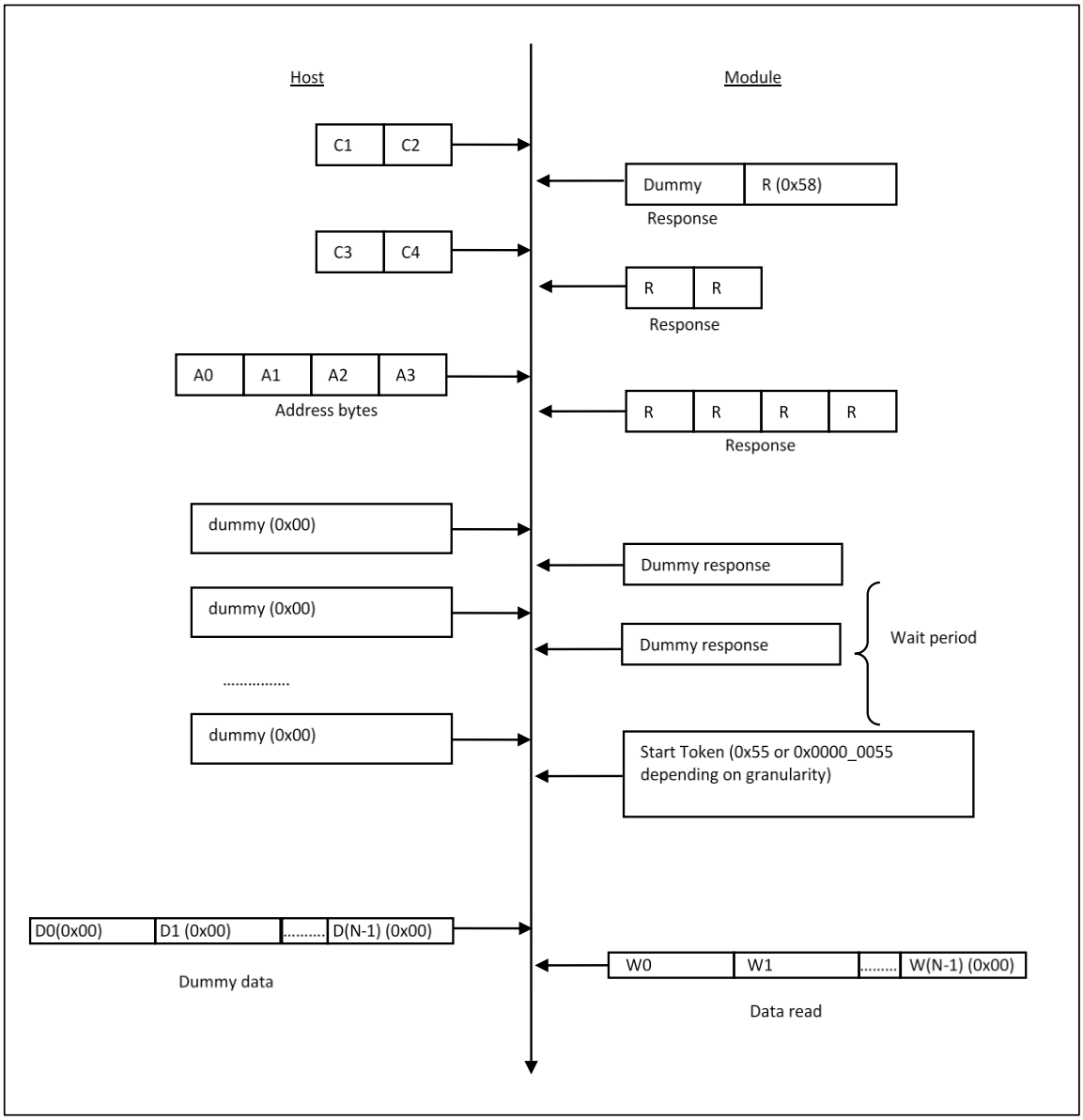


Figure 10: Memory Read

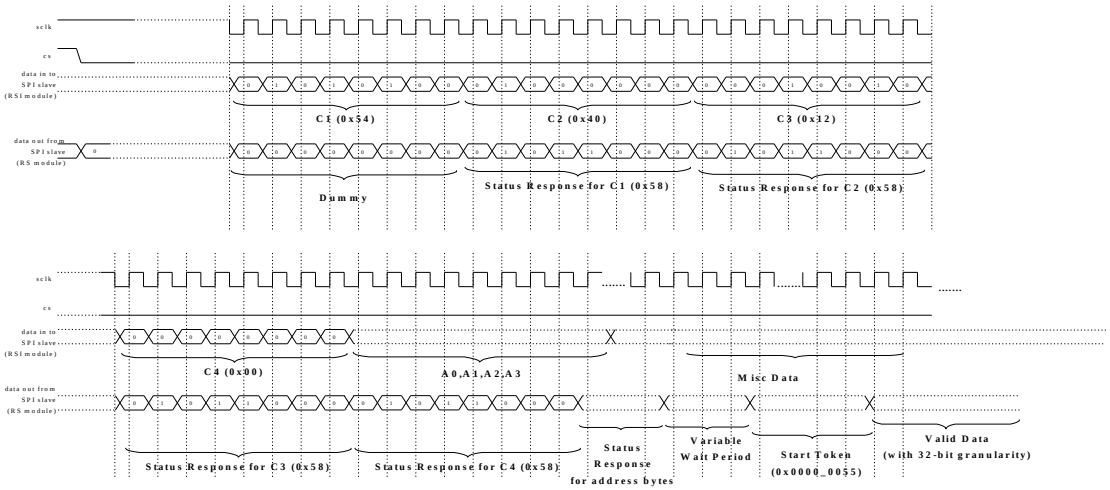


Figure 11: Memory Read at Physical Interface

The following is the procedure to be followed by the Host to do a memory read.

1. Prepare and send the commands C1, C2 as described for Memory read.
2. Read the response from the RS9113-WiSeConnect Module.
3. Status 0x58 indicates that the slave is ready. Host should next send the commands C3, C4, which indicate the length of the data to be read, followed by the address of the memory location to be read.
4. After sending/receiving C3, C4 commands/response and the addresses, the Host should wait for a start token (0x55). Host writes a stream of dummy bytes to read the data from the module each time. The Host should read this data until the start token is received. The data following the start token should be interpreted as the frame of specified length that is read from the Module.
5. Status 0x54, after C1 and C2 bytes, indicates that the device is busy. Host has to retry.
6. Status 0x52, after C1 and C2 bytes, indicates a failure response from the Module.

There is a variable wait period, during which dummy data is sent. After this period, a start token is transmitted from the Module to the Host. The start token is followed by valid data. The start token indicates to the Host the beginning of valid data. To read out the valid data, dummy data [D0, D1, D2, …. D (N-1)] is sent. N is number of 8bit or 32 bit dummy words (based on mode selected) need to send to receive specified length of valid data from the module.

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

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

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

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

2.3.7.4 Fame Read

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

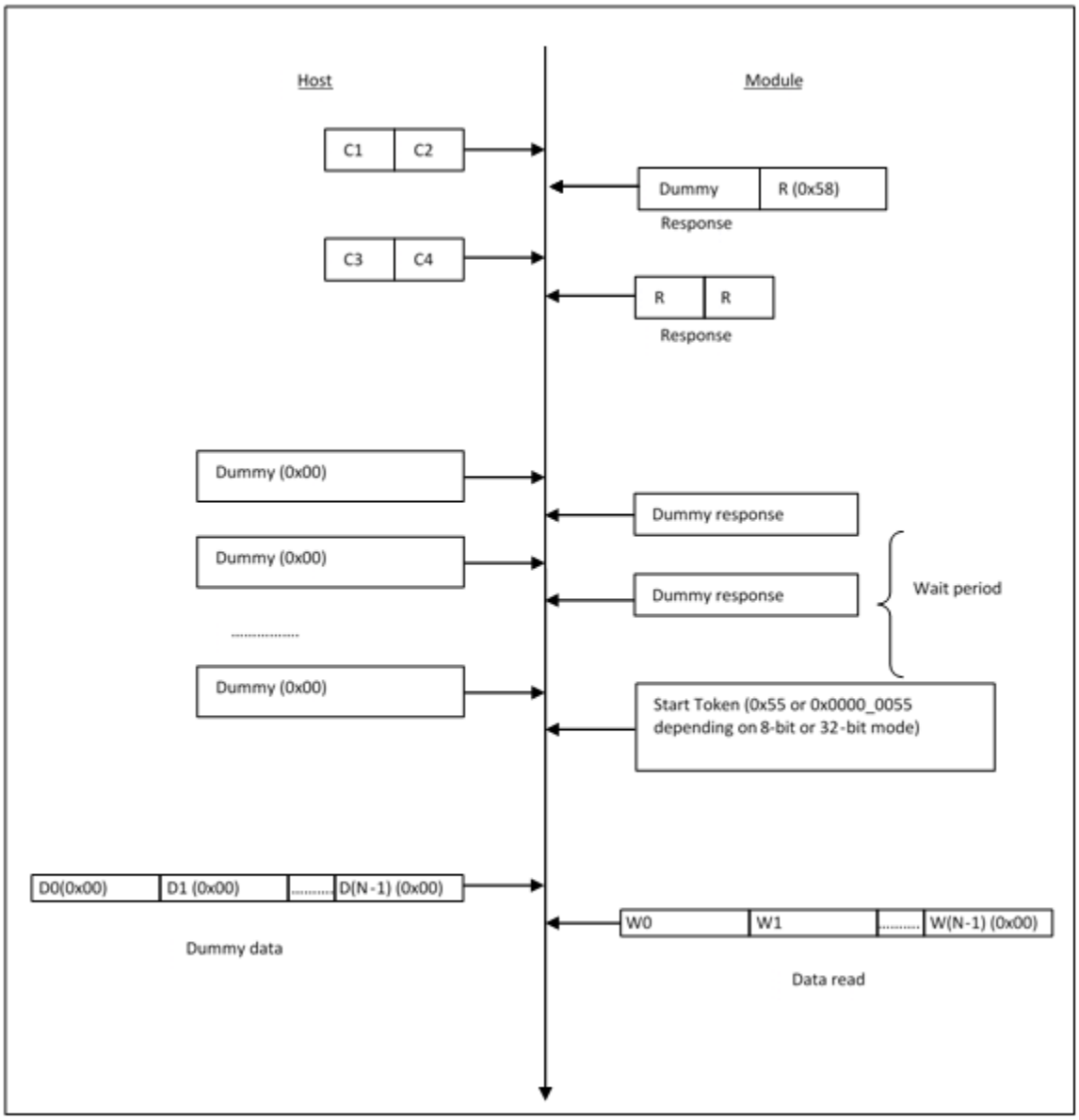


Figure 12: Frame Read

The following is the procedure to be followed by the Host to do a Frame read.

1. Prepare and send the commands C1, C2 as described for Frame Read.

2. Read the response from the RS9113-WiSeConnect Module.
3. Status 0x58 indicates that the Module is ready. Host should send the commands C3, C4 which indicate the length of the data to be read.
4. After sending/receiving C3, C4 commands/response, Host should wait for a start token (0x55). The data then follows after the start token. Host writes a dummy byte to read the data from the module each time. The Host should read this data until the start token is received. The data following the start token should be interpreted as the frame of specified length that is read from the Module. Status 0x54, after C1 and C2 bytes, indicates that the device is busy. Host has to retry. Status 0x52, after C1 and C2 bytes, indicates a failure response from the Module.

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

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

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

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

2.3.7.5 Register Read

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

For Register Reads following sequence need to be followed:

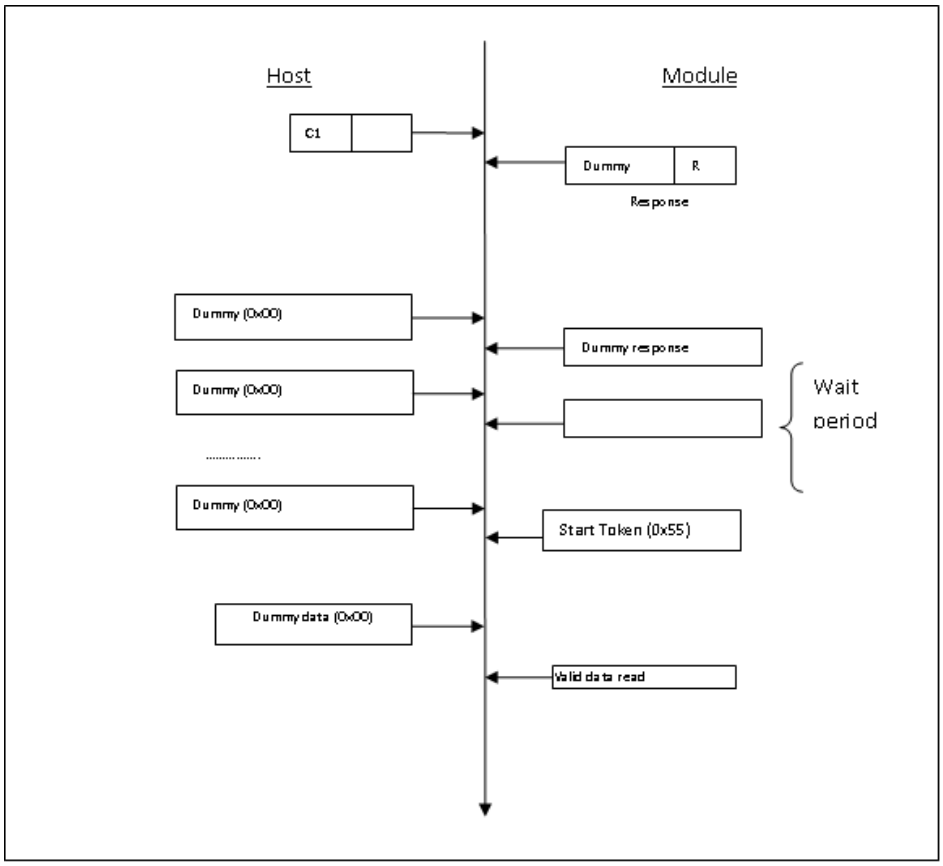


Figure 13: Register Read

2.3.7.6 Register Write

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

For Register writes following sequence need to be followed:

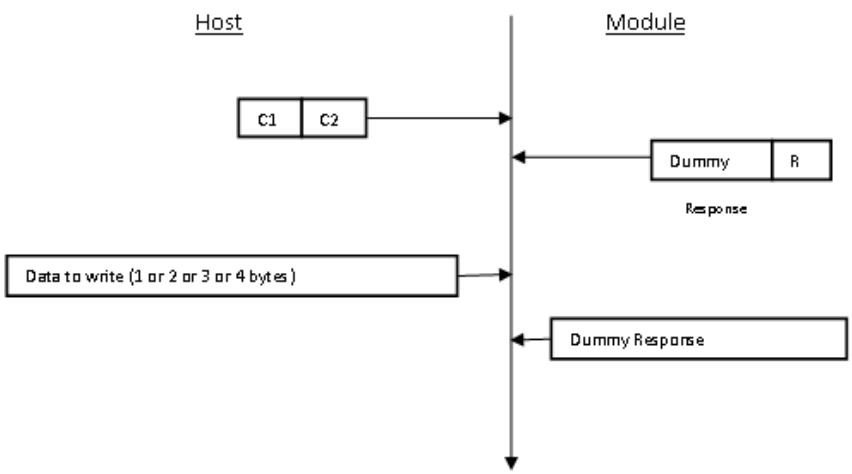


Figure 14: Register Write

2.3.8 Register Summary

Register	Address
SPI_HOST_INTR	0x00

Table 3: RS9113-WiSeConnect Module Register Description

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

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

SPI_HOST_INTR				
Register Address: 0x00				
Bit	**Access**	**Function**	**Default Value**	**Description**
				read by host. Bit 4: Reserved Bit 5: Reserved Bit 6: Reserved

Table 4: SPI Host Interrupt Register

Note:

For SPI Tx/Rx operation please refer to the section SPI Host Interface Mode

3 Module Bootload Process

Module supports two Bootloading modes:

- Host interaction (Non-bypass) mode: In this mode host can interact with the bootloader and can give boot up options (commands) to configure different boot up operations. It tells us to what operations it has to perform based on the selections made by the user.
- Bypass mode: In this mode bootloader interactions are completely bypassed and default firmware image is loaded in the module. This mode is suggested to be used in the final production software to minimize the boot up time.

3.1 Host Interaction Mode

In Host Interaction mode, host interaction varies based on host interface. Host interaction in SPI/USB and UART/USB-CDC are different. In UART & USB-CDC boot up options are menu based and In SPI/USB it is using command exchanges, details are explained in below section.

3.1.1 Host Interaction Mode in UART/USB-CDC

This section explain host interaction mode in UART/USB CDC mode.

3.1.1.1 Startup Operation

On powering up, bootloader checks the validity of the bootup options by computing ones complement checksum. If the checksum fails, it computes the checksum from backup. If checksum passes, it copies the backup to the actual location. If checksum of the backup options also fail, the boot up options are reset. In either of the cases, bootloader bypass is disabled and corresponding error messages are given. Host is required to carry out ABRD(Auto baud rate detection) operation and after successful ABRD, module will give boot up options to upgrade or load firmware or select default image or select bootload bypass mode. Host needs to select the appropriate option. In the case of checksum failure, “LAST CONFIGURATION NOT SAVED” is displayed when the backup checksum passes and “BOOTUP OPTIONS CHECKSUM FAILED” is displayed when the backup checksum fails before displaying the bootup options.

3.1.1.1.1 Hyper Terminal Configuration

RS9113-WiSeConnect Module uses the following UART interface configuration for communication:

Baud Rate: The following baud rates are supported by the module: 9600 bps, 19200 bps, 38400 bps, 57600 bps, 115200 bps, 230400 bps,460800 bps, 921600 bps.

Data bits: 8

Parity: None

Stop bits: 1

Flow control: None

Before the module is powered up, follow sequence of steps as given below:

- Open HyperTerminal and enter any name in the “Name” field then click “OK” button. Here “WiSeConnect” is entered as shown in the figure below.

Note:

Default baud rate of the module is 115200.

Figure 15: HyperTerminal Name field Configuration

- After clicking ”OK” button the following dialog box is displayed as shown in the figure below.

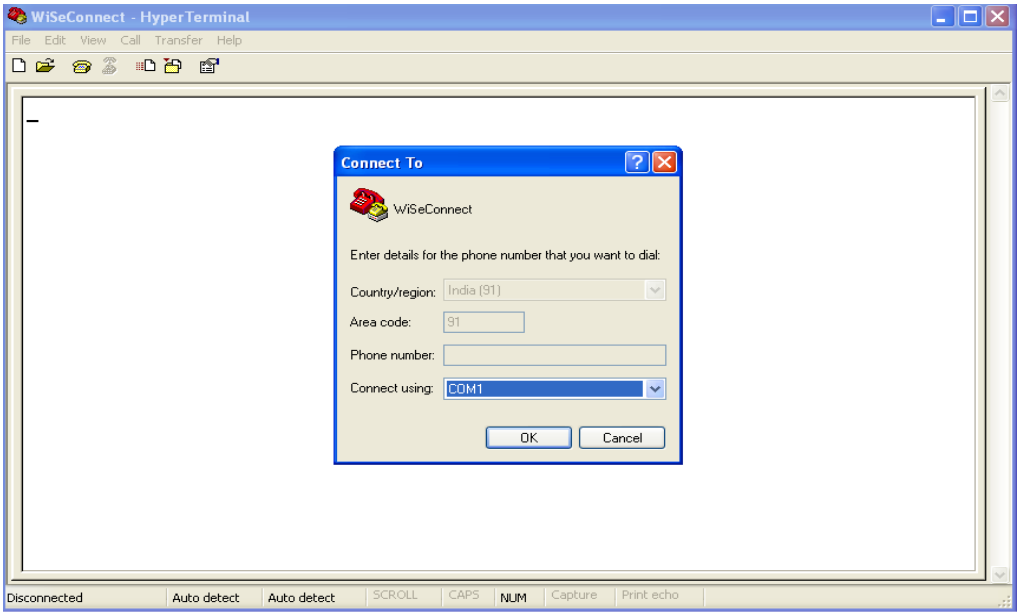


Figure 16: HyperTerminal COM port field Configuration

- In the “Connect using” field select appropriate com port. In the figure above COM1 is selected. Then click “OK” button.
- After clicking ”OK” button the following dialog box is displayed as shown in the figure below.

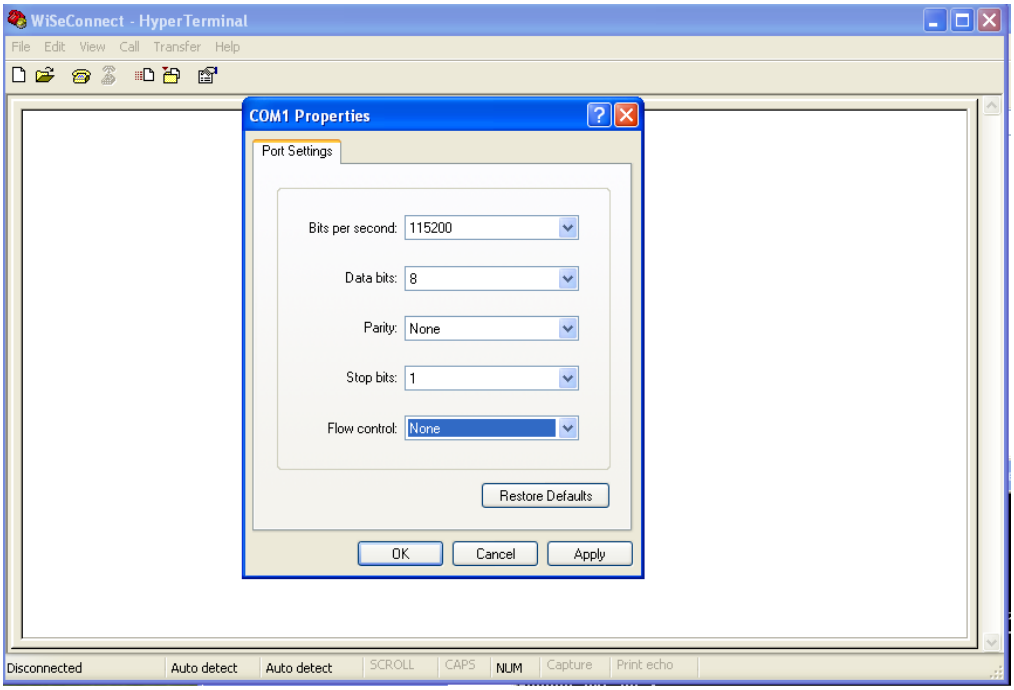


Figure 17: HyperTerminal Baud rate field Configuration

Set the following values for different fields in figure 5 as given below.

- Set baud rate to 115200 in “Bits per second” field.
- Set Data bits to 8 in “Data bits” field.
- Set Parity to none in “Parity” field.
- Set stop bits to 1 in “Stop bits” field.
- Set flow control to none in “Flow control” field.
- Click “OK” button after entering the data in all the fields.

3.1.1.1.2 Auto Baud Rate Detection (ABRD)

The RS9113-WiSeConnect Module automatically detects the baud rate of the Host’s UART interface by exchanging some bytes. The Host should configure the UART interface for the following parameters for ABRD detection.

RS9113-WiSeConnect Module uses the following UART interface configuration for communication:

Baud Rate: The following baud rates are supported: 9600 bps, 19200 bps, 38400 bps, 57600 bps, 115200 bps, 230400 bps,460800 bps, 921600 bps.

Data bits: 8

Stop bits: 1

Parity: None

Flow control: None

The following is the procedure to be followed by the Host for ABRD by the RS9113-WiSeConnect Module.

1. Configure the UART interface of the Host at the desired baud rate.
2. Power on the RS9113-WiSeConnect Module.
3. Host, after releasing the module from reset, should wait for 20 ms for initial boot-up of the module to complete and then transmit 0x1C at the baud rate with which its UART interface is configured. After transmitting '0x1C' to the module, the Host should wait for the module to transmit 0x55 at the same baud rate.
4. If the '0x55' response is not received from the module, the Host has to retransmit 0x1C, after a delay of 200ms.
5. On finally receiving '0x55', the host should transmit '0x55' to the module. The module is now configured with the intended baud rate.

If ABRD process is not triggered/failed from user, module waits for a maximum of 18 seconds and gets configured to default baud rate of 115200 bps.

Note:

Performing ABRD in host interaction mode is must for USB CDC mode.

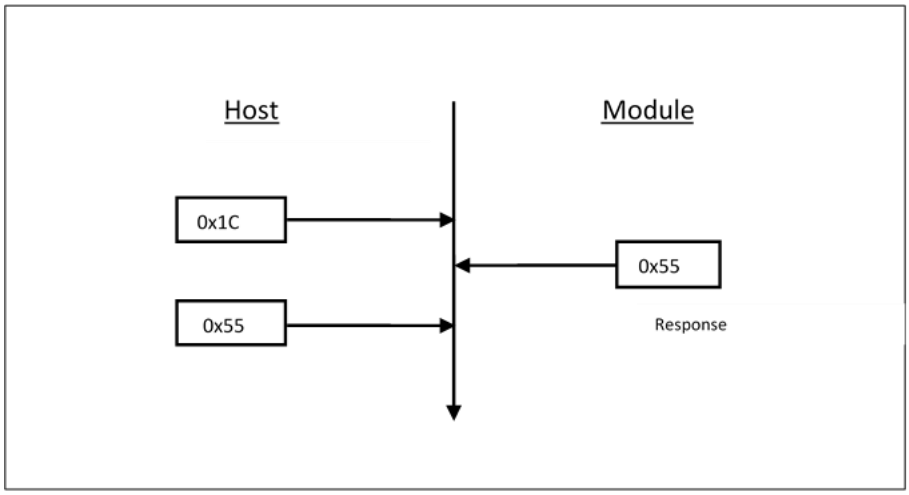


Figure 18: ABRD exchange between Host and module

3.1.1.2 Start Up Messages on Power-Up

On powering up the module and after ABRD you can see a welcome message on host, followed by boot up options as shown below:

Note:

Windows Hyper Terminal is used to demonstrate boot up /up-gradation procedure.

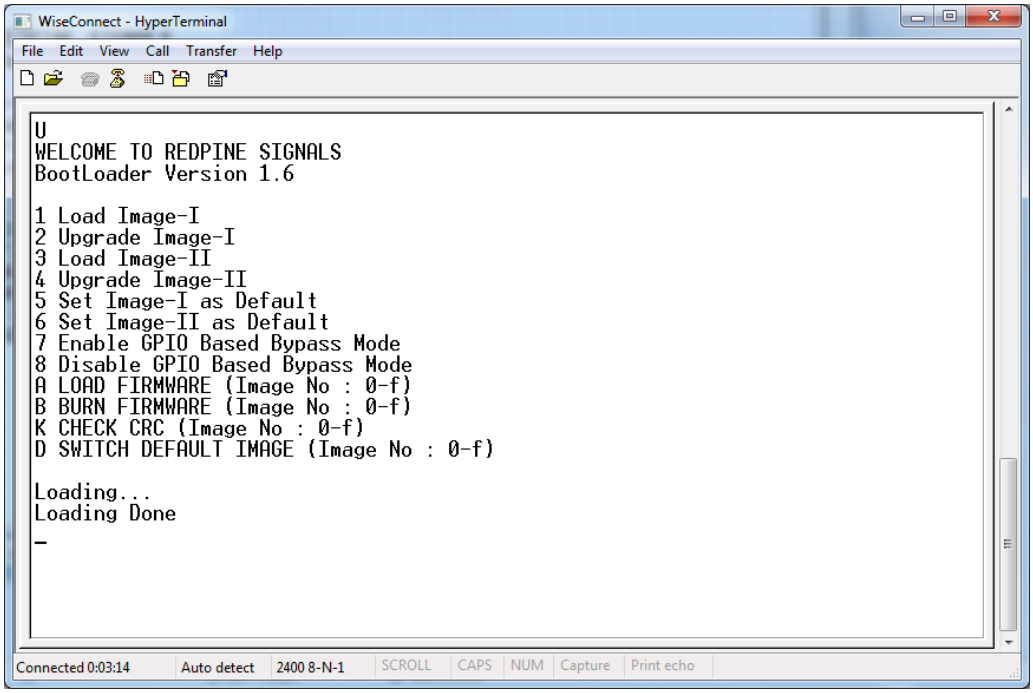


Figure 19: RS9113-WiSeConnect Module UART/USB-CDC Welcome Message

3.1.1.3 Loading the Firmware in the Module

For loading the firmware from flash of module you have to choose Option 1 i.e. “Load Image–I”.

3.1.1.3.1 Load Image – I

After welcome message is displayed as shown in the above figure, select option 1 “Load Image–I ” for loading Image – I.

Note:

1) In order to use host bypass mode user has to select one of the image as default image by selecting options 5 or 6.
2) In Host interaction mode if there is no option selected then selected default image will be loaded after welcome message within 20 seconds.
3) If valid firmware is not present, then a message saying “Valid firmware not present”along with the bootup options will be displayed after ABRD.

Once the loading of the deault firmware is carried out successfully,a message saying “Loading Done” will be displayed on the screen.

If the module interface changes to binary mode, some unrecognizable characters will be displayed.

If this response is viewed in HEX format, it shows 0x89 indicating "CARD READY" condition.

- Loading...
- @‰

The bootloader option "M" can be used for switching from Binary mode to AT mode in order to view the "Loading Done" message.

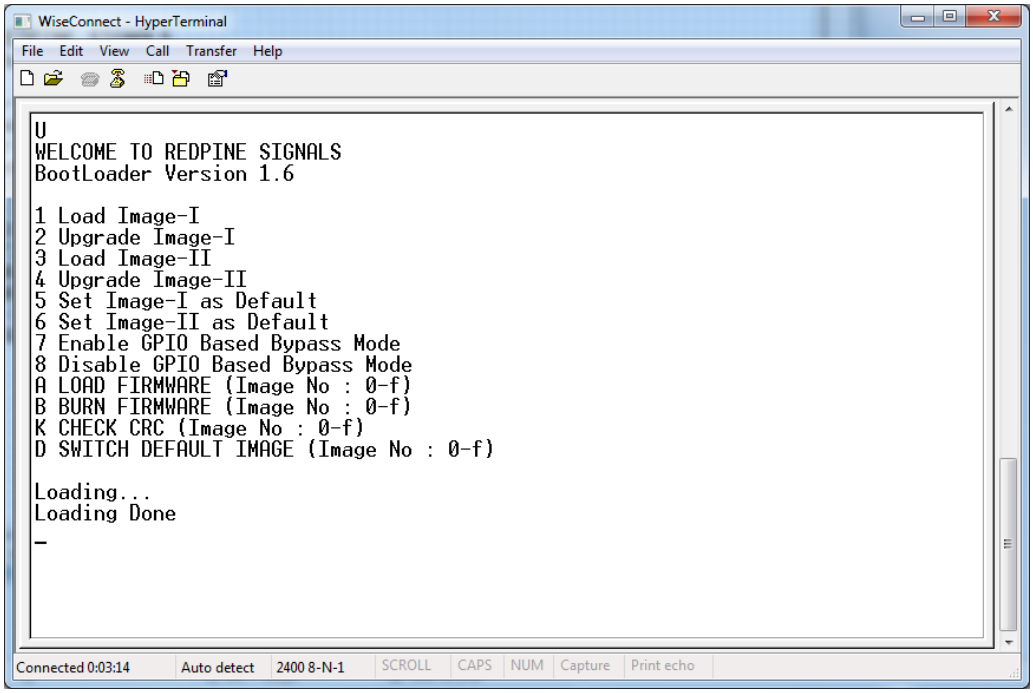


Figure 20: RS9113-WiSeConnect Module UART WLAN Firmware Loading Message

3.1.1.4 Firmware Upgradation

On powering up the module, welcome message is displayed. For the Firmware upgradation option 2 need to be selected for upgrading Image – I.

3.1.1.4.1 Upgrade Image – I

- After welcome message display, select option 2 “Upgrade Image – I ” for upgrading Image –I
- A message “Send RS9113_WC_GENR_x_x_x_wlan.rps” as shown in the figure below is displayed.

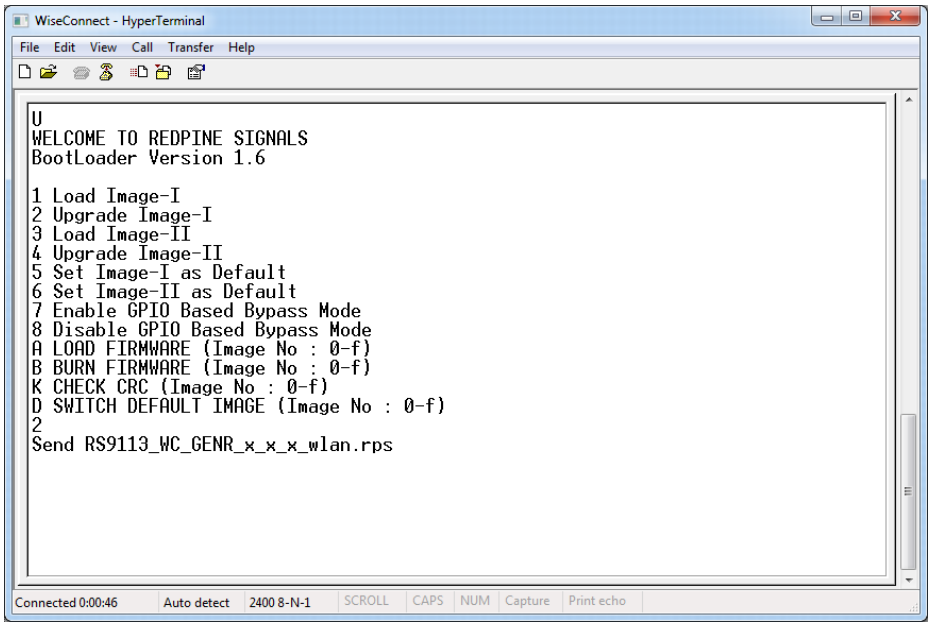


Figure 21: RS9113-WiSeConnect Module Image – I Firmware Upgrade File Prompt Message

- Select from "File" menu of HyperTerminal "send file" option. A dialog box is displayed as shown in the figure below . Browse the path where "RS9113_WC_GENR_x_x_x_wlan.rps" is located and select Kermit as the protocol option. After this Click "Send" button for file transfer.

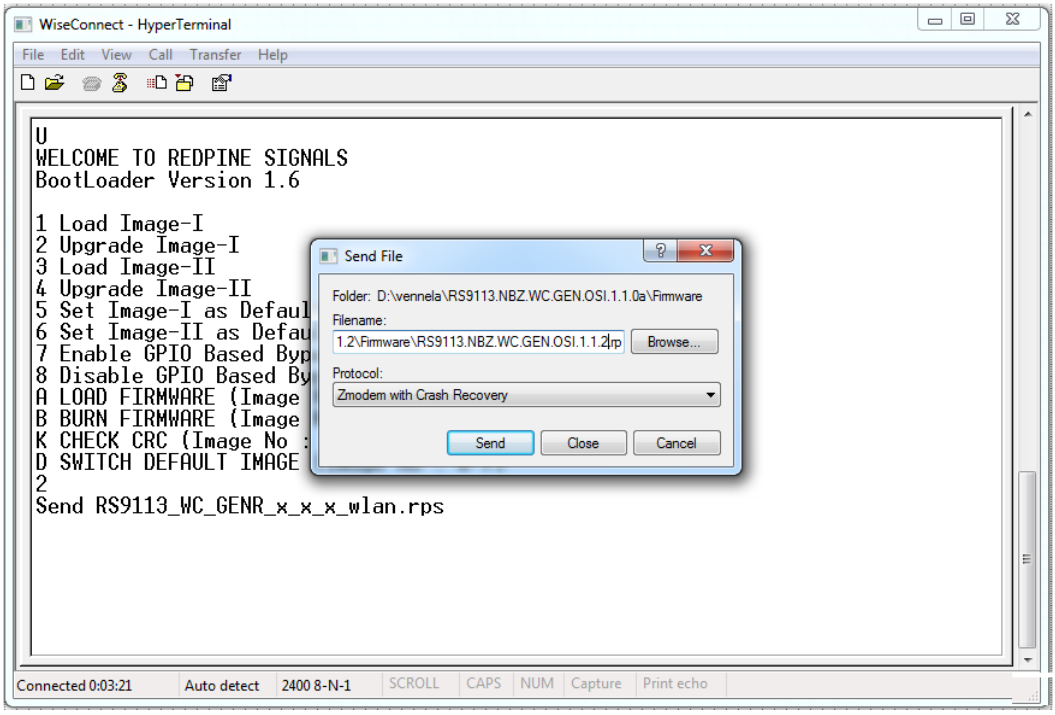


Figure 22: RS9113-WiSeConnect Module Image –I Upgrade File Selection Message

The dialog box message is displayed while file transfer is in progress as shown in the figure below.

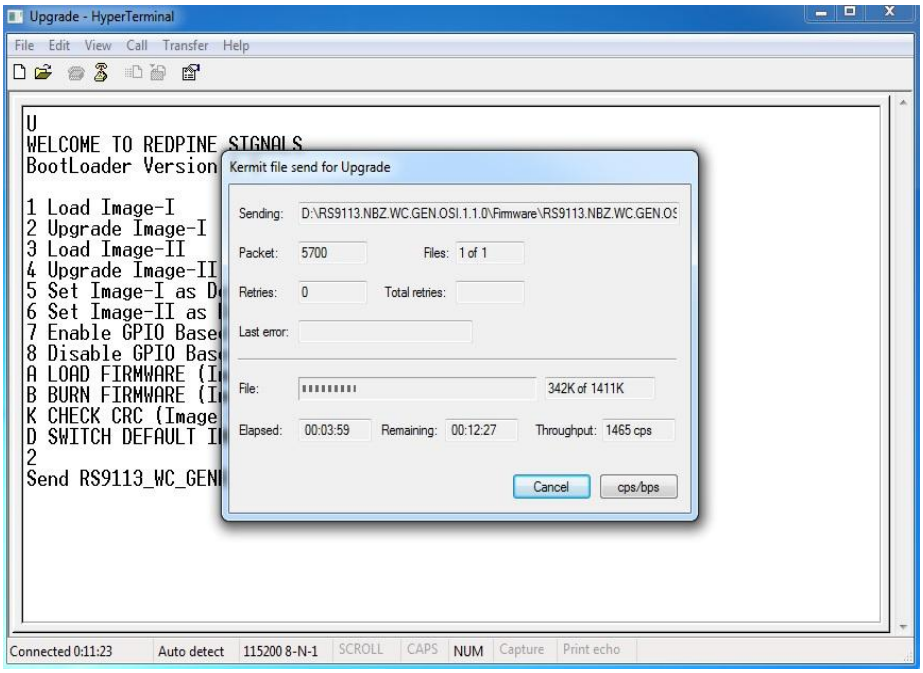


Figure 23: RS9113-WiSeConnect Module Image - I Firmware Upgrade File transfer Message

- After successfully completing the file transfer, module computes the checksum of the image and displays “@@@@@@@Upgradation Failed, re-burn the image@@@@@@@ ” in the case of failure and “@@Upgradation Failed and default image invalid, Bypass disabled @@” in the case of failure and default image gets corrupted.
- In the case of success, module checks if bootloader bypass is enabled and computes the checksum of the default image selected. If checksum fails, it sends “Upgradation successful, Default image invalid, gpio bypass disabled.” If checksum passes or gpio bypass not enabled, it sends “Upgradation Successful” message on terminal as shown in the figure below.

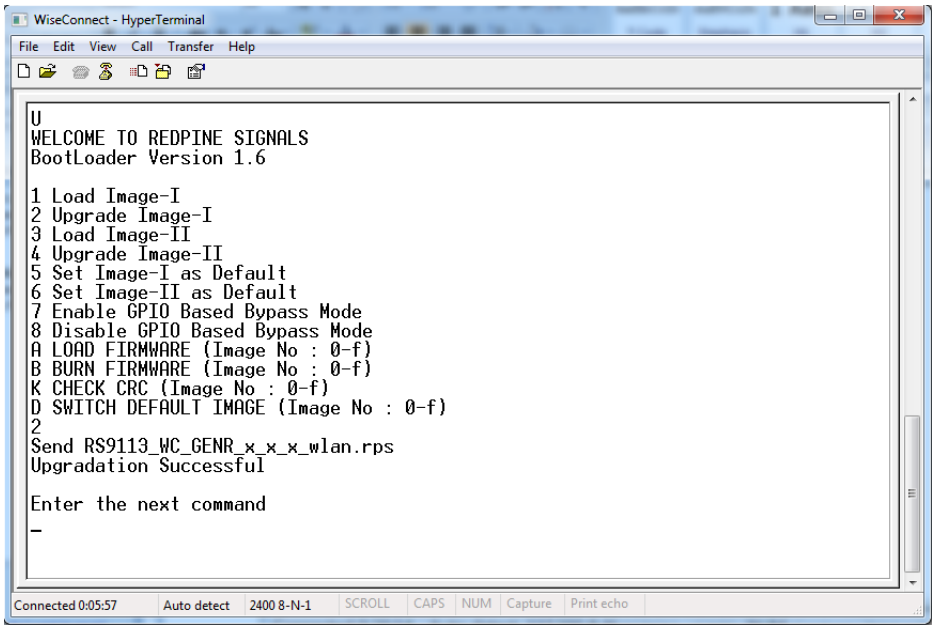


Figure 24: RS9113-WiSeConnect Module Image - I Firmware Upgrade Completion Message

- Till this step firmware image-I was successfully upgraded in the flash memory of the module.
- Follow the steps mentioned in Load Image – Ito load the firmware from flash, select Option 1 as mentioned in figure 21.
- The module is ready to accept commands from the Host.

3.2 Host Interaction Mode in SPI/USB

This section explain the exchange between host and module in host interaction mode (while bootloading) to select different boot options through SPI/USB interfaces.

3.2.1 SPI Startup Operations

If the selected host interface is SPI or USB, Bootloader uses two of the module hardware registers to interact with the host. In case of SPI host, these registers can be read or written using SPI memory read/write operations. In case of USB host vendor specific reads and writes on control end point are used to access these registers. Bootloader indicates its current state through HOST_INTF_REG_OUT and host can give the corresponding commands by writing onto HOST_INTF_REG_IN.

The significance of 2 bytes of the value written in to HOST_INTF_REG_IN register is as follows:

Nibble	Significance
Nibble[1:0]	Message code.
Nibble[2]	Represents the Image number for which the command is valid for.

Nibble	Significance
Nibble[3]	Represents the validity of the value in HOST_INTF_REG_IN register. Bootloader expects this value to be 0xA(HOST_INTERACT_REG_VALID). Booltloader validates the value written into this register if and only if this value is 0xA

Table 5: HOST_INTF_REG_IN Register values significance

Register Name	Memory Read/Write Address
HOST_INTF_REG_OUT	0x4105003C
HOST_INTF_REG_IN	0x41050034
PING Buffer	0x19000
PONG Buffer	0x1A000

Table 6: Bootloader message exchange registers

Bootloader Interaction Messages	Message Codes
HOST_INTERACT_REG_VALID	0xAB00
RSI_UPGRADE_BL	0x0023
RSI_LOAD_IMAGE_I_FW	0x0031
RSI_UPGRADE_IMAGE_I_FW	0x0032
RSI_SELECT_IMAGE_I_BY_DEFAULT	0x0035
RSI_LOAD_IMAGE_I_ACTIVE_LOW_FW	0x0071
RSI_SELECT_IMAGE_I_ACTIVE_LOW_BY_DEFAULT	0x0075
RSI_ENABLE_BOOT_BYPASS	0x0037
RSI_DISABLE_BOOT_BYPASS	0x0038
PING_VALID/PING_AVAIL	0x0049
PONG_VALID/PONG_AVAIL	0x004F
EOF_REACHED	0x0045
FWUP_SUCCESSFUL	0x0053
RSI_CHECK_IMAGE_CRC	0x004B
SEND_RPS_FILE	0x0032
FWUP_CRC_FAILURE	0x00CC

Bootloader Interaction Messages	Message Codes
LAST_CONFIG_NOT_SAVED	0x00F1
BOOTUP_OPTIONS_CHECKSUM_FAILED	0x00F2
INVALID_COMMAND	0x00F3
FWUP_SUCCESSFUL_INVALID_DEFAULT_IMAGE	0x00F4
INVALID_DEFAULT_IMAGE	0x00F5
FWUP_CRC_FAILED_INVALID_DEFAULT_IMAGE	0x00F6
CRC_PASS	0x00AA
CRC_FAIL	0x00CC
INVALID_ADDRESS	0x004C
VALID_FIRMWARE_NOT_PRESENT	0X0023

Table 7: Bootloader Message Codes

3.2.2 SPI Startup Messages on Powerup

Upon power-up the HOST_INTF_REG_OUT register will hold value 0XABxx. Here 0xAB (HOST_INTERACT_REG_VALID) signifies that the content of OUT register is valid. Bootloader checks for the validity of the boot up options by computing ones complement checksum. If the checksum fails, it computes the checksum from backup. If checksum passes, it copies the backup to the actual location and writes (HOST_INTERACT_REG_VALID | LAST_CONFIG_NOT_SAVED) in HOST_INTF_REG_OUT register. If checksum of backup options also fails, the boot up options are reset and (HOST_INTERACT_REG_VALID | BOOTUP_OPTIONS_CHECKSUM_FAILED) is written in HOST_INTF_REG_OUT register. In either of the cases, bootloader bypass is disabled. If the boot up options checksum passes, HOST_INTF_REG_OUT register contains 0xABxx where xx represents the two nibble bootloader version. This message is referred as BOARD_READY indication throughout the document.

For instance, for bootloader version 1.6, value of register will be 0xAB16. Host is expected to poll for one of the three values and should give any succeeding command(based on error codes if present) only after reading the correct value in HOST_INTF_REG_OUT reg.

3.2.3 Loading Firmware in the Module

Host can give options to bootloader to select the firmware load image type that will load the firmware from the flash of the module.

3.2.3.1 Load Image – I Firmware

Upon receiving Boardready, if host wants to load the Image –I firmware, it is expected to write value (HOST_INTERACT_REG_VALID | RSI_LOAD_IMAGE_I_FW) or (HOST_INTERACT_REG_VALID | RSI_LOAD_IMAGE_I_ACTIVE_LOW_FW) in HOST_INTF_REG_IN register. If host command is (HOST_INTERACT_REG_VALID |

RSI_LOAD_ IMAGE_I_FW) it is assumed that host platform is expecting active high interrupts. If command is (HOST_INTERACT_REG_VALID | RSI_LOAD_IMAGE_I_ACTIVE_LOW_FW) it is assumed that host is expecting active low interrupts and SPI hardware will be configured accordingly. After sending this command host should wait for interrupt for card ready message from loaded firmware.

Note:

For USB host interface mode interrupt configuration is not required, host should send (HOST_INTERACT_REG_VALID | RSI_LOAD_IMAGE_I_FW) to load Image – I

3.2.4 Upgrading the Firmware in the Module

With this option host can select to upgrade firmware in the flash of the module.

3.2.4.1 Upgrading Image – I Firmware

Steps for firmware upgradation sequence after receiving board ready are as follows.

1. After reading the valid BOARD_READY (i.e. HOST_INTERACT_REG_VALID | BOOTLOADER VERSION) value in HOST_INTF_REG_OUT, host writes RSI_UPGRADE_IMAGE_I_FW in HOST_INTF_REG_IN and host starts polling for HOST_INTF_REG_OUT.
2. Module polls for HOST_INTF_REG_IN register. When module reads a valid value (i.e. HOST_INTERACT_REG_VALID |RSI_UPGRADE_IMAGE_I_FW) in HOST_INTF_REG_IN, module writes (HOST_INTERACT_REG_VALID |SEND_RPS_FILE) in HOST_INTF_REG_OUT.
3. When host reads valid value (i.e. HOST_INTERACT_REG_VALID |SEND_RPS_FILE) in HOST_INTF_REG_OUT, host will write first 4 Kbytes of firmware image in PING Buffer and writes (HOST_INTERACT_REG_VALID | PING_VALID) in HOST_INTF_REG_IN register. Upon receiving PING_VALID command module starts burning this 4 Kbytes chunk onto the flash. When module is ready to receive data in PONG Buffer it sets value PONG_AVAIL (HOST_INTERACT_REG_VALID | PONG_AVAIL) in HOST_INTF_REG_OUT. Host is required to wait for this value to be set before writing next 4Kbytes chunk onto the module.
4. On reading valid value (i.e. HOST_INTERACT_REG_VALID | PONG_AVAIL) in HOST_INTF_REG_OUT, host starts memory write on PONG location and start polling for HOST_INTF_REG_OUT to read valid value (i.e. HOST_INTERACT_REG_VALID | PING_AVAIL). Module reads (HOST_INTERACT_REG_VALID | PONG_VALID) 0xAB4F value in HOST_INTF_REG_OUT and begin to write the data from PONG location into the flash.
5. This write process continues until host has written all the data into the PING-PONG buffers and there is no more data left to write.
6. Host writes a (HOST_INTERACT_REG_VALID | EOF_REACHED) in to HOST_INTF_REG_IN register and start polling for HOST_INTF_REG_OUT.
7. On the other side module polls for HOST_INTF_REG_IN register. When module reads (HOST_INTERACT_REG_VALID | EOF_REACHED) in HOST_INTF_REG_IN , it computes checksum for entire received firmware image. Then it checks if bypass is enabled. If enabled, it checks for the validity of the default image. If checksum is correct and default image is valid/GPIO bypass not enabled, module writes

(HOST_INTERACT_REG_VALID | FWUP_SUCCESSFUL) in HOST_INTF_REG_OUT register . If checksum is correct and default image is invalid(bypass enabled), module writes (HOST_INTERACT_REG_VALID | FWUP_SUCCESSFUL_INVALID_DEFAULT_IMAGE) in HOST_INTF_REG_OUT register and bypass is disabled. If it is checksum is incorrect and default image is valid/GPIO bypass is not enabled module writes(HOST_INTERACT_REG_VALID | FWUP_CRC_FAILURE) in HOST_INTF_REG_OUT register . If it is checksum is incorrect and default image is invalid(bypass enabled) module writes(HOST_INTERACT_REG_VALID | FWUP_CRC_FAILED_INVALID_DEFAULT_IMAGE) in HOST_INTF_REG_OUT register and bypass is disabled.

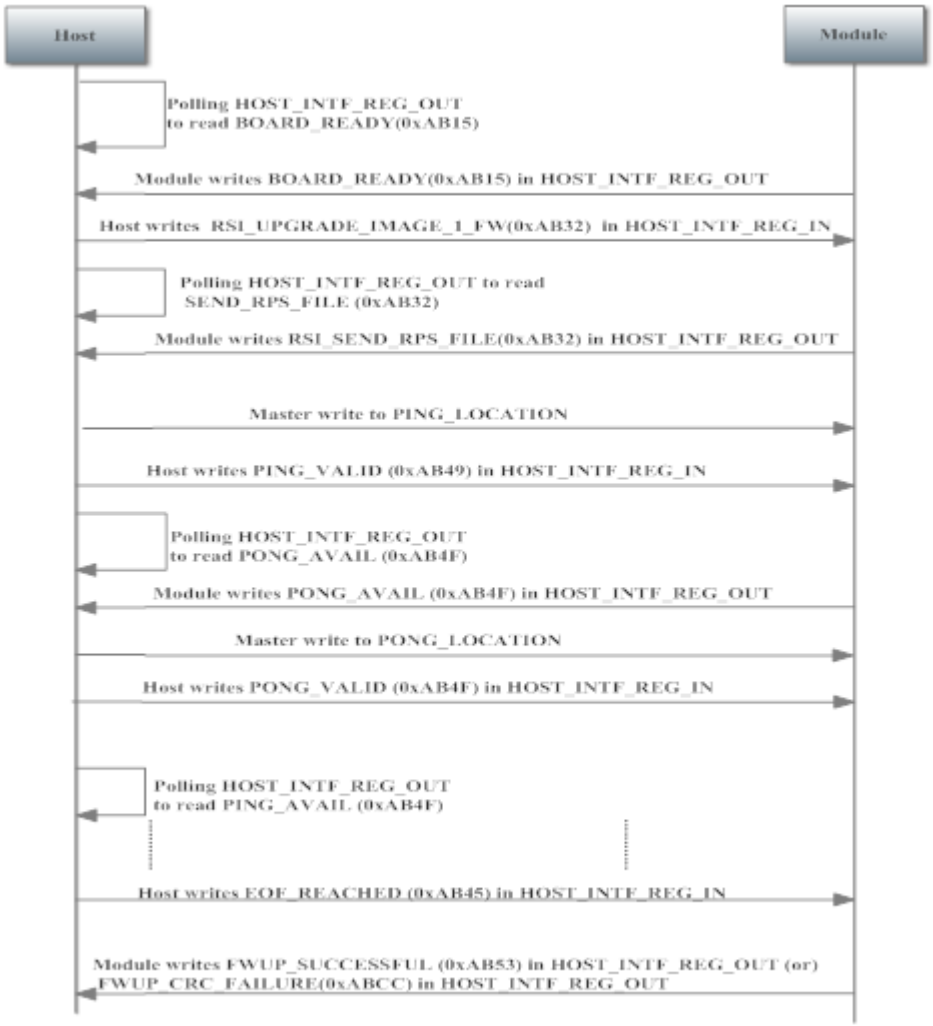


Figure 25: Image – I upgradation through SPI/USB

3.3 GPIO Based Bootloader Bypass Mode

In GPIO based bootloader bypass mode host interactions with bootloader can be bypassed. There are two steps to enable GPIO based Bootloader bypass mode:

1. Host need to select Image-I as default image to load in bypass mode.

2. Enable Bootloader bypass mode.
3. Assert GPIO #15 to Bypass Bootloader on powerup or hardware reset.

To enable Bootloader Bypass mode host first has to give default image that has to be loaded in bypass mode and select the bypass mode(enable).After rebooting the module, it goes to bypass mode and directly loads the default firmware image.

3.3.1 Bypass Mode in UART/USB-CDC

3.3.1.1 Making default image selection

With this option host can select the default firmware image to be loaded.

3.3.1.1.1 Selecting Image – I as the Default Image

- If option '5' is selected default image is switched to Image- I.
- When default image is selected, module checks for the validity of the image selected and displays "Configuration saved" if valid and "Default image invalid" if valid default image is not present.

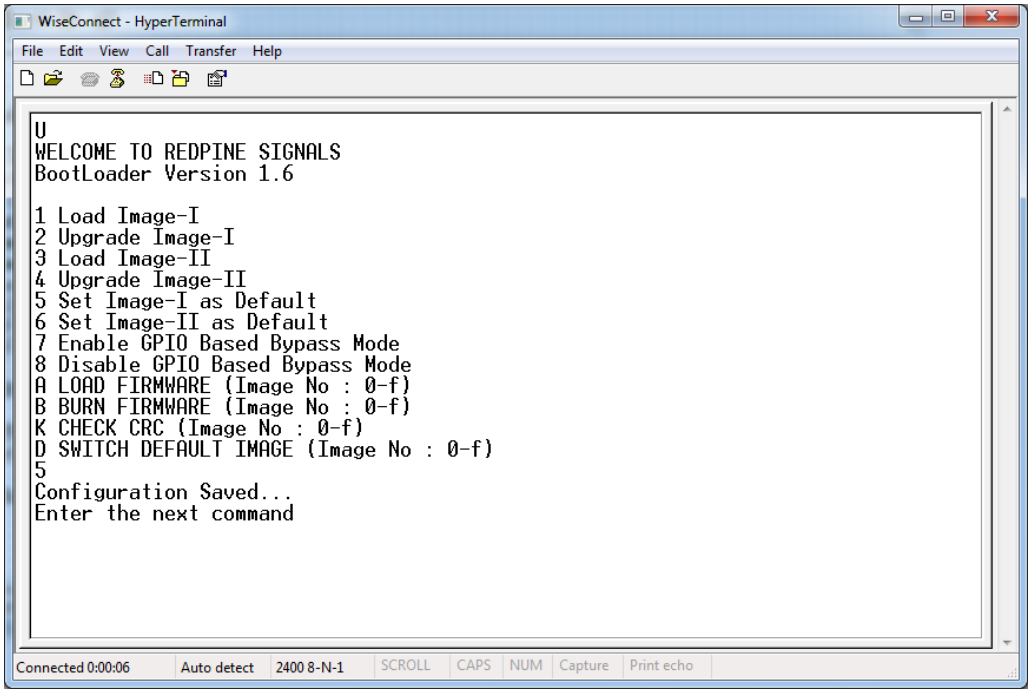


Figure 26: Making Image –I as default image

3.3.1.2 Enable/Disable GPIO Based Bypass Option

This option is for enabling or disabling the GPIO bootloader bypass mode.

3.3.1.2.1 Enabling the GPIO Based Bypass Mode

If you select option 7, GPIO based Bootload bypass gets enabled. When this option is selected, module checks for the validity of the image selected and displays "Configuration saved" if valid and "Default image invalid" if valid default image is not present. Once

enabled, from next bootup, Bootloader will latch the value of GPIO-15. If asserted, it will bypass whole bootloading process and load default firmware image selected.

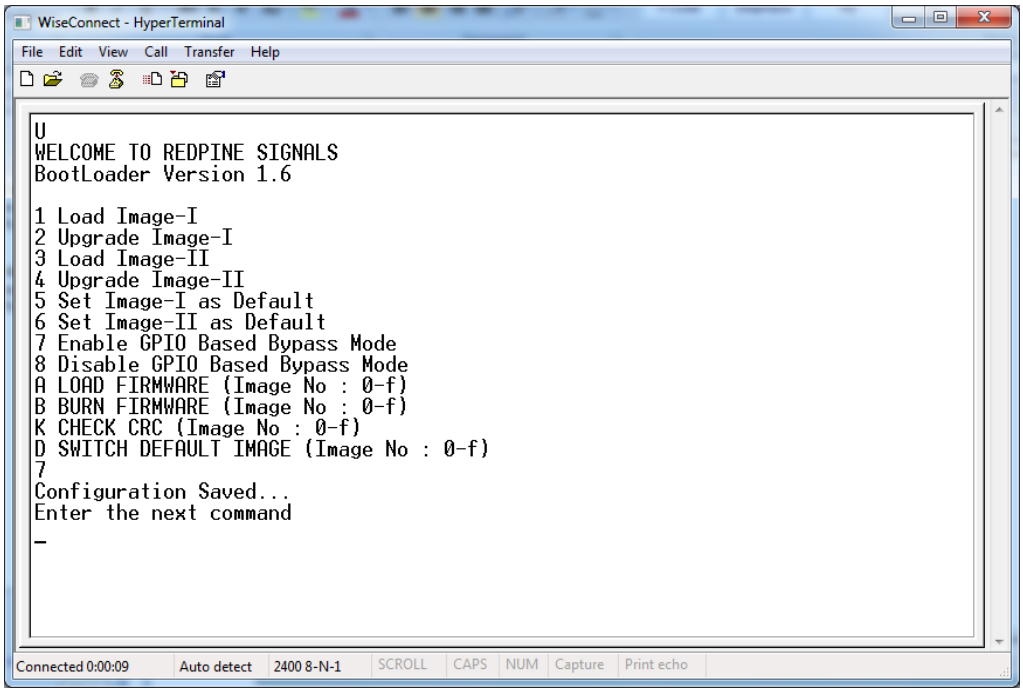


Figure 27: Enabling the GPIO based bypass mode

3.3.1.2.2 Disabling the GPIO Based Bypass Mode

- If host selects option 8, GPIO based bypass gets disabled.

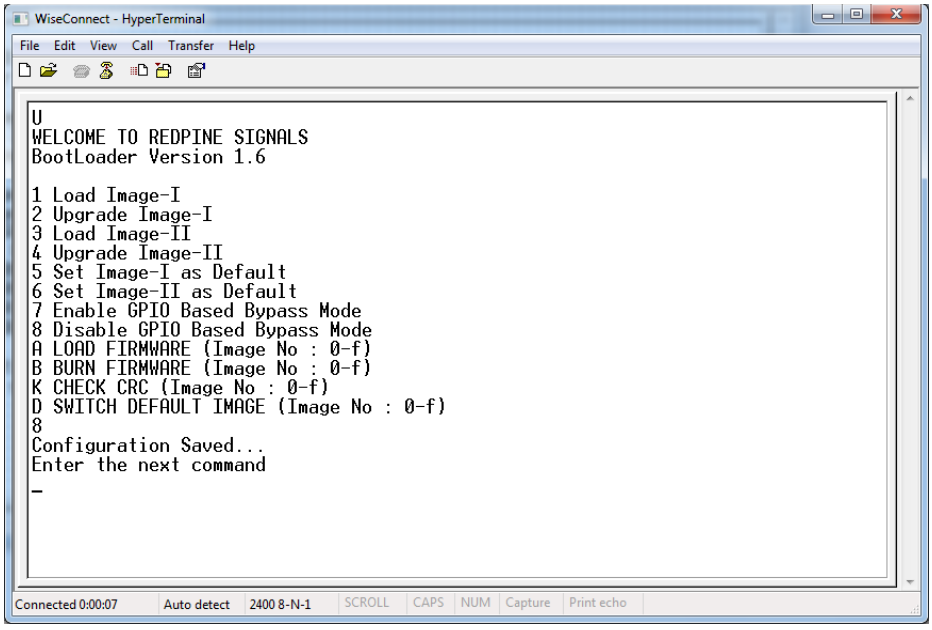


Figure 28: Disabling the GPIO based bypass mode

Note:

GPIO-15 need to deasserted on powerup to move to host interaction mode, to select bootup options like disable Bypass mode or to change default image etc.

3.3.1.3 Check CRC of the Selection Image

This option enables user to check whether the given image is valid or not. When this command is given, bootloader asks for the image for which checksum has to be verified as shown in figure below.

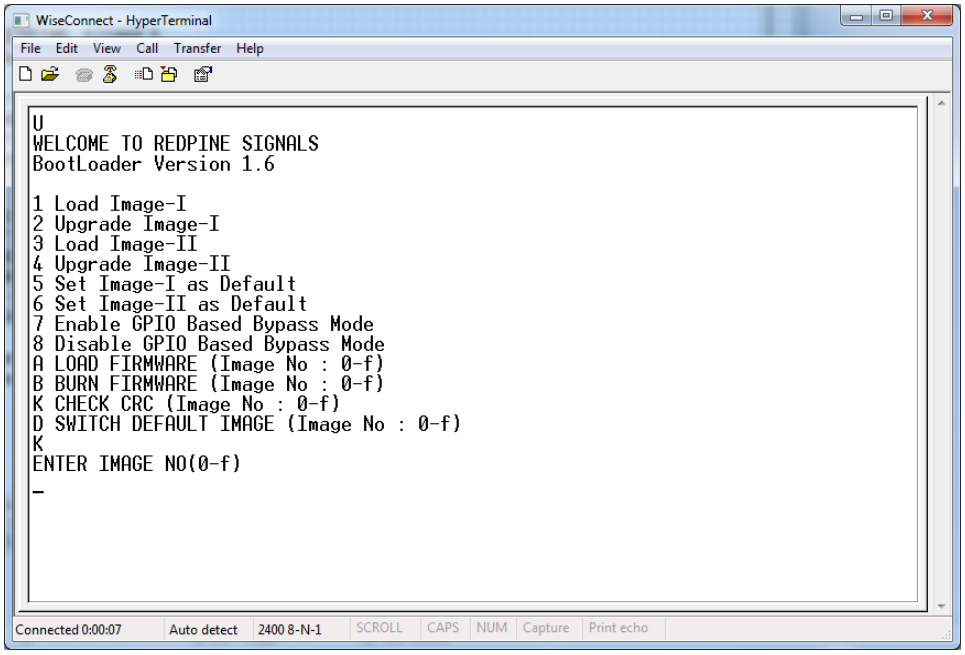


Figure 29: CRC Check example

If the check sum of the Image passes, Bootloader posts a message i.e. “CRC PASS” as shown in below figure.

If the check sum fails,Bootloder posts a message “CRC FAIL”. And if the given image does not exists, Bootloader posts a message of “INVALID ADDRESS”.

For instance, consider Image – I (Image no : 0),given below figure illustrates the CRC check for Image – I.

Bootloader checksum can be queried by giving image number as ‘f’.

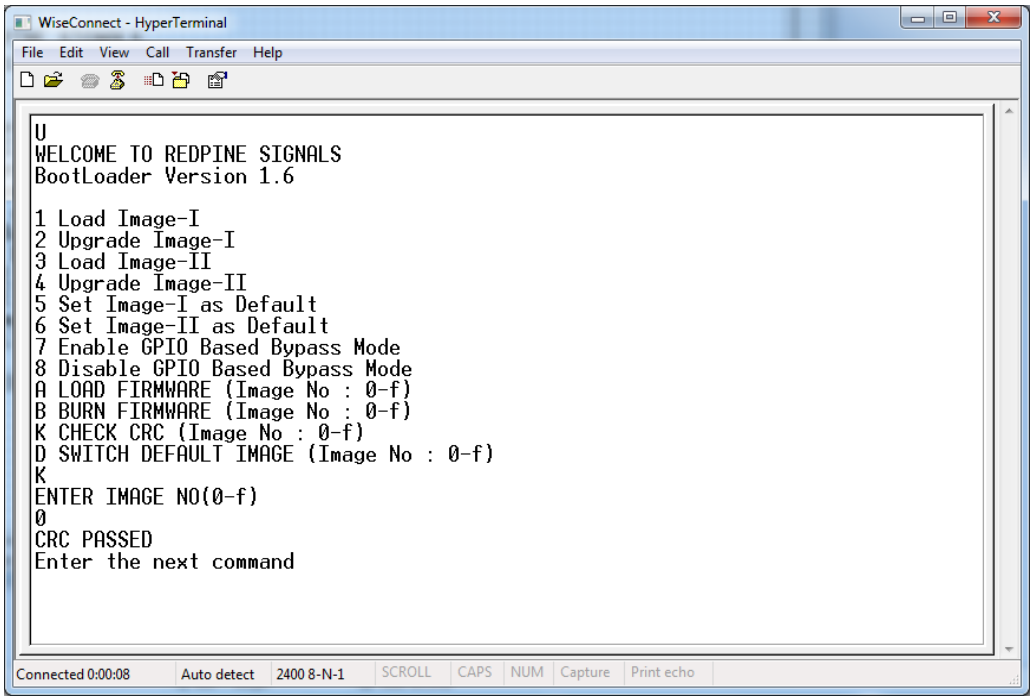


Figure 30: CRC Check passed

3.3.2 Bypass Mode in SPI/USB

3.3.2.1 Selecting the Default Image

Upon receiving Boardready, if host wants to select default functional image, it is expected to write value (HOST_INTERACT_REG_VALID | RSI_SELECT_IMAGE_I_BY_DEFAULT) in HOST_INTF_REG_IN register. Host is expected to receive confirmation (HOST_INTERACT_REG_VALID | RSI_SELECT_IMAGE_I_BY_DEFAULT) in HOST_INTF_REG_OUT register if default image is valid. If default image is invalid, (HOST_INTERACT_REG_VALID | INVALID_DEFAULT_IMAGE) is written in HOST_INTF_REG_OUT register

3.3.2.2 Enable/Disable GPIO Based Bootloader Bypass Mode

Upon receiving Boardready, if host wants to enable or disable GPIO based bypass mode, it is expected to write value (HOST_INTERACT_REG_VALID | RSI_ENABLE_BOOT_BYPASS) or (HOST_INTERACT_REG_VALID | RSI_DISABLE_BOOT_BYPASS) in HOST_INTF_REG_IN register if default image is valid. If default image is invalid, (HOST_INTERACT_REG_VALID | INVALID_DEFAULT_IMAGE) is written in HOST_INTF_REG_OUT register. Host is expected to reboot the board after receiving confirmation (HOST_INTERACT_REG_VALID | RSI_ENABLE_BOOT_BYPASS) or (HOST_INTERACT_REG_VALID | RSI_DISABLE_BOOT_BYPASS) or (HOST_INTERACT_REG_VALID | INVALID_DEFAULT_IMAGE) in HOST_INTF_REG_OUT register. If GPIO based bypass is enabled, from next bootup onwards, bootloader will latch the state of GPIO #15. If GPIO #15 is asserted, bootloader will not give board ready and it will directly load the default functional image selected.

Note:

GPIO #15 need to be deasserted on powerup to move to host interaction mode, to select bootup options like disable Bypass mode or to change default image.

3.4 Check CRC of the Selected Image

Upon receiving Boardready, if host wants to check the CRC of the Image - I, host is expected to write v 0xA04B in HOST_INTF_REG_IN register to query validity of checksum of Image-I or 0xAF4B in HOST_INTF_REG_IN register to query validity of checksum of bootloader. If the Checksum of the image passes ,bootloader writes 0xABAA else if checksum fails, bootloader writes 0xABCC in HOST_INTF_REG_OUT register. If valid image is not present, module sends 0xAB4C in HOST_INTF_REG_OUT register.

Note:

1) Boot loader allows host to save more than one configuration without power on reset of the module on UART/USB-CDC .When "Enter the next command" is displayed on the HyperTerminal, user can give next command.
2) All the protocols like WLAN ,BT,BLE , ZigBee ,WLAN+BT,WLAN+BLE,WLAN+ZigBee are supported in a single image/binary format from 1.0.10 release on wards. Thus only Image –I is valid.

3.5 Reconfirmation Enable

This hidden option 'E' (valid only in UART/USB-CDC) enables user to reconfirm the selected option so as to avoid corruption. When this command is given, module asks for reconfirmation. On giving "YES", the module saves it and asks for reconfirmation for every command for which boot up options get altered.

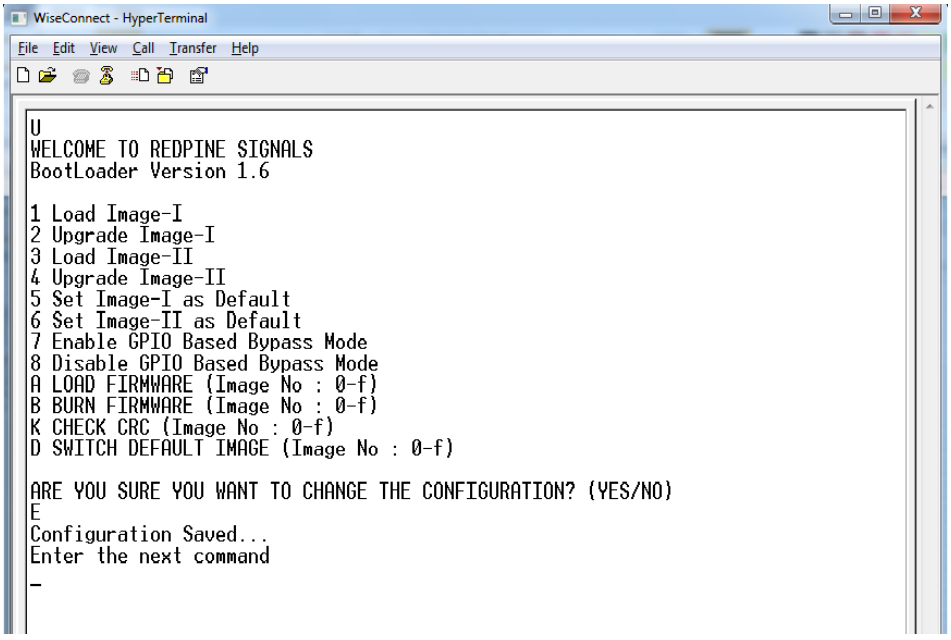


Figure 31: Reconfirmation Enable

On enabling reconfirmation, remaining options(Reconfirmation enable, reconfirmation disable,binary mode selection, binary mode disable) are also displayed on next boot up.

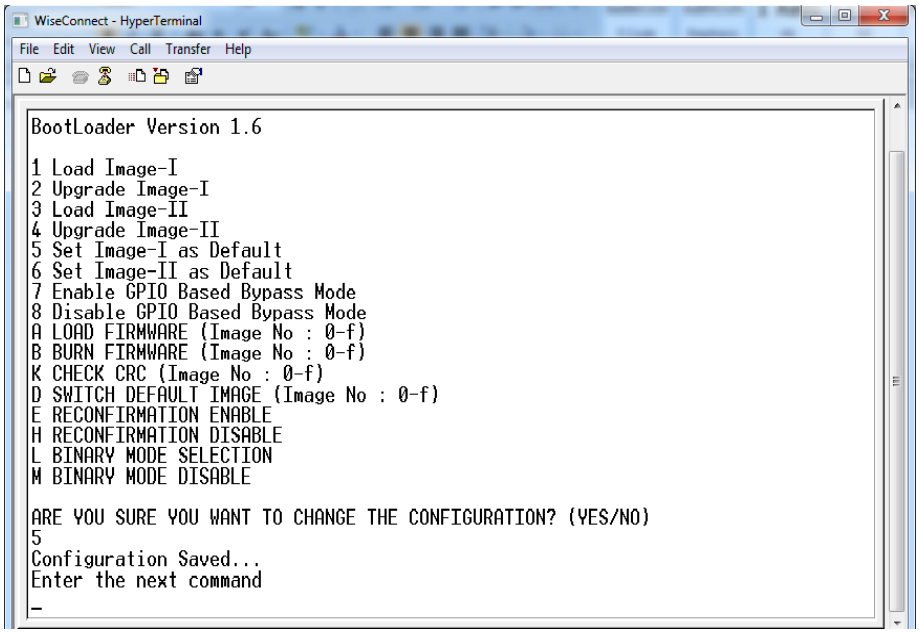


Figure 32: Boot up options after reconfirmation enable

3.6 Reconfirmation Disable

This hidden option 'H' (valid only in UART/USB-CDC) disables user reconfirmation.

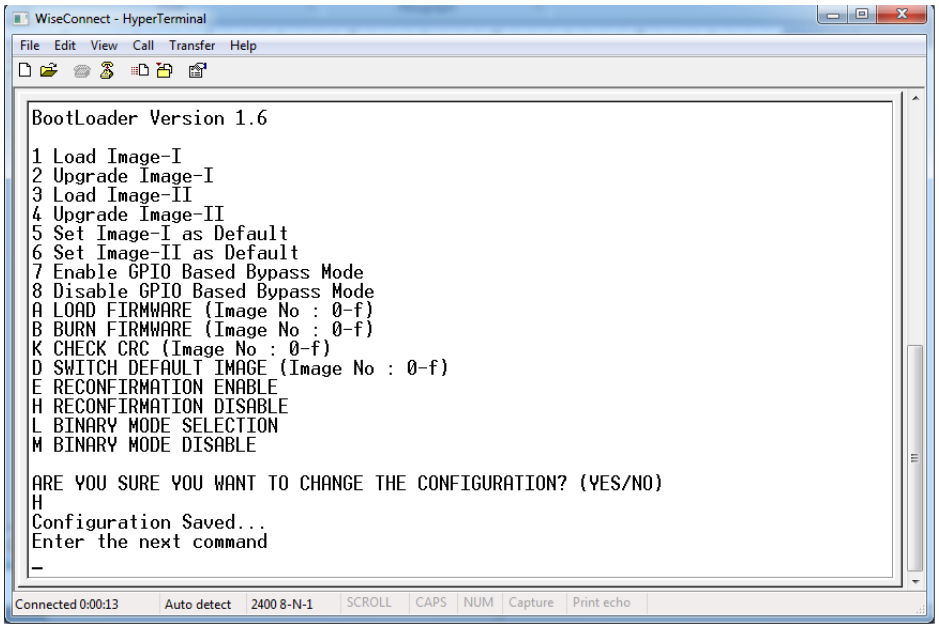


Figure 33: Reconfirmation disable

3.7 Binary Mode Selection

This hidden option 'L' (valid only in UART/USB-CDC) select binary mode.

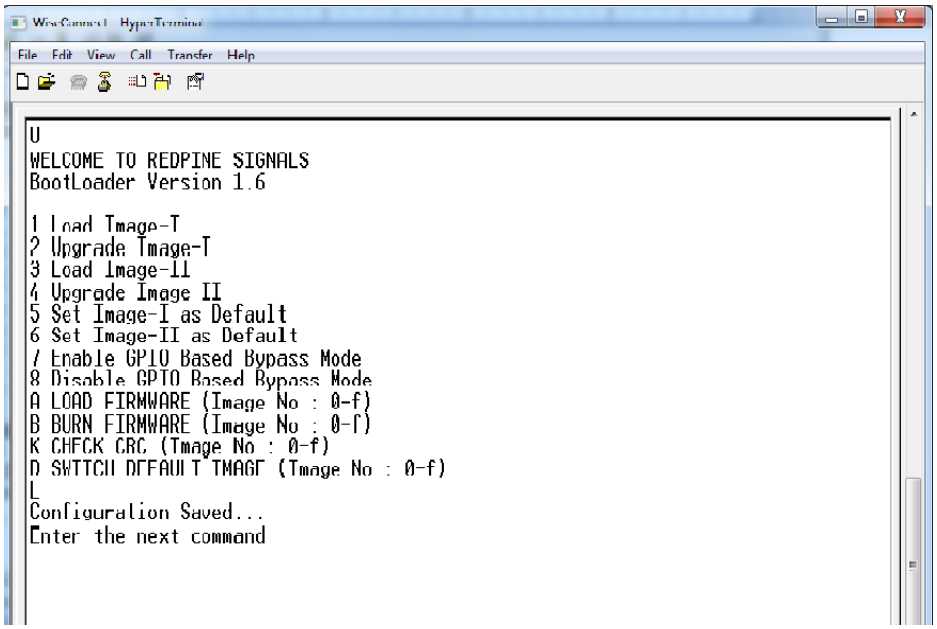


Figure 34: Binary mode enable

3.8 Binary Mode Disable

This hidden option 'M' (valid only in UART/USB-CDC) disable binary mode.

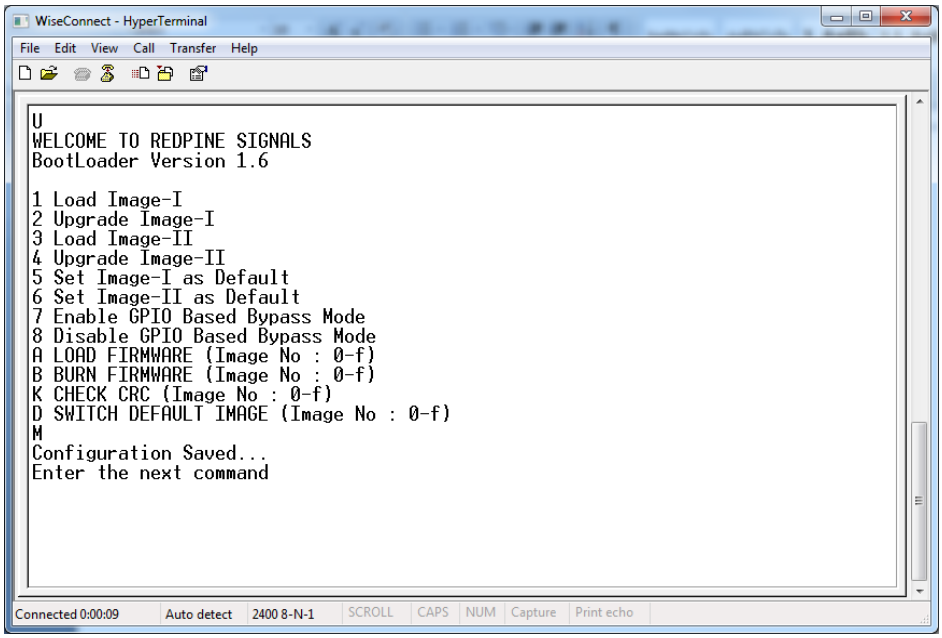


Figure 35: Binary mode disable

4 Wi-Fi Software Programming

4.1 Architecture Overview

The following figure depicts the software architecture of the RS9113-WiSeConnect Module.

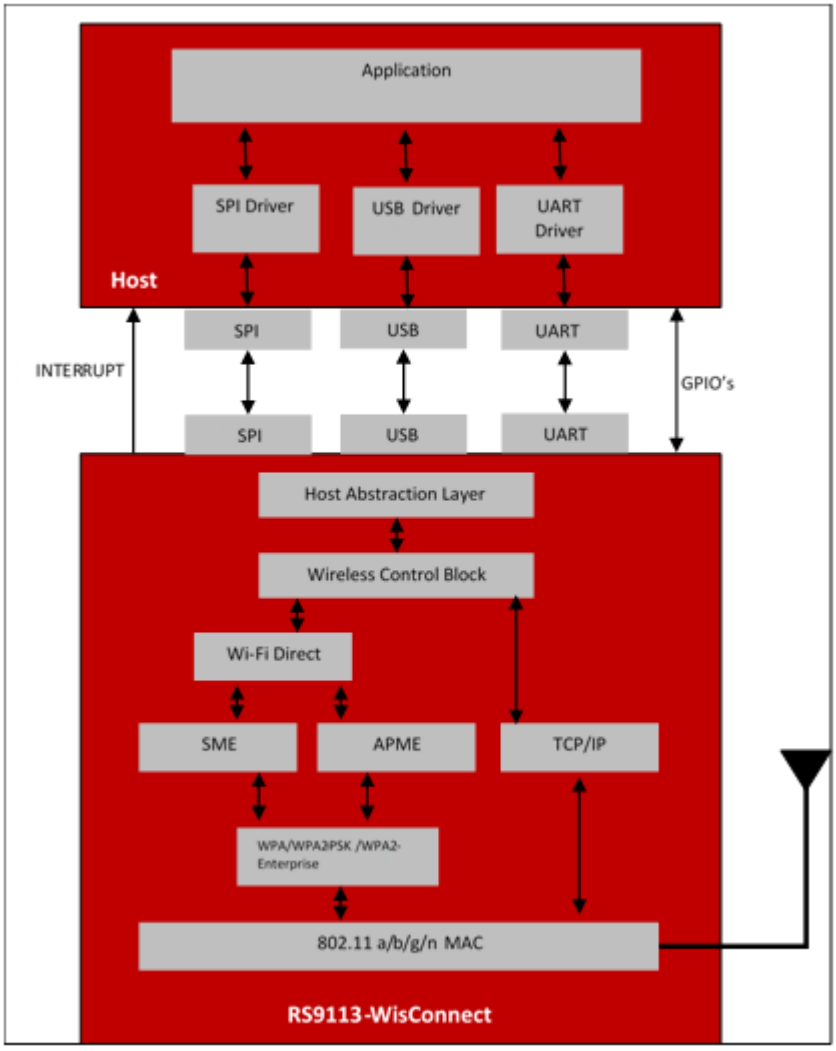


Figure 36: Wi-Fi Software Architecture

4.1.1 Application

The application layer invokes wireless APIs provided by SPI/UART driver. The Application developer can use these APIs to build wireless applications.

4.1.2 SPI

The SPI interface on the RS9113-WiSeConnect Module works in slave mode. It is a 4-wire interface. In addition to the SPI interface, the module provides additional interrupt pin to signal events to the host.

The interrupt is raised by the module in SPI mode for the following condition.

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

The interrupt from module is active high and host has to configure the interrupt in level trigger mode.

4.1.3 USB

The USB interface on the RS9113-WiSeConnect Module transmits/receives data to/from the Host in USB mode.

4.1.4 UART

The UART interface on the RS9113-WiSeConnect Module transmits/receives data to/from the Host in UART mode.

4.1.5 GPIO

Power save mode requires GPIO's.

4.1.6 Hardware Abstraction Layer (HAL)

The HAL abstracts the lower layers in the Host interface with which the RS9113-WiSeConnect Module is connected. The HAL interacts with the Wireless Control Block layer for the processing of the frames obtained from or destined to the Host.

4.1.7 Wireless Control Block (WCB)

The data from/to the Host is classified as Wi-Fi specific frames or TCP/IP specific frames. The functionality of the WCB module depends on the type and direction of the frames.

4.1.7.1 Wi-Fi Control Frames

The WCB interprets the Wi-Fi control information from the Host and interacts with the SME (Station Management Entity) or APME (Access Point Management Entity) based on operating mode of RS9113-WiSeConnect Module. Configuration of the RS9113-WiSeConnect Module from the Host for Wi-Fi access is through AT commands or SPI commands.

4.1.7.2 TCP/IP Control Frames

TCP/IP networking protocol provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination.

If the packets received from the Host by WCB during transmit are interpreted as TCP/IP frames then WCB interacts with TCP/IP stack for further processing before passing the packets to MAC layer. Similarly if the packets are received from TCP/IP stack by WCB during reception then WCB processes these packets passing to host.

4.1.8 Wi-Fi Direct

The Wi-Fi Direct is the core layer to operate the module in P2P mode. Wi-Fi Direct use the SME & APME blocks for p2p operations. After Wi-Fi direct exchanges module will became either Wi-Fi Direct Group owner or Wi-Fi Direct client.

4.1.9 Station Management Entity (SME)

The SME is the core layer which manages the Wi-Fi connectivity in Station mode or P2P client mode. The SME maintains the state machine to detect the activity on the Wi-Fi network and indicates to the user accordingly. It interacts with the WPA supplicant if Security is enabled in the Wi-Fi network.

4.1.10 Access point Management Entity (APME)

The APME is the core layer which manages the connectivity in Access Point and Wi-Fi direct group owner modes. The APME maintains the state machine to handle multiple clients connected to the module. It interacts with WPA supplicant if Security is enabled in the Wi-Fi network.

4.1.11 WPA Supplicant

The main functionality of WPA supplicant is, support for key negotiation between Wi-Fi devices in secure mode. This functionality depends on the mode in which RS9113-WiSeConnect Module operates in Station mode, Access point mode, Wi-Fi Direct mode. The WPA supplicant is used to initiate the 802.1x/E Access Point authentication if WPA/WPA2-PSK is used as the security parameter. It also plays a major part in performing the 4-way handshake to derive the PTK in WPA/WPA2-PSK modes.

5 Command Mode Selection

This section describes the AT command mode or Binary mode selection in UART and USB-CDC.

After bootloader interaction, module gives “Loading Done” string in ASCII format to host. After receiving “Loading Done”, based on first command received from host, module selects command mode.

Module reads first 4 bytes, if it matches with “AT+R”, AT command mode is configured, otherwise Binary mode will be configured. Once mode is configured it will remain in same mode until reset or power cycle.

Note:

“AT+R” is not case sensitive.

Note:

Once you change the mode(i.e ASCII->Binary or Binary-> ASCII) then it cannot be configured auto command selection mode.

6 AT Command Mode

The Wi-Fi AT command set represents the frames that are sent from the Host to operate the RS9113-WiSeConnect Module. The command set resembles the standard AT command interface used for modems.

- All AT commands start with “at” and are terminated with a carriage return(‘\r’) and a new line(‘\n’) character.
- The AT command set for the RS9113-WiSeConnect Module starts with “at+rsi_” followed by the name of the command and any relevant parameters.
- In some commands, a ‘?’ character is used after the command to query data from the module.

Appendix A: Sample flow of commands for Wi-Fi over UARTcaptures sample flow of commands to configure the module in various functional modes.

Syntax of AT command:

```
at+rsi_<command_name>[=][parameters][?]\r\n
```

Example:

```
at+rsi_command=< parameter1 >,< parameter2 >,< parameter3
>\r\n
```

Each parameter should be separated by comma (,).

NOTE:

1) All commands are issued from Host to module as a sequence of ASCII characters. All return messages from module to Host consist of OK or ERROR strings, along with some return parameters. The return parameters may be ASCII or Hex on a case by case basis. ERROR is accompanied by <Error code>.
2) A command should NOT be issued by the Host before receiving the response of a previously issued command from the module.

RS9113-WiSeConnect Module support following host interface in AT Command mode:

- UART
- USB-CDC

7 Binary Command Mode

This section explains the Wi-Fi commands that are used to configure RS9113-WiSeConnect module in Binary Mode.

Following are list of host interfaces supported in Binary Mode:

- UART
- SPI
- USB
- USB-CDC

The Wi-Fi configuration and operation commands are sent to the module and the responses are read from the module using frame write/frame read (as mentioned in the preceding sections) so these configuration and operation commands are called as command frames.

The command frame is categorized as management or data frames. The management frames are used to configure the Wi-Fi module to access Wi-Fi connectivity, TCP/IP stack and operate the module. Data frames are used to send the data.

Management and data frames are exchanged between host and module. Management frame is sent from Host to the module to configure the module, and also is sent from module to host to send responses to these commands.

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

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

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

Note:

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

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

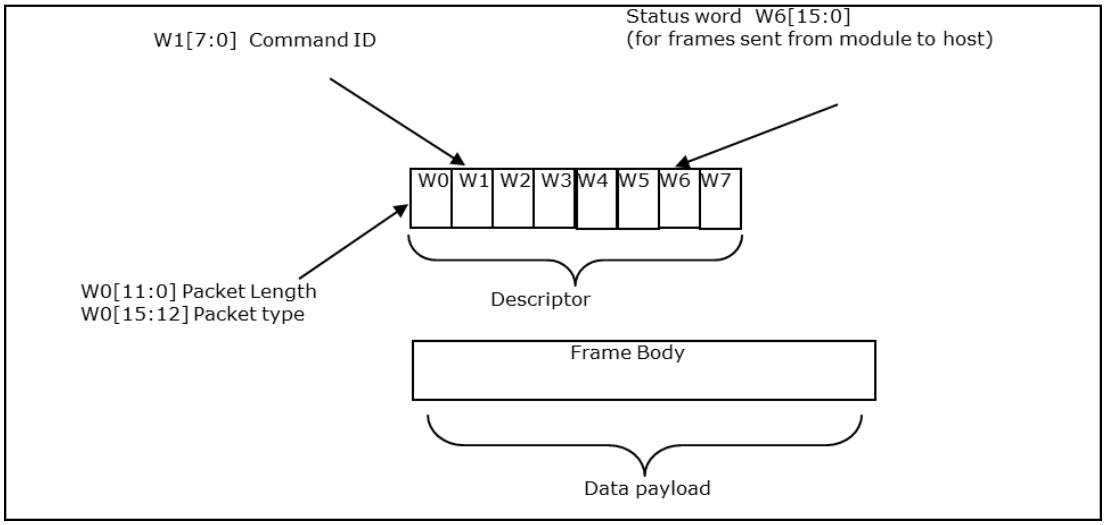


Figure 37: Command Frame Format

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

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

Word	Management Frame Descriptor	Data Frame Descriptor
Word7 W7 [15:0]	Reserved	Reserved

Table 8: Frame Descriptor for Management/Data Frames in Binary Mode

The management frames represent the command frames that are sent from the Host to the RS9113-WiSeConnect Module to configure for Wi-Fi access. These are frame write commands. The following are the types of management requests and responses and the corresponding codes. The first table below is applicable when the Host sends the frames to the module; the second table below is applicable when the module sends the frames to the host. The corresponding code is to be filled in W1 [7:0] mentioned in the table above.

Command	Command ID
Send Data	0x00
Set Operating Mode	0x10
Band	0x11
Init	0x12
Scan	0x13
Join	0x14
Power save mode	0x15
Sleep Timer	0x16
Set Mac address	0x17
Query Network Parameters	0x18
Disconnect	0x19
Antenna Select	0x1B
Soft Reset	0x1C
Set region	0x1D
Config Save	0x20
Config Enable	0x21
Config Get	0x22
User Store configuration	0x23
AP config	0x24
Set WEP Keys	0x25
Debug Prints on UART2	0x26
Ping command	0x29
RSSI Query	0x3A
Flashtype	0X3B
Multicast Address Filter	0x40
Set IP Parameters	0x41

Command	Command ID
Socket Create	0x42
Socket Close	0x43
DNS Resolution	0x44
Query LTCP Connection Status	0x46
Query WLAN Connection Status	0x48
Query Firmware Version	0x49
Get MAC Address	0x4A
Configure P2P	0x4B
Configure EAP	0x4C
Set Certificate	0x4D
Query GO Params	0x4E
Load Web pages	0x50
HTTP GET	0x51
HTTP POST	0x52
HTTP PUT	0x53
DNS Server	0x55
URL response	0x56
FWUP REQ OK	0x5A
BG scan	0x6A
HT Caps REQ	0x6D
Rejoin params	0x6F
WPS method REQ	0x72
Roam Params REQ	0x7B
PER MODE REQ	0x7C
Webpage Clear REQ	0x7F
SNMP Get response	0x83
SNMP Get next response	0x84
SNMP ENABLE	0x85
SNMP TRAP	0x86
SNMP_GET_STATS	0x88
IPv6 Config	0x90
Trigger Auto configuration	0x91
WMM PS REQ	0x97
Firmware upgradation from host	0x99
Webpage Erase REQ	0x9A
JSON erase REQ	0x9B

Command	Command ID
JSON create REQ	0x9C
PER Stats REQ	0xA2
Transparent Mode REQ	0xA3
UART flow control	0xA4
Set PSK/PMK REQ	0xA5
Socket Configuration REQ	0xA7
RF current mode configuration	0xAD
Multicast request	0xB1
Sent Bytes on Socket	0xB2
HTTP Abort	0xB3
HTTP Credentials Request	0xB4
Load MFI ie in beacon	0XB5
Read Authentication certificate	0xB6
IAP co processor init	0xB7
Generate MFI authentication signature	0XB8
Set Region of Access point	0xBD
PUF Enroll	0xD0
PUF disable enroll	0xD1
PUF start	0xD2
PUF set key	0xD3
PUF disable set key	0xD4
PUF get key	0xD5
PUF disable get key	0xD6
PUF load key	0xD7
AES Encrypt	0xD8
AES Decrypt	0xD9
AES MAC	0xDA
PUF_Intrinsic key	0XCE
Power save ACK	0xDE
MDNSD START REQ	0xDB
DHCP USER CLASS REQ	0xEC
FTP REQ	0xE2
FTP FILE WRITE REQ	0xE3
SNTP REQ	0xE4
SMTP REQ	0xE6

Command	Command ID
POP3 Client REQ	0xE7
Set RTC time	0xE9
Config WLAN parameters (RTS)	0xC6

Table 9: Command IDs for Tx Data Operation

Command	Response ID
Set Operating Mode	0x10
Band	0x11
Init	0x12
Scan	0x13
Join	0x14
Power save mode	0x15
Sleep Timer	0x16
Set Mac address	0x17
Query Network Parameters	0x18
Disconnect	0x19
Antenna select	0x1B
Soft Reset	0x1C
Set region	0x1D
Config save	0x20
Config Enable	0x21
Config Get	0x22
User store configuration	0x23
AP Config	0x24
Set WEP Keys	0x25
Debug Prints on UART2	0x26
Ping command	0x29
Async connection accept request from remote wfd device	0x30
RSSI Query	0x3A
Flashtype	0X3B
Multicast Address filter	0x40
IP Parameters Configure	0x41
Socket Create	0x42
Socket Close	0x43
DNS Resolution	0x44

Command	Response ID
Query LTCP Connection Status	0x46
Query WLAN Connection Status	0x48
Query Firmware Version	0x49
Get MAC Address	0x4A
Configure P2P	0x4B
Configure EAP	0x4C
Set Certificate	0x4D
Query GO Params	0x4E
Load webpage	0x50
HTTP GET	0x51
HTTP POST	0x52
Roam Params RSP	0x7B
Async WFD	0x54
DNS Server	0x55
URL response	0x56
FWUP RSP	0x59
Async TCP Socket Connection Established	0x61
Async Socket Remote Terminate	0x62
URL request	0x64
BG scan	0x6a
HT Caps RSP	0x6D
Rejoin params	0x6F
Module state	0x70
WPS method RSP	0x72
Roam Params REQ	0x7b
PER MODE RSP	0x7C
Webpage Clear RSP	0x7F
SNMP GET	0x80
SNMP GET NEXT	0x81
SNMP SET	0x82
SNMP_GET_STATS	0x88
Card Ready	0x89
WMM PS RSP	0x97
Webpage Erase RSP	0x9A
JSON erase RSP	0x9B

Command	Response ID
JSON create RSP	0x9C
JSON Update	0x9D
Config IPv6	0xA1
PER Stats	0xA2
Transparent Mode RSP	0xA3
UART flow control RSP	0xA4
Set PSK/PMK RSP	0xA5
Socket Configuration RSP	0xA7
IP Change notify	0xAA
TCP ACK indication to host	0xAB
Delayed ACK for UART binary mode	0xAC
RF current mode configuration	0xAD
Multicast response	0xB1
HTTP Abort	0xB3
HTTP Credential Response	0xB4
Load MFI ie in beacon	0XB5
Read Authentication certificate	0xB6
IAP co processor init	0xB7
Generate MFI authentication signature	0XB8
Set Region of Access point	0xBD
Station connected Indication	0xC2
Station Disconnected Indication	0xC3
PUF Enroll	0xD0
PUF disable enroll	0xD1
PUF start	0xD2
PUF set key	0xD3
PUF disable set key	0xD4
PUF get key	0xD5
PUF disable get key	0xD6
PUF load key	0xD7
AES Encrypt	0xD8
AES Decrypt	0xD9

Command	Response ID
AES MAC	0xDA
PUF_Intrinsic key	0XCE
Wakeup indication	0xDD
Sleep indication	0xDE
Receive data	Ignore the response ID field while receiving data from a remote terminal
MDNSD START RSP	0xDB
FTP RSP	0xE2
FTP FILE WRITE RSP	0xE3
SNTP RSP	0xE4
SNTP SERVER RSP	0xE5
SNMP GET RESPONSE RSP	0x83
SNMP GET NEXT RESPONSE RSP	0x84
SNMP ENABLE RSP	0x85
SNMP TRAP RSP	0x86
RSI_RSP_SNMP_GET_STATS	0x88
POP3 Client RSP	0xE7
POP3 Client terminate RSP	0xE8
RTC time response	0xE9
Request Timeout	0xEA
HTTP POST DATA	0xEB
DHCP USER CLASS RSP	0xEC
DNS Update	0xED
OTAF	0xEF
Config WLAN parameters (RTS)	0xC6

Table 10: Response IDs for Rx Operation

8 Commands

The following sections will explain RS9113-WiSeConnect commands, their structures, their responses and relevance in AT mode and Binary mode.

8.1 Set Operating Mode

Description:

This is the first command that needs to be sent from the Host after receiving card ready frame from module in case of interface other than UART.

In case of UART interface 'hwflow_control' command can be given as the first command.

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>,<ble_feature_bi
t_map>,<ext_tcp_ip_feature_bit_map>\r\n
```

Note:

If BIT(31) is set to '1' in custom_feature_bitmap

```
at+rsi_opermode=<oper_mode>,<feature_bit_map>,<tcp_ip_feature_
bit_map>,<custom_feature_bitmap><ext_custom_feature_bit_map>\r
\n
```

if BIT(31) is set to '1' in tcp_ip_feature_bit_map

```
at+rsi_opermode=<oper_mode>,<feature_bit_map>,<tcp_ip_feature_
bit_map>,<custom_feature_bitmap><ext_tcp_ip_feature_bit_map>\r
\n
```

if BIT(31) is set to '1' in both custom_feature and ext_custom_feature bit maps

```
at+rsi_opermode=<oper_mode>,<feature_bit_map>,<tcp_ip_feature_
bit_map>,<custom_feature_bitmap><ext_custom_feature_bit_map><b
le_feature_bit_map>\r\n
```

if BIT(31) is set to '1' in tcp_ip_feature_bit_map, custom_feature_bit_map and ext_custom_feature_bit_map

```
at+rsi_opermode=<oper_mode>,<feature_bit_map>,<tcp_ip_feature_
bit_map>,<custom_feature_bitmap><ext_custom_feature_bit_map><b
le_feature_bit_map><ext_tcp_ip_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   ble_feature_bit_map;
      uint32   ext_tcpip_feature_bit_map;
} 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 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 Access Point mode, a maximum of 8 client devices are supported.

8 - PER Mode (RF test mode). This mode is used for calculating packet error rate and used during Radio certification/complaince tests (e.g. ETSI, FCC, TELEC).

9 – Concurrent mode (Station + AP). This mode is used to run module in concurrent mode. In concurrent mode, host can connect to an AP and can create AP simultaneously. Inthis mode, maximum of 4 Station devices can be connected to Concurrent AP.

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.

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

Table 11: 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

Note:

To select correct Coex mode and possible features in particular mode please refer

WiSeConnect_TCPIP_Feature_Selection_v1.7.9.xlsx 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-v1.7.9.pdf

BLE card ready frame is described in

RS9113-WiseConnect-BLE-Software-PRM-API-Guide-v1.7.9.pdf

and ZigBee card ready frame is described

RS9113-WiseConnect-ZigBee-Software-PRM-API-Guide-v1.7.9.pdf

`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 selectmodule 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[13]`– Reserved

`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

Note:

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[18]- To enable RAW Socket feature`

`0 - To disable RAW Socket`

`1 - To Enable RAW Socket`

`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[30]- To enable OTAF(On The Air Firmware) upgradation.`

`tcp_ip_feature_bit_map[31]-` This bit is used to enable the tcp_ip extended feature bitmap.

- 1 – To enable Extended tcp_ip feature bitmap
- 0 – To disable Extended tcp_ip feature bitmap

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_v1.7.9.xlsx

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 specialized 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
- 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

Note:

This is not applicable for BT/BLE.

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

This bit has to be enabled, when we are using certificates. If it is not used, with the expired certificates also connection will get success.

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 extension 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 feature.

- 1 – Enable antenna diversity feature
- 0 – Disable 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 routine 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

BIT[3]: This bit is used to enable SSL certificate with 4096 bit key support

- 1 – Enable 4096 bit key support for SSL sockets
- 0 – Disable 4096 bit key support for SSL sockets

BIT[4]: This bit is applicable only in AP and concurrent AP mode. If this bit is set, module will send broadcast data (if any) immediately without waiting for DTIM.

- 1 – Enable sending broadcast without waiting for DTIM
- 0 – AP sends broadcast packet at DTIM only

Note:

If this bit is enable then connected client who is in power save may miss the packet.

BIT[9]: EXT_HTTP_SKIP_DEFAULT_LEADING_CHARACTER

To skip default leading character ("\") in resource name

ext_tcp_ip_feature_bit_map:

BIT[1]: This bit is used to enable DHCP USER CLASS Option

- 1 - Enable DHCP USER CLASS
- 0 - Disable DHCP USER CLASS

BIT[2]: To bypass the HTTP servers default root configuration page

- 1 - To enable HTTP server root path bypass option
- 0 - To disable HTTP server root path bypass option

BIT [3]: Toreturn IPv4/IPv6 addresses as part of AT_RSI_GOPARMS? response

- 1 - To returnboth IPv4 and IPv6 addresses as part of AT_RSI_GOPARMS? response
- 0 - To return only IPV4 address as part of AT_RSI_GOPARMS? response.

BIT [4]: This bit used to enable or disable passing of parameters in URL.

- 1-Enable
- 0 - Disable

BIT [5]: This bit used to handle ACK number in re-transmission packet.

- 1- Enable
- 0 - Disable

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.

Possible error codes: Possible error codes are

0x0021,0x0025,0xFF73,0x002C,0xFF6E,0xFF6F,0xFF70,0xFFC5.

Example:

AT Mode:

When only oper_mode is given in command:

```
at+rsi_opermode=1\r\n
```

```
0x61   0x74   0x2B   0x72   0x73   0x69   0x5F   0x6F   0x70
0x65   0x72   0x6D   0x6F   0x64   0x65   0x3D   0x31   0x0D
0x0A
```

Response:

```
OK\r\n
```

```
0x4F 0x4B 0x0D 0x0A
```

When other parameters along with mode_val is given in opermode command:

```
at+rsi_opermode=1,1,2,0\r\n
```

```
0x61   0x74   0x2B   0x72   0x73   0x69   0x5F   0x6F   0x70
0x65   0x72   0x6D   0x6F   0x64   0x65   0x3D   0x31   0x2C
0x31   0x2C   0x32   0x2C   0x30   0x0D   0x0A
```

By giving above command module is configured in WFD mode with HTTP server enabled in open mode.

Response:

```
OK\r\n


0x4F 0x4B 0x0D 0x0A
```

8.2 Band

Description:

This command configures the band in which the module has to be configured.

Command Format:

AT Mode:

```
at+rsi_band=< bandVal >\r\n
```

Binary Mode:

```
typedef struct {
      uint8       bandVal;
} bandFrameSnd;
```

Command Parameters:

The valid values for the parameter for this command are as follows:

`bandVal:`

0– 2.4 GHz

1- 5 GHz

2- Dual band (2.4 Ghz and 5 Ghz).

Note:
Dual band is supported in station mode and WiFi Direct mode

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.

Possible error codes:

Possible error codes are 0x0005, 0x0021, 0x0025, 0x002C, 0x003c.

Relevance:

This command is relevant in all operating modes

Example:

AT Mode:

```
at+rsi_band=0\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x62  0x61  0x6E
0x64  0x3D  0x30  0x0D  0x0A
```

Response:

```
OK\r\n
```

```
0x4F 0x4B  0x0D 0x0A
```

8.3 WLAN Config

Description:

This command configures WLAN parameters.

Note:

Only RTS_THRESHOLD is supported.

Command:

AT Mode:

```
at+rsi_config=< config_type >,< config_val >\r\n
```

Binary Mode:

```
struct {
      uint8 config_type[2];
      uint8 config_value[2];
} configFrameSnd;
```

Command Parameters:

The valid values for the parameter for this command are as follows:

`Config_type:` 1 - To configure RTS threshold value

`Config_value:` [256-2346] - Range of rts threshold value

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.

Possible error codes:

Possible error codes are 0x005d, 0x0021, 0x005e.

Relevance:

This command is relevant in all operating modes.

Example:

AT Mode:

```
at+rsi_config=1,256\r\n


0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x63 0x6F 0x6E 0x66 0x69
0x67 0x3D 0x31 0x2C 0x32 0x35 0x36 0x0D 0x0A
```

Response:

```
OK\r\n


0x4F 0x4B  0x0D 0x0A
```

8.4 Set MAC Address

Description:

This command sets the module's MAC address. This command has to be issued before band command.

Command:

AT Mode:

```
at+rsi_setmac=< macAddr >\r\n
```

Binary Mode:

```
typedef  struct {
      uint8 macAddr[6];
}setMacAddrFrameSnd;
```

Command Parameters:

`macAddr` – Mac address to be set for module.

Note:

In concurrent mode, given MAC is applied to station mode and AP mode MAC address last byte will differ from station mode MAC. AP mode MAC address last byte will be one plus the station mode MAC address last byte given by host.

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.

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C.

Relevance:

This command is relevant in all operating modes.

Example:

AT Mode:

```
at+rsi_setmac=001122334455\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F 0x73 0x65 0x74 0x6D
0x61  0x63  0x3D  0x30  0x30  0x31  0x31 0x32 0x32 0x33 0x33
0x34  0x34  0x35  0x35  0x0D  0x0A
```

Response:

```
OK\r\n
```

```
0x4F 0x4B  0x0D 0x0A
```

8.5 Init

Description:

This command programs the module's Baseband and RF components and returns the MAC address of the module to the host.

Command:

AT Mode:

```
at+rsi_init\r\n
```

Binary Mode:
No Payload required.

Command Parameters:

No parameters

Response:
AT Mode:

Result Code	Description
OK<macAddress>	macAddress (6 bytes, Hex)
ERROR<Error code>	Failure

Binary Mode:

```
typedef struct {
   uint8 macAddress[6];
}rsi_initResponse;
```

Response Parameters:
macAddress: The MAC address of the module.

Possible error codes:
Possible error codes are 0x0021, 0x0025, 0x002C,0x0002.

Relevance:

This command is relevant in all operating modes

Example:
AT Mode:

```
at+rsi_init\r\n
```

```
0x61   0x74   0x2B   0x72   0x73   0x69   0x5F   0x69   0x6E
0x69   0x74   0x0D   0x0A
```

Response:

```
OK<MAC_Address>\r\n
OK 0x00 0x23 0xA7 0x13 0x14 0x15\r\n

0x4F  0x4B  0x00  0x23  0xA7  0x13  0x14  0x15  0x0D  0x0A
```

FOR CONCURRENT MODE:

Response:
AT Mode:

Result Code	Description
OK <macAddress1><macAddress2>	macAddress(6 bytes Hex) macAddress(6 bytes Hex)
ERROR<Error code>	Failure

Binary Mode:

```
typedef struct {
   uint8 macAddress1[6];
   uint8 macAddress2[6];
}rsi_initResponse;
```

Response Parameters:
`macAddress`: The MAC address for two interfaces of the module.

Possible error codes:
Possible error codes are 0x0021, 0x0025, 0x002C,0x0002.

Relevance:

This command is relevant in all operating modes

Example:
AT Mode:

```
at+rsi_init\r\n

0x61   0x74   0x2B   0x72   0x73   0x69   0x5F   0x69   0x6E
0x69   0x74   0x0D   0x0A
```

Response:

```
OK<MAC_Address>\r\n
```

```
OK 0x00 0x23 0xA7 0x13 0x14 0x15  0x00 0x23 0xA7 0x13 0x14
0x16\r\n
```

```
0x4F  0x4B  0x00  0x23  0xA7  0x13  0x14  0x15  0x00  0x23
0xA7  0x13  0x14  0x16  0x0D  0x0A
```

8.6 PER Mode

Description:

This command configures the PER (Packet Error Rate) /RF Test Mode in RS9113-WiSeConnect Module. This command should be issued after *Init* command.

Command Format:

AT Mode:

```
at+rsi_per=<per_mode_enable>,<power>,<rate>,<length>,<mode>,
<channel>,<rate_flags>,<aggr_enable>,<no_of_pkts>,<delay>\r\n
```

Binary Mode:

```
typedef struct {
      uint8       per_mode_enable[2];
      uint8       power[2];
      uint8       rate[4];
      uint8       length[2];
      uint8       mode[2];
      uint8       channel[2];
      uint8       rate_flags[2];
      uint8       reserved1[2];
      uint8       aggr_enable[2];
      uint8       reserved2[2];
      uint8       no_of_pkts[2];
      uint8       delay[4];
} perModeFrameSnd;
```

Command Parameters:

`per_mode_enable`: To enable or disable PER Mode.

- 1 – Enable
- 0 - Disable

`power`: To set Tx power in dbm. The valid values are from 2dbm to 18dbm for RS9113 module.

Note:

Tx power 127 is a magic word to automatically pick the max supported power. It depends on module and data rate, and it can vary between 14 dbm to 20 dbm.

`rate`: To set transmit data rate.

Data Rate (Mbps)	Value of rate
1	0
2	2
5.5	4
11	6
6	139
9	143
12	138
18	142
24	137
36	141
48	136
54	140
MCS0	256
MCS1	257
MCS2	258
MCS3	259
MCS4	260
MCS5	261
MCS6	262
MCS7	263

Table 12: PER Mode Data Rates

`length`: To configure length of the tx packet. Valid values are in the range of 24 to 1500 bytes in the burst mode and range of 24 to 260 bytes in the continuous mode

`mode`: transmit mode

0- Burst Mode

1- Continuous Mode

2- Continuous wave Mode (non modulation) in DC mode

3- Continuous wave Mode (non modulation) in single tone mode (center frequency -2.5MHz)

4- Continuous wave Mode (non modulation) in single tone mode (center frequency +5MHz)

Note:

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

i.e 1. Start Burst mode with intended `power` value and `channel` values

Pass any valid values for `rate` and `length`

2. Stop Burst mode

3. Start Continuous Wave mode.

`channel`: For setting the channel number in 2.4 GHz/5GHz .

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

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

Table 13: Channel Number and Frequencies for 20MHz Channel Width in 2.4GHz

Note:

To support PER mode in 12,13,14 channels, set region command has to be given by the host before PER command

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

Channel Numbers (5 GHz)	Center frequencies for 20MHz channel width(MHz)
36	5180
40	5200
44	5220
48	5240
52	5260
56	5280
60	5300
64	5320
100	5500
104	5520
108	5540
112	5560
116	5580
120	5600
124	5620
128	5640
132	5660
136	5680
140	5700
144	5720
149	5745
153	5765
157	5785

161	5805
165	5825

Table 14: Channel Number and Frequencies for 20MHz Channel Width in 5GHz

`rate_flags`: Rate flags contain short GI, Greenfield and channel width values. Various fields in rate flags are divided as specified below

Fields	Short GI	Greenfield	Channel Width	Reserved
Bits:	0	1	2 – 4	5 - 15

Table 15: Rate Flags

To enable short GI – set rate flags value as '1'

To enable Greenfield – set rate flags value as '2'

Channel width should to set to zero to set 20MHz channel width.

`Reserved1`: reserved bytes. This field can be ignored. Set '0'

`aggr_enable`: This flag is for enabling or disabling aggregation support .

Note:

Aggregation feature is supported only in burst mode. This field will be ignored in case of continuous mode.

`Reserved2`: reserved bytes. This filed can be ignored. Set '0'

`no_of_pkts`: This field is used to set the number of packets to be sent in burst mode. If the value given is 'n' then 'n' number of packets will be sent on air, after that transmission will be stopped. If this field is given as zero(0) then packets will be sent continuously until user stops the transmission. This field will be ignored in case of continuous mode

`delay`: This field is used to set the delay between the packets in burst mode. Delay should be given in micro seconds. i.e. if the value is given as 'n' then a delay of 'n' micro seconds will be added for every transmitted packet in the burst mode.

If this field is set to zero (0) then packets will be sent continuously without any delay. This field will be ignored in case of continuous mode.

Note:

`Reserved1, Reserved2` are available only in Binary mode

Only `per_mode_enable, power, rate, length, mode, channel, rate_flags` fields are valid. Remaining fields are not supported

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.

Possible error codes:
Possible error codes are 0x000A,0x0021, 0x0025, 0x002C.

Relevance:

This command is relevant when the module is configured in operating mode 8.

Example:

AT Mode:

To start transmit:

```
at+rsi_per=1,18,139,30,0,1,0,0,0,0\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x70   0x65   0x72
0x3D  0x31  0x2C  0x310x38  0x2C  0x310x330x39  0x2C  0x330x30
0x2C  0x30  0x2C  0x31  0x2C  0x30  0x2C  0x30  0x2C  0x30
0x2C  0x30  0x0D  0x0A
```

To stop transmit:

```
at+rsi_per=0\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x70   0x65   0x72
0x3D  0x30   0x0D  0x0A
```

Response:

```
OK\r\n
```

```
0x4F 0x4B  0x0D 0x0A
```

8.7 Antenna Selection

Description:

This command configures the antenna to be configured. RS9113-WiSeConnect provides two options – an inbuilt antenna and a uFL connector for putting in an external antenna. This

command should be issued after the *init* command. By default (and if the command is not issued at all), the inbuilt antenna is selected.

Command Format:

AT Mode:

at+rsi_antenna=<AntennaVal><Gain_2G>,<Gain_5G>,<antenna_path>,<antenna_type>\r\n

Binary Mode:

```
struct {
      uint8 AntennaVal;
      uint8 Gain_2G;
      uint8 Gain_5G;
      uint8 Antenna_path;
      uint8 Antenna_type;
} AntennaSelFrameSnd;
```

Command Parameters:

AntennaVal:

- 0– RF_OUT_2/Internal Antenna is selected
- 1–RF_OUT_1/uFL connector is selected.

Gain_2G:

Antenna gain offset in db for 2.4 GHz band and valid values are 0 to 10 .

This provides an estimate to the module about how much to offset the power. The output power of the module is inversely proportional to this value.

Gain_5G:

Antenna gain offset in db for 5 GHz band and valid values are 0 to 10. This provides an estimate to the module about how much to offset the power. The output power of the module is inversely proportional to this value.

Note:

Please note since it is only an estimate, we will have to adjust these values after some amount of trial and error.

Antenna_path:

This is used to send the path of antenna to the module.

- 1 – This is used for the place where internal antenna is placed
- 2 - This is used for the place where U.Fl/External antenna is placed

Antenna_type:

This is the type of antenna need to place at the antenna_path.

- 1 – Silicon Labs antenna (Default antenna)
- 2 - FRACTUS antenna
- 3 - MOLEX antenna

The configuration values are used by the module to attenuate the output transmit power and are calculated based on the selected antenna type for the corresponding path so that the regulatory requirements like FCC, ETSI, etc., are not violated.

2G Gain range	5G Gain range	Antenna Type
0 < Gain <= 0.99	0 < Gain <= 4.42	Type 1 (Silicon Labs Antenna)
0.99 < Gain <= 1.8	4.42< Gain <= 4.6	Type 2 (Fractus Antenna)
1.8 < Gain <= 3	4.6 < Gain <= 4.9	Type 3 (Molex Antenna)

If you do not select anything, default value is 1 for Antenna_Path and Antenna_Type. You will have to use the type parameter carefully. If the gain, radiation pattern and other parameters of your external antenna are the same as that of the Silicon Labs antenna, you may use the type as 'Redpine',otherwise see which gain range your antenna fits in and specify the type accordingly. If your antenna does not fit in either of the above, you can use the gain_2G and gain_5G values to offset.

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.

Possible error codes:
Possible error codes are 0x0025, 0x002C.

Relevance:

This command is relevant in all operating modes

Example:

AT Mode:

```
at+rsi_antenna=1,0,0,2,1 \r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x61  0x6E  0x74
0x65  0x6E  0x6E  0x61  0x3D  0x31 0x2C 0x30 0x2C 0x30 0x2C
0x32 0x2C 0x31 0x0D  0x0A
```

Response:

OK\r\n

```
0x4F 0x4B 0x0D 0x0A
```

8.8 Configure Wi-Fi Direct Peer-to-Peer Mode

Description:

This command is used to set the configuration information for Wi-Fi Direct mode. After receiving this command, the module scans for Wi-Fi Direct nodes. If any Wi-Fi Direct node is found, it will send the information to the Host.

Command Format:

AT Mode:

```
at+rsi_wfd=<GOIntent>,<deviceName>,<operChannel>,<ssidPostFix>
,<psk>\r\n
```

Binary Mode:

```
typdef struct {
      uint8  GOIntent[2];
      uint8  deviceName[64];
      uint8  operChannel[2];
      uint8  ssidPostFix[64];
      uint8  psk[64];
}configP2pFrameSnd;
```

Command Parameters:

`GOIntent`: This determines whether the device is intended to form a GO (group owner) or work as a Wi-Fi Direct Peer node. This value is used in the GO negotiation process, when the module negotiates with another Wi-Fi Direct Node on who would become the Group Owner. The valid range of values for this parameter is: 0 to 16. Higher the number, higher is the willingness of the module to become a GO[1]. If the number is between 0 and 15, a GO negotiation takes place. If the value is 16, the module forms an Autonomous GO without negotiating with any other device.

`deviceName`: This is the device name for the module. The maximum length of this field is 32 characters remaining bytes filled with 0x00. Another Wi-Fi Direct device would see this name when it scans for Wi-Fi Direct nodes.

[1] After the module becomes a GO in Wi-Fi Direct mode, it appears as an Access Point to client devices

`operChannel`: Operating channel to be used in Group Owner or Autonomous GO mode. The specified channel is used if the device becomes a GO or Autonomous GO. The supported channels can be any valid channel in 2.4GHz or 5GHz. If band_val=0 is used in the Band command, then a channel in 2.4 GHz should be supplied to this parameter. If band_val=1 or 2 is used in the Band command, then a channel in 5GHz should be supplied to this parameter. The valid values for this parameter are listed in the PER Mode section. '0' is NOT a valid value for this parameter.

`ssidPostFix`: This parameter is used to add a postfix to the SSID in WiFi Direct GO mode and Autonomous GO mode.

If the module becomes a Wi-Fi Direct Group Owner, it would have an SSID with "DIRECT-xy" prefixed to the ssidPostFix parameter. "xy" is any alpha numeric character randomly generated by the module after the GO negotiation process is over. Legacy Wi-Fi nodes (non Wi-Fi Direct) would see this SSID on scanning the device[2].

For example if the ssidPostFix is given as "WiSe", The SSID of the module in GO mode or Autonomous GO mode could be DIRECT-89WiSe. All client devices would see this name in their scan results.

Note:

ssidPostFix should be maximum of 23 bytes.

`psk`: Passphrase of a maximum length of 63 characters (a null character should be supplied to make it 64 bytes in the structure). This PSK is usedif the module becomes a GO owner. Remote clients should use this passphrase while connecting to the module when it is in GO mode.

Category	Primary ID	Sub Category	Sub ID
Computer	0x01	PC	0x01
		Server	0x02
		Media Centre	0x03
		Ultra-mobile PC	0x04
		Notebook	0x05
		Desktop	0x06
		Mobile Internet Device	0x07
		Net book	0x08
Input Device	0x02	Keyboard	0x01
		Mouse	0x02

[2] After the module becomes a GO in Wi-Fi direct mode, it appears as an Access Point to client devices.

Category	Primary ID	Sub Category	Sub ID
		Joystick	0x03
		Trackball	0x04
		Gaming controller	0x05
		Remote	0x06
		Touch screen	0x07
		Biometric Reader	0x08
		Barcode Reader	0x09
Printers, Scanners, Faxes and Copiers	0x03	Printer or Print Server	0x01
		Scanner	0x02
		Fax	0x03
		Copier	0x04
		All-in-one (Printer, Scanner, Fax, Copier)	0x05
Camera	0x04	Digital Still Camera	0x01
		Video Camera	0x02
		Web Camera	0x03
		Security Camera	0x04
Storage	0x05	NAS	0x01
Network Infrastructure	0x06	AP	0x01
		Router	0x02
		Switch	0x03
		Gateway	0x04
Displays	0x07	Television	0x01
		Electronic Picture Frame	0x02
		Projector	0x03

Category	Primary ID	Sub Category	Sub ID
		Monitor	0x04
Multimedia Devices	0x08	DAR	0x01
		PVR	0x02
		MCX	0x03
		Set-top box	0x04
		Media Server/Media Adapter/Media Extender	0x05
		Portable Video Player	0x06
Gaming Devices	0x09	Xbox	0x01
		Xbox360	0x02
		Play station	0x03
		Game Console/Game Console Adapter	0x04
		Portable Gaming Device	0x05
Telephone	0x0A	Windows Mobile	0x01
		Phone-single mode	0x02
		Phone-dual mode	0x03
		Smartphone-single mode	0x04
		Smartphone- dual mode	0x05
Audio Devices	0x0B	Audio tuner/receiver	0x01
		Speakers	0x02
		Portable Music Player	0x03
		Headset	0x04
		Headphones	0x05
		Microphone	0x06

Category	Primary ID	Sub Category	Sub ID
Others	0xFF		

Table 16: Wi-Fi Direct Device Types

Response:

AT Mode:

Result Code	Description
OK	Successful execution of the command
AT+RSI_WFDDEV=<devState><devName><macAddres><devtype>	Asynchronous Message from module to Host, sent when module founds any Wi-Fi Direct node.
AT+RSI_CONNREQ< devName >	Asynchronous message from Module to Host, sent when module receives a connection request from any remote Wi-Fi Direct node.
ERROR<Error code>	Failure

Binary Mode:

There is no response payload for this command.

After the command is received by the module, it scans for Wi-Fi Direct devices. Whenever it finds any devices, it raises an asynchronous interrupt. The Host should perform an Rx Operation on receiving this interrupt. On performing the Rx operation, host receives the code for Async WFD (refer to the Binary Command mode section for more details) and the below structure as the Payload.

Response Parameters:

```
typedef struct {
      uint8  devState;
      uint8  devName[32];
      uint8  macAddress[6];
      uint8  devtype[2];
}rsi_wfdDevInfo;
```

After the command is received, the device is scanned by other Wi-Fi Direct devices too. If any of those devices send a connect request to the module, it raises an asynchronous interrupt. The Host should perform an Rx Operation on receiving this interrupt. On

performing the Rx operation, host receives the code for Async CONNREQ (refer to the Binary Command Mode section for more details) for the Response ID and the below structure as the Payload.

```
typedef struct
{
      uint8  dev_name[32];
}rsi_ConnAcceptRcv;
```

Command Parameters:

`devstate`: State of the remote Wi-Fi Direct node.

- 0– The remote Wi-Fi Direct node was found in previous scan iteration
- 1– A new remote Wi-Fi Direct node has been found

`devName`: Device name of the remote Wi-Fi Direct node, returned in ASCII. The length is 32 bytes. If the device name of the remote node is less than 32 bytes, 0x00 is padded to make the length 32 bytes.

`macAddress`: MAC ID of the remote Wi-Fi Direct node. Returned in Hex

`devType`: Type of the device, returned in two Hex bytes. The first byte returned is the primary ID and the second byte is the sub-category ID. Refer to the Configure Wi-Fi Direct Peer-to-Peer Mode section for more details.

When scanned WFD devices are moved out of range or powered off, the device lost indication will be given to host using the asynchronous (Async WFD) message from module to host.

`device_state`: 0 – The remote Wi-Fi Direct node was found in the previous scan iteration

`device_name`: All are 0x00's

`device_mac`: MAC ID of the remote Wi-Fi Direct node which is moved out of range.

`device_type`: All are 0x00's

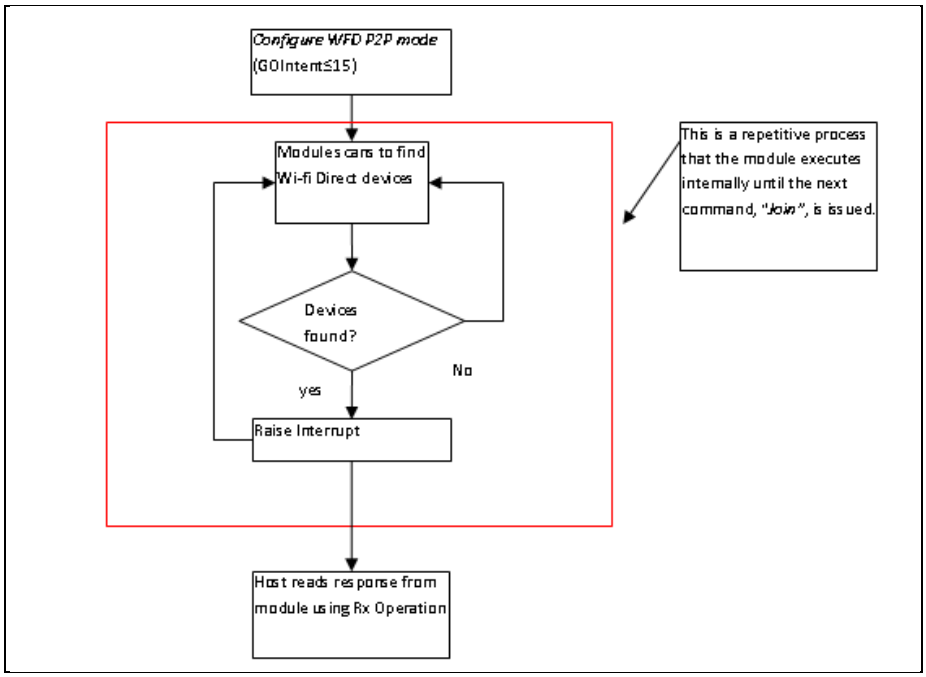


Figure 38: Operation after issuing "configure WFD P2P Mode" 2P Mode" command

Possible error codes:

Possible error codes are 0x001D, 0x0021, 0x0025, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 1.

Note:

After getting the connect request from remote device, host needs to issue the join command with the remote device name from which the connect request is received. Host needs to make sure that the remote device is scanned by module too (Async WFD with remote device name). If the host issues join command before remote device gets scanned by the module, then host will get join response with error code “25”.

Example:

AT Mode:

```
at+rsi_wfd =7,redpine,11, test,012345678\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x77  0x66  0x64
0x3D  0x37  0x2C  0x72  0x65  0x64  0x70  0x69  0x6E  0x65
0x2C  0x31  0x31  0x2C  0x74  0x65  0x73  0x74  0x2C  0x30
0x31  0x32  0x33  0x34  0x35  0x36  0x37  0x38  0x0D  0x0A
```

Response:

```
OK\r\n


0x4F 0x4B 0x0D  0x0A


AT+RSI_WFDDEV=1 wi_fi_phone0x00 0x23 0x12 0x13 0x14 0x160x0A
0x04  0x0D  0x0A


0x41  0x54  0x2B  0x52  0x53  0x49  0x5F  0x57  0x46  0x44
0x44  0x45  0x56  0x3D  0x31  0x77  0x69  0x5F  0x66  0x69
0x5F  0x70  0x68  0x6F  0x6E  0x65  0x00  0x00 0x00  0x00 0x00
0x00 0x00  0x00 0x00  0x00 0x00  0x00 0x00  0x00 0x00  0x00
0x00  0x00 0x00  0x00  0x000x01 0x02 0x03 0x04 0x05 0x06 0x0A
0x04   0x0D  0x0A
```

When scanned WFD devices are moved out of range or powered off, the device lost indication will be given to host using the asynchronous message AT+RSI_WFDDEV from module to host.

```
AT+RSI_WFDDEV=<1byte-NULL><32 bytes -NULL>0x00 0x23 0x12 0x13
0x14 0x16<2bytes-NULL>  0x0D  0x0A


0x41  0x54  0x2B  0x52  0x53  0x49  0x5F  0x57  0x46  0x44
0x44  0x45  0x56  0x3D  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x000x01 0x02 0x03 0x04
0x05  0x06 0x00  0x00   0x0D  0x0A
```

8.9 Configure AP Mode

Description

This command is used to set the configuration information for AP mode

Payload

AT Mode:

```
at+rsi_apconf = <channel_no>,<ssid>,<security_type>,<
encryp_mode>, <psk>,<beacon_interval>,<dtim_period>,<
max_sta_support>,<ap_keepalive_type>,<ap_keepalive_period>\r\n
```

Binary Mode:

```
struct {
```

```
        uint8   channel_no[2];
        uint8   ssid[34];
        uint8   security_type;
        uint8   encryp_mode;
        uint8   psk[64];
        uint8   beacon_interval[2];
        uint8   dtim_period[2];
        uint8   ap_keepalive_type;
        uint8   ap_keepalive_period;
        uint8   max_sta_support[2];
}rsi_apconfig;
```

Parameters

`channel_no`: The channel in which the AP would operate. Refer the PER Mode section . A value of '0' is not allowed.

Note:
DFS channels are not supported in ap mode.

`ssid`: SSID of the AP to be created

`security_type`: Security type.

0-Open

1-WPA

2-WPA2

`encryp_mode`: Encryption type.

0-Open

1-TKIP

2-CCMP

`psk`:PSK of the AP in security mode. If the AP is in Open mode, this parameter can be set to '0'.

Note:
Minimum and maximum length of PSK is 8 bytes and 63 bytes respectively.

`beacon_interval` : Beacon interval of the AP in milliseconds. Allowed values are integers from 100 to 1000 which are multiples of 100.

`dtim_period` : DTIM period. Allowed values are from 1 to 255.

`ap_keepalive_type`: This is the bitmap to enable AP keep alive functionality and to select the keep alive type.

`BIT[0]`: To enable/disable keep alive functionality.

- 1 - To enable keep alive functionality.
- 0 - To disable keep alive functionality.

`BIT[1]`: To select AP keep alive method.

- 1 - To enable null data based keep alive functionality.
- 0 -Default keep alive functionality (i.e disconnect the station if there is no wireless exchanges from station with in ap_keepalive_period).

`ap_keepalive_period`: This is the period after which AP will disconnect the station if there are no wireless exchanges from station to AP.Keep alive period is calculated in terms of 32 multiples of beacon interval(i.e if there are no wireless transfers from station to AP with in (32*beacon_interval*keep_alive_period) milli seconds time period,station will be disconnected).If null data based method is selected,AP checks the connectivity of station by sending null data packet. If station does not ack the packet ,that station will be disconnected after 4 retries.

`max_sta_support`: Number of clients supported. This value should be less than or equal to the value given in custom feature select bit map [BIT [13:16]] of theSet Operating ModeSet Operating Mode. If value is not set in custom feature select bit map[BIT[13:16]] of theSet Operating Modethen maximum supported stations are 4.

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.

Possible error codes
Possible error codes are 0x0021,0x0025,0x002C,0x0026,0x004C,0x0028,0x001A,0x000A,0x001D

Relevance

This command is relevant when the module is configured in Operating Mode 6.

Example:

AT Mode:

Configured AP with channel num =11, ssid = redpine, open mode, beacon interval = 100, DTIM count =3 and max stations support =3.

```
at+rsi_apconf=11,redpine,0,0,0,100,3,3\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x61  0x70  0x63
0x6F  0x6E  0x66  0x3D  0x31  0x31  0x2C  0x72  0x65 0x64
0x70  0x69  0x6E  0x65  0x2C  0x30  0x2C  0x30  0x2C  0x30
0x2C  0x31  0x30  0x30  0x2C  0x33  0x2C  0x33  0x0D  0x0A
```

Response:

```
OK\r\n
```

```
0x4F  0x4B  0x0D   0x0A
```

8.10 WPS PIN Method

Description:

This command configures the WPS PIN method to be used in RS9113-WiSeConnect Module. This command should be issued before join command.

Command Format:

AT Mode:

```
at+rsi_wps_method=<wps_method>,<generate_pin>,<wps_pin>\r\n
```

Binary Mode:

```
#define RSI_WPS_PIN_LEN  8
typedef struct {
      uint8 wps_method[2];
      uint8 generate_pin[2];
      uint8 wps_pin[RSI_WPS_PIN_LEN];
}wpsMethodFrameSnd;
```

Command Parameters:

`wps_method`: WPS method type Should set to '1' for PIN method.

`generate_pin`: This parameter specifies whether to validate entered pin or generate pin .This parameter is valid only if `wps_method` is 1.

- 0-Use entered pin in wps_pin field.
- 1-pin generation

If `generate_pin` is 0, module will validate the given 8 digit `wps_pin`. If pin given is less than 8 digit or if pin is wrong then module will give error.

wps_pin:wps_pinis of 8 digits pin. Module validates and uses this pin only in case of when wps_method is pin method and generate_pin is 0.

Response:

Note:

Response contains following payload only if PIN method is selected. In case of PUSH method response does not contains any payload.

AT Mode:

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

Binary Mode:

```
typedef struct {
      uint8 wps_pin[8];
}rsi_wpsMethodFrameRecv;
```

Response Parameters:

wps_pin: The WPS PIN will be used by the module to connect with WPS AP.

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C, 0x0037, 0x0038.

Relevance:

This command is relevant when the module is configured in Operating Mode 0 and 6.

Example:

AT Mode:

When PIN of length 8 is given

```
at+rsi_wps_method=1,0,12345678\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x77  0x70  0x73
0x5F 0x6D  0x65  0x74  0x68  0x6F  0x64  0x3D  0x31  0x2C
0x30  0x2C 0x310x320x330x340x350x360x370x38  0x0D  0x0A
```

Response:

```
OK 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08\r\n
```

```
0x4F 0x4B 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x0D 0x0A
```

When PIN of length less than 8 is given

```
at+rsi_wps_method=1,1,1234\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x77  0x70  0x73
0x5F 0x6D  0x65  0x74  0x68  0x6F  0x64  0x3D  0x31  0x2C
0x30  0x2C 0x310x320x330x34 0x0D  0x0A
```

8.11 Scan

Description:

This command scans for Access Points and gives the scan results to the host. The scan results are sorted in decreasing order of signal strength (RSSI value). The scanned access point with highest signal strength will be the first in the list.

Command Format:

AT Mode:

```
at+rsi_scan=<channel>,<ssid>,<channel_bit_map_2_4>,<channel_bi
t_map_5>\r\n
```

or

```
at+rsi_scan=<channel>,<ssid>,<scan_feature_bitmap>\r\n
```

Binary Mode:

```
#defineRSI_SSID_LEN 34
typdef struct {
      uint8 channel[4];
      uint8 ssid[RSI_SSID_LEN];
      uint8 reserved[5];
      uint8 scan_feature_bitmap;
      uint8 channel_bit_map_2_4[2];
      uint8 channel_bit_map_5[4];
} scanFrameSnd;
```

Command Parameters:

`Channel`: Channel Number on which scan has to be done. If this value is 0, the module scans in all the channels in the band that is selected through the band command. The values of this parameter are listed in table below3.

Note:

[3] To select DFS channels user need to set custom feature bit in oper mode command.

1) If chan_num is 0 and channel bit maps (selective scan) are provided then module will scan only the channels specified in bitmaps instead of scanning all channels.
2) In case of 5GHz, module performs passive scan in DFS channels only when BIT[8] is set in custom feature bit map in Set Operating Mode command.

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

Table 17: Channel in 2.4 GHz Mode

Note:

Scanning in 12,13,14 channels is allowed based on the region selected inSet Region command.

Channel Number	chan_num parameter

Channel Number	chan_num parameter
All channels	0
36	36
40	40
44	44
48	48
52	52
56	56
60	60
64	64
100	100
104	104
108	108
112	112
116	116
120	120
124	124
128	128
132	132
136	136
140	140
149	149
153	153
157	157
161	161
165	165

Table 18: Channel in 5GHz Mode

ssid: Optional Input. For scanning a hidden Access Point, its SSID can be provided as part of the SCAN command. The maximum number of scanned networks reported to the host is 11. If not used, null characters should be supplied to fill the structure.

Reserved: Set to '0's.Only present in Binary mode.

scan_feature_bitmap: Scan feature bitmap

BIT[0]: To enable/disable quick scan feature.

- 1 - To enable quick scan feature.
- 0 - To disable quick scan feature.BIT[1]-BIT[7]: Reserved.

Note:

scan feature bitmap is valid only if channel number and ssid is given.

If the channel number is not specified then scan command will take as channel_bit_map_2_4 =1 i.e only channel 1 scan results will be provided to host.

Channel number=0 is not valid for a quick scan.

When band=1 (5GHz), channel_bit_map_2_4 is not considered. Only channel_bit_map_5 is considered

-

channel_bit_map_2_4: channel bitmap for scanning in set of selective channels in 2.4Ghz.

Channel_bit_map_5: channel bitmap for scanning a set of selective channels in 5Ghz.

Channel Number	Channel bit position in bitmap
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10

Channel Number	Channel bit position in bitmap
12	11
13	12
14	13

Table 19: Channel Number to Bitmap Mapping in 2.4GHz

Channel Number	chan_num parameter
36	0
40	1
44	2
48	3
52	4
56	5
60	6
64	7
100	8
104	9
108	10
112	11
116	12
120	13
124	14
128	15
132	16
136	17
140	18
149	19
153	20

Channel Number	chan_num parameter
157	21
161	22
165	23

Table 20: Channel number to bitmap mapping in 5GHz

Note:

Channel bit maps , e.g. `channel_bit_map_2_4 and channel_bit_map_5 used for background scan.`

Response:

AT Mode:

Result Code	Description
OK< scanCount>< padding> < rfChannel>< securityMode>< rssiVal>< uNetworkType>< ssid><bssid>< reserved> .....up to the number of scanned nodes	Successful execution of the command.
ERROR<Error code>	Failure

Binary Mode:

```
struct{
      uint8  rfChannel;
      uint8  securityMode;
      uint8  rssiVal;
      uint8  uNetworkType;
      uint8  ssid[34];
      uint8  bssid[6];
      uint8  reserved[2];
}rsi_scanInfo;
```

```
#define RSI_AP_SCANNED_MAX 11
typedef struct {
      uint8    scanCount[4];
      uint8    padding[4];
      rsi_scanInfo      strScanInfo[RSI_AP_SCANNED_MAX];
} rsi_scanResponse;
```

Response Parameters:

`Scancount(4 bytes)`: Number of Access Points scanned

`padding(4 bytes)`: padding bytes which can be ignored.

`rfChannel(1 byte)`: Channel Number of the scanned Access Point

`securityMode(1 byte)`:

0–Open

1–WPA

2–WPA2

3-WEP

4–WPA Enterprise

5–WPA2 Enterprise

`Rssival(1 byte)`: RSSI of the scanned Access Point

`uNetworkType(1 byte)`: Network type of the scanned Access Point

1– Infrastructure mode

`Ssid(34 bytes)`: SSID of the scanned Access Point

`Bssid(6 bytes)`: MAC address of the scanned Access Point.

`Reserved(2 bytes)`: Reserved bytes.

Possible error codes:
Possible error codes are 0x0021, 0x0025, 0x002C, 0x0014, 0x0002, 0x0003, 0x0024, 0x001A, 0x0015, 0x000A, 0x0026

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 2, 6,9.

Example:
AT Mode:

To scan all the networks in all channels

```
at+rsi_scan=0\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x73  0x63  0x61
0x6E  0x3D 0x30  0x0D 0x0A
```

To scan a specific network “Test_AP” in a specific channel 6

```
at+rsi_scan=6,Test_AP\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x73  0x63  0x61
0x3D 0x36 0x2C 0x54 0x65 0x73 0x74 0x5F 0x41 0x50  0x0D 0x0A
```

Note:

1. Sometimes, there are a lot of Wi-Fi Access Points at a location, and the required network to which the device must connect may not appear in the top of the list. The recommendation in these cases is to issue a scan request directed only to the required network by specifying its name (SSID) or do channel specific scan.

2. And if there are several APs (BSSIDs) with that SSID, the WiSeConnect module presents only the one with the strongest RSSI.

Response:

If two networks are found with the SSID “Redpine_net1” and “Redpine_net2”, in channels 6 and 10, with measured RSSI of -20dBm and -14dBm respectively, the return value is

```
O K <scanCount =2>< padding>< rfChannel =0x0A><securityMode
=0x02>< rssiVal =14>< uNetworkType =0x01>< ssid=Redpine_net2><
bssid =0x00 0x23 0xA7 0x1F 0x1F 0x15><reserved >< rfChannel
=0x06>< securityMode =0x00>< rssiVal =20>< uNetworkType
=0x01>< ssid =Redpine_net1>< bssid =0x00 0x23 0xA7 0x1F 0x1F
0x14>< reserved >  \r\n
```

```
0x4F    0x4B    0x02    0x00  0x00    0x00    0x00    0x00
0x00    0x00    0x0A0x020x0D0x010x52    0x65    0x64    0x70
0x69    0x6E  0x65    0x5F    0x6E    0x74  0x32    0x00
0x00    0x00  0x00    0x00    0x00    0x00  0x00    0x00
0x00    0x00  0x00    0x00    0x00    0x00  0x00    0x00
0x00    0x00  0x00    0x000x00    0x23    0xA7    0x1F    0x1F
0x150x00    0x00  0x060x000x140x010x52    0x65  0x64    0x70
0x69    0x6E  0x65    0x5F    0x6E    0x74  0x31    0x00
0x00    0x00  0x00    0x00    0x00    0x00  0x00    0x00
0x00    0x00  0x00    0x00    0x00    0x00  0x00    0x00
0x00    0x00  0x00    0x000x00    0x23  0xA7    0x1F    0x1F
0x140x00    0x00    0x0D    0x0A
```

8.12 Join

Description:

This command is used for following:

1) Associate to an access point (operating mode = 0 or 2)
2) Associate to a remote device in Wi-Fi Direct mode or to create autonomous GO(operating mode 1)
3) Create an Access Point (operating mode 6)
4) Allow a third party to associate to a Wi-Fi Direct group created by the module (operating mode 1)
5) To enable WPS PUSH method in Access point mode

Command Format:

AT Mode:

```
at+rsi_join=<ssid>,<dataRate>,<powerLevel>,<Security_mode>,<jo
in_feature_bitmap>,<listen_interval>,<vap_id>,<join_bssid>\r\n
```

Binary Mode:

```
struct {
      uint8  reserved1;
      uint8  Security_mode;
      uint8  dataRate;
      uint8  powerLevel;
      uint8  psk[64];
      uint8  ssid[34];
      uint8  join_feature_bitmap;
      uint8  reserved2[2];
      uint8  ssid_len;
      uint32 listen_interval;
      uint8  vap_id;
      uint8  join_bssid[6];
} joinFrameSnd;
```

Command Parameters:

`reserved1`: Reserved. Set to '0'

`Security_mode`: This variable is used to define the security mode of the Access point to which module is supposed to connect.

Possible values:

- 0 – Connect only to AP in open mode
- 1 - Connect to AP in WPA mode
- 2 - Connect to AP in WPA2 mode
- 3 – Connect to AP in WEP open mode
- 4 – Connect to AP in EAP WPA mode
- 5 – Connect to AP in EAP WPA2 mode
- 6 - Connect to AP either in WPA/WPA2 mode (Mixed mode)

(Gives priority to WPA2 configured AP)

Note:

1) `Security_mode` parameter is valid only if opermode is 0 or 2.
2) `psk` is required for security mode 1,2,6. Otherwise module returns Join failure with error 0x16.
3) In Enterprise mode(`Security_mode` 4,5), module will derive the PSK using EAP exchanges with Authentication server.
4) Module strictly obey security mode specified in Join command, not depends on `psk`.
5) In opermode 6, Once Access point is created host can enable WPS PUSH method by giving JOIN command(with same parameters which were used to create Access point) again.
6) `psk, reserved2, ssid_len` fields are present only in Binary mode.
7) WPS method is not supported in Coex mode .

`dataRate`: Transmission data rate. Physical rate at which data has to be transmitted.

Data Rate (Mbps)	Value of dataRate
Auto-rate	0
1	1
2	2
5.5	3
11	4
6	5
9	6
12	7
18	8
24	9
36	10

Data Rate (Mbps)	Value of dataRate
48	11
54	12
MCS0	13
MCS1	14
MCS2	15
MCS3	16
MCS4	17
MCS5	18
MCS6	19
MCS7	20

Table 21: Transmission Data Rates

`powerLevel`: This fixes the Transmit Power level of the module. This value can be set as follows:

At 2.4GHz

0– Low power (7+/-1) dBm

1– Medium power (10 +/-1) dBm

2– High power (18 + /- 2) dBm

At 5 GHz

0– Low power (5+/-1) dBm

1– Medium power (7 +/-1) dBm

2– High power (12 +/- 2) dBm

Note:

1. If join_feature_bitmap BIT[3] is set, then notion of powerLevel will change. powerLevel keeps absolute value of the max supported power given by host (it can be upto 20 dBm)

`psk`: Passphrase used in WPA/WPA2-PSK security mode.

In open mode, WEP mode, Enterprise Security and Wi-Fi Direct modes, this should be filled with NULL characters.

`Ssid`: When the module is in Operating modes 0 or 2, this parameter is the SSID of the Access Point (assuming WPS is not enabled in the Access Point).

When the module is in operating modes 0 or 2, and wants to connect to an access point in WPS mode then the value of this parameter is NULL .

In Wi-Fi Direct mode, this parameter is the device name of the remote P2P node to which the module wants to associate.

When an Access Point needs to be created, this parameter should be the same as the parameter ssid in the command “Configure AP mode”.

In Wi-Fi Direct mode, when the module is a Group Owner and already connected to a Wi-Fi Direct node; and another Wi-Fi node wants to join, then this parameter is module’s device name.

Note:

Comma(",") is not supported in the SSID name in the AT commands. Other than comma, you can use any special characters.

`Reserved2`: Reserved, set to ‘0’

`ssid_len`: Actual length of the SSID

`join_feature_bitmap`:

`BIT[0]`: To enable b/g only mode in station mode, host has to set this bit.

- `0` – b/g/n mode enabled in station mode
- `1` – b/g only mode enabled in station mode

`BIT[1]`: To take listen interval from join command.

- `0` – Listen interval invalid
- `1` – Listen interval valid

`BIT[2]`: To enable/disable quick join feature.

- 1 - To enable quick join feature.
- 0 - To disable quick join feature.

`BIT[3]`: To enable CCXV2 need to set this bit. This is valid only for operating mode 2.

- 1 – Enable CCXV2 feature.
- 0 – Disable CCXV2 feature.

`BIT[4]`: To connect to AP based on BSSID together with configured SSID this feature need to enable.

- 1 – Enable BSSID based join.
- 0 – Disable BSSID based join.

`BIT[5]-BIT[7]`: Reserved.

`listen_interval`: This is valid only if BIT(1) in join_feature_bit_map is set. This value is given in Time units(1024 microsecond). This parameter is used to configure maximum sleep duration in power save.

`Vap_id`: Possible values are 0 and 1.

When 0 – Module will try to connect to scanned AP.

When 1 – Module will create AP.

`join_bssid`: This contains BSSID of selected AP. This is valid only if `join_feature_bitmap` BIT[4] is set otherwise module will ignore the value.

Note:

1. vap_id will be considered only in concurrent mode.

2.In concurrent mode, if connected station network is same as default dhcp server network then dhcp server will not start but join command for AP creation will give success message to host.

Response:

AT Mode:

Result Code	Description
OK<GO status>	Successful execution of the command. GO_Status (1 byte, hex): 0x47 (ASCII "G") – If the module becomes a Group Owner (GO) after the GO negotiation stage, or becomes an Access Point. 0x43 (ASCII "C") – If the module does not become a GO after the GO negotiation stage, or becomes a client (or station). Note: The module gets a default IP of 192.168.100.76 if it becomes a Group Owner or Access Point in case of IPv4. and gets a default IP of 2001:db8:0:1:0:0:0:120 in case of IPv6.
ERROR<Error code>	Failure

Binary Mode:

Response Payload:

```
struct {
    uint8 operState ;
}rsi_joinResponse ;
```

Response Parameters:

operState: 0x47 – if the module becomes a Group Owner (GO) after the GO negotiation stage. 0x43 – if the module does not become a GO after the GO negotiation stage.

This parameter should be used by the Host when the module is configured as a Wi-Fi Direct node within Operating mode 1 (refer Configure Wi-Fi Direct Peer-to-Peer Mode (Configure P2P)). Note: The module gets a default IP of 192.168.100.76 if it becomes a Group Owner or Access Point in case of IPv4 and gets default IP of 2001:db8:0:1:0:0:0:120 in case of IPv6.

Possible error codes:

Possible error codes are 0x0002,0x0004, 0x0006,0x0008, 0x0009, 0x000E,0x0015, 0x0016, 0x0018, 0x0019, 0x001E, 0x0020, 0x0021, 0x0023, 0x0025, 0x0026, 0x0028,0x002A, 0x002B, 0x002C, 0xFFF8,0x0033,0x0040,0x0042,0x0043,0x0044, 0x0045,0x0046,0x0047, 0x0048, 0x0049, 0x004B

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 6,9. When the module is in Operating Mode 1, this command initiates a GO negotiation and subsequent association to a Wi-Fi Direct node. In Operating mode 0, it initiates a security authentication and association process with an Access Point.s

8.13 Request timeout

Description:

This command is used to set various timeouts. Currently we can use this command to set the authentication and association request timeouts.

Command Format:

AT Mode:

```
at+rsi_timeout=<timeout_bitmap>,<timeout _value>\r\n
```

Binary Mode:

```
struct request_timeout{
 uint32 timeout_bitmap;
 uint16 timeout_value;
};
```

Command Parameters:

`timeout_bitmap:`

`BIT[0]`: sets timeout for association and authentication request .

`timeout_value` : timeout value in ms(default 300ms).

`BIT[1]`: Sets the each channel active scan time in ms (default 100ms).

`BIT[2]`: Reseverd(Keep alive timeout in RS9116)

`BIT[3]:set timeout for unicast probe timeout`

Example:

AT Mode:

To set authentication and association request timeout of 1.5 seconds

```
at+rsi_timeout=1,1500\r\n
```

Possible error codes:

0x00B0

8.14 Re-Join

Description:

The module automatically tries to re-join if it loses connection to the network it was associated with. If the re-join is successful, then the WLAN link is re-established. During the time the module is trying to re-join, if the Host sends any command, the module does not accept it and throws an error code 37 in status code. The module aborts the re-join after a fixed number of re-tries (maximum number of retries for rejoin is 20 by default). If this happens, an asynchronous message is sent to the Host with an error code 25. User can configure the rejoin parameters using rejoin command.

Note:

When Re-join fails module will close all prior opened TCP/IP sockets.

Command Format:

AT Mode:

```
at+rsi_rejoin_params=< rsi_max_try>,< rsi_scan_interval>,
< rsi_beacon_missed_count>,< rsi_first_time_retry_enabled>\r\n
```

Binary Mode:

```
typedef struct rsi_rejoin_params_s{
      uint8    rsi_max_try[4];
      uint8    rsi_scan_interval[4];
      uint8    rsi_beacon_missed_count[4];
      uint8    rsi_first_time_retry_enabled[4];
} rsi_rejoin_params_t;
```

Command Parameters:

rsi_max_try: This represents the number of attempt for join before giving up the error.

Note:

1.If number of rejoin attempts is 0 then module will try infinitely for rejoin.

2. In WPS mode rejoin will be used after connection with an AP.

rsi_scan_interval:

This is time interval in seconds for the subsequent retry.

rsi_beacon_missed_count:

This is the beacon missed count that module used to declare module connection status. If module found continuous beacon missed is greater than or equal to this value then it will declare connection as disconnected and will start rejoin process again.

rsi_first_time_retry_enabled:

If this is set to 1 then module will retry to connect if first join attempt fails. Number of attempts and scan interval may be configured by `rsi_max_try` and `rsi_scan_interval` respectively.

Note:

Default value for the `rsi_max_try`is 20, `rsi_scan_interval` is 5 second, `rsi_beacon_missed_count` is 40 and `rsi_first_time_retry_enabled` is 0.

Response:

AT Mode:

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

Binary Mode:

No response payload

Possible error codes:

Possible error codes are 0x0025, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 0,2.

Example:

AT Mode:

N/A

Response:

Asynchronous responses from module:

Following message to indicate that module is in process of rejoin, so unable to process requested command.

```
ERROR<Error code=37>\r\n
0x45 0x52 0x52 0x4F 0x52 0x25 0x00 0x0D 0x0A
```

```
Following message to indicate rejoin failure to host.
ERROR<Error code=25>\r\n
0x45 0x52 0x52 0x4F 0x52 0x19 0x00 0x0D 0x0A
```

8.15 WMM PS

Description:

This command is used to enable WMM PS configurations. This command should be issued before join command and before power save command.

Command Format:

AT Mode:

```
at+rsi_wmm_config=< wmm_ps_enable >< wmm_ps_type ><
wmm_ps_wakeup_interval >< wmm_ps_uapsd_bitmap >\r\n
```

Binary Mode:

```
struct {
      uint8 wmm_ps_enable[2];
      uint8 wmm_ps_type[2];
      uint8 wmm_ps_wakeup_interval[4];
      uint8 wmm_ps_uapsd_bitmap;
}wmmPsFrameSnd;
```

Command Parameters:

`wmm_ps_enable`: To enable or disable WMM PS

0 - Disable

1 - Enable

`wmm_ps_type`: WMM PS type

0 - Tx Based

1 - Periodic

`wakeup_interval`: Wakeup interval in milli seconds (range 1-100)

`wmm_ps_uapsd_bitmap`: Bitmap , 0 to 15 possible values.

`wmm_ps_uapsd_bitmap[0]` : Access category: voice

`wmm_ps_uapsd_bitmap[1]` : Access category: video

`wmm_ps_uapsd_bitmap[2]` : Access category: Back ground

`wmm_ps_uapsd_bitmap[3]` : Access category: Best effort U-APSD

`wmm_ps_uapsd_bitmap[4:7]` : All set to '0'. Don't care bits.

Parameters `wmm_ps_type`,`wakeup_interval`, `wmm_ps_uapsd_bitmap` will be used for WMM-PS if Power save is enabled and psp_type given as UAPSD.

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.

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 or 6.

Example:

AT Mode:

```
at+rsi_wmm_config=1,1,0,10\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x77  0x6D  0x6D
0x5F  0x70  0x73  0x3D  0x31  0x2C  0x31  0x2C  0x30  0x2C
0x31  0x30
0x0D  0x0A
```

Response:

```
OK\r\n
```

```
……..
0x4F   0x4B 0x0D    0x0A
```

8.16 Set Sleep Timer

Description:

This command configures the sleep timer mode of the module to go into sleep during power save operation. The command can be issued any time in case of power save mode 9. If this command is not issued, then by default module takes 3 seconds as sleep timer.

Command Format:

AT Mode:

```
at+rsi_sleeptimer=< TimeVal >\r\n
```

Binary Mode:

```
struct {
      uint8 TimeVal[2];
} SleepTimerFrameSnd;
```

Command Parameters:

`TimeVal:`

Sleep Timer value in seconds.

Minimum value is 1, and maximum value is 2100.

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.

Possible error codes:

Possible error codes are 0x0025, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 2.

8.17 Power Mode

Description:

This command configures the power save mode of the module. Power save is disabled by default. The command can be issued any time after the Join command in case of power save mode 1, 2 and 3.

And after Init command before join command in case of power save mode 8 and 9.

Note:

1. RS9113-WiSeConnect doesn't support power save modes while operating in AP or group owner mode.
2. Power save modes 2 and 8 are not supported in USB interface.
3. In SPI interface when ULP mode is enabled, after wakeup from sleep, host has to initialize SPI interface of the module.

Command Format:

AT Mode:

```
at+rsi_pwmode=< powerVal >,< ulp_mode_enable >,
<listen_interval_dtim>,<PSP_type>,<monitor_interval>\r\n
```

Binary Mode:

```
typdef union{
      struct {
            uint8 powerVal;
            uint8 ulp_mode_enable;
            uint8 listen_interval_dtim;
            uint8 PSP_type;
            uint16 monitor_interval;
      } powerFrameSnd;
      uint8 uPowerFrameBuf[6];
}
```

Command Parameters:

powerVal:

- 0–Mode 0: Disable power save mode
- 1–Power save Mode 1
- 2–Power save Mode 2
- 3–Power save Mode 3
- 8- Power save Mode 8
- 9- Power save Mode 9

ulp_mode_enable:

- 0 - Low power mode
- 1 - Ultra low power mode with RAM retention. Valid for powerVal modes 2,3,8
- and 9.
- 2 - Ultra low power mode without RAM retention. Valid for powerVal modes 8
- and 9.

listen_interval_dtim:

This parameter is valid only if BIT(1) is set in the join_feature_bitmap and valid listen interval is given in join command. If this parameter is set, the module computes the desired sleep duration based on listen interval (from join command) and its wakeup align with Beacon or DTIM Beacon (based on this parameter).

- 0 - Module wakes up before nearest Beacon that does not exceed the specified listen interval time.
- 1 - Module wakes up before nearest DTIM Beacon that does not exceed the specified listen interval time.

PSP_type: This parameter shows Power Save Procedure type used. Following is the values for the PSP_type.

- 0 – Max Power save procedure.
- 1 – Fast power save procedure.
- 2 – UAPSD power save

Note:

1. When fast psp is enabled, module will disable power save for monitor interval of time for each data packet received or sent.
2. UAPSD power save is valid only if wmm is enabled through wmm ps command

Monitor_interval: This is time in ms to keep module in wakeup state for each Tx or Rx traffic sent or received respectively. Default value for this is 50 ms.

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.

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFF8,0x0015,0x0026,0x0052

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 2.

8.17.1 Power save Operation

The behavior of the module differs as per the power save mode it is configured with.

The following terminology can be used in the below section in order to describe the functionality.

Protocol	Non Connected State	Connected State
WLAN	This mode is significant when module is not connected with any AP	This mode is significant when module is in associated state with AP
BT Classic	This mode is significant when module is in Idle (standby) state.	This mode is significant when module is in Connected sniff mode, Discoverable mode (ISCAN) and Connectable mode (PSCAN)
BLE	This mode is significant when module is in Idle (standby) state.	This mode is significant when module is in Advertising state, Scan state or Connected state.

Note:

1. In case of WLAN, wake up period will be calculated based on DTIM interval.
2. In case of BT-Classic, wake up period will be caluculated based on inquiry scan interval in discoverable mode, page scan interval in connectable mode and sniff interval in connected mode.

3. In case of BLE, wake up period will be caluculated based on advertise interval in advertising state, scan interval in scanning state and connection interval in connected state.

If incase BT/BLE wakeup period is lesser than the WLAN wakeup period, the module will wakeup and serves BT/BLE and go back to the sleep again.

8.17.1.1 Power save Mode 1

Once the module is configured to power save mode 1, it wakes itself up periodically based upon the DTIM interval configured in connected AP. In power mode 1, only the RF of the module is in power save while SOC continues to work normally. After successful execution of command, confirmation is received in response. This command has to be given only when module is in connected state (with the AP).

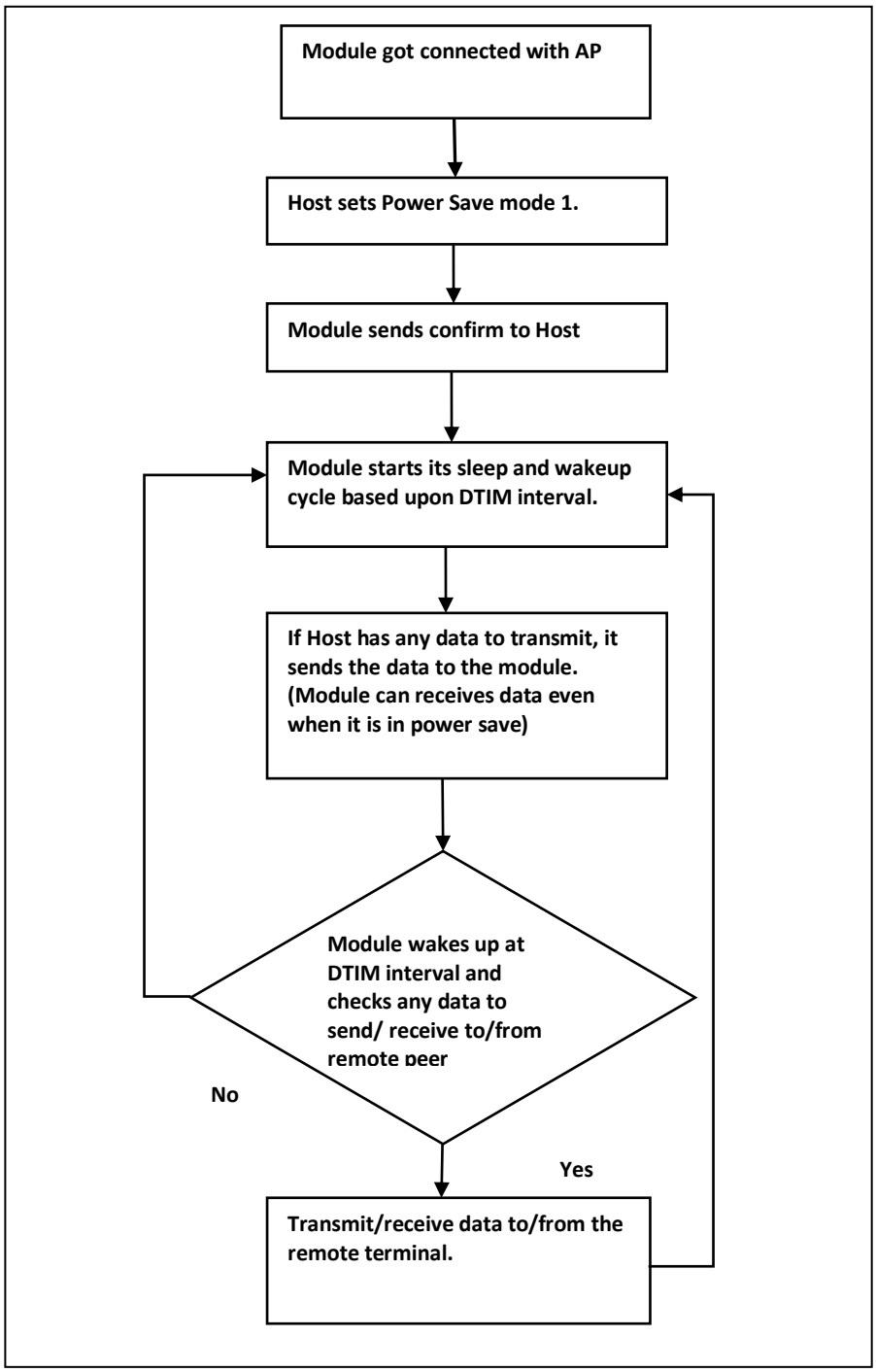


Figure 39: Power save Mode 1

After having configured the module to power save mode, the Host can issue subsequent commands. In power save mode 1, the module can receive data from the host at any point of time but it can send / receive the data to / from remote terminal only when it is awake at DTIM intervals.

8.17.1.2 Power save mode 2

Once the module is configured to power save mode 2, it can be woken up either by the Host or periodically during its sleep-wakeup cycle based upon wakeup period (In case of WLAN, it is DTIM interval andin case of BT, it will vary based on connected states.).

Power mode 2 is GPIO based. In ULP mode, feature_bit_map[4]has to be set in opermode command. In this mode, whenever host want to send data to the module, it gives wakeup indication to the module by setting GPIO pin #15 high in case of LP or ULP GPIO #0 in case of ULP (which make module to wakeup from power save). After wakeup, if the module is ready for data transfer, it sends wakeup indication to host (by setting GPIO pin #21 high or ULP GPIO #1 if feature_bit_map[5]is set to 1 in opermode command). Host is required to wait until module give wakeup indication before sending any data to the module.

After completion of data transfer, the host can give sleep permission to the module by resetting GPIO pin #15 in case of LP or ULP GPIO #0 in case of ULP. After recognizing sleep permission from the host, the module give confirmation to the host by resetting GPIO pin #21 or ULP GPIO #1 (if feature_bit_map[5]is set to 1 in opermode command) and again gets back to its sleep-wakeup cycle.

Module can send received packets or responses to the host at any instant of time. No handshake is required on Rx path.

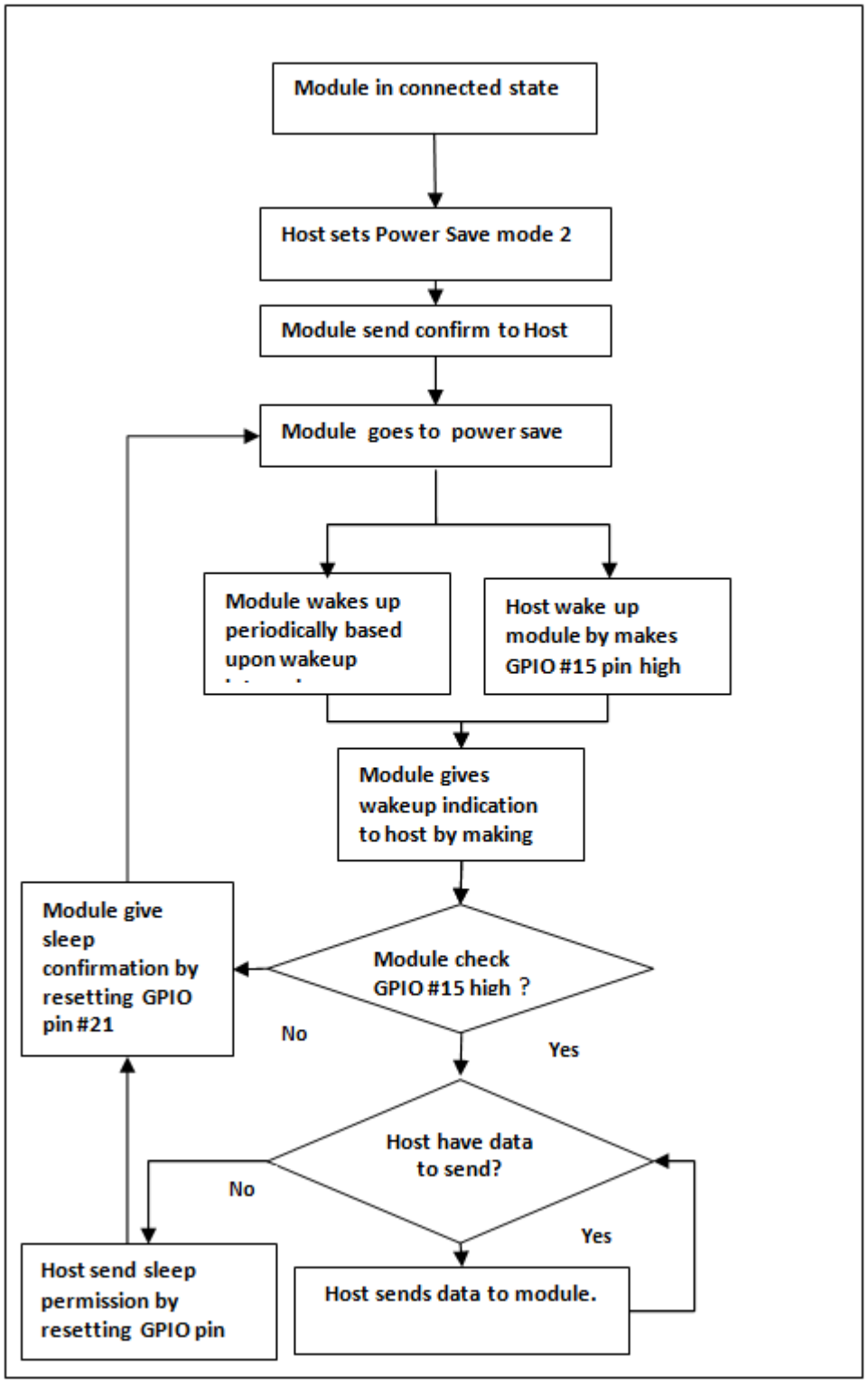


Figure 40: Power save mode 2

8.17.1.3 Power Save mode 3

Power Mode 3 is message based power save. Like Power Mode 2, in Power mode 3 also, both radio and SOC of RS9113-WiSeConnect module are in power save mode. This mode is significant when module is in connected state. Module wakes up periodically upon every wakeup period and gives wakeup message (“WKP”) to the host. The module can not be woken up asynchronously. Every time module intends to go to sleep, it sends a sleep request message (“SLP”) to the host and expects the host to send the ack message. The host either send ack (“ACK”) or any other pending message. But once ack is sent, the host should not send any other message unless and until next wakeup message from module is received.

Module shall not go into complete power-save state if ack is not received from host for given sleep message. The module can send received packets or responses to the host at any instant of time. No handshake is required on Rx path.

AT Mode	Binary Mode
“WKP”	0xDD
“SLP”	0xDE

Table 22: Message From Module in Power save Mode

AT Mode	Binary Mode
“ACK”	0xDE

Table 23: Message from host in Power save Mode

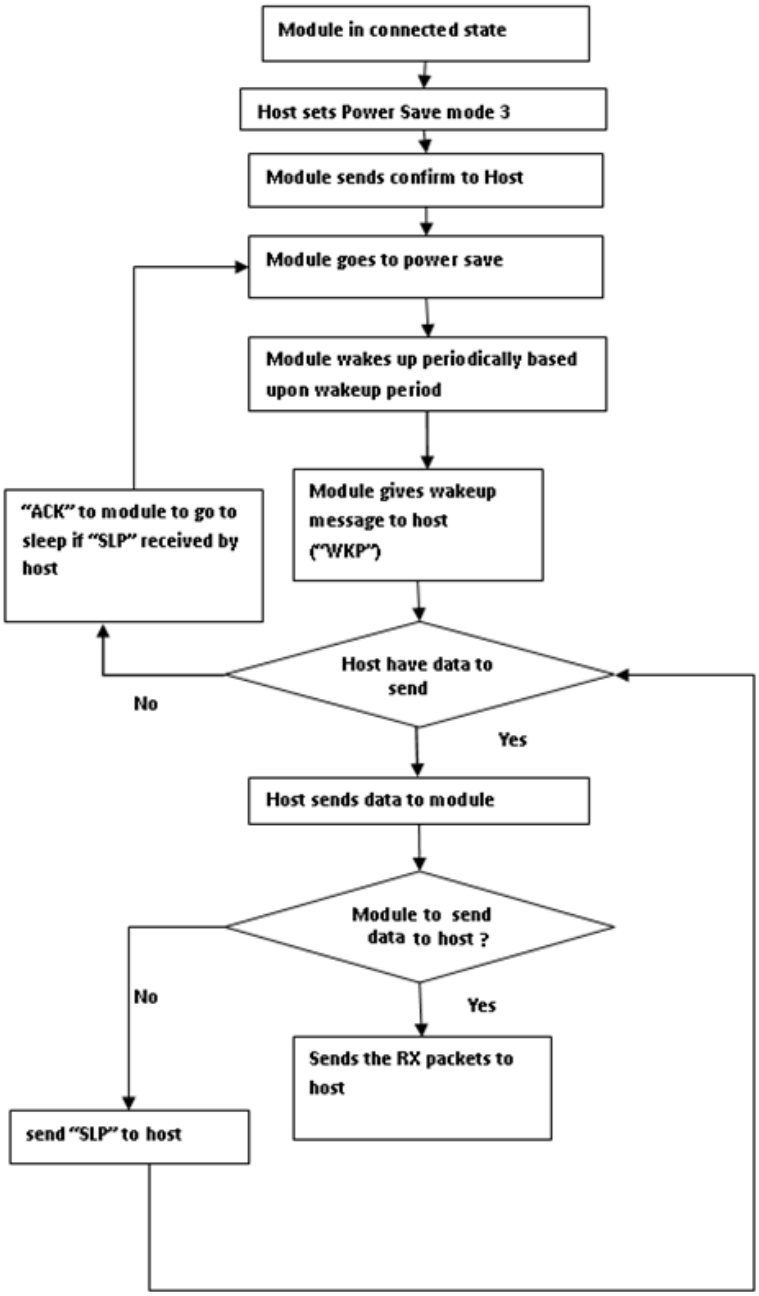


Figure 41: Power save Mode 3

8.17.1.4 Power save mode 8

This command has to be issued after Init command in case of WLAN and in standalone state (idle state) in case of BT-Classic/BLE.

In Power Mode 8 both RF and SOC of RS9113-WiSeConnect module are in complete power save mode. This mode is significant when module is in non connected state. Power mode 8 is GPIO based. In ULP mode , feature_bit_map[4]has to be set in opermode command.

In case of LP(when ulp_mode_enable is ‘0’) host can wakeup the module from power save by making GPIO #15 high.

In case of ULP(when ulp_mode_enable is ‘1’ or ‘2’) host can wakeup the module from power save by making ULP-GPIO #0 high.

When ulp_mode_enable is set to ‘0' or ‘1’, once the module gets wakeup, it continues to be in wakeup state until it gets power mode 8 commands from the host.

When ulp_mode_enable is set to ‘2’, after waking up from the sleep, the module sends the following message (refer to the table **Table 24: Message From Module in ULP Mode 2**) to the host when RAM retention is not enabled. After receiving this message, the host needs to start giving commands from beginning (oper mode) as module’s state is not retained.

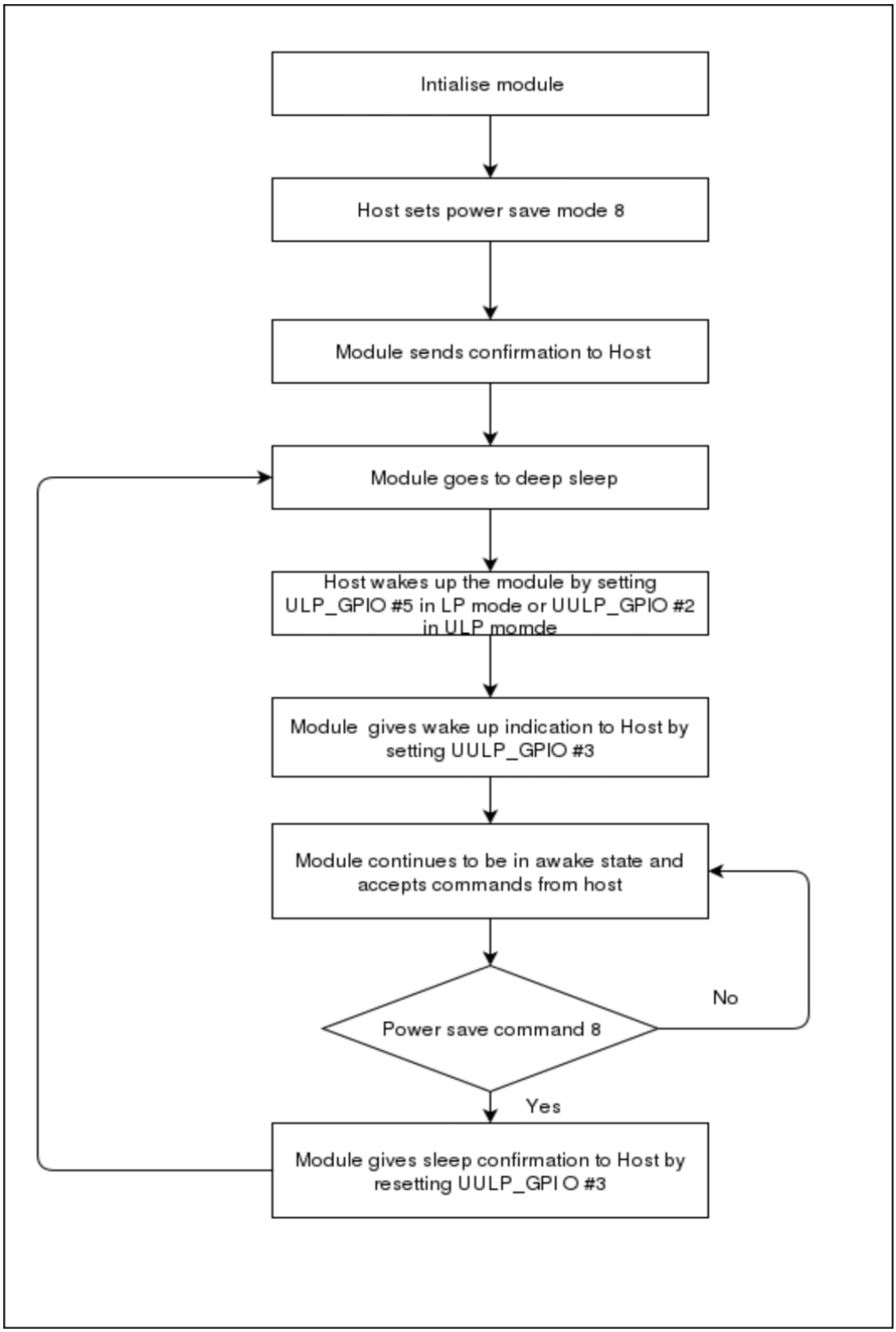


Figure 42: Power save Mode 8

8.17.1.5 Power save mode 9

In Power Mode 9 both Radio and SOC of RS9113-WiSeConnect module are in complete power save mode. This mode is significant when module is in non connected state. Once power mode 9 command is given, the module goes to sleep immediately and wakes up after sleep duration configured by the host by using set sleep timer command. If the host does not sets any default time, the module will wake up in 3sec by default. Upon wakeup, the module sends a wakeup message to the host and expects the host to give ack before it goes into next sleep cycle. The host either send ack or any other messages but once ACK is sent, no other packet should be sent before receiving next wakeup message.

When ulp_mode_enable is set to '2', after waking up from sleep, the module sends the following message to the host when RAM retention is not enabled. After receiving this message, the host needs to start giving commands from the beginning (opemode) as module's state is not retained.

AT Mode	Binary Mode
"WKP FRM SLEEP"	0xCD

Table 24: Message From Module in ULP Mode 2

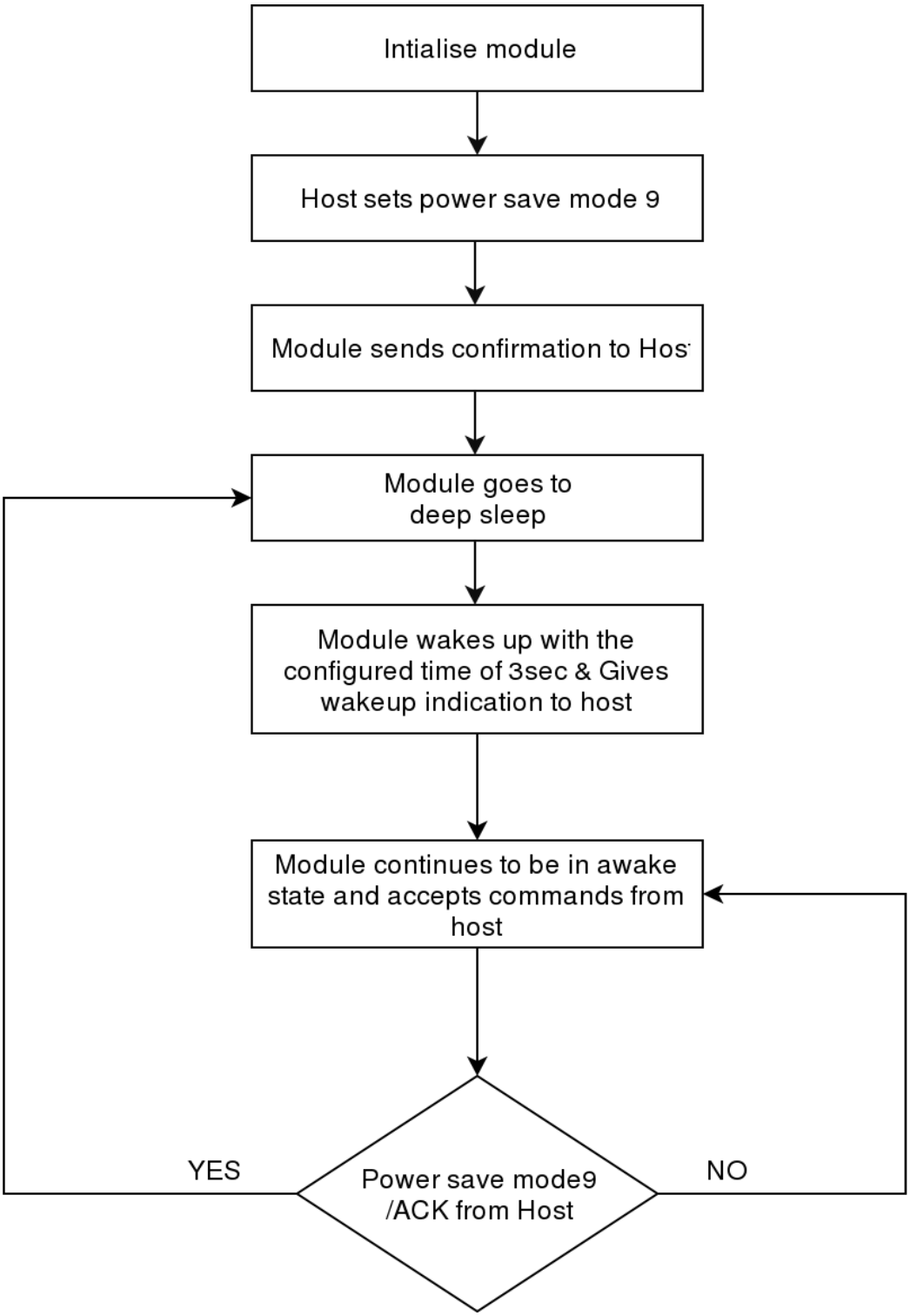


Figure 43: Power save Mode 9

8.18 PSK/PMK

Description:

The command is used to set the PSK (Pre shared key) to join to WPA/WPA2-PSK enabled APs. Using this command user can also pass the PMK (PAIRWISE MASTER KEY) as a parameter and can also generate PMK by providing PSK and SSID of connecting AP.

User can directly give PMK from host to reduce the connection time.This command should be issued after init and before join command if module needs to connect to an secure Access point. This command can be ignored if the AP is in Open mode.

Command Format:

AT Mode:

```
at+rsi_psk=<TYPE>,<psk_or_pmk >,<ap_ssid >\r\n
```

Binary Mode:

```
#define RSI_SSID_LEN 34
#define RSI_PSK_LEN  64
struct {
      uint8 TYPE;
      uint8 psk_or_pmk [RSI_PSK_LEN];
      uint8 ap_ssid [RSI_SSID_LEN];
}PskFrameSnd;
```

Command Parameters:

TYPE : possible values of this field are 1, 2, 3.

- 1 - indicate pre_shared_key is provided in psk_or_pmk field,
- 2 - indicate pairwise_master_key is provided in psk_or_pmk field,
- 3 - indicate generate pairwise master key from given pre shared key and SSID to which module wants to connect.

Note:

AT command mode TYPE 4 and 5 is added to support ‘,’ in PSK.

TYPE 4 – indicate length based PSK

5 – indicate generation of PMK from length based PSK and SSID

To support above type new parameter introduced in the command and command format for the same is

at+rsi_psk=<TYPE>,<length_of_psk>,<psk_or_pmk >,<ap_ssid >\r\n

where length_of_psk (1 byte) : gives the length of psk which can includes ',' for examples at+rsi_psk=4,10,123,456789 Where "123,456789" is psk with 10 bytes length

`psk_or_pmk`: In this field expected parameters are pre shared key of the access point to which module wants to associate or pair wise master key. Length of this field is 64 Bytes. In case of PMK only 32 bytes are valid, In case of PSK length can vary (8 to 63).

Note:

PMK Is of 32 Bytes .

In ATplus command mode, 32 bytes of PMK is given in hex format (64 characters) in the command.

For Example: If PMK is of array, PMK[32] = {0x71, 0x72, 0x01, 0x0A, 0x16, 0x17, 0x07,0x90, 0x71, 0x72, 0x01, 0x0A, 0x16, 0x17, 0x07,0x90, 0x71, 0x72, 0x01, 0x0A, 0x16, 0x17, 0x07,0x90, 0x71, 0x72, 0x01, 0x0A, 0x16, 0x17, 0x07,0x90};, then the command should be given as
at+rsi_psk=2,7172010A161707907172010A161707907172010A161707907172010A16170790\r\n

`SSID`: This field contains the SSID of the access point, this field will be valid only if `TYPE` value is 3.

Note: If user generates PMK using `TYPE` 3 (i.e. by providing psk and ssid) then module will use generated PMK for connection establishment and there is no need to give pre shared key or pair wise master key again.

Response:

Response contains following payload only if TYPE value is 3. In case of TYPE 1 & 2 response does not contains any payload.

AT Mode:

Result Code	Description
OK	Successful execution of the command. If TYPE value is '1'or '2'.
OK< pmk>	Successful execution of the command. If TYPE value is '3'.
ERROR<Error code>	Failure

Binary Mode:

```
typedef struct {
      uint8 pmk[32];
} rsi_PmkResponse;
```

Response Parameters:

Pair wise master key of 32 bytes is given to host if TYPE is 3.

Relevance:

This command is relevant in operating mode 0.

Possible Error Codes:

Possible error codes for this command are 0x0021, 0x0025,0x0026,0x0028,0x002C,0x0039, 0x003a, 0x003b .

Note: If this command is given by the user then there is no need to give pre_shared_key in join command in Binary Mode.

Example:

AT Mode:

To join a WPA2-PSK security enabled network with key “12345ABCDE”, the command is

```
at+rsi_psk=1,12345ABCDE\r\n
```

```
0x61    0x74    0x2B  0x72  0x73  0x69  0x5F   0x70   0x73
0x6B   0x3D 0x31    0x2c  0x31   0x32   0x33   0x34   0x35
0x41   0x42   0x43   0x44   0x45   0x0D   0x0A
```

Response:

```
OK\r\n
```

```
0x4F  0x4B  0x0D   0x0A
```

To join a WPA2-PSK security enable network with pairwise_master_key “ABCDEFABCDEFABCDEF12345678901234ABCDEFABCDEFABCDEF12345678901234” ,the command is

```
at+rsi_psk=2,
ABCDEFABCDEFABCDEF12345678901234ABCDEFABCDEFABCDEF123456789012
34,\r\n
```

Response:

```
OK\r\n
```

```
0x4F  0x4B  0x0D   0x0A
```

To generate pairwise_master_key for the pre_shared_key “12345678” and SSID “wise_ap”, the command is

```
at+rsi_psk=3,12345678,wise_ap\r\n
```

Response:

```
OK<pairwise_master_key>\r\n
0x4F  0x4B <32bytes of pairwise_master_key> 0x0D   0x0A
```

8.19 Set WEP Keys

Description:

This command configures the WEP key in the module to connect to an AP with WEP security. This command should be issued before join.

Command Format:

AT Mode:

```
at+rsi_wepkey=< index >,< key1 >,< key2 >,< key3 >,< key4
>\r\n
```

Binary Mode:

```
struct
{
      uint8  index[2];
      uint8  key[4][32];
}rsi_wepkey;
```

Command Parameters:

`index`: used to select key index configured in AP

- 0-Key 1 will be used.
- 1-Key 2 will be used.
- 2-Key 3 will be used.
- 3-Key 4 will be used.

`Key/Key1/Key2/Key3/Key4`: Actual keys. The module supports WEP hex mode only.

The key to be supplied to the AP should be of 10 characters (for 64 bit WEP mode) or 26 characters (for 128 bit WEP mode), and only the following characters are allowed for the key: A,B,C,D,E,F,a,b,c,d,e,f,0,1,2,3,4,5,6,7,8,9

Response:

AT Mode:

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

Binary Mode:

No response payload for this command.

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C,0x002D

Relevance:

This command is relevant when the module is configured in Operating Mode 0.

Example:

AT Mode:

Give write up for the below key entry

```
at+rsi_wepkey=0,ABCDE12345,ABCDE12346,ABCDE12347,
ABCDE12348\r\n
```

If the user wants to enter only one valid key

```
at+rsi_wepkey=0,ABCDE12345,0,0,0\r\n
```

If the user wants to enter only one valid key

```
at+rsi_wepkey=2,0,0,ABCDE12345,0\r\n
```

Response:

```
OK\r\n
```

```
0x4F  0x4B  0x0D  0x0A
```

8.20 Set WEP Authentication Mode

Description:

This command configures the authentication mode for WEP in the module, if the AP is in WEP security mode. This command supported only in AT Mode.

Command Format:

```
at+rsi_authmode=auth_mode\r\n
```

Command Parameters:

`auth_mode`: set to '0' for open WEP authentication

Note:
WEP shared mode is not supported in RS9113_WC_GENR_x_x_x release.

Response:

Result Code	Description
OK	Successful execution of the command
ERROR<Error code>	Failure, Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFF8,0x002D

Relevance:

This command is relevant when the module is configured in Operating Mode 0

Example:

```
at+rsi_authmode=0\r\n
0x61    0x74    0x2B    0x72    0x73    0x69    0x5F    0x61    0x75
0x74    0x68    0x6D    0x6F    0x64    0x65    0x3D    0x30    0x0D
0x0A
```

Response:

```
OK\r\n
0x4F  0x4B  0x0D   0x0A
```

8.21 Set EAP Configuration

Description:

This command is used to configure the EAP parameters for connecting to an Enterprise Security enabled Access Point. The supported EAP types are EAP-TLS, EAP-TTLS, EAP-PEAP, EAP-FAST, EAP-LEAP.

Note:
EAP-GTC is not supported for EAP-FAST.

Command Format:

AT Mode:

```
at+rsi_eap =< eapMethod >,< innerMethod >,< userIdentity >,<
password >,< okc >,<private_key_password>,<tls_version>\r\n
```

Binary Mode:

```
typedef struct {
      uint8  eapMethod[32];
      uint8  innerMethod[32];
      uint8  userIdentity[64];
      uint8  password[128];
      uint8  okc[4];
      uint8  private_key_password[82];
      uint8 tls_version;
}setEapFrameSnd ;
```

Command Parameters:

`eapMethod`: Should be one of among TLS, TTLS, FAST, LEAP or PEAP. It should be ASCII character string.

`innerMethod`: This field is valid only in TTLS/PEAP. In case of TTLS/PEAP supported inner methods are MSCHAP/MSCHAPV2. In case of TLS/FAST should be fixed to MSCHAPV2.

Here MSCHAP/MSCHAPV2 are ASCII character strings.

`userIdentity`: User ID which is configured in the user configuration file of the radius sever.

`Password`: Password which is configured in the user configuration file of the Radius Server for that User Identity.

`Okc`: To enable or disable opportunistic key caching(OKC)

- 0 – Disable
- 1 - Enable

OKC – When this is enabled, module will use cached PMKID get MSK(Master Session Key) which is need for generating PMK which is needed for 4-way handshake.

`private_key_password`: This is password for encrypted private key given to the module. Module will use this password during decryption of encrypted private key. Password length should not be more than 80 bytes.

Tls_version: Used to indicate which version of TLS will be used for connectivity.

1- TLS1.0

2- TLS1.2

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:

At+rsi_eap= TLS, MSCHAPV2, user1, password \r\n

0x61 0x74 0x2B 0x72 0x73 0x69 0x5f 0x65 0x61 0x70 0x3d 0x54 0x4c 0x53 0x2c
0x4d 0x53 0x43 0x48 0x41 0x50 0x56 0x32 0x2c 0x75 0x73 0x65 0x72 0x31 0x2c
0x70 0x61 0x73 0x73 0x77 0x6f 0x72 0x64 0x5c 0x72 0x5c 0x6e

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 2.

8.22 Set Certificate

Description:

This command is used to load/erase SSL (certificate and private keys) and enterprise security (EAP-TLS or EAP-FAST) certificates. Certificates should be loaded before using SSL/EAP. This command should be sent before join command for enterprise security mode and before socket creation for SSL sockets. Certificates will be loaded in non-volatile memory of the module so certificate load is required to be done only once.

Note:

This command should be sent only after opermode command.

Command Format:

AT Mode:

```
at+rsi_cert =< CertType >,< total_len >,< KeyPwd >,<
Certificate >\r\n
```

Binary Mode:

```
#define MAX_CERT_SEND_SIZE 1400
struct cert_info_s
{
      uint8 total_len[2];
      uint8 CertType;
      uint8 more_chunks;
      uint8 CertLen[2];
      uint8 KeyPwd[128];
};

#define MAX_DATA_SIZE (MAX_CERT_SEND_SIZE – sizeof(struct
cert_info_s))
struct SET_CHUNK_S
{
      struct cert_info_s cert_info;
      uint8 Certificate[MAX_DATA_SIZE];
};
```

Command Parameters:

`total_len`: Certificate's total length in bytes.

Note:

1) For Enterprise security, maximum cert_len should be less than 12280 bytes. For SSL Certificates, the max length is 12280 bytes and for the Private Keys, it is 4088 bytes.
2) Module shares same SSL certificates for all supported SSL sockets.
3) When using the EAP-TLS/PEAP security, there are two cases wherethe user may load the certificate into RS9113 module.

3.1) User can load a single certificate to location 1, for this user need to generate a single certificate which contains a combination of 4 certificates in a given fixed order(3 actual and 1 dummy) - one Private key, one Public key, one dummy, and one CA certificate. The CA certificate can include a chain of certificates. Make sure each certificate has their respective header and footers of -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----.

The dummy certificate can be a copy of one of the certificates for convenience (but must have the header and footer). It is not used for the authentication but is expected by the firmware. The sample wifi-user.pem certificate is an example of this, which is provided in

the release path:
"RS9113.NBZ.WC.GEN.OSI.x.y.z\utils\Radius_Server\raddb\certs\wifiuser.pem".

3.2) User can load individual EAP certificates private key, public key, and CA certificates with CertType as 17,33 and 49 respectively. Maximum certificate length for each individual certificates is 4088 bytes.If the user loads the certificates individually to their respective locations, then he will require only three certificates i.e. - one Private key, one Public key, and one CA certificate.

The CA certificate can include a chain of certificates. The user can load these separately. Again, make sure each has their respective header and footers of -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----.

4) `more_chunks, CertLen` fieldsare available only in Binary Mode.

Note:

1) Recommended to use loading of single certificate method (wifiuser.pem)

2) The certificates loaded shall be PEM format.

Note:

Set BIT(27) in tcp_ip_feature_bit_map to load SSl certificate onto RAM. Bydefault SSL certificates will be loaded onto flash.

`cert_type`: Type of certificate.

- 1– EAP client certificate
- 2– FAST PAC file[4]
- 3-SSL Client Certificate
- 4-SSL Client Private Key
- 5-SSL CA Certificate
- 6-SSL Server Certificate
- 7-SSL Server Private Key
- 17-EAP private key
- 33-EAP public key
- 49-EAP CA certificate

`more_chunks`: A maximum of "`MAX_DATA_SIZE`" bytes of the certificate can be sent to the module from the Host. If the certificate length is more than "`MAX_DATA_SIZE`" bytes, then the certificate need to be sent over multiple segments in case of Binary mode. If more_chunks is 0x01, then it indicates to the module that another segment is coming after the current segment. If it is 0x00, it indicates to the module that it is the last segment. Set this parameter to all '0' if total_length is '0'.

In case of AT mode Host need to send whole certificate at a time.

[4] Please check example PAC file given in release package for manual provisioning. Only that format is supported currently.

`CertLen`: Length of the current segment. This parameter is available only in Binary mode.

`keyPwd(128 bytes)`: Reserved.

`certificate`: This is the data of the actual certificate.

Certificate erase:

For erasing certificate,

more_chunks,cert_length,keypwd,certificate fields should be set to '0' in Binary Mode.
total_len, KeyPwd , Certificate fields should be set to '0' in AT Mode.

cert_type should be set to type of the certificate to erase as mentioned above

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.

Possible error codes:

Possible error codes are 0x0015,0x0021, 0x0025,0x0026, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 0,2

Example:

AT Mode:

It may not be possible to issue this command in Hyper-terminal because the content of a certificate file needs to be supplied as one of the inputs of the command. This can be done by other means, such as using a Python script. A sample Python excerpt is shown below, where wifiuser.pem is the names of the certificate file:

```
def set_cert():
      print "Set certificate\n"
      f3 = open('e:\\certificates\wifiuser.pem', 'r+')
```

```
str = f3.read()
num =len (str)
print 'Certificate len', num
out='at+rsi_cert=1,6522,password,'+str+'\r\n'
print 'Given command'
sp.write(out)
```

For loading certificate:

```
at+rsi_cert=1,10,12345678,@$cd5%ghij\r\n
```

```
0x61    0x74  0x2B  0x72  0x73  0x69  0x5F  0x63  0x65  0x72
0x74    0x3D  0x31  0x2C  0x31  0x30  0x2C  0x31  0x32  0x33
0x34    0x35  0x36  0x37  0x38  0x2C  0x40  0x24  0x63  0x64
0x35    0x25  0x67  0x68  0x69  0x6A  0x0D  0x0A
```

Response:

```
OK\r\n
```

```
0x4F  0x4B  0x0D   0x0A
```

For erasing certificate:

```
at+rsi_cert=1,0,0,0\r\n
```

```
0x61   0x74   0x2B   0x72   0x73   0x69   0x5F   0x63   0x65
0x72   0x74   0x3D   0x31   0x2C   0x30   0x2C   0x30   0x2C
0x30   0x0D   0x0A
```

Response:

```
OK\r\n
```

```
0x4F  0x4B  0x0D   0x0A
```

Binary Mode:

For example, to send a certificate of total length of 3000 bytes, the following flow should be used:

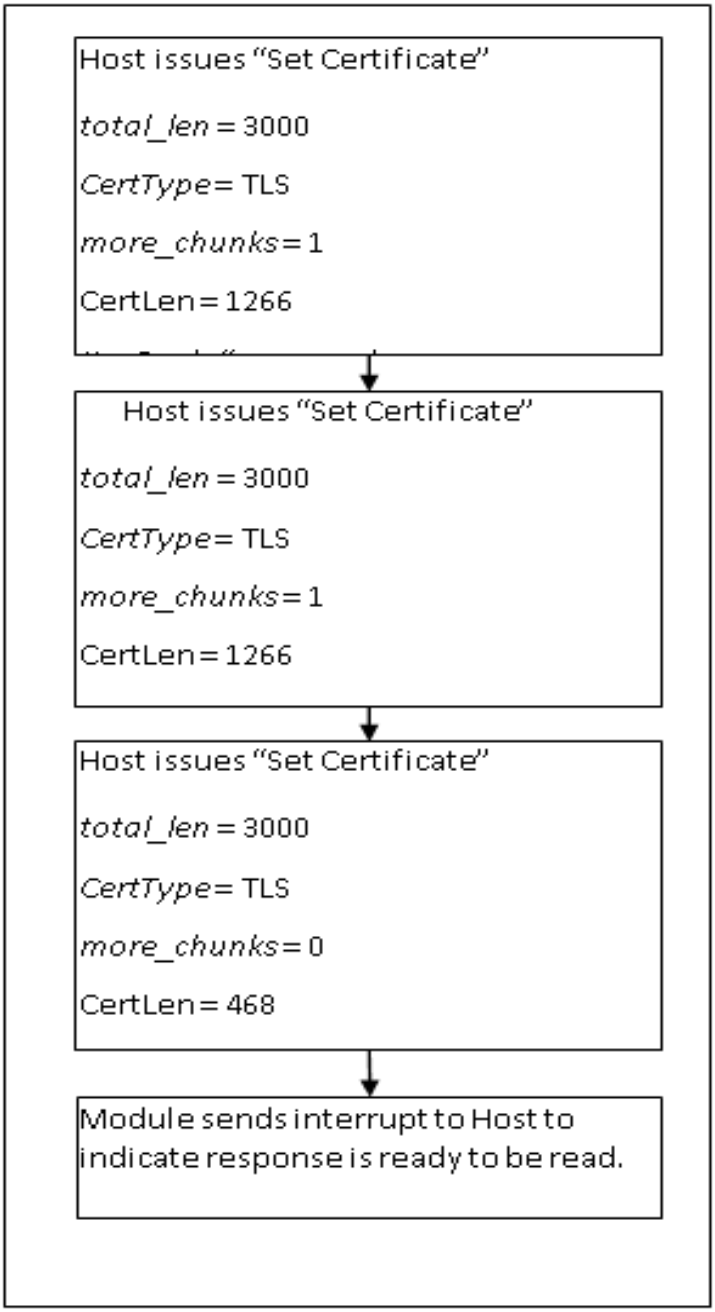


Figure 44: Loading Certificate in Binary Mode

8.23 Disassociate

Description:

This command is issued to request the module to disassociate (disconnect) from an Access Point. The Host can then issue a fresh set of Init, Scan and Join commands to connect to a different Access Point or the same Access Point with a different set of connection parameters. This command can also be used to stop the module from continuing an on-

going rejoin operation. Additionally, this command is used when the module is in AP mode, to remove clients from its list of connected nodes.

Command Format:

AT Mode:

```
at+rsi_disassoc=< mode_flag >,< client_mac_addr >\r\n
```

Binary Mode:

```
struct{
      uint8  mode_flag[2];
      uint8  client_mac_addr[6];
}rsi_disassoc_t;
```

Command Parameters:

`mode_flag`:

0-Module is in client mode. The second parameter mac_addr is ignored when mode is 0.

1-Module is in AP mode.

`mac_addr`: MAC address of the client to disconnect. Used when the module is in AP mode

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.

Possible error codes:

Possible error codes are 0x0006, 0x0013, 0x0021, 0x002C, 0x0015.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Note:

If user issues disconnect command in P2P mode, then there is no way for user to continue further. Module needs a soft reset in that case.

After issuing disconnect command, if there is any power save enabled by that time will be disabled. User can reissue the power save command after initializing the module again.

Example:

AT Mode:

Module is in client mode and is connected to an AP. It wants to formally disconnect from the AP.

```
at+rsi_disassoc=0\r\n
```

```
0x61    0x74    0x2B    0x72    0x73    0x69    0x5F    0x64    0x69
0x73    0x61    0x73    0x73    0x6F    0x63    0x3D    0x30    0x0D
0x0A
```

Module is in AP mode and 3 clients are connected to it. One of the clients, with MAC 0x01 0x02 0x03 0x040 0x05 0x06 , needs to be disconnected by the AP.

```
at+rsi_disassoc=1,010203040506\r\n
```

```
0x61    0x74      0x2B  0x72  0x73  0x69  0x5F  0x64  0x69
0x73  0x61 0x73    0x73  0x6F    0x63   0x3D   0x31   0x2C
0x30  0x31   0x30  0x32  0x30  0x33  0x30  0x34  0x30  0x35
0x30  0x36  0x0D   0x0A
```

Response:

```
OK\r\n
```

```
0x4F  0x4B  0x0D   0x0A
```

8.24 Set IP Parameters

Description:

This command configures the IP address, subnet mask and default gateway for the module.

AT Mode:

```
at+rsi_ipconf=< dhcpMode >, < ipaddr >,< netmask >,< gateway
>, < hostname >, <vap id>,< fqdnFlag>\r\n
```

Binary Mode:

```
struct {
      uint8  dhcpMode;
      uint8  ipaddr[4];
```

```
        uint8  netmask[4];
        uint8  gateway[4];
        uint8  hostname[31];
        uint8 vap_id;
        uint8 fqdnFlag[4];
} ipparamFrameSnd;
```

Command Parameters:

dhcpMode: Used to configure TCP/IP stack in manual or DHCP modes.

- 0– Manual
- 1– DHCP enabled
- 3 - To enable DHCP and to send host name in DHCP discover

Note:

1) In AP mode only DHCP manual mode (static IP) is valid.

2) Sending host name is valid only in DHCP enable mode.

- 4 -To enable FQDN Option with static ip (Option 81 with static IP)
- 7 -To enable DHCP, hostname, DHCP client FQDN option (Option 81 with Dynamic IP)
- 9 - To support DHCP unicast Offer from server

ipAddr: IP address in 4 bytes hex format. This can be 0's in the case of DHCP.

Netmask: Subnet mask in 4 bytes hex format. This can be 0's in the case of DHCP.

Gateway: Gateway in 4 bytes hex format. This can be 0's in the case of DHCP.

hostname: Host name for DHCP Client. This can be null, when DHCP mode is not enabled. This field is valid only when dhcpMode value is 3. Maximum Hostname length is valid up to 31 bytes including NULL.

Vap_id: Possible values are 0 and 1.

When 1 - Used tostart DHCP server in concurrent AP mode (should be given before AP creation i.e. join).

When 0 – Used to assign static IP for client mode.

Note:

vap_id will be considered only in concurrent mode and when dhcp mode is manual.

fqdnFlag : 0 - DNS Client should update TYPE_A(host name) record and TYPE_PTR records.

1 - DHCP Server will update both TYPE_A and TYPE_PTR records.

Response:

AT Mode:

Result Code	Description
OK< macAddr>< ipaddr>< netmask>< gateway>	Successful execution of the command
ERROR<Error code>	Failure

Binary Mode:

```
typedef struct {
      uint8  macAddr[6];
      uint8  ipaddr[4];
      uint8  netmask[4];
      uint8  gateway[4];
} rsi_ipparamFrameRcv;
```

Response Parameters:

`macAddr(6 bytes)`: MAC Address

`ipAddr(4 bytes)` : Assigned IP address

`netmask(4 bytes)`: Assigned subnet address

`gateway(4 bytes)`: Assigned gateway address

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFFC,0xFF74, 0xFF9C, 0xFF9D, 0xFF75.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2, 6 and 9.

Example:

AT Mode:

To configure in manual mode, with 192.168.1.3, 255.255.255.0 and 192.168.1.1 as the IP address, subnet mask and gateway the command is

```
at+rsi_ipconf=0,192.168.1.3,255.255.255.0,192.168.1.1\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x69  0x70  0x63
0x6F  0x6E  0x66  0x3D  0x30  0x2C  0x31  0x39  0x32  0x2E
```

```
0x31  0x36  0x38  0x2E  0x31  0x2E  0x33  0x2C  0x32 0x35
0x35  0x2E  0x32  0x35  0x35  0x2E  0x32  0x35  0x35  0x2E
0x30  0x2C  0x31  0x39  0x32  0x2E  0x31  0x36  0x38  0x2E
0x31  0x2E  0x31  0x0D  0x0A
```

Response:

```
OK<MAC_Address><IP_Address><Subnet_Mask><Gateway>\r\n
```

```
0x4F  0x4B  0x01 0x02 0x03  0x04 0x05 0x06 0xC0 0xA8 0x01 0x03
0xFF 0xFF 0xFF 0x00  0xC0 0xA8 0x01 0x01  0x0D  0x0A
```

To configure the IP in DHCP enabled mode, the command is

```
at+rsi_ipconf=1,0,0,0\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x69  0x70  0x63
0x6F  0x6E  0x66  0x3D  0x31  0x2C  0x30  0x2C  0x30  0x2C
0x30  0x0D  0x0A
```

Response:

```
OK<MAC_Address><IP_Address><Subnet_Mask><Gateway>\r\n
```

```
0x4F  0x4B  0x01  0x02  0x03  0x04  0x05  0x06  0xC0  0xA8
0x01  0x030xFF  0xFF  0xFF  0x00  0xC0  0xA8  0x01  0x01
0x0D  0x0A
```

To configure the IP in DHCP enabled mode with hostname, the command is

```
at+rsi_ipconf=3,dhcp_client\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x69  0x70  0x63
0x6F  0x6E  0x66  0x3D  0x31  0x2C  0x30  0x2C  0x30  0x2C
0x30  0x0D  0x0A
```

Response:

```
OK<MAC_Address><IP_Address><Subnet_Mask><Gateway>\r\n
```

```
0x4F  0x4B  0x01  0x02  0x03  0x04  0x05  0x06  0xC0  0xA8
0x01  0x030xFF  0xFF  0xFF  0x00  0xC0  0xA8  0x01  0x01
0x0D  0x0A
```

8.25 IP Change Notification

Description:

This notification is received when module gets a different IP as compared to modules old IP address, after DHCP renewal. Module indicates this IP change to host by an asynchronous frame.

Command Format:

N/A

Command Parameters:

N/A

Response:

AT Mode:

Result Code	Description
AT+RSI_IPCONF< macAddr>< ipaddr>< netmask>< gateway>	

Binary Mode:

```
typedef struct {
      uint8  macAddr[6];
      uint8  ipaddr[4];
      uint8  netmask[4];
      uint8  gateway[4];
} rsi_recvIpChange;
```

Response Parameters:

macAddr(6 bytes): MAC Address

pAddr(4 bytes): Assigned IP address

netmask(4 bytes): Assigned subnet address

gateway(4 bytes): Assigned gateway address

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Example:

AT Mode:

```
AT+RSI_IPCONF< macAddr >< ipaddr >< netmask >< gateway >\r\n
0x41  0x54  0x2B  0x52  0x53  0x49  0x5F  0x49  0x50  0x43
0x4F  0x4E  0x46  0x3D  0x01  0x02  0x03  0x04  0x05  0x06
0xC0  0xA8  0x01  0x030xFF  0xFF  0xFF  0x00  0xC0  0xA8
0x01  0x01  0x0D 0x0A
```

8.26 Set IPv6 Parameters

Description:

This command configures the IPv6 address, prefix length and default router for the module.

Command Format:

AT Mode:

```
at+rsi_ipconf6=< mode >,< prefixLength >,< ipaddr6 >,
< gateway6 >\r\n
```

Binary Mode:

```
struct {
      uint8  mode[2];
      uint8  prefixLength[2];
      uint32 ipaddr6[4];
      uint32 gateway6[4];
} ipconf6FrameSnd;
```

Command Parameters:

`mode`: Used to configure TCP/IP stack in manual or DHCPv6 modes.

0–Manual

1–DHCPv6

`prefixLength`: Prefix length of the IPv6 address.

`ipaddr6`: IPv6 address. This can be 0's in the case of DHCPv6.

`gateway6`: Default router's IPv6 address. This should be Null in the case of DHCPv6.

IPV6 address is generally of the form - octet of four hexadecimal digits separated by "colons".

e.g. 2001:0db8:1:0:0:0:0:123 (supported format)

2001:db8:1::123 (not supported format)

Double colons are used in place of continuous zeroes in IPV6 address, to minimize the IPV6 address, but in RS9113-WiSeConnect module double colons are not supported. In ipconf6 command you have to supply all the eight groups of four hexadecimal digits.

Response:

AT Mode:

Result Code	Description
OK< prefixLength>< ipaddr6>< defaultgw6>	Successful execution of the command.
ERROR<Error code>	Failure

Binary Mode:

```
typedef struct {
      uint8 prefixLength[2];
      uint8 ipaddr6[16];
      uint8 defaultgw6[16];
} rsi_ipconf6FrameRcv;
```

Response Parameters:

`prefixLength(2 bytes)`: Prefix length of IPv6 address

`ipaddr6(16 bytes)`: Assigned IPv6 address

`gateway6(16 bytes)`: Assigned default_router address

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFFC, 0xFF9C,0xFF74, 0xFF9D.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Example:

AT Mode:

To configure in manual mode, with 2001:db8:0:1:0:0:0:123 as the IPV6 address and with 2001:db8:1:0:0:0:0:100 as router IPV6 address, the command is :

```
at+rsi_ipconf6=0,64,2001:DB8:0:1:0:0:0:123,2001:DB8:0:1:0:0:0:100\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x69  0x70  0x63
0x6F  0x6E  0x66  0x36  0x3D  0x30  0x2C  0x36  0x34  0x2C
0x32  0x30  0x30  0x31  0x3A  0x44  0x42  0x38  0x3A  0x00
      0x3A  0x31  0x3A  0x30  0x3A  0x30  0x3A  0x30  0x3A
0x31  0x32  0x33  0x2C  0x32  0x30  0x30  0x31  0x3A  0x44
```

```
      0x42  0x38 0x3A   0x30  0x3A  0x31  0x3A  0x30  0x3A
0x30  0x3A  0x30   0x3A  0x31  0x00  0x00 0x0D  0x0A
```

Response:

```
OK<prefixLength><ipaddr6><gateway6>\r\n
```

```
0x4F  0x4B  0x40 0x000xB8 0x0D 0x01 0x20 0x01 0x00 0x00 0x00
0x00  0x00  0x00 0x00 0x23 0x01 0x00 0x000xB8 0x0D 0x01 0x20
0x01  0x00  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00
0x0D  0x0A
```

To configure the IPV6 in DHCPV6 enabled mode, the command is

```
at+rsi_ipconf6=1\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x69  0x70  0x63
0x6F  0x6E  0x66  0x36  0x3D  0x31  0x0D  0x0A
```

Response:

```
OK<prefixLength><ipaddr6><gateway6>\r\n
```

```
0x4F  0x4B 0x40 0x000xB8 0x0D 0x01 0x20 0x01 0x00 0x00 0x00
0x00  0x00 0x00 0x00  0x36 0x01 0x00 0x000xB8 0x0D 0x01 0x20
0x01  0x00 0x00 0x00  0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00
0x0D  0x0A
```

The IPv6 address assigned to module by DHCPv6 server is 2001:DB8:1:0:0:0:0:136, default prefix 64 is used and router ipv6 address is 2001:DB8:1:0:0:0:0:100.

8.27 SNMP

Description:

This command configures the SNMP agent in the module. This command can be issued only after set IP parameters or set IPv6 parameters command. This is the very first command for using SNMP feature

Command Format:

AT Mode:

```
at+rsi_snmp_enable=< snmpEnable >\r\n
```

Binary Mode:

```
struct {
      uint8 snmpEnable;
} snmpEnableFrameSnd;
```

Command Parameters:

`snmpEnable:` To enable SNMP agent in module

0 - SNMP disable

1 - SNMP enable

Response:

AT Mode:

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

Binary Mode:

N/A

Response Parameters:

N/A

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C,0xFF74,0x0015,0x100,0xFF82.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Example:

AT Mode:

To enable SNMP in module the command is

```
at+rsi_snmp_enable=1\r\n
```

……

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x73   0x6E   0x6D
0x70   0x5F   0x65   0x6E   0x61   0x62   0x6C   0x65   0x3D
0x31 0x0D   0x0A
```

```
Response:
OK\r\n
```

```
0x4F  0x4B  0x0D 0x0A
```

8.28 SNMP Set

Description:

This is an asynchronous message which module gives to host whenever it receives snmp set message from the remote SNMP server to set the value corresponding to the object id. This message can only be received when module is configured as an SNMP agent.

Command Format:

AT Mode:

N/A

Binary Mode:

N/A

Command Parameters:

N/A

Response:

AT Mode:

Result Code	Description
AT_RSI_SNMP_SET=< object_id>,< length>,< value>	Asynchronous message sent by remote snmp server and received by the module.

Binary Mode:

```
struct{
      uint8 object_id[128];
      uint8 length[4];
      uint8 value[200];
}rsi_snmp_set;
```

Response Parameters:

`object_id (128 bytes)` : object id for which the snmp server wants to set.

`length (4bytes)` : length of value of object id.

LSB is returned first.

`value (200bytes)`: value of object id to be set. Out of these 200 bytes user has to consider only length number of bytes.

Note:

`value` contains maximum of 144 bytes in case of IPV6.

Possible error codes:

N/A

Relevance:

This message is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Example:

AT Mode:

```
AT+RSI_SNMP_SET=1.3.6.1.2.1.1.1.0,4,home\r\n
```

```
0x41   0x54   0x2B   0x52   0x53   0x49   0x5F   0x53   0x4E
0x4D   0x50   0x5F   0x53   0x45   0x54   0x3D   0x31   0x2E
0x33   0x2E   0x36   0x2E   0x31   0x2E   0x32   0x2E   0x31
0x2E   0X31   0x2E   0x31   0x2E   0x30   0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X000x2C   0x00   0x00   0x00   0x04   0x2C
0X68   0X6F   0X6D   0X65   0X00   0X00 0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00   0X00
0X00   0X00   0X0D   0X0A
```

8.29 SNMP Get Response

Description:

This command is given in response to the get request received by the SNMP agent (module) and issued by the SNMP server. Whenever snmp server sends a snmp get request ,module indicates this to host by an asynchronous message. Then module has to issue this command for sending response to corresponding received snmp get request.

Command Format:

AT Mode:

```
at+rsi_snmp_get_rsp=< type >,< value >< OID >\r\n
```

Binary Mode:

```
#define MAX_SNMP_VALUE 200
#define MAX_OID_LENGTH 128
struct{
      uint8 type;
      uint8 value[MAX_SNMP_VALUE];
      uint8 objid[MAX_OID_LENGTH];
}snmpFrameSnd;
```

Command Parameters:

`type`: type of object requested.

SNMP Object Type	Object code
SNMP_ANS1_COUNTER	0x41
SNMP_ANS1_COUNTER64	0x46
SNMP_ANS1_END_OF_MIB_VIEW	0x82
SNMP_ANS1_GAUGE	0x42
SNMP_ANS1_OBJECT_ID	0x6
SNMP_ANS1_INTEGER	0x2
SNMP_ANS1_IP_ADDRESS	0x40
SNMP_ANS1_IPV6_ADDRESS	0x44
SNMP_ANS1_NO_SUCH_INSTANCE	0x81
SNMP_ANS1_NO_SUCH_OBJECT	0x80
SNMP_ANS1_OCTET_STRING	0x4
SNMP_ANS1_TIME_TICS	0x43

Table 25: SNMP Object Types and Codes

`value`: value passed in response corresponding to the type of object requested.

Note:

1. Size of the value should be 200 bytes.

For example, if the value is "redpine" (length of the value is 7 bytes) then remaining 193 bytes should be filled with NULL.

2. OID should be given followed by value, without having any comma(,) separator between value and OID.

Response:

AT Mode:

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

Binary Mode:

N/A

Response Parameters:

N/A

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

The list of example OIDs are:

OID	Description
1.3.6.1.2.1.4.3.0	The total number of input datagrams received from interfaces, including those received in error
1.3.6.1.2.1.4.4.0	The number of input datagrams discarded due to errors in their IP headers, including bad checksums, version number mismatch, other format errors, time-to-live exceeded, errors discovered in processing their IP options, etc.
1.3.6.1.2.1.4.5.0	The number of input datagrams discarded because the IP

OID	Description
	address in their IP header's destination field was not a valid address to be received at this entity. This count includes invalid addresses (e.g., 0.0.0.0) and addresses of unsupported Classes (e.g., Class E). For entities which are not IP routers and therefore do not forward datagrams, this counter includes datagrams discarded because the destination address was not a local address
1.3.6.1.2.1.4.6.0	The number of input datagrams for which this entity was not their final IP destination, as a result of which an attempt was made to find a route to forward them to that final destination. In entities which do not act as IP routers, this counter will include only those packets which were Source-Routed via this entity, and the Source-Route option processing was successful
1.3.6.1.2.1.4.7.0	The number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol."ecause of an unknown or unsupported protocol
1.3.6.1.2.1.4.9.0	The total number of input datagrams successfully delivered to IP user-protocols (including ICMP)
1.3.6.1.2.1.4.10.0	The total number of IP datagrams which local IP user-protocols (including ICMP) supplied to IP in requests for transmission. Note that this counter does not include any datagrams counted in ipForwDatagrams
1.3.6.1. 2.1.4.11.0	The number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but which were discarded (e.g., for lack of buffer space). Note that this counter would include datagrams counted in ipForwDatagrams if any such packets met this (discretionary) discard criterion
1.3.6.1.2.1.4.12.0	The number of IP datagrams discarded because no route could be found to transmit them to their destination. Note that this counter includes any packets counted in ipForwDatagrams which meet this `no-route' criterion. Note that this includes any datagrams which a host cannot route because all of its default routers are down
1.3.6.1.2.1.4.14.0	Number of Internet Protocol (IP) fragments received which needed to be reassembled at this entity
1.3.6.1.2.1.4.15.0	Number of Internet Protocol (IP) datagrams successfully re-assembled
1.3.6.1.2.1.4.16.0	Number of failures detected by the IP reassembly algorithm (for whatever reason: timed out, errors, etc). Note that this is not necessarily a count of discarded IP fragments since some algorithms (notably the algorithm in RFC 815) can lose track of the number of fragments by combining them as they are

OID	Description
	received.
1.3.6.1. 2.1.4.17.0	Number of Internet Protocol (IP) datagrams that have been successfully fragmented at this entity
1.3.6.1.2.1.4.18.0	The number of Internet Protocol (IP) datagrams that have been discarded because they needed to be fragmented at this entity but could not be, e.g. because their Don't Fragment flag was set
1.3.6.1.2.1.4.19.0	Number of Internet Protocol (IP) datagram fragments that have been generated as a result of fragmentation at this entity
1.3.6.1.2.1.5.1.0	The total number of ICMP messages which the entity received. Note that this counter includes all those counted by icmpInErrors
1.3.6.1.2.1.5.2.0	The number of ICMP messages which the entity received but determined as having ICMP-specific errors (bad ICMP checksums, bad length, etc.)
1.3.6.1.2.1.5.8.0	The number of ICMP Echo (request) messages received
1.3.6.1.2.1.5.9.0	The number of ICMP Echo Reply messages received
1.3.6.1.2.1.5.14.0	The total number of ICMP messages which this entity attempted to send. Note that this counter includes all those counted by icmpOutErrors
1.3.6.1.2.1.5.15.0	Number of ICMP messages which this entity did not send due to problems discovered within ICMP such as a lack of buffers. This value should not include errors discovered outside the ICMP layer such as the inability of IP to route the resultant datagram. In some implementations there may be no types of error which contribute to this counter's value
1.3.6.1.2.1.5.21.0	The number of ICMP Echo (request) messages sent.
1.3.6.1.2.1.5.22.0	The number of ICMP Echo Reply messages sent
1.3.6.1.2.1.6.5.0	The number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state
1.3.6.1.2.1.6.6.0	The number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state
1.3.6.1.2.1.6.7.0	The number of times TCP connections have made a direct transition to the CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN

OID	Description
	state from the SYN-RCVD state
1.3.6.1.2.1.6.8.0	The number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state
1.3.6.1.2.1.6.10.0	The total number of segments received, including those received in error. This count includes segments received on currently established connections
1.3.6.1.2.1.6.11.0	The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets
1.3.6.1. 2.1.6.12.0	The total number of segments retransmitted - that is, the number of TCP segments transmitted containing one or more previously transmitted octets
1.3.6.1.2.1.6.14.0	The total number of segments received in error (e.g., bad TCP checksums)
1.3.6.1. 2.1.6.15.0	The number of TCP segments sent containing the RST flag
1.3.6.1.2.1.7.1.0	The total number of UDP datagrams delivered to UDP users
1.3.6.1.2.1.7.4.0	The total number of UDP datagrams sent from this entity
1.3.6.1.2.1.7.3.0	The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port

Example:

AT Mode:

To respond for string type of SNMP object requested by the remote SNMP server "AT+RSI_SNMP_GET=1.3.6.1.2.1.1.1.0" the command is :

For OID of type STRING, OID, NO_SUCH_OBJECT, NO_SUCH_INSTANCE and END_OF_MIB_VIEW, the response shall be given in the form of ASCII.

To respond for string type of SNMP object requested by the remote SNMP server "AT+RSI_SNMP_GET=1.3.6.1.2.1.1.3.0" the command is :

at+rsi_snmp_get_rsp=4,redpine<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NU
L>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL

```
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□1.3.6.1.2.1.1.3.0\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x73 0x6E 0x6D 0x70 0x5F 0x67 0x65 0x74 0x6E 0x65
0x78 0x74 0x5F 0x72 0x73 0x70 0x3D 0x34 0x2C 0x72 0x65 0x64 0x70 0x69 0x6E 0x65 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x31 0x2E 0x33 0x2E 0x36 0x2E 0x31 0x2E 0x32 0x2E 0x31 0x2E
0x31 0x2E 0x33 0x2E 0x30 0x0D 0x0A
```

Response:

```
OK\r\n
```

```
0x4F  0x4B  0x0D 0x0A
```

For, OID of type INTEGER, COUNTER, COUNTER64, GAUGE, TIMER_TICKS input shall be given in hexa-equivalent form.

To respond for INTEGER type of SNMP object requested by the remote SNMP server “AT+RSI_SNMP_GET=1.3.6.1.2.1.1.3.0” the command is :

```
fOr Integer value 6:-
```

```
HEX EQUIVALENT:
```

```
61 74 2B 72 73 69 5F 73 6E 6D 70 5F 67 65 74 5F 72 73 70 3D 32
2C 06 0D 0A
```

For Integer value 10:-
HEX EQUIVALENT:

61 74 2B 72 73 69 5F 73 6E 6D 70 5F 67 65 74 5F 72 73 70 3D 32 2C 0A 0D 0A

For Integer value 276:-
HEX EQUIVALENT:
61 74 2B 72 73 69 5F 73 6E 6D 70 5F 67 65 74 5F 72 73 70 3D 32 2C 14 01 0D 0A

For, OID of type IP_ADDRESS, if the response ip-address want to give is 192.168.10.163
the command shall be given in hexa form as shown below:
0x61 0x74 0x2b 0x72 0x73 0x69 0x5f 0x73 0x6e 0x6d 0x70 0x5f 0x67 0x65 0x74 0x5f 0x72 0x73 0x70 0x3d 0x36 0x34 0x2c 0xa3 0x0a 0xa8 0xc0 0x0d 0x0a

For, OID of type IPV6_ADDRESS, if the response ipv6-address want to give is 2001:db8:0:1:0:0:0:123
the command shall be given in hexa form as shown below:
0x61 0x74 0x2b 0x72 0x73 0x69 0x5f 0x73 0x6e 0x6d 0x70 0x5f 0x67 0x65 0x74 0x5f 0x72 0x73 0x70 0x3d 0x36 0x38 0x2c 0xb8 0x0d 0x01 0x20 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x23 0x01 0x00 0x00 0x0d 0x0a

8.30 SNMP Get Next Response

Description:

This command is given in response to the get request received by the SNMP agent (module) and issued by the SNMP server. Whenever snmp server sends a snmp get_next request ,module indicates this to host by an asynchronous message. Then module has to issue this command for sending response to corresponding received snmp get next request.

Command Format:

AT Mode:

```
at+rsi_snmp_getnext_rsp=< type >,< value >< next objid in MIB table >\r\n
```

Binary Mode:

```
#define MAX_SNMP_VALUE 200
#define MAX_OID_LENGTH 128

struct{
      uint8 type;
      uint8 value[MAX_SNMP_VALUE];
      uint8 objid[MAX_OID_LENGTH];
}snmpFrameSnd;
```

Command Parameters:

`type` : type of object requested.

SNMP Object Type	Object code
SNMP_ANS1_COUNTER	0x41
SNMP_ANS1_COUNTER64	0x46
SNMP_ANS1_END_OF_MIB_VIEW	0x82
SNMP_ANS1_GAUGE	0x42
SNMP_ANS1_OBJECT_ID	0x6
SNMP_ANS1_INTEGER	0x2
SNMP_ANS1_IP_ADDRESS	0x40
SNMP_ANS1_IPV6_ADDRESS	0x44
SNMP_ANS1_NO_SUCH_INSTANCE	0x81
SNMP_ANS1_NO_SUCH_OBJECT	0x80
SNMP_ANS1_OCTET_STRING	0x4
SNMP_ANS1_TIME_TICS	0x43

Table 26: SNMP Object Types and Codes

`value`: value passed in response corresponding to the type of object requested.

Note:

1. Size of the value should be 200 bytes.

For example, if the value is "redpine" (length of the value is 7 bytes) then remaining 193 bytes should be filled with NULL.

2. OID should be given followed by value, without having any comma(,) separator between value and OID.

Response:

AT Mode:

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

Binary Mode:

N/A

Response Parameters:

N/A

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Example:

AT Mode:

To respond for string type of SNMP object requested by the remote SNMP server "AT+RSI_SNMP_GETNEXT=1.3.6.1.2.1.1.1.0" the command is :

To respond for string type of SNMP object requested by the remote SNMP server "AT+RSI_SNMP_GETNEXT=1.3.6.1.2.1.1.3.0" the command is :

```
at+rsi_snmp_getnext_rsp=4,redpine<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□
<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NU
L>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL
>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<NUL>□<
NUL>□<NUL>□1.3.6.1.2.1.1.4.0\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x73 0x6E 0x6D 0x70 0x5F
0x67 0x65 0x74 0x6E 0x65 0x78 0x74 0x5F 0x72 0x73 0x70 0x3D
0x34 0x2C 0x72 0x65 0x64 0x70 0x69 0x6E 0x65 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

```
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x31 0x2E
0x33 0x2E 0x36 0x2E 0x31 0x2E 0x32 0x2E 0x31 0x2E 0x31 0x2E
0x34 0x2E 0x30 0x0D 0x0A
```

Response:

```
OK\r\n
0x4F  0x4B  0x0D 0x0A
```

8.31 SNMP Get Stats

Description:

This command is given, when get request received by SNMP agent (module) and issued by SNMP server. Whenever SNMP server sends a SNMP get request, module gives this command to host, to get the network (TCP, IP, UDP, ICMP etc.) statistics.

Command Format:

AT Mode:

```
at+rsi_snmp_get_stats\r\n
```

Binary Mode:

N/A

Command Parameters:

N/A

Response:

AT Mode:

Result Code	Description
<OK><rsi_total_packet_send_requests><rsi_total_packets_sent><rsi_total_bytes_sent><rsi_total_packets_received><rsi_total_packets_delivered><rsi_total_bytes_received><rsi_packets_forwarded><rsi_packets_reassembled><rsi_reassembly_failures><rsi_invalid_packets><	Network (TCP, IP, UDP, ICMP etc.) statistics of module.

Result Code	Description
rsi_ip_invalid_transmit_packets><rsi_invalid_receive_address><rsi_unknown_protocols_received><rsi_transmit_resource_errors><rsi_transmit_no_route_errors><rsi_receive_packets_dropped><rsi_receive_checksum_errors><rsi_send_packets_dropped><rsi_total_fragment_requests><rsi_successful_fragment_requests><rsi_fragment_failures><rsi_total_fragments_sent><rsi_total_fragments_received><rsi_arp_requests_sent><rsi_arp_requests_received><rsi_arp_responses_sent><rsi_arp_responses_received><rsi_arp_aged_entries><rsi_arp_invalid_messages><rsi_arp_static_entries><rsi_udp_packets_sent><rsi_udp_bytes_sent><rsi_udp_packets_received><rsi_udp_bytes_received><rsi_udp_invalid_packets><rsi_udp_no_port_for_delivery><rsi_udp_receive_packets_dropped><rsi_udp_checksum_errors><rsi_tcp_packets_sent><rsi_tcp_bytes_sent><rsi_tcp_packets_received><rsi_tcp_bytes_received><rsi_tcp_invalid_packets><rsi_tcp_receive_packets_dropped><rsi_tcp_checksum_errors><rsi_tcp_connections><rsi_tcp_passive_connections><rsi_tcp_active_connections><rsi_tcp_disconnections><rsi_tcp_connections_dropped><rsi_tcp_retransmit_packets><rsi_tcp_resets_received><rsi_tcp_resets_sent><rsi_icmp_total_messages_received><rsi_icmp_checksum_errors><rsi_icmp_invalid_packets><rsi_icmp_unhandled_messages><ip_pings_sent><rsi_ping_timeouts><rsi_ping_threads_suspended><rsi_ping_responses_received><rsi_pings_received><rsi_pings_responded_to><rsi_igmp_invalid_packets><rsi_igmp_reports_sent><rsi_igmp_queries_received><rsi_igmp_checksum_errors><rsi_igmp_groups_joined><rsi_icmp_total_messages_sent>	

Binary Mode:

```
Struct rsi_snmp_stats
{
        unsigned long rsi_total_packet_send_requests;
        unsigned long rsi_total_packets_sent;
        unsigned long rsi_total_bytes_sent;
```

```
unsigned long rsi_total_packets_received;
unsigned long rsi_total_packets_delivered;
unsigned long rsi_total_bytes_received;
unsigned long rsi_packets_forwarded;
unsigned long rsi_packets_reassembled;
unsigned long rsi_reassembly_failures;
unsigned long rsi_invalid_packets;
unsigned long rsi_invalid_transmit_packets;
unsigned long rsi_invalid_receive_address;
unsigned long rsi_unknown_protocols_received;
unsigned long rsi_transmit_resource_errors;
unsigned long rsi_transmit_no_route_errors;
unsigned long rsi_receive_packets_dropped;
unsigned long rsi_receive_checksum_errors;
unsigned long rsi_send_packets_dropped;
unsigned long rsi_total_fragment_requests;
unsigned long rsi_successful_fragment_requests;
unsigned long rsi_fragment_failures;
unsigned long rsi_total_fragments_sent;
unsigned long rsi_total_fragments_received;
unsigned long rsi_arp_requests_sent;
unsigned long rsi_arp_requests_received;
unsigned long rsi_arp_responses_sent;
unsigned long rsi_arp_responses_received;
unsigned long rsi_arp_aged_entries;
unsigned long rsi_arp_invalid_messages;
unsigned long rsi_arp_static_entries;
unsigned long rsi_udp_packets_sent;
unsigned long rsi_udp_bytes_sent;
unsigned long rsi_udp_packets_received;
unsigned long rsi_udp_bytes_received;
unsigned long rsi_udp_invalid_packets;
unsigned long rsi_udp_no_port_for_delivery;
unsigned long rsi_udp_receive_packets_dropped;
unsigned long rsi_udp_checksum_errors;
unsigned long rsi_tcp_packets_sent;
```

```
        unsigned long rsi_tcp_bytes_sent;
        unsigned long rsi_tcp_packets_received;
        unsigned long rsi_tcp_bytes_received;
        unsigned long rsi_tcp_invalid_packets;
        unsigned long rsi_tcp_receive_packets_dropped;
        unsigned long rsi_tcp_checksum_errors;
        unsigned long rsi_tcp_connections;
        unsigned long rsi_tcp_passive_connections;
        unsigned long rsi_tcp_active_connections;
        unsigned long rsi_tcp_disconnections;
        unsigned long rsi_tcp_connections_dropped;
        unsigned long rsi_tcp_retransmit_packets;
        unsigned long rsi_tcp_resets_received;
        unsigned long rsi_tcp_resets_sent;
        unsigned long rsi_icmp_total_messages_received;
        unsigned long rsi_icmp_checksum_errors;
        unsigned long rsi_icmp_invalid_packets;
        unsigned long rsi_icmp_unhandled_messages;
        unsigned long rsi_pings_sent;
        unsigned long rsi_ping_timeouts;
        unsigned long rsi_ping_threads_suspended;
        unsigned long rsi_ping_responses_received;
        unsigned long rsi_pings_received;
        unsigned long rsi_pings_responded_to;
        unsigned long rsi_igmp_invalid_packets;
        unsigned long rsi_igmp_reports_sent;
        unsigned long rsi_igmp_queries_received;
        unsigned long rsi_igmp_checksum_errors;
        unsigned long rsi_igmp_groups_joined;
        unsigned long rsi_icmp_total_messages_sent;
}
```

Response Parameters:

All the network statistics related to TCP, UDP, IP, ICMP of size 276bytes will be given as response.

rsi_total_packet_send_requests – no. of ip_in requests send by module.

rsi_total_packets_sent – no. of ip packets send by module.

rsi_total_bytes_sent – no. of bytes send in ip packets

rsi_total_packets_received – no. of ip packets received by module;

rsi_total_packets_delivered – no. of ip packets delivered by module.

rsi_total_bytes_received – no. of bytes received in ip packets.

rsi_packets_forwarded – no. of ip packets sent by module, for which receiver address is not the final destination.

rsi_packets_reassembled – no. of ip packets successfully reassembled by module.

rsi_reassembly_failures – no. of ip packets failed to reassemble by module.

rsi_invalid_packets – no. of ip packets discarded due to wrong ip header at module.

rsi_invalid_transmit_packets – no. of invalid packets transmitted by module.

rsi_invalid_receive_address – no. of invalid packets received by module.

rsi_unknown_protocols_received – no. of ip packets discarded due to unsupported protocols.

rsi_transmit_resource_errors – no. of ip packets discarded at module due to lack of buffer-space.

rsi_transmit_no_route_errors - no. of ip packets discarded at module due to unable to find route to destination.

rsi_receive_packets_dropped – no. of received ip packets drooped at module.

rsi_receive_checksum_errors – no. of ip packets received with checksum error.

rsi_send_packets_dropped – no. of ip packets dropped at module without further transmission.

rsi_total_fragment_requests – no. of ip packets fragmented at this entity.

rsi_successful_fragment_requests - no. of ip packets that have been successfully fragmented.

rsi_fragment_failures – no. of ip packets that have been failed to fragment.

rsi_total_fragments_sent – no. of ip fragments that have been sent from module.

rsi_total_fragments_received – no. of ip fragments successfully received by module.

rsi_arp_requests_sent – no. of arp request frames sent from module.

rsi_arp_requests_received – no. of arp request frames receivced by module.

rsi_arp_responses_sent – no. of arp response frames sent from module.

rsi_arp_responses_received – no. of arp response frames received by module.

rsi_arp_aged_entries – no. of arp aged messages at arp table.

rsi_arp_invalid_messages – no. of arp invalid messages.

rsi_arp_static_entries – no. of arp static entries.

rsi_udp_packets_sent – no. of udp packets sent by module.

rsi_udp_bytes_sent – no. of udp frame bytes sent by module.

rsi_udp_packets_received – no. of UDP packets received by module.

rsi_udp_bytes_received – no. of UDP frame bytes received by module.

rsi_udp_invalid_packets – no. of invalid UDP packets received by module.

rsi_udp_no_port_for_delivery – no. of udp packets with no delivery port

rsi_udp_receive_packets_dropped – no. of UDP packets dropped at module without forwarding.

rsi_udp_checksum_errors – no. of UDP packets received with checksum error.

rsi_tcp_packets_sent – no. of TCP packets sent from module.

rsi_tcp_bytes_sent –no. of tcp packets bytes sent from module.

rsi_tcp_packets_received – no. of tcp packets received by module.

rsi_tcp_bytes_received – no. of tcp packets bytes received by module.

rsi_tcp_invalid_packets – no. of tcp packets with invalid header or bad checksum received by module.

rsi_tcp_receive_packets_dropped – no. of tcp packets dropped by module without forwarding.

rsi_tcp_checksum_errors – no. of tcp packets received with checksum error.

rsi_tcp_connections – no. of tcp connection formed with module.

rsi_tcp_passive_connections – no. of passive tcp connections formed with module.

rsi_tcp_active_connections – no. of active tcp connections formed with module.

rsi_tcp_disconnections – no. of tcp disconnections happened with module.

rsi_tcp_connections_dropped – no. of tcp connections dropped without forming, may be because of closed state.

rsi_tcp_retransmit_packets – no. of packets retransmitted from module.

rsi_tcp_resets_received – no. of resets received by module, to move to CLOSE state.

rsi_tcp_resets_sent – no. of tcp packets sent from module contains the RST flag.

rsi_icmp_total_messages_received – no. of ping frames sent and pings responded.

rsi_icmp_checksum_errors – no. of pings received with checksum errors;

rsi_icmp_invalid_packets – no. of invalid pings

rsi_icmp_unhandled_messages – no. of pings not handled at module, due to lack of buffers

rsi_pings_sent – no. of pings sent from module.

rsi_ping_timeouts – no. of ping timeouts happened at module.

rsi_ping_threads_suspended – no. of pings suspended at module without transmitting.

rsi_ping_responses_received – no. of ping responses received by module.

rsi_pings_received – no. of pings received by module.

rsi_pings_responded_to – no. of ping responses sent by module.

rsi_igmp_invalid_packets – no. of invalid IGMP packets received by module.

rsi_igmp_reports_sent – no. of igmp reports sent by module.

rsi_igmp_queries_received – no. of IGMP queries received by module.

rsi_igmp_checksum_errors – no. of igmp packets received with checksum error.

rsi_igmp_groups_joined – no. of IGMP groups formed at module.

rsi_icmp_total_messages_sent – no. of pings sent by module.

Possible error codes:

N/A

Relevance:

This message is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Example:

AT Mode:

AT+RSI_SNMP_GET_STATS\r\n

0x61 0x74 0X2B 0X72 0X73 0X69 0X5f 0X73 0X6E 0X6D 0X70 0X5F 0X67 0X65 0X74 0X5F 0X73 0X74 0X61 0X74 0X73 0X0D 0X0A

Response:

```
OK\0x02\0x00\0x00\0x00\0x02\0x00\0x00\0x000\0x03\0x00\0x00\0x0
2\0x00\0x00\0x00\0x02\0x00\0x00\0x00h\0x02\0x00\0x00\0x00\0x00
\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0
x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x0
0\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\
0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x
00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00
\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0
x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x0
0\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x02\0x00\0x00\0x00\0x03\
0x00\0x00\0x02\0x00\0x00\0x00X\0x02\0x00\0x00\0x00\0x00\0x00\0
x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x0
0\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\
0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x
00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00
\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0
x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x0
0\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\
0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x
00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00
\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0
x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x00\0x0
0\r\n
```

```
0x4f 0x4b 0x02 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x30 0x03
0x00 0x00 0x02 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x68 0x02
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

```
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x02 0x00 0x00 0x00 0x20 0x03 0x00 0x00 0x02 0x00
0x00 0x00 0x58 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x0d 0x0a
```

8.32 SNMP trap

Description:

This command is issued by the SNMP agent (running in module), to notify the management station of significant events . This command has to be issued whenever user wants to send snmp trap to snmp server.

Command Format:

AT Mode:

```
at+rsi_snmp_trap= < snmp_version >,< ip_version >,
                  < ipv4/ipv6 address >,< community >,
                  < trap_type >,<trap_oid>,
                  < elapsed_time>,<object_list_count>,
                  <snmp_buf>\r\n
```

Binary Mode:

#define RSI_SNMP_TRAP_BUFFER_LENGTH 1024

```
struct{
      uint8       snmp_version;
      uint8       ip_version[4];
      union{
            uint8             ipv4_address[4];
            uint32            ipv6_address[4];
      }destIPaddr;
      uint8       community[32];
      uint8       trap_type;
      uint8       elapsed_time[4];
      uint8       trap_oid[51];
      uint8       obj_list_count;
      uint8       snmp_buf[RSI_SNMP_TRAP_BUFFER_LENGTH]
}snmptrapFrameSnd;
```

Formate of snmp obect list variable:

```
typedef struct SNMP_OBJECT_DATA_STRUCT
{
  uint8           snmp_object_data_type[4];
  uint8           snmp_object_data_msw[4];
  uint8           snmp_object_data_lsw[4];
  uint8           snmp_ip_version[4];
  union{
    uint8   ipv4_address[4];
    uint8   ipv6_address[16];
  }snmp_nxd_address;
  uint8           snmp_object_octet_string_size[4];

} SNMP_OBJECT_DATA;

typedef struct SNMP_TRAP_OBJECT_STRUCT
{
    uint8               snmp_object_string_ptr[40];
    SNMP_OBJECT_DATA    snmp_object_data;
} SNMP_TRAP_OBJECT;
```

Command Parameters:

`snmp_version`: snmp version (1/2/3). Currently only version 2 is supported for SNMP trap.

`ip_version`: IP version (4 or 6)

`destIPaddr.ipv4_address`: ipv4 address based upon the ip_version 4 selected. Module ignores remaining 12 bytes in case of ip version 4.

`destIPaddr.ipv6_address`: ipv6 address based upon the ip_version 6 selected.

`community`: community name

`trap_type`: type of trap

0 - (coldStart)

1 - (warmStart)

2 - (linkDown)

3 - (linkUp)

4 - (authenticationFailure)

5 - (egpNeighborLoss)

6 - (User specific trap type)

`trap_oid`: trap oid of the user specific trap.

`elapsed time`: Total device time elapsed .

`obj_list_count`: Obeject variables list count

`snmp_buf`: snmp buf contains the array of snmp trap object variablestructures (`SNMP_TRAP_OBJECT`).

- User has to fill `snmp_buf` with number of `SNMP_TRAP_OBJECT(obj_list_count)` structure.
- Based on `snmp_object_data_type`, User need to fill `SNMP_TRAP_OBJECT` structure.
- To use Octet string object data type, user needs to fill length of the string in `snmp_object_octet_string_size` parameter.

  Based on string size user has to fill octect string. If string length is zero, Octet string has to be filled with NULL .

Note:

1) Maximum supported `length for snmpBuffer` is 1024.
2) Maximum supported `object list count` is 10.

Response:

AT Mode:

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

Binary Mode:

N/A

Response Parameters:

N/A

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C, 0xFF82,0x0100,0x0104,.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Note:

For trap_type value 0 – 5 trap_oid parameter has to be NULL character

Example:

AT Mode:

For ipv4:

1. Command usage for Integer Object data type:

```
at+rsi_snmp_trap=2,4,192.168.0.178,public,1,,1234,1,1.3.2.2.2.
1.1.2\000\000\000\000\000\n\000\000\000\016\320\334G\001\000\0
00\000x\350\360G\n\000\000\000\002\000\000\000\020\000\000\000
\f\000\000\000\004\000\000\000\270\r\001
\000\000\001\000\000\000\000\000#\001, '\000' <repeats 117
times>,\r\n
```

```
0x61  0x74  0x2b  0x72  0x73  0x69  0x5f  0x73  0x6e  0x6d
0x70  0x5f  0x74  0x72  0x61  0x70  0x3d  0x32  0x2c  0x34
0x2c  0x31  0x39  0x32  0x2e  0x31  0x36  0x38  0x2e  0x00
0x2e  0x31  0x37  0x38  0x2c  0x70  0x75  0x62  0x6c  0x69
0x63  0x2c  0x31  0x2c  0x2c  0x31  0x32  0x33  0x34  0x2c
0x31  0x2c  0x31  0x2e  0x33  0x2e  0x32  0x2e  0x32  0x2e
0x32  0x2e  0x31  0x2e  0x31  0x2e  0x32  0x00  0x00  0x00
0x00  0x00  0x0a  0x00  0x00  0x00  0x0e  0xd0  0xdc  0x47
```

```
0x01  0x00  0x00  0x00  0x78  0xe8  0xf0  0x47  0x0a  0x00
0x00  0x00  0x02  0x00  0x00  0x00  0x10  0x00  0x00  0x00
0x0c  0x00  0x00  0x00  0x04  0x00  0x00  0x00  0xb8  0x0d
0x01  0x20  0x00  0x00  0x01  0x00  0x00  0x00  0x00  0x00
0x23  0x01  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x0D

0x0A
```

2. Command usage for Octect string object data type:

```
at+rsi_snmp_trap=2,4,192.168.0.178,public,1,,1234,1,1.4.4.4.4.
4.4.3\000\000\000\000\000\n\000\000\000\016\320\334G\001\000\0
00\000x\350\360G\n\000\000\000\004", '\000' <repeats 11
times>, "\004\000\000\000\270\r\001
\000\000\001\000\000\000\000\000#\001\000\000\031\000\000\000S
ample SNMP Trap String!!, '\000' <repeats 111 times>,\r\n
```

```
0x61  0x74  0x2b  0x72  0x73  0x69  0x5f  0x73  0x6e  0x6d
0x70  0x5f  0x74  0x72  0x61  0x70  0x3d  0x32  0x2c  0x34
0x2c  0x31  0x39  0x32  0x2e  0x31  0x36  0x38  0x2e  0x00
0x2e  0x31  0x37  0x38  0x2c  0x70  0x75  0x62  0x6c  0x69
0x63  0x2c  0x31  0x2c  0x2c  0x31  0x32  0x33  0x34  0x2c
0x31  0x2c  0x31  0x2e  0x34  0x2e  0x34  0x2e  0x34  0x2e
0x34  0x2e  0x34  0x2e  0x34  0x2e  0x33  0x00  0x00  0x00
0x00  0x00  0x0a  0x00  0x00  0x00  0x0e  0xd0  0xdc  0x47
0x01  0x00  0x00  0x00  0x78  0xe8  0xf0  0x47  0x0a  0x00
0x00  0x00  0x04  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x04  0x00  0x00  0x00  0xb8  0x0d
0x01  0x20  0x00  0x00  0x01  0x00  0x00  0x00  0x00  0x00
0x23  0x01  0x00  0x00  0x19  0x00  0x00  0x00  0x53  0x61
0x6d  0x70  0x6c  0x65  0x20  0x53  0x4e  0x4d  0x50  0x20
0x54  0x72  0x61  0x70  0x20  0x53  0x74  0x72  0x69  0x6e
0x67  0x21  0x21  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00

0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x0D  0x0A
```

For ipv6:

```
at+rsi_snmp_trap=2,6,2001:db8:1:0:0:0:0:123,public,1,,1234,1,1
.4.4.4.4.4.4.3\000\000\000\000\000\n\000\000\000\016\320\334G\
001\000\000\000x\350\360G\n\000\000\000\004", '\000' <repeats
11 times>, "\004\000\000\000\270\r\001
\000\000\001\000\000\000\000\000#\001\000\000\031\000\000\000S
ample SNMP Trap String!!, '\000' <repeats 111 times>,\r\n
```

```
0x61  0x74  0x2b  0x72  0x73  0x69  0x5f  0x73  0x6e  0x6d
0x70  0x5f  0x74  0x72  0x61  0x70  0x3d  0x32  0x2c  0x36
0x2c  0x32  0x30  0x30  0x30  0x31  0x3A  0x44  0x42  0x38
0x3A  0x31  0x3A  0x31  0x32  0x33  0x2c  0x2c  0x70  0x75
0x62  0x6c  0x690x63  0x2c  0x31  0x2c  0x2c  0x31  0x32
0x33  0x34  0x2c  0x31  0x2c  0x31  0x2e  0x34  0x2e  0x34
0x2e  0x34  0x2e  0x34  0x2e  0x34  0x2e  0x34  0x2e  0x33
0x00  0x00  0x00  0x00  0x00  0x0a  0x00  0x00  0x00  0x0e
0xd0  0xdc  0x47  0x01  0x00  0x00  0x00  0x78  0xe8  0xf0
0x47  0x0a  0x00  0x00  0x00  0x04  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x04  0x00  0x00
0x00  0xb8  0x0d  0x01  0x20  0x00  0x00  0x01  0x00  0x00
0x00  0x00  0x00  0x23  0x01  0x00  0x00  0x19  0x00  0x00
0x00  0x53  0x61  0x6d  0x70  0x6c  0x65  0x20  0x53  0x4e
0x4d  0x50  0x20  0x54  0x72  0x61  0x70  0x20  0x53  0x74
0x72  0x69  0x6e  0x67  0x21  0x21  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00   0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00 0x00   0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00

0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x0D  0x0A
```

Response:

```
OK\r\n
```

```
0x4F  0x4B  0x0D 0x0A
```

8.33 Open Socket

Description:

This command opens a TCP/UDP/SSL/Websocket client socket, a Listening TCP/UDP/SSL socket .

Note:

1. A maximum of 10 sockets can be opened. Range of socket handles are between 1 to 10.

2. Module supports maximum of 2 SSL sockets. These can be either client or server sockets. When HTTPS server is enabled then user can open only 1 SSL socket.

3. Module supports maximum of 8 non SSL Web sockets and 2 SSL Web sockets.

4. Each SSL/Web socket will occupy one TCP socket. SSL is supported only in Wi-Fi client mode

5. If SSL is enabled, module will use same SSL version for all SSL sockets until module reboots.

6. If SSL is enabled module will use same CA certificate for both SSL server socket and SSL client socket until module reboots.

7. Above case is valid only when BIT(27) is not set in tcp_ip_feature_bit_map (module will use SSL certificates from FLASH)

8. If BIT(27) (`SSL certificate on to the RAM feature`) is set in tcp_ip_feature_bit_map, module will only support either SSL server socket or SSL client socket.

Command Format:

AT Mode:

To open TCP/SSL/Web socket over IPv4:

```
at+rsi_tcp=< destIPaddr >,< destSocket >,< moduleSocket >,
< tos >,< ssl_ws_enable >,< ssl_ciphers >,
< webs_resource_name >,< webs_host_name >,
      <tcp_retry_count>,<socket_bitmap>,
      <rx_window_size>, <tcp_keepalive_timeout>,
      < vap_id >,<tcp_retransmission_timer>\r\n
```

To open TCP/SSL/Web socket over IPv6:

```
at+rsi_tcp6=< destIPaddr >,< destSocket >,< moduleSocket >,
< tos >,< ssl_ws_enable >,< ssl_ciphers >,
< webs_resource_name >,< webs_host_name >,
            <tcp_retry_count>,<socket_bitmap>,
            <rx_window_size>,<tcp_keepalive_timeout>,
            < vap_id >,<tcp_retransmission_timer>\r\n
```

To open TCP/SSL server socket over IPv4:

```
at+rsi_ltcp=< moduleSocket >,< max_count >,< tos >,
< ssl_ws_enable >,< ssl_ciphers >,
            <tcp_retry_count>,<socket_bitmap>,
            <rx_window_size>,<tcp_keepalive_timeout>,
            < vap_id >,<tcp_retransmission_timer>\r\n
```

Note:

The tcp_keepalive_timeout is denoted in seconds.

To open TCP/SSL server socket over IPv6:

```
at+rsi_ltcp6=< moduleSocket >,< max_count >,< tos >,
< ssl_ws_enable >,< ssl_ciphers >,
            <tcp_retry_count>,<socket_bitmap>,
            <rx_window_size>,<tcp_keepalive_timeout>,
            < vap_id >,<tcp_retransmission_timer>\r\n
```

To open LUDP socket over IPv4:

```
at+rsi_ludp=< moduleSocket >,< tos >,<socket_bitmap>,
< vap_id >\r\n
```

To open LUDP socket over IPv6:

```
at+rsi_ludp6=< moduleSocket >,< tos >,<socket_bitmap>,
< vap_id >\r\n
```

Binary Mode:

```
#define WEBS_MAX_URL_LEN   51
#define WEBS_MAX_HOST_LEN  51
struct {
      uint8  ip_version[2];
      uint8  socketType[2];
      uint8  moduleSocket[2];
      uint8  destSocket[2];
      union{
            uint8   ipv4_address[4];
            uint32   ipv6_address[4];
      }destIPaddr;
      uint8 max_count[2];
```

```
        uint8  tos[4];
        uint8  ssl_ws_enable;
        uint8  ssl_ciphers;
        uint8  webs_resource_name[WEBS_MAX_URL_LEN];
        uint8  webs_host_name[WEBS_MAX_HOST_LEN];
        uint8  tcp_retry_count;
        uint8  socket_bitmap;
        uint8  rx_window_size;
        uint8  tcp_keepalive_initial_time[2];
        uint8  vap_id;
        uint8  tcp_retransmission_timer;
} socketFrameSnd;
```

Command Parameters:

Note:

`ip_version, socketType` fields are available only in Binary Mode.

`ip_version`: IPversion used, either 4 or 6.

`socketType`: Type of the socket

0– TCP/SSL Client

2– TCP/SSL Server (Listening TCP)

4– Listening UDP

`moduleSocket`: Port number of the socket in the module. Value ranges from 1024 to 49151.

`destSocket`: destination port. Value ranges from 1024 to 49151. Ignored when TCP server or Listening UDP sockets are to be opened.

`destIPaddr.ipv4_address`: IP Address of the target server. Ignored when TCP server or Listening UDP sockets are to be opened. If ip_version is 4 then only first four bytes of the ipv4_address is filled rest twelve bytes will be 0.

`destIPaddr.ipv6_address`: IPv6 Address of the target server. Ignored when TCP server or Listening UDP sockets are to be opened. All 16 bytes are filled if ip_version is 6.

`max_count`: Maximum number of clients can be connect in case of LTCP.

Note:

Module supports maximum 2 SSL sockets, so max_count should be less than or equal to 2 in case of LTCP. If max_count is 2 then host can create only one SSL based LTCP socket with two clients support.

This field can be ignored if the socket type is other than 2(TCP server).

`tos`: type of service field. Possible values are 0-7.

TOS value	Description
0	Best Effort
1	Priority
2	Immediate
3	Flash - mainly used for voice signaling
4	Flash Override
5	Critical - mainly used for voice RTP
6	Internet
7	Network

Table 27: TOS Values

`ssl_ws_enable`: This field is used to enable following:

Possible values:

0 – To open TCP socket.

BIT(0) - To open SSL client socket.

BIT(1) - To open Web socket client.

BIT(2) - To open SSL socket with TLS 1.0 version.

BIT(3) - To open SSL socket with TLS 1.2 version.

BIT(7) – To open High performance TCP RX socket.

Examples:

`ssl_ws_enable` value should be

- '0' to open normal TCP socket
- '1' to open SSL over TCP socket. By default module will open SSL socket which support both TLS 1.0 and TLS 1.2.
- '2' to open web socket client over TCP socket

- '3' to open web socket over SSL TCP socket
- '5' to open SSL over TCP socket with TLS 1.0 version
- '9' to open SSL over TCP socket with TLS 1.2 version
- '7' to open web socket client over SSL TCP socket with TLS 1.0 version
- '11' to open web socket client over SSL TCP socket with TLS 1.2 version
- '128' to open High performance TCP socket
- '129' to open High performance TCP socket over SSL
- '130' to open High performance web socket TCP client socket
- '131' to open High performance web socket TCP client socket over SSL

`ssl_ciphers`: 1 byte bitmap used to select the various cipher modes. This field is optional and the possible values are listed below:

BIT(0): 1: TLS_RSA_WITH_AES_256_CBC_SHA256

BIT(1): 2: TLS_RSA_WITH_AES_128_CBC_SHA256

BIT(2): 4: TLS_RSA_WITH_AES_256_CBC_SHA

BIT(3): 8: TLS_RSA_WITH_AES_128_CBC_SHA

BIT(4): 16: TLS_RSA_WITH_AES_128_CCM_8

BIT(5): 32: TLS_RSA_WITH_AES_256_CCM_8

BIT(6): 64: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256

BIT(7): 128:TLS_DHE_RSA_WITH_AES_128_CBC_SHA256

BIT(8): 256:TLS_DHE_RSA_WITH_AES_256_CBC_SHA

BIT(9): 512:TLS_DHE_RSA_WITH_AES_128_CBC_SHA

These values can be OR'd together to select multiple ciphers. To select all the ciphers, either all bits can be set or alternatively, 0 can be passed.

`webs_resource_name`:Web socket resource name.

A string of 50 characters is the maximum possible input to this variable

`webs_host_name`:Web socket host name.

A string of 50 characters is the maximum possible input to this variable

`tcp_retry_count`: To configure tcp retransmissions count

`socket_bitmap`: To configure socket bit map

BIT(0) : Set to enable synchronous data read

Module sends data to host only after receiving data read request from host.

BIT(1) : To open a listening TCP socket, on which to accept client connection host need to provide accept command.

BIT(2): Set to enable TCP ACK indication. This bit is valid for TCP/SSL Client, TCP/SSL Server (Listening TCP) and Web Sockets.

When this bit is enabled module gives an TCP ACK indication(Frame type 0xAB) to host, when it receives TCP ACK from remote peer. Host has to send next data packet to module only after receiving this TCP ACK indication.

BIT(3): If this bit is set module handles small size received packets effectively.

Recommended to set for the sockets which receives small size packets .

BIT(4): Set to configure TCP RX window size. This bit is valid for TCP/SSL Client, TCP/SSL Server (Listening TCP).

When this bit is enabled module opens socket with RX window size based on the value provided in `rx_window_size`field

`rx_window_size`: This field is used to configure the RX window size for the TCP socket. Possible values are 1 to 15 based on the availability of memory.

If the above bit is enable, then TCP RX window size will be

"220+(rx_window_size-1)*512"bytes.

`tcp_keepalive_initial_time`: This field is used to configure TCP keepalive initial timeout in seconds.

`Vap_id`: Possible values are 0 and 1.

0 - Interface index for client

1 - Interface index for AP

`tcp_retransmission_timer`: This field is used to configure TCP re-transmission timer.

By default this value is 10 i.e, TCP re-transmission timer is 1 sec. User can configure as shown below.

e.g. 3 for 300 msec , 4 for 400 msec, 5 for 500 msec and 10 for 1000 msec .....etc.

Response:

AT Mode:

For TCP/SSL/Web socket over IPv4/IPv6:

Result Code	Description
OK< ip_version>< socketType>< socketDescriptor>< moduleSocket>< ipv4_addr /ipv6_addr> < mss>< window_size>	Successful execution of the command.
ERROR<Error code>	Failure

For TCP/SSL server socket over IPv4/IPv6:

or

For LUDP socket over IPv4/IPv6:

Result Code	Description
OK< ip_version>< socketType>< socketDescriptor>< moduleSocket>< ipv4_addr/ipv6_addr>	Successful execution of the command.
ERROR<Error code>	Failure

Binary Mode:

```
typedef struct {
uint8  ip_version[2];
      uint8  socketType[2];
      uint8  socketDescriptor[2];
      uint8  moduleSocket[2];
      union{
                  uint8   ipv4_addr[4];
                  uint32   ipv6_addr[4];
}moduleIPaddr;
      uint8  mss[2];
      uint8  window_size[4];
      } rsi_socketFrameRcv;
```

Response Parameters:

`ip_version(2 bytes)`: IP version used, either 4 or 6.

`socketType(2 bytes)`: Type of the created socket.

0–TCP/SSL Client

2–TCP/SSL Server (Listening TCP)

4–Listening UDP

`socketDescriptor(2 bytes)`: Created socket's descriptor or handle, starts from 1. sockeDescriptor ranges from 1 to 10. The first socket opened will have a socket descriptor of 1, the second socket will have 2 and so on.

`moduleSocket(2 bytes)`: Port number of the socket in the module.

`moduleIPaddr.ipv4_address(16 bytes)`: The IPv4 address of the module. Only first four bytes of ipv4_address is filled rest 12 bytes are '0' in case of IPv4.

`moduleIPaddr.ipv6_address(16 bytes)`: The IPv6 address of the module in case of IPv6.

`mss(2 bytes)`: maximum segment size of the remote peer. In case of Ludp/Ltcp this field will not present.

`window_size(4 bytes)` **:** Window size of the remote peer. In case of Ludp/Ltcp this field will not present.

Possible error codes:

Possible error codes are 0xBB46,0xBB22,0xBB23,0xBB33,0xBB34,0xBB35,0xBB36,0xBB45,0xBB46,0x0015,0x0021,0x0025,0x002C,0xFF74,0xBBD3,0xBBD2,0xBBD1,0xFF80,

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Example:

AT Mode:

To TCP socket over IPv4:

at+rsi_tcp=192.168.40.10,8000,1234,0,1,0\r\n

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x74 0x63 0x70 0x3D 0x31
0x39 0x32 0x2E 0x31 0x36 0x38 0x2E 0x34 0x30 0x2E 0x31 0x30
0x2C 0x38 0x30 0x30 0x30 0x2C 0x31 0x32 0x33 0x34 0x2C 0x30
0x2C 0x31 0x2C 0x30 0x0D 0x0A
```

Response:

```
OK< ip_version =0x04 0x00>< socketType =0x0000 ><
socketDescriptor =0x0001>< moduleSocket =0x4d2>< ipv4_addr=
0xC0 0xA8 0x28 0x120x00(12 times)>< mss =0xB4 0x05><
window_size =0x00 0x00 0x01 0x0>\r\n
```

```
0x4F 0x4B 0x04 0x00 0x00 0x00 0x01 0x00 0xd2 0x04 0xC0 0xA8
0x28 0x12 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00
```

```
0xB4 0x05 0x00 0x00 0x01 0x00 0x0D 0x0A
```

To open Web socket over IPv4:

```
at+rsi_tcp=174.129.224.73,80,1234,0,2,0,,echo.websocket.org\r\
n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x74 0x63 0x70 0x3D 0x31
0x37 0x34 0x2E 0x31 0x32 0x39 0x2E 0x32 0x32 0x34 0x2E 0x37
0x33 0x2C 0x38 0x30 0x2C 0x35 0x30 0x30 0x30 0x2C 0x30 0x2C
0x32 0x2C 0x30 0x2C 0x2C 0x65 0x63 0x68 0x6F 0x2E 0x77 0x65
0x62 0x73 0x6F 0x63 0x6B 0x65 0x74 0x2E 0x6F 0x72 0x67 0x3C
0x43 0x52 0x3E 0x3C 0x4C 0x46 0x3E
```

Response:

```
OK< ip_version =0x04 0x00>< socketType =0x0000 ><
socketDescriptor =0x0001>< moduleSocket =0x4d2>< ipv4_addr=
0xC0 0xA8 0x28 0x120x00(12 times)>< mss =0xB4 0x05><
window_size =0x00 0x00 0x01 0x0>\r\n
```

```
0x4F 0x4B 0x04 0x00 0x00 0x00 0x01 0x00 0xd2 0x04 0xC0 0xA8
0x28 0x12 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0xB4 0x05 0x00 0x00 0x01 0x00 0x0D 0x0A
```

To open Web socket over IPv6:

```
at+rsi_tcp6=2001:DB8:1:0:0:0:0:153,8000,1234,2,1,0\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x74 0x63 0x70 0x36 0x3D
0x32 0x30 0x30  0x31 0x3A 0x44 0x42 0x38 0x3A 0x31 0x3A 0x30
0x3A 0x30 0x3A 0x30 0x3A 0x30 0x3A 0x31 0x35 0x33  0x2C 0x38
0x30  0x30 0x30 0x2C 0x31 0x32 0x33 0x34 0x2C 0x32  0x2C 0x31
0x2C 0x30 0x0D 0x0A
```

Response:

```
OK< ip_version =0x06 0x00>< socketType =0x0000 ><
socketDescriptor =0x0001>< moduleSocket =0x4d2>< ipv6_addr=
addr 0xB8 0x0D 0x01 0x20 0x00 0x00 0x01 0x00 0x00 0x00 0x00
0x00 0x53 0x01 0x00 0x00 >< mss =0xB4 0x05>< window_size =0x00
0x00 0x01 0x00>\r\n
```

```
0x4F 0x4B 0x06 0x00 0x00 0x00 0x01 0x00 0xd2 0x04  0xB8 0x0D
0x01 0x20 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x53 0x01
0x00 0x00  0xB4 0x05 0x00 0x00 0x01 0x00 0x0D 0x0A
```

To open LTCP socket over IPv4 :

```
at+rsi_ltcp=1234,5,7,1,0r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6C 0x74 0x63 0x70 0x3D
0x31 0x32  0x33 0x34 0x2C 0x35 0x2C 0x37 0x2C 0x31 0x2C 0x30
0x0D 0x0A
```

Response:

```
OK<ip_version><socket_type><socket_handle><Lport><module_ipv4a
ddr>\r\n
```

```
OK< ip_version =0x04 0x00>< socketType =0x0002 ><
socketDescriptor =0x0001>< moduleSocket =0x4d2>< ipv4_addr=
0xC0 0xA8 0x12 0xD2 0x00(12 times) > \r\n
```

```
0x4F 0x4B 0x02 0x00 0x01 0x00 0xd2 0x04 0xC0 0xA8 0x28 0x12
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x0D 0x0A
```

To open LTCP socket over IPv6:

```
at+rsi_ltcp6=1234,5,7,1,0r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6C 0x74 0x63 0x70 0x3D
0x31 0x32  0x33 0x34 0x2C 0x35 0x2C 0x37 0x2C 0x31 0x2C 0x30
0x0D 0x0A
```

Response:

```
OK<ip_version><socket_type><socket_handle><Lport><module_ipv6_
addr>\r\n
```

```
OK< ip_version =0x06 0x00>< socketType =0x0002 ><
socketDescriptor =0x0001>< moduleSocket =0x4d2>< ipv6_addr=
0xB8 0x0D 0x01 0x20 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00
0x36 0x01 0x00 0x00 > \r\n
```

```
0x4F 0x4B 0x06 0x00 0x02 0x00 0x01 0x00 0xd2 0x04 0xB8 0x0D
0x01 0x20 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x36 0x01
0x00 0x00 0x0D 0x0A
```

To open LUDP socket over IPv4:

```
at+rsi_ludp=1234, 7\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F  0x6C  0x75  0x64  0x70
0x3D  0x31  0x32  0x33  0x34 0x2C 0x7 0x0D  0x0A
```

Response:

```
OK< ip_version =0x04 0x00>< socketType =0x0004 ><
socketDescriptor =0x0001>< moduleSocket =0x4d2>< ipv4_addr=
0xC0 0xA8 0x28 0x12  0x00(12 times) > \r\n
```

```
0x4F 0x4B 0x04 0x00 0x04 0x00 0x01 0x00 0xd2 0x04 0xC0 0xA8
0x28 0x12  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x0D 0x0A
```

To open LUDP socket over IPv6:

```
at+rsi_ludp6=1234,5\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F  0x6C  0x75  0x64  0x70
0x36 0x3D  0x31  0x32  0x33  0x34 0x2C 0x35 0x0D  0x0A
```

Response:

```
OK< ip_version =0x06 0x00>< socketType =0x0004 ><
socketDescriptor =0x0001>< moduleSocket =0x4d2>< ipv6_addr=
0xB8 0x0D 0x01 0x20 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00
0x36 0x01 0x00 0x00 > \r\n
```

```
0x4F 0x4B 0x06 0x00 0x04 0x00 0x01 0x00 0xd2 0x04 0xB8 0x0D
0x01 0x20 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x36 0x01
0x00 0x00  0x0D 0x0A
```

8.34 TCP Socket Connection Established

Description:

If a server TCP socket is opened in the module, the socket remains in listening state till the time the remote terminal opens and binds a corresponding client TCP socket. Once the socket binding is done, the module sends an asynchronous message to the Host to indicate that its server socket is now connected to a client socket

Note:

If SSL is enabled, module will use same SSL version for all SSL sockets until module reboots.

Command Format:

AT Mode:

N/A

Binary Mode:

N/A

Command Parameters:

N/A

Response:

AT Mode:

Result Code	Description
AT+RSI_LTCP_CONNECT=<ip_version><socket>,<fromPortNum>,<ipv4_address/ipv6_address>< mss>< window_size>< SrcPortNum>	Asynchronous message from module to host on tcp connection establishment .

Binary Mode:

```
struct {
      uint8  ip_version[2];
      uint8  socket[2];
      uint8  fromPortNum[2];
      union{
```

```
            uint8  ipv4_address[4];
            uint32  ipv6_address[4];
      }dst_ip_address;
      uint8     mss[2];
      uint8     window_size[4];
      uint8     SrcPortNum[2];
} rsi_recvLtcpEst;
```

Response Parameter:

`ip_version(2 bytes)` : IP version used, either 4 or 6.

`Socket(2 bytes)` : Socket descriptor of the server TCP socket.

`fromPortNum(2 bytes)` : Port number of the remote socket

`dst_ip_address .ipv4_address(16 bytes)` : Remote IPv4 address. Only first four bytes of ipv4_address are filled, rest 12 bytes are zero.

`dst_ip_address.ipv6_address(16 bytes)` : Remote IPv6 address

`mss(2 bytes)` : Maximum segment size of remote peer.

`window_size(4 bytes)` : Window size of the remote peer.

`SrcPortNum(2 bytes)`: Source Port Number to which client is connected.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

8.35 Query a Listening Socket's Active Connection Status

Description:

This command is issued when a listening/server TCP/SSL socket has been opened in the module, to know whether the socket got connected to a client socket.

Command Format:

AT Mode:

```
at+rsi_ctcp=< socketDescriptor >\r\n
```

Binary Mode:

```
struct {
      uint8 socketDescriptor[2];
} queryLtcpConnStatusFrameSnd;
```

Response:

AT Mode:

Result Code	Description
OK< socketDescriptor>< ip_version>< ipv4_address / ipv6_address>< destPort>	Successful Execution of Command.
ERROR <Error>	Failure

Binary Mode:

```
struct {
            uint8   socketDescriptor[2];
            uint8   ip_version[2];
union{
                  uint8   ipv4_address[4];
                  uint8   ipv6_address[16];
}dest_ip;
uint8   destPort[2];
      } rsi_LTCPConnStatusFrameRcv;
```

Response Parameters:

`ip_version(2 bytes)` **:** IP version used, either 4 or 6.

`socketDescriptor(2 bytes)` **:** Socket handle for an already open listening TCP/SSL socket in the module.

`destIPaddr(16 bytes)` : Destination IPv4/IPv6 address of the remote peer whose socket is connected. Last 12 bytes will be filled to '0' incase of IPv6.

`destPort(2 bytes)` **:** Port number of the remote peer client which is connected

Possible Error codes:

Possible error codes are 0x0021, 0x0025, 0x002C, 0xFF86,0xFFFA,0xFF82.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1 , 2 or 6.

Example:

AT Mode:

Response:

```
OK<socketDescriptor =7><ipv4_address =192.168.40.10>
<destPort =8001> \r\n


0x4F 0x4B 0x07  0x000xC0   0xA8   0x28   0x0A0x41 0x1F
0x0D  0x0A
```

8.36 Close Socket

Description:

This command closes a TCP/LTCP/UDP/SSL/Web socket in the module.

Command Format:

AT Mode:

```
at+rsi_cls=< socketDescriptor >,< port_number >\r\n
```

Binary Mode:

```
struct {
      uint8 socketDescriptor[2];
      uint8      port_number[2];
} socketCloseFrameSnd;
```

Parameters :

`socketDescriptor`: Socket descriptor of the socket to be closed.

`port_number`: This field is valid only for LTCP socket.

When this filed is mentioned module closes all LTCP connections which are opened with provided port number.

Note:

1) This field will be ignored in case of closing UDP/TCP client sockets.
2) In order to use port based socket close to close all LTCP sockets user has to set socket_handle as zero.
3) For web socket: If close command is given socket will be closed without waiting for server to initiate web socket close.

In order get graceful closure , host has to issue data send command with an opcode of 0x8 and fin bit set and with an dummy data. This dummy data will be ignored by server side web socket(Example: at+rsi_snd=1,0,0,136,\r\n in AT mode).

In this case module sends web socket close frame to server. On receiving this frame server initiates web socket close. After successful socket close exchanges, module gives remote terminate indication to host.

Response :

AT Mode:

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

NOTE: In the case of TCP socket, when a remote peer closes the socket connection, the module sends the "AT+RSI_CLOSE<socket_handle><number of bytes sent>\r\n" message to the Host. This is an asynchronous message sent from module to host and not the response of a specific command. Socket_handle is sent in 2 bytes, number of bytes sent is 4 bytes in hex. The least significant byte is returned first. AT+RSI_CLOSE is returned in uppercase and ASCII format.

Binary Mode:

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

Response Parameters :

`socketDsc(2 bytes)`: Socket descriptor of the socket closed.

`bytesSent(4 bytes)` **:** Number of bytes sent successfully on that socket.

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C, 0xFF86,0xBB35,0xBB27,0xBB42

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Example:

AT Mode:

To close the socket with handle 1, the command is

```
at+rsi_cls=1\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x63 0x6C 0x73 0x3D 0x31
0x0D 0x0A
```

Response:

```
OK<socket_handle><number of bytes sent>\r\n
```

```
0x4F 0x4B  0x01 0x00 0x01 0x02 0x03 0x04 0x0D 0x0A
```

8.37 Send Data

This section explains how to send data from the host to the module.

8.37.1 Send Data in AT mode

Description:

This command is used to send data from the host to the module, to be transmitted over a wireless media in AT mode.

Note:

1) Maximum data which can be send over TCP/LTCP socket is 1460 Bytes
2) Maximum data which can be send over LUDP socket is 1472 Bytes
3) Maximum data which can be send over Web socket is 1452 Bytes
4) Maximum data which can be send over TCP-SSL/LTCP-SSL is 1390 Bytes
5) Maximum data which can be send over Web socket - SSL is 1382 Bytes

Note: If you are using IPv6, the maximum size would be 1452 bytes, as IPv6's header size is 40 bytes vs. IPv4's 20 byte size (and either way, one must still allow 8 bytes for the UDP header).

Command Format:

To send data over IPv4:

```
at+rsi_snd=< socketDescriptor >,< sendBufLen >,
< destIPaddr >,< destPort >, < sendDataBuf >\r\n
```

To send data over IPv6:

```
at+rsi_snd6=< socketDescriptor >,< sendBufLen >,
```

```
< destIPaddr >,< destPort >, < sendDataBuf >\r\n
```

Command Parameters:

`socketDescriptor`– Socket handle of the socket over which data is to be sent.

`sendBufLen` – Length of the data that is getting transmitted, wrong parameter may cause module hang in some cases. In case of Web socket correct length is mandatory.

`destIPaddr`– Destination IPv4/IPv6 Address. Should be '0' in case of data transmitting on a TCP/LTCP/SSL socket.

`destPort`– Destination Port. Should be '0' in case of data transmitting on a TCP/LTCP/SSL socket.

In case of Web sockets, this field represents web socket information.

In web socket information seventh bit indicates the FIN packet and bit [3:0] indicates the opcode (type of the packet to be included in web socket header).

OPCODE should be as follows(Refer RFC 6455):

0 - Continuation frame

1 – Text frame

2 – Binary frame

[3-7] - Reserved for further non-control frames

8 - Connection close frame

9 - Ping frame

10 - Pong frame

[B-F] - Reserved for further control frames

FIN Bit should be as follows:

0: More web socket frames to be followed.

1: Final frame web socket message.

`sendDataBuf` – Actual data to be sent to be sent to the specified socket.

Response:

Result Code	Description

Result Code	Description
OK<length> (Or) OK<socket id ><length>	2 bytes length(2 bytes hex), length of data sent. 1 byte socket id ,2 bytes length(2 bytes hex), length of data sent.
ERROR<Error code>	Failure On a failure while sending the data on the TCP socket, if the error code indicates “TCP connection closed”, then the module closes the socket. Possible error codes are 0x0030,0xFFFE,0xFF7E,0xFFF8,0x003F ,0xFFF7.

When enable the socket BIT(2)(open socket)

Note:

1). When enable the socket bit map(2)(open socket) the send response gives 3bytes

User need to consider following for “snd” command in case of TCP_ACK(bit[2]) is enabled

User will get an immediate “OK<socket id ><length>\r\n” response for “snd” command. This indicates the “snd” command transaction happened successfully at the host interface level. This doesn’t mean that the packet is successfully transmitted to the remote peer. Module responds with “OK<socket id ><length>\r\n” and takes the next “snd” command till it has buffers to buffer those packets.

2). In case of SSL socket the response of send command gives length of data (Includes SSL data)on the TCP socket.

Note:

The parameter sendDataBuf contains the actual data not the ACII representations of the data.

User need to consider following for “snd” command in case of UART mode.

User will get an immediate “OK<socket id ><length>\r\n” response for “snd” command. This indicates the “snd” command transaction happened successfully at the host interface level. This doesn’t mean that the packet is successfully transmitted to the remote peer.

Module responds with “OK<socket id ><length>\r\n” and takes the next “snd” command till it has buffers to buffer those packets.

User need to take care that the data_len value that is given in “snd” command should be same as the number of bytes that are getting transmitted with “snd” command.

In TCP/IP bypass only < sendDataBuf > parameter is valid and remaining parameters are dummy user can send 0.

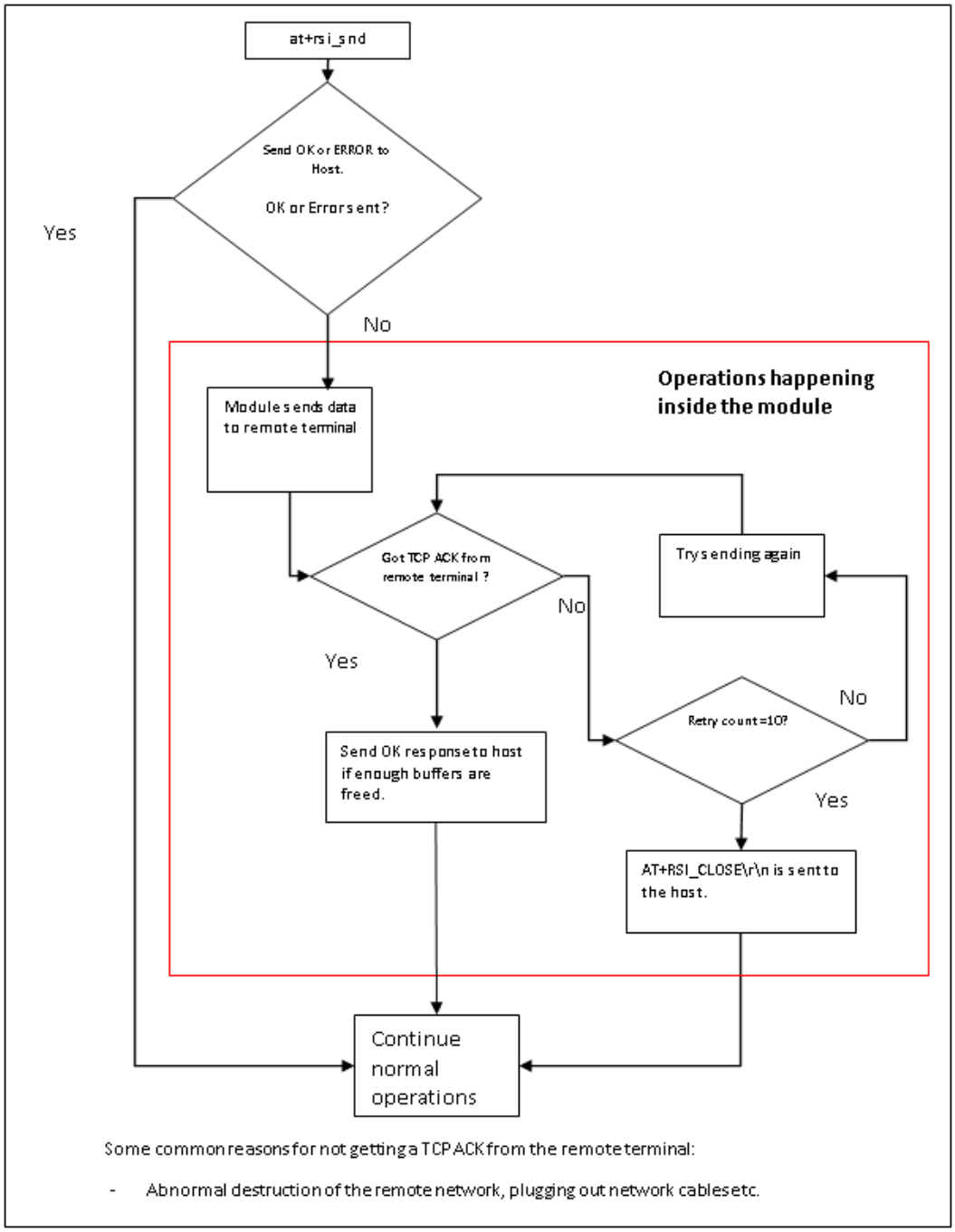


Figure 45: Send Operation

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 or 6.

NOTE on Byte Stuffing: The '\r\n' character sequence (0x0D, 0x0A in hex) is used to indicate the termination of an AT command. If the actual data to be sent from Host comprises of \r\n characters in sequence, the host should replace this set of characters with (0xDB) and (0xDC). If (0xDB) itself is part of the data then (0xDB 0xDD) has to be sent. If (0xDB 0xDC) itself is part of the data then (0xDB 0xDD 0xDC) has to be sent. If either 0xDD or 0xDC is not sent after 0xDB, then an error (-9) is sent.

Example 1 : If **0x41 0x42 0x43 0x0D 0x0A** is the actual data stream that needs to be sent then the command is

at+rsi_snd <hn><sz=5><Dip><Dport>< 0x41><0x42><0x43><0xDB><0xDC><0x0D><0x0A>

Example 2 : If **0x41 0x42 0x43 0x0D 0x0A 0x31 0x32** is the actual data stream that needs to be sent then the command is

at+rsi_snd <hn><sz=7><Dip><Dport>< 0x41><0x42><0x43><0xDB><0xDC><0x31><0x32><0x0D><0x0A>

Example 3 : If **0x41 0x42 0x43 0xDB 0x31 0x32** is the actual data stream that needs to be sent then the command is

at+rsi_snd <hn><sz=7><Dip><Dport>< 0x41><0x42><0x43><0xDB><0xDD><0x31><0x32><0x0D><0x0A>

Example 4: If **0x41 0x42 0x43 0xDB 0xDC 0x31 0x32** is the actual data that needs to be transmitted, then the command is

at+rsi_snd <hn><sz=8><Dip><Dport><0x41><0x42><0x43><0xDB><0xDD><0xDC><0x31><0x32><0x0D><0x0A>

Example 5: If **0x41 0x42 0x43 0x0D 0x0A 0xDB 0x31 0x32** is the actual data that needs to be transmitted, then the command is

at+rsi_snd <hn><sz=9><Dip><Dport><0x41><0x42><0x43><0xDB><0xDC><0xDB><0xDD><0x31><0x32><0x0D><0x0A>

Example 6: If **0x41 0x42 0x43 0x0D 0x0A 0xDB 0xDC 0x31 0x32** is the actual data that needs to be transmitted, then the command is

at+rsi_snd <hn><sz=10><Dip><Dport><0x41><0x42><0x43><0xDB><0xDC><0xDB><0xDD><0xDC><0x31><0x32><0x0D><0x0A>

at+rsi_snd and HTTP commands require byte stuffing to be done by the Host before sending to the module. There are NO other commands (from Host to module) that require byte stuffing. There are NO responses (from module to Host) that are byte stuffed by module before giving to Host.

Table 28: Data Stuffing

Example:

Command:

To send a data stream 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A over a TCP socket

at+rsi_snd=1,10,0,0, 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A\r\n

```
0x61  0x74  0x2B  0x72 0x73 0x69    0x5F  0x73  0x6E  0x64
0x3D 0x31 0x2C 0x31 0x30 0x2C 0x30  0x2C  0x30  0x2C  0x01
0x02  0x03  0x04  0x05  0x06  0x07  0x08  0x09  0x0A  0x0D
0x0A
```

To send a data stream 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A over a UDP socket to a destination IP 192.168.1.20 and destination port 8001

```
at+rsi_snd=1,10,192.168.1.20,8001, 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A\r\n
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x73  0x6E  0x64
0x3D  0x31  0x2C  0x31  0x30  0x2C  0x31  0x39  0x32  0x2E
0x31  0x36  0x38  0x2E  0x31  0x2E  0x32  0x30  0x2C  0x38
0x30  0x30  0x31  0x2C  0x01  0x02  0x03  0x04  0x05  0x06
0x07  0x08  0x09  0x0A  0x0D  0x0A
```

To send a stream “abcdefghij” over a Multicast socket

```
at+rsi_snd=1,10,239.0.0.0,1900,abcdefghij\r\n
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x73 0x6E 0x64 0x3D 0x31
0x2C 0x31 0x30 0x2C 0x32 0x33 0x39 0x2E 0x30 0x2E 0x30 0x2E
0x30 0x2C 0x31 0x39 0x30 0x30 0x2C0x61 0x62 0x63 0x64 0x65
0x66 0x67 0x68 0x69 0x6A 0x0D 0x0A
```

Response:

For 250 bytes sent, the response is

```
OK 250\r\n
0x4F 0x4B 0xFA 0x00  0x0D 0x0A
```

When TCP_ACK(bit[2]) is enabled the send response gives 3bytes,1byte for socket id and 2bytes represents the length .

```
at+rsi_snd=1,10,0,0,0123456789\r\n
```

Response :

```
OK  0x02\0x00
0x4F 0x4B 0x02 0x0A 0x00 0x0D 0x0A
```

When enable socket bit map of tcp socket the response of send command over ssl

```
at+rsi_snd=1,10,0,0,hellohello\r\n
```

Response:

```
   0x4F 0x4B 0x01 0x45 0x00 0x0D 0x0A
```

8.37.2 Send Data in Binary mode

Description:

This command sends data from the host to the module, to be transmitted over a wireless media.

This same command can be used to send SSL/web socket data. No additional parameters are required.

Note:

1) Maximum data which can be send over TCP/LTCP socket is 1460 Bytes
2) Maximum data which can be send over LUDP socket is 1472 Bytes
3) Maximum data which can be send over Web socket is 1452 Bytes
4) Maximum data which can be send over TCP-SSL/LTCP-SSL is 1390 Bytes
5) Maximum data which can be send over Web socket - SSL is 1382 Bytes

Command Format:

```
#define PROTOCOL_OFFSET  46  for TCP
#define WEBSOCKET_HEADER_SIZE 6 for payload size less than 126
#define WEBSOCKET_HEADER_SIZE 8 for payload size more than or
equal to 126
#define PROTOCOL_OFFSET  34 for UDP
#define RSI_MAX_PAYLOAD_SIZE 1460 for TCP
#define RSI_MAX_PAYLOAD_SIZE 1472 for UDP
#define RSI_MAX_PAYLOAD_SIZE 1452 for WS
#define RSI_MAX_PAYLOAD_SIZE 1390 for SSL
#define RSI_MAX_PAYLOAD_SIZE 1382 for WSS
#define RSI_TXDATA_OFFSET_TCP   46
#define RSI_TXDATA_OFFSET_LUDP  16
#define RSI_IP_ADD_LEN          4
```

```
typedef union {
      struct {
            uint8   ip_version[2];
            uint8   socketDescriptor[2];
            uint8   sendBufLen[4];
            uint8   sendDataOffsetSize[2];
            uint8   padding[RSI_MAX_PAYLOAD_SIZE];
      } sendFrameSnd;
      struct {
            uint8   ip_version[2];
            uint8   socketDescriptor[2];
            uint8   sendBufLen[4];
            uint8   sendDataOffsetSize[2];
            uint8   destPort[2];
            union{
                  uint8   ipv4_address[RSI_IP_ADD_LEN];
                  uint32  ipv6_address[RSI_IP_ADD_LEN];
            }destIPaddr;
      uint8   sendDataOffsetBuf[RSI_TXDATA_OFFSET_LUDP];
      uint8   sendDataBuf[RSI_MAX_PAYLOAD_SIZE;
      } sendFrameLudpSnd;
      uint8 uSendBuf[RSI_MAX_PAYLOAD_SIZE];
} rsi_uSend;
```

Command Parameters:

`ip_version`: IP version used, either 4 or 6 .

`socketDescriptor`: Lower byte is used as Socket number over which data is to be sent.

If websocket is enabled, higher byte is used to send the web socket information.

In web socket information seventh bit indicates the FIN packet and bit [3:0] indicates the opcode (type of the packet to be included in web socket header).

OPCODE should be as follows(Refer RFC 6455):

- 0 - Continuation frame
- 1 – Text frame
- 2 – Binary frame
- [3-7] - Reserved for further non-control frames
- 8 - Connection close frame
- 9 - Ping frame

- 10 - Pong frame
- [B-F] - Reserved for further control frames

FIN Bit should be as follows:

- 0: More web socket frames to be followed.
- 1: Final frame web socket message.

`sendBufLen`: Data payload length to be sent .

`sendDataOffsetSize`: Offset at which payload present.

`padding/sendDataBuf`: Actual data to be sent. A maximum of 1460bytes in case of TCP and 1472 bytes in case of UDP, can be sent in a packet.

`destPort`: Destination port number, of the socket in the remote terminal.

`ipv4_address`: IPv4 address of the remote terminal.Only first 4 bytes of ipv4 address gets filled, remaining 12 bytes are zero.

`ipv6_address`: IPv6 address of the remote terminal.

`sendDataOffsetBuf`: Dummy bytes.

When the TCP/IP stack is bypassed, the Host uses its own TCP/IP stack and sends Ethernet II frames to the module. The maximum size of the payload is 1514 bytes. The user needs to use rsi_send_raw_data.c API to send the data in TCP/IP bypass mode. Refer **8.1Set Operating Mode**to know how to enable TCP/IP bypass mode.

Response:

There is no response payload for this command.

Note :

In case of USB-CDC and UART Binary Modes(To overcome the flow control),for each and every data packet, ack is received as response. Before sending the next data packet, user must ensure of getting the data packet ack for the previous data packet.

Frame Type of data packet ack is 0xAC

If the error code of 0x3F is received in the data packet ack, this means buffer full condition has occurred and give some delay for the next data packet to send.

Note:

if you are using IPv6, the maximum size would be 1452 bytes, as IPv6's header size is 40 bytes vs. IPv4's 20 byte size (and either way, one must still allow 8 bytes for the UDP header).

Possible error codes:

Possible error codes are -2, 63.

Relevance:

This command is relevant when module is configured in operating mode 0,1,2 or 6.

8.38 Read Data

This section explains how to read socket data from the module to host.

Note:
To use this feature we need to set BIT(0) in the socket bitmap while opening a socket

8.38.1 Read Data in AT mode

Description:

This command is used to request number of bytes received on given socket. Module will give requested number of bytes received on particular socket (synchronous) only if this command is given from host.If requested numbers of bytes are greater than bytes available on given socket module will return only available number of bytes to host.If no data available on given socket module will wait till data received on given socket to serve this command.

Command Format:

To read data

```
at+rsi_read=< socketDescriptor >,< no of bytes >,<timeout in ms>\r\n
```

Command Parameters:

`socketDescriptor`– Socket handle of the socket over which data is to be received.

`no of bytes` – Length of the data that has to be read from the module.

`time out in ms` – Read timeout in milliseconds to configure read data time out.

Response:

Result Code	Description
AT+RSI_READ<`ip_version`> `< recvSocket>< recvBufLen>< ipv4_address` / `ipv6_address>< src_port>< recvDataBuf>`	`ip_version` (2 bytes, hex):IP version of the data received. 4 – for IPv4 6 – for IPv6 `recvSocket` (2 bytes, hex) – socket handle of the socket over which the data is received. The least significant byte is returned first. If Web socket has been enabled, upper byte holds the web socket info. Seventh bit

Result Code	Description
	indicates the FIN packet and bit 3:0 gives the opcode information from web socket header. `recvBufLen` (2 bytes, hex) – Number of bytes received. The least significant byte is sent first. For example, 900 bytes (0x0384) would be sent as <0x84><0x03> `ipv4_address` / `ipv6_address` (16 bytes, hex) – Source IPv4/IPv6 address. i.e IP address from which data is received. This field is not present in the message if the data is received over a TCP socket. Only first 4 bytes are filled rest 12 bytes are zero for IPv4 address. `src_port` (2 bytes, hex) – Source port. i.e port number from which data is received. This field is not present in the message if the data is received over a TCP socket. `recvDataBuf` – actual received data stream. A maximum of 1472 bytes can be received in case of UDP and 1460 bytes in case of TCP, in this field. When the module sends data to the Host, byte stuffing is NOT done by the module. The size parameter should be used to know how many bytes of valid data is expected.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 or 6.

Example:

Command:

at+rsi_read=1,4,100\r\n

Response:

For IPV4:

If 'abcd' is sent from remote terminal to module, on an UDP socket with handle 1, from source ipv4 address 192.168.1.1 to destination ipv4 address 192.168.1.2(module address) and source port 8001, the module sends the following response to the host.

AT+RSI_READ 4 14192 168 1 1 8001abcd \r\n

```
0x41 0x54  0x2B  0x52 0x53 0x49 0x5F 0x52 0x45 0x41 0x44 0x04
0x00 0x01  0x000x04 0x00 0xC0 0xA8 0x01 0x010x41 0x1F0x61 0x62
0x63  0x64 0x0D 0x0A
```

If 'abcd' is sent from remote terminal to module, on a TCP socket with handle 1, the module sends the following response to the host.

AT+RSI_READ 4 14abcd \r\n

```
0x41 0x54 0x2B 0x52 0x53 0x49 0x5F 0x52 0x45 0x41 0x44 0x04
0x00 0x01 0x000x04 0x00 0x61 0x62 0x63 0x64 0x0D 0x0A
```

For IPV6:

If 'abcd' is sent from remote terminal to module, on an UDP socket with handle 1, from source ipv6 address 2001:db8:1:0:0:0:0:153to sdestinationipv6 address 2001:db8:1:0:0:0:0:154 (module address)and source port 8001, the module sends the following response to the host.

AT+RSI_READ 6 142001:db8:1:0:0:0:0:153 8001abcd \r\n

```
0x41 0x54 0x2B 0x52 0x53 0x49 0x5F 0x52 0x45 0x41 0x44 0x06
0x00 0x01 0x000x04 0x00 0x32 0x30 0x30 0x31 0x3A 0x44 0x42
0x38 0x3A 0x31 0x3A 0x30 0x3A 0x30 0x3A 0x30 0x3A 0x30 0x3A
0x31 0x35 0x33 0x41 0x1F0x61 0x62 0x63 0x64 0x0D 0x0A
```

Note:

The data delivered to the Host on receiving data on a TCP socket does not include the source IP address and source port (Sip and Sport).

8.38.2 Read Data in Binary mode

Description:

This command is used to request number of bytes received on given socket. Module will give requested number of bytes received on particular socket (synchronous) only if this command is given from host.If requested numbers of bytes are greater than bytes available on given socket module will return only available number of bytes to host.If no data available on given socket module will wait till data received on given socket to serve this command.

Command Format:

```
#define RSI_MAX_PAYLOAD_SIZE 1460 for TCP
```

```
#define RSI_MAX_PAYLOAD_SIZE 1472 for UDP
#define RSI_MAX_PAYLOAD_SIZE 1452 for WS
#define RSI_MAX_PAYLOAD_SIZE 1390 for SSL
#define RSI_MAX_PAYLOAD_SIZE 1382 for WSS

typedef struct {
    uint8      socketDescriptor;
    uint8      data_length[4];
    uint8      timeout_in_ms[2];
} rsi_uRead;
```

Command Parameters:

`socketDescriptor`– Socket handle of the socket over which data is to be received.

`no of bytes` – Length of the data that has to be read from the module.

`timeout_in_ms` – Read timeout in milliseconds to configure read data time out.

Response:

```
#define RSI_RXDATA_OFFSET_TCP_V4                26
#define RSI_RXDATA_OFFSET_TCP_V6                46
#define RSI_RXDATA_OFFSET_UDP_V4                14
#define RSI_RXDATA_OFFSET_UDP_V6                34
#define RSI_MAX_PAYLOAD_SIZE 1460 for TCP
#define RSI_MAX_PAYLOAD_SIZE 1472 for UDP
```

For UDP :

```
typedef struct {
      uint8         ip_version[2];
      uint8         recvSocket[2];
      uint8         recvBufLen[4];
      uint8         recvDataOffsetSize[2];
      uint8         fromPortNum[2];
      union{
            uint8       ipv4_address[4];
            uint8            ipv6_address[16];
      }fromIPaddr;
      uint8       recvDataOffsetBuf[RSI_RXDATA_OFFSET_UDP_V4];
      uint8       recvDataBuf[RSI_MAX_PAYLOAD_SIZE];
```

```
} rsi_recvFrameUdp;
```

For UDP on IPv6 address:

```
typedef struct {
      uint8         ip_version[2];
      uint8     recvSocket[2];
      uint8     recvBufLen[4];
      uint8     recvDataOffsetSize[2];
      uint8     fromPortNum[2];
      union{
            uint8       ipv4_address[4];
            uint8          ipv6_address[16];
      }fromIPaddr;
      uint8 recvDataOffsetBuf[RSI_RXDATA_OFFSET_UDP_V6];
      uint8 recvDataBuf[RSI_MAX_PAYLOAD_SIZE];
} rsi_recvFrameUdp6;
```

For TCP:

```
typedef struct {
      uint8    ip_version[2];
      uint8    recvSocket[2];
      uint8    recvBufLen[4];
      uint8    recvDataOffsetSize[2];
      uint8    fromPortNum[2];
      union{
            uint8     ipv4_address[4];
            uint8     ipv6_address[16];
      }fromIPaddr;
      uint8     recvDataOffsetBuf[RSI_RXDATA_OFFSET_TCP_V4];
      uint8     recvDataBuf[RSI_MAX_PAYLOAD_SIZE];
} rsi_recvFrameTcp;
```

For TCP on ipv6 address:

```
typedef struct {
      uint8         ip_version[2];
      uint8         recvSocket[2];
```

```
        uint8          recvBufLen[4];
        uint8          recvDataOffsetSize[2];
        uint8          fromPortNum[2];
        union{
              uint8        ipv4_address[4];
              uint8        ipv6_address[16];
        }fromIPaddr;
        uint8        recvDataOffsetBuf[RSI_RXDATA_OFFSET_TCP_V6];
        uint8        recvDataBuf[RSI_MAX_PAYLOAD_SIZE];
} rsi_recvFrameTcp6;
```

Response Parameters:

`ip_version`: IP version used, either 4 or 6 .

`recvSocket`: Out of 2 bytes lower byte represents socket number on which data is received.

If web socket is enabled, higher byte represents the web socket information. In web socket information seventh bit indicates the FIN packet and bit [3:0] indicates the opcode from web socket header.

`recvBufLen`: The size of the data received

`recvDataOffsetSize`: This is the offset in the received payload, after which actual data begins.

`fromPortNum`: Port number of the remote peer

`fromIPaddr.ipv4_address`: The IPv4 address of the remote peer. Only first four bytes of ipv4 address are filled rest 12 bytes are zero.

`fromIPaddr.ipv6_address`: The IPv6 address of the remote peer.

`recvDataOffsetBuff`: Dummy bytes which can be ignored by host.

`recvDataBuf`: Actual data sent from remote peer.

Possible error codes:

Possible error codes are 0x0021, 0xff74, 0xff80,0xff6a

Relevance:

This command is relevant when module is configured in operating mode 0,1,2 or 6

Note:

1. Maximum data can be send over TCP/LTCP socket is 1460 Bytes
2. Maximum data can be send over LUDP socket is 1472 Bytes

8.39 Receive Data on Socket

This section explains how module sends received data to Host.

Note:
Module support maximum SSL record size is 8K. If it is more than 8K module will close the socket.

8.39.1 Receive Data on Socket in AT mode

Description:

The module delivers the data obtained on a socket to the Host with this message. This is an asynchronous response. It is sent from the module to the host when the module receives data from a remote terminal. SSL data is also received in a similar fashion.

Command Parameters:

N/A

Response:

Result Code	Description
AT+RSI_READ<ip_version> < recvSocket>< recvBufLen> ipv4_address / ipv6_address>< src_port> recvDataBuf>	ip_version (2 bytes, hex):IP version of the data received. 4 – for IPv4 6 – for IPv6 recvSocket (2 bytes, hex) – socket handle of the socket over which the data is received. The least significant byte is returned first. If Web socket has been enabled, upper byte holds the web socket info. Seventh bit indicates the FIN packet and bit 3:0 gives the opcode information from web socket header. recvBufLen (2 bytes, hex) – Number of bytes received. The least significant byte is sent first. For example, 900 bytes (0x0384) would be sent as <0x84><0x03> ipv4_address /

Result Code	Description
	`ipv6_address` (16 bytes, hex) – Source IPv4/IPv6 address. i.e IP address from which data is received. This field is not present in the message if the data is received over a TCP socket. Only first 4 bytes are filled rest 12 bytes are zero for IPv4 address. `src_port` (2 bytes, hex) – Source port. i.e port number from which data is received. This field is not present in the message if the data is received over a TCP socket. `recvDataBuf` – actual received data stream. A maximum of 1472 bytes can be received in case of UDP and 1460 bytes in case of TCP, in this field. When the module sends data to the Host, byte stuffing is NOT done by the module. The size parameter should be used to know how many bytes of valid data is expected.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 or 6.

Example:

Command:

N/A

Response:

For IPV4:

If 'abcd' is sent from remote terminal to module, on an UDP socket with handle 1, from source ipv4 address 192.168.1.1 to destination ipv4 address 192.168.1.2(module address) and source port 8001, the module sends the following response to the host.

AT+RSI_READ 4 14192 168 1 1 8001abcd \r\n

```
0x41 0x54  0x2B  0x52 0x53 0x49 0x5F 0x52 0x45 0x41 0x44 0x04
0x00 0x01  0x000x04 0x00 0xC0 0xA8 0x01 0x010x41 0x1F0x61 0x62
0x63  0x64 0x0D 0x0A
```

If 'abcd' is sent from remote terminal to module, on a TCP socket with handle 1, the module sends the following response to the host.

AT+RSI_READ 4 14abcd \r\n

```
0x41 0x54 0x2B 0x52 0x53 0x49 0x5F 0x52 0x45 0x41 0x44 0x04
0x00 0x01 0x000x04 0x00 0x61 0x62 0x63 0x64 0x0D 0x0A
```

For IPV6:

If 'abcd' is sent from remote terminal to module, on an UDP socket with handle 1, from source ipv6 address 2001:db8:1:0:0:0:0:153to sdestinationipv6 address 2001:db8:1:0:0:0:0:154 (module address)and source port 8001, the module sends the following response to the host.

AT+RSI_READ 6 142001:db8:1:0:0:0:0:153 8001abcd \r\n

```
0x41 0x54 0x2B 0x52 0x53 0x49 0x5F 0x52 0x45 0x41 0x44 0x06
0x00 0x01 0x000x04 0x00 0x32 0x30 0x30 0x31 0x3A 0x44 0x42
0x38 0x3A 0x31 0x3A 0x30 0x3A 0x30 0x3A 0x30 0x3A 0x30 0x3A
0x31 0x35 0x33 0x41 0x1F0x61 0x62 0x63 0x64 0x0D 0x0A
```

Note:

The data delivered to the Host on receiving data on a TCP socket does not include the source IP address and source port (Sip and Sport).

8.39.2 Receive Data on Socket in Binary mode

Description:

When the module receives data from a remote client, it triggers an asynchronous interrupt to indicate the Host that there is data to be read from the module..

Note:

1. Maximum data which can be received over TCP socket is 1460 Bytes
2. Maximum data which can be received over TCP socket is 1472 Bytes
3. Maximum data which can be received over WEB socket is 1452 Bytes
4. Maximum data which can be received over TCP-SSL socket is 1390 Bytes
5. Maximum data which can be received over Web Socket – SSL is 1382 Bytes

Command Format:

N/A

Command Parameters:

N/A

Response:

```
#define RSI_RXDATA_OFFSET_TCP_V4                26
#define RSI_RXDATA_OFFSET_TCP_V6                46
#define RSI_RXDATA_OFFSET_UDP_V4                14
#define RSI_RXDATA_OFFSET_UDP_V6                34
#define RSI_MAX_PAYLOAD_SIZE 1460 for TCP
#define RSI_MAX_PAYLOAD_SIZE 1472 for UDP
```

For UDP :

```
typedef struct {
      uint8         ip_version[2];
      uint8         recvSocket[2];
      uint8         recvBufLen[4];
      uint8         recvDataOffsetSize[2];
      uint8         fromPortNum[2];
      union{
            uint8       ipv4_address[4];
            uint8            ipv6_address[16];
      }fromIPaddr;
      uint8       recvDataOffsetBuf[RSI_RXDATA_OFFSET_UDP_V4];
      uint8       recvDataBuf[RSI_MAX_PAYLOAD_SIZE];
} rsi_recvFrameUdp;
```

For UDP on IPv6 address:

```
typedef struct {
      uint8         ip_version[2];
      uint8     recvSocket[2];
      uint8     recvBufLen[4];
      uint8     recvDataOffsetSize[2];
      uint8     fromPortNum[2];
      union{
            uint8       ipv4_address[4];
            uint8          ipv6_address[16];
```

```
      }fromIPaddr;
      uint8 recvDataOffsetBuf[RSI_RXDATA_OFFSET_UDP_V6];
      uint8 recvDataBuf[RSI_MAX_PAYLOAD_SIZE];
} rsi_recvFrameUdp6;
```

For TCP:

```
typedef struct {
      uint8    ip_version[2];
      uint8    recvSocket[2];
      uint8    recvBufLen[4];
      uint8    recvDataOffsetSize[2];
      uint8    fromPortNum[2];
      union{
            uint8     ipv4_address[4];
            uint8     ipv6_address[16];
      }fromIPaddr;
      uint8     recvDataOffsetBuf[RSI_RXDATA_OFFSET_TCP_V4];
      uint8     recvDataBuf[RSI_MAX_PAYLOAD_SIZE];
} rsi_recvFrameTcp;
```

For TCP on ipv6 address:

```
typedef struct {
      uint8         ip_version[2];
      uint8         recvSocket[2];
      uint8         recvBufLen[4];
      uint8         recvDataOffsetSize[2];
      uint8         fromPortNum[2];
      union{
            uint8       ipv4_address[4];
            uint8       ipv6_address[16];
      }fromIPaddr;
      uint8       recvDataOffsetBuf[RSI_RXDATA_OFFSET_TCP_V6];
      uint8       recvDataBuf[RSI_MAX_PAYLOAD_SIZE];
} rsi_recvFrameTcp6;
```

Response Parameters:

`ip_version`: IP version used, either 4 or 6 .

`recvSocket`: Out of 2 bytes lower byte represents socket number on which data is received.

If web socket is enabled, higher byte represents the web socket information. In web socket information seventh bit indicates the FIN packet and bit [3:0] indicates the opcode from web socket header.

`recvBufLen`: The size of the data received

`recvDataOffsetSize`: This is the offset in the received payload, after which actual data begins.

`fromPortNum`: Port number of the remote peer

`fromIPaddr.ipv4_address`: The IPv4 address of the remote peer. Only first four bytes of ipv4 address are filled rest 12 bytes are zero.

`fromIPaddr.ipv6_address`: The IPv6 address of the remote peer.

`recvDataOffsetBuff`: Dummy bytes which can be ignored by host.

`recvDataBuf`: Actual data sent from remote peer.

When the TCP/IP stack is bypassed, the module receives WLAN frames from the remote terminal and passes on to the Host through an interrupt.

Module forwards receive packet in ETHERNET II format.

Possible error codes:

No possible error code as it is asynchronous message from module to host.

Relevance:

This command is relevant when module is configured in operating mode 0,1,2 or 6

8.40 Wireless Firmware Upgrade

Description:

This command is sent as a response to the wireless firmware upgradation request.

When user clicks on Upgrade button on the wireless up gradation page, module sends an asynchronous message("AT+RSI_FWUPREQ" in AT mode and Frame type 0x59 in Binary mode) to host. Upon receiving this message host has to send wireless firmware upgrade request message if upgrade is required. Host can ignore if upgrade is not required. For more details on wireless firmware upgrade refer Wireless Firmware Upgradesection.

Command Format:

AT Mode:

```
at+rsi_fwupok\r\n
```

Binary Mode:

N/A

Command Parameters:

N/A

Response:

AT Mode:

There is no response for this command.

After successful upgradation,firmware gives a success indication with an asynchronous message as “AT+RSI_FWUPSUCCESS\r\n”. Also “Firmware up gradation successful” pop-up window appears on the browser.

On firmware upgrade failure or host not responding for firmware upgrade request , module gives an error message on pop-up window:“module not responding” on the browser.

Binary Mode:

After successful upgradation,firmware gives a success indication with an asynchronous frame type 0x5A. Also “Firmware up gradation successful” pop-up window appears on the browser.

On firmware upgrade failure or host not responding for firmware upgrade request , module gives an error message on pop-up window:“module not responding” on the browser.

Response Parameters:

N/A

Possible Error Codes:

Possible error codes are 0x0021,0x0025,0x002C, 0x0034.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 or 6.

Note:

Wireless firmware upgradation is supported only for the latest versions of firefox and google chrome.

8.41 Back ground scan(BG scan)

Description:

This command is to scan Access Points, when module is in connected state. The scan results are sorted in decreasing order of signal strength (RSSI value). The scanned access point with highest signal strength will be the first in the list.

Upon issuing this command, RS9113 WiSeConnect module validates the Channel bit map issued through the scan command, to ensure that background scan is performed only on those channels.

This is used in station/client mode for roaming. If user wants to scan AP's for both band's (2.4GHz/5GHz) than make sure you enable custom_feature_bit_map bit[27] in opermode command.

Command Format:

AT Mode:

```
at+rsi_bgscan= < bgscan_enable >,< enable_instant_bgscan >,
< bgscan_threshold >,< rssi_tolerance_threshold >,
< bgscan_periodicity >, < active_scan_duration >,
< passive_scan_duration >,< multi_probe >\r\n
```

Binary Mode:

```
struct {
    uint8       bgscan_enable[2];
    uint8       enable_instant_bgscan[2];
    uint8       bgscan_threshold[2];
    uint8       rssi_tolerance_threshold[2];
    uint8       bgscan_periodicity[2];
    uint8       active_scan_duration[2];
    uint8       passive_scan_duration[2];
    uint8       multi_probe;
 } bgscanFrameSnd;
```

Command Parameters:

`bgscan_enable`: To enable/Disable bgscan

0 – Disable

1 - Enable

`enable instant_bgscan`: If this is set to 1 then module will send probe request immediately on air and bgscan results will be given to host.

Note:

If host requires BGScan results then `instant_bg_enable` has to be set to 1.

`bgscan_threshold`: This is the threshold in dBm to trigger the bgscan. After `bgscan_periodicity`, If connected AP RSSI falls below this value then bgscan will be triggered.

`rssi_tolerance_threshold`: This is difference of last RSSI of connected AP and current RSSI of connected AP. Here last RSSI is the RSSI calculated at the last beacon received and current RSSI is the RSSI calculated at current beacon received. If this difference is more than rssi_tolerance_threshold then bgscan will be triggered irrespective of periodicity.

`bgscan_periodicity`: This is time period in seconds to trigger bgscan if RSSI of connected AP is above (assuming RSSI is positive value) the given `bgscan_threshold`.

`active_scan_duration`: This is active scan duration per channel in milli seconds.

`passive_scan_duration`: This is passive scan duration per DFS channel in 5GHz in milli seconds.

`multi_probe`: If set to one then module will send two probe request one with specific SSID provided during join command and other with NULL ssid (to scan all the access points).

Note:

1.Channel to scan in background scan is taken from Channel bit maps of scan command , e.g. `channel_bit_map_2_4 and channel_bit_map_5.`

`2.Active scan duration should be less than DTIM period to avoid multicast packet loss.`

`3.Number of channels supported in bgscan are 24 only.`

Response:

AT Mode:

If `instant_bg_enable` is disabled:

OK\r\n

If `instant_bg_enable` is enabled:

Result Code	Description
OK< `scanCount`>< `padding`> < rfChannel >< securityMode >< rssiVal ><	Successful execution of the command.

Result Code	Description
uNetworkType >< ssid >< bssid >< reserved > …….up to the number of scanned nodes	
ERROR<Error code>	Failure

Binary Mode:

If enable instant_bgscan is enabled:

```
struct{
      uint8  rfChannel;
      uint8  securityMode;
      uint8  rssiVal;
      uint8  uNetworkType;
      uint8  ssid[34];
      uint8  bssid[6];
      unit8  reserved2[2];
}rsi_scanInfo;

#define RSI_AP_SCANNED_MAX 11
typedef struct {
      uint8    scanCount[4];
      uint8  reserved1[4];
      rsi_scanInfo strScanInfo[RSI_AP_SCANNED_MAX] ;
} rsi_scanResponse;
```

Response Parameters:

`Scancount(4 byte)`: Number of Access Points scanned

`Reserved1(4 byte)`: reserved bytes.

`rfChannel(1 byte)`: Channel Number of the scanned Access Point

`securityMode(1 byte)`:

- 0–Open
- 1–WPA
- 2–WPA2
- 3-WEP
- 4–WPA Enterprise
- 5–WPA2 Enterprise

`rssival(1 byte)`: RSSI of the scanned Access Point

`uNetworkType(1 byte)`: Network type of the scanned Access Point

1– Infrastructure mode

`Ssid(34 bytes)`: SSID of the scanned Access Point

`Bssid(6 bytes)`: MAC address of the scanned Access Point

`Reserved2(2 byte)` **:** Reserved bytes.

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C, 0x004A

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 2.

Note:

If host does not provide channel bit map and scan channel number in scan command, module will scan all channels in 2.4 GHz and all non-DFS channels in 5GHz.

Example:

AT Mode:

When instant bgscan is enabled, host will get OK response followed by the response of BG scan. Module will do back ground scanning in the configured channel given in the channel bit map or scan all channels if bitmap is not provided (all non DFS channels in 5GHz)and send the scanned result to host.

```
at+rsi_bgscan=1,1,10,4,10,15,20,1\r\n
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x62  0x67  0x73
0x63  0x61  0x6E  0x3D  0x31  0x2C  0x31  0x2C  0x31  0x30
0x2C  0x34  0x2C  0x31  0x30  0x2C  0x31  0x35  0x2C  0x32
0x30  0x2C  0x31  0x0D  0x0A
```

Response:

If two networks are found with the SSID “Redpine_net1” and “Redpine_net2”, in channels 6 and 11, with measured RSSI of -20dBm and -14dBm respectively, the return value is

```
O K <scanCount =2>< padding>< rfChannel =0x0A><securityMode
=0x02>< rssiVal =14>< uNetworkType =0x01>< ssid=Redpine_net2><
bssid =0x00 0x23 0xA7 0x1F 0x1F 0x15><reserved >< rfChannel
```

```
=0x06>< securityMode =0x00>< rssiVal =20>< uNetworkType
=0x01>< ssid =Redpine_net1>< bssid =0x00 0x23 0xA7 0x1F 0x1F
0x14>< reserved >  \r\n
```

```
0x4F    0x4B    0x02    0x00    0x00    0x00    0x00    0x00
0x00    0x00    0x0A0x020x0D0x010x52    0x65    0x64    0x70
0x69    0x6E    0x65    0x5F    0x6E    0x74    0x32    0x00
0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x00    0x00    0x00    0x000x00    0x23    0xA7    0x1F    0x1F
0x150x00    0x00    0x060x000x140x010x52    0x65    0x64    0x70
0x69    0x6E    0x65    0x5F    0x6E    0x74    0x31    0x00
0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x00    0x00    0x00    0x000x00    0x23    0xA7    0x1F    0x1F
0x140x00    0x00    0x0D    0x0A
```

When instant bgscan is disabled and When connected AP RSSI falls below -10dBm (e.g. -15, -12 etc) then bgscan will be triggered after 10 seconds period get over. But host will get only OK message here.

```
at+rsi_bgscan=1,0,10,4,10,15,20,1\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x62  0x67  0x73
0x63  0x61  0x6E  0x3D  0x31  0x2C  0x31  0x2C  0x31  0x30
0x2C  0x34  0x2C  0x31  0x30  0x2C  0x31  0x35  0x2C  0x32
0x30  0x2C  0x31  0x0D  0x0A
```

```
Response:
```

```
OK\r\n
```

```
0x4F  0x4B  0x0D  0x0A
```

8.42 Roam Parameters

Description:

This command is used to enable roaming and set roaming parameters. This command can be issued any time after init command but this command will come into action only after bgscan.

Command Format:

AT Mode:

```
at+rsi_roam_params= < roam_enable>,<
roam_threshold>,<roam_hysteresis>\r\n
```

Binary Mode:

```
struct {
      uint8 roam_enable[4];
      uint8 roam_threshold[4];
      uint8 roam_hysteresis[4];
}roamParamsFrameSnd;
```

Command Parameters:

`roam_enable`: To Enable/Disable roaming.

0 – Disable

1 - Enable

`roam_threshold`: If connected AP RSSI falls below this then module will search for new AP from background scanned list.

`roam_hysteresis`: If module found new AP with same configuration (SSID, Security etc) and if (connected_AP_RSSI – Selected_AP_RSSI) is greater than `roam_hysteresis` then it will try to roam to the new selected AP.

Response:

AT Mode:

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

Binary Mode:

No response payload for this command.

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C, 0x0026.

Relevance:

This command is relevant when the module is configured in Operating Mode 0,2.

Example:

AT Mode:

```
at+rsi_roam_params= 1,5,2\r\n
0x61  0x74  0x2B  0x72   0x73  0x69  0x5F  0x72  0x6F  0x61
0x6D  0x5F  0x70  0x61   0x72  0x61  0x6D  0x73  0x3D  0x31
0x2C  0x35  0x2C  0x32   0x0D  0x0A
```

Response:

```
OK\r\n
0x4F  0x4B  0x0D  0x0A
```

8.43 HT Caps

Description:

This command is used to enable HT(high throughput) Caps (capabilities) in module when operating in AP mode. This command has to be issued after ap configuration parameters command.

Command Format:

AT Mode:

```
at+rsi_ht_caps=< mode_11n_enable > ,< ht_caps_bitmap>\r\n
```

Binary Mode:

```
struct {
      uint8   mode_11n_enable[2];
      uint8   ht_caps_bitmap[2];
}htCapsFrameSnd;
```

Command Parameters:

`mode_11n_enable`: Enable/Disable 11n capabilities in AP Mode.

1 - Enable

0 - Disable

`ht_caps_bit_map`: Bit map corresponding to high throughput capabilities.

`ht_caps_bit_map[10:15]`: All set to '0'

`ht_caps_bit_map[8:9]`: Rx STBC support

00- Rx STBC support disabled

01- Rx STBC support enabled

`ht_caps_bit_map[6:7]`: Set to '0'

`ht_caps_bit_map[5]`: short GI for 20Mhz support

0- short GI for 20Mhz support disabled

1- short GI for 20Mhz support enabled

`ht_caps_bit_map[4]`: Green field support

0 -Green field support disabled

1 -Green field support enabled

ht_caps_bit_map[0:3]:Set to '0'.

Response:

AT Mode:

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

Binary Mode:

No reponse payload for this command.

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C,0x004D.

Relevance:

This command is relevant when the module is configured in Operating Mode 6.

Example:

AT Mode:

```
at+rsi_ht_caps=1,2\r\n
```

```
0x61  0x74  0x2B  0x72  0x73  0x69  0x5F  0x68  0x74  0x5F
0x63  0x61  0x70  0x73  0x3D  0x31  0x2C  0x32  0x0D 0x0A
```

Response:

```
OK\r\n
0x4F   0x4B   0x0D   0x0A
```

8.44 DNS Server

Description:

This command is used to provide the DNS server's IP address to the module. This command should be issued before the "DNS Resolution" command and after the "Set IP Parameter" command.

Command Format:

AT Mode:

For IPv4:

```
at+rsi_dnsserver=< DNSMode>,< primary_dns_ip.ipv4_address >,
< secondary_dns_ip.ipv4_address >\r\n
```

For IPv6:

```
at+rsi_dnsserver6=< DNSMode>,< primary_dns_ip.ipv6_address >,
< secondary_dns_ip.ipv6_address >\r\n
```

Binary Mode:

```
typedef struct {
      uint8 ip_version[2];
      uint8 DNSMode;
      union{
            uint8 ipv4_address[4];
            uint32 ipv6_address[4];
      }primary_dns_ip;
      union{
            uint8 ipv4_address[4];
            uint32 ipv6_address[4];
      }secondary_dns_ip;
}dnsServerFrameSnd;
```

Command Parameters:

ip_version: IPv4/IPv6

4 – IPv4

6 – IPv6

This parameter is available only in Binary mode.

DNSMode:

1 - The module can obtain DNS Server IP address during the command “Set IP Params” if the DHCP server in the Access Point supports it. In such a case, value of ‘1’ should be used if the module wants to read the DNS Server IP obtained by the module

0 - Value of ‘0’ should be used if the user wants to specify a primary and secondary DNS server address.

primary_dns_ip.ipv4_address: This is the IPv4 address of the Primary DNS server to which the DNS Resolution query will be sent. Should be set to ‘0’ if DNSMode =1. Only first 4 bytes of ipv4 address are filled, rest 12 bytes are zero.

primary_dns_ip.ipv6_address: This is the IPv6 address of the Primary DNS server to which the DNS Resolution query will be sent. Should be set to ‘0’ if DNSMode =1.

secondary_dns_ip.ipv4_address: This is the IPv4 address of the Secondary DNS server to which the DNS Resolution query will be sent. If DNSMode =1 or if the user does not want to specify a secondary DNS Server IP address, this parameter should be set to ‘0’. Only first 4 bytes of ipv4 address are filled, rest 12 bytes are zero.

secondary_dns_ip .ipv6_address: This is the IPv6 address of the Secondary DNS server to which the DNS Resolution query will be sent. If DNSMode =1 or if the user does not want to specify a secondary DNS Server IP address, this parameter should be set to ‘0’

Response:

AT Mode:

For IPv4:

Result Code	Description
OK<primary_dns_ip.ipv4_address><secondary_dns_ip.ipv4_address>	Successful execution of command.
ERROR<Error code>	Failure.

For IPv6:

Result Code	Description
OK< primary_dns_ip.ipv6_address >< secondary_dns_ip.ipv6_address>	Successful execution of command.
ERROR<Error code>	Failure.

Binary Mode:

```
typedef struct{
      union{
            uint8  ipv4_address[4];
            uint8  ipv6_address[16];
      }primary_dns_ip;
      union{
            uint8  ipv4_address[4];
            uint8  ipv6_address[16];
      }secondary_dns_ip;
}rsi_dnsserverResponse;
```

Response Parameters:

`primary_dns_ip.ipv4_address(16 bytes)`: IPv4 address of the primary DNS server. Only first 4 bytes of IPv4 address is filled, rest 12 bytes are zero.

`primary_dns_ip.ipv6_address(16 bytes)`: IPv6 address of the primary DNS server

`secondary_dns_ip.ipv4_address (16 bytes)`: IP address of the secondary DNS server. Only first 4 bytes of IPv4 address is filled, rest 12 bytes are zero.

`secondary_dns_ip.ipv6_address (16 bytes)`: IPv6 address of the secondary DNS server.

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002C, 0xFFF8,0xFF74,0xBBA8,0xBBB2,0xBBAF,0xBB17,0xBBB3

Relevance:

This command is relevant in Operating Modes 0, 1, 2, 6.

Example :

AT Mode:

For IPv4:

```
at+rsi_dnsserver=1,0,0\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x64 0x6E 0x73 0x73 0x65
0x72 0x76 0x65 0x72 0x3D 0x31 0x2C 0x30 0x2C 0x30 0x0D 0x0A
```

Response:

```
OK<primary=1.2.3.4><secondary=5.6.7.8>\r\n
```

```
0x4F 0x4B 0x01 0x02 0x03 0x04 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x05 0x06 0x07 0x08 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0D 0x0A
```

Command:

```
at+rsi_dnsserver=0,8.8.8.8,0\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x64 0x6E 0x73 0x73 0x65
0x72 0x76 0x65 0x72 0x3D 0x30 0x2C 0x38 0x2E 0x38 0x2E 0x38
0x2E 0x38 0x2C 0x30 0x0D 0x0A
```

Response:

```
OK< primary=8.8.8.8><secondary =0>\r\n
```

```
0x4F 0x4B 0x08 0x08 0x08 0x080x00 0x00 0x00 0x00 0x0D 0x0A
```

For IPv6:

```
at+rsi_dnsserver6=1,0,0\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x64 0x6E 0x73 0x73 0x65
0x72 0x76 0x65 0x72 0x36 0x3D 0x31 0x2C 0x30 0x2C 0x30 0x0D
0x0A
```

Response:

```
OK< primary v6 address=2001:db8:1:0:0:0:0:2>< secondary v6
address = 2001:db8:1:0:0:0:0:3>\r\n
```

```
0x4F 0x4B 0xB8 0x0D 0x01 0x20 0x00 0x00 0x01 0x00 0x00 0x00
0x00 0x00 0x02 0x00 0x00 0x00 0xB8 0x0D 0x01 0x20 0x00 0x00
0x01 0x00 0x00 0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x0D 0x0A
```

Command:

```
at+rsi_dnsserver6=0,2001:DB8:1:0:0:0:0:5,0\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x64 0x6E 0x73 0x73 0x65
0x72 0x76 0x65 0x72 0x36 0x3D 0x30 0x2C 0x32 0x30 0x30 0x31
0x3A 0x44 0x42 0x38 0x3A 0x31 0x3A 0x30 0x3A 0x30 0x3A 0x30
0x3A 0x30 0x3A 0x35 0x2C 0x30 0x0D 0x0A
```

Response:

```
OK< primary v6 address = 2001:DB8:1:0:0:0:0:5 ><secondary v6
address=0>\r\n
```

```
0x4F 0x4B 0xB8 0x0D 0x01 0x20 0x00 0x00 0x01 0x00 0x00 0x00
0x00 0x00 0x05 0x00 0x00 0x000x00 0x00 0x00 0x000x00 0x00 0x00
0x000x00 0x00 0x00 0x000x00 0x00 0x00 0x00 0x0D 0x0A
```

8.45 DNS Resolution

Description:

This command is to obtain the IP address of the specified domain name.

Command Format:

AT Mode:

For IPv4:

```
at+rsi_dnsget=< aDomainName >,< uDNSServerNumber >\r\n
```

For IPv6:

```
at+rsi_dnsget6=< aDomainName >,< uDNSServerNumber >\r\n
```

Binary Mode:

```
#define MAX_NAME_LEN 90
struct {
      uint8 ip_version[2];
      uint8 aDomainName [MAX_URL_LEN];
      uint8 uDNSServerNumber[2];
}dnsQryFrameSnd ;
```

Command Parameters:

`ip_version`: IP version used, either 4 or 6. This parameter is available in Binary mode.

`aDomainName`: This is the domain name of the target website. A maximum of 90 characters is allowed.

`uDNSServerNumber`: Reserved.

Response:

AT Mode:

For IPv4:

Result Code	Description
OK< ip_version>< uIPCount><aIPaddr.ipv4_address>..repeats for 10 times	Successful execution of command
ERROR<Error code>	Failure.

For IPv6:

Result Code	Description
OK< ip_version>< uIPCount><aIPaddr.ipv6_address>..repeats for 10 times	Successful execution of command
ERROR<Error code>	Failure.

Binary Mode:

```
#define MAX_DNS_REPLY 10
typedef struct TCP_EVT_DNS_Query_Resp {
      uint8  ip_version[2];
      uint8 uIPCount[2];
      union{
            uint8 ipv4_address[4];
            uint8 ipv6_address[16];
      }aIPaddr[MAX_DNS_REPLY];
}TCP_EVT_DNS_Query_Resp;
```

Response Parameters:

`ip_version(2 bytes)`: IP version used , either 4 or 6.This parameter is available only in Binary mode.

`uIPCount(2 bytes)`: Number of IP addresses resolved
`aIPaddr.ipv4_address(16 bytes)`: Individual IPv4 addresses, up to a maximum of 10 . Only first 4 bytes are filled in ipv4 address, rest 12 bytes are zero.

`aIPaddr.ipv6_address(16 bytes)`: Individual IPv6 addresses, up to a maximum of 10.

Note: Maximum timeout for DNS resolution is 120 seconds.

Possible error codes:

Possible error codes are 0x0015,0x0021, 0x0025, 0x002C,0xFFBB,0xBBA1,0xBBAA,0xBBA3,0xBBA4,0xBBAC

Relevance:

This command is relevant in Operating Modes 0, 1, 2 and 6.

Example:

AT Mode:

For IPv4:

```
at+rsi_dnsget=www.redpine.com,1\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x64 0x6E 0x73 0x67 0x 65
0x74 0x3D 0x77 0x77 0x77 0x2E 0x72 0x65 0x64 0x70 0x69 0x6E
0x65 0x73 0x69 0x67 0x6E 0x61 0x6C 0x73 0x2E 0x63 0x6F 0x6D
0x2C 0x31 0x0D 0x0A
```

Response:

```
OK<num_IPAddr=1><IPAddr1=201.168.1.100>\r\n
```

```
0x4F 0x4B 0x01 0x00 0xC9 0xA8 0x01 0x64 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0D 0x0A
```

For IPv6:

```
at+rsi_dnsget6=www.redpine.com,1\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x64 0x6E 0x73 0x67 0x 65
0x74 0x36 0x3D 0x77 0x77 0x77 0x2E 0x72 0x65 0x64 0x70 0x69
0x6E 0x65 0x73 0x69 0x67 0x6E 0x61 0x6C 0x73 0x2E 0x63 0x6F
0x6D 0x2C 0x31 0x0D 0x0A
```

Response:

```
OK<num_IPAddr=1><IPv6Addr1=2001:DB8:1:0:0:0:0:6>\r\n
```

```
0x4F 0x4B 0x01 0x000xB8 0x0D 0x01 0x20 0x00 0x00 0x01 0x00
0x00 0x00 0x00 0x00 0x06 0x00 0x00 0x00 0x0D 0x0A
```

8.46 DNS UPDATE

Description:

This command is to update client host name (type A record) in DNS server.

Command Format:

AT Mode:

```
at+rsi_dnsupdate=< ipversion >,< aZoneName >,< aHostName >,<
uDNSServerNumber >,< ttl >\r\n
```

Command Parameters:

`ip_version`: IP version used 4 (No support for ip version 6). This parameter is available in Binary mode.

`aZoneName`: This is the zone name of the Domain. A maximum of 31 characters is allowed.

`aHostName`: This is the host name of the Domain. A maximum of 31 characters is allowed.

`uDNSServerNumber`: DNS Server number.

`ttl`: Time To Live, Time in seconds for which hostname should be active.

Response:

AT Mode:

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

Binary Mode:

```
#define MAX_ZONE_LEN 31
```

```
#define MAX_HOST_NAME_LEN 31
typedef union {
    struct {
        uint8    ip_version;
        uint8    aZoneName[MAX_ZONE_LEN];
        uint8    aHostName[MAX_HOST_NAME_LEN];
        uint8    uDNSServerNumber[2];
            uint8    ttl[2];
    }dnsUpdateFrameSnd;
    uint8    uDnsUpdateBuf[MAX_ZONE_LEN + MAX_HOST_NAME_LEN +
5];
          }rsi_uDnsUpdate;
```

Command Parameters:

`ip_version`: IP version used 4 (No support for ip version 6). This parameter is available in Binary mode.

`aZoneName`: This is the zone name of the domain. A maximum of 31 characters is allowed.

`aHostName`: This is the host name of the domain. A maximum of 31 characters is allowed.

`uDNSServerNumber(2 bytes)`: DNS Server number.

`ttl(2 bytes)`: Time To Live, Time in seconds for which service should be active.

Possible error codes:

Possible error codes are 0x0015,0x0021, 0x0025, 0x002C,0xFFBB,0xBBA1,0xBBAA,0xBBA3,0xBBA4,0xBBAC

Relevance:

This command is relevant in Operating Modes 0, 1, 2 and 6.

Example:

AT Mode:

at+rsi_dnsupdate=4,RPS,REDPINE,1,53 \r\n

Response:
OK\r\n
0x4F 0x4B 0x0D 0x0A

8.47 HTTP GET

Description:

This command is used to transmit HTTP GET request from the module to a remote HTTP server. A subsequent HTTP GET request can be issued only after receiving the response of the previously issued HTTP GET request. Module acts as a HTTP client when this command is issued.

Command Format:

AT Mode:

For IPv4:

```
at+rsi_httpget=< https_enable >,< http_port >,< User_name >,<
Password >,< Host_name >,< Ip_address >,< url >,
< Extended_header >\r\n
```

For IPv6:

```
at+rsi_httpget6=< https_enable >,< http_port >,< User_name >,<
Password >,< Host_name >,< Ip_address >,< url >,
< Extended_header >\r\n
```

Binary Mode:

```
#define     HTTP_BUFFER_LEN   1200
typedef struct
{
      uint8 ip_version[2];
      uint8 https_enable[2];
      uint16 http_port;
      uint8  buffer[HTTP_BUFFER_LEN];
}HttpReqFrameSnd;
```

Command Parameters:

`ip_version`: ip version (4 or 6). This field is available only in Binary mode.

`https_enable`:

Set BIT(0) to enable HTTPS.

Set BIT(1) to enable NULL delimiter for HTTP buffer instead of comma

Set BIT(2) to use SSL TLS 1.0 version if HTTPS is enabled.

Set BIT(3) to use SSL TLS 1.2 version if HTTPS is enabled.

Set BIT(4) to use SSL TLS 1.1 version if HTTPS is enabled.

Note : If SSL is enabled by default it will use SSL TLS 1.0 and TLS 1.2 version. BIT(2) and BIT(3) are valid only when HTTPS is enabled.

Note: If SSL is enabled, module will use same SSL version for all SSL sockets until module reboots.

Set BIT(6) to enable HTTP version 1.1.

`http_port` – HTTP server port number. If this is not mentioned default port numbers will be used.

`Buffer` – This field available in Binary mode which contains following values in the order of < `User_name` >,< `Password` >,< `Host_name` >,< `Ip_address`>,< `url`>, < `Extended_header`>

If BIT(1) is not set in `https_enable` field

< `User_name` >,< `Password` >,< `Host_name` >,< `Ip_address`>,< `url`>, < `Extended_header`> fields should be delimited with comma(,)

If BIT(1) is set in `https_enable` field

< `User_name` >'\0'< `Password` >'\0'< `Host_name` >'\0'< `Ip_address`>'\0'<URL>'\0' < `Extended_header`> fields should be delimited with NULL('\0')

`User_name` – User_name for HTTP/HTTPS server authentication.

`Password` – Password for HTTP/HTTPS server authentication.

`Host_name` – Host name of the HTTP/HTTPS server.

`Ip_address` – IPv4/IPv6 address of HTTP/HTTPS server.

`url` – requested URL.

`Extended_header` - To append user configurable header fields to the default HTTP/HTTPS header

Note:

1) Maximum supported length for `User_name`, `Password` together is 278 bytes..
2) Maximum supported length for `Buffer` is (872-(length of `User_name`+length of `Password`)) bytes excluding delimiters.
3) If username, password, hostname and extended http headers are not required, user should send empty string separated by delimiter.
4) If content of any field contains comma(,) then NULL delimiter should be used.
5) Host needs to do byte stuffing in extended header field. Please refer Table 28: Data Stuffing

6) Maximum HTTP_GET data supported is (2^32 -1).

7) HTTP/S connection timeout is 20 seconds.

For example,

Https_enable = BIT(0)

http_port = 443

Username : username

Password : password

Hostname: www.google.com

IP = 192.168.40.50

URL=/index.html

Extended HTTP Header = ContentType: */*\r\n

Response:

Module may give http response in multiple chunks for a single HTTP GET request.

AT Mode:

Result Code	Description
AT+RSI_HTTPRSP=< more >< offset >< data_len >< data >	After the module sends out the HTTP GET request to the remote server, it may take some time for the response to come back. The response from the remote server is sent out to the Host from the module in the following form: AT+RSI_HTTPRSP=<More><Data Offset><Data Length><Data> The string AT+RSI_HTTPRSP is in uppercase ASCII.
ERROR<Error_code>	Failure

Binary Mode:

```
typedef struct TCP_EVT_HTTP_Data_t
{
      uint32 more;
      uint32 offset;
```

```
        uint32  data_len;
        uint8 data[1024];
}rsi_uHttpRsp;
```

Response Parameters:

`More(4 bytes)`: This indicates whether more HTTP data for the HTTP GET request is pending or not.

0–More data is pending. Further interrupts may be raised by the module till all the data is transferred to the Host.

1– End of HTTP data.

Offset(4 bytes): Always contains '0'.

`data_len`(4 bytes): data length in current chunk.

`Data`(Maximum 1024 bytes) : Actual http data.

Possible error codes:

Possible error codes for this command are 0x0015,0x0021, 0x0025, 0x002C, 0xFF74, 0xBBF0, 0xBBE1

Relevance:

This command is relevant when the module is configured in Operating Mode 0,1, 2 and 6.

Example:

AT Mode:

For IPv4:

Https_enable : 0

http_port : 80

Username : username

Password : password

Host_name : www.google.com

IP Address: 192.168.40.86

URL: /index.html

Extended HTTP Header: ContentType: */*\r\n

the below command is used

```
at+rsi_httpget=0,80,username,password,hostname,192.168.40.86,/
index.html,ContentType: */*\r\n\r\n
```

For IPv6:

Https_enable : = 0, to disable https

Http_port :8080

Username : username

Password : password

Hostname :www.google.com

IP Address: 2001:DB8:1:0:0:0:0:4

URL: /index.html

Extended HTTP Hedaer: ContentType: */*\r\n

the below command is used

```
at+rsi_httpget6=0,8080,username,password,hostname,2001:DB8:1:0
:0:0:0:4,/index.html,ContentType: */*\r\n\r\n
```

8.48 HTTP POST

Description:

This command is used to transmit HTTP POST request to a remote HTTP server. A subsequent HTTP POST request can be issued only the response to a previously issued HTTP POST request is received. Module acts as a HTTP client when this command is issued.

Command Format:

AT Mode:

For IPv4:

```
at+rsi_httppost=< https_enable >,< http_port >,< User_name >,<
Password >,< Host_name >,< Ip_address >,< url >,
< Extended_header >,< Data/Data_length >\r\n
```

For IPv6:

```
at+rsi_httppost6=< https_enable >,< http_port >,< User_name
>,< Password >,< Host_name >,< Ip_address >,< url >,
< Extended_header >,< Data/Data_length >\r\n
```

Binary Mode:

```
#define     HTTP_BUFFER_LEN   1200
struct
{
      uint8 ip_version[2];
```

```
        uint8 https_enable[2];
        uint16 http_port;
        uint8  buffer[HTTP_BUFFER_LEN ];
}HttpReqFrameSnd;
```

Command Parameters:

`ip_version`: ip version (4 or 6). This field is available only in Binary mode.

`https_enable`:

Set BIT(0) to enable HTTPS.

Set BIT(1) to enable NULL delimiter for HTTP buffer instead of comma

Set BIT(2) to use SSL TLS 1.0 version.

Set BIT(3) to use SSL TLS 1.2 version.

Set BIT(4) to use SSL TLS 1.1 version.

Note : If SSL is enabled by default it will use SSL TLS 1.0 and TLS 1.2 version.

BIT(2) and BIT(3) are valid only when HTTPS is enabled.

Note: If SSL is enabled, module will use same SSL version for all SSL sockets until module reboots.

Set BIT(5) to enable HTTP post data feature.

Set BIT(6) to enable HTTP version 1.1.

`http_port`: HTTP server port number. If this is not mentioned default port numbers will be used.

`buffer`: This field available in Binary mode which contains following values in the order of < `User_name` >,< `Password` >,< `Host_name` >,< `Ip_address`>,< `url`>, < `Extended_header`>,< `Data/Data_length`>. Data is the actual data to be posted to the server or Data_length is the length of the total http post data to be posted to the server. The parameter Buffer is a character buffer.

If BIT(1) is not set in `https_enable` field

< `User_name` >,< `Password` >,< `Host_name` >,< `Ip_address`>,< `url`>, < `Extended_header`>, < `Data`> fields should be delimited with comma(,)

If BIT(1) is set in `https_enable` field

< `User_name` >'\0'< `Password` >'\0'< `Host_name` >'\0'< `Ip_address`>'\0'<URL>'\0' < `Extended_header`>'\0' < `Data`> fields should be delimited with NULL('\0')

`User_name` – User_name for HTTP/HTTPS server authentication.

`Password` – Password for HTTP/HTTPS server authentication.

`Host_name` – Host name of the HTTP/HTTPS server.

`Ip_address` – IPv4/IPv6 address of HTTP/HTTPS server.

`url` – requested URL.

`Extended_header` - To append user configurable header fields to the default HTTP/HTTPS header

`Data` – Post data to be send.

`Data/Data_length` – Post data to be send/ Total length of the http data.

Note:

1) When BIT(6) is enabled in https_enable feature bitmap, hostname is mandatory (To support HTTP version 1.1)
2) When BIT(5) is enabled in https_enable feature bitmap, instead of Data, host need to give total HTTP data length.
3) Maximum supported length for `User_name`, `Password` together is 278 bytes.
4) Maximum supported length for `Buffer` is (872-(length of `User_name` + length of `Password`)) bytes excluding delimiters.
5) If username, password, hostname and extended http headers are not required, user should send empty string separated by delimiter.
6) If content of any field contains comma(,) then NULL delimiter should be used.
7) Host needs to do byte stuffing in extended header and data field. Please refer Table 28: Data Stuffing
8) Set BIT(6) in enabled in https_enable feature bitmap, to enable HTTP version 1.1 support.
9) Set BIT(5) is enabled in https_enable feature bitmap, to enable HTTP post data feature.

Example 1:

Https_enable = 0

Http_port = 80

Username = username

Password = password

Hostname = www.googel.com

IP = 192.168.40.50

URL=/index.html

Extended HTTP Header = ContentType: */*\r\n

Data=<data>

Example 2 (Set BIT(5) in https_enable field):

Https_enable = 32

Http_port = 80

Username = username

Password = password

Hostname = posttestserver.com

IP = 64.90.48.15

URL=/post.php

Extended HTTP Header = ContentType: */*\r\n

Data_length= 1800

Response:

Module may give http response in multiple chunks for a single HTTP POST request.

AT Mode:

Result Code	Description
AT+RSI_HTTPRSP=< more >< offset >< data_len >< data >	After the module sends out the HTTP POST request to the remote server, it may take some time for the response to come back. The response from the remote server is sent out to the Host from the module in the following form: AT+RSI_HTTPRSP=<More><Data Offset><Data Length><Data> The string AT+RSI_HTTPRSP is in uppercase ASCII.
ERROR<Error_code>	Failure

Binary Mode:

```
typedef struct TCP_EVT_HTTP_Data_t
{
      uint32 more;
      uint32 offset;
      uint32 data_len;
      uint8  data[1024];
}rsi_uHttpRsp;
```

Response Parameters:

More(4 bytes): This indicates whether more HTTP data for the HTTP GET request is pending or not.

0–More data is pending. Further interrupts may be raised by the module till all the data is transferred to the Host.

1– End of HTTP data.

2– HTTP post request success response.

Offset(4 bytes): Always contains '0'.

data_len(4 bytes): data length in current chunk.

Data(Maximum 1024 bytes): Actual http data.

Possible error codes:

Possible error codes for this command are 0x0015,0x0021, 0x0025, 0x002C, 0xFF74, 0xBBF0

Relevance:

This command is relevant when the module is configured in Operating Mode 0,1,2 and 6 .

Example 1:

AT Mode:

For IPv4:

at+rsi_httppost=1,443,username,password,hostname,192.168.40.86,/index.html,
ContentType: */*\r\n,<data=abcd>\r\n

For IPv6:

at+rsi_httppost6=0,443,username,password,hostname,2001:DB8:1:0:0:0:0:4,/index.html,
ContentType: */*\r\n,<data=abcd>\r\n

Example 2:

AT Mode:

For IPv4:

at+rsi_httppost=32,80,username,password,posttestserver.com,64.90.48.15,/index.html,
ContentType: */*\r\n, 100\r\n

at+rsi_httppost=65,443,username,password,posttestserver.com,64.90.48.15,/index.html,
ContentType: */*\r\n, 100\r\n

at+rsi_httppost=97,443,username,password,posttestserver.com,64.90.48.15,/index.html,
ContentType: */*\r\n, 100\r\n

For IPv6:

at+rsi_httppost6=32,80,username,password,hostname,2001:DB8:1:0:0:0:0:4,/index.html, ContentType: */*\r\n, 100\r\n

at+rsi_httppost6=65,443,username,password,hostname,2001:DB8:1:0:0:0:0:4,/index.html, ContentType: */*\r\n, 100\r\n

at+rsi_httppost6=97,443,username,password,hostname,2001:DB8:1:0:0:0:0:4,/index.html, ContentType: */*\r\n, 100\r\n

8.49 HTTP POST DATA

Description:

This command is used to transmit HTTP POST request to a remote HTTP server. A subsequent HTTP POST request can be issued only the response to a previously issued HTTP POST request is received. Module acts as a HTTP client when this command is issued.

Command Format:

AT Mode:

at+rsi_httppost_data=< current_chunk_length >,< Data>\r\n

`current_chunk_length: Length of the current data.`

`Data: HTTP data`

Note:
httppost_data command is valid only when BIT(5) is enabled in the https_enable feature bitmap in HTTP POST command.

Binary Mode:

```
#define     HTTP_POST_BUFFER_LEN    900
struct
{
      uint8  current_chunk_length[2];
      uint8  buffer[HTTP_POST_BUFFER_LEN ];
}HttpPostDataReqFrameSnd;
```

Response:

Module may give http response in multiple chunks for a single HTTP POST request.

AT Mode:

Result Code	Description
AT+RSI_HTTPPOSTRSP=< more >< offset >< data_len >< data >	After the module sends out the HTTP POST request to the remote server, it may take some time for the response to come back. The response from the remote server is sent out to the Host from the module in the following form: AT+RSI_HTTPRSP=<More><Data Offset><Data Length><Data> The string AT+RSI_HTTPRSP is in uppercase ASCII.
ERROR<Error_code>	Failure

Response Parameters:

More(4 bytes): This indicates whether more HTTP data for the HTTP POST request is pending or not .

4- More data is pending from host

5-End of HTTP data content length

8–More data is pending. Further interrupts may be raised by the module till all the data is transferred to the Host.

9– End of HTTP data from server.

Offset(4 bytes): Always contains '0'.

data_len(4 bytes): data length in current chunk.

Data(Maximum 1024 bytes): Actual http data.

Possible error codes: Possible error codes for this command are 0x0015,0x0021, 0x0025, 0x002C, 0xFF74, 0xBBF0,0XBB38,0xBBEF,0xBB3E,0xBB38,0xBBE7.

Relevance:

This command is relevant when the module is configured in Operating Mode 0,1,2 and 6 .

Example

AT Mode:

at+rsi_httppost_data=10,1234567890

8.50 HTTP PUT

Description:

This command is used to transmit HTTP PUT request to a remote HTTP server. Module acts as a HTTP client when this command is issued.

This section explains different commands to use HTTP client PUT.

This command should be given only after Set IP Parameters command.

Following table explains list of HTTP PUT commands and their description.

HTTPPUT Command	Description
PUT_CREATE	Creates HTTP client thread and HTTP client socket. This should be the first command to use the HTTP client PUT
PUT_START	Connects to the specified HTTP server and creates the specified resource.
PUT_PACKET	To send the resource data packet
PUT_DELETE	To delete the HTTP client thread and socket

- **PUT_CREATE** should be called as a first command to use HTTP PUT.
- Once put create is successful **PUT_START** should be called to create the specified resource on the specified HTTP server.
- Once put start is successful **PUT_PACKET** should be called to send the resource data packet for the previously create resource.
- Call **PUT_DELETE** to delete HTTP Client thread and socket.

AT Mode:

HTTP PUT client has different command types. Based on the command type following parameters will change accordingly.

at+rsi_httpput=<command_type>,<remaining parameters>\r\n

Following are available command types.

HTTP PUT command	Command Type	Command Format
PUT create	**1**	at+rsi_httpput=1\r\n

PUT start	**2**	at+rsi_httpput=2,<ip_version>,<https _enable>,<port number>,<total contentlengtht>,<http buffer>\r\n Ip_version: IP version to use. 4 – For IPv4 6 – For IPv6 ip_address: Ipv4/Ipv6 address of the HTTP server. port number: HTTP server port number total_length: Total content length of the HTTP PUT data http_buffer : http buffer contains the username, password, host name, ip address, URL address, extended http header.
PUT PACKET	**3**	at+rsi_httpput=3,<current length>,<http buffer>\r\n current length : length of the current http put content chunk http_buffer: http buffer contains the put data
PUT DELETE	**4**	at+rsi_httpput=4\r\n

Binary Mode:

```
#define  HTTP_CLIENT_PUT_CREATE      1
#define  HTTP_CLIENT_PUT_START       2
#define  HTTP_CLIENT_PUT_PACKET      3
#define  HTTP_CLIENT_PUT_DELETE      4
#define  RSI_HTTP_CLIENT_PUT_BUFFER_LENGTH   900

typedef struct
{
   //! HTTP server ip version
   uint8 ip_version;
   //! HTTPS bit map
```

```
    uint8 https_enable[2];
    //! HTTP server port number
    uint8 port_number[4];
    //! HTTP Content Length
    uint8 content_length[4];
} rsi_http_client_put_start_t;

typedef struct
{
    //! Current chunk length
    uint16 current_length;
} rsi_http_client_put_data_req_t;

typedef struct
{
   //! Command type
   uint8  command_type;
   union
   {
      rsi_http_client_put_start_t http_client_put_start;
      rsi_http_client_put_data_req_t http_client_put_data_req;
    }  http_client_put_struct;
    uint8 http_put_buffer[RSI_HTTP_CLIENT_PUT_BUFFER_LENGTH];
} rsi_http_client_put_req_t;
```

Note:

Maximum supported PUT buffer length is 900 bytes

ip_version: ip version (4 or 6). This field is available only in Binary mode.

https_enable:

Set BIT(0) to enable HTTPS.

Set BIT(1) to enable NULL delimiter for HTTP buffer instead of comma

Set BIT(2) to use SSL TLS 1.0 version.

Set BIT(3) to use SSL TLS 1.2 version.

Set BIT(4) to use SSL TLS 1.1 version.

Note : If SSL is enabled by default it will use SSL TLS 1.0 and TLS 1.2 version.

BIT(2) and BIT(3) are valid only when HTTPS is enabled.

Note: If SSL is enabled, module will use same SSL version for all SSL sockets until module reboots.

Set BIT(6) to enable HTTP version 1.1.

port_number: HTTP server port number. If this is not mentioned default port numbers will be used.

http_put_buffer: This field contains following values in the order of < User_name >,< Password >,< Host_name >,< Ip_address >,< url >, < Extended_header > . The parameter Buffer is a character buffer.

If BIT(1) is not set in https_enable field

< User_name >,< Password >,< Host_name >,< Ip_address >,< url >, < Extended_header > fields should be delimited with comma(,)

If BIT(1) is set in https_enable field

< User_name >'\0'< Password >'\0'< Host_name >'\0'< Ip_address >'\0'<URL>'\0' < Extended_header > fields should be delimited with NULL('\0')

content_length – Total length of the resource content.

User_name – User_name for HTTP/HTTPS server authentication.

Password – Password for HTTP/HTTPS server authentication.

Host_name – Host name of the HTTP/HTTPS server.

Ip_address – IPv4/IPv6 address of HTTP/HTTPS server.

url – requested URL.

Extended_header - To append user configurable header fields to the default HTTP/HTTPS header

current_length – Current content chunk length

data – resource data to be send (This parameter is valid only for PUT PACKET command).

Note:

1. Maximum supported length for `User_name`, `Password` together is 278 bytes.

2. Maximum supported length for `Buffer` is (872-(length of `User_name`+length of `Password`)) bytes excluding delimiters.

3. If username, password, hostname and extended http headers are not required, user should send empty string separated by delimiter.

4. If content of any field contains comma(,) then NULL delimiter should be used.

5. When BIT(6) is enabled in https_enable feature bitmap, hostname is mandatory (To support HTTP version 1.1)

6. Multiple headers can be configured in 'extended_header' parameter. In this case, each header should be terminated by '/r/n' character sequence (0x0D, 0x0A in Hex) andHost needs to do byte stuffing. Refer to Table 28: Data Stuffing

7. Following HEADERS should NOT be provided in extended header if related info is provided in http_buffer or other parameters

Header	Condition
Host	If host related information is provided in 'http_buffer'
Authorization	If username and password are provided in 'http_buffer'
Connection	If using HTTP1.1

8. By default, 'Content-Type' header is appended to HTTP request if not provided in 'http_buffer'. Based on URL extension, Content-Type value is set to following,

URL extension	Content-Type
.txt/.TXT	text/plain
.htm/.HTM	text/html
default	text/plain
.gif/.GIF	image/gif
.xbm/.XBM	image/x-xbitmap

Response from the server:

Module may give http put server response in multiple chunks for a HTTP PUTpacket request.

AT Mode:

Result Code	Description
AT+RSI_HTTPRSP=< command_type >< end_of_file >< offset >< data_len >< data >	After the module sends out the HTTP POST request to the remote server, it may take some time for the response to come back. The response from the remote server is sent out to the Host from the module in the following form: AT+RSI_HTTPRSP=<command_type>< end_of_file >< Offset><Data

Result Code	Description
	Length><Data> The string AT+RSI_HTTPRSP is in uppercase ASCII.
ERROR<Error_code>	Failure

Response:

//! HTTP Client PUT response structure

typedef struct rsi_http_client_put_rsp_s

{

//! Receive HTTP Client PUT command type

uint8 command_type;

//! End of file/resource content

uint8 end_of_file;

}rsi_http_client_put_rsp_t;

Response Parameters:

command_type: HTTP Client PUT command type

end_of_file (Valid for HTTP PUT PKT): End of file or HTTP resource content

- 0 - More data pending from host
- 1 - End of HTTP file/resource content

Response from Server:

//! HTTP Client PUT response structure for server response for Put Packet command:

typedef struct rsi_http_client_put_rsp_s {

//! Receive HTTP Client PUT command type

Uint32command_type;

//! End of file/resource content

Uint32 end_of_file

//! Offset(4 bytes): Always contains '0'.

Uint32 offset;

//!data length in current chunk

Uint32 data_len;

//!Data(Maximum 1024 bytes): Actual http data from server.

Uint32 data;

}rsi_http_client_put_rsp_t;

Response Parameters:

command_type: "5", HTTP Client PUT packet command type during the response from the server.

end_of_file (Valid for HTTP PUT PKT): End of file or HTTP resource content.

- 8 - More data pending from server
- 9 - End of HTTP file from server/resource content

Offset(4 bytes): Always contains '0'.

Data_length: data length in current chunk

Data:(Maximum 1024 bytes): Actual http data from server.

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x0015, 0x0025, 0xBB38.

Relevance:

This command is relevant in Operating Modes 0, 1, 2, 6.

Example:

AT Mode:

HTTP client PUT Service:

at+rsi_httpput=1\r\n

at+rsi_httpput=2,4,0,80,19,username,password,192.168.1.100,192.168.1.100,/,\r\n

at+rsi_httpput=3,19,<html><body></html>\r\n

at+rsi_httpput=4\r\n

8.51 DFS Client (802.11h)

Description:

DFS client implementation is internal to the module. In 5 GHz band, the module does only passive scan in DFS channels. If the Access Point detects radar signals, it indicates to the module (client) to switch to a different channel by using the "channel switch frame". The module performs channel switch as per the AP's channel switch parameters. There is no command required to enable this feature, it is enabled by default.

8.52 Query Firmware Version

Description:

This command is used to query the version of the firmware loaded in the module.

Command Format:

AT Mode:

```
at+rsi_fwversion?\r\n
```

Binary Mode:

No payload required for this command.

Response:

AT Mode:

Result Code	Description
OK< fwversion><\0>	Successful exectution of command. All values returned in ASCII.
ERROR<Error code>	Failure.

Binary Mode:

```
struct {
      uint8 fwversion[20];
}rsi_qryFwversionFrameRecv;
```

Response Parameters:

`Fwversion`: Firmware version.

Possible error codes:

Possible error codes for this command are 0x002c.

Relevance:

This command is relevant In all modes.

Example:

AT Mode:

```
at+rsi_fwversion?\r\n
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x66 0x77 0x76 0x65 0x72
0x73 0x69 0x6F 0x6E 0x3F 0x0D 0x0A


Response:
OK 1.1.0\0\r\n
0x4F 0x4B  0x31  0x2E  0x31  0x2E  0x30  0X00 0x0D  0x0A
```

8.53 Query RSSI Value

Description:

This command is used to retrieve the RSSI value for Access Point to which the module is connected.

Command Format:

AT Mode:

```
at+rsi_rssi?\r\n
```

Binary Mode:

No Payload required.

Response:

AT Mode:

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

Binary Mode:

```
struct{
      uint8 rssiVal;
}rsi_rssiFrameRcv;
```

Response Parameters:

`rssiVal` : RSSI value (-n dBm) of the Access Point to which the module is connected. It returns absolute value of the RSSI. For example, if the RSSI is -20dBm, then 20 is returned.

Possible error codes:

Possible error codes for this command are 0x0021, 0x0025, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1(WiFi Direct client mode) and2.

Note:

Closer the RSSI value to '0', stronger the signal strength.

Example:

AT Mode:

```
at+rsi_rssi?\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x72 0x73 0x73 0x69 0x3F
0x0D 0x0A
```

Response:

```
For a RSSI of -20dBm, the return string is
```

```
OK < rssiVal =-20> \r\n
```

```
0x4F 0x4B 0x14 0x0D 0x0A
```

8.54 Flashtype

Description:

This command is used to Query the Flashtype of the module. This command can be given anywhere, once card ready is done.

Command Format:

AT Mode:

at+rsi_flashtype?\r\n

Binary Mode:

```
Nopayload required for this command.
```

Response:

AT Mode:

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

Binary Mode:

```
Struct {
        Uint8 Flashtype;
}rsi_qryflashtypeFrameRcv;
```

Response Parameters

Flashtype: Type of the flash whether it is micron or macronix.

1 – Micron

3 – Macronix

Possible error codes:

Possible error codes for this command are 0xfff8.

Relevance:

This command is relevant for all the operating modes.

Example:

AT Mode:

```
at+rsi_flashtype?\r\n
0x61 0x74 0x2B 0x72 0x73 0x69 0x5f 0x66 0x6c 0x61 0x73 0x68
0x74 0x79 0x70 0x65 0x3F 0x0D 0x0A
```

Response:

```
Ok 0x03\r\n
0x4F 0x4B 0x03 0x0D 0x0A
```

8.55 Query MAC Address

Description:

This command is used to query the MAC address of the module. This command can be issued anytime after init command.

Command Format:

AT Mode:

```
at+rsi_mac?\r\n
```

Binary Mode:

No payload required for this command

Response:

AT Mode:

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

Binary Mode:

```
struct {
        uint8     macAddress[6];
}rsi_qryMacFrameRecv;
```

Response Parameters:

`macAddress(6 bytes)`: MAC address of the module.

Possible error codes:

Possible error codes for this command are 0x002c .

Relevance:

This command is relevant in all operating modes.

Example:

AT Mode:

```
at+rsi_mac?\r\n


0x61   0x74   0x2B   0x72    0x73   0x69   0x5F  0x6D  0x61
0x63   0x3F   0x0D   0x0A
```

Response:

If the MAC ID is 0x00 0x23 0xA7 0x1B 0x8D 0x31, then the response is

```
OK< macAddress >\r\n
0x4F 0x4B 0x00  0x23  0xA7  0x1B  0x8D  0x31  0x0D  0x0A
```

For Concurrent Mode:

Response:

AT Mode:

Result Code	Description
OK<macAddress 1><macAddress2>	Successful execution of command.
ERROR<Error code>	Failure.

Binary Mode:

```
struct {
        uint8     macAddress1[6];
        uint8     macAddress2[6];

}rsi_qryMacFrameRecv;
```

Response Parameters:

`macAddress1(6 bytes), macAddress2(6 bytes)`: MAC address of the module for two interfaces.

Possible error codes:

Possible error codes for this command are 0x002c .

Relevance:

This command is relevant in all operating modes.

Example:

AT Mode:

```
at+rsi_mac?\r\n
```

```
0x61   0x74   0x2B   0x72    0x73   0x69   0x5F  0x6D  0x61
0x63   0x3F   0x0D   0x0A
```

Response:

```
OK< macAddress >\r\n
```

```
0x4F 0x4B 0x00  0x23  0xA7  0x1B  0x8D  0x31  0x00 0x23 0xA7
0x1B 0x8D 0x32 0x0D 0x0A
```

8.56 Query Network Parameters

Description:

This command is used to retrieve the WLAN and IP configuration parameters. This command should be sent only after the connection to an Access Point is successful.

Command Format:

AT Mode:

```
at+rsi_nwparams?\r\n
```

Binary Mode:

No payload required for this command.

Response:

AT Mode:

Result Code	Description
OK< wlan_state>< Chn_no >< psk>< mac_addr>< ssid>< connType>< sec_type>< dhcpMode >< ipaddr>< subnetMask>< gateway>< num_open_socks>< prefix_length>< ipv6addr><	Successful execution of command

Result Code	Description
defaultgw6> tcp_stack_used>[< sock_id> socket_type> sPort>< dPort> destIPaddr.ipv4_ address/ destIPaddr.ipv6_ address>] upto Max sockets supported.	
ERROR<Error Code>	Failure.

Binary Mode:

```
struct sock_info_query_t {
      uint8  sock_id[2];
      uint8  socket_type[2];
      uint8  sPort[2];
      uint8  dPort[2];
      union{
            uint8       ipv4_address[4];
            uint8       ipv6_address[16];
      }destIPaddr;
}sock_info_query_t;

struct  EVT_NET_PARAMS {
      uint8  wlan_state;
      uint8  Chn_no;
      uint8  psk[64];
      uint8  mac_addr[6];
      uint8  ssid[34];
      uint8  connType[2];
      uint8  sec_type;
      uint8  dhcpMode ;
      uint8  ipaddr[4];
      uint8  subnetMask[4];
      uint8  gateway[4];
      uint8  num_open_socks[2];
```

```
        uint8  prefix_length[2];
        uint8  ipv6addr[16];
        uint8  defaultgw6[16];
        uint8  tcp_stack_used;
        sock_info_query_t socket_info[10];
} rsi_qryNetParmsFrameRcv;
```

Response Parameters:

`wlan_state(1 byte)` : This indicates whether the module is connected to an Access Point or not.

0 – Not Connected

1 – Connected

`Chn_no(1 byte)` : Channel number of the AP to which the module joined

`Psk(64 bytes)` : Pre-shared key used

`Mac_Addr(6 bytes)` : MAC address of the module

`ssid(34 bytes)` : This value is the SSID of the Access Point to which the module is connected

`ConnType(2 byte)` :

0x0001 – Infrastructure

`Sec_type(1 byte)` : Security mode of the AP to which the module joined.

0– Open mode

1– WPA security

2– WPA2 security

3- WEP

4– WPA-Enterprise

5– WPA2-Enterprise

`Ipaddr(4 bytes)` : This is the IP Address of the module.

`SubnetMask(4 bytes)` : This is the Subnet Mask of the module

`Gateway(4 bytes)` : This is the Gateway Address of the module.

`dhcpMode(1 byte)` : This value indicates whether the module is configured for DHCP or Manual IP configuration for both IPv4 and IPv6.

dhcp_mode (0^{th} bit) -- 0 – Static IP configuration for IPv4

1 – Dynamic IP configuration for IPv4

dhcp_mode (1^{th} bit) -- 0 – Static IP configuration for IPv6

1 – Dynamic IP configuration for IPv6

`Num_open_socket(2 bytes)`: This value indicates the number of sockets currently opened

`Prefix_length(2 bytes)` **:** Prefix length of IPv6 address.

`Ipv6addr(16 bytes)` **:** This is the IPv6 Address of the module

`Defaultgw6(16 bytes)` **:** This is the IPv6 address of the default router

`tcp_stack_used(1 byte)` **:** TCP/IP stack used

1-IPv4

2-IPv6

4-Both IPv4 and IPv6 (dual stack).

`Sock_id(2 bytes)`: Socket handle of an existing socket

`Socket_type(2 bytes)` **:** Type of socket

0–TCP/SSL/websocket Client

2– TCP/SSL Server (Listening TCP)

4– Listening UDP

`Sport(2 bytes)`: Port number of the socket in the module.

`Dport(2 bytes)`: Destination port number of the remote peer.

`destIPaddr.ipv4_address(16 bytes)` **:** IPv4 address of the remote terminal. Only first 4 bytes of ipv4 address gets filled, remaining 12 bytes are zero.

`destIPaddr.ipv6_address(16 bytes)` **:** IPv6 address of the remote terminal

If the Set IP Params command was not sent to the module before Query Network Parameters command, the module returns a default values for

Following fields which should be ignored.

```
dhcpMode, ipaddr, subnetMask, gateway, num_open_socks,
prefix_length, ipv6addr, defaultgw6, tcp_stack_used,
socket_info
```

Possible error codes:

Possible error codes for this command are 0x0021, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2.

Example:

AT Mode:

```
at+rsi_nwparams?\r\n


0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6E 0x77 0x70 0x61 0x6D
0x73 0x3F 0x0D 0x0A
```

Response:

Response when nwparams command was given before ipconfig command so ipparameters are not set in this response, and sockets are not opened yet. Here SSID is 'cisco' and gateway is '192.168.100.76'

Note:

nwparams? command will not get PSK, it will get PMK to reduce connection time with AP from the next boot up. In the nwparams response structure you can find 32 bytes of PMK remaining are filled with Zeros.

```
< wlan_state>< Chn_no >< psk>< mac_addr>< ssid>< connType><
sec_type>< dhcpMode >< ipaddr>< subnetMask>< gateway><
num_open_socks>< prefix_length>< ipv6addr>< defaultgw6><
tcp_stack_used>[< sock_id>< socket_type>< sPort>< dPort><
destIPaddr.ipv4_address/ destIPaddr.ipv6_address>]
```

OK< wlan_state =0x01>< Chn_no =0x06>< psk =0x00(repeats 63 times)>< mac_addr =0x00 0x23 0xA7 0x16 0x16 0x16>< ssid =0x63 0x69 0x73 0x63 0x6F 0x00<repeats 27 times>< connType =0x01 0x00>< sec_type =0x00 >< dhcpMode =0x01>< ipaddr =0x00 0x00 0x00 0x00 >< subnetMask =0xFF 0xFF 0xFF 0x00 0>< gateway =0xC0 0xA8 0x64 0x4C>< num_open_socks =0x00 0x00>< prefix_length =0x00 0x00>< ipv6addr =0x00(16times)>< defaultgw6=0x00(16 times) >[<sock_id =0x00 0x00>< socket_type =0x00 0x00>< sPort =0x00 0x00>< dPort =0x00 0x00>< destIPaddr.ipv4_address/ destIPaddr.ipv6_address =0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00>]<repeats 11 times > 0x0D 0x0A.

8.57 Query Group Owner Parameters

Description:

This command is used to retrieve Group Owner (in case of Wi-Fi Direct) or connected client (in case of AP) related parameters.

This command should issue to the module only if the module has become a Group Owner in Wi-Fi Direct mode, or has been configured as an Access Point.

Command Format:

AT Mode:

```
at+rsi_goparams?\r\n
```

Binary Mode:

There is no payload for this command.

Response:

AT Mode:

Result Code	Description
OK< ssid>< bssid>< channel_number>< psk>< ipv4_address>< ipv6_address>< sta_count> [< ip_version>< mac>< ip_address. ipv4_address/ ip_address. Ipv6_address>] …Upto 10 sockets	Successful execution of command.
ERROR<Error Code>	Failure.

Binary Mode:

```
#define MAX_STA_SUPPORT 8

struct  go_sta_info_s
{
      uint8 ip_version[2];
      uint8 mac[6];
      uint8 ipv4_address[4];
      uint8 ipv6_address[16];
};

typedef struct {
      uint8   ssid[34];
      uint8   bssid[6];
      uint8   channel_number[2];
```

```
        uint8   psk[64];
        uint8   ipv4_address[4];
        uint8   ipv6_address[16];
        uint8   sta_count[2];
        struct  go_sta_info_s sta_info[MAX_STA_SUPPORT];
}rsi_qryGOParamsFrameRcv;
```

Response Parameters:

`SSID(34 bytes)`*:* SSID of the Group Owner/Access point.

`BSSID(6 bytes)`*:* MAC address of the module

`Channel_number(2 bytes)`: Channel number of the group owner/Access point.

`PSK(64 bytes)`: PSK configured in Wi-Fi Direct GO/Access point.

`IPv4 address(4 bytes)`: IPv4 Address of the module.

`IPv6 address(16 bytes)`: IPv6 Address of the module.

`Sta_count(2 bytes)`: Number of clients associated to the Group Owner/Access point. The least significant byte is returned first. A maximum of 8 clients is supported.

`Ip_version (2 bytes)`**:** IP version of the connected client.

`MAC(6 bytes)`: MAC address of the connected client

`ip_address. ipv4_address/ ip_address. Ipv6_address(16 bytes):`

IPv4/IPv6 address of the connected client. Last 12 bytes will be set to '0' in case of IPv4.

Possible error codes:

Possible error codes for this command are 0x0021, 0x0022, 0x0025, 0x002C.

Relevance:

This command is relevant when the module is configured in Operating Mode 1, 6 .

Example:

AT Mode:

```
at+rsi_goparams?\r\n


0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x67 0x70 0x61 0x72 0x61
0x6D 0x73 0x3F 0x0D 0x0A
```

```
Response:

OK< ssid =red>< bssid =0023A7556677>< channel_number =6>< psk
=1234>< ipv4_address =192.168.40.10>< ipv6_address =0x0(16
times)>< sta_count =1>
```

```
[< ip_version =4>< mac =68234286A3B2>< ip_address.
ipv4_address/ ip_address. Ipv6_address =192.168.40.12>]\r\n
```

```
0x4F 0x4B 0x72 0x65 0x64 0x00<repeats 30 times> 0x00 0x23 0xA7
0x55 0x66 0x77 0x06 0x00 0x31 0x32 0x33 0x34 0x00<repeats 60
times> 0x00 0xC0 0xA8 0x28 0x0A 0x01 0x04 0x00 0x00 0x68 0x23
0x42 0x86 0xA3 0xB2 0xC0 0xA8 0x28 0x0C 0x0D 0x0A
```

8.58 Soft Reset

Description:

This command acts as a software reset to the module. The module will reset all information regarding the WLAN connection and IP configuration after receiving this command. The Host has to start right from the beginning, from issuing the first command “Set Operating Mode” after issuing this command. This command is valid only in case of UART and USB-CDC interface.

Note:

Only OK response comes in UART, there will not be any CARD READY indication.

Command Format:

AT Mode:

```
at+rsi_reset\r\n
```

Binary Mode:

There is no payload for this command. This command is applicable only in case of UART, USB and USB-CDC.

Command Parameters:

```
N/A
```

Response:

AT Mode:

Result Code	Description
OK	Success
ERROR<Error Code>	Failure.

Binary Mode:

No payload required

Response Parameters:

N/A

Possible error codes:

Possible error codes are 0xFF82.

Relevance:

This command is relevant in all operating modes.

Example:

AT Mode:

```
at+rsi_reset
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x72 0x65 0x73 0x65 0x74
0x0D 0x0A
Response:

OK\r\n
0x4F 0x4B 0x0D 0x0A
```

8.59 Set/Reset multicast filter

Description:

This command is to set/reset the multicast MAC address bitmap to filter multicast packets. This command should be given after init command.

Payload:

AT Mode:

```
at+rsi_multicast_filter=< uMcastBitMapFrame >\r\n
```

Binary Mode:

```
uint16 uMcastBitMapFrame;
```

Payload Parameters:

`uMcastBitMapFrame`**:** There are two bytes in the payload which represent 2 parts. Lower byte represent the command type (cmd as mentioned below) and higher byte is the hash value (6 Bits) generated from the desired multicast MAC address (48 Bits) using hash function.

`uMcastBitMapFrame[0:1]` : These 2 bits represents the command type. Possible values are:

0- RSI_ MULTICAST_MAC_ADD_BIT (To set particular bit in multicast bitmap)

1- RSI_ MULTICAST_MAC_CLEAR_BIT (To reset particular bit in multicast bitmap)

2- RSI_ MULTICAST_MAC_CLEAR_ALL (To clear all the bits in multicast bitmap)

3- RSI_ MULTICAST_MAC_SET_ALL (To set all the bits in multicast bitmap)

`uMcastBitMapFrame[2:7]` : reserved.

`uMcastBitMapFrame[8:13]` : 6bit hash value generated from the hash algorithm which corresponds to the multicast MAC address is used to set/reset corresponding bit in multicast filter bitmap. This field is valid only if 0 or 1 is selected in command type (uMcastBitMapFrame[0:1]).

`uMcastBitMapFrame[14:15]` : reserved

Response:

AT Mode:

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

Binary Mode:

N/A

Response Parameters:

N/A

Possible error codes:

Possible error codes are 0x0021, 0x0025, 0x002c.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, and 2.

Note:

The Hash function is 8 CRC generator with the 2 MSB's ignored.

8.60 Join or Leave Multicast group

Description:

This command is used to join or leave a multicast group.

Command Format:

AT Mode:

For IPv4:

```
at+rsi_multicast=< req_type >,< group_address. ipv4_address
>\r\n
```

Note:

Only oneSOCKET can be configured as a MULTICAST Socket in IPv4

For IPv6:

```
at+rsi_multicast6=< req_type >,< group_address. Ipv6_address
>\r\n
```

Binary Mode:

```
struct {
      uint8 ip_version[2];
      uint8 req_type[2];
      union{
            uint8 ipv4_address[4];
            uint32 ipv6_address[4];
      }group_address;
} multicastFrameSnd;
```

Command Parameters:

`ip_version`: IP version 4 or 6. This parameter is available only in Binary mode.

`req_type`: Request type i.e. join request or leave request

0 - Leave multicast group

1 - Join multicast group

`group_address.ipv4_address/ group_address.ipv6_address` :

IPv4/IPv6 address of multicast group. Last 12 bytes are set to '0' in case of IPv4.

Response:

AT Mode:

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

Binary Mode:

N/A

Response Parameters:

N/A

Possible error codes:

Possible error codes 0x0021, 0x0025, 0x002C, 0xBB21,0xBB4c,0xBB17,0xBB55.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 and 6.

Example:

AT Mode:

For IPv4:

To join a multicast group:

```
at+rsi_multicast=1,239.0.0.0\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6D  0x75  0x6C  0x74
0x69  0x63  0x61  0x73  0x74 0x3D 0x31 0x2C 0x32 0x33 0x39
0x2E 0x30 0x2E 0x30 0x2E 0x30 0x0D 0x0A
```

Response:

```
OK \r\n
```

```
0x4F 0x4B  0x0D 0x0A
```

To leave a multicast group:

```
at+rsi_multicast=0,239.0.0.0\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6D  0x75  0x6C  0x74
0x69  0x63  0x61  0x73  0x74 0x3D 0x30 0x2C 0x32 0x33 0x39
0x2E 0x30 0x2E 0x30 0x2E 0x30 0x0D 0x0A
```

Response:

```
OK \r\n
0x4F 0x4B  0x0D 0x0A
```

For IPv6:

To join multicast ipv6 group:

```
at+rsi_multicast6=1,FF0E:0:0:0:0:0:0:1\r\n
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6D  0x75  0x6C  0x74
0x69  0x63  0x61  0x73  0x74 0x3D 0x31 0x2C 0x46 0x46 0x30
0x45 0x3A 0x30 0x3A 0x30 0x3A 0x30 0x3A 0x30 0x3A 0x30 0x3A
0x30 0x3A 0x31 0x0D 0x0A
```

Response:

```
OK \r\n
0x4F 0x4B 0x0D 0x0A
```

To leave multicast ipv6 group:

```
at+rsi_multicast6=0,FF0E:0:0:0:0:0:0:1\r\n
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x6D  0x75  0x6C  0x74
0x69  0x63  0x61  0x73  0x74 0x36 0x3D 0x30 0x2C 0x46 0x46
0x30 0x45 0x3A 0x30 0x3A 0x30 0x3A 0x30 0x3A 0x30 0x3A 0x30
0x3A 0x30 0x3A 0x31 0x0D 0x0A
```

Response:

```
OK \r\n
0x4F 0x4B 0x0D 0x0A
```

Note:

if MDNS feature is enabled,multicast group is not supported.

8.61 Ping From Module

Description:

This command is used to send the ping request to the target IP address.

Command Format:

AT Mode:

```
at+rsi_ping=< ip_version >,< ping_address. ipv4_address/
ping_address. Ipv6_address >,<  ping_size >,<timeout>\r\n
```

Binary Mode:

```
typedef struct rsi_ping_request_s {
      uint8  ip_version[2];
      uint8  ping_size[2];
      union {
            uint8  ipv4_address[4];
            uint32 ipv6_address[4];
      }ping_address;
      uint8 timeout[2];
} rsi_ping_request_t;
```

Command Parameters:

`ip_version` : IP version of the ping request.

4 - For IPV4

6 - For IPV6.

`ping_size` : ping data size to send. Maximum supported is 300 bytes.

Ping_address.ipv4_address /Ping_address.ipv6_address :Destination IPv4/IPv6 address.

`timeout`: ping request command response timeout.

Response:

AT Mode:

Result Code	Description
OK< ip_version><ping_size> < ping_address .ipv4_address/ ping_address .ipv6_address>	Successful execution of command
ERROR<Error code>	Failure.

Binary Mode:

```
typedef struct {
      uint8 ip_version[2];
      uint8 ping_size[2];
```

```
        union {
                uint8   ipv4_address[4];
                uint32  ipv6_address[4];
        }ping_address;
} rsi_uPingRsp;
```

Response Parameters:

`ip_version(2 bytes)`: IP version of the ping reply.

`ping_size (2 bytes)`: Contains the length of the data which is present in the ping reply.

`ping_address. ipv4_address/ping_address.ipv6_address`:

Ipv4/IPv6 address of the ping reply. Last 12 bytes are set to '0' in case of IPv4.

Note:

Default ping request command response timeout is 1 second.

Relevance:

This command is relevant when the module is configured in Operating Mode 0, 1, 2 or 6.

Possible Error Codes:

Possible error codes are 0x0025, 0x002C,0x002F, 0xBB29, 0xFF74,0x0015,0xBB21,0xBB4B,0xBB55.

Example:

AT Mode:

For IPv4:

```
at+rsi_ping=4,192.168.1.100,10\r\n
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x70 0x69 0x6E 0x67 0x3D
0x34 0x2C 0x31 0x39 0x32 0x2E 0x31 0x36 0x38 0x2E 0x31 0x2E
0x31 0x30 0x30 0x2C 0x31 0x30 0x0D 0x0A
```

Response:

```
0x4F 0x4B 0x04 0x00 0x0A 0x00 0xC0 0xA8 0x01 0x64 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0D 0x0A
```

For IPv6:

```
at+rsi_ping=6,2001.db8.1.0.0.0.0.123,10\r\n\
```

```
0x61 0x74 0x2B 0x72 0x73 0x69 0x5F 0x70 0x69 0x6E 0x67 0x3D
0x36 0x2C 0x32 0x30 0x30 0x31 0x2E 0x64 0x62 0x38 0x2E 0x31
0x2E 0x30 0x2E 0x30 0x2E 0x30 0x2E 0x30 0x2E 0x31 0x32 0x33
0x2C 0x31 0x30 0x0D 0x0A
```

Response:

```
0x4F 0x4B 0x06 0x00 0x0A 0x00 0xB8 0x0D 0x01 0x20 0x00 0x00
0x01 0x00 0x00 0x00 0x00 0x00 0x23 0x01 0x00 0x00 0x0D 0x0A
```

8.62 Loading the webpage

RS9113 features two types of WebPages, static and dynamic which can be stored in the module flash. Static pages allow plain html, CSS and JavaScript whereas dynamic pages allow only JSON data to be associated with the static WebPages. This JSON data can be stored, retrieved and updated independently. WebPages can fetch this data and display dynamically. Webpage fields can be modified from the host side as well as from the browser.

Maximum 10 webpages, each up to 4K in size can be stored in the module from Host. The size of each page can exceed the 4K limitation, but this would result in lesser number of webpages that can be stored. For example, host can request the module to store one 12K file, and seven 4K files. Maximum 10 JSON data objects with each NOT exceeding 512 bytes can be stored in the module up on request from Host. These JSON objects can only be stored only if an associated webpage exists.

8.62.1 Loading static webpage

Description:

This command is used to load or store a static webpage. This is also used to overwrite an existing static webpage. This command should be issued before join command.

If webpage total length is more than MAX_WEBPAGE_SEND_SIZE which is 1024 bytes, then the host has to send webpage in multiple chunks.

Command Format:

AT Mode:

```
at+rsi_webpage= < filename >,< total_len >,< current_len >,<
has_json_data >,< webpage >\r\n
```

Binary Mode:

```
#define MAX_WEBPAGE_SEND_SIZE 1024
typedef struct {
      uint8 filename[24];
      uint8 total_len[2];
      uint8 current_len[2];
      uint8 has_json_data;
```

```
      uint8 webpage[MAX_WEBPAGE_SEND_SIZE];
} WebpageSnd_t;

struct{
      WebpageSnd_t    Webpage_info;
}webServFrameSnd;
```

Command Parameters:

The command parameters are outlined below:

filename: Name of the file to load the webpage.

total_len: Total length of the webpage

current_len: Length of the current webpage chunk

has_json_data:

1 - If file has associated JSON data

0 - If there is no associated data. In this case, webpage is static page

webpage: This is the webpage in HTML

Response:

AT Mode:

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

Binary Mode:

There is no response payload.

Response Parameters:

There is no response payload.

Possible Error codes:

Possible error codes are 0x0021, 0x0015,0x0025, 0x00C1, 0x00C2, 0x00C3, 0x00C5, 0x00C6,0x00C8

Relevance:

This command is relevant when the module is configured in Operating Mode - 0, 1, 2 or 6.

Example:

The host can also overwrite an existing page. Thiscan be achieved by issuing the same command with the same filename. In this case no explicit erase is required. Please note that the size of the new file should not exceed the size of the old file rounded up to the 4K chunk boundary. Precisely, if an old file uses 2 chunks of 4K (i.e. Up to 8K in size), then the new file cannot exceed 8K in size. To overcome this limitation, the host should erase the existing file and then write a new file which can exceed the size of the old file provided the device has enough space available.

AT Mode:

```
at+rsi_webpage=sample.html,2783,1024,0,<html><head><title></ti
tle>[1024-chars]\r\n
```

Response:

```
OK\r\n
```

Example

```
at+rsi_webpage=page1.html,1024,1024,0,<html><head><title></tit
le>[1024-chars]\r\n
```

8.62.2 Loading the dynamic webpage(Create JSON)

Description:

This command is used to load the associatedJSON data with the static WebPages.

Command Format:

AT Mode:

```
at+rsi_jsoncreate= < filename >,< total_length >,<
current_length >,< json_data >\r\n
```

Binary Mode:

```
typedef struct rsi_jsonCreateObject_s {
#define RSI_JSON_MAX_CHUNK_LENGTH 1024
      char        filename[24];
      uint8       total_length[2];
      uint8       current_length[2];
      char        json_data[RSI_JSON_MAX_CHUNK_LENGTH];
} rsi_jsonCreateObject_t;
```

Command Parameters:

`filename`: This is the webpage filename with which this JSON data is associated.

`total_length`: This is the total length of the JSON

`current_length`: This is the length of the current JSON chunk

`json_data`: This is the JSON object that stores the data of the webpage.

Note:
The maximum supported JSON object length is 512 bytes.

Response:

AT Mode:

Result Code	Description
OK	Success.
ERROR<Error code>	Failure.

Binary Mode:

There is no response payload.

Response Parameters:

There is no response payload.

Possible Error codes:

Possible error codes are 0x0015, 0x0021, 0x0025,0x002C,0x00B1,0x00B2,0x00B3,0x00B4,0x00B5,0x00B6.

Relevance:

This command is valid when the module is configured in Operating Mode 0, 1, 2 or 6.

Example:

AT Mode:

The webpage should already be present in module's flash with bool json set to 1 before issuing this command.

```
at+rsi_jsoncreate=sample.html,60,60,{“temp”:27, “accx”:2.4,
“accy”:2.6, “accz”:1.1, “enabled”:true}\r\n
```

Response:

```
OK\r\n
```

8.63 Clearing the webpage

These commands are used to erase the webpage and information related to webpage from the flash.

8.63.1 Erasing the webpage

Description:

This command is used to erase the webpage file from the FLASH. It should be issued before join command. The erase command should be used specifying the filename as this will free up the number of 4K chunks occupied by the webpage.

Command Format:

AT Mode:

```
at+rsi_erasefile= < filename > \r\n
```

Binary Mode:

```
typedef struct rsi_tfs_erase_file_s {
      char filename[24];
} rsi_tfs_erase_file_t;
```

Command Parameters:

filename: name of the webpage file that has to be erased.

Response:

AT Mode:

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

Binary Mode:

There is no response payload.

Response Parameters:

There is no response payload.

Relevance:

This command is valid when the module is configured in Operating Mode - 0, 1, 2 or 6.

Possible Error codes:

Possible error codes are 0x0021, 0x0025,0x002C, 0x00C4

Example:

AT Mode:

```
at+rsi_erasefile=sample.html\r\n
```

Response:

```
OK\r\n
```

8.63.2 Erasing the JSON Data

Description:

This command is used to erase the JSON data file associated with a webpage from the FLASH.

Command Format:

AT Mode:

```
at+rsi_erasejson= < filename > \r\n
```

Binary Mode:

```
typedef struct rsi_tfs_erase_file_s {
      char filename[24];
} rsi_tfs_erase_file_t;
```

Command Parameters:

`filename` : This is the name of the webpage file of which JSON data has to be erased.

Response:

AT Mode:

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

Binary Mode:

There is no response payload.

Response Parameters:

There is no response payload.

Relevance:

This command is relevant when the module is configured in Operating Mode - 0, 1, 2 or 6.

Possible Error Codes:

Possible error codes are 0x0021, 0x0025, 0x002C,0x00B4.

Example:

AT Mode:

```
at+rsi_erasejson=sample.html\r\n
```

Response:

```
OK\r\n
```

8.63.3 Clear all the WebPages

Description:

This command is used to erase all the WebPages in the file system.

Command Format:

AT Mode:

```
at+rsi_clearfiles= < clear > \r\n
```

Binary Mode:

```
typedef struct rsi_tfs_clear_files_s {
      uint8       clear;
```

```
} rsi_tfs_clear_files_t;
```

Command Parameters:

`clear`: Set '1' to clear files.

Response:

AT Mode:

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

Binary Mode:

There is no response payload.

Response Parameters:

There is no response payload.

Relevance:

This command is relevant when the module is configured in Operating Mode - 0, 1, 2 or 6.

Possible Error Codes:

Possible error codes are 0x0021, 0x0025, 0x002C.

Example:

AT Mode:

Command:

```
at+rsi_clearfiles=1\r\n
```

Response:

```
OK\r\n
```

8.64 Loading of Store Configuration Page

Store Configuration Page is a static web page.

In order to load,store 'Configuration Page',the host needs to use **"store_config.html"** as the URL name and no need of loading the associated JSON data (Dynamic web page).

The size of the page should not exceed the 40K.

The fields present in the webpage can be disabled by the host. But,the host cannot change the webpage field's structure. The host can change the **"LOGO"** and **"TITLE"**in this page.

For loading **Store configuration page** refer to the section**8.62Loading the webpage.**

8.64.1 Wireless Configuration Page Bypass and Override Option

8.64.1.1 Default configuration page bypass

If host enables this feature, the module will bypass the HTTP server root path. Whenever user gives module's IP address in the browser, the module will give an asynchronous indication to the host requesting for web page. The host has to respond with URL RESPONSE frame. To enable this feature, the host need to set BIT(2) in ext_tcp_ip_ feature_bit_map of opermode command.

Note:

1. Please note url request (file) should have .html extension, if the contetnt has html data.
2. Index.html is the default url_name

For more information ont web page bypass, refer to the section **8.65Web Page request to Host**.

8.64.1.2 Default configuration page override

This feature overrides the existing default Wireless Configuration Page (index.html). In this, the host needs to load a static webpage with file name as **index.html** which will override the module's default webpage. Whenever the user gives module's IP address in the browser, the module will give the static index.html page (as the default web page).

For more information on static web page loading, refer to the section **8.62.1Loading static webpage**.

8.65 Web Page request to Host

Description:

This command is used to send URL response.

Whenever unavailable page request is received, the module will give asynchronous indication to the host for requesting web page. The host has to respond with **URL RESPONSE** frame.

When the module receives the query, it checks whether the page is already present in its memory or not. If yes, it sends out the page to the remote terminal and the query is serviced. If not, it sends the asynchronous URL REQUEST.

The Asynchronous Message is outlined below.

AT Mode:

```
AT+RSI_URLREQ< url_length>< url_name >< request_type
><post_content_length >< post_data >\r\n
```

After receiving this asynchronous frame from the module, the host has to respond with URL RESPONSE frame.

Note:

Please note that whenever you are giving URL request from a browser to the module, .html should be given.

Binary Mode:

```
#define MAX_URL_LENGTH        40
#define MAX_POST_DATA_LENGTH  512
typedef struct    {
      uint8  url_length;
      uint8  url_name[MAX_URL_LENGTH];
      uint8  request_type;
      uint8  post_content_length[2];
      uint8  post_data[MAX_POST_DATA_LENGTH];
}rsi_urlReqFrameRcv;
```

`url_length(1 byte)` : This is the number of characters in the requested URL.

`url_name (41 bytes)` : This is the actual URL name.

`request_type(1 byte)` : –This is the type of the request received.

0 – HTTP GET request

1 – HTTP POST request

`post_content_length(2 bytes)` : This is the length of the post content

`post_data(512 bytes)` : This is the HTTP POST received.

post_content_length, post_data fields are valid if request_type is **HTTP POST**. These fields can be ignored in case the request_type is **HTTP GET**.

Command Format:

AT Mode:

```
at+rsi_urlrsp=< total_len >,< more_chunks >,< webpage >\r\n
```

Once the host receivs this asynchronous message from the module,it should fetch the page from its memory and give it back to the module with the message:

```
at+rsi_urlrsp=< total_len >,< more_chunks >,< webpage >\r\n
```

Binary Mode:

```
#define MAX_HOST_WEBPAGE_SEND_SIZE 1400
typedef struct {
      uint8 total_len[4];
      uint8 more_chunks;
      uint8 webpage[MAX_HOST_WEBPAGE_SEND_SIZE];
} HostWebpageSnd_t;
```

Command Parameters:

`total_len` - This is the total number of characters in the page. If the queried web page is not found, the Host should send '0' for this parameter.

`more_chunks` –

'0'- There are no more segments coming from the Host after this segment

'1'- There is one more segment coming from the Host after this Segment

`webpage` - This is the actual source code of the current segment

Response:

AT Mode:

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

Binary Mode:

There is no response payload.

Response Parameters:

There is no response payload.

Relevance:

This command is relevant when the module is configured in Operating Mode - 0, 1, 2 or 6.

Possible Error Codes:

Possible error codes are 0x0015,0x0021,0x0025,0x002C.

Example:

AT Mode:

Example 1:

If the web page source code is of 3000 characters, the Host should send it through 3 segments, the first two of 1400 bytes, and the last one of 200 bytes as shown below:

```
AT+RSI_URLRSP=< total_len =3000>, < more_chunks =1>,
< webpage =code of the 1st segment>\r\n.
AT+RSI_URLRSP=< total_len =3000>,< more_chunks =1>,
< webpage =code of the 2nd segment>\r\n
AT+RSI_URLRSP=< total_len =3000>,< more_chunks =0>,
< webpage =code of the 3rd segment>\r\n
```

Example 2:

If the queried web page is not found in the host, the host should send the below command:

```
AT+RSI_URLRSP=0\r\n
```

8.66 Set Region

Description:

This command is used to configure the device in order to operate as per the regulations of its operating country.This command should be followed immediately after init command.

Command Format:

AT Mode:

```
at+rsi_setregion=<setregion_code_from_user_cmd>,<region_code>,
<module_type>/r/n
```

Binary Mode:

```
typedef struct{
      uint8  setregion_code_from_user_cmd;
      uint8  region_code;
      uint16 module_type;
}rsi_setregion_t;
```

Command Parameters:

`setregion_code_from_user_cmd`: This is to **Enable/Disable**the set region code from user.

- 1 - Enable - Use the region information from user command
- 0 – Disable - Use the region information from beacon(country Ie)

Note:
1. In Wi-Fi Direct mode, setting the region information from beacon is not supported.
2. For world mode domain, setting the region information from beacon is not supported.

`Region_code`:

0/1-US domain(Default)

2-Europe domain

3-Japan domain

4-World mode domain

`Module_type`: This is to enter the module on board antenna. The valid values are outlined below:

- 1 - If on board antenna is present
- 0 - If on board antenna is not present

Note:

1. The module type from host is applicable if module manufacturing version is below 3.1 (i.e. manufacturing version 3 and subversion 1)
2. To take module type from host, the host need to set extended custom feature bitmap BIT[2] of opermode command.

Response:

AT Mode:

Result Code	Description

Result Code	Description
OK< region_code>	Successful execution
ERROR	Failure.

Binary Mode:

```
typedef struct {
      uint8 region_code;
}rsi_uSetRegionRsp;
```

Response Parameters:

`region_code(1 bytes):`

0x00 (Invalid region code due to no beacon found)

0x01(US domain)

0x02(Europe domain)

0x03(Japan Domain)

0x04(World mode domains)

Note:

1) In dual band mode, if country element is extracted from beacon and in any case, if 2.4GHz band and 5GHz bands are not matching, then an error would be displayed and the region would be set to the default one i.e. **US.**

Relevance:

This command is relevant when the module is configured in Operating Mode 0,1,2 and 8 modes.

The rules followed for the regions supported by the Silicon Labs module are tabulated

Rule No.	band	First channel	Number Of channels	Last Channel	Maximum power in dBm	Scan type
1	2.4GHz	1	11	11	27	Active
2	5GHz	36	4	48	16	Active
3	5GHz	52	4	64	23	Passive
4	5GHz	100	5	116	23	Passive
5	5GHz	120	3	128	23	Passive

Rule No.	band	First channel	Number Of channels	Last Channel	Maximum power in dBm	Scan type
6	5GHz	132	3	140	23	Passive
7	5GHz	149	5	165	29	Active

Table 29: Regulations followed for US domain

Rule No	band	First channel	Number Of channels	Last Channel	Maximum power in dBm	Scan type
1	2.4GHz	1	13	13	20	Active
2	5GHz	36	4	48	23	Active
3	5GHz	52	4	64	23	Passive
4	5GHz	100	11	140	30	Passive

Table 30 : Regulations followed for Europe domain

Rule No	band	First channel	Number Of channels	Last Channel	Maximum power in dBm	Scan type
1	2.4GHz	1	14	14	20	Active
2	5GHz	36	4	48	20	Active
3	5GHz	52	4	64	20	Passive
4	5GHz	100	11	140	30	Passive

Table 31 : Regulations followed for Japan domain

Note:
Even though the transmit power levels w.r.t region goes beyond 20dBm,Silicon Labs module will be supporting maximum of 18dBm only.

Possible Error Codes:

Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0xFF82, 0x00CC, 0x00C7, 0x00CD, 0x00CE

Example:

AT Mode:

Command: Setting Japan region for the module type 1 (on board antenna present).

```
at+rsi_setregion=1,3,1\r\n
```

Command: Setting world mode region domain for the module type 1 (on board antenna present).

```
at+rsi_setregion=1,4,1\r\n
```

Response:

```
OK\r\n
```

8.67 Set Region of Access point

Description:

This command is used to set the region domain of the module in Access point mode. This command helps device to self-configure and operate according to the regulations of its operating country and includes parameters like country name, channel quantity and maximum transmission level. These parameters are added in Country information element in the beacons and probe responses. This command should be issued immediately after init command.

Command Format:

AT Mode:

```
at+rsi_setregion_ap=<setregion_code_from_user_cmd>,<
country_code>,<no_of_rules>,[<first_channel>,<Number of
channel>,< max_tx_power>,< First channel >,<Number of
channels>,<max_tx_power>],…….no of rules times/r/n
```

Binary Mode:

```
#define MAX_POSSIBLE_CHANNEL 24
struct{
      uint8  setregion_code_from_user_cmd;
      uint8  country_code[3];
      uint32 no_of_rules;
      struct{
            uint8 first_channel;
            uint8 no_of_channels;
            uint8 max_tx_power;
      }channel_info[MAX_POSSIBLE_CHANNEL];
}setRegionApFrameSnd;
```

Command Parameters:

`setregion_code_from_user_cmd`: set region code from user command enable/disable

1- Enable-Get the region information from user command

0- Disable-Get the region information based on region code from module's internal memory

Note:

In Wifi Direct mode, setting the region information from user command is not supported.

`Country code` :Country code is of 3 bytes.Country code is case sensitive and should be in *Upper case*.If the first parameter is 1,the second parameter should be one of the these 'US','EU','JP' country codes.

Note:

If the country code is of 2 characters,3rd character should be <space>

Note:

The below parameters are considered only if the first parameter is 1

`Number of rules`: Number of rules (n) for the given domain

`First channel`: Start channel for the nth rule

`Number of channels`: number of channels holding the same rule from the start channel

`Maximum transmit power`: Maximum transmit power used in that set of channels.

Response:

AT Mode:

Result Code	Description
OK	Successful execution
ERROR	Failure.

Binary Mode:

There is no response payload for this command.

Relevance:

This command is relevant when the module is configured in Operating Mode 6.

NOTE:

1) AP configuration in DFS channels (52 to 140) is not supported

2) Though the transmit power levels w.r.t., to region are greater than 20dBm, Silicon Labs module is supporting maximum of 18dBm

Possible Error Codes:

Possible error codes for this command are 0x0021,0x0025,0x002C, 0x00ca, 0x00cb,0x00cc,0xFF71,0xFF82

Example:

AT Mode:

Example:1

`at+rsi_setregion_ap=1,US<space>,2,1,4,23,5,7,30\r\n`

Explanation:

From the above command, consider the first rule 1,4,23

First channel is given as 1, and no of channels is 4.this means channels 1 to 4 (1,2,3,4)holds the maximum Tx power 23dBm

Consider the second rule 5,7,30

First channel is given as 5, and no of channels is 7.this means channels 5 to 11 (5,6,7,8,9,10,11)holds the maximum Tx power 30dBm

Note: The Country code given in the command reflects as it is in the beacon frame

Example: 2

Command:

at+rsi_setregion_ap=0,US<space>\r\n

Response:

OK\r\n

Note: Refer to the tables in the Set Region section for the region supported and domain rules followed by Silicon Labs module

8.68 PER statistics of the module

Description :

This command is used to get the Transmit(TX) & Receive(RX) packets statistics. When this command is given by the host by enabling this feature with valid channel number ,module gives the statistics to host for every second until this feature is disabled. This command can be given after init command.

Command Format:

AT Mode:

```
at+rsi_per_stats=< per_stats_enable >,< per_stats_channel
>\r\n
```

Binary Mode:

```
struct {
      uint8 per_stats_enable[2];
      uint8 per_stats_channel[2];
} perStatsFrameSnd;
```

Command Parameters:

per_stats_enable: Enable /disable per status feature

0 – Enable

1 – Disable

per_stats_channel: valid channel number in which user wants to get the stats. Channel number is not required if the first parameter is 1(Disable)

Response:

AT Mode:

OK	Successful execution
ERROR	Failure.

Once PER stats is enabled, module sends asynchronous message for every second. Following list of fields will be given to host.

Result Code	Description
AT+RSI_PER_STATS=<tx_pkts>,<reserved_1>,<tx_retries>,<crc_pass>,<crc_fail>, <cca_stk>,<cca_not_stk>,<pkt_abort>,<fls_rx_start>,<cca_idle>,<reserved_2>,<rx_retries>,<reserved_3>,<cal_rssi>,<reserved_4>,<xretries>,<max_cons_pkts_dropped>,<reserved_5><bss_broadcast_pkts>,<bss_multicast_pkts>,<bss_filter_matched_multicast_pkts>	

Binary Mode:

There is no response payload for this command. Once PER stats is enabled, module sends asynchronous message for every 1 second. Following list of fields will be given to host.

```
typedef struct per_stats_s {
      uint8 tx_pkts[2];
      uint8 reserved_1[2];
      uint8 tx_retries[2];
      uint8 crc_pass[2];
      uint8 crc_fail[2];
      uint8 cca_stk[2];
      uint8 cca_not_stk[2];
      uint8 pkt_abort[2];
      uint8 fls_rx_start[2];
      uint8 cca_idle[2];
      uint8 reserved_2[26];
      uint8 rx_retries[2];
      uint8 reserved_3[2];
      uint8 cal_rssi[2];
      uint8 reserved_4[4];
      uint8 xretries[2];
      uint8 max_cons_pkts_dropped[2];
      uint8 reserved_5[2];
      uint8 bss_broadcast_pkts[2];
      uint8 bss_multicast_pkts[2];
      uint8 bss_filter_matched_multicast_pkts[2];
}rsi_uPerStatsRsp;
```

Response Parameters:

`tx_pkts`(2 bytes):Number of TX packets transmitted

`reserved_1(2 bytes)` : Reserved

`tx_retries(2 bytes)` : Number of TX retries happened

`crc_pass(2 bytes)` : Number of RX packets that passed CRC

`crc_fail(2 bytes)` : Number of RX packets that failed CRC

`cca_stk(2 bytes)` : Number of times cca got stuck

`cca_not_stk(2 bytes)` : Number of times cca didn't get stuck

`pkt_abort(2 bytes)` : Number of times RX packet aborts happened

`fls_rx_start(2 bytes)` : Number of false rx starts.If Valid wlan packet is received and is dropped due to some reasons.

`cca_idle(2 bytes)` : CCA idle time

`reserved_2(2 bytes)` : Reserved

`rx_retries(2 bytes)` : Number of RX retries happened

`reserved_3(2 bytes)` : Reserved

`cal_rssi(2 bytes)` : The calculated RSSI value of recently received RX packet

`reserved_4(2 bytes)` : Reserved

`xretries(2 bytes)` : Number of TX Packets dropped after maximum retries

`max_cons_pkts_dropped(2 bytes)` : Number of consecutive packets dropped after maximum retries

`reserved_5(2 bytes)` : Reserved

`bss_broadcast_pkts(2 bytes)` : BSSID matched broadcast packets count.

`bss_multicast_pkts(2 bytes)` : BSSID matched multicast packets count.

`bss_filter_matched_multicast_pkts(2 bytes)` : BSSID and multicast filter matched packets count. The filtering is based on the parameters given in multicast filter command Set/Reset Multicast filter. If multicast filter is not set then this count is equal to bss_multicast_pkts count.

Note:

1) In PER mode(opermode 8) following stats related to RX packets (`crc_pass`, `crc_fail`, `cca_stk`, `cca_not_stk`, `pkt_abort`, `fls_rx_start`, `cca_idle`) are only valid, remaining fields can be ignored
2) The multicast stats are valid only in associated state in client mode and are invalid in non-associated state.In associated state,the stats are for packets which are destinated for the module's MAC address only.And in non associated state, the stats are for all the packets received,irrespective of the destination MAC address.
3) The parameters valid in other than PER mode(opermode 0,1,2,6) mode : `tx_pkts`, `tx_retries cal_rssi`, `xretries`, `crc_pass`, `max_cons_pkts_dropped`, `crc_fail`, `cca_stk`, `cca_not_stk`, `pkt_abort`, `fls_rx_start`, `cca_idle`, `bss_broadcast_pkts`, `bss_multicast_pkts`, `bss_filter_matched_multicsast_pkts`

Relevance:

This command is valid in all operating modes.

Possible Error Codes:

Possible error codes for this command are 0x0021, 0x0025, 0x002c, 0x000A.

8.69 Query WLAN Connection Status

Description:

This command queries the WLAN connection status of the Wi-Fi module.

This command is available only in Binary mode.

Command Format:

No Payload required.

Command Parameters:

No parameters

Response:

```
typedef struct {
uint8   state[2];
} rsi_conStatusFrameRcv;
```

Response Parameters:

`State(2 bytes):`

- 1 - Connected to AP
- 0 - Not connected to AP

Possible error codes:

Possible error codes are 33, 37, 44

Relevance:

This command is relevant when the module is configured in Operating Mode 0, and 2.

8.70 Remote Socket Closure

Description:

This is an asynchronous message which will be given to host in the following cases.

1. When the remote peer closes connected TCP/SSL/Web socket.

2. When module is not able to send data over socket because of unavailability of remote peer.

3. When remote peer disappears without intimation then module closes socket after TCP keep alive time(~20 minutes) and sends this message to host.

Command Format:

N/A

Command Parameters:

N/A

Response:

AT Mode:

```
AT+RSI_CLOSE< socketDsc>< bytesSent>\r\n
```

Binary Mode:

```
struct {
      uint8 socketDsc[2];
      uint8 bytesSent[4];
}rsi_socketCloseFrameRecv;
```

Response Parameter:

`socketDsc(2 bytes)` : Socket descriptor of the socket which is closed.
`bytesSent(4 bytes)` : Number of bytes sent successfully on that socket.

Possible error codes:

No possible error code as it is asynchronous message from module to host.

Relevance:

This command is relevant when the module is configured in Operating Mode 0,1 , 2 and 6.

8.71 Bytes Transmitted Count On Socket

Description:

This command is used to get the number of bytes transmitted successfully by the module on a given socket.

Command Format:

AT Mode:

```
at+rsi_bytes_sent_count=< sock_handle>\r\n
```

Binary Mode:

```
typedef struct {
```

```
        uint8 sock_handle[2];
}rsi_SentBytesReq;
```

Command Parameters:

`socket_handle`: socket handle on which number of bytes have been successfully transmitted.

Response:

AT Mode:

OK<`sock_handle`><`SentBytescnt`>	Success.
ERROR	Failure.

Binary Mode:

```
typedef struct {
        uint8 sock_handle[2];
        uint8 SentBytescnt[4];
}rsi_SentBytesRsp;
```

Response Parameters:

`socket_handle(2 bytes)`: Socket handle on which number of bytes have been successfully transmitted.

`SentBytescnt(4 bytes)`: Number of bytes sent successfully on the given socket handle

Relevance:

This command is relevant when the module is configured in Operating Mode 0,1 , 2 and 6.

Possible Error Codes:

Possible error codes for this command are 0xFF86, 0XFFFA, 0xFF82, 0x002C, 0x0025, 0x0021.

8.72 Debug prints on UART 2

Description:

This command is used for debug prints on UART 2 interface. Host can get 4 types of debug prints based on the assertion level and assertion type.

Command Format:

AT Mode:

```
at+rsi_debug=< assertion_type >,< assertion_level >\r\n
```

Binary Mode:

```
struct {
      uint32     assertion_type;
      uint32     assertion_level;
} debugFrameSnd;
```

Command Parameters:

`assertion_type`: Possible values are 0 to 4.

`assertion_level` : Possible values 0 to 15. 1 being least level and 15 being highest level of debug prints. 0 is to disable all the prints.

Note:

1) If debug prints are enabled once, to disable the debug prints host is supposed to give the same command with assertion type and assertion level as 0.
2) Baud rate for UART 2 on host application side should be 460800.

Response:

AT Mode:

OK	Successful execution
ERROR	Failure.

Binary Mode:

There is no response payload for this command.

Possible Error Codes:

Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0xFFF8

Relevance:

This command can be given at any time.

8.73 Asynchronous message for connection state notification

Description:

Asynchronous message are used to indicate module state to host.These message are enabled by setting 10^{th} bit in customer feature select bitmap in opermode command, please refer opermode command.

Command Format:

N/A

Response:

AT Mode:

AT+RSI_STATE-I<TimeStamp >,<StateCode>,<reason_code>,<rsi_channel>,<rsi_rssi>,<rsi_bssid>\r\n	This type of asynchronous message is given by the module when it is in scanning state.
AT+RSI_STATE-II<TimeStamp >,<StateCode>,<reason_code>,<rsi_channel>,<rsi_rssi>,<rsi_bssid>\r\n	This kind of message is given by the module once the scan results are observed and decided to join or not to join/Rejoin to AP.
AT+RSI_STATE-III< TimeStamp >,<StateCode>,<reason_code>,<rsi_channel>,<rsi_rssi>,<rsi_bssid>\r\n	Once the association or disassociation is done, module will give final state asynchronous message

Binary Mode:

```
typedef struct rsi_state_notification_s {
      uint8   TimeStamp[4];
      uint8   StateCode;
      uint8   reason_code;
      uint8   rsi_channel;
      uint8   rsi_rssi;
      uint8   rsi_bssid[6];
}
```

Response Parameters:

`TimeStamp(4 bytes)` **:** This is value of counter at the time of message. This counter is continuously incrementing by one per 100ms time.

`StateCode(1 byte)` **:** This field indicates state of the module. state code contain two parts (upper nibble/lower nibble).

Upper nibble represent the state of rejoin process. Following are the possible values of StateCode represented by the upper nibble of state code.

Scan Trigger State (state-I): Indicates the reason for scan triggered

0x00-Startup (Initial Roam)

0x10-Beacon Loss (Failover Roam)

0x20-De-authentication (AP induced Roam / Disconnect from supplicant)

Scan Result/Decision State(state – II): Indicates a state change based on scan result

0x50-Current AP is best

0x60-Better AP found

0x70-No AP found

Final Connection State (state – III) :Indicates the connection state change

0x80-Associated

0x90-Unassociated

Following are the possible values of StateCode represented by the lower nibble of state code

Indicates reason for state change

0x00-No reason specified

0x01-Authentication denial

0x02-Association denial

0x03-AP not present

0x05-WPA2 key exchange failed

`reason_code(1 byte)` **:** This is used to get the reason code from firmware point of view. Following are the possible reason code for the failure.

0x00-No reason specified

0x01-Authentication denial

0x02-Association denial

0x10-Beacon Loss (Failover Roam)

0x20-De-authentication (AP induced Roam/Deauth from supplicant)

0x07-PSK not configured

0x09-Roaming not enabled

`rsi_channel(1 byte)`**:**

Following represents the meaning of AP channel at the given stage.

State-I: channel of association or Invalid if it is startup.

State-II: channel of next association if module finds better AP in bgscan result.

State-III: Channel at the time of association.

If value of rsi_channel is 0, it means channel information is not available.

`rsi_rssi(1 byte)` **:** Following represents the meaning of AP RSSI at the given stage.

State-I: RSSI of AP at the time of trigger.

State-II: RSSI of next association.

State-III: RSSI at the time of final association.

If value of rsi_rssi is 100, it means RSSI information is not available.

`rsi_bssid(6 bytes)` **:** Following represents the meaning of AP MAC at the given stage.

State-I: MAC of AP at the time of scan trigger.

State-II: MAC of next association.

State-III: MAC at the time of association.

Note:
If the value of AP MAC is 00:00:00:00:00:00,it means MAC information is not available

Response:

N/A

Relevance:

This command is relavant in opermodes 0,1,2 and 6.

Possible Error Codes:

N/A

Note:
By default this feature is disabled. To enable this feature host has to set the custom bit 0x400 in opermode command

8.74 TSF Synchronization packet

Description:

This command is used to send a frame for timing synchronization .Thiscommand is valid only if module is connected with the specified peer. Ifthe specified peer is not connected, module will return error code(0x0078) to host.

Command Format in UART mode:

at+rsi_sync_pkt

Usage:

at+rsi_sync_pkt=MAC_address,payload\r\n

Command Parameters:

MAC_address: MAC address of receiver

payload: payload to be sent to the receiver

Response:

Transmitter End:

In response to above command, module returns TSF timer value captured

after sending the packet on air at transmitter end.

OK<TSF Value (8 bytes)>

Example: 4f 4b 06 2b 25 02 00 00 00 00 0d 0a

Receiver End:

On the receiver end, an asynchronous message with senders MAC

address, TSF timer value followed by payload is forward to host.

AT+RSI_TSF_SYNC_PKT_RECVD=<Mac Addr(6bytes)><reserved

(2bytes)><TSF Value(8bytes) ><payload>

Command Format in SPI mode:

Description

Host will issue the command for sending TSF synchronization packet with frame type (0xC0). At reception of this command, Module will prepare a management frame (Action frame) attaching the payload. The module then captures the timestamp after sending the packet on air and sends back the response frame to host. On the receiver side, the module filters the packet based on Action frame type, attaches the timestamp and forwards the asynchronous packet to the host.

Request Frame Type:

RSI_REQ_HOST_TSF_SYNC_PKT 0xC0

Request Frame Body Structures:

```
#define MAX_PAYLOAD_LENGTH 100
typedef struct rsi_req_tsf_sync_pkt_s
{
uint8 MAC_address[6];
uint8 reserved[22]
uint8 payload[MAX_PAYLOAD_LENGTH];
}rsi_req_tsf_sync_pkt_t;
```

Response Frame Type:

RSI_RSP_HOST_TSF_SYNC_PKT

0xC0 (Transmitter side)

RSI_RSP_HOST_TSF_SYNC_PKT_RECVD

0xC1 (Receiver side)

Response Frame Body structure on transmitter side:

typedef struct rsi_rsp_tsf_sync_pkt_s

{

uint32 TSF_value[2];

}rsi_rsp_tsf_sync_pkt_t;

Response Frame Body structure on receiver side:

typedef struct rsi_rsp_tsf_sync_pkt_s

{

uint8 sender_mac_address[6];

uint8 reserved[2];

uint32 Current_TSF_value[2];

uint8 payload[MAX_PAYLOAD_LENGTH];

}rsi_rsp_tsf_sync_pkt_t;

Station connect/disconnect indication in AP mode

Description:

Asynchronous message are used to indicate host in AP mode when the station is connected(frame type 0xC2)/disconnected(frame type 0xC3).

Command Format:

N/A

Response:

AT Mode:

AT+RSI_CLIENT_CONNECTED= < MAC_address >	MAC address of station connected
AT+RSI_CLIENT_DISCONNECTED= < MAC_address >	MAC address of station disconnected

Binary Mode:

typedef struct{

uint8 MAC_address[6];

}

Response Parameters:

MAC_address(6 bytes): MAC address of station connected/disconnected

Relevance:

This command is valid when opermode is 1 or 6.

Possible Error Codes:

N/A

8.75 Station connect/disconnect indication in AP mode

Description:

Asynchronous message are used to indicate host in AP mode when the station is connected(frame type 0xC2)/disconnected(frame type 0xC3).

Command Format:

N/A

Response :

AT Mode:

AT+RSI_CLIENT_STATION_CONNECTED= < MAC_address >	MAC address of station connected
AT+RSI_CLIENT_STATION_DISCONNECTED= < MAC_address >	MAC address of station disconnected

Binary Mode:

```
typedef struct{
      uint8 MAC_address[6];
}
```

Response Parameters:

`MAC_address(6 bytes)`: MAC address of station connected/disconnected

Relevance:

This command is valid when opermode is 1 or 6.

Possible Error Codes:

N/A

8.76 Transparent Mode Command

Description:

This command is used to Enter/Start transparent mode, parameters for transparent mode are to be provided in this command. On reception of this command, module tries to start transparent mode and replies with “AT+RSI_TMODE “ message with status.

Command Format:

AT Mode:

```
at+rsi_ trans_mode_params=<packetization Length>,<Escape
character>,<gap time>,<frame time>,<escape time>,
<IP version>,<socket type>,<local port>,<Destination port>,
<IP Address>,<Max_count>,<Type of service>,<SSL Parameters>,
<SSL Ciphers>\r\n
```

Binary Mode:

N/A

Command Parameters:

`Packetization Length`: Possible values are 10 to 1024

`Escape character`: Any special character.

This is a special character which has significance if module receives 3 consecutive escape characters from host after gap time. This escape sequence will indicate either to force the current data in buffers to frame a network frame or to exit from Transparent mode and moves into command mode.

`Gap Time(in milliseconds)`: Maximum time gap between bytes received from host. Within this time, escape characters are not checked or considered.

Varies from 0 to 65533

`Frame time(in milliseconds)`: varies from 1 to 65534(should be greater then gap time)

`Escape Time(in milliseconds)`: Varies from 2 to 65535(should be greater then frame time)

`IP Version` : Possible values are 4 or 6

4 - IPV4

6 - IPV6

`Socket Type` : Possible values are

0 – TCP

2 – LTCP

4 – LUDP

`Local Port number`: Local port number

`Destination Port number`: Destination port number

`IP Address`: Server IP address if socket type is LUDP/LTCP

`Max_count` : Maximum no. of clients in LUDP/LTCP, fixed to 1 in transparent mode.

`Type of Service`: Type of service , varies from 0 to 8

`SSL parameters`: This field is used to enable SSL for selected socket.

Possible values:

0 – To open TCP socket.

1 - To open SSL client socket.

5 - To open SSL socket with TLS 1.0 version.

9 - To open SSL socket with TLS 1.2 version.

`ssl_ciphers` : to select various cipher modes, possible values

1- TLS_RSA_WITH_AES_256_CBC_SHA256

2 - TLS_RSA_WITH_AES_128_CBC_SHA256

4 - TLS_RSA_WITH_AES_256_CBC_SHA

8 - TLS_RSA_WITH_AES_128_CBC_SHA

16 - TLS_RSA_WITH_AES_128_CCM_8

32 - TLS_RSA_WITH_AES_256_CCM_8

Response:

AT Mode:

Result Code	Description
`AT+RSI_TMODE0`	Successfully entered into transparent mode
`AT+RSI_TMODE1`	Graceful exit from Transparent mode (by giving escape sequence from host after escape time)
`AT+RSI_TMODE2`	Exited from transparent mode due to WiFi Disconnected.
`AT+RSI_TMODE3`	Exited from transparent mode due to TCP Remote terminate from Peer.
`AT+RSI_TMODE4`	Exited from transparent mode due to TCP retries

Result Code	Description
	over terminated TCP connection.
AT+RSI_TMODE5	Did not enter transparent mode due to invalid transparent mode params .
AT+RSI_TMODE6	Did not enter transparent mode due to module doesnot have IP
AT+RSI_TMODE7	Did not enter transparent mode as could not create requested socket.
Error Codes	Failure. Possible error codes for this command are 0xFF87, 0x0021,0x0025,0x002C,0xFFF8

Binary Mode:

N/A

Relevance:

This command can be given only after successful connection with AP, module should have a valid IP and there should be no prior sockets opened.

Note:

This command is only valid in AT mode using UART interface.

8.77 UART Hardware Flow control

Description:

This command is used to Enable/Disable the Hard ware flow control feature.

This command is valid only in case of UART host interface.

Command Format:

AT Mode:

```
at+rsi_uart_hwflowctrl=<uart_hw_flowcontrol_enable>\r\n
```

Binary Mode:

```
struct {
```

```
        uint8 uart_hw_flowcontrol_enable;
}HwFlowControlEnableFrameSnd;
```

Command Parameters:

uart_hw_flowcontrol_enable : Enable or Disable UART hardware flow control.

1 – Enable

0 - Disable

Response:

AT Mode:

OK	Successful execution
ERROR	Failure.

Binary Mode:

N/A

Response Parameters:

N/A

Possible Error Codes:

Possible error codes for this command are 0x004E,0x002C.

Note:

1) Hardware flow control must be enabled in the module before enabling it in the host
2) This command can be given as the first command after card ready given to host.

8.78 Socket Configuration Parameters

Description:

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

Command Format:

AT Mode:

```
at+rsi_socket_config=<total_sockets>,<total_tcp_sockets>,<tota
l_udp_sockets>,<total_tcp_tx_only_sockets>,<total_tcp_rx_only_
sockets>,<total_udp_tx_only_sockets>,<total_udp_rx_only_socket
s>,<total_tcp_rx_high_performance_sockets>\r\n
```

Binary Mode:

```
typedef struct{
```

```
        uint8 total_sockets;
        uint8 total_tcp_sockets;
        uint8 total_udp_sockets;
        uint8 total_tcp_tx_only_sockets;
        uint8 total_tcp_rx_only_sockets;
        uint8 total_udp_tx_only_sockets;
        uint8 total_udp_rx_only_sockets;
        uint8 total_tcp_rx_high_performance_sockets;
}
```

Command Parameters:

`total_sockets`: Desired total number of sockets to open.

`total_tcp_sockets`: Desired total number of TCP sockets to open.

`total_udp_sockets`: Desired total number of UDP sockets to open.

`total_tcp_tx_only_sockets`: Desired total number of TCP sockets to open which are used only for data transmission.

`total_tcp_rx_only_sockets`: Desired total number of TCP sockets to open which are used only for data reception.

`total_udp_tx_only_sockets`: Desired total number of UDP sockets to open which are used only for data transmission.

`total_udp_rx_only_sockets`: Desired total number of UDP sockets to open which are used only for data reception.

`total_tcp_rx_high_performance_sockets` : Desired total number of high performance TCP sockets to open. High performance sockets can be allocated with more buffers based on the buffers availability. This option is valid only for TCP data receive sockets. Socket can be opened as high performance by setting high performance bit in socket create command.

Following conditions has to be met:

1) total_sockets <= Maximum allowed sockets(10)
2) (total_tcp_sockets + total_udp_sockets) <= total_sockets
3) (total_tcp_tx_only_sockets + total_tcp_rx_only_sockets) <= total_tcp_sockets
4) (total_udp_tx_only_sockets + total_udp_rx_only_sockets) <= total_udp_sockets
5) total_tcp_rx_high_performance_sockets <= total_tcp_rx_only_sockets

Response:

AT Mode:

OK	Successful execution
ERROR	Failure.

Binary Mode:

N/A

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible error codes for this command are x0021,0x0025,0x002C,0xFF6D.

Example

AT Mode:

```
at+rsi_socket_config=4,2,2,1,1,1,1,1,1\r\n
```

8.79 RF Current mode Configuration

Description:

This command is used to configure modules RF in different current/power consumption modes. This command should be given only before init command.

Command Format:

AT Mode:

```
at+rsi_rf_current_mode=< rf_rx_curr_mode>,< rf_tx_curr_mode>,< rf_tx_dbm>\r\n
```

Binary Mode:

```
typedef struct rf_current_config_s {
      uint8 rf_rx_curr_mode;
      uint8 rf_tx_curr_mode;
      int16 rf_tx_dbm;
} rsi_rf_current_config_t;
```

Command Parameters:

`rf_rx_curr_mode`: Current/Power Mode in which modules RF-Receive(RX) should be programmed.

0 – High Current/Power mode

1 – Medium Current/Power mode

2 – Low Current/Power mode

`rf_tx_curr_mode`: Current/Power Mode in which modules RF-Transmit(TX) should be programmed.

0 – High Current/Power mode

1 – Medium Current/Power mode

2 – Low Current/Power mode

`rf_tx_dbm`: This two bytes are reserved. Set to '0'.

Response:

AT Mode:

OK	Successful execution
ERROR	Failure.

Binary Mode:

N/A

Response Parameters:

N/A

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0x0051.

Example:

AT Mode:

at+rsi_rf_current_mode=0,0,0\r\n : Program Modules RF TX and RX in high power mode

at+rsi_rf_current_mode=1,1,0\r\n : Program Modules RF TX and RX in medium power mode

at+rsi_rf_current_mode=2,1,0\r\n : Program Modules RF-TX in medium power mode and RF-RX in low power mode.

8.80 Trigger Auto Configuration

Description:

This command is used to trigger the Stored Auto Configuration.This command should be given only after Card Ready response.

Command Format:

AT Mode:

```
at+rsi_trigger_auto_config\r\n
```

Binary Mode:

No payload Required.

Response:

AT Mode:

None	Successful execution
ERROR	Failure.

Binary Mode:

N/A

Response Parameters:

N/A

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible Error codes are 0x0021,0xFF36,0xFF74, 0xFF35

Note:

1) To avail this feature(Wait On Host) user need to set BIT(20) in Custom feature bit map in opermode command.
2) This feature is valid Only when store configuration feature is enabled.
3) Binary Mode: If the last byte of the CARD READY is set to '1' it indicates store configuration is enabled. Now host can choose to either start the auto join by giving this command or stop the auto join and continue with the opermode command.
4) AT mode: If this feature is enabled, after "Loading Done" message "AT+RSI_TRIGGER_AUTO_JOIN " will come, so that user can give either at+rsi_trigger_auto_join command to trigger the auto configuration, (or) user can continue with the Opermode command

8.81 Http Abort

Description:

This command is used to abort the HTTP/HTTPS GET/POST

Request .

This command should be given only after Ipconf command.

Command Format:

AT Mode:

```
at+rsi_http_abort\r\n
```

Binary Mode:

No payload Required.

Response:

AT Mode:

OK	Successful execution
ERROR	Failure.

Bianry Mode:

N/A

Response Parameters:

N/A

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x0025, 0x002C.

8.82 HTTP Server Credentials From Host

Description:

This command is used to set the HTTP ServerCredentials.

Request .

This command should be given only after Opermode command.

Command Format:

AT Mode:

```
at+rsi_credentials=<username>,<password>\r\n
```

Binary Mode:

#define MAX_USERNAME_LEN 31 (Including NULL character)

#define MAX_PASSWORD_LEN 31 (Including NULL character)

```
struct {
        uint8    username[MAX_USERNAME_LEN];
        uint8    password[MAX_PASSWORD_LEN];
```

```
} httpCredentialsFrameSnd;
```

`username`**: Username for HTTP Server**

`password`**: Password for HTTP Server**

Response:

AT Mode:

OK	Successful execution
ERROR	Failure.

Bianry Mode:

N/A

Response Parameters:

N/A

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x0025, 0x00F1,0x0015.

8.83 FTP client

Description:

This section explains different commands to use FTP client.

This command should be given only after Set IP Parameters command.

Following table explains list of ftp commands and their description.

FTP Command	Description
Create	Creates FTP objects. This should be the first command for accessing FTP.
Connect	Connects to FTP server.
Make Directory	Creates directory in a specified path.
Delete Directory	Deletes directory in a specified path
Change Working Directory	Changes working directory to a specified path.
Directory List	Lists directory contents in a specified path.

File Read	Reads the file
File Write	Open file to write
File Write Content	Writes content into file which is opened using File Write command. File content can be written in multiple chunks using this command.
File Delete	Deleted file
File Rename	Renames file
Disconnect	Disconnects from FTP server. Once disconnect is done user can connect again using connect command.
Destroy	Destroys FTP objects. Once destroy is given user cant use FTP unless it is created again.
Mode set	Used to set the FTP client mode , either in Passive mode or Active Mode

- **Create** should be called as a first command to use FTP.
- Once create is successful **connect** should be called to connect to a FTP server.
- After connection is successful host can issue remaining commands.
- After FTP operations host has to give **disconnect** command to disconnect from FTP server.
- Once disconnect is done, host can again connect to the FTP server using **connect** command.
- To destroy FTP objects host has to give **destroy** command. Once destroy is given user cant use FTP unless it is created again using **create** command.
- FTP mode set command should be called after successful connection with the FTP server.

Note:

By default RS9113 module's FTP client will be in Active Mode .

Command Format:

AT Mode:

FTP client has different command types. Based on the command type next parameters will change.

```
at+rsi_ftp=<command_type>,<remaining parameters>\r\n
```

Following are available command types.

FTP command	Command Type	Command Format
Create	**1**	`at+rsi_ftp=1\r\n`

Connect	**2**	`at+rsi_ftp=2,<ip_version>,`
		`<IPaddress>,<username>,<password>,<server_port>\r\n`
		`ip_version`: IP version to use.
		4 – For IPv4
		6 – For IPv6
		`IP address`: IPv4/IPv6 address for FTP server to connect.
		`username`: username of FTP server
		`password`: password of FTP server
		`server_port`: FTP server port number

Make Directory	**3**	`at+rsi_ftp=3,<Directory_path>\r\n` `Directory_path`: Path of the directory to make.
Delete Directory	**4**	`at+rsi_ftp=4,<Directory_path>\r\n` `Directory_path`: Path of the directory to delete.
Change Working Directory	**5**	`at+rsi_ftp=5,<Directory_path>\r\n` `Directory_path`: Change of directory path.
Directory List	**6**	`at+rsi_ftp=6,<Directory_path>\r\n` `Directory_path`: path of the directory for list.
File Read	**7**	`at+rsi_ftp=7,<file_name>\r\n` `file_name`: name of the file to read.
File Write	**8**	`at+rsi_ftp=8,<file_name>\r\n` `file_name`: Name of the file to write.
File Write content	**9**	`at+rsi_ftp_file_content=<end_of_file>,<file_content>\r\n`

		end_of_file: Represents whether end of file is reached or not. 0 – More data is coming to write into file. 1 – Current chunk is the last chunk and no more data is coming. File_content: Content of the file to write.
File Delete	**10**	at+rsi_ftp=10,<file_name>\r\n file_name: Name of the file to delete.
File Rename	**11**	at+rsi_ftp=11,<file_name>,<new_file_name>\r\n file_name: Old name of the file. new_file_name: New file name.
Disconnect	**12**	at+rsi_ftp=12\r\n
Destroy	**13**	at+rsi_ftp=13\r\n
Passive mode	**14**	at+rsi_ftp=14\r\n
Active mode	**15**	at+rsi_ftp=15\r\n

Binary Mode:

```
#define FTP_USERNAME_LENGTH  31
#define FTP_PASSWORD_LENGTH 31
#define FTP_PATH_LENGTH 51
#define FTP_MAX_CHUNK_LENGTH 1400

typedef struct ftp_connect
{
  //! FTP client IP version
  uint8 ip_version;
  union
  {
    //! IPv4 address
    UINT8  ipv4_address[4];
    //! IPv6 address
    UINT8  ipv6_address[16];
  } server_ip_address;
```

```
  //! FTP client username
  uint8 username[FTP_USERNAME_LENGTH];

  //! FTP client password
  uint8 password[FTP_PASSWORD_LENGTH];

  //! FTP server port number
  uint8 server_port[4];
} ftp_connect_t;

typedef struct ftp_command
{
  //! Directory or file path
  uint8 path[FTP_PATH_LENGTH];
  //! New file name
  uint8 new_file_name[FTP_PATH_LENGTH];
} ftp_command_t;

typedef struct
{
  //! FTP command type
  uint8              command_type;
  union
  {
    //! structure for FTP connect
    ftp_connect_t    ftp_connect;

    //! Structure for other commands
    ftp_command_t    ftp_command;

  }ftp_client_struct;

}rsi_ftp_client_t;

typedef struct ftp_file_write
```

```
{
  //! command type
  uint8 command_type;
  //! End of file
  uint8 end_of_file;

  //! Path of file to write
  uint8 file_content[FTP_MAX_CHUNK_LENGTH];

} rsi_ftp_file_write_t;
```

command_type: Type of the FTP command. This parameter is valid for all commands.

Ip_version: IP version to use. This parameter is valid for **connect** command.

4 – For IPv4

6 – For IPv6

server_ip_address.ipv4_address: IPv4 address of the FTP server. This parameter is valid for **connect** command.

server_ip_address.ipv6_address: IPv6 address of the FTP server. This parameter is valid for **connect** command.

username: username for the FTP server. This parameter is valid for **connect** command.

password: Password for the FTP server. This parameter is valid for **connect** command.

server_port: FTP server port number. This parameter is valid for **connect** command.

path: path of the directory/file. This parameter is valid for **Make Directory**, **Delete Directory**, **Change Working Directory**, **Directory List**, **File Read**, **File Write**, **File rename**, **File Delete** commands.

new_file_name: New file name. This parameter is valid for **File Rename** command.

end_of_file: Represents whether end of file is reached or not. This parameter is valid for **File Write Content** command.

0 – More data is coming to write into file.

1 – Current chunk is the last chunk and no more data is coming.

file_content: Content of the file to write. This parameter is valid for **File Write Content** command.

Response:

AT Mode:

FTP command	Command Response

Directory List	AT+RSI_FTP_DIR_LIST=<command_type><more><length><data>\r\n Command_type: 1 byte. This filed contains value '6'. More: 1 byte. Represents whether more response is pending from module or not. 1 – More response is pending 0 – End of response Length: 2 bytes. Length of current chunk response Data: variable bytes. Content of the directory list
File Read	AT+RSI_FTP_FILE=<command_type><more><length><data>\r\n Command_type: 1 byte. This filed contains value '7'. More: 1 byte. Represents whether more response is pending from module or not. 1 – More response is pending 0 – End of response Length: 2 bytes. Length of current chunk response Data: Variable bytes. Content of the file
For all other commands	OK<command_type>\r\n Command_type: 1 byte. Type of the FTP command.

Binary Mode:

```
typedef struct ftp_rsp_t
{
  uint8 command_type;
  uint8 more;
  uint16 length;
  uint8 data[1024];
}rsi_ftp_rsp_t;
```

Response Parameters:

Command_type: Type of the FTP command.

More: Represents whether more response is pending from module or not.

1 – More response is pending

0 – End of response

Length: Length of current chunk response

`Data`: Response data

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x0015, 0xFF6B, 0xBB01, 0xBB50, 0xBBD3, 0xBBD4, 0xBBD5, 0xBBD6, 0xBBD9, 0xBBDA, 0xBBDB, 0xBBDC, 0xBBDD, 0xBBDE.

Example:

AT Mode:

1.FTP File Read

```
at+rsi_ftp=1\r\n
at+rsi_ftp=2,4,192.168.0.150,admin,test123,201\r\n
at+rsi_ftp=7,file_read1.txt\r\n
at+rsi_ftp=12\r\n
at+rsi_ftp=2,4,192.168.0.150,admin,test123,201\r\n
at+rsi_ftp=7,file_read2.txt\r\n
at+rsi_ftp=12\r\n
at+rsi_ftp=13\r\n
```

2.FTP File Write

```
at+rsi_ftp=1\r\n
at+rsi_ftp=2,4,192.168.0.150,admin,test123,201\r\n
at+rsi_ftp=8,file_write1.txt\r\n
at+rsi_ftp_file_content=0,This is start of sample data\r\n
at+rsi_ftp_file_content=1,This is end of sample data\r\n
at+rsi_ftp=12\r\n
at+rsi_ftp=13\r\n
```

1.FTP Client Passive

```
at+rsi_ftp=1\r\n
at+rsi_ftp=2,4,192.168.0.150,admin,test123,201\r\n
at+rsi_ftp=14\r\n
at+rsi_ftp=7,file_read1.txt\r\n
at+rsi_ftp=12\r\n
at+rsi_ftp=2,4,192.168.0.150,admin,test123,201\r\n
```

```
at+rsi_ftp=7,file_read2.txt\r\n
at+rsi_ftp=12\r\n
at+rsi_ftp=13\r\n
```

8.84 SNTP Client

Description:

This section explains different commands to use SNTP client.

This command should be given only after Set IP Parameters command.

Following table explains list of sntp commands and their description.

SNTP Command	Description
Create	Creates SNTP objects. This should be the first command To get time updates from the SNTP server
Get Time	To Get the Current time in seconds
Get Time-Date	To Get the Current time in Time-Date format
Get Server Address	To Get the SNTP server Details.
Get Server Info	To Get the SNTP server Details.
Delete	To Delete the SNTP client.

- **Create** should be called as a first command to use SNTP.
- Once create is successful **Get Server Address** should be called to get details of the SNTP server.
- Call **Get Time** to get the time in seconds from SNTP server.
- Call **Get Time Date** to get the Time Date format from SNTP server.

Command Format:

AT Mode:

SNTP client has different command types. Based on the command type next parameters will change.

at+rsi_sntp=1,<ip_version>,<IPaddress><sntp method>,<sntp timeout>\r\n

Following are available command types.

SNTP command	Command Type	Command Format
Create	**1**	at+rsi_sntp=1,<ip_version>,<IPaddress><sntp method>,<sntp timeout>\r\n

		ip_version: IP version to use. 4 – For IPv4 6 – For IPv6 IP address: IPv4/IPv6 address for SNTP server to connect. sntp method: **SNTP method to use.** 1 – For BroadCast Method 2 – For UniCast Method
Get Time	**2**	at+rsi_sntp=2\r\n
Get Time Date	**3**	at+rsi_sntp=3\r\n
Get Server Address	**4**	at+rsi_sntp=4\r\n
Delete	**5**	at+rsi_sntp=5\r\n
Get Server info	**6**	at+rsi_sntp=6\r\n

Binary Mode:

```
#define SNTP_BROADCAST_MODE      1
#define SNTP_UNICAST_MODE        2
typedef struct
{
  UINT8  command_type;
  UINT8  ip_version;
  union
  {
    UINT8  ipv4_address[4];
    UINT8  ipv6_address[16];
  }server_ip_address;

   UINT8  sntp_method;
   UINT8  sntp_timeout[2];
}rsi_sntp_client_t
```

command_type: Type of the SNTP command. This parameter is valid for all commands.

Ip_version: IP version to use. This parameter is valid for **create** command.

4 – For IPv4

6 – For IPv6

server_ip_address.ipv4_address: IPv4 address of the SNTP server. This parameter is valid for **create** command.

server_ip_address.ipv6_address: IPv6 address of the SNTP server. This parameter is valid for **create** command.

sntp_method: Mode of the SNTP client to run

- 1 – For Broadcast
- 2 – For Unicast

Response:

AT Mode:

SNTP command	Command Response
Create	Ok<1>\r\n
Get Time	OK<2><Time in seconds>\r\n
Get Time Date	OK<3><Time in Ddat-Time format>\r\n
Get Server Address	OK<Ip version><Ip address><sntp method>\r\n Ip_version: Ip version of the SNTP server. Ip_address: Ip address of the SNTP server. sntp_method: sntp method of the server.
Get Server Info	OK<6><Ip version><Ip address><sntp method>\r\n Ip_version: Ip version of the SNTP server. Ip_address: Ip address of the SNTP server. sntp_method: sntp method of the server.
Invalid SNTP server response	AT+RSI_INVALID_SNTP_SERVER=<ip_version><ip_address><sntp_method>\r\n Ip_version: Ip version of the SNTP server. Ip_address: Ip address of the SNTP server.

	sntp_method: sntp method of the server.
For remaining commands	OK<Command_type>\r\n Command_type: 1byte. Type of the SNTP command

Binary Mode:

```
typedef struct rsi_sntp_rsp_t
{
  uint8 command_type;
  uint8 sntp_buffer[50];
}rsi_sntp_rsp_t;

typedef struct rsi_sntp_server_rsp_t
{
  UINT8  ip_version;
  union
  {
    UINT8  ipv4_address[4];
    UINT8  ipv6_address[16];
  }server_ip_address;

  UINT8  sntp_method;
}rsi_sntp_server_rsp_t;

typedef struct rsi_sntp_server_info_rsp_t
{
  UINT8  command_type;
  UINT8  ip_version;
  union
  {
    UINT8  ipv4_address[4];
    UINT8  ipv6_address[16];
  }server_ip_address;

  UINT8  sntp_method;
}rsi_sntp_server_info_rsp_t;
```

Response Parameters:

`Command_type`: Type of the SNTP command.

`ip_version`: IP version of the SNTP server.

`server_ip_address`: IP address of the SNTP server.

`sntp_method`: sntp method of the SNTP server

`sntp_buffer`: sntp_buffer contains time/data string, this parameter is valid only for Get_Time or Get_Time_Data commands.

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x0015, 0x0074,0xBB10.

Example:

AT Mode:

1.SNTP client:

```
at+rsi_sntp=1,4,192.168.0.100,2,5\r\n
at+rsi_sntp=2\r\n
at+rsi_sntp=3\r\n
at+rsi_sntp=4\r\n
at+rsi_sntp=6\r\n
at+rsi_sntp=5\r\n
```

8.85 MDNS and DNS-SD

Description:

This section explains different commands to use MDNS and DNS-SD.

This command should be given only after Set IP Parameters command.

Following table explains list of MDNS and DNS-SD commands and their description.

MDNSD Command	Command Type	Description
Init	1	Creates MDNS Daemon . This should be the first command to initialize.
Register	3	To add a service/start service discovery.

Service		
Deinit	6	To Stop MDNS responder in module

- **Init** should be called as a first command to use MDNS/DNS-SD.
- Once Init is successful, Add a service using Register Service.
- Reset more bit in Register Service command to indicate module to start MDNS/DNS-SD service.
- To stop MDNS/DNS-SD service use Deinit command.

Command Format:

AT Mode:

MDNS/DNS-SD client has different command types. Based on the command type following parameters will change accordingly.

`at+rsi_mdns=<command_type>,<remaining parameters>\r\n`

Following are available command types.

MDNSD command	Command Type	Command Format
Init	**1**	`at+rsi_mdns=1,<ip_version>,<ttl>,<buffer>\r\n` `ip_version`: IP version to use. 4 – For IPv4 6 – For IPv6 `ttl`: Time To Live, Time in seconds for which service should be active. `buffer`: Host name which is to used as host name in Type-A record.
Register Service	**3**	`at+rsi_mdns=3,<port_number>,<ttl>,<more>,<buffer>\r\n` `port_number`: Port number on which service which should be added. `ttl`: Time To Live, Time in seconds for which service should be active. `more`: This byte should be set to '1' when there are more services to add. 0 – This is last service, starts MDNS service. 1 – Still more services will be added.

		`buffer`: This field contains strings separated by null character(ascii : 0x00(Hex)) Buffer contains 3 string fields separated by NULL, fields are i. Name to be added in Type-PTR record ii. Name to be added in Type-SRV record(Service name) iii. Text field to be added in Type-TXT record
Deinit	**6**	`at+rsi_mdns=6\r\n`

Binary Mode:

```
#define MDNS_INIT                       1
#define MDNS_REGISTER_SERVICE           3
#define MDNS_DEINT                      6

typedef struct rsi_mdns_t
{
  uint8    command_type;
  union
  {
    mdns_init_t      mdns_init;
    mdns_reg_srv_t   mdns_reg_srv;
  } mdns_struct;
  uint8    buffer[1000];
} rsi_mdns_t;

typedef struct
{
  uint8      ip_version;
  uint8      ttl[2];
} mdns_init_t;

typedef struct
{
  uint8      port[2];
```

```
  uint8       ttl[2];
  uint8       more;
} mdns_reg_srv_t;
```

command_type: Type of the MDNSD command. This parameter is valid for all commands.

ip_version: IP version to use. This parameter is valid for **Init** command.

4 – For IPv4

6 – For IPv6

ttl: Time To Live, this field is valid for Init command(type - 1), register service command(type - 3).

more: This field is only valid for Register service command.

Response:

AT Mode:

MDNS command	Command Response
Any MDNS Command	OK<Command Type>\r\n

Binary Mode:

```
typedef struct mdns_rsp_t
{
  uint8 command_type;
}rsi_mdns_rsp_t;
```

Response Parameters:

Command_type: Type of the MDNSD command.

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x0015, 0x0074,0xFF2B.

Example:

AT Mode:

MDNSD Add Service:

```
at+rsi_mdns=1,4,600,http-wsc_obe.local.\r\n
at+rsi_mdns=3,80,600,0,_http._tcp.local.NULL
wsc_obe._http._tcp.localNULLtext_field\r\n
at+rsi_mdns=6\r\n
```

1. Currently registering only one service is supported
2. IPv4 is only supported for MDNS/DNS-SD service
3. NULL – Is a NULL character with ASCII value '0x00'(HEX)
4. If module joins multicast group,MDNS is not supported

8.86 SMTP Client

Description:

This section explains different commands to use SMTP client.

This command should be given only after Set IP Parameters command.

Following table explains list of smtp commands and their description.

SMTP Command	Description
Create	Creates SMTP related thread, This should be the first command to use the SMTP client
Init	To initialize the SMTP client with username, password , from mail address and domain name
Send	To Send the mail the to recepient
Deinit	To Delete the SMTP client

- **Create** should be called as a first command to use SMTP.
- Once create is successful **Init** should be called to initialize the SMTP client with username, password, from address and domain name of the SNMP agent.
- Call **Send** to send the SMTP client mail to the SMTP server agent.
- Call **Deinit** to delete the SMTP client.

Command Format:

AT Mode:

SMTP client has different command types. Based on the command type following parameters will change accordingly.

```
at+rsi_smtp=<command_type>,<remaining parameters>\r\n
```

Following are available command types.

SMTP command	Command Type	Command Format
Create	**1**	`at+rsi_smtp=1\r\n` Creates the SMTP client
Init	**2**	`at+rsi_smtp=2,<ip_version>,<ip address>,<auth_type>,<server port>,<smtp buffer>\r\n` `ip_version`: IP version to use. 4 – For IPv4 6 – For IPv6 `ip_address`: Ipv4/Ipv6 address of the SMTP server agent. `auth_type`: Authentication type of the SMTP server. 1-For Auth-Login type 3-For Auth_plain type 5-For Auth_login type with SSL 7-For Auth_plain type with SSL `server_port`: SMTP server agent port number `smtp_buffer`:smtp buffer contains server's username, password, from mail address, smtp server local domain name.
Mail send	**3**	`at+rsi_smtp=3,<smtp_feature>,<mail body_length>,<smtp buffer>\r\n` `smtp_feature`: smtp feature bitmap BIT(0): Low priority mail BIT(1): Normal priority mail BIT(2): High priority mail BIT(3): Smtp extended header feature `smtp_mail_body_length`: Length of the mail body

		smtp_buffer: smtp buffer contains recipient mail address, mail subject line, mail body, smtp extended header.
Deinit	**4**	at+rsi_smtp=4\r\n To delete the smtp client

Binary Mode:

```
#define  RSI_SMTP_BUFFER_LENGTH  1024


typedef struct
{
  uint8 ip_version;
  union
  {
    uint8  ipv4_address[4];
    uint8  ipv6_address[16];
  } server_ip_address;
  uint8  auth_type;
  uint8 server_port[4];
}  smtp_client_init_t;

typedef struct
{
  uint8 smtp_feature;
  uint8 smtp_client_mail_body_length[2];
} smtp_mail_send_t;


typedef struct
{
  uint8 command_type;
  union
  {
    smtp_client_init_t  smtp_client_init;
    smtp_mail_send_t    smtp_mail_send;
```

```
  } smtp_struct;
  uint8  smtp_buffer[RSI_SMTP_BUFFER_LENGTH];
} rsi_smtp_client_t;
```

command_type: Type of the SMTP command. This parameter is valid for all commands.

ip_version: IP version to use. This parameter is valid for **Init** command.

4 – For IPv4

6 – For IPv6

ipv4_address: Ipv4 address of the SMTP agent.

ipv6_address: Ipv6 address of the SMTP agent.

auth_type: Authentication method used by the SMTP server agent.

1 for AUTH_LOGIN

3 for AUTH_PLAIN

5 for AUTH_LOGIN with SSL

7 for AUTH_PLAIN with SSL

server_port: SMTP server agent port number.

smtp_feature: SMTP client feature bit map

BIT(0): MAIL_PRIORITY_LOW

BIT(1): MAIL_PRIORITY_NORMAL

BIT(2): MAIL_PRIORITY_HIGH

BIT(3): To Enable SMTP_EXTENDED_HEADER feature

smtp_client_mail_body_length: Length of the SMTP client mail body.

smtp_buffer: smtp_buffer contains remaining parameters based on command type.

Note:

1. Maximum supported length for username, password, domain name, recipient mail address, from mail address is 100 bytes each excluding NULL character
2. Maximum supported length for recipient mail address, mail subject line, mail body together is 1024 bytes including NULL characters(3 bytes)
3. Maximum supported length for mail subject line is 750 bytes
4. Maximum supported length for mail body is 950 bytes

5. In the SMTP client command, there is an SMTP buffer parameter which uses NULL characters to separate the values. In such cases, you cannot use NULL characters as part of the value itself and also comma values since these are also separators.
6. In opermode command host need to set SMTP client and SSL feature bit maps in tcp/ip feature bitmap to support SMTP client over SSL .

Response Parameters:

`Command_type`: Type of the SMTP command.

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x000e, 0xBBAA,0xBBAD,0x002C,0x0015,0xBBA5,0xBB21,0x003E,0xBBB2,0x003E,0xBBA5,0xBBA3,

0xBA0,0xBBA1,0xBBA2,OxBBA4,OxBBA6,0xBBA7,0xBBA8,0xBBA9,0xBBAA,0xBBAB,0xBBAC,

0xBBAD,0xBBAE,0xBBAF,0xBBB0,0xBBB1,0x0025,0xFF74,0xBBF0.

Example:

AT Mode:

SMTP client mail send Service:

```
at+rsi_smtp=1\r\n
```

```
at+rsi_smtp=2,4,192.168.0.100,1,25, usernameNULLpasswordNULL
redpine1@redpinesignals.comNULLmail.redpinesignals.comNULL\r\n
```

```
at+rsi_smtp=3,3,4,redpine2@redpinesignals.comNULLsubjectlineNU
LLbody\r\n
```

```
at+rsi_smtp=4\r\n
```

8.87 POP3Client

Description:

This section explains different commands to use POP3 client.

This command should be given only after Set IP Parameters command.

Following table explains list of pop3 commands and their description.

POP3 Command	Description
Session Create	Creates POP3 client Daemon , this should be the first command to use the POP3 client. This command initiates POP3 client to

	establish a connection with POP3 server and then gets authenticated to it.
Get mail stats	To get the total number of mails count and size
Get mail list	To get the size of the mail for the passed index
Retrieve mail	To retrieve the mail content for the passed mail index
Mark mail	To mark a mail as deleted for the passed mail index
Unmark mail	To unmark(reset) all the marked(deleted) mails in the current session.
get server status	To get the pop3 server status
session delete	To delete pop3 client session

- **Create** should be called as a first command to use POP3.
- Once create is successful call **get mail** stats to get the mail statistics i.e number ofmails present in server and the size of total mails
- Call get **mail list** to get the size of the mail for the passed index
- Call **Retrieve mail** to get the mail content for the passed mail index
- **Call Mark mail** to delete particular mail in the POP3 server, by passing the mail index
- Call **Unmark mail** to reset all the mails in the current session
- Call **get server status** to get the pop3 server status
- Call **session delete** to delete the pop3 client session

Note:

Retrieve mail command should be issued only after get mail list command, Providing index of the intended mail

Note:

Maximum allowed username and password length is 101 (Including NULL character)

Command Format:

AT Mode:

POP3 client has different command types. Based on the command type following parameters will change accordingly.

at+rsi_pop3=<command_type>,<remaining parameters>\r\n

Following are available command types.

POP3 command	Command Type	Command Format
Session Create	**1**	at+rsi_pop3=1,<ip_version>,>,<ipaddress>,<serverport,<auth_type>,<username>,<password>\r\n ip_version: IP version to use. 4 – For IPv4 6 – For IPv6 ip_address: Ipv4/Ipv6 address of the POP3 server. server_port: POP3 server port number auth_type: Authentication type of the pop3 server(reserved) username: username for POP3 server authentication. password: password for POP3 server authentication.
Mail stats	**2**	at+rsi_pop3=2\r\n To get the mail stats from the POP3 server.
Mail list	**3**	at+rsi_pop3=3,<mail index>\r\n mail index : Index of the particular mail which is used to get the size of the mail.
Retrieve mail	**4**	at+rsi_pop3=4,<mail index>\r\n mail index : Index of the particular mail which is used in the mail list command.
Mark mail	**5**	at+rsi_pop3=5,<mail index>\r\n mail index : Index of the particular mail which is used for deletion by passing the index
Unmark mail	**6**	at+rsi_pop3=6\r\n To reset all the marked mails in

		the current session
server status	**7**	at+rsi_pop3=7\r\n To get the POP3 server status
session delete	**8**	at+rsi_pop3=8\r\n To delete the POP3 client session

Binary Mode:

```
#define POP3_CLIENT_MAX_USERNAME_LENGTH          101
#define POP3_CLIENT_MAX_PASSWORD_LENGTH          101

//! POP3 client session create structure
typedef struct  rsi_pop3_client_session_create
{
    //! POP3 server ip version
  uint8_t ip_version;
  union
  {
    //! Server ipv4 address
    uint8_t  ipv4_address[4];
    //! Server ipv6 address
    uint8_t  ipv6_address[16];
  } server_ip_address;

  //! POP3 server port number
  uint8_t server_port_number[2];
  //! POP3 client authentication type
  uint8_t auth_type;
  //! POP3 client username
  uint8_t username[POP3_CLIENT_MAX_USERNAME_LENGTH];
  //! POP3 client password
  uint8_t password[POP3_CLIENT_MAX_PASSWORD_LENGTH];

} rsi_pop3_client_session_create_t;

//!POP3 client request structure
```

```
typedef struct rsi_req_pop3_client_s
{
  //! POP3 client command type
  uint8_t command_type;
  //! POP3 client command structure
  union
  {
    //! POP3 client session create structure
    rsi_pop3_client_session_create_t
pop3_client_session_create;
    //! POP3 client mail index
    uint8_t     pop3_client_mail_index[2];
  } pop3_struct;
} rsi_req_pop3_client_t;
```

command_type: Type of the POP3 command. This parameter is valid for all commands.

ip_version: IP version to use. This parameter is valid for **session create** command.

4 – For IPv4

6 – For IPv6

ipv4_address: Ipv4 address of the POP3 server.

ipv6_address: Ipv6 address of the POP3 server.

auth_type:Authentication method used by the POP3 server agent

(reserved)

Note:

auth_type field is reserved.

server_port_number: POP3 server agent port number.

username: username for POP3 server authentication

password: password for POP3 server authentication

pop3_client_mail_index: POP3 mail index number

Response Parameters:

```
//! POP3 client response structure
typedef struct rsi_pop3_client_resp_s
{
```

```
  uint8_t  command_type;
  //! Total number of mails
  uint8_t  mail_count[2];
  //! Total size of all the mails
  uint8_t  size[4];
} rsi_pop3_client_resp_t;

//! POP3 client mail data response structure
typedef struct rsi_pop3_mail_data_resp_s
{
  //! Type of the POP3 client command
  uint8_t command_type;
  //! More data pending flag
  uint8_t more;
  //! Length the mail chunk
  uint8_t length[2];
  //! Mail content buffer
  uint8_t data[1000];
}
```

Command_type: Type of the POP3 command.

mail_count: Total mails count/ mail index

size: Total size of the mails/ size of the particular mail

More(4 bytes): This indicates whether more POP3 data for the retrieve command is pending or not.

0–More data is pending. Further interrupts may be raised by the module till all the data is transferred to the Host.

1– End of POP3 mail content.

length: Length of the current pop3 mail content chunk

data: POP3 client mail content buffer

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x0015, 0x0025,0xFF74,0xBB87,0xBBFF,BBC5.

Example:

AT Mode:

```
POP3 client mail receive Service:
at+rsi_pop3=1,4,192.168.0.100,110,0,testuserNULL,test123\r\n
at+rsi_pop3=2\r\n
at+rsi_pop3=3,100\r\n
at+rsi_pop3=4,100\r\n
at+rsi_pop3=5,100\r\n
at+rsi_pop3=6\r\n
at+rsi_pop3=7\r\n
at+rsi_pop3=8\r\n
```

8.88 IAP Init

Description:

This command initializes the interface between RS9113 and IAP(iPod Accessory protocol) co processor before starting the Apple Authentication process.

Command:

AT Mode:

N/A

Binary Mode:

No payload required

Command Parameters:

No parameters

Response:

AT Mode:

N/A

Binary mode:

There is no response payload for this command.

Relevance:

This command is relevant in AP mode and Station mode

Note:
This command is required only if IAP co-processor is mounted on RS9113 chip with I2C interface.This command is not required if IAP chip is mounted on the host MCU.For more information,contact Silicon Labs Signals

8.89 Load MFI IE

Description:

This command is used to load the MFI Information Element in the beacon generated by the WAC(Wireless accessory configuration) server.

Command:

AT Mode:

N/A

Binary Mode:

```
typedef struct rsi_mfi_load_ie_request_s
{
  uint8 ie_len;

  uint8 ie[200];

}rsi_mfi_load_ie_request_t;
```

Command Parameters:

`ie_len`: Length of the MFI information element need to be loaded in the beacon of RS9113 Accessory.

`ie`: array of MFI information element.

Response:

AT Mode:

N/A

Binary mode:

There is no response payload for this command

Relevance:

This command is relevant in AP mode

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x0015, 0x002C.

8.90 Over The Air Firmware Upgradation

Description:

This command upgrades the firmware on the module over a TCP socket. It is designed to work with a remote application.

Command Format:

AT Mode:

```
at+rsi_otaf=<ip_version>, <destIPaddr>, <server_port>,
     <chunk_number>, <rx_timeout> ,<tcp_retry_count>\r\n
```

Command Parameters:

`ip_version`: IP version used, either 4 or 6.

`destIPaddr`: IP Address of the target server.

`server_port`: destination port. Value ranges from 1024 to 49151.

`chunk_number`: chunk number of the firmware to be upgraded. Initially keep it as 1.

`rx_timeout`: To configure timeout.

`tcp_retry_count`: To configure tcp retransmissions count.

Binary Mode:

```
typedef union {
  struct {
    uint8    ip_version;
    union{
      uint8   ipv4_address[RSI_IP_ADD_LEN];
      uint8   ipv6_address[RSI_IP_ADD_LEN * 4];
    }server_address;
    uint8    server_port[4];
    uint8    chunk_number[2];
    uint8    time_out[2];
    uint8    retry_count[2];
  }OtafReqFrameSnd;
  uint8    uOtafReqBuf[27];
}rsi_uOtafReq;
```

Command Parameters:

`ip_version`: IP version used, either 4 or 6.

`destIPaddr`: IP Address of the otaf server.

`server_port`: Destination port of the otaf server.

`chunk_number`: chunk number of the firmware to be upgraded. Initially keep it as 1.

`rx_timeout`: tcp packet receive timeout.

`tcp_retry_count`: tcp retransmissions count.

Response:

After successful up gradation firmware gives a success indication with an asynchronous message as "**AT+RSI_FWUPSUCCESS\r\n**". In addition "**reach end of file**" message comes at server side.

On firmware upgrade failure, firmware gives error message as

"**ERROR<Error_Code><Number of chunk where up gradation stopped**>". User needs to give same command again from the chunk number specified here.

Possible error codes:

0xBB01, 0xBB38

Example:

```
at+rsi_otaf = 4, 192.168.0.100, 5001, 1, 100, 10\r\n
```

Response:

`AT+RSI_FWUPSUCCESS` on success up gradation.

`ERROR<Error_Code>`<number of chunk where up gradation

stopped>

`ERROR<Error_Code = 0x01 0xBB>`<number of chunk where

upgradation stopped = `0xD2 0x04>`

After getting this error give command

```
at+rsi_otaf = 4,192.168.0.100, 5001, 1234, 100, 10\r\n
```

8.91 Read MFI Authentication Certificate

Description:

This command is used to read the IAP co processor Authentication certificate which is used in the authentication process while configuring accessory using MFI WAC server.

Command:

AT Mode:

N/A

Binary Mode:

No request payload required

Response:

AT Mode:

N/A

Binary mode:

There is no response payload for this command

Relevance:

This command is relevant in AP and station mode

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x0015, 0x002C.

Note:

This command is required only if IAP co-processor is mounted on RS9113 chip with I2C interface.This command is not required if IAP chip is mounted on the host MCU. For more information,contact Silicon Labs Signals

8.92 Generate MFI Authentication Signature

Description:

This command is used to generate signature during Accessory configuration using MFI WAC server. The digest given by the MFI WAC server is sent to the IAP co processor to generate signature.

Command:

AT Mode:

N/A

Binary Mode:

```
typedef struct rsi_mfi_auth_create_request_s
{
      uint32 digest_length;
      uint8 digest[40];
}rsi_mfi_auth_create_request_t;
```

Command Parameters:

`digest_length`: Input Digest length from the host.

`digest`: Digest to give to the IAP co processor.

Response:

AT Mode:

N/A

Binary mode:

```
typedef struct rsi_mfi_auth_create_response_s
{
     uint8 signature_length[2];
     uint8 signature[100];
}rsi_mfi_auth_create_response_t;
```

Response Parameters:

`signature_length`: Length of signature generated for the given digest.

`Signature`: Array of signature generated by the IAP co processor

Relevance:

This command is relevant in AP and station mode

Possible Error Codes:

Possible Error codes for this command are 0x0021, 0x0015, 0x002C,0x0055.

Note: This command is required only if IAP co-processor is mounted on RS9113 chip with I2C interface.This command is not required if IAP chip is mounted on the host MCU. For more information,contact Silicon Labs Signals

8.93 Storing Configuration Parameters

In client mode:

The module can connect to a pre-configured access point after it boots up (called auto-join in these sections). This feature facilitates fast connection to a known network.

In Access Point mode:

The module can be configured to come up as an Access Point every time it boots-up (called auto-create in these sections)

The feature is valid in operating modes 0, 2 , 6 and 9.

8.93.1 Storing Configuration Parameters in Client mode

8.93.1.1 Store Configuration in Flash Memory

Description:

This command is used to save the parameters into non-volatile memory which are used either to join to an Access point (auto-join mode) or to create an Access point(auto-create mode) .

Command Format:

AT Mode:

```
at+rsi_cfgsave\r\n
```

Binary Mode:

There is no payload for this command **Response**:

AT Mode:

OK	Successful execution

ERROR	Failure.

Binary Mode:

There is no response payload for this command

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible error codes:

Possible error codes for this command are 0x0021, 0x0025, 0x002C, 0x002F

8.93.1.2 Enable auto-join to AP or Auto-create AP

Description:

This command is used to enable or disable the feature of auto-join or auto-create on power up.

Command Format:

AT Mode:

```
at+rsi_cfgenable=<cfg_enable>\r\n
```

Binary Mode:

```
struct {
      uint8    cfg_enable;
}cfgEnableFrameSnd;
```

Command Parameters:

cfg_enable:

0 - Disables auto-join or auto-create

1 - Enables auto-join or auto-create

Response:

AT Mode:

OK	For response payload parameters description, refer to the section

	8.94Store configuration structure parameters
ERROR	Failure.

Binary Mode:

There is no response payload for this command.

Relavance:

This command is valid when opermode is 0,1,2 or 6.

Possible error codes:

Possible error codes for this command are 0x0021, 0x0025, 0x002C.

8.93.1.3 Get Information about Stored Configuration

Description:

This command is used to get the configuration values that have been stored in the module's memory which are used in auto-join or auto-create modes.

Command Format:

AT Mode:

```
at+rsi_cfgget?
```

Binary Mode:

There is no payload for this command

Response:

AT Mode:

OK<response payload>	For response payload parameters description,referStore configuration structure parameters
ERROR	Failure.

Binary Mode:

```
#define IP_ADDRESS_SZ    4
```

```
#define RSI_SSID_LEN    34
#define WISE_PMK_LEN    32
#define RSI_PSK_LEN     64
#define MAX_HTTP_SERVER_USERNAME 31 (Including NULL Character)
#define MAX_HTTP_SERVER_PASSWORD 31 (Including NULL Character)
typedef struct {
      uint8   channel_no[2];
      uint8   ssid[RSI_SSID_LEN];
      uint8   security_type;
      uint8   encryp_mode;
      uint8   psk[RSI_PSK_LEN];
      uint8   beacon_interval[2];
      uint8   dtim_period[2];
      uint8  ap_keepalive_type;
      uint8  ap_keepalive_period;
      uint8   max_sta_support[2];
}rsi_apconfig;

typedef struct {
      uint8   index[2];
      uint8   key[4][32];
}rsi_wepkey;

typedef union {
      struct {
            uint8   roam_enable[4];
            uint8   roam_threshold[4];
            uint8   roam_hysteresis[4];
      }roamParamsFrameSnd;
      uint8   uRoamParamsBuf[12];
}rsi_uRoamParams;

typedef struct rsi_rejoin_params_s {
      uint8   rsi_max_try[4];
      int8    rsi_scan_interval[4];
      int8    rsi_beacon_missed_count[4];
```

```
	uint8   rsi_first_time_retry_enable[4];
} rsi_rejoin_params_t;

typedef struct {
	uint8    cfg_enable;
	uint8    opermode[4];
	uint8    feature_bit_map[4];
	uint8    tcp_ip_feature_bit_map[4];
	uint8    custom_feature_bit_map[4];
	uint8    band;
	uint8    scan_feature_bitmap;
	uint8    join_ssid[RSI_SSID_LEN];
	uint8    uRate;
	uint8    uTxPower;
	uint8    join_feature_bitmap;
	uint8    reserved_1;
	uint8    scan_ssid_len;
	uint8    reserved_2;
	uint8    csec_mode;
	uint8    psk[RSI_PSK_LEN];
	uint8    scan_ssid[RSI_SSID_LEN];
	uint8    scan_cnum;
	uint8    dhcp_enable;
	uint8    ip[IP_ADDRESS_SZ];
	uint8    sn_mask[IP_ADDRESS_SZ];
	uint8    dgw[IP_ADDRESS_SZ];
	uint8    eap_method[32];
	uint8    inner_method[32];
	uint8    user_identity[64];
	uint8    passwd[128];
	uint8    go_intent[2];
	uint8    device_name[64];
	uint8    operating_channel[2];
	uint8    ssid_postfix[64];
	uint8    psk_key[64];
	uint8    pmk[WISE_PMK_LEN];
```

```
rsi_apconfig apconfig;
uint8    module_mac[6];
uint8    antenna_select[2];
uint8    reserved_3[2];
rsi_wepkey wep_key;
uint8    dhcp6_enable[2];
uint8    prefix_length[2];
uint8    ip6[16];
uint8    dgw6[16];
uint8    tcp_stack_used;
uint8    bgscan_magic_code[2];
uint8    bgscan_enable[2];
uint8    bgscan_threshold[2];
uint8    rssi_tolerance_threshold[2];
uint8    bgscan_periodicity[2];
uint8    active_scan_duration[2];
uint8    passive_scan_duration[2];
uint8    multi_probe;
uint8    chan_bitmap_magic_code[2];
uint8    scan_chan_bitmap_stored_2_4_GHz[4];
uint8    scan_chan_bitmap_stored_5_GHz[4];
uint8    roam_magic_code[2];
rsi_uRoamParams  roam_params_stored;
uint8    rejoin_magic_code[2];
rsi_rejoin_params_t rejoin_param_stored;
uint8     region_request_from_host;
uint8     rsi_region_code_from_host;
uint8     region_code;
uint8     reserved_4[43];
uint8     multicast_magic_code[2];
uint8     multicast_bitmap[2];
uint8     powermode_magic_code[2];
uint8     powermode;
uint8     ulp_mode;
uint8     wmm_ps_magic_code[2];
uint8     wmm_ps_enable;
```

```
uint8     wmm_ps_type;
uint8     wmm_ps_wakeup_interval[4];
uint8     wmm_ps_uapsd_bitmap;
uint8     listen_interval[4];
uint8     listen_interval_dtim;
uint8     ext_custom_feature_bit_map[4];
uint8     private_key_password[82];
uint8     join_bssid[6];
uint8      fast_psp_enable;
uint8      monitor_interval[2];
uint8      timeout_value[2];
uint8      timeout_bitmap[4];
uint8      request_timeout_magic_word[2];
rsi_uHtCaps ht_caps;
uint8      ht_caps_magic_word[2];
//! AP IP parameters in Concurrent mode
UINT8     dhcp_ap_enable;
UINT8     ap_ip[4];
UINT8     ap_sn_mask[4];
UINT8     ap_dgw[4];
uint8     dhcp6_ap_enable[2];
UINT8     ap_prefix_length[2];
UINT8     ap_ip6[16];
UINT8     ap_dgw6[16];
uint8     ext_tcp_ip_feature_bit_map[4];
uint8     http_credentials_avail;
uint8     http_username[MAX_HTTP_SERVER_USERNAME];
uint8     http_password[MAX_HTTP_SERVER_PASSWORD];

}rsi_cfgGetFrameRcv;
```

Note:

Transparent mode parameters are only valid in UART interface in AT command mode only, In binary mode they should be discarded/neglected.

Note:

After firmware upgradation, previous saved configuration may lost due to checksum fail.

Note:

Cfgget command will not store PSK, it will store PMK to reduce connection time with AP from the next boot up. In the cfgget response structure you can find 32 bytes of PMK.

Response Parameters:

For response payload parameters description, refer to the section**8.94Store configuration structure parameters**.

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible error codes:

Possible error codes for this command are 0x0021, 0x0025, 0x002C.

8.93.1.4 Store configuration from User

Description:

This command is used to give the configuration values which are supposed to be stored in the module's non-volatile memory and that are used in auto-join or auto-create modes.

Note:This store configuration command will be supported in sapis

Command Format:

AT Mode:

```
at+rsi_usercfg=<length_of_payload>,<Store configuration
parameters >\r\n
```

Binary Mode:

```
#define IP_ADDRESS_SZ   4
#define RSI_SSID_LEN    34
#define WISE_PMK_LEN    32
#define MAX_HTTP_SERVER_USERNAME 31 (Including NULL Character)
#define MAX_HTTP_SERVER_PASSWORD 31 (Including NULL
Character)typedef struct {
      uint8   channel_no[2];
      uint8   ssid[RSI_SSID_LEN];
      uint8   security_type;
      uint8   encryp_mode;
      uint8   psk[RSI_PSK_LEN];
      uint8   beacon_interval[2];
      uint8   dtim_period[2];
      uint8   ap_keepalive_type;
```

```
	uint8	ap_keepalive_period;
	uint8	max_sta_support[2];
}rsi_apconfig;

typedef struct {
	uint8	index[2];
	uint8	key[4][32];
}rsi_wepkey;

typedef union {
	struct {
		uint8	roam_enable[4];
		uint8	roam_threshold[4];
		uint8	roam_hysteresis[4];
	}roamParamsFrameSnd;
	uint8	uRoamParamsBuf[12];
}rsi_uRoamParams;

typedef struct rsi_rejoin_params_s{
	uint8	rsi_max_try[4];
	int8	rsi_scan_interval[4];
	int8	rsi_beacon_missed_count[4];
	uint8	rsi_first_time_retry_enable[4];
} rsi_rejoin_params_t;

typedef struct {
	uint8	cfg_enable;
	uint8	opermode[4];
	uint8	feature_bit_map[4];
	uint8	tcp_ip_feature_bit_map[4];
	uint8	custom_feature_bit_map[4];
	uint8	band;
	uint8	scan_feature_bitmap;
	uint8	join_ssid[RSI_SSID_LEN];
	uint8	uRate;
	uint8	uTxPower;
```

```
uint8    join_feature_bitmap;
uint8    reserved_1;
uint8    scan_ssid_len;
uint8    reserved_2;
uint8    csec_mode;
uint8    psk[RSI_PSK_LEN];
uint8    scan_ssid[RSI_SSID_LEN];
uint8    scan_cnum;
uint8    dhcp_enable;
uint8    ip[IP_ADDRESS_SZ];
uint8    sn_mask[IP_ADDRESS_SZ];
uint8    dgw[IP_ADDRESS_SZ];
uint8    eap_method[32];
uint8    inner_method[32];
uint8    user_identity[64];
uint8    passwd[128];
uint8    go_intent[2];
uint8    device_name[64];
uint8    operating_channel[2];
uint8    ssid_postfix[64];
uint8    psk_key[64];
uint8    pmk[WISE_PMK_LEN];
rsi_apconfig apconfig;
uint8    module_mac[6];
uint8    antenna_select[2];
uint8    reserved_3[2];
rsi_wepkey wep_key;
uint8    dhcp6_enable[2];
uint8    prefix_length[2];
uint8    ip6[16];
uint8    dgw6[16];
uint8    tcp_stack_used;
uint8    bgscan_magic_code[2];
uint8    bgscan_enable[2];
uint8    bgscan_threshold[2];
uint8    rssi_tolerance_threshold[2];
```

```
uint8    bgscan_periodicity[2];
uint8    active_scan_duration[2];
uint8    passive_scan_duration[2];
uint8    multi_probe;
uint8    chan_bitmap_magic_code[2];
uint8    scan_chan_bitmap_stored_2_4_GHz[4];
uint8    scan_chan_bitmap_stored_5_GHz[4];
uint8    roam_magic_code[2];
rsi_uRoamParams  roam_params_stored;
uint8    rejoin_magic_code[2];
rsi_rejoin_params_t rejoin_param_stored;
uint8     region_request_from_host;
uint8     rsi_region_code_from_host;
uint8     region_code;
uint8     reserved_4[43];
uint8     multicast_magic_code[2];
uint8     multicast_bitmap[2];
uint8     powermode_magic_code[2];
uint8     powermode;
uint8     ulp_mode;
uint8     wmm_ps_magic_code[2];
uint8     wmm_ps_enable;
uint8     wmm_ps_type;
uint8     wmm_ps_wakeup_interval[4];
uint8     wmm_ps_uapsd_bitmap;
uint8     listen_interval[4];
uint8     listen_interval_dtim;
uint8     ext_custom_feature_bit_map[4];
uint8     private_key_password[82];
uint8     join_bssid[6];
uint8      fast_psp_enable;
uint8      monitor_interval[2];
 uint8      timeout_value[2];
 uint8      timeout_bitmap[4];
 uint8      request_timeout_magic_word[2];
 rsi_uHtCaps ht_caps;
```

```
 uint8      ht_caps_magic_word[2];
//! AP IP parameters in Concurrent mode
 UINT8     dhcp_ap_enable;
 UINT8     ap_ip[4];
 UINT8     ap_sn_mask[4];
 UINT8     ap_dgw[4];
 uint8     dhcp6_ap_enable[2];
 UINT8     ap_prefix_length[2;
 UINT8     ap_ip6[16];
 UINT8     ap_dgw6[16];
 uint8     ext_tcp_ip_feature_bit_map[4];
 uint8     http_credentials_avail;
 uint8     http_username[MAX_HTTP_SERVER_USERNAME];
 uint8     http_password[MAX_HTTP_SERVER_PASSWORD];


}rsi_user_store_config_t;
```

Command Parameters:

length_of_payload: Length in bytes of the Store configuration parameters field.

Store configuration parameters: Store configuration parameter in hex format.

For payload parameters description, refer section Store Configuration structure parameters.

Response:

AT Mode:

OK	Successful execution
ERROR	Failure.

Binary Mode:

There is no response payload.

Relevance:

This command is valid when opermode is 0,1,2 or 6.

Possible Error Codes:

Possible error codes for this 0x003D,0x0021,0x002C,0x0025,0x0015.

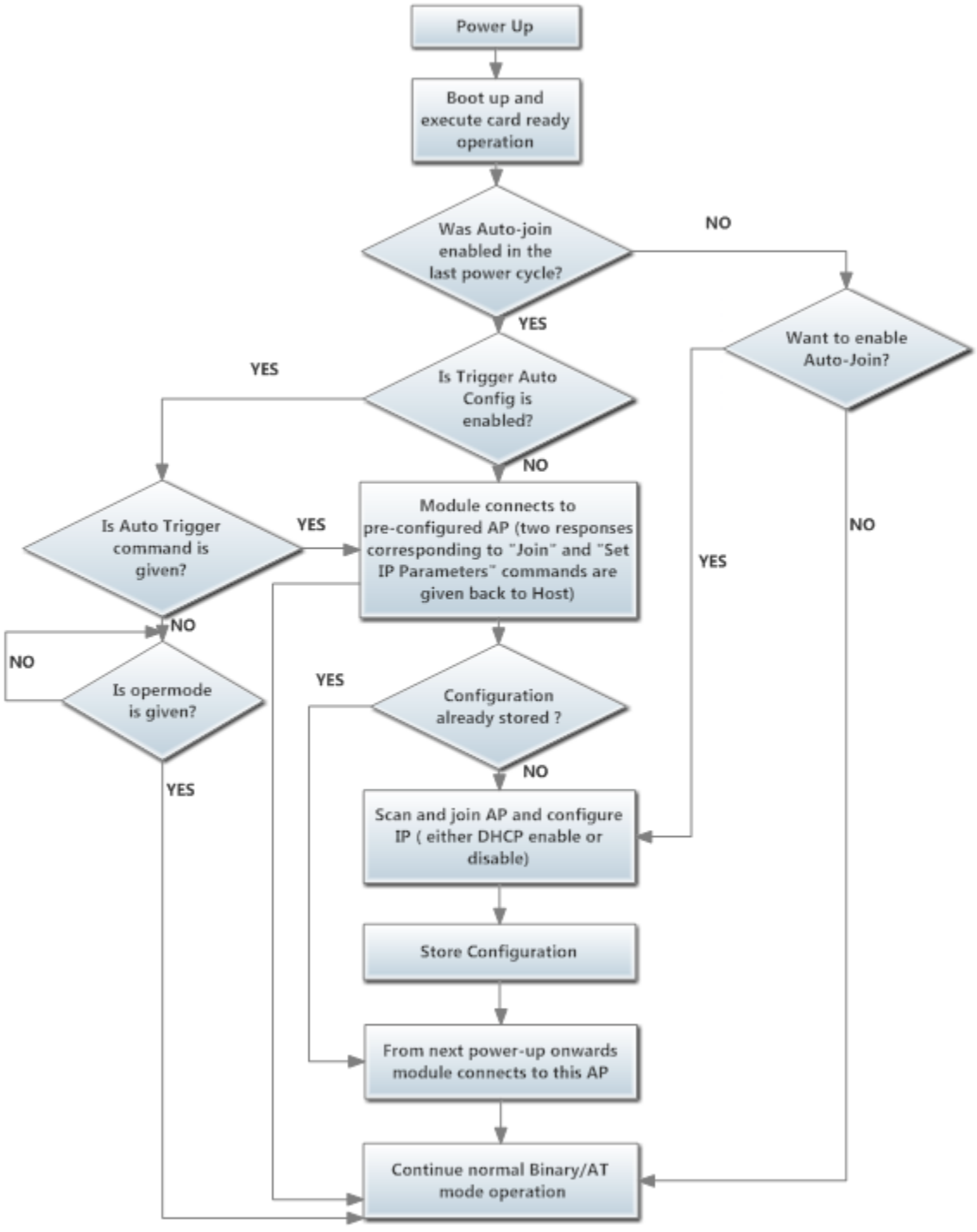


Figure 46: Connecting to Pre-configured AP

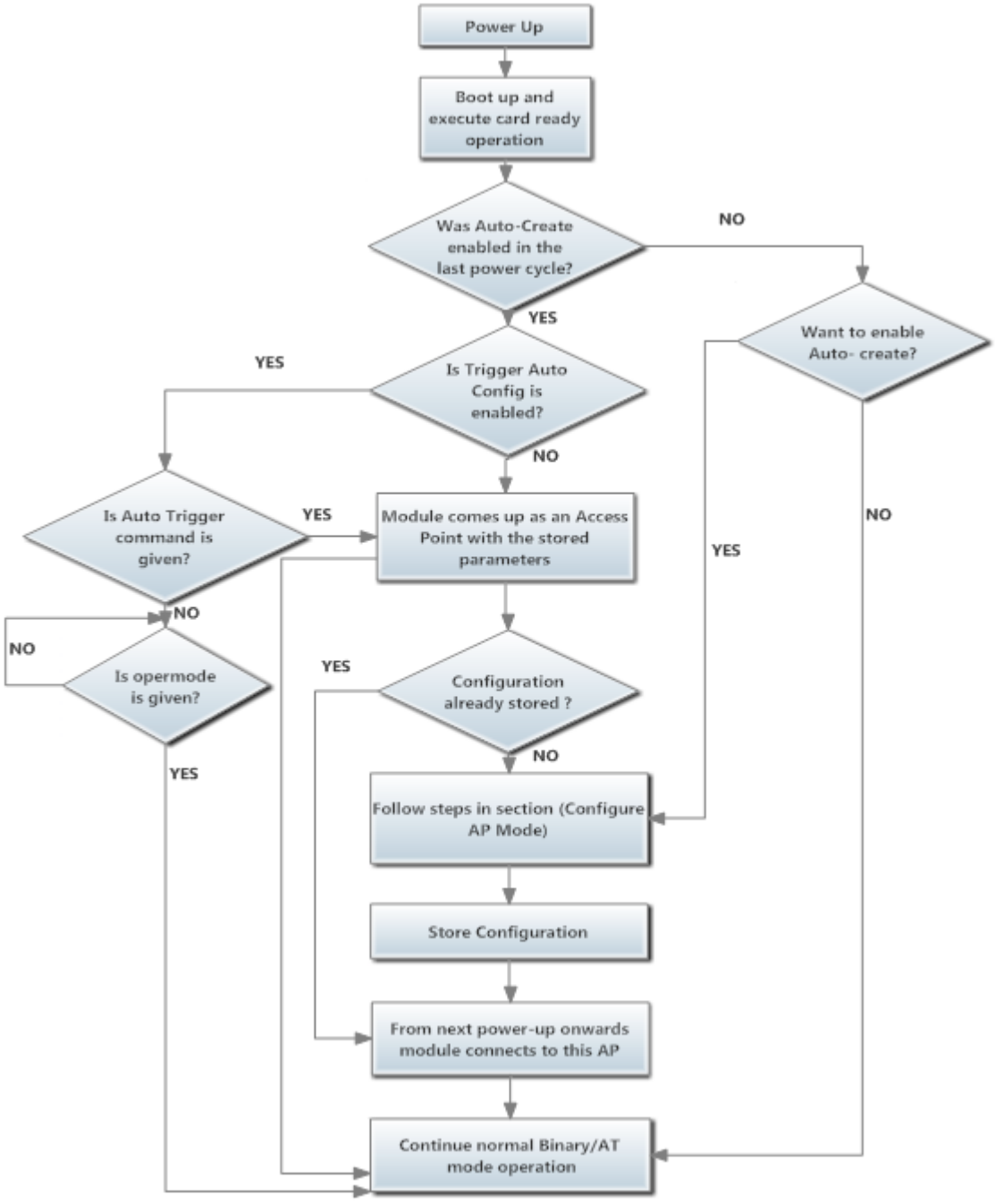


Figure 47: Creating Preconfigured AP

8.94 Store configuration structure parameters

The parameters/variables which are used in store configuration are explained in this section.Same structure is used in storing and getting the configuration parameters.

Transparent mode parameters are valid only in AT command mode on UART interface.In Binarymode and on the other interfaces,these parameters can be ignored.In binary mode these parameters are marked as reserved.

`cfg_enable (1 byte):`

0x00- auto-join or auto-create modes are disabled

0x01- auto-join or auto-create modes are enabled

`opermode (4 bytes):`

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 - WiFi 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 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 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 – Concurrent mode. In this mode, the module acts as both Access Point and Client.

`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.

Table 32: Coex Modes Supported

Note:

1) In coexistence mode (3,5,13) module supports WLAN client mode (Open mode, PSK security) and AP mode (Open mode, PSK security).
2) Embedded TCP/IP stack is not supported in WLAN+BT.
3) In coexistence mode (0) module supports all WLAN modes and embedded TCP/IP stack.

`feature_bit_map`: this bitmap 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

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 selectmodule 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

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

NOTE: feature_bit_map[0], feature_bit_map[1] are valid only in WiFi 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

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

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[30:31]-All set to '0'.

tcp_ip_feature_bit_map[31]- This bit is used to enable the tcp_ip extended feature bitmap.

1 – To enable Extended tcp_ip feature bitmap

0 – To disable Extended tcp_ip feature bitmap

Note1:

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

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 specialized 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.

2- 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 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 (Wake on Wireless) indication in UART mode

1 - Enable packet pending indication

0 - Disable packet pending indication

BIT[12]: Used to bypass AP blacklist feature.

1 – Bypass 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 wifi Direct mode.Possible values are 1 to 8 in AP mode and 1 to 4 in WiFi 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 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

0 - Enable

1 - Disable

BIT[22]: Used to enable per station power save packet buffer limit. 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[0:1],BIT[3:4],BIT[7],BIT[21],BIT[31:27]: Reserved, should be set to all '0'.

Note:

The below parameters are valid based on the operating mode given

ext_custom_feature_bit_map:

This feature bitmap is extension 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 feature.

1 – Enable antenna diversity feature

0 – Disable 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 routine 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.

BIT[3]: This bit is used to enable SSL certificate with 4096 bit key support

1 – Enable 4096 bit key support for SSL sockets

0 – Disable 4096 bit key support for SSL sockets

BIT[4]: This bit is applicable only in AP and concurrent AP mode. If this bit is set, module will send broadcast data (if any) immediately without waiting for DTIM.

1 – Enable sending broadcast packet without waiting for DTIM

0 – AP sends broadcast packet at DTIM only

Note:

If this bit is enable then connected client who is in power save may miss the packet.

ext_tcp_ip_feature_bit_map:

BIT[1] - To use DHCP User class Option

1 - To enable DHCP user class option

0 - To disable DHCP user class option

BIT[2] - To bypass the HTTP servers default root configuration page

1 - To enable HTTP server root path bypass option

0 - To disable HTTP server root path bypass option

Band (1 byte):

0x00- Module configured to operate in 2.4 GHz

0x01- Module configured to operate in 5 GHz

0x02- Dual band(2.4 Ghz and 5Ghz). Dual band is valid in station mode and p2p mode

scan_feature_bitmap(1 byte): Scan feature bitmap

BIT[0]: To enable/disable quick scan feature.

1 - To enable quick scan feature.

0 - To disable quick scan feature.

BIT[1]-BIT[7]: Reserved.

`Join_ssid (34 bytes)`:

SSID of the AP configured in auto-join or in auto-create mode. If the actual length is not 34 bytes, 0x00 filler bytes are appended to make the length 34 bytes.

`uRate ( 1 byte)`: Data rate to be configured in the module.

Please refer the PER Mode commandfor the data rates supported by the Silicon Labs module `uTXPower (1 byte)`: Tx power to be configured in the module.

At 2.4GHz

0– Low power (7+/-1) dBm

1– Medium power (10 +/-1)dBm

2– High power (18 +/- 2)dBm

At 5 GHz

0– Low power (5+/-1) dBm

1– Medium power (7 +/-1) dBm

2– High power (12 +/- 2) dBm

`join_feature_bitmap(1 byte)`:

BIT[0]: To enable b/g only mode in station mode, host has to set this bit.

0 – b/g/n mode enabled in station mode

1 – b/g only mode enabled in station mode

`BIT[1]`: To take listen interval from join command.

0 – Listen interval invalid

1 – Listen interval valid

`BIT[2]`: To enable/disable quick join feature.

1 - To enable quick join feature.

0 - To disable quick join feature.

`BIT[3]-BIT[7]`: Reserved.

`Reserved_1(1byte)`: Reserved

`Scan_ssid_len(1 byte)`: Scan ssid length

`Reserved_2(1byte)`: Reserved

`Csec_mode(1byte)`: Security mode of access point to connect in auto join mode or security mode of devut in auto create mode.This variable is used to define the security mode of the Access point to which module is supposed to connect.

Possible values:

0 – Open mode

1 – WPA security

2 – WPA2 Security

6 - Mixed mode(WPA/WPA2)

Other values are assumed to be don't care.

`psk (64 bytes)` **:** Pre shared key of the access point to which module wants to associate in auto-join or auto-create mode. Filler bytes of 0x00 are added to make it 64 bytes if the original PSK is less than 64 bytes.

`Scan_ssid (34 bytes)` **:** SSID of the AP to be scanned in auto-join. If the actual length is not 34 bytes, 0x00 filler bytes are appended to make the length 34 bytes.

`Scan_cnum(1 byte)` **:** channel number to be scanned in auto join mode. Refer to the PER Mode command for the supported channels in 2.4GHz and 5 GHz band

`dhcp_enable (1 byte)` **:**

0x00- DHCP client is disabled in module (auto-join mode)

0x01- DHCP client is enabled in module (auto-join mode)

`ip (4 bytes)` **:** Static IP configured in the module in auto-join or auto-create mode. For auto-join mode, this is valid when dhcp_enable is 0.

`Sn_mask(4 bytes)` **:** Subnet mask,this is valid only if dhcp_enable is 0.

`dgw(4 bytes)` **:** Default gateway, this is valid only if dhcp_enable is 0.

`eapMethod(32 bytes)` **:** Should be one of among TLS, TTLS, FAST or PEAP, ASCII character string used to configure the module in Enterprise security mode

`innerMethod(32 bytes)` **:** Should be fixed to MSCHAPV2, ASCII character string.This parameter is used to configure the module in Enterprise security mode.

`user_identity (64 bytes)` **:** User ID in enterprise security mode.

`Passwd (128 bytes)` **:** Password configured for enterprise security. Refer to the parameter *Password* in the command *at+rsi_eap.* Filler bytes of 0x00 are used to make the length 128 bytes, of the original length is less than 128 bytes.

`Go_intent (2 bytes)` **:** This determines whether the device is intended to form a GO (group owner) or work as a Wi-Fi Direct Peer node.. If the number is between 0 and 15, a GO negotiation takes place. If the value is 16, the module forms an Autonomous GO without negotiating with any other device.

Note:

Wi-Fi Direct, currently not supported in store configuration.

`Device_name(64 bytes)` **:** This is the device name for the module. Another Wi-Fi Direct device would see this name when it scans for Wi-Fi Direct nodes.

`Operating_channel(2 bytes)` **:** Operating channel to be used in Group Owner (GO). The specified channel is used if the device becomes a GO. The supported channels can be any valid channel.

`Ssid_postfix (64 bytes)` : This parameter is used to add a postfix to the SSID in WiFi Direct GO mode.

`Psk_key(64 bytes)` : The minimum length is 8 characters. This PSK is used by client devices to connect to the module if the module becomes a GO. WPA2-PSK security mode is used in the module in Wi-Fi Direct GO mode.

`Pmk(32 bytes)`: PMK key

`channel_no( 2 bytes )` : The channel in which the AP would operate. Refer to the PER Mode command for more details on the channels supported by the module. A value of '0' is not allowed.

`Ssid(34 bytes)` : SSID of the AP to be created

`security_type(1 byte)` : Security type of AP to be configured

0-Open

1-WPA

2-WPA2

`encryp_mode(1 byte)` : Encryption type

0-Open

1-TKIP

2-CCMP

`Psk(64 bytes)` : PSK of the AP in security mode. If the AP is in Open mode, this parameter can be set to '0'.

`beacon_interval(2 bytes)` :Beacon interval of the AP in milliseconds. Allowed values are integers from 100 to 1000 which are multiples of 100.

`dtim_period(2 bytes)` : DTIM period to be configured in AP mode

`ap_keepalive_type` : This is the bitmap to enable AP keep alive functionality and to select the keep alive type.

`BIT[0]` : To enable/disable keep alive functionality.

1 - To enable keep alive functionality.

0 - To disable keep alive functionality.

`BIT[1]` : To select AP keep alive method.

1 - To enable null data based keep alive functionality.

0 - To enable based keep alive functionality.

`ap_keepalive_period` : This is the period after which AP will disconnect the station if there are no wireless exchanges from station to AP.Keep alive period is calculated in terms of 32 multiples of beacon interval(i.e if there are no wireless transfers from station to AP with in (32*beacon_interval*keep_alive_period) milli seconds time period,station will be disconnected).If null data based method is selected,AP checks the connectivity of station by sending null data packet. If station does not ack the packet ,that station will be disconnected.

`max_sta_support(2 bytes)` : Number of clients supported. The maximum value allowed is based on the value given in custom feature select bit map[BIT[13] – BIT[16]].max_sta_support should be less than or equal to the value given in custom feature bitmap given in opermode.For example, if this value is 3, not more than 3 clients can associate to the client.

Note:

AP parameters are valid only if opermode is given as 6

`Module_mac(6bytes)` : Mac address to be set for module.

Note:

Host should send mac address with 00:00:00:00:00:00 values to use module's default mac address.

`antenna_select (2 bytes)` : This variable configures the antenna to be used. RS9113-WiSeConnect Module provides two options – an inbuilt antenna and a uFL connector for putting in an external antenna.

0– Inbuilt antenna selected

1– UFL connector selected

`Reserved_3(2 bytes)` : reserved

`Index(2 bytes)` : In some APs, there is an option to provide four WEP keys.

0-Key 1 will be used.

1-Key 2 will be used.

2-Key 3 will be used.

3-Key 4 will be used.

`Key(32 bytes)` : Actual keys. There are two modes in which a WEP key can be set in an Access Point- WEP (hex) mode and WEP (ASCII) mode. The module supports WEP (hex) mode only.

`dhcpv6_enable(2 bytes)` : DHCPv6 mode.

`prefix_length(2 bytes)` : prefix length of ipv6 address.

`ip6(16 bytes)` : IPv6 address of module.

`Dgw6(16 bytes)` : IPv6 address of default router.

`tcp_stack_used(1 byte)` : shows which TCP stack is used. Possible values are:

0x01- ipv4 Stack

0x02- ipv6 Stack

0x03- dual Stack, both IPv4 and IPv6 Stack

`bgscan_magic_code(2 bytes)` : This magic code is used to validate the bgscan parameter present in flash memory.

`bgscan_enable(2 bytes)` : To enable/Disable bgscan

0 – Disable

1 - Enable

`bgscan_threshold(2 bytes)` : This is the threshold in dBm to trigger the bgscan. After bgscan periodicity, If connected AP RSSI falls below this then bgscan will be triggered.

`rssi_tolerance_threshold(2 bytes)` : This is difference of last RSSI of connected AP and current RSSI of connected AP. Here last RSSI means RSSI calculated at last time beacon received and current RSSI is RSSI calculated at current beacon received. If this difference is more than rssi_tolerance_threshold and current RSSI is greater then bgscan_threshold then bgscan will be triggered irrespective of periodicity.

`bgscan_periodicity(2 bytes )` : This is time period in seconds to trigger bgscan if RSSI of connected AP is above (assuming RSSI is positive value) than the given bgscan_threshold.

`active_scan_duration(2 bytes )`: This is active scan duration and it is in ms.

`passive_scan_duration(2 bytes )`: This is passive scan duration in ms.

`multi_probe(1 byte)` : If set to one then module will send two probe request one with specific SSID provided during join command and other with NULL ssid (to scan all the access points).

`chan_bitmap_magic_code(2 bytes)` : This variable is used to validate the given scan channel bitmaps.If magic code is 0x4321,then only the scan channel bitmaps are considered as valid.

`scan_chan_bitmap_stored_2_4_GHz(4bytes)` : channel bitmap for scanning in set of selective channels in 2.4 ghz band

`scan_chan_bitmap_stored_5_GHz(4bytes)` : channel bitmap for scanning in set of selective channels in 5 ghz band

`roam_magic_code(2 bytes)` : This magic code is used to validate the roaming parameters stored in the flash memory.

`roam_enable(4 bytes)` : To Enable/Disable roaming.

0 – Disable

1 - Enable

`roam_threshold(4 bytes)` : If connected AP RSSI falls below this then module will search for new AP from background scanned list.

`roam_hysteresis(4 bytes)` : If module found new AP with same configuration (SSID, Security etc) and if (connected_AP_RSSI – Selected_AP_RSSI) is greater than roam_hysteresis then it will try to roam to the new selected AP.

`rejoin_magic_code(2 bytes)` : This magic code is used to validate the rejoin parameters stored in the flash memory.

`rejoin_max_retry(4 bytes)` : This is 4 byte unsigned integer. This represents the number of attempt for join before giving up the error.

Note:

If number of rejoin attempts is 0 then module will try infinitely for rejoin.

`Rsi_scan_interval(4 bytes)` : This is 4 byte signed integer. This is time interval in second for the subsequent retry.

`Rsi_beacon_missed_count(4 bytes)` : This is 4 byte signed integer. This is the beacon missed count that module used to declare module connection status. If module found continuous beacon missed is greater than or equal to this value then it will declare connection as disconnected and will start rejoin process again.

`Rsi_first_time_retry_enabled(4 bytes)`:

This is 4 byte unsigned integer. If this is 1 then module will retry to connect for the first time itself for join. Number of attempt and scan interval may be configured by rejoin_max_attempts and scan_interval respectively.

`region_request_from_host(1 byte)`:

If this variable is 1,region of the module is set either in auto join or auto create mode based on the opermode.

1 – Enable set region in Auto create or Auto join mode

0 – Disable set region in Auto create or Auto join mode

`rsi_region_code_from_host(1 byte)`:

Enable/Disable set region code from user.

If opermode is 0 or 2:

1 - Enable - Use the region information from user command

0 – Disable - Use the region information from beacon(country Ie)

If opermode is 6:

0- Disable-Get the region information based on region code from internal memory

`region_code(1 byte)`:

If the region code is given as 0(zero), US domain is considered by default and device is configured according to the US domain regulations.

1-US domain

2-Europe domain

3-Japan Domain

Note:

1) All the magic codes should be 0x4321.Firmware validates the respective details if and only if the magic code is matching
2) Below given parameters are transparent mode specific.These variables are named as reserved_6 in binary mode command mode.

`Trans_mode_enable(2 bytes)` : If transparent mode magic word (0x5C5C) is given transparent mode is enabled, after successful connection/creation of socket.

`packet_len(2 bytes)` : This is the number of bytes (payload) with which network frames are formed and transmitted (bytes/data received within gap timeout) over TCP/IP network.

`Escape_char (1,ASCII)` : Special character provided which will be detected and its 3 character sequence will have special meaning depending of time of arrival.

`Gap_time (2 bytes)` : Maximum time gap between bytes received from host within which escape characters are not expected/checked.

`Frame_time (2 bytes)` : The timeout period for framing network packet from the bytes received. if this timeout is occurred , bytes available in Rx buffers will be forced to form a network frame and queue it for transmission. Ideally Framing period = 2 * Gap Time.

`Escape_time (2 bytes)` : If escape character sequence is received after framing timeout and within escape time, then module breaks out of transparent mode,making GPIO low. Ideally Escape Time = 2 * Framing period.

`Ip_version(2 bytes)`: Signifies which IP version to be used in transparent mode.

4 – IP version 4
6 - IP version 6

`nSocketType (2 bytes)` : This gives the protocol which is to be used in transparent mode(TCP/UDP/LTCP/LUDP).

0 - TCP
2 – LTCP
4 – LUDP

`stLocalPort(2 bytes)` : station Local port number to be used in transparent mode.

`Dst_port(2 bytes)` : Remote port number, to which module need to communicate with in transparent mode.

`Ipv4_address/Ipv6_address(16 bytes)`: IP(v4/v6) of remote server which is to be communicated with from module(in client mode).

`Max_count(2 bytes)` : maximum no of clients allowed in transparent mode is fixed to 1, so default value should be 1.

`TOS(4 bytes)` : Type of service.

Refer Table 27: TOS Values for more details

`ssl_enabled(1 byte)` : If ssl socket is to be used corresponding parameters are to be provided here.

0 – To open TCP socket.

1 - To open SSL client socket.

5 - To open SSL socket with TLS 1.0 version.

9 - To open SSL socket with TLS 1.2 version.

`Ssl_ciphers(1 byte)` : If ssl socket is to be used corresponding ciphers used is to be provided here.

BIT(0) : 1: TLS_RSA_WITH_AES_256_CBC_SHA256

BIT(1) : 2: TLS_RSA_WITH_AES_128_CBC_SHA256

BIT(2) : 4: TLS_RSA_WITH_AES_256_CBC_SHA

BIT(3) : 8: TLS_RSA_WITH_AES_128_CBC_SHA

BIT(4) : 16: TLS_RSA_WITH_AES_128_CCM_8

BIT(5) : 32: TLS_RSA_WITH_AES_256_CCM_8

multicast_magic_code(2 bytes):

This magic code is used to validate the multicast parameters stored in the flash memory

multicast_bitmap(2 bytes):

There are two bytes in the command which represent 2 parts. Lower order byte represent the command type (cmd as mentioned below) and higher order byte is the hash value (6 Bits) generated from the desired multicast mac address (48 Bits) using hash function.
multicast_bitmap[0:1]: These 2 bits represents the command type.
Possible values are:

0 - RSI_ MULTICAST_MAC_ADD_BIT (To set particular bit in multicast bitmap)

1 - RSI_ MULTICAST_MAC_CLEAR_BIT (To reset particular bit in multicast bitmap)

2 - RSI_ MULTICAST_MAC_CLEAR_ALL (To clear all the bits in multicast bitmap)

3 - RSI_ MULTICAST_MAC_SET_ALL(To set all the bits in multicast bitmap)

multicast_bitmap[2:7]: reserved.

multicast_bitmap[8:13]: 6bit hash value generated from the hash algorithm which corresponds to the multicast mac address is used to set/reset corresponding bit in multicast filter bitmap.This field is valid only if 0 or 1 is selected in command type (multicast_bitmap[0:1]).

multicast_bitmap[14:15]: reserved

powermode_magic_code(2 bytes):

This magic code is used to validate the power mode parameters stored in the flash memory

powermode(1 byte):

powermode variable configures the power save mode of the module.

1–Power save Mode 1

2–Power save Mode 2

3–Power save Mode 3

ulp_mode(1 byte):

0 - Low power mode.

1 - Ultra low power mode with RAM retention.

2 - ULtra low power mode without RAM retention.

Refer section Powersave operation for detail description about power save operation.

`wmm_ps_magic_code(2 bytes)`:

This magic code is used to validate the WMM power save parameters stored in the flash memory

`wmm_ps_enable(1 byte)`: To enable or disable WMM

0 - Disable

1- Enable

`wmm_ps_type(1 byte)`: WMM PS type

0 - Tx Based

1- Periodic

`wakeup_interval(4 bytes)`: Wakeup interval in milli seconds.

`wmm_ps_uapsd_bitmap(1 byte)`: Bitmap , 0 to 15 possible values.

`wmm_ps_uapsd_bitmap`[0]:Access category: voice

`wmm_ps_uapsd_bitmap`[1]: Access category:video

`wmm_ps_uapsd_bitmap`[2]: Access category:Back ground

`wmm_ps_uapsd_bitmap`[3]: Access category:Best effort U-APSD

`wmm_ps_uapsd_bitmap`[4:7]: All set to '0'. Don't care bits.

`Listen_interval(4 byte)`:

This is valid only if BIT(1) in join_feature_bit_map is set. This value is given in Time units(1024 milliseconds). This parameter to configure maximum sleep duration in power save.

`private_key_password[82](1 byte)`: Private Key Password is required for encrypted private key, format is like "\"12345678\"".

`join_bssid`: This contains BSSID of selected AP.

`Listen_interval_dtim(1 byte)`:

This parameter is valid only if BIT(1) is set in the join_feature_bitmap and valid listen interval is given in join command. If this parameter is set, the module computes the desired sleep duration based on listen interval (from join command) and its wakeup align with Beacon or DTIM Beacon (based on this parameter).

0 - module wakes up before nearest Beacon that does not exceed the specified listen interval time.

1 - module wakes up before nearest DTIM Beacon that does not exceed the specified listen interval time.

Note:

1) All the magic codes should be 0x4321.Firmware validates the respective details if and only if the magic code is matching
2) Transparent mode is only available in AT mode with UART interface. In Transparent Mode only one socket can be used(which is mentioned in transparent mode params). Minimum packetization length is 10(bytes) Maximum packetization length is

1024(bytes). Gap time should be greater then 100(milli seconds), timing parameters condition to be met is Escape time > Framing time > Gap time.

`Fast_psp_enable(1 byte) :`

When fast psp is enabled, module will disable power save for monitor interval of time for each data packet received or sent.

`Monitor_interval (2 bytes):`

This is time in ms to keep module in wakeup state for each Tx or Rx traffic sent or received respectively. Default value for this is 50 ms.

`Timeout_value (2 bytes ) :` timeout value in ms(default 300ms).

`timeout_bitmap (4 bytes):`

`BIT[0]:` sets timeout for association and authentication request.

`Request_timeout_magic_word( 2 bytes):` Magic word of request time out.

`ht_caps_magic_word (2 bytes)` : Magic word od HT caps.

`dhcp_ap_enable (1 byte)` : DHCPv4 mode enable or disable

`ap_ip(4 bytes)` : Module IP address

`ap_sn_mask(4 bytes)` : Sub-net mask

`ap_dgw(4 bytes)` : Default gateway

`dhcpv6_ap_enable(2 bytes)` : DHCPv6 mode enable or disable

`ap_prefix_length(2 bytes)` : prefix length of ipv6 address.

`ap_ip6(16 bytes)` : IPv6 address of module.

ap_dgw6(16 bytes): IPv6 address of default router.

Note:

The above AP IP parameters are valid in concurrent mode.

`http_credentials_avail(1 byte):Reserved.`

`http_username[MAX_HTTP_SERVER_USERNAME](1 byte)` : HTTPserver username.

`http_password[MAX_HTTP_SERVER_PASSWORD](1 byte)` : HTTP server password.

8.95 Store Profile configuration

This command is used to give the configuration (A.P profile, client profile, Enterprise profile and P2p Autonomous GO mode profile) values which are supposed to be stored in the module's non-volatile memory and that are used in auto-join or auto-create modes.

Note:

This store profile configuration command is supported in SAPIs only.

Command Format:

Binary Mode:

```
typedef struct ap_profile
{
  uint8_t    ap_profile_magic_word[4];
  uint8_t    wlan_feature_bit_map[4];
  uint8_t    tcp_ip_feature_bit_map[4];
  uint8_t    custom_feature_bit_map[4];
  uint8_t    data_rate;
  uint8_t    tx_power;
  uint8_t    band;
  uint8_t    channel[2];
  uint8_t    ssid[RSI_SSID_LEN];
  uint8_t    security_type;
  uint8_t    encryption_type;
  uint8_t    psk[RSI_PSK_LEN];
  uint8_t    beacon_interval[2];
  uint8_t    dtim_period[2];
  uint8_t    keep_alive_type;
  uint8_t    keep_alive_counter;
  uint8_t    max_no_sta[2];
    network_profile_t network_profile;

} ap_profile_t;


typedef struct client_profile
{
  uint8_t    client_profile_magic_word[4];
  uint8_t    wlan_feature_bit_map[4];
  uint8_t    tcp_ip_feature_bit_map[4];
  uint8_t    custom_feature_bit_map[4];
  uint8_t    listen_interval[4];
  uint8_t    data_rate;
```

```
 uint8_t    tx_power;
 uint8_t    band;
 uint8_t    ssid[RSI_SSID_LEN];
 uint8_t    ssid_len;
 uint8_t    channel[2];
 uint8_t    scan_feature_bitmap;
 uint8_t    scan_chan_bitmap_magic_code[2];
 uint8_t    scan_chan_bitmap_2_4_ghz[4];
 uint8_t    scan_chan_bitmap_5_0_ghz[4];
 uint8_t    security_type;
 uint8_t    encryption_type;
 uint8_t    psk[RSI_PSK_LEN];
 uint8_t    pmk[RSI_PMK_LEN];
 wep_key_ds_t wep_key;

 network_profile_t network_profile;

} client_profile_t;

typedef struct network_profile
{
 uint8_t    tcp_stack_used;

 uint8_t    dhcp_enable;
          uint8_t    ip_address[4];
          uint8_t    sn_mask[4];
          uint8_t    default_gw[4];

          uint8_t    dhcp6_enable;
          uint8_t    prefix_length[2];
          uint8_t    ip6_address[16];
 uint8_t    deafault_gw[16];

} network_profile_t;
```

```
typedef struct eap_client_profile
{
  uint8_t    eap_profile_magic_word[4];
  uint8_t    wlan_feature_bit_map[4];
  uint8_t    tcp_ip_feature_bit_map[4];
  uint8_t    custom_feature_bit_map[4];
  uint8_t    listen_interval;
  uint8_t    data_rate;
  uint8_t    tx_power;
  uint8_t    band;
  uint8_t    ssid[RSI_SSID_LEN];
  uint8_t    ssid_len;
  uint8_t    channel[2];
  uint8_t    scan_feature_bitmap;
  uint8_t    scan_chan_bitmap_magic_code[2];
  uint8_t    scan_chan_bitmap_2_4_ghz[4];
  uint8_t    scan_chan_bitmap_5_0_ghz[4];
  uint8_t    security_type;
  uint8_t    eap_method[32];
  uint8_t    inner_method[32];
  uint8_t    user_identity[64];
  uint8_t    passwd[128];
    network_profile_t network_profile;
} eap_client_profile_t;

typedef struct p2p_profile
{

  uint8_t    p2p_profile_magic_word[4];
  uint8_t    wlan_feature_bit_map[4];
  uint8_t    tcp_ip_feature_bit_map[4];
  uint8_t    custom_feature_bit_map[4];
  uint8_t    data_rate;
  uint8_t    tx_power;
```

```
 uint8_t    band;
 uint8_t    join_ssid[RSI_SSID_LEN];
 uint8_t    go_intent[2];
 uint8_t    device_name[64];
 uint8_t    operating_channel[2];
 uint8_t    ssid_postfix[64];
 uint8_t    psk_key[64];;

network_profile_t network_profile;

} p2p_profile_t;


typedef struct rsi_config_profile_s
{
 uint8_t    profile_type[4];

 union
 {
  //! AP config profile
  ap_profile_t     ap_profile;

  //! Client config profile
  client_profile_t  client_profile;

  //! EAP client config profile
  eap_client_profile_t eap_client_profile;

  //! P2P config profile
  p2p_profile_t  p2p_profile;

 } wlan_profile_struct;

} rsi_config_profile_t;
```

```
typedef struct rsi_profile_req_s
{
 uint8_t    profile_type[4];

} rsi_profile_req_t;

typedef struct rsi_auto_config_enable_s
{
  uint8_t   config_enable;
  uint8_t   profile_type[4];

} rsi_auto_config_enable_t;
```

Command Parameters:

length_of_payload: Length in bytes of the Store configuration parameters field.

Store configuration parameters: Store configuration parameter in hex format.

For payload parameters description, refer section Store Configuration structure parameters.

Binary Mode:

There is no response payload.

Relevance:

This command is valid when opermode is 0,1,2 or 6.

8.96 Set RTC time

Description:

This command is used to set/initialize the real time clock of the module from the host.

Note:

To enable this feature, host need to set BIT[28] of custom feature bitmap through opermode command

Command Format:

```
at+rsi_host_rtc_time=<second>,<minute>,<hour>,<day>,<month>,<year>\r\n
```

Binary Mode:

```
struct
```

```
{
      uint8  second[4];
      uint8  minute[4];
      uint8  hour[4];
      uint8  day[4];
      uint8  month[4];
      uint8  year[4];
} module_rtc_time;
```

Command Parameters:

`second`: This is current real time clock seconds, which needs to set for module.

`minute`: This is current real time clock minute, which needs to set for module.

`hour`: This is current real time clock hour, which needs to set for module.

Note
Hour is 24 hour format only (valid values are 0 to 23)
Valid values for Month are 0 to 11 (Jan to Dec)

`day`: This is current real time clock day, which needs to set for module.

`month`: This is current real time clock month, which needs to set for module.

`year`: This is current real time clock year, which needs to set for module.

Response:

AT Mode:

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

Binary Mode:

No response payload for this command.

Possible error codes:

Possible error codes are 0x0021, 0x0025

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

Below example configure the module rtc time to APRIL 2 10:10:10 2016

```
at+rsi_host_rtc_time=10,10,10,2,4,2016\r\n
```

Response:

```
OK\r\n
0x4F  0x4B  0x0D  0x0A
```

8.97 PUF ENROLL

Description:

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

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_puf_enroll\r\n
```

Command Parameters:

No Parameters.

Response:

AT Mode:

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC2F, 0xCC33, 0xCC34, 0xCC35`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_puf_enroll\r\n
```

Response:

```
OK\r\n
0x4F 0x4B 0x0D 0x0A
```

8.98 PUF DISABLE ENROLL

Description:

This command will block further Enrollment of PUF.

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_puf_dis_enroll\r\n
```

Command Parameters:

No Parameters.

Response:

AT Mode:

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC32, 0xCC33, 0xCC34`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_puf_dis_enroll\r\n
```

Response:

```
OK\r\n
0x4F 0x4B 0x0D 0x0A
```

8.99 PUF SATRT

Description:

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

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_puf_start\r\n
```

Command Parameters:

No Parameters.

Response:

AT Mode:

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC31, 0xCC32, 0xCC33, 0xCC35`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_puf_start\r\n
```

Response:

```
OK\r\n
0x4F 0x4B 0x0D 0x0A
```

8.100 PUF SET KEY

Description:

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

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_puf_set_key=<key_index>,<key_size>,<key>\r\n
```

Command Parameters:

Key_index: Key Index of the key. Valid range is between 0 – 15.

Key_size:

- 0 – 128 bit
- 1 – 256 bit

Key: Actual key for which key code is generated.

Response:

AT Mode:

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC31, 0xCC32, 0xCC33, 0xCC35`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_puf_set_key=1,0,303132333435363738396566676869 70\r\n
```

Response:

```
OK<keycode>\r\n
```

```
0x4F 0x4B 0x00 0x01 0x00 0x02 0x79 0x4c 0xa4 0x45 0x0c 0xe2
0xec 0xb4 0x8a 0xb3 0x34 0x52 0xc0 0x80 0x2d 0x37 0xeb 0xda
0xe7 0x36 0x5c 0xfc 0x3f 0xf8 0x88 0xe2 0x19 0xda 0x8c 0xeb
0x1c 0x9b 0xe9 0xb6 0x6b 0xb4 0x19 0x59 0x46 0x21 0x0D 0x0A
```

Note:

Key should be given in the Hexadecimal format.

8.101 PUF DISABLE SET KEY

Description:

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

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_puf_dis_set_key\r\n
```

Command Parameters:

No Parameters.

Response:

AT Mode:

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC32, 0xCC33, 0xCC34`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_puf_dis_set_key\r\n
```

Response:

```
OK\r\n
0x4F  0x4B  0x0D  0x0A
```

8.102 PUF GET KEY

Description:

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

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_puf_get_key=<keycode>\r\n
```

Command Parameters:

Key_code: Keycode which is prior generated by PUF.

Response:

AT Mode:

Result Code	Description

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC31, 0xCC32, 0xCC33, 0xCC35`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_puf_get_key=0010002794ca4450ce2ecb48ab33452c0802d37ebda
e7365cfc3ff888e219da8ceb1c9be9b66bb419594621\r\n
```

Response:

```
OK0123456789efghip\r\n
```

```
0x4F 0x4B 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39
0x65 0x66 0x67 0x68 0x69 0x70 0x0D 0x0A
```

Note:

Keycode should be given in the Hexadecimal format.

8.103 PUF DISABLE GET KEY

Description:

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

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_puf_dis_get_key\r\n
```

Command Parameters:

No Parameters.

Response:

AT Mode:

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC32, 0xCC33, 0xCC34`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_puf_dis_get_key\r\n
```

Response:

```
OK\r\n
0x4F  0x4B  0x0D  0x0A
```

8.104 PUF LOAD KEY

Description:

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

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_puf_load_key=<keycode>\r\n
```

Command Parameters:

Key_code: Keycode which is prior generated by PUF.

Response:

AT Mode:

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC31, 0xCC32, 0xCC33, 0xCC35`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_puf_load_key=0010002794ca4450ce2ecb48ab33452c0802d37ebd
ae7365cfc3ff888e219da8ceb1c9be9b66bb419594621\r\n
```

Response:

```
OK\r\n
```

```
0x4F 0x4B 0x0D 0x0A
```

Note:

Keycode should be given in the Hexadecimal format.

8.105 PUF INTRINSIC KEY

Description:

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

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_puf_intr_key=<key_index>,<key_size>\r\n
```

Command Parameters:

Key_index: Key Index of the key. Valid range is between 0 – 15.

Key_size:

- 0 – 128 bit
- 1 – 256 bit

Response:

AT Mode:

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC31, 0xCC32, 0xCC33, 0xCC35`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_puf_set_key=1,0\r\n
```

Response:

```
OK<keycode>\r\n
```

```
0x4F 0x4B 0x01 0x01 0x00 0x02 0x79 0x4c 0xa4 0x45 0x0c 0xe2
0xec 0xb4 0x8a 0xb3 0x34 0x52 0xc0 0x80 0x2d 0x37 0xeb 0xda
0xe7 0x36 0x5c 0xfc 0x3f 0xf8 0x88 0xe2 0x19 0xda 0x8c 0xeb
0x1c 0x9b 0xe9 0xb6 0x6b 0xb4 0x19 0x59 0x46 0x21 0x0D 0x0A
```

8.106 AES ENCRYPT

Description:

This command encrypts data inputted with Key provided or with key which is already loaded into AES by PUF. This command provides provision for encryption with AES engine in two modes(ECB, CBC). Parameters should be given to command depending on the mode of usage. The command will return failure if there is an error in input.

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_aes_encrypt=<mode>,<key>,<IV>,<data_size>,<data>\r\n
```

Command Parameters:

mode: BIT[0]: 1 – CBC, 0 – ECB

- BIT[1]: 1 – key size 256, 0 – key size 128
- BIT[2]: 1 – key from AES, 0 – key from PUF
- BIT[3 : 7]: Reserved

Key: Key which is to be used for encryption.

IV: Initialization vector (IV) which is used for encryption (valid only for CBC mode).

Data size: Data size of the data to be encrypted in bytes; the data size should be in multiple of key size. The maximum data size is 1400.

Data: Data which is to be encrypted.

Response:

AT Mode:

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC32`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_aes_encrypt=5,30313233343536373839656667686970,
6162636465666768696A6B6C6D6E6F70,16,303132333435363738396566676
686970\r\n
```

Response:

```
OK<Encrypted data>\r\n
```

```
0x4F 0x4B 0x4E 0x9A 0xF8 0x2D 0x1B 0xBB 0x4D 0x32 0xC3 0x7E
0xAC 0x5C 0x26 0x08 0x85 0x00 0x0D 0x0A
```

Note:

Key, IV, data should be given in the Hexadecimal format.

8.107 AES DECRYPT

Description:

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

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_aes_encrypt=<mode>,<key>,<IV>,<data_size>,<data>\r\n
```

Command Parameters:

mode: BIT[0]: 1 – CBC, 0 – ECB

- BIT[1]: 1 – key size 256, 0 – key size 128
- BIT[2]: 1 – key from AES, 0 – key from PUF
- BIT[3 : 7]: Reserved

Key: Key which is to be used for decryption.

IV: Initialization vector (IV) which is to be used for decryption (valid only for CBC mode).

Data size: Data size of the data to be decrypted in bytes; the data size should be in multiple of key size. The maximum data size is 1400.

Data: Data which is to be decrypted.

Response:

AT Mode:

Result Code	Description

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC32`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_aes_decrypt=5,30313233343536373839656667686970,
6162636465666768696A6B6C6D6E6F70,16,4E9AF82D1BBB4D32C37EAC5C26
088500\r\n
```

Response:

```
OK<Decrypted data>\r\n
```

```
0x4F 0x4B 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39
0x65 0x66 0x67 0x68 0x69 0x70 0x0D 0x0A
```

Note:

Key, IV, data should be given in the Hexadecimal format.

8.108 AES MAC

Description:

This command generates Message authentication check (MAC) for the data inputted with provided key as well as initialization vector (IV). The parameters should be given to command depending on the mode of usage. The command will return failure if there is any error in input.

Note:

To enable this feature, host need to set BIT[31] of custom feature bitmap and BIT[7] of ext_custom_feature_bit_map through opermode command.

Command Format:

```
at+rsi_aes_mac=<mode>,<key>,<IV>,<data_size>,<data>\r\n
```

Command Parameters:

mode: BIT[0]: Reserved.

- BIT[1]: 1 – key size 256, 0 – key size 128
- BIT[2 : 7]: Reserved

Key: Key which is to be used for MAC generation.

IV: Initialization vector (IV) which is to be used for MAC generation.

Data size: Data size of the data to be used for MAC generation in bytes; the data size should be multiple of key size. The maximum data size is 1400.

Data: Data which is to be used for MAC generation.

Response:

AT Mode:

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

Possible error codes:

Possible error codes are `0x0021, 0xFF74, 0xCC32`

Relevance:

This command is relevant in all modes.

Example:

AT Mode:

```
at+rsi_aes_mac=0,30313233343536373839656667686970,
6162636465666768696A6B6C6D6E6F70,32,303132333435363738396566676
8697030313233343536373839656667686970\r\n
```

Response:

```
OK<Generated MAC>\r\n
```

```
0x4F 0x4B 0xE1 0x3F 0xDC 0x0E 0x43 0x17 0x97 0x7D 0x05 0x36
0x44 0x0C 0x22 0x13 0xA2 0xC0 0x0D 0x0A
```

Note:

Key, IV, data should be given in the Hexadecimal format.

9 Error Codes

This section explains error codes for different commands.

Error Codes (in hexadecimal format)	Description
0x0002	Scan command issued while module is already associated with an Access Point
0x0003	No AP found
0x0004	Wrong PSK is issued while the module client tries to join an Access Point with WEP security enabled
0x0005	Invalid band
0x0006	Association not done or in unassociated state
0x0008	Deauthentication received from AP
0x0009	Failed to associate to Access Point during “Join”
0x000A	Invalid channel
0x000E	1) Authentication failure during “Join” 2) Unable to find AP during join which was found during scan.
0x000F	Missed beacon from AP during join
0x0013	Non-existent MAC address supplied in “Disassociate” command
0x0014	Wi-Fi Direct or EAP configuration is not done
0x0015	Memory allocation failed or Store configuration check sum failed
0x0016	Information is wrong or insufficient in Join command
0x0018	Push button command given before the expiry of previous push button command.
0x0019	1) Access Point not found 2) Rejoin failure
0x001A	Frequency not supported
0x001C	EAP configuration failed
0x001D	P2P configuration failed
0x001E	Unable to start Group Owner negotiation
0x0020	Unable to join
0x0021	Command given in incorrect state

Error Codes (in hexadecimal format)	Description
0x0022	Query GO parameters issued in incorrect operating mode
0x0023	Unable to form Access Point
0x0024	Wrong Scan input parameters supplied to “Scan” command
0x0025	Command issued during re-join in progress
0x0026	Wrong parameters the command request
0x0028	PSK length less than 8 bytes or more than 63 bytes
0x0029	Failed to clear or to set the Enterprise Certificate (Set Certificate)
0x002A	Group Owner negotiation failed in Wi-Fi Direct mode
0x002B	Association between nodes failed in Wi-Fi Direct mode/ WPS Failed due to timeout
0x002C	If a command is issued by the Host when the module is internally executing auto-join or auto-create
0x002D	WEP key is of wrong length
0x002E	ICMP request timeout error
0x002F	Invalid beacon interval
0x0030	Send data packet exceeded the limit or length that is mentioned
0x0031	ARP Cache entry not found
0x0032	UART command timeout happened
0x0033	Fixed data rate is not supported by connecting AP.
0x0037	Wrong WPS PIN
0x0038	Wrong WPS PIN length
0x0039	Wrong PMK length
0x003a	SSID not present for PMK generation
0x003b	SSID incorrect for PMK generation(more than 34 bytes)
0x003C	Band not supported
0x003D	User store configuration invalid length
0x003E	Error in length of the command(Exceeds number of characters is mentioned in the PRM).
0x003F	Data packet dropped

Error Codes (in hexadecimal format)	Description
0x0040	WEP key not given
0x0041	Wrong PSK length
0x0042	PSK or PMK not given
0x0043	Security mode given in join command is invalid
0x0044	Beacon misscount reaches max beacon miss count(Deauth due to beacon miss)
0x0045	Deauth received from supplicant
0x0046	Deauth received from AP after channel switching
0x0047	Synchronization missed
0x0048	Authentication timeout occurred
0x0049	Association timeout
0x004A	BG scan in given channels is not allowed
0x004B	Scanned SSID and SSID given in Join are not matching
0x004C	Given number of clients exceeded max number of stations supported
0x004D	Given HT capabilities are not supported
0x004E	Uart Flow control not supported
0x004F	ZB/BT/BLE packet received and protocol is not enabled.
0x0050	Parameters error
0x0051	Invalid RF current mode
0x0052	Power save support is not present for a given interface.
0x0053	Concurrent AP in connected state
0x0054	Connected AP or Station channel mismatch
0x0055	IAP co processor error
0x0056	WPS not supported in current operating mode
0x0057	Concurrent AP has not same channel as connected station channel
0x0058	PBC session overlap error
0x005A	4/4 confirmation of 4 way handshake failed

Error Codes (in hexadecimal format)	Description
0X005C	Concurrent mode, both AP and Client should up, to enable configuration
0x005B	MAC address not present in MAC based join
0x005D	Config-Value is not with in the range
0x005E	Invalid configuration type is given
0x0074	Feature not enabled
0x00B0	Timeout bitmap failed
0x00B1	Memory Error: No memory available.
0x00B2	Invalid characters in JSON object
0x00B3	Update Commands: No such key found.
0x00B4	No such file found: Re-check filename
0x00B5	No corresponding webpage exists with same filename
0x00B6	Space unavailable for new file.
0x00C1	Invalid input data, Re-check filename, lengths etc
0x00C2	Space unavailable for new file
0x00C3	Existing file overwrite: Exceeds size of previous file. Use erase and try again
0x00C4	No such file found. Re-check filename.
0x00C5	Memory Error: No memory available.
0x00C6	Received more webpage data than the total length initially specified.
0x00C7	Error in set region command
0x00C8	Webpage current chunk length is incorrect
0x00CA	Error in Ap set region command
0X00CB	Error in AP set region command parameters
0x00CC	Region code not supported
0x00CD	Error in extracting country region from beacon
0x00CE	Module does not have selected region support
0x00D1	SSL Context Create Failed.

Error Codes (in hexadecimal format)	Description
0x00D2	SSL Handshake Failed. Socket will be closed.
0x00D3	SSL Max sockets reached. Or FTP client is not connected
0x00D4	Cipher set failure
0x00F1	HTTP credentials maximum length exceeded.
0x0100	SNMP internal error.
0x0104	SNMP invalid IP protocol error
0xBB01	No data received or receive time out.
0xBB0A	Invalid SNTP server address
0xBB0B	SNTP client not started
0xBB10	SNTP server not available, Client will not get any time update service from current server
0xBB15	SNTP server authentication failed
0xBB0E	Internal error.
0xBB16	Entry not found for multicast IP address
0xBB17	No more entries found for multicast
0xBB21	IP address error
0xBB22	Socket already bound.
0xBB23	Port not available.
0xBB27	Socket is not created
0xBB29	ICMP request failed
0xBB33	Maximum listen sockets reached.
0xBB34	DHCP duplicate listen
0xBB35	Port Not in close state.
0xBB36	Socket is closed or in process of closing
0xBB37	Process in progress
0xBB38	Trying to connect non-existenting TCP server socket/ Connection got terminated by the server.
0xBB3E	Error in length of the command(Exceeds number of characters is mentioned in the PRM).

Error Codes (in hexadecimal format)	Description
0xBB42	Socket is still bound
0xBB45	No free port
0xBB46	Invalid port
0xBB4B	Feature not supported
0xBB50	Socket is not in connected state. Disconnected from server. In case of FTP, user need to give destroy command after receiving this error
0xBB87	POP3 session creation failed/ POP3 session got terminated
0xBB9C	DHCPv6 Handshake failure
0xBB9D	DHCP invalid IP response
0xBBA0	SMTP Authentication error
0xBBA1	No DNS server was specified, SMTP over size mail data
0xBBA2	SMTP invalid server reply
0xBBA3	DNS query failed, SMTP internal error
0xBBA4	Bad DNS address, SMTP server error code received
0xBBA5	SMTP invalid parameters
0xBBA6	SMTP packet allocation failed
0xBBA7	SMTP GREET reply failed
0xBBA8	Parameter error, SMTP Hello reply error
0xBBA9	SMTP mail reply error
0xBBAA	SMTP RCPT reply error
0xBBAB 0xBBA1	Empty DNS server list, SMTP message reply errorNo DNS server was specified
0xBBAC 0xBBA3	SMTP data reply errorDNS query failed.
0xBBAD 0xBBA4	SMTP authentication reply errorBad DNS address
0xBBAE 0xBBA8	SMTP server error replyParameter error
0xBBAF 0xBBAB	DNS duplicate entry.Empty DNS server list.
0xBBB1 0xBBAF	SMTP oversize server replyDNS duplicate entry.

Error Codes (in hexadecimal format)	Description
0xBBB2	SMTP client not initialized
0xBBB3	DNS IPv6 not supported
0xBBC5	Invalid mail index for POP3 mail retrieve command
0xBBD2	SSL handshake failed
0xBBD3	FTP client is not connected or disconnected with the FTP server
0xBBD4	FTP client is not disconnected
0xBBD5	FTP file is not opened or SSL ERROR - RX length exceeded
0xBBD6	SSL handshake timeout or FTP file is not closed
0xBBD9	Expected 1XX response from FTP server but not received
0xBBDA	Expected 2XX response from FTP server but not received
0xBBDB	Expected 22X response from FTP server but not received
0xBBDC	Expected 23X response from FTP server but not received
0xBBDD	Expected 3XX response from FTP server but not received
0xBBDE	Expected 33X response from FTP server but not received
0xBBE1	HTTP Timeout
0xBBE2	HTTP Failed
0xBBE7	HTTP Timeout for HTTP PUT client or HTTP End of content data
0xBBEB	Authentication Error
0xBBED	Invalid packet length, content length and received data length is mismatching
0xBBEE	Expected 2XX response from HTTP server but not received
0xBBEF	Server responds before HTTP client request is complete
0xBBF0	HTTP/HTTPS password is too long
0xBBFF	POP3 error for invalid mail index
0XFFFF	Listening TCP socket in module is not connected to the remote peer, or the LTCP socket is not yet opened in the module
0xFFFB	Cannot create IP in same interface in concurrent mode
0xFFFE	Sockets not available. The error comes if the Host tries to open

Error Codes (in hexadecimal format)	Description
	more than 10 sockets
0xFFFC	IP configuration failed
0xFFF7	Byte stuffing error in AT mode
0xFFF8	1) Invalid command (e.g. parameters insufficient or invalid in the command). Invalid operation (e.g. power save command with the same mode given twice, accessing wrong socket, creating more than allowed sockets)
0XFFFA	2) TCP socket is not connected
0xFFC5	Station count exceeded max station supported
0xFFC4	Unable to send tcp data
0xFFBC	Socket buffer too small
0xFFBB	Invalid content in the DNS response to the DNS Resolution query
0xFFBA	DNS Class error in the response to the DNS Resolution query
0xFFB8	DNS count error in the response to the DNS Resolution query
0xFFB7	DNS Return Code error in the response to the DNS Resolution query
0xFFB6	DNS Opcode error in the response to the DNS Resolution query
0xFFB5	DNS ID mismatch between DNS Resolution request and response
0xFFAB	Invalid input to the DNS Resolution query
0xFF42	DNS response was timed out
0xFFA1	ARP request failure
0xFF9D	DHCP lease time expired
0xFF9C	DHCP handshake failure
0xFF75	Static IP address which is already in use by some other device(IP Conflict error)
0xFF88	This error is issued when Websocket creation failed
0xFF87	This error is issued when module tried to connect to a non-existent TCP server socket on the remote side
0xFF86	This error is issued when tried to close non-existent socket. or invalid socket descriptor
0xFF85	Invalid socket parameters
0xFF82	Feature not supported

Error Codes (in hexadecimal format)	Description
0xFF81	Socket already open
0xFF80	Attempt to open more than the maximum allowed number of sockets
0XFF7E	Data length exceeds mss.
0xFF74	Feature not enabled
0xFF73	DHCP server not set in AP mode
0xFF71	Error in AP set region command parameters
0xFF70	SSL not supported
0xFF6F	JSON not supported
0xFF6E	Invalid operating mode
0xFF6D	Invalid socket configuration parameters
0xFF6C	Web socket creation timeout
0xFF6B	Parameter maximum allowed value is exceeded
0xFF6A	Socket read timeout
0xFF69	Invalid command in sequence
0xFF42	DNS response timed out
0xFF41	HTTP socket creation failed
0xFF40	TCP socket close command is issued before getting the response of the previous close command
0xFF36	Wait On Host feature not enabled
0xFF35	Store configuration checksum validation failed
0xFF33	TCP keep alive timed out
0xFF2D	TCP ACK failed for TCP SYN-ACK
0xFF2C	Memory limit exceeded in a given operating mode
0xFF2A	Memory limit exceeded in operating mode during auto join/create
0xFF2B	MDNSD command not supported
0xCC2F	PUF Operation is blocked
0xCC31	PUF Activation code invalid
0xCC32	PUF input parameters invalid
0xCC33	PUF in error state
0XCC34	PUF Operation not allowed
0XCC35	PUF operation Failed

Table 33: Error Codes

10 SPI Host Interface Mode

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

1) Host sends 0x0015 to module.
2) On successful initialization module sends 0x58(SPI success) to host else module sends 0x54(SPI busy) or 0x52(SPI failure).

10.1 Operations through SPI

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

10.1.1 Tx Operation

10.1.2 The Host uses Tx operations:

- To send management commands to the module from the Host
- To send actual data to the module, to be transmitted onto the air.

Host should follow the steps below to send the command frames to the Module:

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

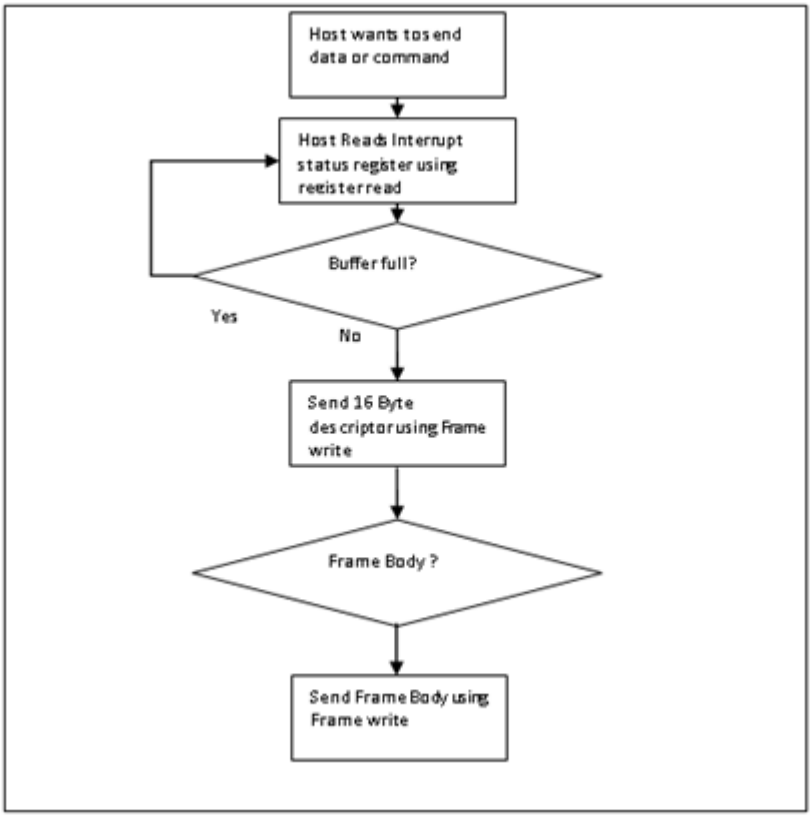


Figure 48: Tx From Host to Module

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

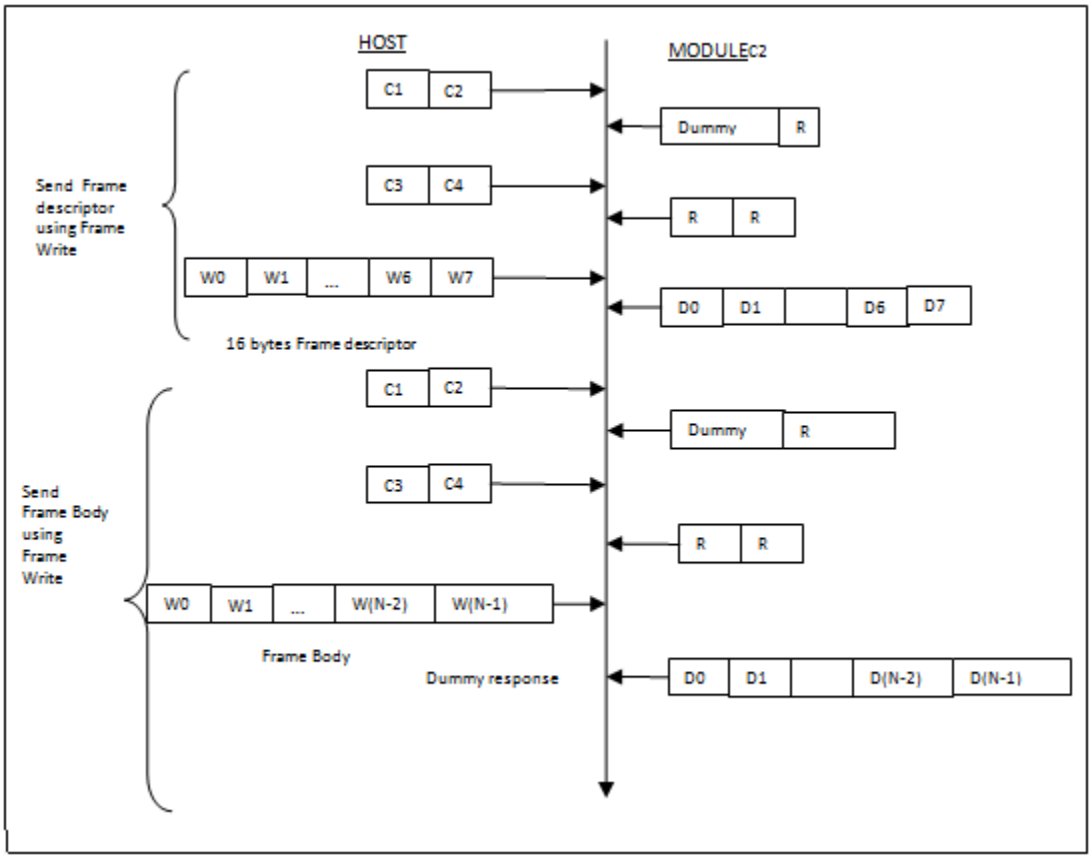


Figure 49: Exchanges between Host and Module for Tx operation

10.1.3 Rx Operation

The Host uses this operation:

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

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

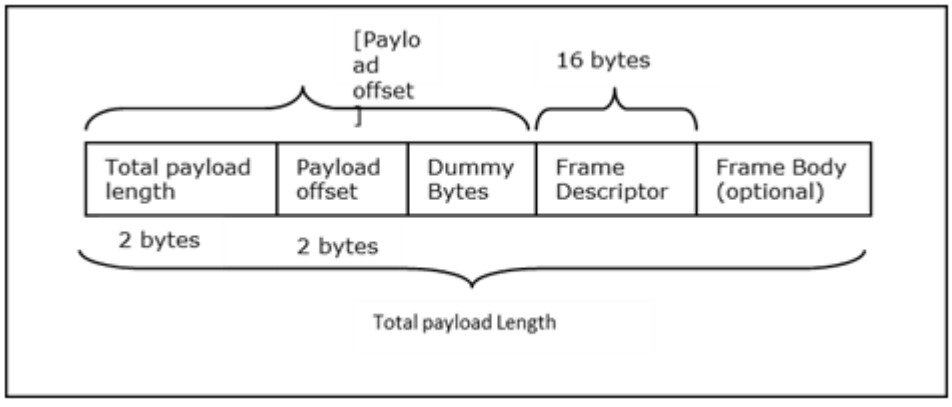


Figure 50: RX Frame format

Note:

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

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

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

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

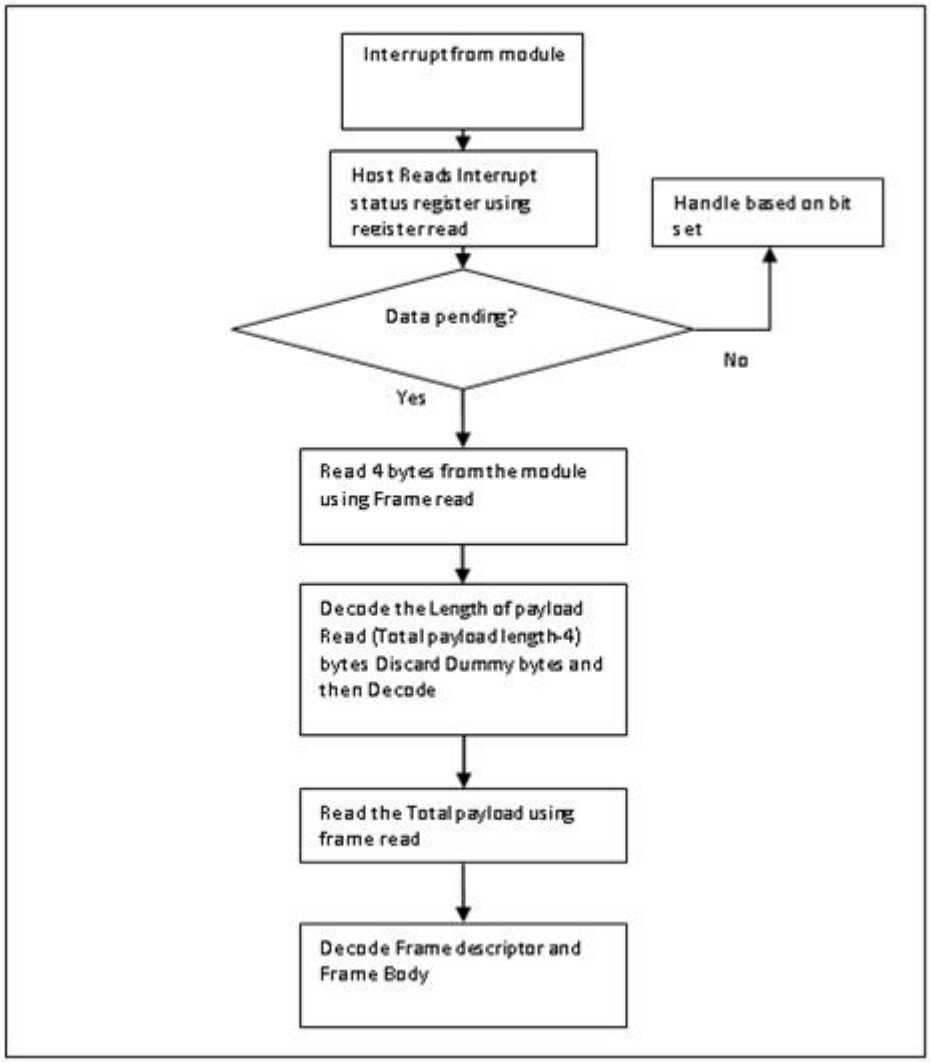


Figure 51: RX Operation from Module to Host

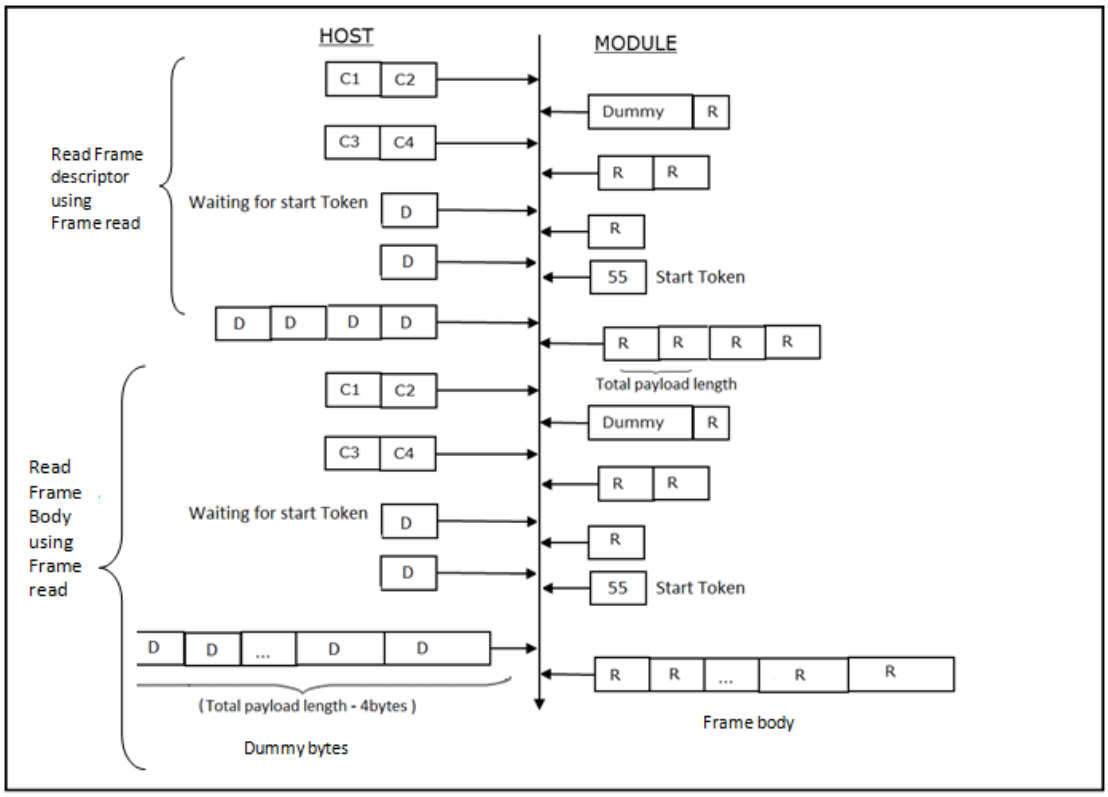


Figure 52: Message exchanges between Host and Module for Rx operation

11 USB Host Interface Mode

This section explains the procedure to follow to configure and the send Wi-Fi commands to the module and receive response from the module using USB.

R9113_WiSeConnect Module supports two modes using USB interface.

- USB mode
- USB CDC mode

11.1 USB mode

In USB mode all Wi-Fi command frame formats (Frame Descriptor), Command ID's/response ID's and Error codes are exactly same as Wi-Fi SPI commands.

RS9113-WiSeConnect module USB interface support 2 endpoints:

- Control endpoint : control endpoint used during enumeration process.
- Bulk endpoint : Bulk endpoint used to send/receive data between host and module through USB interface.

In USB mode command/data packets transfer from host to module(Tx packet) required headroom as shown in below figure:

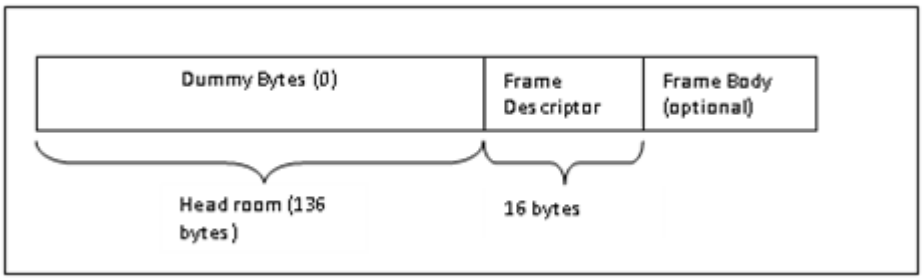


Figure 53: Command/Data Packet format from host to module in USB mode

In receive path first 4 bytes contain Total packet length and descriptor offset. Frame Descriptor starts at descriptor offset location from packet start.

Note:

Head room for transfer packet from host to module is 136 byte

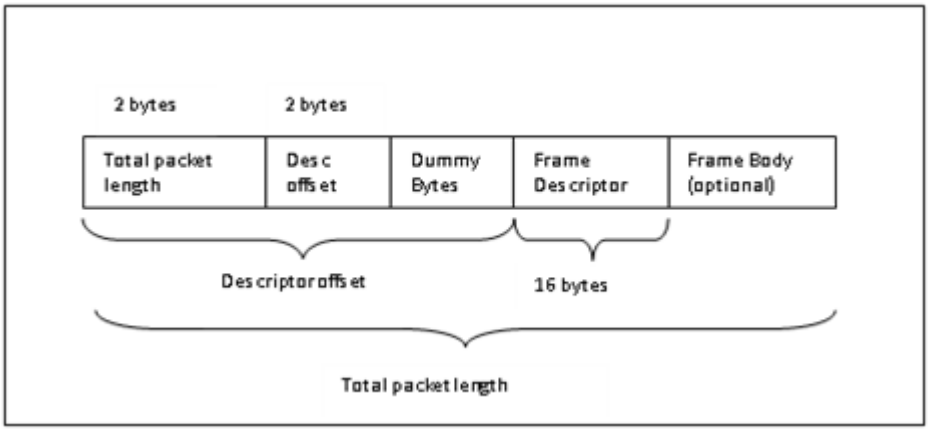


Figure 54: Command/Data Packet format from module to host in USB mode

11.1.1.1 Operations through USB interface:

This section explains the procedure to be followed by the host to send Wi-Fi command frame to module and to receive response from the module.

Tx operation :

Following are sequence of steps to be followed to send command frame to module through USB interface.

- Prepare command frame with headroom as shown in figure 51: Command/Data Packet format from host to module in USB mode.
- Forward packet to module.

Rx operation:

Following are sequence of steps to follow to receive response from module .

Send an empty buffer from host .

After receiving packet from module, process the Frame Descriptor and Frame Body accordingly.

11.1.2 USB CDC-ACM mode

The USB interface in the mode corresponds to the CDC-ACM class and presents itself as a USB Device to the Host USB. In order to communicate with the module the user should install a driver file (provided with the software package) in the Host .

USB CDC-ACM mode usage:

A sample flow is provided below to use the module with a PC' s USB interface.

1. Connect the module's USB port to USB interface of the PC. The PC prompts for installing the USB-CDC driver. Install the driver file from RS9113.WC.GENR.x.x.x\utils\usb_cdc\rsi_usbcdc.inf. The file needs to be installed only once.
2. Power cycle the module. Check the list in "Ports" in the *Device Manager* Settings of the PC. It should show the device as "RSI WSC Virtual Com Port.

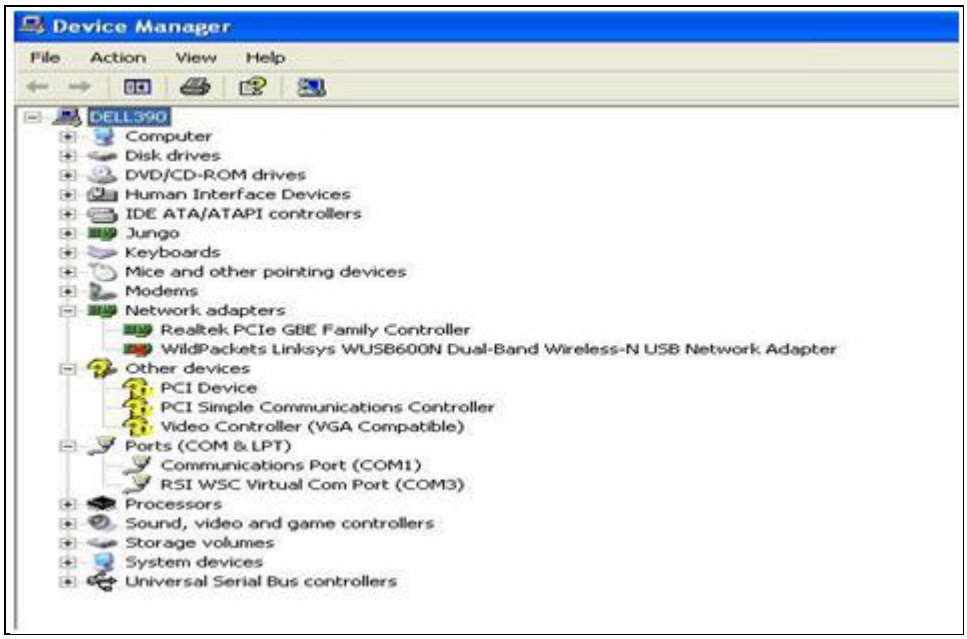


Figure 55: Device Manager

3. Open HyperTerminal and set Flow Control to “None”. Baud rate, Data bits,
4. Parity and Stops bits are “Don’t care” fields in USB mode. Now AT commands can be issued to communicate with the module through the virtual Com port. The behavior of the module, commands, command responses, error codes and sequence of commands are exactly same as in the UART mode. The USB interface of the module supports the full speed USB mode (12 Mbps physical data rate).

12 Using Different Wi-Fi Operation Modes

The module can be configured in the following modes :

- Wi-Fi Direct™ mode
- Access Point Mode
- Client Mode to connect to an AP in open mode or with Personal Security
- Client mode to connect to an AP with Enterprise Security
- PER mode

12.1 Wi-Fi Direct Mode

Wi-Fi Direct™ is a standard that enables two Wi-Fi devices to connect and communicate to one another without an Access Point in between. The technology allows seamless and direct peer-to-peer communication between a RS9113-WiSeConnect module and a variety of hand-held devices such as smart phones, tablet PCs etc. The flow diagram below shows scenarios of setting up Wi-Fi Direct nodes with a RS9113-WiSeConnect Wi-Fi Direct network. In this mode, the module connects to a Wi-Fi direct node by following the below mentioned steps. The module can either act as a Group Owner or a client. “GO Negotiation” is the phase when this is decided. The decision of which node becomes the group owner depends on the value of Group_Owner_intent. The node with a higher value of Group_Owner_intent would get preference over a lower value in becoming a GO. If the values advertised by both nodes are same, then a tie-break sequence is automatically initiated to resolve contention. A Group Owner Wi-Fi Direct node behaves as an Access Point to the client Wi-Fi Direct Peer-to-Peer (P2P) nodes. It acts as a DHCP server to dispatch IP addresses to the P2P nodes.

12.2 Access Point Mode

The following sequence of commands should be used to create an Access Point in the module. The module can support eight external clients when it is configured in Access Point mode. By default the module can act as a DHCP server.

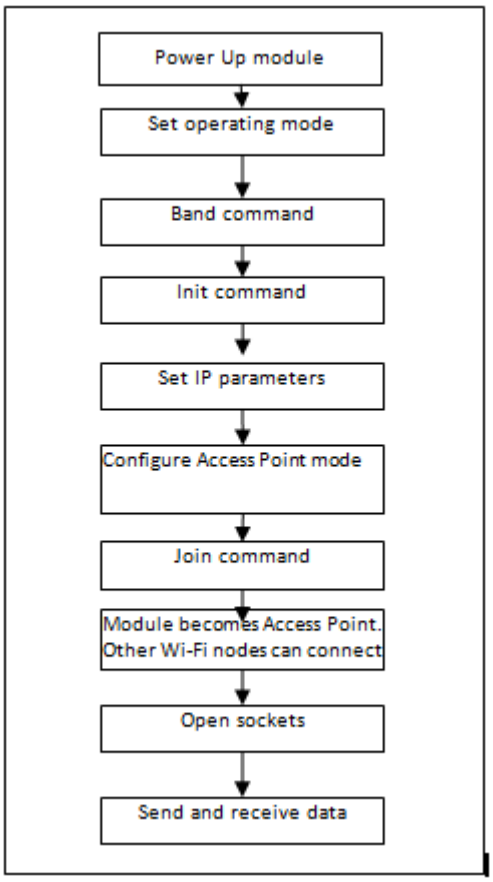


Figure 56: Access Point Mode

12.3 Client Mode with Personal Security

In this mode, the module works as a Wi-Fi client. It can connect to an Access Point with open mode or Personal Security.

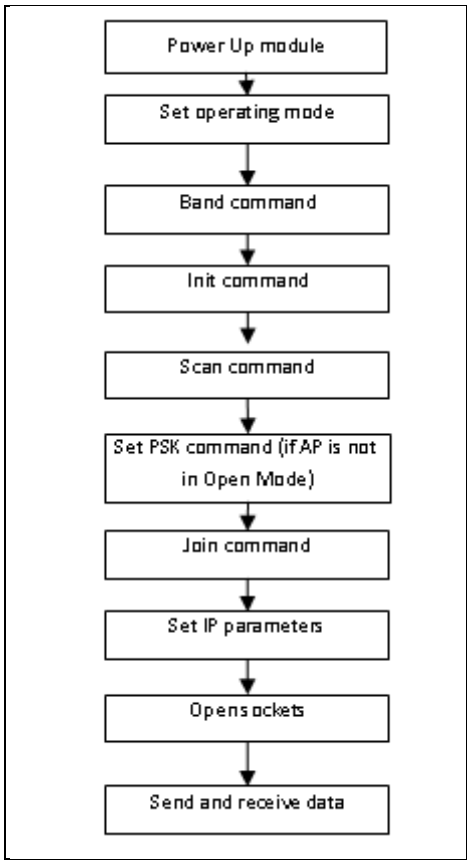


Figure 57: Client Mode with Personal Security

12.4 Client Mode with Enterprise Security

In this mode, the module works as a client to connect to an Enterprise security enabled network that Hosts a Radius Server.

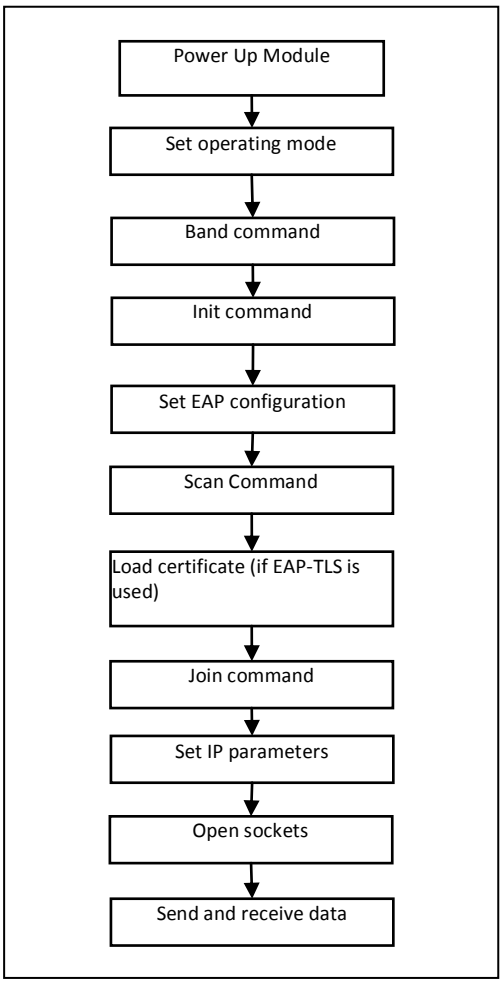


Figure 58: Client Mode with Enterprise Security

12.5 PER Mode

This mode is used for regulatory tests (FCC, IC, ETSI, TELEC etc) or Packet Error Rate analysis.

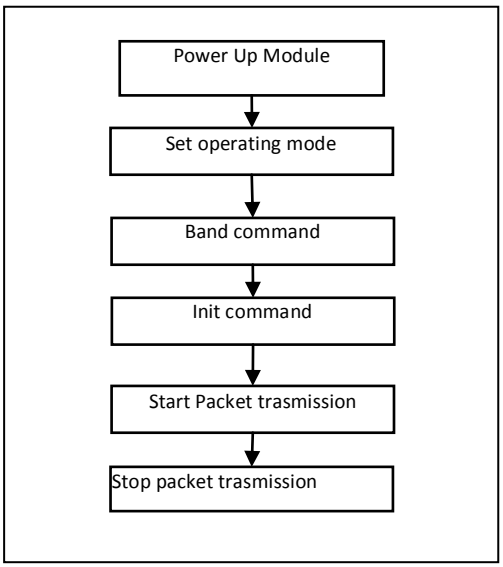


Figure 59: PER Mode

13 Wireless Configuration

The module can be configured wirelessly to join a specific AP (referred to as “auto-connect”) or create an Access Point (referred to as “auto-create”).

13.1 Configuration to Join a Specific AP

Flow 1: In this flow, an AP is first created in the module, to which a remote device connects and configures the module.

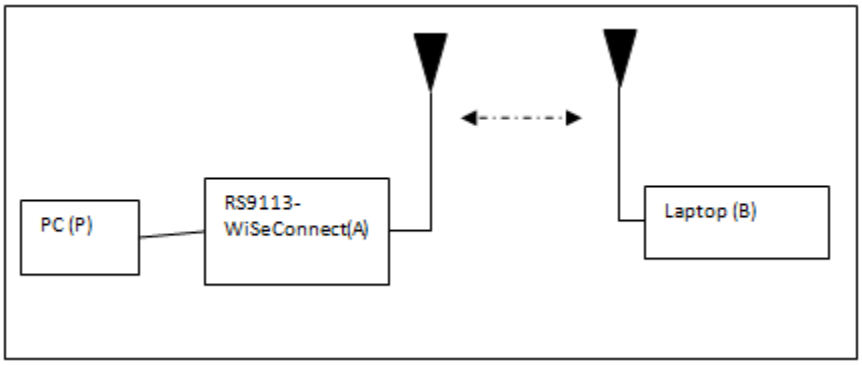


Figure 60: Setup for Configuration to Join a Specific AP – Flow 1

1. Connect a PC or Host to the module through the any interface and power up the module.
2. Configure the module to become an AP by issuing commands from PC (P) (refer APPENDIX A: Sample flow of commands for Wi-Fi over UART).
3. Connect a Laptop (B) to the created AP. Open the URL http://<Module’s IP address> in the Laptop. For example, if the module was configured to have an IP of 192.168.2.5, then the URL is http://192.168.2.5. Make sure the browser in the laptop does not have any proxies enabled. This will open the following index page:
4. Give the login credentials username as “redpine” and password as “admin” and click on ok button to make changes in configurations .
5. In the opened web page , select “Client mode” and enter desired values.
   - SSID: This is the SSID of the AP to which the module should connect after configuration is over.
   - Band : Single band (2.4 GHz or 5.0 GHz) or dual band (2.4Ghz and 5GHz).
   - Tx Power: RF power for Tx (refer to the TxPower parameter in command Join) . Allowed values are 0, 1 and 2.
   - Channel: Channel number at which the target AP is present. Refer to the PER Mode command section for more details.
   - Security Enable: This should match the security mode of the AP to which the module should connect.
   - IPversion: select IPv4/IPv6/Dual stack mode.
   - DHCP: If DHCP is selected, the module will work as a DHCPv4 client, otherwise, an IPv4 address should be hard coded in the web page.

- IPv6 DHCP: If IPv6 DHCP is selected, the module will work as a DHCPv6 client, and otherwise, an IPv6 address should be hard coded in the web page.
- Enable optional features like Dynamic web ages, HTTP client, SNMP client, DNS Client, PING,SSL Feature select bit maps based on features used. Refer Set Operating Mode command for further details.

Click on "Submit" button. The information is sent to the module and stored in its internal flash.

6. The module should now be power cycled or hard reset. It boots up and then automatically scans channels for the target AP and connects to it and gets an IP address. The module will send out two responses to the Host, the first corresponds to the "Join" command and the second to the "Set IP Parameters" command. Note that once the module is restarted, no commands need to be given. The module automatically scans and joins the target AP, after which the stored configuration parameters can be retrieved using the command Get Information about Stored Configuration. If the auto-connect feature needs to be disabled, issue the command Enable auto-join to AP or Auto-create AP to the module.

Flow 2: In this flow, the module is connected to an AP. A remote device connects to the same AP and configures the module.

1. Connect a PC or Host to the module through the any interface and power up the module.
2. Configure the module to become a client and connect to an AP, by issuing commands from PC (P) (refer APPENDIX A: Sample flow of commands for Wi-Fi over UART.

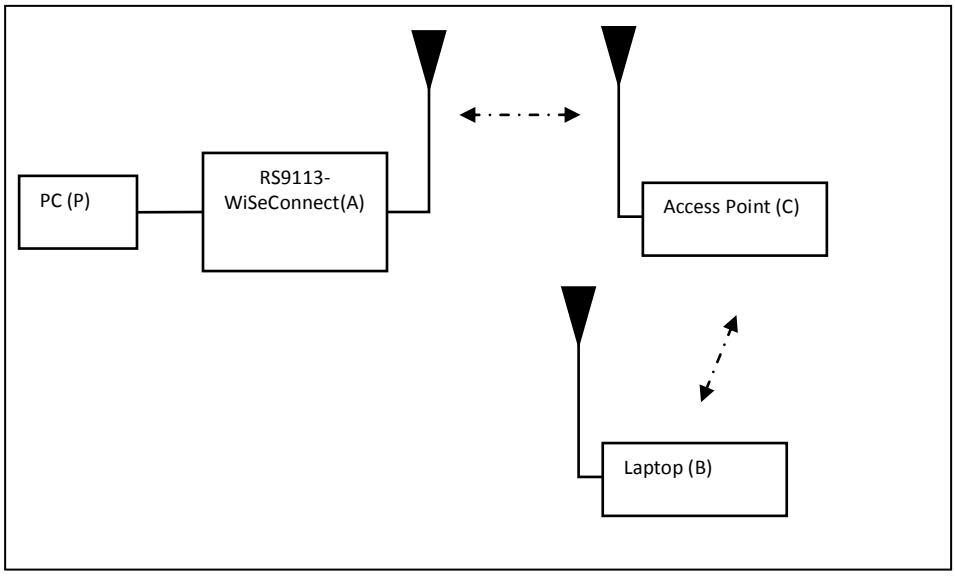


Figure 61: Setup for Configuration to Join a Specific AP – Flow 2

3. Connect a Laptop (B) to the same AP. Open the URL http://<Module's IP address> in the Laptop. For example, if the module was configured to have an IP of 192.168.2.5, then the URL is http://192.168.2.5. Make sure the browser in the laptop does not

have any proxies enabled. This will open the index page as shown in Flow 1 above give the credentials username as “redpine” and password as “admin”.

4. In the opened web page , select “Client mode” and enter desired values.
   - SSID: This is the SSID of the AP to which the module should connect after configuration is over.
   - Band: Single Band (2.4GHz or 5GHz) or Dual Band (2.4 GHz and 5.0GHz).
   - Tx Power: RF power for Tx (refer to the TxPower parameter in command Join) . Allowed values are 0, 1 and 2.
   - Security Enable: This should match the security mode of the AP to which the module should connect.
   - DHCP: If DHCP is selected, the module will work as a DHCP client, otherwise, an IP should be hard coded in the web page.
   - Channel: Channel number at which the target AP is present. Refer to the PER Mode command section for more details.

Figure 62: Webpage Screenshot

   - Security Enable: Select if security mode enable and fill the PSK, Security Type, encryption Type fields accordingly.
   - IPconfiguration: Select the IP version (IPv4/IPv6/Both) and provide static IP details or enable DHCP.
5. Enable optional features Dynamic WebPages, HTTP client, SNMP client, DNS Client, Feature select bit maps based on features used. Refer operation mode command for further details.
6. Click on “Submit” button. The information is sent to the module and stored in its internal flash.
7. The module should now be power cycled or hard reset. It boots up and then automatically scans channels for the target AP and connects to it and gets an IP address. The module will send out two responses to the Host, the first corresponds to the “Join” command and the second to the “Set IP Parameters” command. Note that once the module is restarted, no commands need to be given. The module

automatically scans and joins the target AP, after which the stored configuration parameters can be retrieved using the command Get Information about Stored Configuration. If the auto-connect feature needs to be disabled, issue the command Enable auto-join to AP or Auto-create AP to the module.

13.2 Configuring to Create an AP

Flow 1: In this flow, an AP is first created in the module, to which a remote device connects and configures the module.

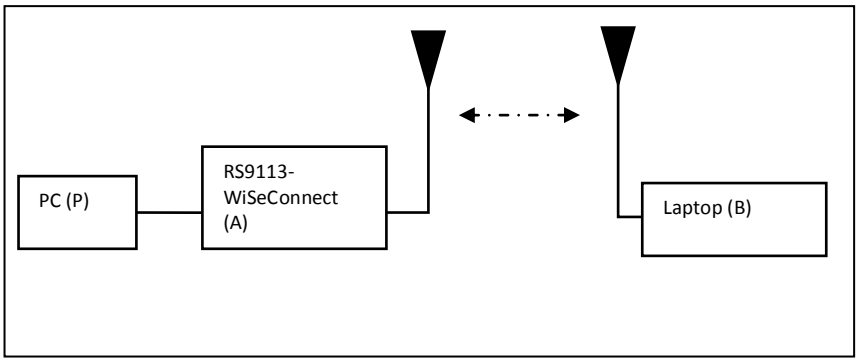


Figure 63: Setup for Configuration to Create an AP – Flow 1

1. Connect a PC or Host to the module through the any interface and power up the module.
2. Configure the module to become an AP by issuing commands through PC (P). (refer APPENDIX A: Sample flow of commands for Wi-Fi over UART).
3. Connect a Laptop (B) to the created AP. Open the URL http://<Module's IP address> in the Laptop. For example, if the module was configured to have an IP of 192.168.2.5, then the URL is http://192.168.2.5 Make sure the browser in the laptop does not have any proxies enabled. This will open the index page as shown in section 6.1 Flow 1 give the credentials username as "redpine" and password as "admin".
4. In the web page that opens, select "Access Point" mode and enter desired values.
   - SSID: This is the SSID of the AP which will be created after configuration is over.
   - Band: Single Band (2.4GHz or 5.0GHz).
   - Tx Power: RF power for Tx (refer to the TxPower parameter in command Join) . Allowed values are 0, 1 and 2.
   - Security Enable: PSK, security type, encryption type. This is to configure the security mode of the AP.
   - Channel: Channel number at which the target AP is present. Refer to the PER Mode command section for more details. Value of '0' is not allowed.
   - Security Type:WPA/WPA2/Mixed
   - Encryption type: Type of encryption(TKIP/AES).
   - IP, Mask, and Gateway: These parameters set the IP parameters of the AP.
   - Beacon Interval and DTIM interval: This to set the beacon parameters of the AP. For example, if beacon interval is 200 (msecs) and DTIM count is 3, the DTIM interval would be 2x300=600 msecs.

5. Enable optional features like aggregation Dynamic WebPages, HTTP client, SNMP client, DNS Client, Feature select bit maps based on features used. Refer operation mode command for further details.

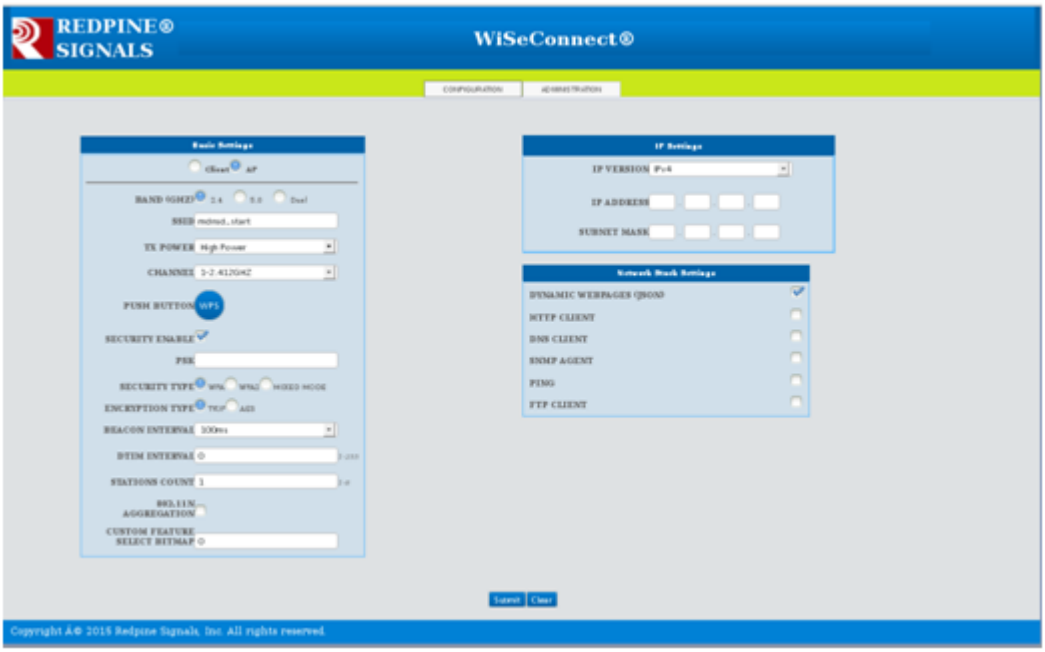


Figure 64: Webpage Screenshot

6. Click on “Submit” button. The information is sent to the module and stored in its internal flash.
7. The module should now be power cycled or hard reset. It boots up and then automatically creates an AP with the configured parameters. The module will send out two responses to the Host, the first corresponds to the“Set IP Parameters” command and the second to the “Join” command. Note that once the module is restarted, no commands need to be given. The module automatically and internally executes the commands to create an AP. The stored configuration parameters can be retrieved using the command Get Information about Stored Configuration. If the auto-connect feature needs to be disabled, issue the command Enable auto-join to AP or Auto-create AP to the module.

Flow 2: In this flow, the module is connected to an AP. A remote device connects to the same AP and configures the module.

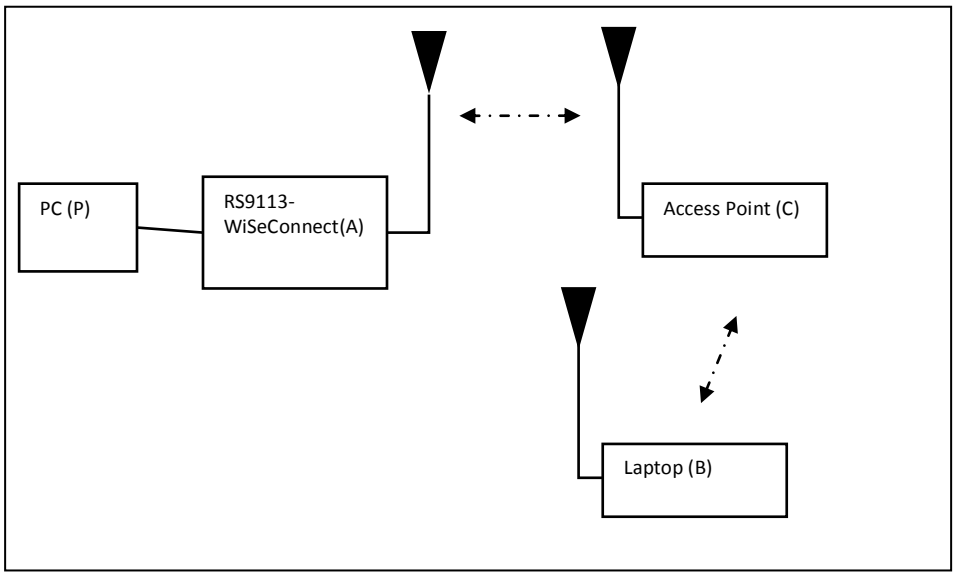


Figure 65: Setup for Configuration to Create an AP – Flow 2

1. Connect a PC or Host to the module through the any interface and power up the module.
2. Configure the module to become a client and connect to an AP by issuing commands from the PC (P) (refer APPENDIX A: Sample flow of commands for Wi-Fi over UART).
3. Connect a Laptop (B) to the created AP. Open the URL http://<Module's IP address> in the Laptop. For example, if the module was configured to have an IP of 192.168.2.5, then the URL is http://192.168.2.5 Make sure the browser in the laptop does not have any proxies enabled. Give the credentials username as "redpine" and password as "admin".
4. In the web page that opens, select "Access Point" mode and enter desired values.
   - SSID: This is the SSID of the AP which will be created after configuration is over.
   - Band: Single band (2.4GHz) or Dual Band (5GHz).
   - Tx Power: RF power for Tx (refer to the TxPower parameter in command Join) . Allowed values are 0, 1 and 2.
   - Security Enable: PSK, security type, encryption type: This is to configure the security mode of the AP.
   - Channel: Channel number at which the target AP is present. Refer to the PER Mode command section for more details. Value of '0' is not allowed.
   - IP, Mask, and Gateway: These parameters set the IP parameters of the AP.
   - Beacon Interval and DTIM count: This to set the beacon parameters of the AP. For example, if beacon interval is 200 (msecs) and DTIM count is 3, the DTIM interval would be 2x300=600 msecs.
5. Enable optional features like Aggregation, Dynamic WebPages, HTTP client, SNMP client, DNS Client, Feature select bit maps based on features used. Refer operation mode command for further details.

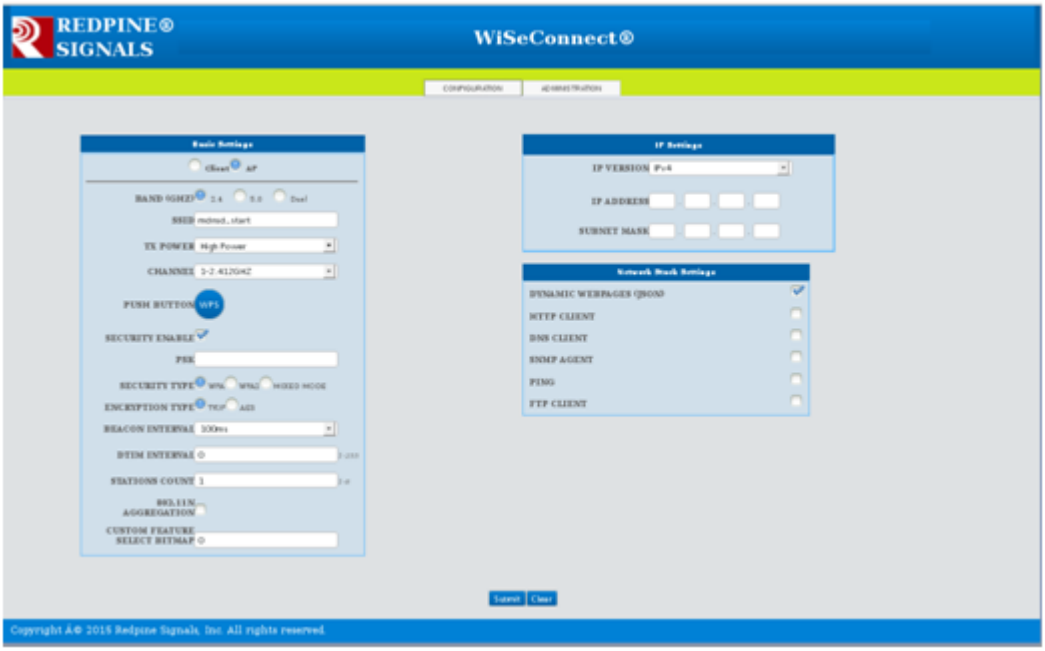


Figure 66: Webpage Screenshot

6. Click on “Submit” button. The information is sent to the module and stored in its internal flash.
7. The module should now be power cycled or hard reset. It boots up and then automatically creates an AP with the configured parameters. The module will send out two responses to the Host, the first corresponds to the “Set IP Parameters” command and the second to the “Join” command. Note that once the module is restarted, no commands need to be given. The module automatically and internally executes the commands to create an AP. The stored configuration parameters can be retrieved using the command Get Information about Stored Configuration. If the auto-connect feature needs to be disabled, issue the command Enable auto-join to AP or Auto-create AP to the module.

13.3 Configuration to Join an AP with Enterprise Security

Flow 1: In this flow, an AP is first created in the module, to which a remote device connects and configures the module.

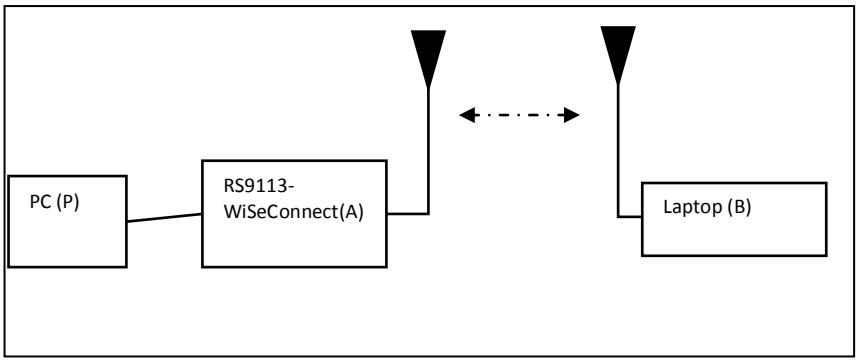


Figure 67: Setup for Configuration to Join an AP with Enterprise Security – Flow 1

1. Connect a PC or Host to the module through the any interface and power up the module.
2. Configure the module to become an AP by issuing commands from PC (P) (refer APPENDIX A: Sample flow of commands for Wi-Fi over UART).
3. Connect a Laptop (B) to the created AP. Open the URL http://<Module's IP address> in the Laptop. For example, if the module was configured to have an IP of 192.168.2.5, then the URL is http://192.168.2.5 Make sure the browser in the laptop does not have any proxies enabled. Give the credentials username as "redpine" and password as "admin".
4. In the web page that opens, select "Enterprise" in security mode and enter desired values.
   - SSID: This is the SSID of the AP to which the module should connect after configuration is over.
   - Band: Single Band(2.4GHz or 5GHz) or Dual Band(5GHz and 2.4GHz).
   - Tx Power: RF power for Tx (refer to the TxPower parameter in command Join) . Allowed values are 0, 1 and 2.
   - Security type : This should match the security mode of the AP to which the module should connect.
   - DHCP: If DHCP is selected, the module will work as a DHCP client, otherwise, an IP should be hard coded in the web page.
   - Channel: Channel number at which the target AP is present. Refer to the PER Mode command section for more details.
   - EAP: EAP method of the target AP.
   - Inner method: Inner method of the target AP.
   - User identity: User ID. This is present in the user configuration file in the radius sever.
   - Password: This should be same as the password in the user configuration file in the Radius Server for that User Identity.
5. Enable optional features like Aggregation, Dynamic WebPages, HTTP client, SNMP client, DNS Client, Feature select bit maps based on features used. Refer operation mode command for further details.

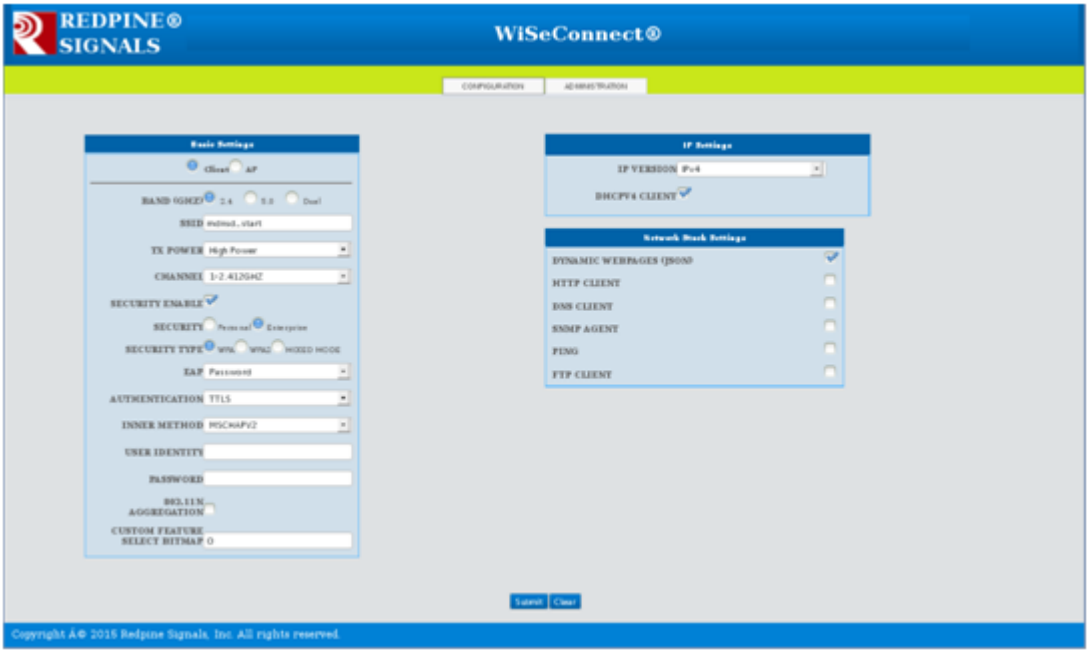


Figure 68: Webpage Screenshot

6. Click on “Submit” button. The information is sent to the module and stored in its internal flash.
7. The module should now be power cycled or hard reset. It boots up and then automatically scans channels for the target AP and connects to it and gets an IP address. The module will send out two responses to the Host, the first corresponds to the “Join” command and the second to the “Set IP Parameters” command. Note that once the module is restarted, no commands need to be given. The module automatically scans and joins the target AP, after which the stored configuration parameters can be retrieved using the command Get Information about Stored Configuration. If the auto-connect feature needs to be disabled, issue the command Enable auto-join to AP or Auto-create AP to the module.

Flow 2: In this flow, the module is connected to an AP. A remote device connects to the same AP and configures the module

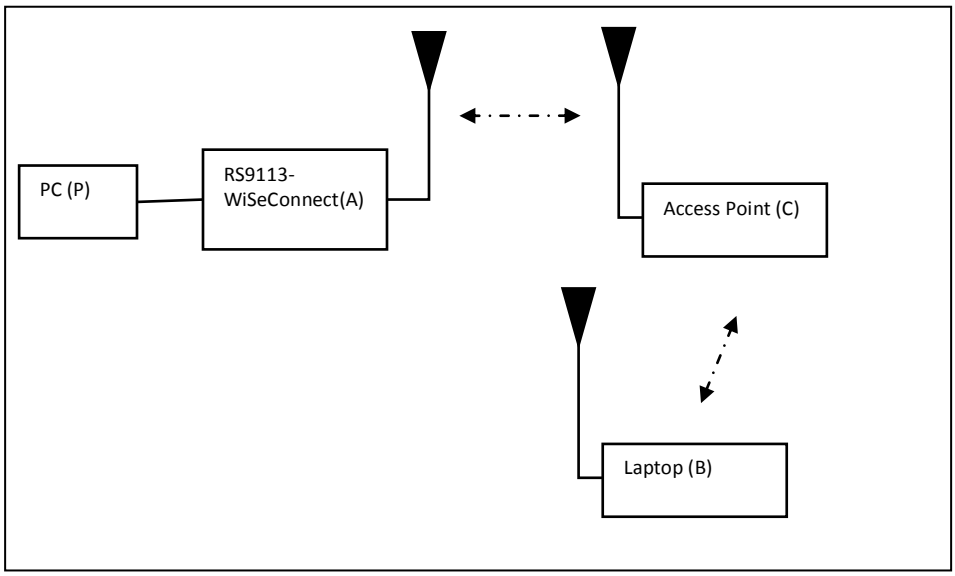


Figure 69: Setup for Configuration to Join an AP with Enterprise Security – Flow 2

1. Connect a PC or Host to the module through the any interface and power up the module.
2. Configure the module to become a client and connect to an AP, by issuing commands from PC (P) (refer APPENDIX A: Sample flow of commands for Wi-Fi over UART).
3. Connect a Laptop (B) to the same AP. Open the URL http://<Module's IP address>in the Laptop. For example, if the module was configured to have an IP of 192.168.2.5, then the URL is http://192.168.2.5. Make sure the browser in the laptop does not have any proxies enabled. Give the credentials username as "redpine" and password as "admin".
4. In the web page that opens, select "Client mode" and enter desired values.
   - SSID: This is the SSID of the AP to which the module should connect after configuration is over.
   - Data rate: Physical data rate (refer to the PER Mode command section for more details).
   - Tx Power: RF power for Tx (refer to the TxPower parameter in command Join) . Allowed values are 0, 1 and 2.
   - Security mode and PSK: This should match the security mode of the AP to which the module should connect.
   - DHCP: If DHCP is selected, the module will work as a DHCP client, otherwise, an IP should be hard coded in the web page.
   - Channel: Channel number at which the target AP is present. Refer to the PER Mode command section for more details.
   - EAP: EAP method of the target AP.
   - Inner method: Inner method of the target AP.
   - User identity: User ID. This is present in the user configuration file in the radius sever.

- Password: This should be same as the password in the user configuration file in the Radius Server for that User Identity.

5. Enable optional features like Aggregation, Dynamic WebPages, HTTP client, SNMP client, DNS Client, Feature select bit maps based on features used. Refer operation mode command for further details.

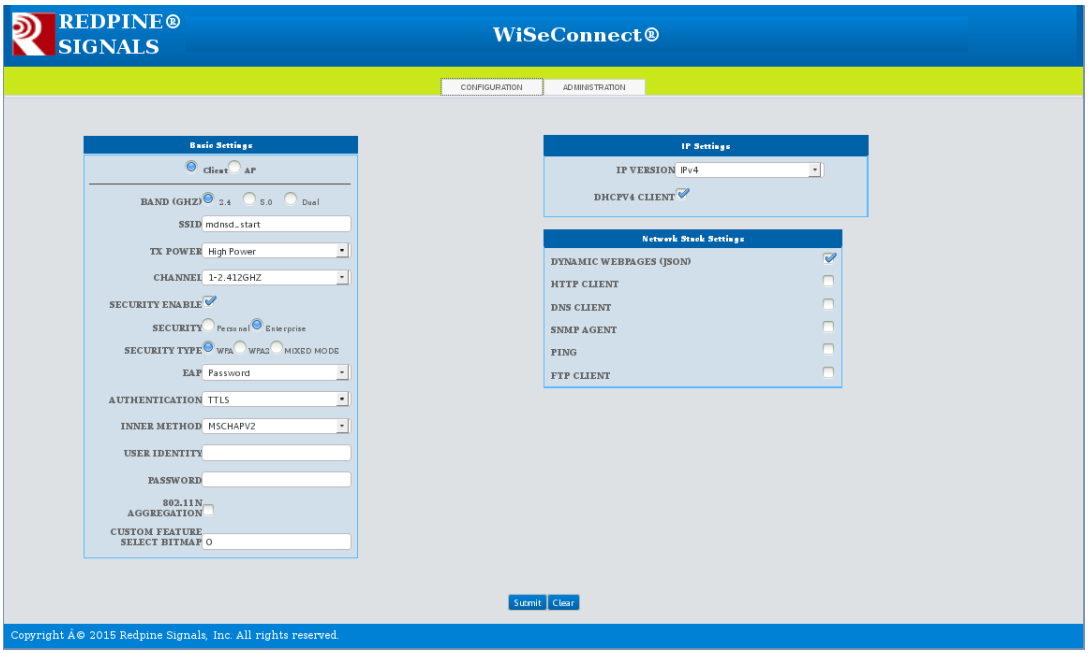


Figure 70: Webpage Screenshot

6. Click on “Submit” button. The information is sent to the module and stored in its internal flash.
7. The module should now be power cycled or hard reset. It boots up and then automatically scans channels for the target AP and connects to it and gets an IP address. The module will send out two responses to the Host, the first corresponds to the “Join” command and the second to the “Set IP Parameters” command. Note that once the module is restarted, no commands need to be given. The module automatically scans and joins the target AP, after which the stored configuration parameters can be retrieved using the command Get Information about Stored Configuration. If the auto-connect feature needs to be disabled, issue the command Enable auto-join to AP or Auto-create AP to the module.

14 Wireless Firmware Upgrade

The firmware of the module can be upgraded wirelessly through web server. To upgrade the firmware wirelessly user has to open configuration page. In the given example, module is in WLAN client mode. Some other host and module connected to an AP. Module got IP 192.168.2.5.When opened the module's webpage on the other host, it asks for the login credentials. The credentials for Username should be given as "redpine" and password should be given as "admin" to open the modules configuration page.

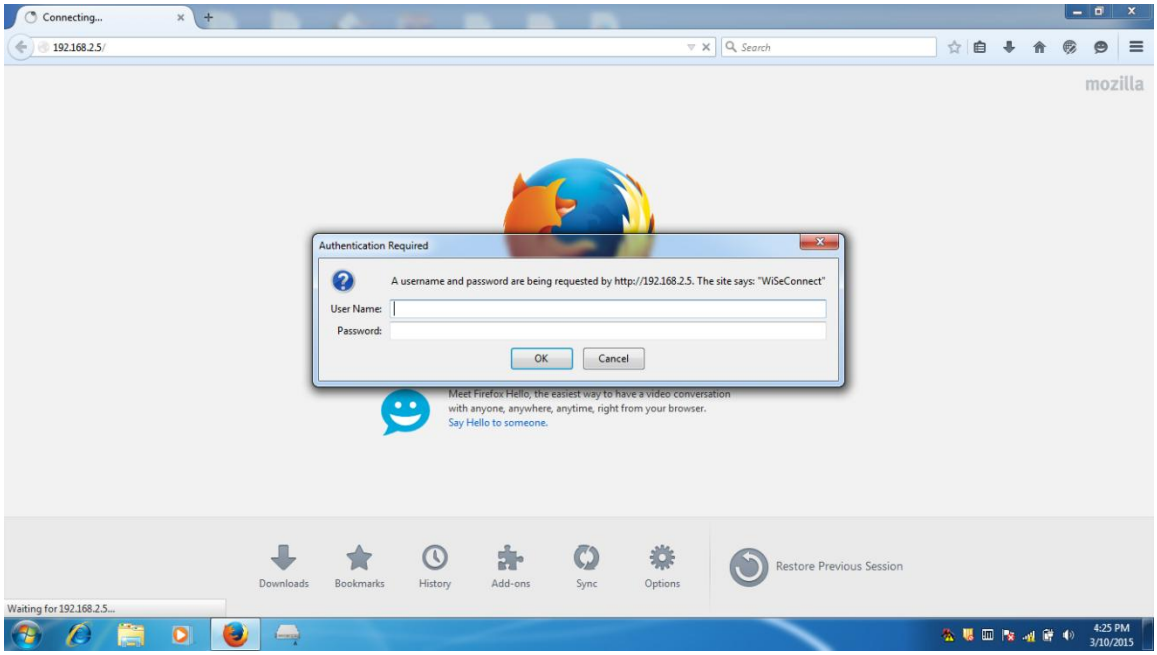


Figure 71: Login Credentials

After giving the login credentials, the module's configuration page is opened as shown in the figure below.

Note:

'Authentication Required' pop up window will only appear if BIT[23] in Custom feature bitmap is enabled.

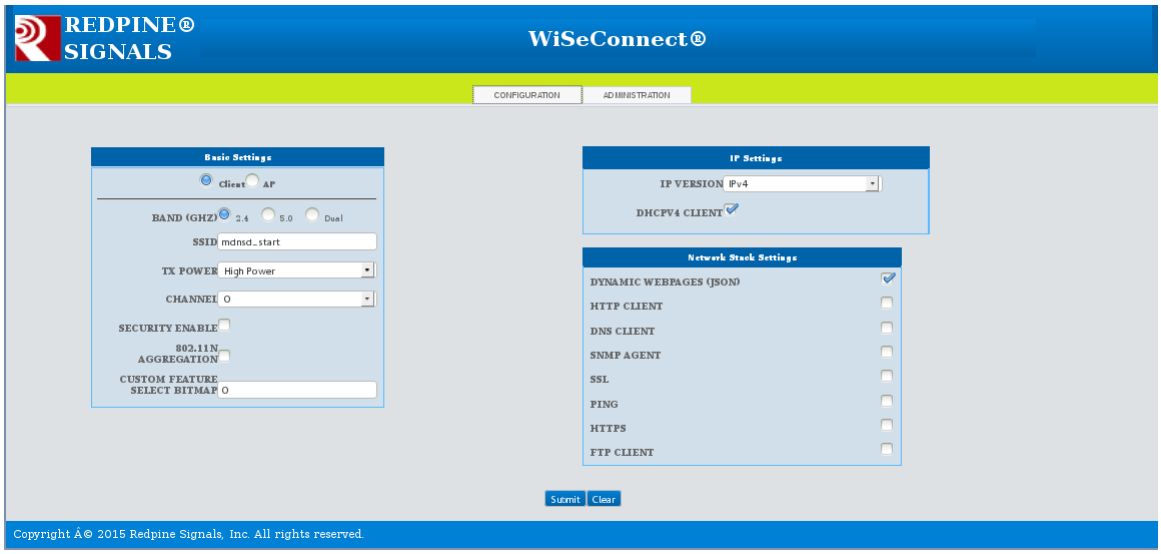


Figure 72: Configuration Page

Click on ADMINISTRATION button to go to the wireless firmware upgradation page. Browse for the rps file(RS9113.WC.GEN.OSI.x_x_x.rps) on the host to upgrade the module firmware, and click on UPGRADE, as shown in the below screen shot.

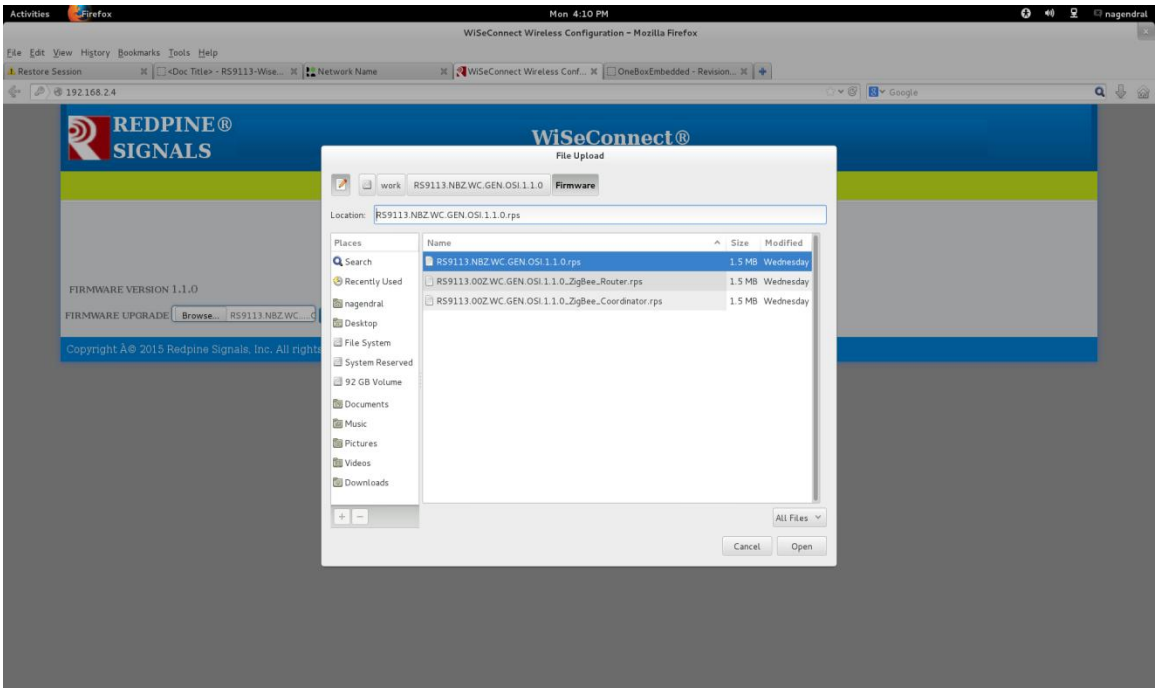


Figure 73: Browse for RPS File

Once the remote peer has pressed upgrade button on the webpage, if module is connected through UART or USB-CDC interface host will get asynchronous notification as“AT+RSI_FWUPREQ”. So host has to issue AT+RSI_FWUPOK. For SPI and USB interfaces, an asynchronous message with response id 0x59 is sent to host and then host has to

respond with request id 0x59 (through API) . If host failed to reply within specified timeout(~20 seconds), request will expire and upgradation process terminates.

After the firmware upgraded successful, one popup will come on remote peer screen to intimate the process complete . Similarly asynchronous success message(for UART/USB-CDC "AT+RSI_FWUPSUCCESS" and for SPI/USB response id 0x5A) will be forward to host connected with the module.

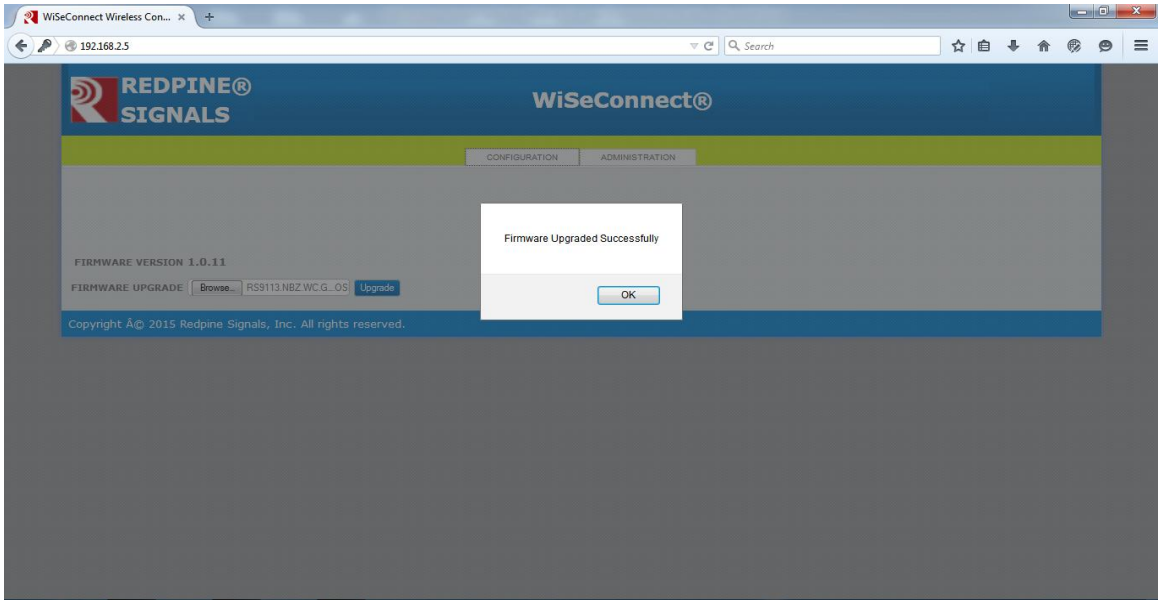


Figure 74: Firmware Upgraded Successfully

Note:

1) Wireless firmware upgradation is supported only for the latest versions of Firefox and Google chrome.
2) When user clicks on upgrade button, module starts erasing flash for storing image. This may take few seconds,then up gradation automatically will start.
3) After wireless firmware upgrade, after reboot user need to wait for few minutes(~ 1.5 minutes) so that bootloader will copy upgraded image into actual flash location.

After the firmware upgraded failure, one popup will come on remote peer screen to intimate the process is failed . Similarly asynchronous failure message(for UART/USB-CDC "AT+RSI_FWUPFAILED" and for SPI/USB response id 0x5A) will be forward to host connected with the module.

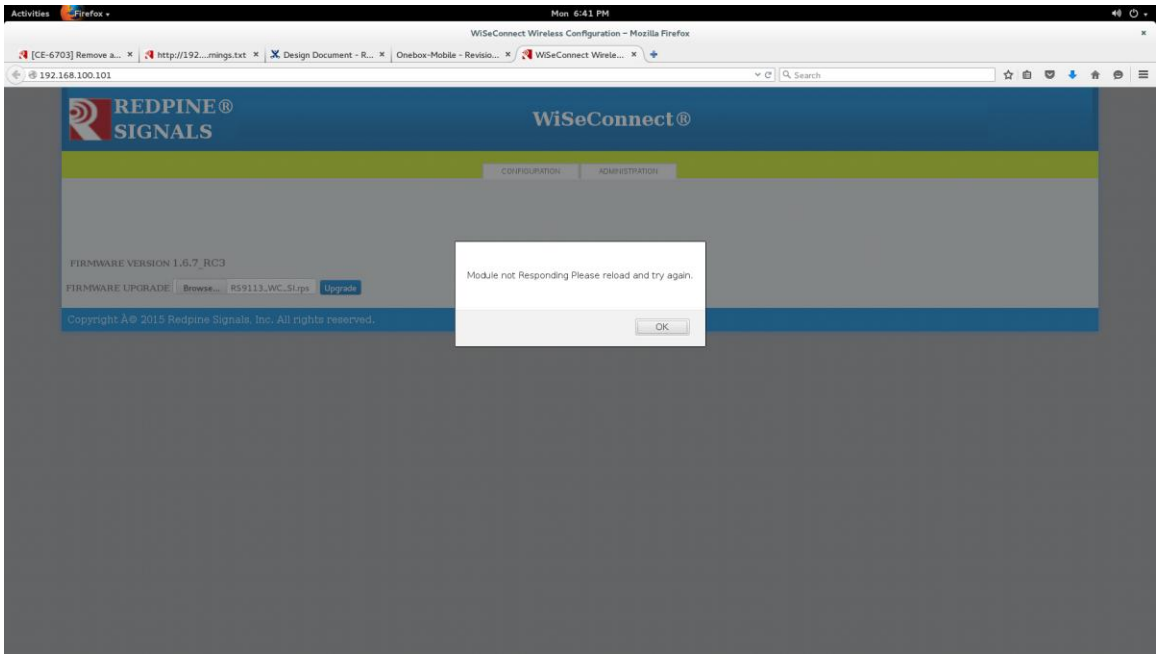


Figure 75: Module Not Responding

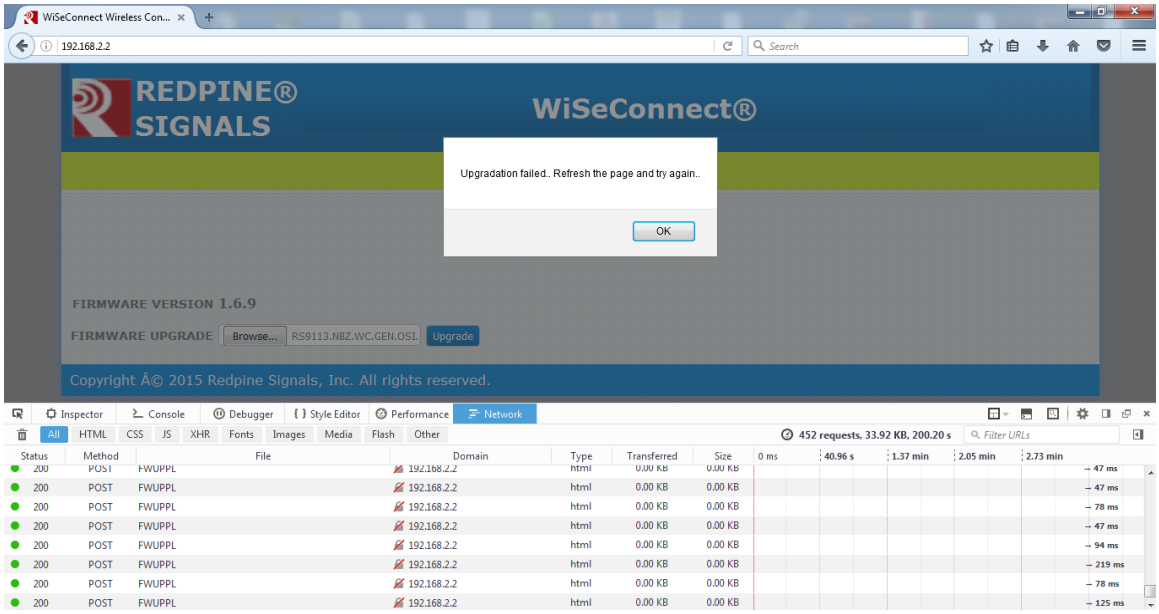


Figure 76: Upgradation failed

Note:

1) Firmware upgrade will be failed, if server(module) didn't receive packets from client(browser) within timeout i.e 40seconds.

2) Firmware upgrade failed message will come, if server/client closes the TCP connection.

15 Wake on Wireless

Whenever RS9113 module want to send packets to host, it will de-assert GPIO-2 pin.

In UART mode following is the sequence:

- RS9113 module de-assert GPIO-2 Pin when data pending from module and poll for ACK (GPIO-15 high) before start the transfer.
- After recognizing GPIO-2 low, host is required to ACK the request from module by asserting GPIO-15 and poll GPIO-2 Pin for module confirmation (GPIO-2 high).
- After recognizingACK (GPIO-15 high) from host, module will assert GPIO-2 pin and start transfer.
- Once GPIO-2 is asserted, host should de-assert GPIO-15 pin and receive the packet from module.

Note:

1. Host required toset BIT(11) in custom feature bit map of opermode command to enable this feature in UART mode.

2. This feature will work only in UART mode.

In other host interface modes, after completion of packet transfer to host, RS9113 module will assert the GPIO-2 pin automatically.

16 Appendix A: Sample flow of commands for Wi-Fi over UART

Sample command sequences are shown below for operating the module in different modes.

Operate in Wi-Fi Direct Mode to associate to a Wi-Fi Direct Phone

```
at+rsi_opermode=1,0,20,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_wfd=0,directrp,6,redpine,12345678\r\n
```

This command starts the Wi-Fi Direct mode of the module. The first parameter in this command is called the Group_Owner_Intent. It gives the willingness of the module to become a Group Owner. It has been set to the lowest value of 0 in this case. The module responds with “OK”.

After issuing this command, the module starts scanning for Wi-Fi Direct devices, and reports any that are found through the asynchronous message AT+RSI_WFDDEV

```
at+rsi_join=AndroidP2P,0,2,0\r\n
```

This command initiates the association operation between the module and the Wi-Fi Direct phone. The device name of the Wi-Fi Direct phone in this example is “AndroidP2P8031”. It is assumed that the phone has become a Group Owner.

```
at+rsi_ipconf=1\r\n
```

The module will acquire an IP address from the phone. A ping can be issued from the phone to the module.

For exchanging data between the module and the Wi-Fi Direct Phone, an application may be written by the user at the mobile phone to open sockets and transfer or receive data. Sockets at the module can be created by using one of the socket related commands. For example,

```
at+rsi_ltcp=5001,5,0\r\n
```

opens a server TCP socket inside the module with port number 5001 with tos(type of service) type 0 and max clients support count as ‘5’

To send a test string “This is a test” from the module to the remote node, issue the below command

```
at+rsi_snd=2,14,0,0,This is a test\r\n
```

If the remote node sends data, the module receives the data and transfers to the Host with a ***AT+RSI_READ*** message. The first parameter (value 2) is the socket_handle of the socket in the module. Refer to the section for at+rsi_ltcp for more details.

Create an Access Point

```
at+rsi_opermode=6,1,48,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_ipconf=0,192.168.50.1,255.255.255.0,192.168.50.1\r\n
```

This command can be used optionally in this flow to configure the IP (192.168.50.1 in this example) of the AP. If this command is not issued, a default IP of 192.168.100.76 will be used

```
at+rsi_apconf=1,REDPINE,2,2,12345678,300,2,4\r\n
```

This command will configure the SSID of the AP to “REDPINE” and password will be set to “12345678”.

```
at+rsi_join=REDPINE,0,2 \r\n
```

This command will create the Access Point with SSID redpine where xy is a pair of alphanumeric character.

A client device (Named “Device A” in this example) can now associate to the AP, open sockets and transfer data.

For example,

```
at+rsi_ltcp=5001,5,0\r\n
```

opens a server TCP socket inside the module with port number 5001 with tos(type of service) type 0 and max clients support count as 5.

A client socket at the remote node (Device A) can connect to the server socket.

To send a test string “This is a test” from the module to the remote node, issue the below command

```
at+rsi_snd=1,14,0,0,This is a test\r\n
```

If the remote node (Device A) sends data, the module receives the data and transfers to the Host with a **AT+RSI_READ** message

Another client (Named “Device B” in this example) can also connect to the Access Point in the module and data transfer can be executed between Device A and Device B through the AP. A maximum of 4 clients are supported.

Associate to an Access Point (with WPA2-PSK security) as a client

```
at+rsi_opermode=0,1,4,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_scan=0\r\n
```

This command scans for Aps and reports the Aps found.

Note:

After a scan, if the user want to join an AP as enterprise client then user need to issue a soft reset first and then follow the flow of commands as in "Associate to an Enterprise Security enabled Access Point as a client"

```
at+rsi_psk=1,12345678\r\n
```

This command configures the PSK to be used to associate to the Access Point.

```
at+rsi_join=Test_AP,0,2,2\r\n
```

This command associates the module to the AP. It is assumed that the SSID of the AP is Test_AP with WPA2-PSK security key of 12345678.

```
at+rsi_ipconf=1,0,0,0\r\n
```

This configures the IP address of the module in DHCP mode.

```
at+rsi_dnsserver=1,0,0\r\n
```

Optional command to provide the IP address of a DNS server.

```
at+rsi_dnsget=<domain_name>,1\r\n
```

Optional command to resolve IP of a given domain name.

```
at+rsi_ltcp=5001,5,0\r\n
```

opens a server TCP socket inside the module with port number 5001 with tos(type of service) type 0 and max clients support count as 5.

Now connect another client (called "Device A" in this example) to the same Access Point and open a client socket to bind to the module's socket.

To send a test string "This is a test" from the module to the remote node, issue the below command

```
at+rsi_snd=1,14,0,0,This is a test\r\n
```

If the remote node (Device A) sends data, the module receives the data and transfers to the Host with a ***AT+RSI_READ*** message

Associate to an Access Point (with WEP security) as a client

```
at+rsi_opermode=0,2,4,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_scan=0\r\n
```

This command scans for APs and reports the APs found.

```
at+rsi_wepkey=0,ABCDE12345,0,0,0\r\n
```

This command configures the PSK to be used to associate to an Access Point.

```
at+rsi_join=Test_AP,0,2,3\r\n
```

This command associates the module to the AP.

```
at+rsi_ipconf=1,0,0,0\r\n
```

This configures the IP address of the module in DHCP mode.

```
at+rsi_ltcp=5001,5,0\r\n
```

opens a server TCP socket inside the module with port number 5001 with tos(type of service) type 0 and max clients support count as 5.

Now connect another client (called “Device A” in this example) to the same Access Point and open a client socket to bind to the module’s socket.

To send a test string “This is a test” from the module to the remote node, issue the below command

```
at+rsi_snd=1,14,0,0,This is a test\r\n
```

If the remote node (Device A) sends data, the module receives the data and transfers to the Host with a AT+RSI_READ message

Associate to a WPS enabled Access Point

```
at+rsi_opermode=0,2,4,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_scan=0\r\n
```

This command scans for available Aps and reports the Aps found.

```
at+rsi_join=,0,2\r\n
```

This command associates the module to the AP using WPS push button method. Note that we have to send null in place of ssid .

```
at+rsi_ipconf=1,0,0,0\r\n
```

This command configures the IP address of the module in DHCP mode.

```
at+rsi_ltcp=5001,5,0\r\n
```

opens a server TCP socket inside the module with port number 5001 with tos(type of service) type 0 and max clients support count as 5.

Now connect another client (called “Device A” in this example) to the same Access Point and open a client socket to bind to the module’s socket.

To send a test string “This is a test” from the module to the remote node, issue the below command

```
at+rsi_snd=1,14,0,0,This is a test\r\n
```

If the remote node (Device A) sends data, the module receives the data and transfers to the Host with a AT+RSI_READ message.

Associate to an Enterprise Security enabled Access Point as a client

The example demonstrates the flow for EAP-TLS mode.

```
at+rsi_opermode=2,0,4,0\r\n
```

This sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_eap=TLS,MSCHAPV2,user1,password1\r\n
```

This command sets the Enterprise mode.

```
at+rsi_scan=0\r\n
```

This command scans for Aps and reports the Aps found.

Note:

After a scan, if the user want to join an AP with WPA2-AES PSK then user need to issue a soft reset first and then follow the flow of commands as in “Associate to an Access Point (with WPA2-PSK security) as a client”

```
at+rsi_cert=1,cert_len,key_password,<TLS certificate>\r\n
```

This command provides the TLS certificate to the module

```
at+rsi_join=Test_AP,0,2,4\r\n
```

This command associates the module to the AP. It is assumed that the SSID of the AP is Test_AP.

```
at+rsi_ipconf=1,0,0,0\r\n
```

This command configures the IP address of the module in DHCP mode.

```
at+rsi_ltcp=5001,5,0\r\n
```

opens a server TCP socket inside the module with port number 5001 with tos(type of service) type 0 and max clients support count as 5.

Now connect another client (called “Device A” in this example) to the same Access Point and open a client socket to bind to the module’s socket.

To send a test string “This is a test” from the module to the remote node (Device A), issue the below command

```
at+rsi_snd=1,14,0,0,This is a test\r\n
```

If the remote node (Device A) sends data, the module sends the received data with a AT+RSI_READ message to the Host.

PER Mode command flow in Burst mode

```
at+rsi_opermode=8\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_per=1,18,139,30,0,1,0,0,0,0\r\n
```

This command sends the packets in burst mode in channel number 1, with packet length 30, with power 18, with data rate 139.

PER Mode command flow in Continuous mode

```
at+rsi_opermode=8\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_per=1,18,139,30,1,1,0,0,0,0\r\n
```

This command sends the packets in continuous mode in channel number 1, with packet length 30, with power 18, with data rate 139.

```
at+rsi_per=0\r\n
```

To stop transmission.

Enabling BG scan in open mode as a client

```
at+rsi_opermode=0,1,4,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

```
This command initializes the module.
```

```
at+rsi_scan=0\r\n
```

This command scans for available APs in the channels mentioned in channel bitmap and reports the APs found.

```
at+rsi_join=test,0,2,0\r\n
```

This command associates the module to the AP in open mode.

```
at+rsi_bgscan=1,1,10,4,10,15,20,1\r\n
```

This command enables the back ground scan functionality in module. Here instant bgscan is enabled hence module will send probe requests in the air on to the channels mentioned in the channel bitmap and results will go to host.

```
at+rsi_ipconf=1,0,0,0\r\n
```

This command configures the IP address of the module in DHCP mode.

Enabling Roaming in open mode as a client

```
at+rsi_opermode=0,1,4,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_scan=0\r\n
```

This command scans for available APs in the channels mentioned in channel bitmap and reports the APs found.

```
at+rsi_join=test,0,2,0\r\n
```

This command associates the module to the AP in open mode.

```
at+rsi_bgscan=1,1,10,4,10,15,20,1\r\n
```

This command enables the back ground scan functionality in module. Here instant bgscan is enabled hence module will send probe requests in the air and results will go to host and in background scan will be continued based upon the parameters given in the command.

```
at+rsi_roam_params= 1,5,2\r\n
```

This command enables roaming in module.

```
at+rsi_ipconf=1,0,0,0\r\n
```

This command configures the IP address of the module in DHCP mode.

Enabling WMM PS in open mode as a client

```
at+rsi_opermode=0,1,4,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_scan=6,test\r\n
```

This command scans for available AP in the given channels with the given SSID and reports if the is AP found.

```
at+rsi_wmm_config=1,0,0,1\r\n
```

This command enables the WMM feature in the module.

```
at+rsi_join=test,0,2,0\r\n
```

This command associates the module to the AP in open mode.

```
at+rsi_pwmode=1\r\n
```

This command configures the power save mode 1 in the module.

```
at+rsi_ipconf=1,0,0,0\r\n
```

This command configures the IP address of the module in DHCP mode.

Open a multicast IPv4 socket in client mode

```
at+rsi_opermode=0,1,4,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_scan=0\r\n
```

This command scans for APs and reports the APs found.

```
at+rsi_join=Test_AP,0,2,0\r\n
```

This command associates the module to the AP.

```
at+rsi_ipconf=1,0,0,0\r\n
```

This configures the IP address of the module in DHCP mode.

```
at+rsi_multicast=1,239.0.0.0\r\n
```

To join IPv4 multicast group with address 239.0.0.0.

```
at+rsi_ludp=5001\r\n
```

To open ludp socket with port number 5001.

```
at+rsi_multicast=0,239.0.0.0\r\n
```

To leave multicast group with address 239.0.0.0.

Open a SSL server socket in client mode

```
at+rsi_opermode=0,1,8,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_scan=0\r\n
```

This command scans for APs and reports the APs found.

```
at+rsi_join=Test_AP,0,2,0\r\n
```

This command associates the module to the AP.

```
at+rsi_ipconf=1,0,0,0\r\n
```

This command configures the IP address of the module in DHCP mode.

```
at+rsi_ltcp=5001,2,0,1,0\r\n
```

Opens a SSL server socket inside the module with port number 5001 with tos(type of service) type 0, with maximum 2 SSL clients support and with all SSL ciphers support.

To send a test string “This is a test” from the module to the remote node (Device A), issue the below command

```
at+rsi_snd=1,14,0,0,This is a test\r\n
```

If the remote node (Device A) sends data, the module sends the received data with an AT+RSI_READ message to the Host.

Operate in Concurrent Mode

```
at+rsi_opermode=9,1,20,0\r\n
```

This command sets the operating mode of the module.

```
at+rsi_band=0\r\n
```

This command sets the operating band of the module.

```
at+rsi_init\r\n
```

This command initializes the module.

```
at+rsi_scan=0\r\n
```

This command scans for AP's and reports the AP's found.

Note down the channel number of the desired AP which will be used in the apconf command.

```
at+rsi_psk=1,12345678\r\n
```

This command configures the PSK to be used to associate to the Access Point.

```
at+rsi_join=Test_AP,0,2,2,0,0,0\r\n
```

This command associates the module to the AP. It is assumed that the SSID of the AP is Test_AP with WPA2-PSK security key of 12345678

```
at+rsi_ipconf=1,0,0,0,0,0\r\n
```

This configures the IP address of the module in DHCP mode(StationMode)

```
at+rsi_ipconf=0,192.168.50.1,255.255.255.0,192.168.50.1,0,1\r\
n
```

This command can be used optionally in this flow to configure the IP (192.168.50.1 in this example) of the AP. If this command is not issued, a default IP of 192.168.100.76 will be used

```
at+rsi_apconf=1,REDPINE,2,2,12345678,300,2,4\r\n
```

This command will configure the SSID of the AP to “REDPINE” and password will be set to “12345678”. Assuming that Test_AP was in channel 1.

```
at+rsi_join=REDPINE,0,2,2,0,0,1\r\n
```

This command will create the Access Point with SSID REDPINE.

A client device (Named “Device A” in this example) can now associate to the AP, open sockets and transfer data.

```
at+rsi_ltcp=5001,2,0,0,0,0,0,0,0,1\r\n
```

opens a server TCP socket inside the module(AP mode) with port number 5001 with tos(type of service) type 0 and max clients support count as 2.

A client socket at the remote node can connect to the server socket running on Station mode.

```
at+rsi_ltcp=5002,2,0,0,0,0,0,0,0,0\r\n
```

opens a server TCP socket inside the module(Station mode) with port number 5002 with tos(type of service) type 0 and max clients support count as 2.

To send a test string “This is a test from AP mode socket” from the module(AP mode) to the remote node, issue the below command

```
at+rsi_snd=1,34,0,0,This is a test from AP mode socket\r\n
```

To send a test string “This is a test from Station mode socket” from the module(Station mode) to the remote node, issue the below command

```
at+rsi_snd=2,39,0,0,This is a test from Station mode
socket\r\n
```

17 Appendix B: Sample flow of APIs for Wi-Fi over SPI

Sample command sequences are shown below for operating the module in different modes.

Operate in Wi-Fi Direct Mode to associate to a Wi-Fi Direct Phone

Set Operating Mode: This command sets the operating mode of the module.

Band: This command sets the operating band of the module.

Init: This command initializes the module.

Configure Wi-Fi Direct Peer-to-Peer: This command starts the Wi-Fi Direct mode of the module. GOIntent equal to 0)

Join: This command initiates the association operation between the module and the Wi-Fi Direct phone.

Assuming that the phone has become a Group owner and the module has acquired an IP from phone,

Open Socket and transfer data

Create an Access Point

Set Operating Mode: This command sets the operating mode of the module.

Band: This command sets the operating band of the module.

Init: This command initializes the module.

Set IP Parameters: This command can be used optionally in this flow to configure the IP of the AP.

Configure AP Mode: This command will configure the parameters of the AP.

Join: This command will create the Access Point.

Open Socket and transfer data

Associate to an Access Point (with WEP security) as a client

Set Operating Mode: This command sets the operating mode of the module.

Band: This command sets the operating band of the module.

Init: This command initializes the module.

Scan: This command scans for Aps and reports the Aps found.

Set WEP Key: This command configures the PSK to be used to associate to the Access Point.

Join: This command associates the module to the AP.

Set IP Parameters: This configures the IP address of the module.

Open Socket and transfer data

Associate to a WPS enabled Access Point

Set Operating Mode: This command sets the operating mode of the module.

Band: This command sets the operating band of the module.

Init: This command initializes the module.

Scan: This command scans for available Aps and reports the Aps found.

Join: This command associates the module to the AP using WPS push button method. Note that we have tp pass the NULL string in case of station mode and ssid in case of access point mode .

Set IP Parameter: This command configures the IP address.

Open Socket and transfer data

Associate to an Enterprise Security enabled Access Point as a client

The example demonstrates the flow for EAP-TLS mode.

Set Operating Mode: This command sets the operating mode of the module.

Set Certificate: This command provides the TLS certificate to the module.

Band: This command sets the operating band of the module.

Init: This command initializes the module.

Set EAP Configuration: This command sets the Enterprise mode.

Scan: This command scans for Aps and reports the Aps found

Note:

After a scan, if the user want to join an AP with WPA2-PSK, then user need to issue a soft reset first and then follow the flow of commands as in “Associate to an Access Point (with WPA2-PSK security) as a client”

Join: This command associates the module to the AP.

Set IP Parameters: This command configures the IP address of the module.

Open Socket and transfer data.

Associate to an Access Point (with WPA2-PSK security) as a client, with TCP/IP stack bypassed in the module

Configure the Operating Mode to select station mode and TCP/IP bypass mode: This command sets the operating mode of the module.

Band: This command sets the operating band 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.

Configure Network Interface at the Host: Refer to the note in section Set Operating Mode.

Open Socket and transfer data

PER Mode

Set Operating Mode: This command sets the operating mode of the module.

Band: This command sets the operating band of the module.

Init: This command initializes the module.

PER Mode: This command sets the per mode in module to transmit packets into air.

IMPORTANT:

1) Commands should be given as per the protocol explained under each command. Wrong parameters passed for the command may cause the module to hang. Even though it is taken care for lot of commands, it is recommended to pass correct parameters.
2) User needs to do a hard reset, if the module goes to bad state because of any abnormal operations. This can be done by connecting a GPIO from host the module reset.

Enabling BG scan in open mode as a client

Set Operating Mode: This command sets the operating mode of the module.

Band: This command sets the operating band of the module.

Init: This command initializes the module.

Set channel bitmap: This command configures the channel bitmap for the channels in which

Scan has to be performed.

Scan: This command scans for Aps in the channels mentioned in channel bitmap and reports the Aps found.

Join: This command associates the module to the AP.

BG scan: This command enables background scan functionality in module.

Set IP Parameters: This configures the IP address of the module.

Enabling Roaming in open mode as a client

Set Operating Mode: This command sets the operating mode of the module.

Band: This command sets the operating band of the module.

Init: This command initializes the module.

Set channel bitmap: This command configures the channel bitmap for the channels in which

Scan has to be performed.

Scan: This command scans for Aps in the channels mentioned in channel bitmap and reports the Aps found.

Join: This command associates the module to the AP.

BG scan: This command enables background scan functionality in module.

Roaming params: This command enables Roaming functionality in module.

Set IP Parameters: This configures the IP address of the module.

Enabling WMM PS in open mode as a client

Set Operating Mode: This command sets the operating mode of the module.

Band: This command sets the operating band of the module.

Init: This command initializes the module.

WMM config: This command configures the WMM mode in module.

Scan: This command scans for Aps and reports the Aps found.

Join: This command associates the module to the AP.

Set Power mode: This command sets the power

Set IP Parameters: This configures the IP address of the module.

18 Appendix C: Links for BT, BLE and ZigBee Documentation

Paths for the documentation of BT,BLE and ZigBee are given below

Path for the BT PRM guide is given below

RS9113-WiseConnect-BT-Classic-Software-PRM-API-Guide-v1.7.9.pdf

Path for the BLE PRM guide is given below

RS9113-WiseConnect-BLE-Software-PRM-API-Guide-v1.7.9.pdf

Path for the ZigBee PRM guide is given below

RS9113-WiseConnect-ZigBee-Software-PRM-API-Guide-v1.7.9.pdf

*****

Revision History

Revision No.	Version No.	Date	Changes
Initial	0.1	Nov 2013	

1	0.2	March 2013	• Added USB Interface explanation.
			  ○ USB features (Section 2.2)
			  ○ USB commands (Section 4.8)
			• Added Soft reset command description in uart mode. (section 4.2.42)
			• Added ping request feature from module (section 4.2.25 and 4.6.39)
			• Added get statistics command/API feature (section 4.2.45 and 4.6.40)
			• Added selective scan feature description (Section 4.2.30)
			• Added setmac command/API description (section 4.2.3 and 4.6.3)
			• Added power mode description (section 4.2.12 and 4.6.26)
			• Added wireless firmware upgrade feature (section 6)
			• Added query SNR description (section 4.2.38 and 4.6.29)
			• Added DFS client (802.11h) feature description (section 4.2.35 and 4.6.36)
			• Added new error codes (0xFFFF, 0xFFFE and 0x FF82)
			• Added Request and Response ID for wireless firmware upgrade

2	1.0.8	June 2014	1.Added web page bypass section 2.Updated LTCP with maximum clients support description 3.Added port based socket close description
3	1.0.8	June 2014	Added section for set region.
4	1.0.9d	Sep 2014	Added DFS related channels, removed OT references, Added Bluetooth Profile command.
5	1.0.10di	Oct 2014	Splitted Uart command modes to AT and binary command

Revision No.	Version No.	Date	Changes
			modes Added additional Bootloader commands and modified command names Added additional fields to custom_feature_bit_map Added support for scanning DFS channels Notes added in many sections Added Debug prints on UART 2 section (4.6.59) In store configuration from user section, Csec_mode(1byte) is added(4.7.1.4) Added additional error codes for AT command mode and Binary mode(4.8) Modified payload structure for send data(4.6.35) Changed HTTP_GET(4.6.48), HTTP_HOST(4.6.49) parameters Added spi host interface mode section(4.9) Added Wake on wireless section(8) Removed Bluetooth Setting Section(8)
6	1.0.10ei	Oct 2014	Added additional fields to custom_feature_bit_map(4.2.1) Table 36: Channel Numbers and Frequencies for 20MHz and Table 37:Channel Number and Frequencies for 20MHz Channel Width in 5GHz Modified max_sta_support parameter in Configure AP mode(4.2.8) Added certificate erase in set certificate(4.2.20) section Modified Response in Get Information about Stored Configuration(4.3.1.3) Added opermode, custom_feature_bit_map and antenna select description in Store configuration from user(4.3.1.4) Added additional error codes in AT+Command Mode Error Codes(4.4) Modified Response Payload in Get information about stored configuration(4.7.1.3)
7	1.0.10fi	Oct 2014	Note added in wireless fw upgrade(4.2.42), Re-join(4.6.17) Added wireless firmware upgrade section(4.6.60)
8	1.0.10gi	Oct 2014	Added notes in AT+Command mode(4.2), Configure AP

Revision No.	Version No.	Date	Changes
			mode(4.2.8), Set certificate(4.2.20), Set IP parameters(4.2.22), Close socket(4.2.38), HTTP Get(4.2.50), HTTP Get6(4.2.51), HTTP Post(4.2.52), HTTP Post6(4.2.53), Close a socket(4.6.30). Added DHCP configuration in set IP parameters(4.2.22) Added fields in opermode in binary mode commands(4.6) Added field(hostname) in payload structure in set IP parameters(4.6.20)
9	1.0.10h	Oct 2014	Added IP version in Query a listening sockets Active connection status(4.2.37) Added number of bytes sent in response of close socket(4.2.38) Added bytes transmitted count on socket section(4.2.73) Added error codes in AT+command mode error codes (4.4) and binary mode error codes(4.8) Added bytes_sent field in response payload in close a socket(4.6.30), Remote socket closure(4.6.37) Added fields in response code in HTTP Post(4.6.49) Added bytes transmitted count on socket section(4.6.59) Added dtim_period in Get information about stored configuration(4.7.1.3) Added command mode selection section(9)
10	1.0.10i	Nov 2014	Removed 3 baud rates support for UART(2) Added deauth based roaming in custom_feature_bit_map(2.1, 5.7.1) Added notes for set certificate(2.20), set SNMP set(5.6.25) Added window size in TCP socket(2.31, 2.34, 6.38) Added error code(74) in AT+Command mode error codes(5.4) and Binary mode error codes(5.8)
11	1.0.10k	Nov 2014	Added error code (75) in AT+Command mode error codes(5.4) and Binary mode error codes(5.8) Added BIT(18) and BIT(19) bits of custom feature bit map as reserved and added supported inner methods for EAP configuration in sections 5.2.1, 5.3.1.3,

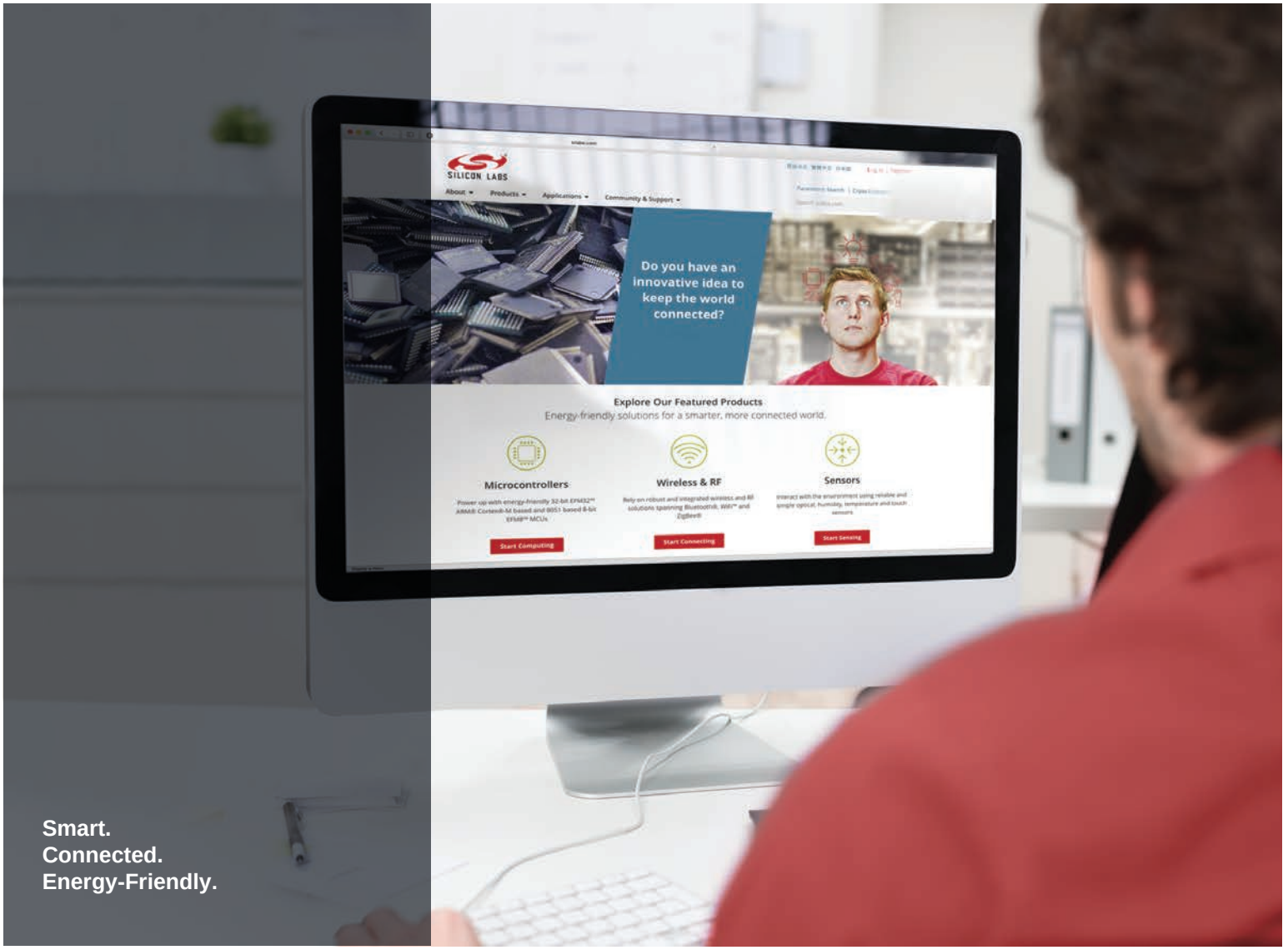
Revision No.	Version No.	Date	Changes
			5.3.1.4, 5.6.1, 5.7.1.3, 5.7.1.4 Added condition in Set certificate command in sections 5.2.20, 5.6.19 Added condition in Roam parameters command in sections 5.6.13 ,5.2.44 Added Links for BLE,BT and ZigBee documents
12	1.0.10l	Dec 2014	Changed the hyper link for ZigBee ,BT,BLE documents
13	1.0.10n	Dec 2014	Added Asynchronous messages when a station is connected or disconnected in AP mode Changed the hyper link for ZigBee ,BT,BLE documents
14	1.0.10o	Jan 2015	Added Supported features in Co-Ex mode in opermode command. Join command example changes, Added transparent mode related changes in user store configuration. Added support for continuous wave mode in PER mode command.
15	1.1.0	March 2015	1.Changed the Booltloader section 2.Corrected the Error list for Each command 3.Added new command for Uart Hardware Flow control 4.Added More error codes 5.Added command for trigger Auto configuration 6. Added command HTTP Abort and transparent mode command 7.Modified HT capabilities command parameters 8. Merged AT and Binary command sections
16	1.1.2	May 2015	1.Added AP keep alive feature information in AP config command parameters 2.Added FTP client feature 3.Added HTTP server credentials command 4.Added HTTP credentials parameters in store configuration and user configuration structure 5.Added Description about listen interval and join feature bit map in join command 6.Added new Tcp/Ip and Custom feature bits in Opermode command 7.Updated bootloader section as per bootloader Version 1.6

Revision No.	Version No.	Date	Changes
			8.Updated certificate loading command 9.Updated maximum supported host webpages 10. Added DTIM or Beacon aligned listen interval support in power mode command 11.updated error codes 12. updated user configuration structure
17	1.3.0	June 2015	1.Added quick scan and quick join feature description in scan and join sections 2.Changed the waiting time after wireless firmware upgradation from 4 minutes to 1.5 minutes.
18	1.3.2	Sep 2015	1.Updated SNMP trap feature 2.Added tcp retry count and socket bit map to socket create command
19	1.4.0	Aug 2015	1.Updated opermode command and added note for concurrent mode. 2.Updated set mac command functionality in concurrent mode. 3. Updated IP configuration command for concurrent mode. 4. Updated join command for concurrent mode. 5. Updated error codes. 6. Updating power mode command with new parameters explanation. 7. Added SMTP client 8. Added Synchronous read command 9. Added MFI commands 10. Added a note in UART Hardware flow control command 11.Corrected Power mode 3 and power mode 8 flow charts 12.Added Example to give PMK from host 13.Added WiSeConnect_TCPIP_FeatureSelection_v.1.4.0.xls
20	1.4.1	Dec 2015	1. Added POP3 client feature

Revision No.	Version No.	Date	Changes
			2. Updated READ data command 3. Added support to load SSL certificates on RAM
21	1.5.0	April 2016	1. Added HTTP PUT client feature 2. Added API/command description for setting real time clock from host. 3. Added custom feature bitmap BIT[28] for enabling real clock time feature from host.
22	1.6.0	August 2016	1. Added join feature bit BIT[3] to enable/disable CCKM feature. 2. Added extended custom feature bit BIT[2] and set region command explanation. 3. Corrected set region in AP mode command 4. Added missing response Ids
23	1.6.1	Sep 2016	1. Added explanation for private key password given through eap configuration command. 2. Added Changes in HTTP POST command. 3. Added HTTP POST DATA command. 4. Added private key password explanation from host in set EAP command/API 5. Added PSK type 4 and 5 (only for UART mode) explanation under NOTE box.
24	1.6.2	Oct 2016	1. Added explanation for MAC based join. 2. Added extended feature bit BIT[3] and BIT[4] explanation. 3. Added OTAF feature 4. Added DHCP option 81 and DNS update changes
25	1.6.3	Dec 2016	1. Added FTP client passive mode support
26	1.6.4	Jan2017	1. Updated socket create command 2. Updated SNMP get next command
27	1.6.5	Apr2017	1. Updated SNTP command 2. Updated IP config command 3. Updated Extended Custom feature bitmap

Revision No.	Version No.	Date	Changes
28	1.6.6	May2017	1. Updated SMTP client with SSL bitmap changes
29	1.6.7	June2017	1. Added objid parameter in snmp get response command to take SNMP object OID from host 2. Added SNMP Get Stats command 3. Added support for default wireless configuration page bypass and override option
30	1.6.8		None.
31	1.6.9	Sept2017	1. Updated the Wireless firmware upgradation failure message to host 2. Corrected the SNMP Get Stats command response structure 3. Corrected the channel numbers
32	1.7.0	Dec2017	None.
33	1.7.1	Jan2018	1. Added HTTP PUT header information 2. Added HTTP GET max size supported.
34	1.7.1a	Mar2018	1. Alignment Changes 2. Updated Concurrent mode in Opermode section 3. Updated rx_window_size information. 4. Added concurrent mode command flow in the Appendix A.
35	1.7.2	May 2018	1. Removed timeout parameter in Ping command 2. Added unit for keep_alive_initial_timeout 3. Updated information for Quick Scan 4. Added example for at+rsi_http_post_data 5. Updated section **8.22Set Certificate** 6. Added WLAN config command for configuring RTS Threshold 7. Updated description of the error code 0x002F in chapter 9**Error Codes** 8. Added the error code named 0x002F in apconf response parameter 9. Added Bit [4] and Bit [5] details in the chapter **8.1Set Operating Mode** 10. Added description of the error code 0xBBEE in chapter 9**Error Codes**
36	1.7.3	Aug 2018	1. Added HTTP PUT response body handling coming from the server. 2. Added TCP retransmission timeout paramaeter. 3. Corrected RTC command parameter.

Revision No.	Version No.	Date	Changes
37	1.7.5	Mar 2019	1. Added timeout bit for unicast_probe_timeout. 2. Added description of the error code 0x00B0 in chapter 9 **Error Codes** 3. Added possible error code(0x00B0) for timeout command
38	1.7.6	May 2019	1. Added flow chart for Powersave Mode9 2. Added note of Maximum number supported BG channel.
39	1.7.8	Oct 2019	1. Updated EAP command parameters
40	1.7.9	May 2020	1. Added Note in section 5:command mode selection 2. Added new command for getting Flashtype (Micron/Macronix). 3. Added a note, when to use set rtc time feature.



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